LFS303

Linux for Cloud Technicians

Version 1.2

Version 1.2

Published by:

the **Linux Foundation**
https://www.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.

Contents

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.

List of Figures

List of Tables

Chapter 1

Introduction

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.

Linux Foundation Events

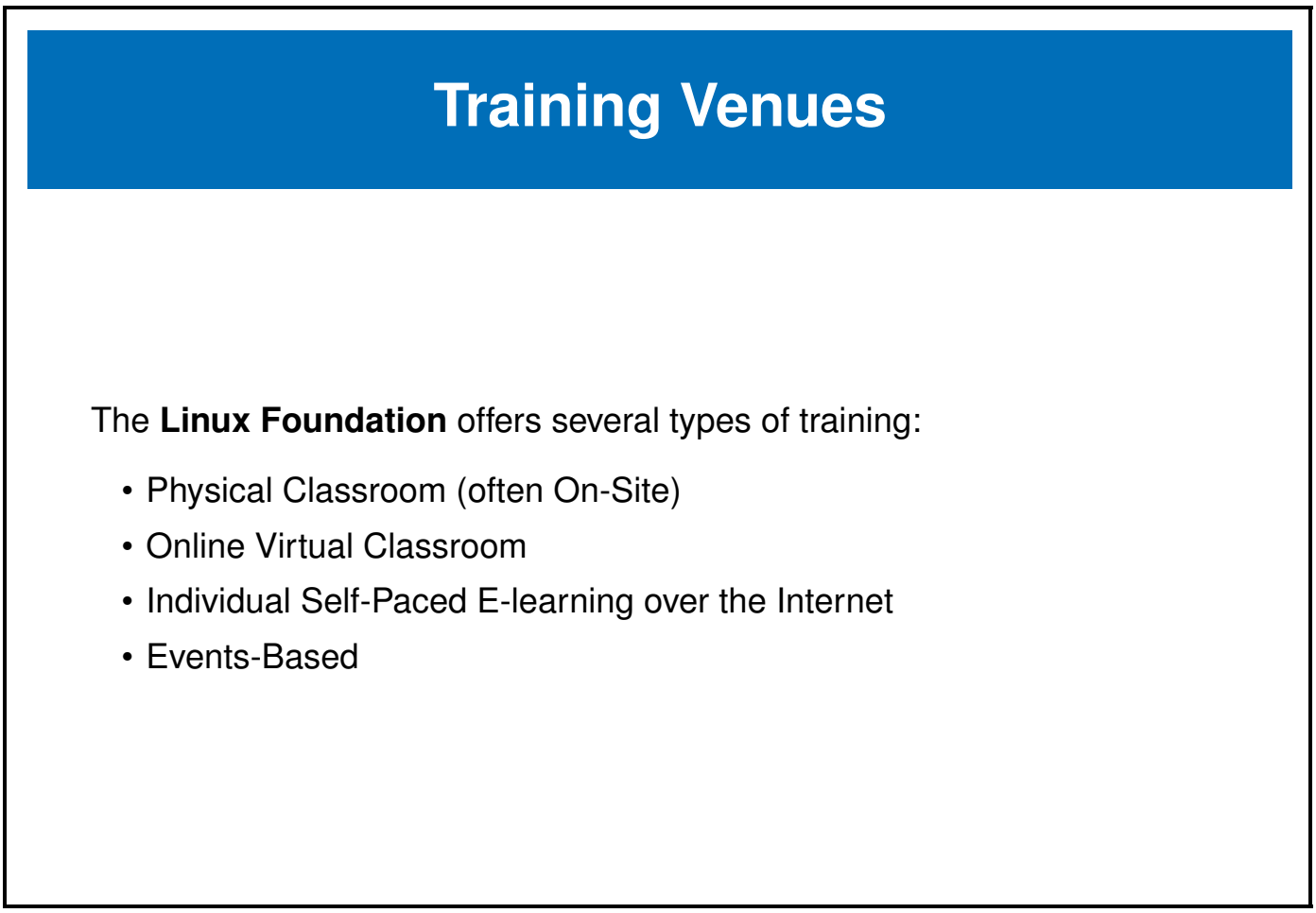
This is only a very partial list of **Linux Foundation** events. Some are held in multiple locations yearly, such as North America, Europe and Asia.

- 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.

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.

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 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.

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.

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.

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.

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

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.

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.

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/LFS303
- If you do not have a browser available you can obtain with:

```
$ wget --user=LFtraining --password=<ask instructor> \
  https://training.linuxfoundation.org/cm/LFS303/LFS303_V1.2_SOLUTIONS.tar.xz
```

  and similarly for the RESOURCES file.

```
$ wget --user=LFtraining --password=<ask instructor> \
  https://training.linuxfoundation.org/cm/LFS303/LFS303_V1.2_RESOURCES.tar
```

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

```
$ tar xvf LFS303_V1.2_SOLUTIONS.tar.xz
$ tar xvf LFS303_V1.2_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.

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.

1.7 E-Learning Course: LFS203

Companion E-Learning Course: LFS203

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

  LFS203: Linux for Cloud Technicians 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 **LFS203** as part of your enrollment in this course. Otherwise you can contact separately through https://training.linuxfoundation.org/.

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.

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.

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.

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.

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.

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

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.

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.

Chapter 2

System Configuration from the Graphical Interface

Learning Objectives

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

- Explain the advantages and disadvantages of using a graphical user interface (**GUI**) as opposed to a command line interface (**CLI**).
- Apply system, display, and date and time settings using the System Settings panel.
- Track the network settings and manage connections using Network Manager in Linux.
- Synchronize the time with servers on the Internet.
- Install and update software in Linux from a graphical interface.
- Work from terminal windows and consoles on the desktop.
- Understand the basic architecture of the Windowing System and Desktop Manager.

2.1 GUI or Command Line?

GUI or Command Line?

- Users are familiar with **GUI** interfaces from other Operating Systems.
- But **GUI**s more variable and less powerful than command line
- We will concentrate on the command line after this section

This section explains how to perform System Administration tasks from within the graphical user interface. However, the rest of the course will concentrate on working from the command line.

Users coming to **Linux** from other operating systems may be accustomed to accomplishing many (*if not all!*) tasks using a **G**raphical **U**ser **I**nterface (**GUI**).

GUIs are generally easier to figure out when you have to do something only once or rarely. However, while GUIs require less knowledge of details of how the system internals work and are configured, they:

- Vary more than command line methods from one **Linux** distribution to another, and from one version to another (e.g., **Debian** vs **CentOS-Stream**, or **Ubuntu 20.04** vs **Ubuntu 22.04**.)
- Have fewer capabilities; configuration choices are usually limited to what is present on canned menus.
- Are inefficient when the same operation has to be performed multiple times, or would be amenable to scripting and grouping and batching.
- Are often not available in cloud-based environments or containerized methods; the only access may be through a remote command line shell using the **SSH** facility. (Although it might be possible to run graphical utilities over the remote connection and display locally.)

For these reasons the command line methods must be mastered to accomplish tasks successfully, even if one does not have to do serious **Linux** system administration.

In this section we will show how to do some of the main system configuration tasks using the available GUIs, but we will not really use them the rest of the course.

2.2 System Settings

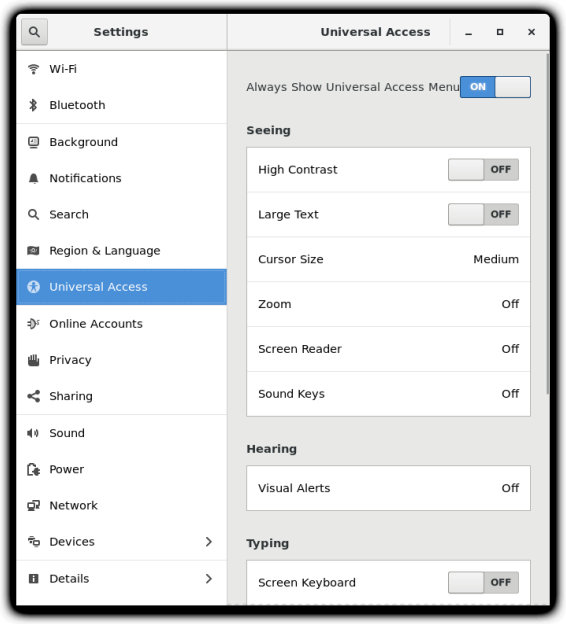
System Settings

- On most **Linux** distributions an icon on the taskbar or desktop, either visible all the time or when clicking on the top panel, or right-clicking anywhere on the desktop:
  - Gear
  - Wrench
- Also found in menus few different places, such as
  - Top left menu: `Applications -> System Tools -> System Settings`
  - Bottom left menu: `Favorites -> Configure Desktop`
  - Top right menu: `System Settings`
- Most system and per-user settings are here
- Most settings apply immediately, but some ask for approval before being made permanent.

Most basic configuration options and desktop settings can be controlled via the System Settings panel, accessible from various places depending on the distribution.

In general, settings are applied immediately when selected; you only need to press Apply or Save when the button is present.

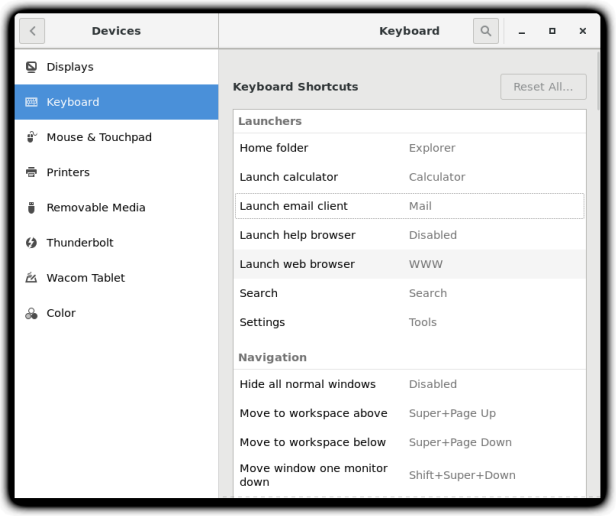
System Settings: GNOME



- The settings menu is brought up by clicking on the right upper panel and then on the settings icon, which can be a gear, a wrench, etc.
- Many settings are buried in the `Devices` and `Details` submenus
- Many settings appear by running **gnome-tweaks**
- Appearances differ in details among distributions

Figure 2.1: **System Settings: GNOME**

The settings interface has important submenus, including `Devices` and `Details` as shown below, Distributions will differ in how things are laid out in details.



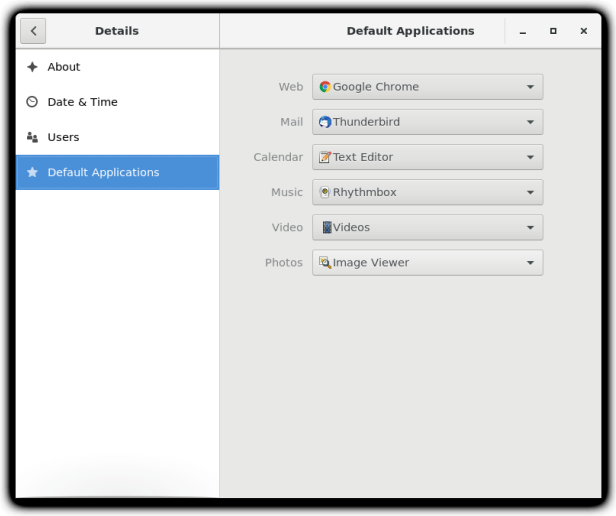


Figure 2.2: **System Settings: Devices and Details**

Many more settings are available in the `gnome-tweaks` utility.

System Settings: KDE

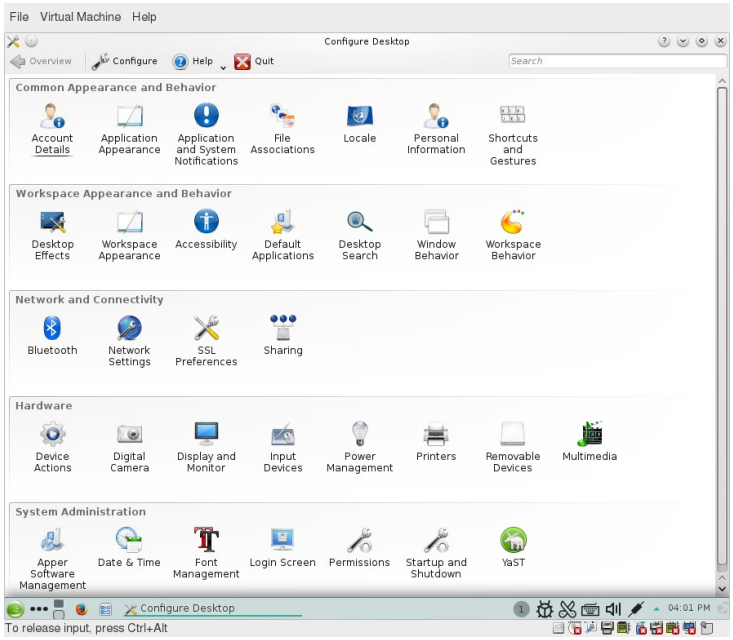


Figure 2.3: **System Settings: KDE**

Systems based on the **KDE** have a rather different looking but equivalent settings screen.

2.3 Display Settings

Changing Display Settings

- Both graphics card driver and monitor must be configured
- **GNOME**: `System Settings, Displays` option
- **KDE**: `Configure Desktop, Display and Monitor` option
- Proprietary driver configuration settings (e.g., **Nvidia**)
  - Not in `System Settings/Configure Desktop`
  - Generally more options, some specific to the driver
  - Use only if the standard Displays configuration panel does not do what is needed
- `/etc/X11/xorg.conf`
  - Standard **X** server configuration file
  - Not present on most modern distributions
  - May be necessary in some situations

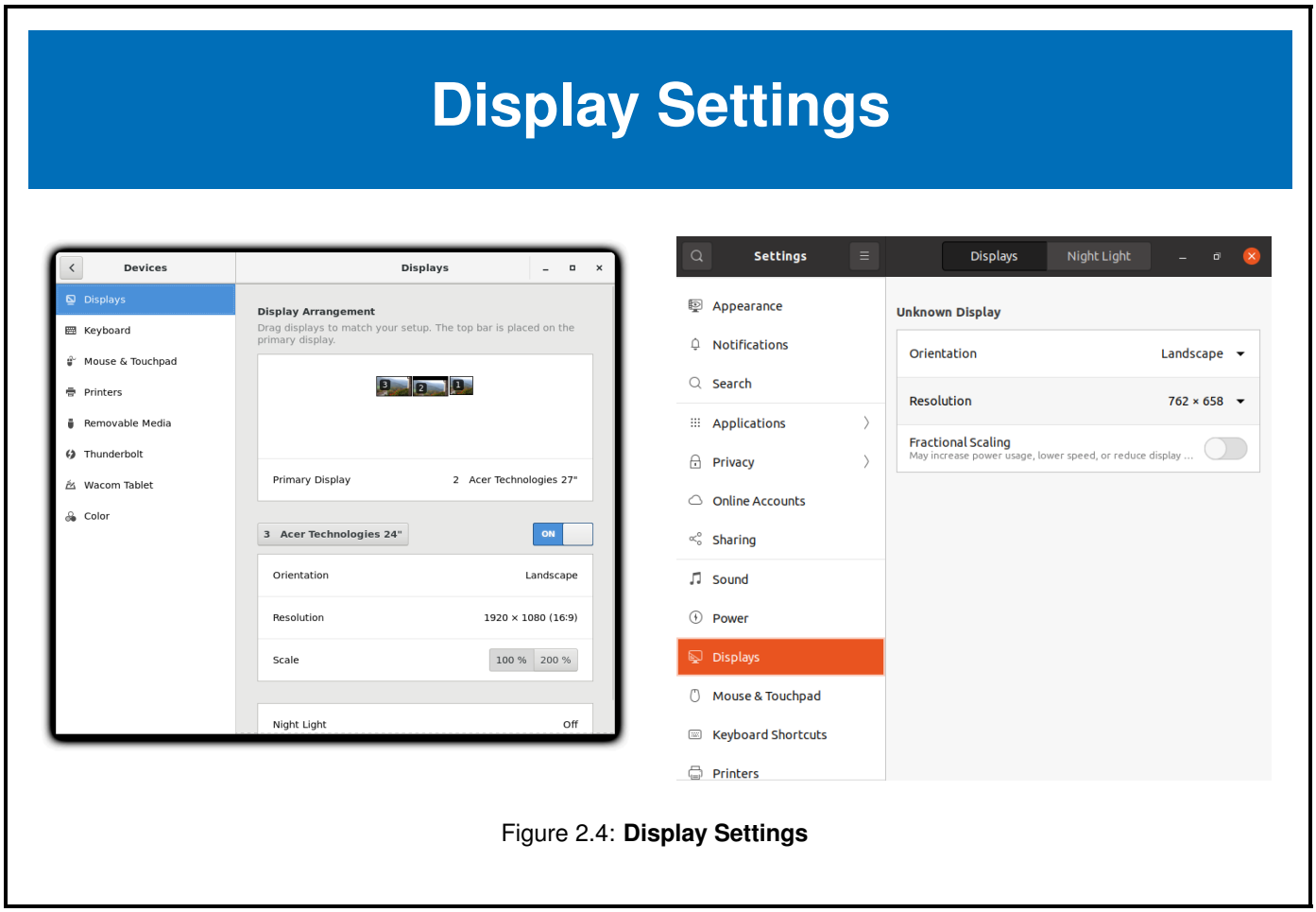


Figure 2.4: **Display Settings**

Above are **GNOME** graphical display settings from **RHEL 8** and **Ubuntu 20.10**. The equivalent **KDE** program is just as easy to deal with.

The `Displays` panel under `System Settings` (or `Display and Monitor` panel under `Configure Desktop`) contains the most common settings that you will need to change.

If your system uses a proprietary driver such as one from **Nvidia** or **ATI**, you should also have an auxiliary configuration program for that video card driver. This program may give many more configuration options, but will also be more complicated, and may require root access. In general, if you can do what you want from the normal `Displays` panel, that is preferable to doing it in the proprietary configuration program.

Finally, the **X** server uses `/etc/X11/xorg.conf` as its configuration file if it exists. The file is usually only there in unusual circumstances, such as when certain graphics drivers are in use, and changing this file directly is usually for more advanced users.

Multiple Monitors

- **Linux** systems do a good job of auto-detection multiple monitors, but it is possible to tweak:
  - Can set resolution of each screen independently
  - Can either mirror displays or extend the desktop
  - Placement of extended displays can be controlled via drag and drop

Users often have multiple monitors, (also known as **Displays** and **Screens**). One common configuration is a large monitor connected to a laptop. The large monitor can be configured to be easier to read than the laptop through the display settings.

Multiple monitors can be configured in several modes like, single, joined (extended) or mirror.

In single mode, one monitor is on and the rest are blank or display a logo.

In joined or extended mode, all the monitors are on and the effective desktop encompasses all monitors. The mouse can move to any monitor. When in joined mode the placement of the monitor in the display settings can be moved around the primary monitor with one edge touching the primary monitor. This edge is where the mouse can travel from one monitor to another.

In mirror mode both displays show the same information.

Multiple Monitors Example

A system with 3 display monitors:

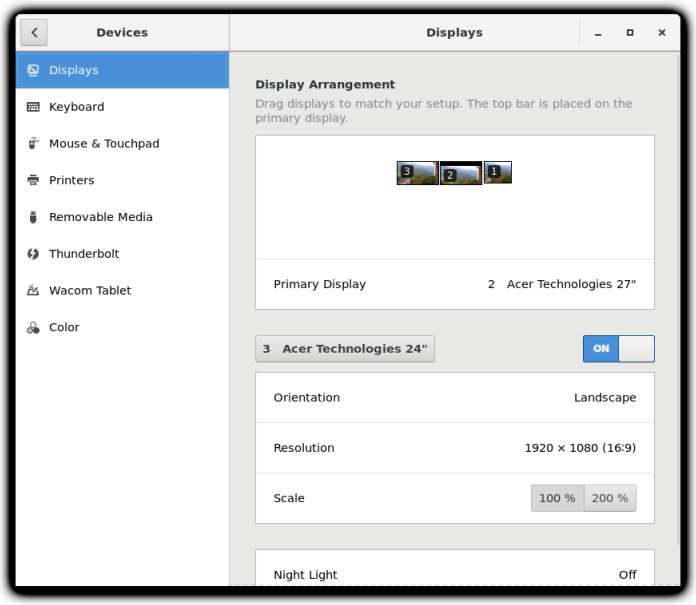


Figure 2.5: **Configuring Multiple Monitors**

If you have dual monitors, the `Displays` panel shows the current layout of the screens.

You can click and drag the displays around to rearrange their relative layouts as needed.

One monitor will be considered **primary monitor** on which the taskbars etc. will sit. You can drag and drop it to the other screen if desired.

The system will remember the chosen screen layout and return to it the next time it detects the same displays.

Changing Screen Resolution

- Click on resolution

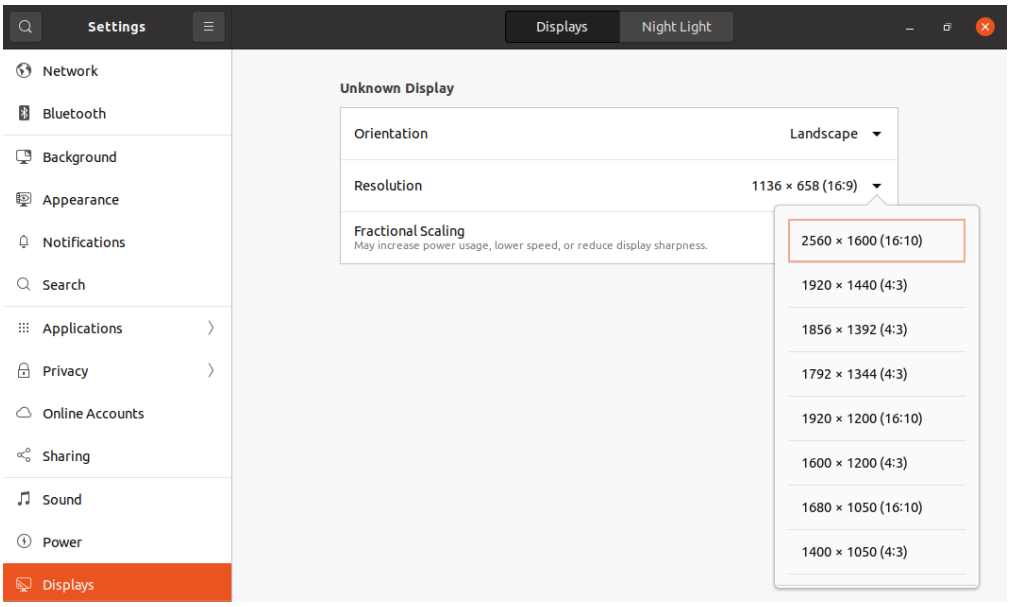


Figure 2.6: **Ubuntu Changing Screen Resolution**

Most of the time, the system will figure out the best resolution for your screen automatically. In some cases, it may get this wrong, or you may want to change it for some special circumstances.

The `Displays` panel will only allow you to change to resolutions supported by your display. It will also switch to the new resolution when you click `Apply`, and confirm that the resolution is working. In case the resolution fails, the system will switch back to the original resolution after a short timeout.

2.4 Network Manager

Graphical Network Configuration

- **Network Manager** handles all network connections:
  - Wired
  - Wireless
  - VPN
  - Mobile Broadband
- Usually configured from a GUI
- Static configuration files
  - Differ significantly among **Linux** distributions
  - Require more knowledge to control
  - Sometimes must be directly edited in non-trivial situations
- Launch GUI by clicking on network icon on toolbar or desktop

All **Linux** distributions have network configuration files. However, the exact name, location and format of each distribution's files tend to be different. This historical approach can handle some complicated setups, but is not very dynamic or easy to use.

Network Manager was developed to address dynamic situations and ease of use. It can present wired and available wireless networks, allow the choice of a wireless or mobile broadband network, handle passwords, and set up **VPN**s. Except for unusual situations, it's generally best to let **Network Manager** keep track of your networking settings.

Configuring Network Manager

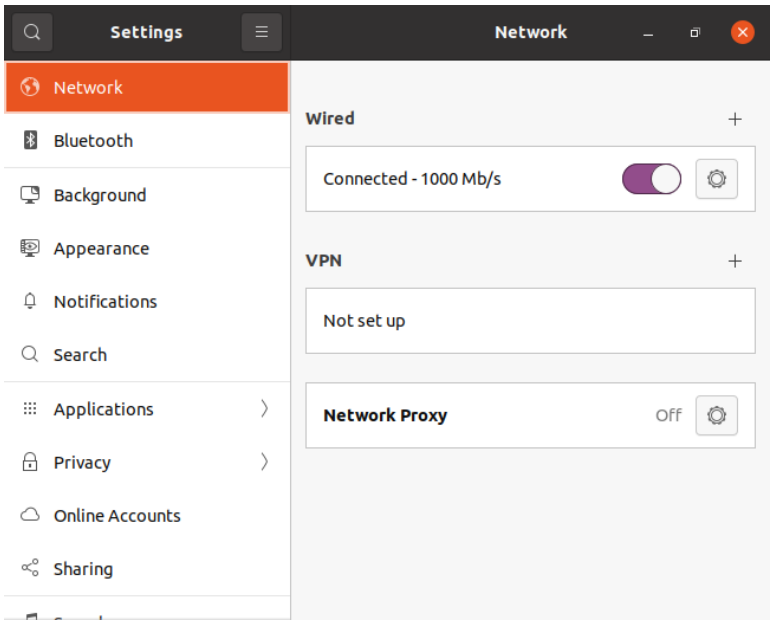


Figure 2.7: **Configuring Network Manager (Ubuntu 20.04): 1**

The above screenshot is gotten by clicking on the `Settings` icon on the top right and then clicking on `Network`.

Network Manger Distribution Differences are Small

These screenshots are for **Ubuntu 20.04**. On other distributions, such as **RHEL/CentOS 8** there are only very minor visually cosmetic differences.

It is not important to chase these down, as the graphical interfaces are easy to figure out, by definition.

Configuring Network Manager 2

After clicking on the `wired` icon:

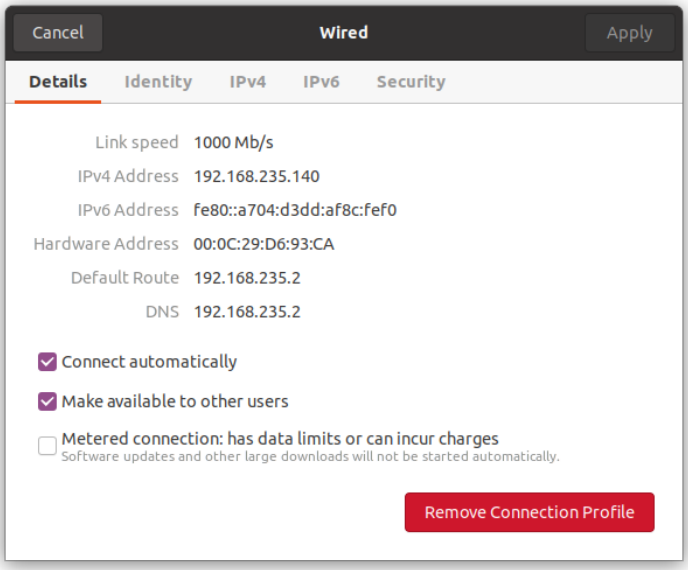


Figure 2.8: **Configuring Network Manager: 2**

The next slide shows the result of clicking on **IPv4**.

Configuring Network Manager: 3

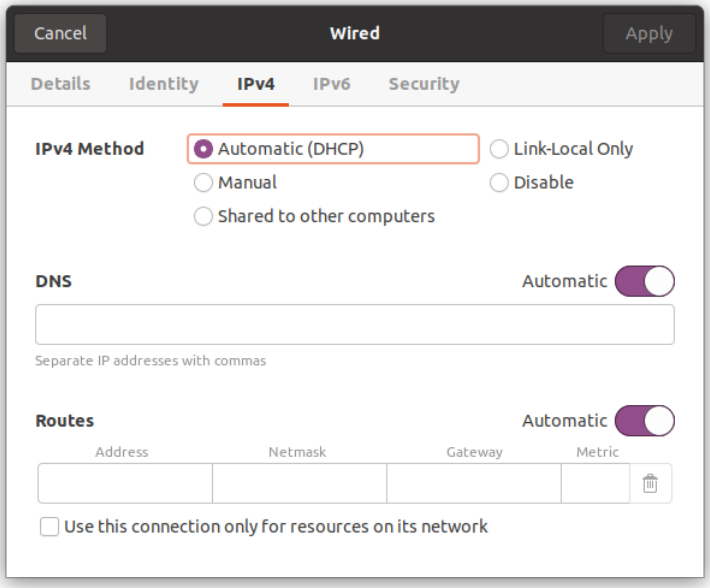


Figure 2.9: **Configuring Network Manager: 3**

Manual settings can be inserted here for routing, gateways, etc, the kind of information that can also be directly entered into text configuration files under `/etc`.

Wired

- Direct Ethernet connections
- Generally automatically configured
- Can set up static connection settings if needed

Direct wired connections usually need no configuration; the hardware interface and signal presence are detected automatically, and the actual network settings are then set up via **DHCP**. Static configurations need manual setup, however, and this tab is the easiest place to set that up. It's also possible to change the reported Ethernet MAC address from here if your hardware supports that.

Wireless

- None connected by default
- Can choose a network to connect to via the menu
- Wireless tab on `Edit Connections` dialog
  - Can add, edit, or remove known wireless networks

The **Network Manager** drop-down menu includes a list of detected wireless networks, as well as the network you're connected to (if any).

Mobile Broadband

- Connects via a cellular data plan
- Wizard interface for setting up each available network

If you have a mobile broadband connection, you can use it with **Network Manager**.

The interface for setting this up uses a wizard-like interface to set up the specific connection details for each connection.

Once configured, the networks are configured automatically when the adapter for the broadband network is attached.

VPN

- Encrypted secure tunnels
- Many VPN technologies supported, including:
  - **OpenVPN**
  - **StrongSWAN**: native **Linux IPSec** client
  - **Cisco**'s **IPSec**-based **VPN**
  - **PPTP**: **Microsoft**'s **VPN** system

Network Manager can also manage your virtual private network (**VPN**) connections.

The most popular services are supported, including native **IPSec**, **Cisco OpenConnect** (via either the **Cisco** client or a native open-source client), **Microsoft PPTP**, and **OpenVPN**.

Note that many distributions distribute support for each **VPN** as a separate package; if your **VPN** does not appear to be supported when you add a new connection, you may need to install the support package for that **VPN** type.

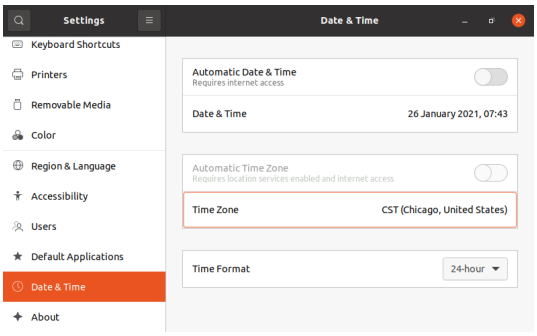
2.5 NTP (Network Time Protocol)

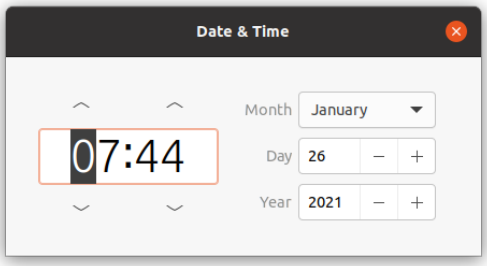
Basic Date and Time Settings

- Accessed either by:
  - Clicking or right clicking on time in taskbar
  - Clicking on settings in upper right corner, and then finding the sub-menu, `Details->Time & Date`
- Set location
  - Used for time zone setting
  - Improper setting can cause displayed time to be wrong
- Set clock type: 24-hour or AM/PM time
- **Linux** uses **UTC** (**C**oordinated **U**niversal **Time** in French)

Linux always uses Universal Time (**UTC**) for its own internal time-keeping; displayed or stored time values rely on the system time zone setting to get the proper time.

Setting Date and Time





- The date and time settings can be found under `Details` in the settings window.
- The windows for `Date & Time` and `Time Zone` will open only if the Automatic features are turned off.
- On some distributions you will have to unlock by supplying a password before you can change values.

Figure 2.10: **Setting Date and Time**

Note the **Automatic Time** button; it will be used to turn **NTP** on or off.

The above screenshot is from **Ubuntu**, but all recent **Linux** distributions with **GNOME**-based desktops look quite similar.

2.6 Graphical Software Package Management

Software Packaging Concepts

- **Linux** distribution software is divided into **packages**
  - Each package is a piece of complete **Linux** system or application
  - Packages can be organized into groups
  - Packages depend on each other
  - Order of installation and removal important
- Two general package management systems:
  - **RPM**
  - **Debian** (**APT**)
- Low-level part: unpacks archive and installs (**rpm**, **dpkg**)
- Higher-level part: handles online repositories, groups, dependencies (**dnf**, **zypper**, **apt**, **apt-get**, **apt-cache**)
- Will discuss details later in the **command line** interface

Each **package** in a **Linux** distribution provides one piece of the system, such as the **Linux** kernel, the **C** compiler, the shared software for interacting with **USB** devices, or the **Firefox** web browser.

Packages often depend on each other; for example, since a browser such as **Firefox** or **Chrome** can communicate using **SSL**/**TLS**, it will depend on a package which provides the ability to properly encrypt and decrypt **SSL** and **TLS** communication, and will not install unless that package is also installed at the same time (unless the facility is built into the application itself, an increasingly common tendency.)

The low-level details of unpacking a package and putting the pieces in the right places is handled by one utility; most of the time, you will be working with a higher-level utility which knows how to download packages from the Internet and can sort out dependencies and groups properly for you.

In this section we will deal only with the graphical interface to the package management systems; in a latter section we will discuss the command line interface which is probably more commonly used by system administrators.

RPM

- **Red Hat Package Manager**
- Used by all **Red Hat**-based and **SUSE**-based distributions
- **RPM** is the low-level part; the high-level part differs
  - **RHEL/CentOS**, **Fedora**: **dnf**
  - **openSUSE**, **SUSE**: **zypper**, **YaST**
- **PackageKit**: unified interface to different package managers, includes a **GUI**

RPM is a package management system popular on **Linux** distributions. It was developed by **Red Hat**, and adopted by a number of other distributions, including the **SUSE** family, **Mandriva**, **CentOS**, **Oracle Linux**, and others.

The high-level package manager differs between distributions; most use the basic repository format used in **dnf** (the package manager used by **Fedora** and **Red Hat Enterprise Linux** and **CentOS**), but with enhancements and changes to fit the features they support.

dnf vs yum

RHEL/CentOS 7 used the older **yum** utility rather than **dnf**. Note that **dnf** retains a backwards compatibility wrapper so that most **yum** commands still work.

RPM Package Management GUIs

gnome-software or `Applications->System Tools->Software`

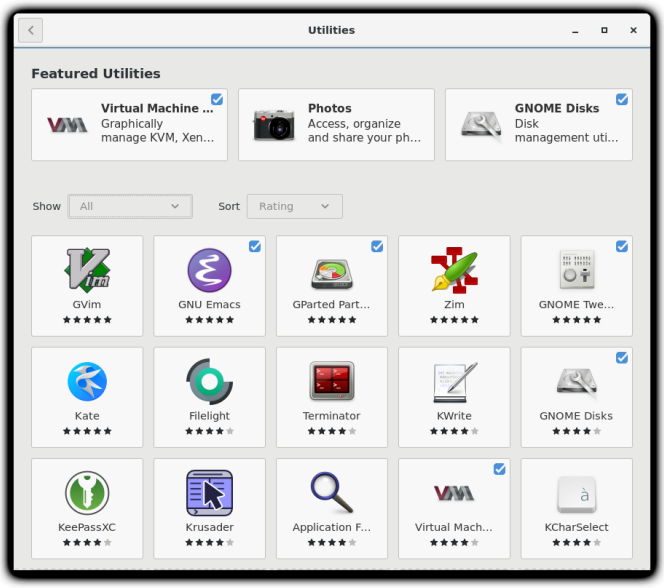


Figure 2.11: **gnome-software**

gnome-software is a graphical interface to package management which is available on most **Linux** distributions. Exactly what it looks like is somewhat dependent on the specific distribution and version it is hosted on.

The image shown here is taken from **RPM** GUI provided by **PackageKit**'s add and remove software utility, shown on **RHEL**.

This GUI is rather limited, and is really more of an update configuration facility. For example, it does not have a good searching capability.

Older versions of **RHEL**, **CentOS** and **Fedora** had a true **PackageKit** GUI.

openSUSE Package Management with Yast

- **Yast** has very detailed capabilities for installing, updating, finding and configuring software on **SUSE**-based systems.

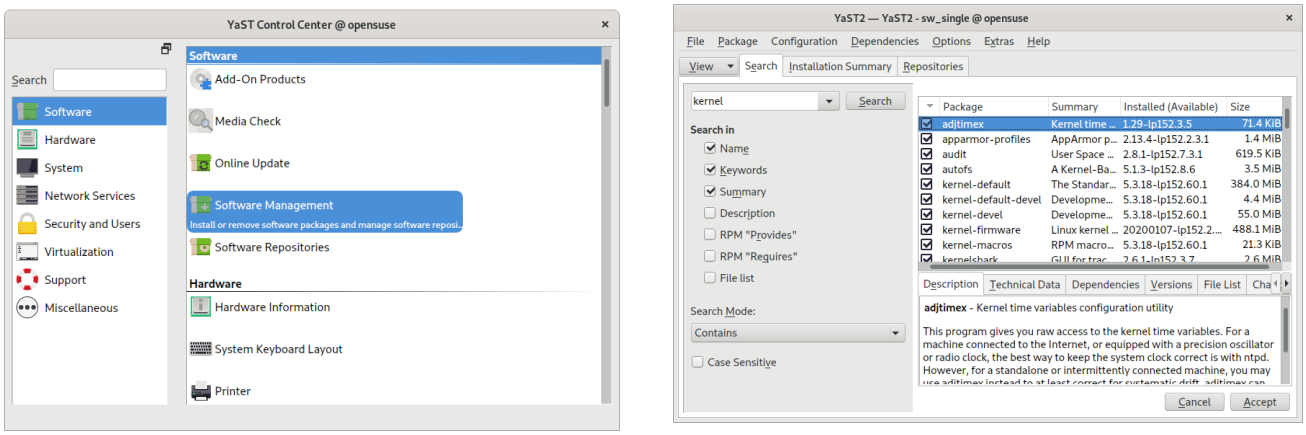


Figure 2.12: **Yast Package Management on openSUSE**

Debian Family System

- Low-level utility: **dpkg**
- High-level system: **APT** (**A**dvanced **P**ackaging **T**ool)
- Graphical interfaces built on top of **APT**
  - **Debian**: **synaptic**
  - **Ubuntu**: **Ubuntu Software Center**, **Update Manager**, **synaptic**

One family of distributions base themselves on **Debian**. These use **dpkg** and **APT** for package management. The most high-profile **Debian**-based distribution is **Ubuntu**; another is **Linux Mint**.

Generally, each system uses **APT**, but creates their own graphical user interface to it. Although **APT** repositories are generally compatible with each other, the software they contain generally is not; most repositories therefore target a particular distribution, and often software distributors will ship multiple repositories to support multiple distributions.

Synaptic

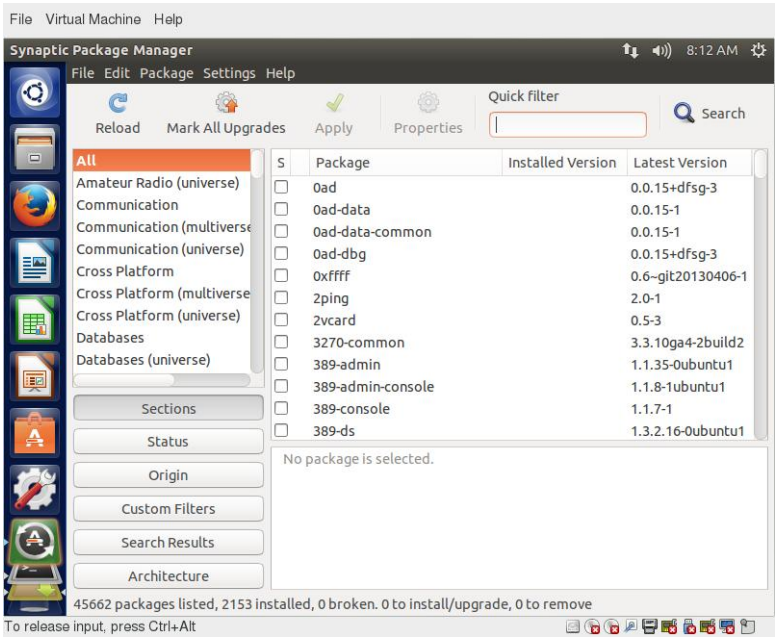


Figure 2.13: **Synaptic**

This is how the **synaptic** looks upon launching. Software categories are listed in the left panel.

It is very easy to see what is installed and what is available for installation.

It is also easy to search for packages based on name or key words.

After any choices are made for installation or removal, one can click on `Apply` to accomplish the task. Any additional packages needed to satisfy dependencies will also be installed at the same time, depending on your approval.

Ubuntu no longer installs **synaptic** by default, as it prefers use of the **Ubuntu Software Center** for marketing reasons. Thus, the first thing you should do is install **synaptic** on your system.

2.7 Console

Console, or Text-Mode Login

- Can boot to a console instead of a **GUI**
  - Can configure the system to do it only once using the boot loader (using interactive **GRUB**)
  - Can do it permanently by configuring files in `/etc`
- Can jump to a console (Virtual Terminal) while running a **GUI**
  - Hit `CTRL-ALT-F[1-7]`
  - GUI usually on **VT 1** or **VT 7**
- Will discuss in detail later

When logging into the text console or a virtual terminal, the user gives their name and password.

They will then have a command line interface, running the user's default **shell** program.

The default command shell is **bash** (the **GNU** Bourne Again Shell), but there are a number of other advanced command shells available.

The shell prints a text prompt, indicating it is ready to accept commands; after the user types the command and presses `Enter`, the command is executed, and another prompt is displayed after the command is done.

Consoles can be accessed using `ALT` plus a function key. Most distributions start six text consoles, and one graphical console (usually VT 1 or VT 7). If the user is already within a graphical environment, started, switching to a VT requires pressing `CTRL-ALT-F[1-7]`. However, on some **Linux** systems this may not work.

2.8 X Window System and Desktop Manager

X Window System and Desktop Manager

- Reaching a graphical interface involves three levels:
  - **X Window System** (**X Server**)
    The component that handles display, keyboard, mouse, etc. Traditionally **Xorg**. On some recent systems **Wayland**
  - Window Manager
    The component that handles appearance and placement of windows, visual effects, etc. (**metacity**, **kwin**, **enlightenment**, etc.)
  - Desktop Manager
    The component that handles appearance of windows, placement, drag and drop operations, taskbars etc. (**GNOME**, **KDE**, **xfce**, etc)
- The **greeter**, or **display manager** (usually **gdm**, **lightdm**, or **kdm**) handles graphical login and choosing the user, and is usually packaged with the desktop manager
- Can also start the GUI from a console with **startx**

Generally the **X Window System** is loaded as the final step in the boot process. A service called the **display manager** keeps track of the displays being provided, and loads the **X server** (so-called because it provides graphical services to desktop apps, sometimes called **X clients**). The display manager also handles graphical logins, and starts the appropriate desktop environment after a user logs in.

A desktop environment consists of a session manager, which starts and maintains the components of the graphical session, and the window manager, which controls the placement and movement of windows, window title-bars, and controls. Although these can be mixed, generally a set of utilities, session manager, and window manager are used together as a unit, and together provide a seamless desktop environment.

If the display manager is not started by default in the default runlevel, you can start **X** a different way, after logging on to a text-mode console, by running **startx** from the command line.

Please Note

Many servers either do not have a GUI installed, or do no routinely enable it, in order to keep the system more secure and leaner.

2.9 Labs

Video Demonstration Resources

using_package_management_demo.mp4
using_package_gui_ubuntu_demo.mp4
opensusesoftwaremanage.mp4

Exercise 2.1: Screen Resolution

Find out the current screen resolution for your desktop.

Change it to something else, and change it back to its original value.

Note: you can also ascertain your current resolution by typing at a command line:

```
$ xdpyinfo | grep dim
```

```
  dimensions:    3200x1080 pixels (847x286 millimeters
```

Exercise 2.2: Networking Connections

Get the current networking configuration for your desktop.

Are you on a wired or a wireless connection? Or both?

If you have wireless hardware, see what wireless networks are available, if any?

Solution 2.2

Click the Network Manager icon. This should bring up the wired and wireless connection state, and which wireless networks are available.

Exercise 2.3: Time Settings

Change the time zone of your system to London time (or New York Time if you are currently in London time). How does the displayed time change?

After noting the time change, change the time zone back to your local time zone.

Solution 2.3

On a **GNOME** desktop, click on `System Settings->Date and Time`, and note the current location settings.

You will either have to click on `Unlock` to give the root password, or will be asked for it when you try to change the time zone, depending on your **GNOME** version.

Use the `Region` and `City` drop-downs to switch the region to `Europe` and the city to `London`, or type `London` into the Location widget and pick the proper location from the list that appears. (You may also just be able to click at the right place on a map of the world.)

Look at the clock and note that the displayed time has changed. Now, set the time zone back in the same way.

On **KDE** desktops, the details are somewhat different but it is not rocket science to figure out how to do this, and it should take you less time to find the configuration methods than it does to describe them.

Exercise 2.4: Installing and Removing Software

Using graphical package management tools, find the **dump** package.

If it is already installed, remove it, and then re-install it.

If it is not installed, install it.

Solution 2.4

Please Note

We cannot really give precise instructions here, once again; you will just have to learn to navigate through the menus until you find the package management interface, but once you get there it is easy enough to play with.

Exercise 2.5: Switching to A Virtual Terminal (VT)

1. Switch from the graphical desktop to a virtual terminal.
2. Log in to your user account
3. Switch to another virtual terminal
4. Switch back to your original graphical desktop

Solution 2.5

1. Hit `CTRL-ALT-Fn` where `n` is between 2 and 6.

   Note 1 or 7 may be reserved for switching back to the desktop, and because you are running the desktop you must hit `CTRL-ALT`, not just `ALT`.
2. Just type in your user name and password when prompted.
3. Hit `ALT-Fn` where `n` is between 2 and 6. (Note it won't hurt if you still use the `CTRL-ALT` combination.)
4. Hit `ALT-F1` or `ALT-F7` depending on your system, to arrive back on your original graphical desktop

Chapter 3

Boot Process and System Initialization

Learning Objectives

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

- Understand the role of the bootloader.
- Explain how the **Linux** kernel starts up and how it uses an **initramfs** to accomplish this on many types of hardware.
- Understand how the **init** process works once the kernel has accomplished booting in full, and how subsequent system services are put in motion.
- Explain the role of the **systemd** facility in managing the running system
- Know where the main files are that control **systemd**
- Use **systemctl** to start or stop system services at run time or system boot or shutdown.

3.1 Bootloader

Bootloader

- Executed before **Linux** kernel starts
  - User can interactively select alternative operating systems, versions, options at boot.
- Closely coupled to:
  - **BIOS**
  - **EFI/UEFI**

The boot loader is usually stored on one of the hard disks in the system, either in the **boot sector** (for traditional **BIOS/MBR** systems) or the **EFI partition** (for **EFI/UEFI** systems).

A number of boot loaders exist for **Linux**; some common ones include **GRUB** (for **GR**and **U**nified **B**ootloader) (used on most **Linux** systems), **Das U-Boot** (used on many embedded platforms) and **ISOLINUX** (used mainly for booting from removable media).

Most **Linux** boot loaders can present a user interface for choosing alternative options for booting **Linux**, and even other operating systems that might be installed. When booting **Linux**, the boot loader is responsible for loading the kernel image and the initial **RAM Disk** image into memory.

3.2 Linux Kernel and initramfs

Linux Kernel and initramfs

- On startup, the **Linux** kernel detects hardware devices and loads needed device drivers from the kernel image itself
- It then loads (if needed) an **initramfs** (**init**ial **RAM f**ile**s**ystem):
  - Complete but minimal **Linux** filesystem used only for first stages of system initialization, and then later discarded
  - Contains essential device driver and filesystem **modules** and other information necessary for starting some hardware, including some **firmware**
  - Loaded by the boot loader along with the kernel and then stored in memory
  - Historically also called **initrd** (for **init**ial **RAM d**isk)
  - Must be rebuilt every time the kernel is updated or reconfigured

The boot loader loads the selected kernel image and passes control to it. On most architectures, the kernel is compressed for technical reasons, so its first job is to decompress itself.

Afterwards, it continues to check system hardware and load and initialize any additional operating system components, such as device drivers and network protocols needed to bring the system up to full operation.

In addition to the kernel, the boot loader almost always loads a special file called an **initial RAM disk**, usually with a name like **initramfs** or **initrd**. This contains a very minimal filesystem with some hardware drivers necessary for the system to boot, such as drivers for disk controllers, as well as filesystem drivers and scripts and other support files for loading these drivers. Once those drivers are loaded, the kernel mounts the real root file system and throws away the initial RAM disk.

The initramfs must be built individually for each **Linux** system, as it must contain the specific device drivers and protocols needed by both the hardware and the kernel configuration. **Linux** distributors can deploy the same kernel for every system they encounter, but the initramfs must be rebuilt every time a system is installed or has a kernel update or reconfiguration.

3.3 init and Services

/sbin/init

- First process, with process ID = 1
- Responsible for starting most other processes on the system
  - Starts some processes directly, which start other processes.
  - Kernel processes: started by kernel, to manage kernel internals
- Continues running after boot to manage the system
- On recent **systemd**-based systems (all major **Linux** distributions):

```
$ ls -l $(which init)
lrwxrwxrwx 1 root root 22 Feb 13 07:22 /sbin/init -> ../lib/systemd/systemd
```

Once the kernel has set up all its hardware and mounted the root file system, the kernel runs the program at `/sbin/init` on that file system. This becomes the initial process, which then starts other processes to get the system running. Most other processes on the system trace their origin ultimately to **init**; the exceptions are kernel processes, started by the kernel directly for managing internal kernel details.

Besides starting the system, **init** is responsible for keeping the system running and for shutting it down cleanly. It acts as the "manager of last resort" for all non-kernel processes, cleaning up after them when necessary, and restarts user login services as needed when users log in and out.

Exactly which program actually runs as **init** can vary according to distribution and version, with **systemd**-based systems now actually running **systemd** with appropriate arguments. It is also possible to run an alternative program through a kernel command line argument specified in the system configuration or interactively at boot, when using **grub** or another boot loader.

Services

- System **services** usually start at boot
- Usually run until shutdown
- Example:
  - Serve web pages with Apache (**httpd**)
  - Make files available via the FTP server (**vsftpd**)
  - Manage printers with CUPS (**cupsd**)
- Services can be made to start or not start at every boot
- Services can be turned on or off at runtime
- Historically were managed by **SysVinit**
  - All modern systems use **systemd**
  - Earlier replacement **Upstart**, has been phased out

The **init** process is responsible for determining what other processes need to be run. Traditionally, this was done using conventions that date back to **System V Unix**, with different **runlevels** containing different collections of scripts that start and stop services.

Modern systems use **systemd** to manage the boot and initialization process, as well for starting and stopping various system services after the system is running fully.

To enable or disable a service starting at boot on **systemd**-based systems (all modern systems):

```
$ sudo systemctl enable httpd
$ sudo systemctl disable httpd
```

To turn a service on or off after boot:

```
$ sudo systemctl start httpd
$ sudo systemctl stop httpd
```

While all Enterprise distributions have moved away from the older **SysVinit** procedure, they often support (through wrappers) the **System V** startup/stop commands for compatibility purposes. But one can expect this support to dwindle with time, so learn how to use **systemctl**.

3.4 systemd

systemd Features

- Boots faster than previous init systems
- Provides aggressive parallelization capabilities
- Uses **socket** and **D-Bus** activation for starting services
- Replaces shell scripts with programs
- Offers on-demand starting of daemons
- Keeps track of processes using **cgroups**
- Maintains mount and automount points
- Implements an elaborate transactional dependency-based service control logic
- Can work as a drop-in replacement for **SysVinit** and is compatible with **SysVinit** scripts.

The **systemd** system and session manager for **Linux** is now dominant in all major distributions.

Note that **systemd** is backward compatible with **SysVinit** and the concept of runlevels is supported via runlevel **targets**. The **telinit** program is emulated to work with runlevels.

Instead of **bash** scripts, **systemd** uses `.service` files. In addition, **systemd** sorts all daemons into their own **Linux cgroups** (control groups).

systemd Configuration Files

- `/etc/hostname` replaced:
  - `/etc/sysconfig/network` in **Red Hat**-based systems
  - `/etc/HOSTNAME` in **SUSE**-based systems
  - `/etc/hostname` (was already adopted as the standard) in **Debian**-based systems.
- Other files might include:
  - `/etc/vconsole.conf`: default keyboard mapping and console font
  - `/etc/sysctl.d/*.conf`: drop-in directory for kernel **sysctl** parameters
  - `/etc/os-release`: distribution ID file

Although **systemd** prefers to use a set of standardized configuration files, it can use distribution-dependent legacy configuration files as a fall-back.

Exactly which configuration files will depend on how each distribution sets things up; for example, `/etc/vconsole.conf`, which configures virtual terminal defaults, does not appear on **Ubuntu** systems.

Compatibility with SysVinit

systemd is configured to be mostly backward compatible with **SysVinit** so using old commands will generally work.

It supports the use of runlevels conceptually, through the mechanism of runlevel *targets*. In addition, **telinit** is emulated to work with runlevels.

3.5 systemctl

systemctl

```
$ systemctl [options] command [name]
```

- To show the status of everything **systemd** controls:

  ```
  $ systemctl
  ```

- Show all available services:

  ```
  $  systemctl list-units -t service --all
  ```

- Show only active services:

  ```
  $ systemctl list-units -t service
  ```

- To start (activate) one or more **units**:

  ```
  $ sudo systemctl start foo
  $ sudo systemctl start foo.service
  $ sudo systemctl start /path/to/foo.service
  ```

  where a unit can be a service or a socket.

systemctl is the main utility for managing services. Some **systemctl** commands can be run as non-root user, others require running as root or with **sudo**.

For most commands you can omit the `.service` attached to the service name.

For an excellent summary of how to go from **SysVinit** to **systemd**, see the **SysVinit to systemd Cheatsheet** at https://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet.

systemctl (cont'd)

- To stop (deactivate):

```
$ sudo systemctl stop foo.service
```

- Enable/disable a service:

```
$ sudo systemctl enable sshd.service
$ sudo systemctl disable sshd.service
```

  These commands do not actually start or stop a service; they control whether or not it is started up at system boot.

For most commands you can omit the `.service` attached to the service name.

3.6 Labs

Video Demonstration Resources

using_systemctl_demo.mp4

Exercise 3.1: Adding a New Startup Service with systemd

To add a new startup service we need to create (as root) a file directly under `/etc/systemd/system` or somewhere else in that directory tree; distributions have some varying tastes on this. For example a very minimal file named `/etc/systemd/system/fake.service` (which can be extracted from your downloaded `SOLUTIONS` file as `fake.service`) containing the following:

fake.service

```
[Unit]
Description=fake
After=network.target

[Service]
ExecStart=/bin/sh -c '/bin/echo I am starting the fake service ; /bin/sleep 30'
ExecStop=/bin/echo I am stopping the fake service

[Install]
WantedBy=multi-user.target
```

Now there are many things that can go in this **unit** file. The `After=network.target` means the service should start only after the network does, while the `WantedBy=multi-user.target` means it should start when we reach multiple-user mode. This is equivalent to runlevels 2 and 3 in **SysVinit**. Note `graphical.target` would correlate with runlevel 5.

Now all we have to do to start, stop and check the service status are to issue the commands:

```
$ sudo systemctl start fake.service
$ sudo systemctl status fake.service
$ sudo systemctl stop fake.service
```

If you are fiddling with the unit file while doing this you'll need to reload things with:

```
$ sudo systemctl daemon-reload
```

as the system will warn you.

To keep an eye directly on the output you can do:

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

(use `/var/log/syslog` on **Ubuntu**) either in background or in another window while the service is running.

To set things up so the service turns on or off on system boot:

```
$ sudo systemctl enable fake.service
$ sudo systemctl disable fake.service
```

Once again, you really need to reboot to make sure it has taken effect.

Chapter 4

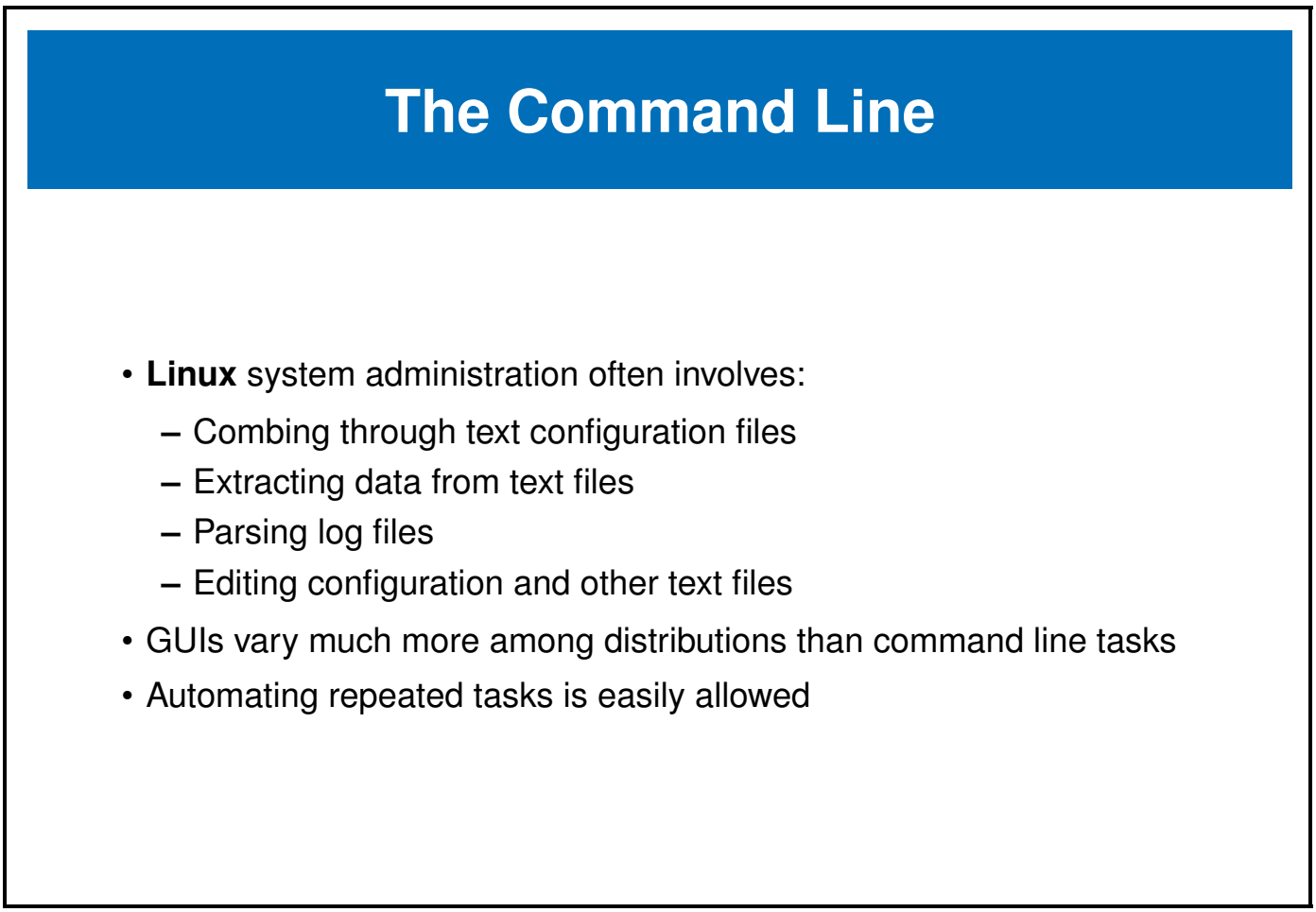
Command-line Operations

Learning Objectives

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

- Use the command line to perform operations in **Linux**
- Log in and log out, reboot and shutdown systems.
- Locate both applications and files.
- Understand directories (folders) and paths (both relative and absolute).
- Use wildcards efficiently.
- Reset the command line prompt to a desired format.
- Install and update software packages.

4.1 Command Line Operations and Options



The bread and butter of a **Linux** system administrator is the command line.

The bulk of system administration tasks usually involve manipulating text files in one way or another.

The reason the command line is almost always the interface of choice is because there is less overhead than the graphical user interface and it allows for automation. Furthermore, its use tends to vary less among **Linux** distributions and desktops.

Good system administrators write scripts to automate their tasks.

Scripts are usually customized and standardized to each environment but they usually utilize the various built-in tools outlined in this course.

Why Use Text Mode?

- Avoid larger GUI overhead
- All tasks can be performed and replicated at the command line
- Ability to script often-performed tasks
- Ability to **SSH** into remote machines
- If needed, can launch graphical applications from text mode command line

Most **Linux** system administrators spend most, if not all, of their time at the command line prompt, troubleshooting, scripting and running various tasks.

The ability to automate and troubleshoot in the text environment makes **Linux** an efficient and flexible operating system. There is an oft-quoted saying:

> "graphical user interfaces make easy tasks easy, while command line interfaces make difficult tasks possible".

The power of **Linux** relies heavily on the abundance of often-used and well-documented command line tools.

Not Installing or Disabling the GUI

- Server systems are often installed without **X** or any other graphical desktop
- Thus, most distributions make installation of a **GUI** optional
- Even if a **GUI** desktop is installed, it can be disabled by booting only into a non-graphical runlevel
- In either case, the system boots directly into a text console mode

The customizable nature of **Linux** allows you to:

- Not install **X**, **Wayland**, or another Graphical User Interface (GUI)
- Remove the GUI after installation
- Leave the GUI in place, but disable many or all of its features and services.

Certain **Linux** distributions distinguish versions of the install media between desktop (with a graphical interface) and server (without one).

Production servers are usually installed without a **GUI**. This is helpful in maintaining a lean system that is intended for situations that do not require a graphical desktop. Minimizing what is installed generally leads to a diminished attack service and hence a more secure system.

Virtual Terminal Consoles

- **Linux** provides multiple **virtual terminals** (**VT**s) by default
  - Such **VT**s can be used for either text or graphical interfaces
- Use `CTRL-ALT-Fn` key combinations to switch
  - Typically `CTRL-ALT-F1` or `CTRL-ALT-F7` for graphical interface
  - Text terminals on the other function keys
- If not in a graphical **VT** can just hit `ALT-Fn`

A VT is not the same as no GUI

Switching VT on a GUI desktop gives a display screen identical to the one seen when booting a pure text mode runlevel.

However, it is not really the same: one can easily switch back to a full GUI, and still encounter the full overhead of the graphical desktop.

Virtual terminals in **Linux** are independent consoles that use the connected display and keyboard. They are considered "virtual" because there are several of them, but only one is visible at any given time.

One virtual terminal (usually number 1 or 7) is reserved for the graphical environment, and text logins are enabled on the unused **VT**s.

To switch between them when in a graphical environment, use `CTRL + ALT + Fn`; For example `CTRL + ALT + F6` for switching to **VT** 6. (When not in a graphical environment, you can avoid using the `CTRL` key, and just do `ALT + F6` for example.)

An example of a situation where the virtual terminals can be very useful, is when there are problems with the graphical desktop. Changing to a text **VT** provides a great troubleshooting environment.

Turning the Graphical Desktop Off

- To start, stop or restart the graphical desktop:

```
$ sudo systemctl [stop|start|restart] [gdm|lightdm|kdm]
```

- On many distributions, this can also be done by setting the **runlevel**, as in:

```
$ sudo telinit 3
$ sudo telinit 5
```

Runlevels are outmoded

systemd still interprets runlevels on most systems, but this is just a backwards compatible option, and is probably best not to frame things this way any more.

Exactly which method is used to start, stop or restart the graphical desktop used to depend quite a bit on the **Linux** distribution and version in use.

Modern systems from all distributions now are generally configured to use **systemd** to start and stop as well as enable and disable services. Thus, the utility used is **systemctl**.

Text Terminals on the Graphical Desktop

- Several essential utility programs emulate a text-mode terminal as a window on a graphical desktop
- Many multiple text windows can be open simultaneously
- Each text window can also have multiple **tabs**
- Most modern systems use **gnome-terminal** by default
- **xterm**, available for decades, is still available but rarely used
- Other options available include:
  - **rxvt**
  - **konsole**
  - **terminator**

  where **konsole** is the terminal emulator packaged with the **KDE** desktop system.

A terminal emulator program on the desktop works by emulating a terminal within a window on the desktop. All features of regular text mode are supported.

gnome-terminal, **konsole** and other modern terminal emulators support:

- Multiple simultaneous windows
- Multiple tabs within each windows
- Extensive customization and multiple **profiles**, each of which can have custom fonts, colors, title bars, etc.

4.2 Logging In and Out, Rebooting and Shutting Down

Logging In and Out

- Logging In
  - Log in via text terminal
  - Log in remotely via console:

```
$ ssh student@somewhereelse.com
```

- Logging Out
  - Log out via text terminal:

```
$ exit
$ logout
```

An available text terminal will present a prompt for a username, usually ending with the string `login:`. Type your username and press `Enter`. You will then be asked for your password.

Passwords are hidden

When typing the password, nothing is displayed on the terminal, not even a generic symbol to indicate typing progress. After typing your password, press `Enter`.

Once your session is started (either by logging in to a text terminal or via a graphical terminal program), you can connect and log in to remote systems via **SSH** (**S**ecure **SH**ell), which either uses passwords, as with regular logins, or cryptographic keys to prove your identity across the network.

Logging out of a text-mode session is done by typing `exit` or `logout`.

Note that an **SSH** session is separate from the local session from which you started it, so you may need to log out multiple times to completely exit a session.

Rebooting and Shutting Down

- Rebooting the system:

```
$ sudo reboot
```

or

```
$ sudo shutdown -r now
```

- Shutting down the system:

```
$ sudo halt
```

or

```
$ sudo poweroff
```

or

```
$ sudo shutdown -h now
```

If you are working on a graphical desktop, it is probably preferable to logout, restart or shutdown using the always available button. This may do a better job of terminating properly any running programs.

The preferred method to shut down or reboot the system is to use **shutdown**, which sends a warning message and prevents new users from logging in. The **init** process will then follow its procedure for shutting down or rebooting the system. It is important to always shut down properly; failure to do so can result in damage to the system.

halt and **poweroff**, which call **shutdown -h**, perform the essential duties required to shut the system down.

reboot calls **shutdown -r**, which causes the machine to reboot instead of halt. Note that rebooting or shutting down from the command line requires superuser (root) access.

To shutdown and remind users of a scheduled maintenance:

```
$ shutdown -h 10:00 "Shutting down for scheduled maintenance."
```

4.3 Setting Time and Date

Setting Time and Date with timedatectl

- **timedatectl** is a **systemd**-supplied program for setting date, time and timezone etc.

```
$ timedatectl -a
```

```
               Local time: Tue 2022-08-09 12:24:42 CDT
           Universal time: Tue 2022-08-09 17:24:42 UTC
                 RTC time: Tue 2022-08-09 17:24:42
                Time zone: America/Chicago (CDT, -0500)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

```
$ timedatectl show
$ timedatectl set-timezone America/Toronto
$ timedatectl set-ntp 1
$ timedatectl set-time 2022-05-27
```

```
$ timedatectl -h
```

```
timedatectl [OPTIONS...] COMMAND ...

Query or change system time and date settings.

Commands:
  status                  Show current time settings
  show                    Show properties of systemd-timedated
  set-time TIME           Set system time
  set-timezone ZONE       Set system time zone
  list-timezones          Show known time zones
  set-local-rtc BOOL      Control whether RTC is in local time
  set-ntp BOOL            Enable or disable network time synchronization

systemd-timesyncd Commands:
  timesync-status         Show status of systemd-timesyncd
  show-timesync           Show properties of systemd-timesyncd

Options:
  -h --help               Show this help message
     --version            Show package version
     --no-pager           Do not pipe output into a pager
     --no-ask-password    Do not prompt for password
  -H --host=[USER@]HOST   Operate on remote host
  -M --machine=CONTAINER  Operate on local container
     --adjust-system-clock Adjust system clock when changing local RTC mode
     --monitor            Monitor status of systemd-timesyncd
  -p --property=NAME      Show only properties by this name
  -a --all                Show all properties, including empty ones
     --value              When showing properties, only print the value

See the timedatectl(1) man page for details.
```

Network Time Protocol (NTP)

- Standard protocol for retrieving the current time from the Internet
- Historically, the standard configuration file for **NTP** utilities was `/etc/ntp.conf`
- **RHEL**/**CentOS**/**Fedora** now ship with **chrony**:

```
$ sudo systemctl status chronyd
```

- **Ubuntu** still ships with **ntp**:

```
$ sudo systemctl status ntpd
$ sudo systemctl status systemd-timesync.service
```

- Can always turn off and set the time manually

NTP is the most popular and reliable protocol for setting the local time via Internet servers.

Most **Linux** distributions ship with a working **NTP** setup which refers to specific time servers run by the distribution, which means that there is almost never any setup needed for network time synchronization beyond **on** or **off**.

On older distributions (and **Ubuntu**) more detailed configuration is possible by editing `/etc/ntp.conf`.

Most recent distributions use **chrony** as the **ntp** interface. On **RHEL**/**CentOS**/**Fedora** and **OpenSUSE** systems configuration can be done with `/etc/sysconfig/chronyd`.

4.4 Locating Applications

Locating Applications

- Programs are usually located in:
  - /bin: Essential binary programs and scripts
  - /usr/bin: Non-essential binary programs and scripts
  - /sbin: Essential system binary programs and scripts
  - /usr/sbin: Non-essential system binary programs and scripts
  - /opt: Optional software application packages
- On many recent systems:

```
[student@Fedora ~]$ ls -lF / | grep bin
lrwxrwxrwx.   1 root root     7 Jul 12  2018 bin -> usr/bin/
lrwxrwxrwx.   1 root root     8 Jul 12  2018 sbin -> usr/sbin/
[student@Fedora ~]$
```

- To locate binaries:

```
$ which
$ whereis
```

Depending on choice of **Linux** distribution and the policies of your organization, programs and software packages can be installed in a number of directories. Generally these packages can be located in the /bin, /usr/bin or /opt directories.

A common way to locate the specific location of a program is to run the **which** utility. For example, to find **emacs**:

```
$ which emacs
/usr/bin/emacs
```

If **which** does not find the package that you are looking for, **whereis** is a good alternative. **whereis** looks for packages in a broader range of system directories.

```
$ whereis emacs
emacs: /usr/bin/emacs /usr/libexec/emacs /usr/share/emacs /usr/share/man/man1/emacs.1.gz
```

4.5 Directories and Paths

Accessing Directories

- Display present working directory

```
$ pwd
$ echo $PWD
```

- Change to your home directory

```
$ cd
$ cd ~
$ cd $HOME
```

- Change to parent directory

```
$ cd ..
```

- Change to previous directory

```
$ cd -
```

Upon logging into a system or opening a terminal, the default directory depends on the **Linux** distribution and user-defined preferences.

On most distributions you are placed by default in the `Desktop` subdirectory off your home directory, i.e., `$HOME/Desktop`, rather than the more convenient home directory itself, which you can print the path for, by simply typing `echo $HOME`.

Your current directory can be ascertained by using the **pwd** utility, or typing `echo $PWD` since the current directory is always stored in the `PWD` environmental variable.

A quick way to jump back to your home directory is to either run `cd` or `cd ~`. If you find that you need to go back to your previous directory without having to type a long path name, run `cd -`.

```
$ pwd
```

```
/home/bjmoose
```

```
$ cd /tmp
$ cd
$ pwd
```

```
/home/bjmoose
```

```
$ cd ..
$ pwd
```

```
/home
```

```
$ cd -
```

```
/home/bjmoose
```

```
$ cd ~
$ pwd
```

```
/home/bjmoose
```

Absolute Path vs Relative Path

- Absolute path
  - The pathname for a file or directory beginning at the root directory
  - Always starts with a /

  `/etc/passwd`
- Relative path
  - The pathname that shows a path in relation to the current working directory
  - Never starts with a /

  `../passwd`

There are two ways to identify paths: absolute pathnames or relative pathnames.

An absolute pathname begins with the root directory (/), and follows the tree branch by branch until it reaches the desired directory or file.

A relative pathname starts from the present working directory.

Using the absolute pathname is straightforward, while using the relative pathname usually uses the shortcuts provided by . (present directory), .. (parent directory) and ~ (home directory).

Usually one goes with the path that requires the least typing.

The following two ways will bring you to the same directory from your home directory:

```
$ cd /usr/bin
$ cd ../../usr/bin
```

Exploring the Filesystem

- Get to the root of the filesystem:
  ```
  $ cd /
  ```
- List files and subdirectories in current directory:
  ```
  $ ls
  ```
- List hidden files and hidden directories:
  ```
  $ ls -a
  ```
- Get a tree view of the filesystem:
  ```
  $ tree
  ```
- Create a shortcut using a **symbolic link**:
  ```
  $ ln -s /lib/modules/5.19 5.19
  $ cd 5.19
  ```

Traversing up and down the file system tree can get tedious. The **tree** command is a good way to get a bird's-eye view of the file system tree. Use **tree -d** to view just the directories and obscure the files.

An easy way to create shortcut from your home directory to long pathnames is to create symbolic links using `ln -s`.

```
c9:> cd / ; ls -F
```

```
ALL/         bin@   dev/   home/        lib64@       mnt/  PICTURES/  RESOURCES/  run/   sys/       usr/  W10/
AUDIO/       boot/  etc/   ISO_IMAGES/  lost+found/  opt/  proc/      RHEL9/      sbin@  teaching@  var/
backitup.sh  DEAD/  FULL/  lib@         media/       P/    Q8/        root/       srv/   tmp/       VMS/
```

```
c9:/>tree | head -10
```

```
.
|-- ALL
|   |-- local
|   |   |-- bin
|   |   |   |-- addkernel.sh
|   |   |   |-- airplane.sh
|   |   |   |-- atob
|   |   |   |-- avconv
|   |   |   |-- avprobe
|   |   |   |-- awkit.sh
```

```
c9:/tmp>ln -s /lib/modules/5.19.11 /tmp/5.19.11 ; cd /tmp/5.19.11
c9:/tmp/5.19.11>pwd
/tmp/5.10.11
c9:/tmp/5.19.11>/bin/pwd
/usr/lib/modules/5.19.11
c9:/tmp/5.19.11>
```

Note the first **pwd** gives a different result than using /**bin**/**pwd**, as it is the built-in version residing in **bash**.

Directory History, popd and pushd

- Change to previous directory

  `$ cd -`

- Change to a named directory, remembering where you were

  `$ pushd some/path`

- Going back to the remembered directory

  `$ popd`

- Display directory list

  `$ dirs`

When using **cd**, the system remembers where you last were, and lets you get back there with `cd -`.

However, in order to remember more than just the last directory visited, use **pushd** to change directory instead of **cd**; this pushes your starting directory onto a list.

Using **popd** will then send you back to those directories in reverse order (i.e. the most recent directory will be the first one retrieved with **popd**). The list of directories is displayed with the **dirs** command.

```
$ pwd
```

```
/home/rjsquirrel
```

```
$ cd /tmp
$ cd -
```

```
/home/rjsquirrel
```

```
$ pushd /tmp
```

```
/tmp ~
```

```
$ pushd /var/tmp
```

```
/var/tmp /tmp ~
```

```
$ dirs
```

```
/var/tmp /tmp ~
```

```
$ popd
```

```
/tmp ~
```

```
$ pwd
```

```
/tmp
```

4.6 Wildcards

Wildcards and File Name Matching

- **bash** (and other interpreters) recognizes special wildcard characters in file names:
  - `?`: The question mark matches any single character
  - `*`: The star (or asterisk) matches any string of characters
  - `[set]`: Matches any character in `set`
  - `[!set]`: Matches any character **not** in `set`
- There are some other variations, such as

```
$ ls -l lab1[1-4].tex
$ ls -l lab1[1,4].tex
```

  and sometimes squiggly brackets can be used (`{}`) instead of square brackets (`[]`)

Here are some examples of using bash wildcard characters for file name matching:

```
$ ls -a
```

```
.   bar.c  bar.out  baz.o    .dot-file  foo.o
..  bar.o  baz.c    baz.out  foo.c      foo.out
```

```
$ ls ba?.?
```

```
bar.c bar.o baz.c baz.o
```

```
$ ls bar.?
```

```
bar.c bar.o
```

```
$ ls *.out
```

```
bar.out baz.out foo.out
```

```
$ ls baz.*
```

```
baz.c baz.o baz.out
```

```
$ ls b*.?
```

```
bar.c bar.o baz.c baz.o
```

```
$ ls ba[a-s].c
```

```
bar.c
```

```
$ ls ba[!a-s].c
```

```
baz.c
```

```
$ ls -d .*
```

```
. .. .dot-file
```

```
$ ls -d .?*
```

```
.. .dot-file
```

```
$ ls -d .??*
```

```
.dot-file
```

4.7 Searching for Files

locate

- Find files whose names contain a string:

```
$ locate ffmpeg
```

```
/usr/bin/ffmpeg
/usr/include/ffmpeg
/usr/include/ffmpeg/libavcodec
/usr/include/ffmpeg/libavdevice
/usr/include/ffmpeg/libavfilter
...
```

- Database is updated by **updatedb**
  - Updated every night (by default)
  - Can be updated manually by

```
$ sudo updatedb
```

When invoked, **locate** performs a database search for pathnames including the string given as an argument.

It is common for this to return a long list of results; efficient use, therefore, often includes piping to **grep** to narrow the search (we will discuss **grep** in detail later):

```
$ locate zip | grep bin
```

```
/usr/bin/bunzip2
/usr/bin/bzip2
/usr/bin/bzip2recover
/usr/bin/funzip
/usr/bin/gpg-zip
/usr/bin/gunzip
....
```

The accuracy of the results from **locate** depend on having an up to date database, which is created and maintained by **updatedb**. This database is usually updated once a day. To give further accurate search results, you may run **updatedb** manually as root before running **locate**.

```
$ locate file-just-created
$ touch file-just-created
$ sudo updatedb
$ locate file-just-create
```

```
/home/student/file-just-created
```

Find

- Search for files in a directory and its subdirectories:
  `$ find [location] [criteria] [actions]`
  - `location`:
    The directory (or list of directories) in which to perform the search. If omitted, current directory (`.`) assumed
  - `criteria`:
    Tests such as name, date of creation etc, used to narrow search. If omitted, all files displayed
  - `actions`:
    Procedures to perform on the results. If omitted, obtain a non-detailed listing of file names
- `$ find /tmp -name "*.temp" -exec ls {} ';'`
  `{}` a placeholder for search results

find is an extremely useful command in the daily life of a **Linux** system administrator.

find traverses down the file system tree from one or more pathnames, and locates files that meet specified conditions.

Useful criteria include:

- `-name`: Matches any file whose name includes a string
- `-iname`: Case insensitive file name match
- `-type`: Matches only files of a certain type, such as directory, normal file etc.
- `-ctime` and `-newer`: Find files of certain age or newer than another file
- `-size` Matches based on size

Any criteria can be negated, as in:

```
$ find . ! -newer timestampfile
```

and criteria can be combined as in:

```
$ find . -newer timestampefile -and -type d
```

Using find to Execute Commands

- Get detailed (long) file listings:

  ```
  $ find . -name "*.tex" -ls
  ```

  or

  ```
  $ find . -name "*.tex" -exec ls -l {} ';'
  ```

- Remove the files:

  ```
  $ find . -name "*.tex" -exec rm {} \;
  $ find . -name "*.tex" -exec rm {} ';'
  ```

  or

  ```
  $ find . -name "*.tex" -ok -exec rm {} \;
  ```

  Note

  - `{}` is a placeholder for search results
  - Can use either `\;` or `';'` to denote the end of the command
  - Using `-ok` prompts for each action, good to use first for a “dry-run”

Another good use of **find** is being able to run commands on the files that match your search criteria. The `-exec` option is used for this purpose.

To find and remove all files that end with `.swp` :

```
$ find -name "*.swp" -exec rm {} \;
```

The `-ok` behaves the same as `-exec` except that **find** will prompt you for permission before executing the command. This makes it a good way to test intended actions before blindly executing any commands.

4.8 Command Prompt

Customizing the Command Line Prompt

- Default prompt:
  - $ for normal users
  - # for root, super-user
- Modified by setting `PS1` environment variable:

```
$ PS1="\h:\u:\w>"
  c9:coop:/tmp>
```

- To make persistent for a single user, edit `~/.bashrc`
- To make persistent for all users, edit `/etc/bashrc` on most **Linux** distributions

Most distributions set the `PS1` variable to a known default state.

While this default value works in most cases, some people may want some custom information to be present on the command line.

For example, some system administrators require the host name to show up on the command line, e.g.,

```
user@hostname $
```

Table 4.1: **Controlling the Prompt**

Character	Meaning	Example Output
\t	time in HH:MM:SS	08:43:40
\d	date in "Weekday Month Date"	Fri Mar 12
\n	newline	
\s	shell name	bash
\w	current working directory	/usr/local/bin
\W	basename of current working directory	bin
\u	user	coop
\h	hostname	c8
\#	command number (this session)	43
\!	history number (in history file)	1057

4.9 Package Management

Package Management

- All core software provided in **packages**
  - Bundle of files that together provide a software component
  - Can depend on each other
- Two basic systems:
  - **Debian**-based: **Debian**, **Ubuntu**, **Mint Linux**, etc.
  - **RPM**-based: **RHEL/CentOS/Fedora**, **SUSE/openSUSE**, etc.
- Two levels of package management software:
  - Individual package(s): **rpm**, **dpkg**
  - Including system dependencies: **apt-get, apt** , **dnf**, **zypper**
- Most distributions have a graphical interface, although it can be hard to find.
- Command line interface generally faster and more efficient.

The core parts of a **Linux** distribution and most of its add-on software are installed via the package management system.

Each package contains the files and other instructions needed to make one software component work on the system. Packages can depend on each other; for example, a package for a Web-based application written in **PHP** can depend on the **PHP** package.

There are two broad families of package managers: those based on **Debian**, and those which use **RPM** as their low-level package manager. The two systems are incompatible (although there are translation utilities), but broadly speaking provide the same features.

Avoid Packaging System Holy Wars

Many **Linux** users have strong opinions about which system is better; this is one of the many holy wars you can find in the community.

It is best to simply learn at least the basic essentials for both systems.

There are other distributions that use their own custom crafted systems, such as **PacMan** on **ArchLinux** and **emerge** on **GENTOO**. We will not discuss these systems as they are not as common and users tend to be relatively advanced and familiar with them.

Package Managers: Two Levels

- Low-level package manager: **dpkg** or **rpm**
  - Does the work of unpacking, copying, etc.
  - Not very intelligent working with multiple packages
- High-level package manager: **apt**, **dnf**, **zypper**, etc.
  - Works with repositories
  - Figures out dependencies
  - Handles updates

Both package management types provide two levels: a low-level tool, which takes care of the details of unpacking individual packages, running scripts, and otherwise getting the software installed correctly, and a high-level tool that works with groups of packages, downloads packages from the vendor, and figures out dependencies.

Most of the time, users will work with the high-level tool, which will take care of calling the low-level tool as needed.

Dependency tracking is a particularly important feature of the high-level tool, as it handles the details of finding and installing each dependency for us. Be careful, however, as installing a single package could result in many dozens or even hundreds of dependent packages being installed.

Basic Packaging Commands

Table 4.2: **Basic Packaging Commands**

Operation	RPM	deb
Install a package	`rpm -i foo.rpm`	`dpkg --install foo.deb`
Install a package with dependencies from repository	`dnf install foo`	`apt install foo`
Remove a package	`rpm -e foo.rpm`	`dpkg --remove foo.deb`
Remove a package and dependencies using a repository	`dnf remove foo`	`apt remove foo`
Update package to a newer version	`rpm -U foo.rpm`	`dpkg --install foo.deb`
Update package using repository and resolving dependencies	`dnf update foo`	`apt install foo`
Update entire system	`dnf update`	`apt dist-upgrade`
Show all installed packages	`rpm -qa` or `dnf list installed`	`dpkg --list`
Get information about an installed package including files	`rpm -qil foo`	`dpkg --listfiles foo`
Show available packages with “foo” in name	`dnf list foo`	`apt-cache search foo`
Show all available packages	`dnf list`	`apt-cache dumpavail`
What package does a file belong to?	`rpm -qf file`	`dpkg --search file`

Most of the time administrators and users will use the higher level commands (**dnf**, **zypper**, **apt**), as dependency resolution is a critical component of package management.

- If you try to remove a package with the lower level commands (**rpm**, **dpkg**) it often will fail because another package requires it.
- If you try to install a package with the lower level commands, it often will fail because other packages need to be installed first.

Dependency management is thus essential. There are many other useful commands as well, so please see the appropriate **man** pages for more detail.

apt

- Available on **Debian**-based distributions
- Update the software catalogs:

```
$ sudo apt update
```

- Install a package:

```
$ sudo apt install package
```

- Remove a package:

```
$ sudo apt remove package
```

- Install all software updates:

```
$ sudo apt [upgrade|dist-upgrade]
```

- Get the status of a package:

```
$ sudo apt-cache policy package
```

- Search for a package:

```
$ sudo apt-cache search keyword
```

The **A**dvanced **P**ackaging **T**ool (**APT**) is a package management system that manages installed software on **Debian**-based systems. While it forms the back end for graphical package managers, its native user interface is at the command line, with programs that include **apt**, **apt-get** and **apt-cache**.

apt-get vs apt

You will often see **apt** rather than **apt-get** used in commands. Most of the time it makes no difference which you use. However, some prefer to use **apt-get** at the command line; **apt** is better designed for use in non-interactive scripts.

If you know the name of the package, it is very easy to install a package using

```
$ sudo apt install terminator
```

```
        . . .
After this operation, 13.1 MB of additional disk space will be used.
Do you want to continue [Y/n]?
```

To search for a package in the archive run `apt-cache search keyword`, which will search for the keyword in the package name or the description of the package. The main configuration file is `/etc/apt/sources.list` which lists the archives of the package distribution system in use on the system.

```
$ sudo apt-cache search terminator
```

```
ruby-rails-observers - toolkit to build Rails observers (part of Rails)
terminator - multiple GNOME terminals in one window
terminatorx - realtime audio synthesizer
```

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

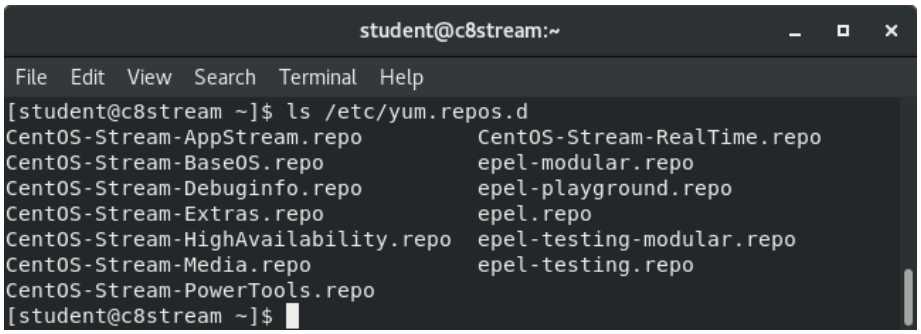


Figure 4.1: **rpm Repositories**

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.

dnf Examples

- Install package:

  ```
  $ sudo dnf install package
  ```

- Remove package:

  ```
  $ sudo dnf remove package
  ```

- Search installed package:

  ```
  $ sudo dnf search keyword
  ```

- Get information on an installed package

  ```
  $ sudo dnf info package
  ```

- List all available packages, both installed and not installed

  ```
  $ sudo dnf list info package
  ```

In order to install the **atop** package, do:

```
$ sudo dnf install atop
```

```
Updating Subscription Management repositories.
Last metadata expiration check: 0:16:33 ago on Mon 25 Jan 2021 09:18:06 AM CST.
Dependencies resolved.
================================================================================
 Package         Architecture      Version                Repository       Size
================================================================================
Installing:
 atop            x86_64            2.4.0-4.el8            epel            187 k

Transaction Summary
================================================================================
Install  1 Package

Total download size: 187 k
Installed size: 432 k
Is this ok [y/N]: y
Downloading Packages:
atop-2.4.0-4.el8.x86_64.rpm                    526 kB/s | 187 kB     00:00
.....
Running transaction
  Preparing        :                                                        1/1
  Installing       : atop-2.4.0-4.el8.x86_64                                1/1
  Running scriptlet: atop-2.4.0-4.el8.x86_64                                1/1
  Verifying        : atop-2.4.0-4.el8.x86_64                                1/1
Installed products updated.

Installed:
  atop-2.4.0-4.el8.x86_64

Complete!
```

zypper

- Available on **SUSE**-based systems including **openSUSE**
- Install package:

```
$ sudo zypper install package
```

- Remove package:

```
$ sudo zypper remove package
```

- Search installed package:

```
$ sudo zypper search string
```

- Low level **rpm** commands work the same as on **Red Hat**-based systems.

zypper is **SUSE**'s **RPM**-based package management system. zypper is fairly straightforward to use and is similar to **dnf**, and slightly less so to **apt**.

To add and remove repositories, and then packages, with **zypper**:

```
$ sudo zypper addrepo url alias
$ sudo zypper removerepo alias

$ sudo zypper install terminator
```

```
        . . .
Continue? [y/n/? shows all options] (y):
        . . .
(29/29) Installing: terminator-lang-0.97-1.1 ...................[done]
```

```
student@linux-n292:~> sudo zypper remove terminator
```

```
        . . .
Continue? [y/n/? shows all options] (y):
(1/2) Removing terminator-lang-0.97-1.1 .......................[done]
(2/2) Removing terminator-0.97-1.1 ............................[done]
```

```
student@linux-n292:~> sudo zypper search terminator
```

```
        . . .
S | Name            | Summary                                  | Type
--+-----------------+------------------------------------------+--------
  | terminator      | Store and run multiple GNOME terminals i| package
  | terminator-lang | Languages for package terminator         | package
```

4.10 Labs

Video Demonstration Resources

find_demo.mp4
locatefiles_demo.mp4
using_dnf.mp4
apt.mp4
zypper.mp4

Exercise 4.1: Using timedatectl to set time, date and timezone

1. Save your current values so you can restore them when the lab is done!
2. Change your timezone to something different.
   Hint: Run

   ```
   $ timedatectl list-timezones
   ```

   first to understand the required format.
3. Change the local time to something in the past or future. **Hint:** You may have to turn off **ntp** first.
4. Restore your original values!

Solution 4.1

1. ```
   $ timedatectl status > timedatesaved.output
   ```
2. ```
   $ timedatectl set-timezone Europe/Stockholm
   ```
3. ```
   $ timedatectl set-time "2011-01-01 14:03:12"
   ```

   ```
   Failed to set time: Automatic time synchronization is enabled
   ```

   ```
   $ timedatectl set-ntp 0
   $ timedatectl status | grep Local
                Local time: Sat 2011-01-01 14:03:58 CST
   ```
4. Restoring the actual time should be easy by re-invoking **ntp**:

   ```
   $ timedatectl set-ntp 1
   ```

   And reset the timezone to its original value.

Exercise 4.2: Locating applications

Find out the location of the **ip** network utility.

Solution 4.2

```
$ which ip
```

```
/usr/sbin/ip
```

```
$ whereis ip
```
```

```
ip: /usr/sbin/ip /usr/share/man/man7/ip.7.gz /usr/share/man/man8/ip.8.gz
```

Note **whereis** also reports the location of the **man** page.

Exercise 4.3: Finding directories and creating symbolic links

Find the directory `init.d`, starting from `/`, and then create a symbolic link from within your home directory to this directory.

Solution 4.3

```
$ find / -type d -name init.d
$ cd ~
$ ln -s /etc/init.d .
```

Exercise 4.4: Creating, moving and removing files

Create an empty file named `exercise.txt` and move this file to the `/tmp` directory using a relative pathname from your home directory. Then delete this file using an absolute pathname.

Solution 4.4

```
$ touch exercise.txt
$ mv exercise.txt ../../tmp/
$ rm /tmp/exercise.txt
```

Exercise 4.5: Changing the Command Line Prompt

It is nice to have your current working directory as part of your prompt so that a quick glance will give you some information without typing **pwd** every time.

If you often work on multiple computers, especially if you network from one into another with **ssh**, it is very convenient to have the computer name be part of your prompt.

1. Put your current working directory in your command line prompt.
2. Put your computer (machine) name in your prompt.
3. Put both your current directory and computer name in your prompt.

How can you make this persistent, so that whenever you start a **bash** command shell, this is your prompt?

Solution 4.5

1. `$ echo $PWD`

   ```
   /tmp
   ```

   `$ PS1='\w>'`

   ```
   /tmp>
   ```

2. `/tmp> PS1='\h>'`

   ```
   student7>
   ```

3. `student7> PS1='\h:\w>'`

```
student7:/tmp>
```

To make the changes permanent, add the following line to your `$HOME/.bashrc` resource file:

$HOME/.bashrc

```
export PS1='\h:\w>'
```

and then start a new shell by typing **bash**, or opening up a new terminal window through any means you prefer.

You may want to remove that line after you certify that it is working as intended.

Exercise 4.6: Installing and Removing Software Packages

Using the upper level package management system appropriate for your **Linux** distribution:

1. Install the `dump` package on your system.
2. Remove the `dump` package from your system.

If **dump** is already installed (you will be told so when you try to install), then do things in opposite order, i.e., remove and then install.

Solution 4.6

Use whichever utility is appropriate to your system:

```
$ [apt-get | dnf | zypper ]  install dump
$ [apt-get | dnf | zypper ]  remove dump
```

Chapter 5

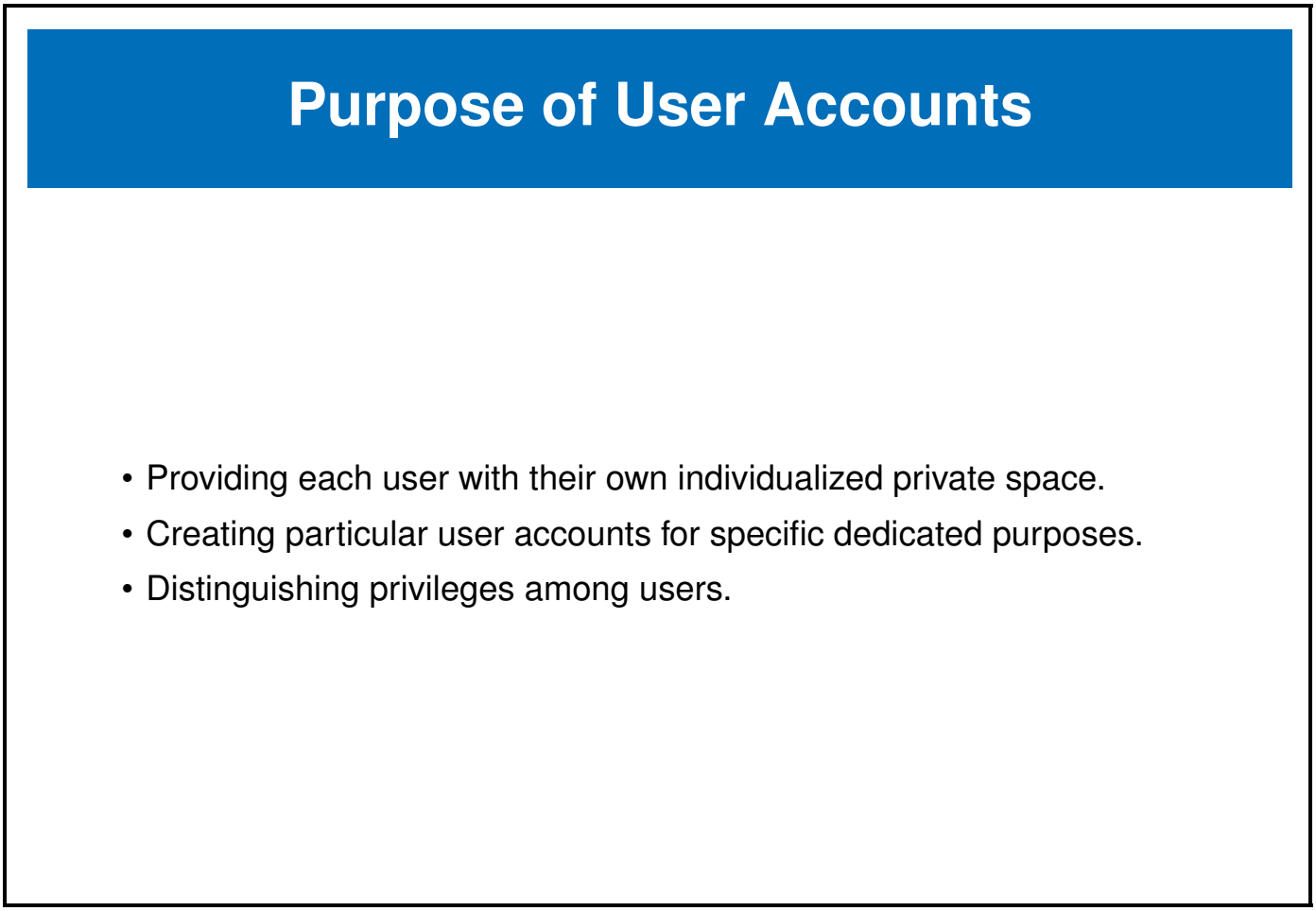
User Accounts and Environment

Learning Objectives

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

- Explain the purpose of user accounts and how they are used and managed.
- Describe the purpose of groups and how they are used and managed.
- Use system startup files, especially `$HOME/.bashrc`.
- Understand how passwords are properly chosen and used,, and the role of `/etc/passwd` and `/etc/group`.
- Set file ownership and permissions.
- Use **SSH**.
- Explain, set and use environment variables.
- Set and use keyboard shortcuts
- Define and use aliases.
- Use the previous command history to simplify command line interaction.

5.1 User Accounts



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.

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.

Determine the Current User

- **Linux** is a multi-user system
- Find out users who are currently logged in:
  - **who**
- Find out current user ID:
  - **whoami**

Linux is a multi-user system that could have many users logged in to the machine at any time. The user ID is how the system sorts out what a user has access to. Here are some commands to identify which users are logged in.

- To find the users who are currently logged in use the **who** command.
- To verify who you are currently logged in as, use the **whoami** command.

Note: Some commands can change your **effective user id**, so these commands are important for verifying who you are executing commands as.

For example: The **who** command will print information about the users currently logged in to the system.

```
$ who
```

```
bjmoose     :0           2015-03-17 07:55 (:0)
bjmoose     pts/0        2015-03-17 07:55 (:0)
bjmoose     pts/1        2015-03-17 07:55 (:0)
bjmoose     pts/2        2015-03-17 07:55 (:0)
bjmoose     pts/3        2015-03-18 15:13 (:0)
bjmoose     pts/4        2015-03-18 15:13 (:0)
rjsquirrel  pts/5        2015-03-18 16:54 (fbfalls7)
```

The command **whoami** will display the current user's identity.

```
$ whoami
```

```
bjmoose
```

root Account

- Root account has full access to system
- Never give root access to users
- To change to root account:

```
$ su
```

- To execute command as root:

```
$ sudo command
```

- **sudo** configuration stored in files in:

```
/etc/sudoers.d/
```

- **sudo** far preferred over **su**

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 a **su** sub-shell to execute commands that do not require elevated privileges.

5.2 Groups

Groups

- Allowing users to share a work area (directories, files etc.)
- Setting up file permissions to allow access to group members, but not the entire world
- Permitting certain specified users to access resources they would not be allowed to otherwise

Linux systems form collections of users called **groups**, whose members share some common purpose. To further that end, they share certain files and directories and maintain some common privileges; this separates them from others on the system, sometimes collectively called the **world**. Using groups aids collaborative projects enormously.

Users belong to one or more **groups**.

Groups are defined in `/etc/group`, which has the same role for groups as `/etc/passwd` has for users. Each line of the file looks like:

/etc/group

```
....
groupname:password:GID:user1,user2,...
....
```

- `groupname` is the name of the group.
- `password` is the password placeholder. 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.

5.3 Group Management

Group Management

Group accounts may be managed and maintained with:

- **groupadd**: Add a new group
- **groupmod**: Modify a group and add new users
- **groupdel**: Remove a group
- **usermod**: Manage a user's group memberships

These group manipulation utilities modify `/etc/group` and (if it exists) `/etc/gshadow`, and may only be executed by root.

Examples:

```
$ sudo groupadd -r -g 215 staff
$ sudo groupmod -g 101 blah
$ sudo groupdel newgroup
$ sudo usermod -G student,group1,group2 student
```

Note: Be very careful with the **usermod -G** command; the group list that follows is the complete list of groups, not just the changes. Any supplemental groups left out will be gone! Non-destructive use should utilize the `-a` option which will preserve pre-existing group memberships when adding new ones.

5.4 Shell Startup Files

Startup Files

- Shell uses multiple startup files to create environment
- Each file affects interactive environments differently
- Files in `/etc` provide global settings
- Files in home directory override global settings

When shells like Bash start, it is common to customize the shell's environment.

Elements like the bash prompt can be customized, here is an example of a customized prompt for user **student**:

```
student@ubuntu:~$
```

or for the **root** user:

```
root@ubuntu:~#
```

Other elements can be customized like this entry defining which version of an application is to be used:

```
TLYEAR=2021
#TLYEAR=2022
pathmunge /usr/local/texlive/$TLYEAR/bin/x86_64-linux/
```

The above examples are stored in `~/.bashrc`.

Advantages of Startup Files

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

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

Shell startup files provide an advantage over a single file or static environment variables by having several files that get used depending on the method of access.

The files used for interactive logins are slightly different from non-interactive login shells.

There are startup files that are common between interactive and non-interactive shells.

There are system default startup files that can be overridden by the users' personal files.

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`.

5.5 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.

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.

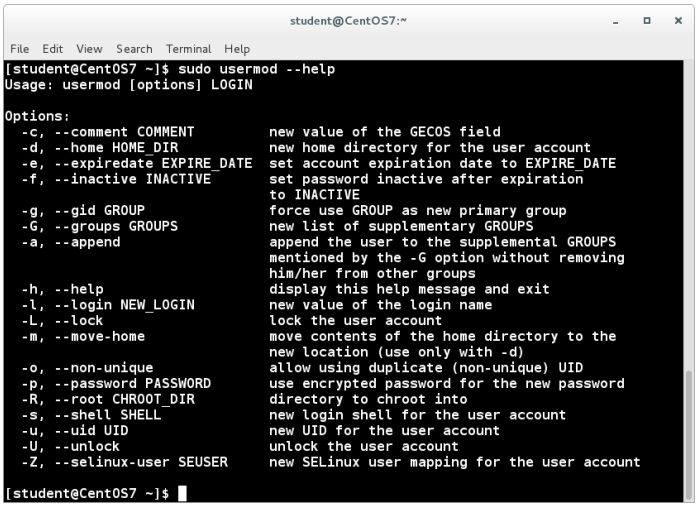


Figure 5.1: **Using usermod**

5.6 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 placeholder ("x" when /etc/shadow is used)
  - UID: user identification number
  - GID: primary group identification number for the user
  - comment: comment area, usually the user's real name
  - home directory pathname for the user's home directory
  - shell: absolutely qualified name of the shell to invoke at login
- UIDs start at UID_MIN = 1000

The convention most **Linux** distributions have used is that any account with a **UID** less than 1000 is considered special and belongs to the system; normal user accounts start at 1000. The actual value is defined as UID_MIN and is defined in /etc/login.defs.

Historically, **Red Hat**-derived distributions used UID_MIN=500, not 1000, but beginning with **RHEL 7** the more common value of 1000 was adopted.

If a **UID** is not specified when using **useradd**, the system will incrementally assign user IDs starting at UID_MIN.

Additionally, each user gets a **Primary Group ID** which by default is the same number as the **UID**. These are sometimes called **User Private Groups** (UPG).

It is bad practice to edit /etc/passwd, /etc/group or /etc/shadow directly: instead use appropriate utilities such as **usermod**.

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.

/etc/shadow II

- /etc/shadow file contains one record (one line) for each user
- Each record contains fields separated by colons (:)
  - username: unique user name.
  - password: the hashed (sha512) value of the password
  - lastchange: days since Jan 1, 1970 that password was last changed
  - mindays: minimum days before password can be changed
  - maxdays: maximum days after which password must be changed
  - warn: days before password expires that the user is warned
  - grace: days after password expires that account is disabled
  - expire: date that account is/will be disabled
  - reserved: reserved field

/etc/shadow contains one record (one line) for each user as in:

/etc/shadow

```
daemon:*:16141:0:99999:7:::
.....
beav:$6$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.

Password Management

- Passwords can be changed with **passwd**
- Users can change their own password
- Root can change any user password

By default, the password choice is examined by pam_cracklib.so, which furthers making good password choices.

A normal user changing their password:

```
$ passwd
```

```
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.

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
```

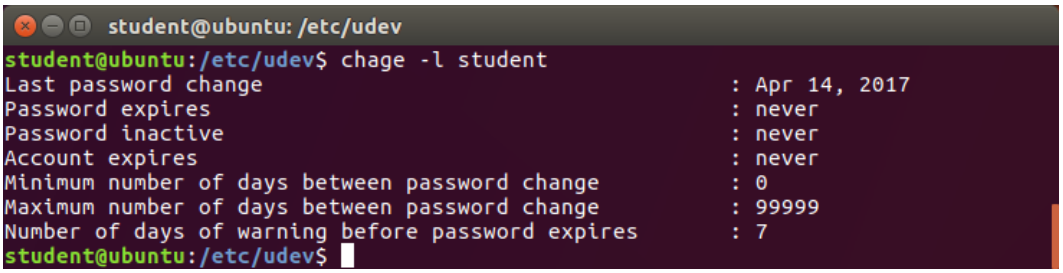


Figure 5.2: **Using chage**

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
```

5.7 File Ownership and Permissions

File Ownership

- Files are owned by one particular **user**
- Ownership can be changed by:

```
$ sudo chown owner filename
```

- Files are also owned by one particular **group**
- File group ownership is changed by:

```
$ sudo chgrp group filename
```

- These two operations can be combined in one step:

```
$ sudo chown owner:group filename
```

  And also done recursively through the directory tree:

```
$ sudo chown -R owner:group some_directory
```

- You must have proper privileges on the files to do these steps

```
$ ls -l
```

```
total 4
-rw-rw-r--. 1 rkimble rkimble    0 Mar 16 19:04 file-1
-rw-rw-r--. 1 rkimble rkimble    0 Mar 16 19:04 file-2
```

```
$ sudo chown root file-1 ; ls -l
```

```
[sudo] password for rkimble:

total 4
-rw-rw-r--. 1 root     rkimble    0 Mar 16 19:04 file-1
-rw-rw-r--. 1 rkimble  rkimble    0 Mar 16 19:04 file-2
```

```
$ sudo chgrp bin file-2 ; ls -l
```

```
total 4
-rw-rw-r--. 1 root     rkimble    0 Mar 16 19:04 file-1
-rw-rw-r--. 1 rkimble  bin        0 Mar 16 19:04 file-2
```

```
$ sudo chown sgerard:police file-? ; ls -l
```

```
total 4
-rw-rw-r--. 1 sgerard:poilce      0 Mar 16 19:04 file-1
-rw-rw-r--. 1 sgerard:police      0 Mar 16 19:04 file-2
```

You can also use a period (`.`) instead of a colon (`:`) as the delimiter between owner and group.

File Permissions

- Three kinds of permissions (`rwx`):
  - Read (`r`)
  - Write (`w`)
  - Execute (`x`)
- Three classes of users that permissions affect (`ugo`):
  - User/Owner (`u`)
  - Group (`g`)
  - Others (`o`)
- Three groups, three permissions, nine fields:

```
$ ls -l /home/spock/.bashrc

  u  g  o
 rwxrwxrwx
-rw-rw-r-- 1 spock spock 6987 Mar 18 17:25 /home/spock/.bashrc
```

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 the one of these three sets of permissions, in the following order:

- If the requester is the file owner, the file owner permissions are used.
- Otherwise, if the requester is in the group that owns the files, the group permissions are examined;
- If that doesn't succeed, the world permissions are examined.

Note that permissions can be changed with **chmod** and ownership with **chown**.

Changing File Permission

- The three kinds of permissions can be represented by an octal digit

```
rwx r-- rw- r-x
421 4   42  4 1
 7   4   6   5
```

- The set of permissions for the three owners can be represented similarly

```
rwxrwxrwx  rw-rw-r--  rwxr-xr-x  r--r-----
 7  7  7    6  6  4    7  5  5    4  4  0
```

- File permissions changed with:

```
$ chmod <octal-digits> filename
$ chmod <symbolic-form> filename
```

```
$ ls -l
```

```
total 4
-rw-rw-r--. 1 root mom    0 Mar 16 19:04 file-1
-rw-rw-r--. 1 mom  bin    0 Mar 16 19:04 file-2
drwxrwxr-x. 2 mom  mom 4096 Mar 16 19:04 temp
```

```
$ chmod 640 file-2
$ ls -l
```

```
total 4
-rw-rw-r--. 1 root mom    0 Mar 16 19:04 file-1
-rw-r-----. 1 mom  bin    0 Mar 16 19:04 file-2
drwxrwxr-x. 2 mom  mom 4096 Mar 16 19:04 temp
```

```
$ chmod o+w temp
$ ls -l
```

```
total 4
-rw-rw-r--. 1 root mom    0 Mar 16 19:04 file-1
-rw-r-----. 1 mom  bin    0 Mar 16 19:04 file-2
drwxrwxrwx. 2 mom  mom 4096 Mar 16 19:04 temp
```

5.8 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-get 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
```

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.)

ssh Configuration Files

- In home directory:
  - `id_rsa`: the user's **private** encryption key.
  - `id_rsa.pub`: the user's **public** encryption key.
  - `authorized_keys`: Public keys that are permitted to login.
  - `known_hosts`: Hosts from which logins have been allowed in the past.
  - `config`: File for specifying various options.
- Public and private keys generated with **ssh-keygen**

One can configure **SSH** further to expedite its use, in particular to permit logging in without a password. User-specific configuration files are created under every user's home directory in the hidden `.ssh` directory:

First a user has to generate their private and public encryption keys with **ssh-keygen**:

```
$ ssh-keygen
```

```
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. It should also be added to your `authorized_keys` file, together with all the public keys from other users who have accounts on your machine and you want to permit password-less access to their accounts. `known_hosts` is gradually built up as **ssh** accesses occur. If the system detects changes in the users who are trying to log in through **ssh** it will warn you of them and afford the opportunity to deny access.

Note that `authorized_keys` contains information about users and machines:

```
$ cat authorized_keys
```

```
ssh-rsa AAAAB3Nza........student@debian
ssh-rsa AAAAB3Nza........student@fedora
```

while `known_hosts` contains only information about computer nodes:

```
$ cat known_hosts
```

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

Do `man ssh_config` to see the options that can go into the **ssh** configuration files.

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
    ```

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-get] install tigervnc*
```

using the right package management system command. (The exact package name have varied between **Linux** distributions, so we are not giving exact package names. You may wind up installing more than you need, but the packages are not large.) Start the server as a normal user with:

```
$ vncserver
```

You can test with

```
$ vncviewer localhost:2
```

(You may have to play with numbers other than 2 (1, 3, 4, ...) depending on what you are running at the moment and how your machine is configured.) To view from a remote machine it is just slightly different:

```
$ vncviewer -via student@some_machine localhost:2
```

If you get a rather strange message about having to authenticate because of “color profile” and no passwords work, you have to kill the **colord** daemon on the server machine, as in

```
$ sudo systemctl stop colord
```

This is a bug (not a feature) and will only appear in some distributions and some systems for unclear reasons.

5.9 Environment Variables

Environment Variables

Environment variables are name/value pairs that can be accessed any time.

- An example of some common names of variable data:

```
HOME
HOST
PATH
```

- Can be listed in different formats with the following commands:

```
$ 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
```

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
```

Setting Environment Variables

The value of a name/variable pair can be displayed with the **echo** command and to set a new name/variable pair, use **name**=**value** format. Some examples are:

- List a specific variable by name:
  `$ 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

Display the current value of the SHELL variable.

```
$ echo $SHELL
```

```
    /bin/bash
```

Display the value used to describe the default editor, then change the editor to nano:

```
$ echo $EDITOR
```

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

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

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

Create a permanent assignment of **myvar** by editing the .bashrc file. The variable is not available until the bashrc file is re-read.

```
$ echo $MYVAR

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

$ source .bashrc
$ echo $MYVAR
```

```
    myvalue
```

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"
...
```

5.10 Key Shortcuts

Key Shortcuts

Shells like **bash** have key combinations that perform common functions or actions. The key combinations usually make use of **special** or **control** keys. Here are some examples.

- CTRL-L
  - Clears the screen
- CTRL-D
  - Exit the current shell
- CTRL-Z
  - Puts current process into suspended background (Can be reversed by typing **fg**)
- CTRL-C
  - Kills current process

Typing CTRL-L clears the screen. It has the same effect as typing **clear**. Try it!

Here are examples of the other key shortcuts listed above:

```
$ bash                  # Go into a new sub-shell
```

```
  sub-shell:$ exit      # Actually, a CTRL-D was typed and "exit" appeared
```

```
$ sleep 1000            # Sleep for 1,000 seconds
```

```
  ^Z                    # CTRL-Z typed after a few seconds
  [1]+  Stopped                 sleep 1000
```

```
$ fg                    # Resume execution of "sleep 1000" in foreground
```

```
$ sleep 1000
```

```
  ^C                    # Type CTRL-C after a few seconds
```

```
$                       # Kills the "sleep" and gives us the shell
```

More Key Shortcuts

- `CTRL-H`
  - Same as backspace
- `CTRL-A`
  - Go to beginning of the line
- `CTRL-W`
  - Delete word before the cursor
- `Tab`
  - Auto-complete files, directories and binaries

5.11 Command History

History

The **bash** shell maintains a file of commands typed into a console. This list of commands is referred to as the **history**. The **history** of the commands can be quite useful for looking for errors, repeating the same command and creating scripts from entered commands.

- 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`

```
$ echo $HISTFILE
```

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

```
$ echo $HISTFILESIZE
```

```
2000
```

```
$ echo $HISTSIZE
```

```
1000
```

```
$ history | head -5
```

```
40  cd RELEASE
41  cd LFS430
42  l
43  cd V1.2
44  l
```

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.

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`

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
```

5.12 Command Aliases

Aliases

Aliases are customizable shortcuts. A long command can be shortened to a few characters. Aliases are temporary if typed in at the command line and can be made persistent when added to the bash startup file. Here are some examples of 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'
```

5.13 Labs

Video Demonstration Resources

using_user_accounts_demo.mp4

Exercise 5.1: Accounts

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

Solution 5.1

Run the **whoami** utility:

```
$ whoami
student
```

Exercise 5.2: 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 5.2

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

   ```
   /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`

```
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
```

Exercise 5.3: Adding the /tmp directory to your path

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

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

Then make it executable by doing

```
$ chmod +x /tmp/ls
```

1. Append /tmp 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 /tmp/ls.
2. Pre-pend /tmp 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 /tmp/ls.

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

Solution 5.3

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

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

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

```
$ exit
```

2.
```
$ bash
$ PATH=/tmp:$PATH
$ ls /usr
```

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

```
$ exit
```

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 `/tmp`, they can trick you into running it accidentally.

Exercise 5.4: Command history

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 5.4

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 5.5: 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.

Solution 5.5

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.

Chapter 6

Text Operations

Learning Objectives

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

- Display and append to file contents using **cat** and **echo**.
- Edit and print file contents using **sed** and **awk**.
- Search for patterns using **grep**.
- Use multiple other utilities for manipulating the contents of text files, including sorting, cutting and pasting, cutting lines etc.

6.1 cat

cat

- Short for con**cat**enate
- Read and print files to standard output
- Useful for viewing small files

```
$ cat some_file
```

- Concatenate multiple files:

```
$ cat file1 file2
```

- Combine several files into one file:

```
$ cat file1 file2 > newfile
```

- Append file to end of existing file:

```
$ cat file >> existingfile
```

cat is one of the most frequently used utilities on **Linux** systems.

Using **cat** without options is a great way to view files on the fly at the command line.

However, the main purpose of **cat** is to **concatenate** files. Using it to concatenate one file with nothing else, however, is a waste of a process.

For example,

```
$ cat /etc/passwd | grep root
```

is more efficiently done without the piped processes as:

```
$ grep root /etc/passwd
```

The **tac** program (**cat** spelled backwards) simply prints the lines in reverse order (i.e., backwards) to standard output.

```
$ tac file
```

Using cat Interactively

- `cat > newfile`
  - Quick way to create new file
  - All input typed into new file
  - To finish input type `CTRL-D` or do:

```
$ cat << EOF > newfile

...
EOF
```

- To append to existing file:

```
$ cat << EOF >> existingfile

...
EOF
```

cat can be used to read standard input if no files are specified. You can use the > operator to combine several files into a new file, or >> to append files to an existing file.

In the previous example we used the construct **here document** to pass information to the **cat** command. The construct consists of a couple of parts:

1. First part is the **command** , in our case the **cat** command.
2. The second part is the `<< EOF`, this means the line after this one has a line of data to pass to the command in first section. The input will continue to be passed to the command until it reaches the termination code.
3. The termination code is simply a control field to stop the input and can be anything you like, in our example it is the second **EOF**.

For more information on **here document** see https://tldp.org/LDP/abs/html/here-docs.html

6.2 echo

echo

- Display a line of text by default:

```
$ echo some_string

some_string
```

- Used to input string to standard output
- Can enable special character sequences with the -e option

```
$ echo -e This is across '\n' two lines

This is across
 two lines
```

- Escape sequences:

```
\n: Newline
\r: Carriage return
\t: Horizontal tab
```

```
$ sudo sh -c "echo some string  > /etc/afile"
```

echo is particularly useful for showing the values of environmental variables.

```
$ echo SHELL= $SHELL
```

```
SHELL= /bin/bash
```

```
$ echo -e SHELL= $SHELL'\n'HOME= $HOME'\n'EDITOR= $EDITOR
```

```
SHELL= /bin/bash
HOME= /home/fzappa
EDITOR= /usr/bin/emacs
```

Very Important

If you try to do something like:

```
$ sudo echo some string  > /etc/afile
```

it will fail. You need to do:

```
$ sudo sh -c "echo some string  > /etc/afile"
```

This is a common error; the **sudo** must start a new shell in order to do this.

6.3 sed

sed

- **s**tream **ed**itor
- Powerful text processing tool
- Filters text on data streams
- Perform substitution on data streams
- Can be used two ways:
  - Specify editing commands on command line:
    ```
    $ sed -e 'command' file
    ```
  - Specify a scriptfile containing **sed** commands:
    ```
    $ sed -f scriptfile file
    ```
- Often used in piped commands

sed is one of the earliest **UNIX** utilities designed for command line processing of data files.
sed has four **spaces**:

- input stream
- pattern space
- hold buffer
- output stream

sed reads the input stream and produces the output stream.
Several editing commands can be put on the command line by using the `-e 'command'` option. For example:

```
$ sed -e 'command1' -e 'command2' -e 'command3' file
```

Using a `scriptfile` with the `-f` option allows for a large number of editing commands if needed.

Basic Operations

- Substitute for first occurrence of `pattern` on each line:

```
$ sed 's/pattern/new_pattern/' file
```

- Substitute for all occurrences of `pattern` on each line i.e., **g**lobally:

```
$ sed 's/pattern/new_pattern/g' file
```

- Substitute all occurrences on a range of lines:

```
$ sed '1,3s/pattern/new_pattern/g' file
```

- Save changes for string substitution in the same file (i.e., in place, wipe out original file):

```
$ sed -i 's/pattern/new_pattern/g' file
```

The `g` at the end of the substitute line stands for **global**, and causes **sed** to apply the substitution on all occurrences on each line, not just the first one.

Use the `-i` option with caution; changes are irreversible. It is always safer to run **sed without** the `-i` option, evaluate the changes to see if they are what is desired, and if so replace the file with the new one. For example:

```
$ sed 's/pattern/new_pattern/g' file > newfile
# verify that newfile is what you really want
$ mv newfile file
```

6.4 awk

awk

- Powerful utility and programming language
- Manipulates data files
- Retrieves and processes text
- Works on **records** (lines) and **fields** (columns within lines)
- Can be used two ways:
  - Specify a script directly on the command line:
    ```
    $ awk 'script' var=value file
    ```
  - Specify a script containing **awk** commands:
    ```
    $ awk -f scriptfile var=value file
    ```
- Often used in piped commands

awk is an interpreted programming language typically used as a data extraction and reporting tool.

It was created at **Bell Labs** in the 1970s and derived its name from the last names of it authors: Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan.

awk makes extensive use of the string datatype, associative arrays and regular expressions.

As with **sed**, a trivial **awk** script can go on the command line, but a more complex script probably belongs in a file that you then include with the `-f` option.

Basic Operations

- Print entire file:

```
$ awk '{ print $0 }' /etc/passwd
```

- Print first field:

```
$ awk -F: '{ print $1 }' /etc/passwd
```

- Print first and sixth field:

```
$ awk -F: '{ print $1 $6 }' /etc/passwd
```

awk reads the input files one line at a time. For each line for which **awk** matches the given pattern in the given order, it performs the corresponding action.

The `-F` option allows you to define a field separator character. For example, the `/etc/passwd` file uses ':' to separate the fields, so `-F:` is used in the `/etc/passwd` examples above.

```
$ awk -F: '{ print $1 " " $6 }' /etc/passwd
```

```
root /root
bin /bin
daemon /sbin
adm /var/adm
lp /var/spool/lpd
sync /sbin
shutdown /sbin
halt /sbin
mail /var/spool/mail
operator /root
   . . .
```

6.5 Miscellaneous Text Utilities

tr

- Utility for **tr**anslating characters
- Filters from standard input to standard output
- Handles special characters
- To translate lowercase characters to uppercase:

```
$ cat filename | tr 'a-z' 'A-Z'
```

The **tr** command, short for **tr**anslate, will translate values from one list to another list. It will read from standard input a value, say the value "e" which is the 5th element in the first list and output the 5th character from the second list which would be the letter "E". The most common application of this command is to translate uppercase and lowercase characters.

```
$ cat city
```

```
Anytown
Mytown
Yourtown
Youngstown
```

```
$ cat city | tr 'a-z' 'A-Z'
```

```
ANYTOWN
MYTOWN
YOURTOWN
YOUNGSTOWN
```

wc

- Stands for **w**ord **c**ount
- Prints out the number of:
  - Lines
  - Words
  - Characters

  for one or more files
- Example:

```
$ wc /home/vcorleone/.bashrc*
  210   608  7023 /home/vcorleone/.bashrc
  211   619  7051 /home/vcorleone/.bashrc~
   11    36   231 /home/vcorleone/.bashrc_ORIGINAL
  432  1263 14305 total
```

wc will display the number of lines (`-l`), words (`-w`), characters(`-m`) or bytes (`-c`). (Generally byte and character counts will be the same.)

With no arguments it prints out lines, words and bytes.

If more than one file is given, totals are also printed.

A word is defined as a non-zero-length sequences of characters, delimited by white space.

tee

- Pipe a command into **tee** and the output stream is **forked**, or **tee'ed**
  - One stream goes to usual standard output
  - One stream goes to a named file.
- Example:

```
$ ls -l | tee ls-l-ouptut
```

- **tee** is very useful when you both want to see what is happening and also save the results.

Commands and programs normally output to **standard out** which may be re-directed to a file if desired. There may be a requirement for output to be sent to both **standard out** and a file at the same time. This is what **tee** does.

```
$ ls -l /tmp/vmware-coop | tee ls-l-output
```

```
total 1552
-rw-rw-r-- 1 coop coop   2373 Mar 14 12:00 fuseMount-14141.log
srwxrwxr-x 1 coop coop      0 Mar 17 11:23 unity-helper-ipc-:0
-rw-r----- 1 coop coop   4315 Mar 16 14:43 vmware-10448.log
-rw-r----- 1 coop coop   4444 Mar 14 12:00 vmware-14141.log
-rw-r----- 1 coop coop   4315 Mar  9 09:05 vmware-23091.log
....
-rw-r----- 1 coop coop  43878 Mar  9 09:15 vmware-vmplayer-26520.log
-rw-r----- 1 coop coop  70653 Mar 12 12:52 vmware-vmplayer-26949.log
-rw-r----- 1 coop coop  24770 Mar 17 11:23 vmware-vmplayer-7613.log
```

```
$ cat ls-l-output
```

```
total 1552
-rw-rw-r-- 1 coop coop   2373 Mar 14 12:00 fuseMount-14141.log
srwxrwxr-x 1 coop coop      0 Mar 17 11:23 unity-helper-ipc-:0
-rw-r----- 1 coop coop   4315 Mar 16 14:43 vmware-10448.log
-rw-r----- 1 coop coop   4444 Mar 14 12:00 vmware-14141.log
-rw-r----- 1 coop coop   4315 Mar  9 09:05 vmware-23091.log
....
-rw-r----- 1 coop coop  43878 Mar  9 09:15 vmware-vmplayer-26520.log
-rw-r----- 1 coop coop  70653 Mar 12 12:52 vmware-vmplayer-26949.log
-rw-r----- 1 coop coop  24770 Mar 17 11:23 vmware-vmplayer-7613.log
```

strings

- Extract printable character **strings** from files
- Helps to locate human readable content in:
  - Binary files
  - Spreadsheets
- Such information may identify who created the file etc.
- Search for string in a spreadsheet:

```
$ strings book1.xls | grep string
```

Many commands are available to work with text based files, there is a requirement to look into a non-text file like a compiled binary program for some information. The **strings** command will separate out the text information from the non text information to allow our text tools to function.

```
$ strings /bin/cp | grep GPL
```

```
License GPLv3+: GNU GPL version 3 or later  <http://gnu.org/licenses/gpl.html>.
```

6.6 Sorting, Cutting, Pasting, Joining, Splitting

sort

- Sorts lines of text files
- Sort lines in a file:

```
$ sort filename
$ cat file1 file2 | sort
```

- Sort in reverse order:

```
$ sort -r filename
```

- Sort according to a certain field on each line:

```
$ sort -k 3 filename
```

By default, **sort** will sort by ASCII character order. There are many options!

```
$ cat /tmp/junk
```

```
Filesystem           Type      Size  Used Avail Use% Mounted on
/dev/sdb1            ext4       20G  7.8G   11G  43% /
devtmpfs             devtmpfs  7.8G     0  7.8G   0% /dev
tmpfs                tmpfs     7.8G  1.2M  7.8G   1% /dev/shm
/dev/mapper/VG-local ext4       24G   17G  5.5G  76% /usr/local
/dev/loop0           squashfs  3.3G  3.3G     0 100% /usr/src/KERNELS
```

```
$ sort /tmp/junk
```

```
devtmpfs             devtmpfs  7.8G     0  7.8G   0% /dev
/dev/sdb1            ext4       20G  7.8G   11G  43% /
/dev/mapper/VG-local ext4       24G   17G  5.5G  76% /usr/local
/dev/loop0           squashfs  3.3G  3.3G     0 100% /usr/src/KERNELS
tmpfs                tmpfs     7.8G  1.2M  7.8G   1% /dev/shm
Filesystem           Type      Size  Used Avail Use% Mounted on
```

```
$ sort -k 7 /tmp/junk
```

```
/dev/sdb1            ext4       20G  7.8G   11G  43% /
devtmpfs             devtmpfs  7.8G     0  7.8G   0% /dev
tmpfs                tmpfs     7.8G  1.2M  7.8G   1% /dev/shm
Filesystem           Type      Size  Used Avail Use% Mounted on
/dev/mapper/VG-local ext4       24G   17G  5.5G  76% /usr/local
/dev/loop0           squashfs  3.3G  3.3G     0 100% /usr/src/KERNELS
```

uniq

- Eliminates duplicate lines (records or entries)
- Useful for simplifying text display
- Eliminate entries from a file:

  ```
  $ cat file1 file2 | sort | uniq
  ```

- Count occurrences of duplicate entries:

  ```
  $ uniq -c filename
  ```

The duplicate entries need to be on consecutive lines, so running through **sort** may be necessary. You can also use the `-u` option to sort that runs the sorted output through **uniq** essentially doing **sort — uniq** if you prefer.

```
$ cat names
```

```
Connie
Fredo
Michael
Sonnv
Fredo
Sonny
```

```
$ sort names | uniq
```

```
Connie
Fredo
Michael
Sonny
```

```
$ sort -u names
```

```
Connie
Fredo
Michael
Sonny
```

paste

- Combines fields from files
- Combines lines from multiple sources
- To combine contents from two files:

  ```
  $ paste file1 file2
  ```

- To use a different delimiter:

  ```
  $ paste -d"," file1 file2
  ```

```
$ cat names
Fredo
Michael
Sonny
```

```
$ cat phone
555-251-3525
555-999-0011
555-212-2120
```

```
$ paste names phone
Fredo           555-251-3525
Michael         555-999-0011
Sonny           555-212-2120
```

```
$ paste -d':' names phone
Fredo:555-251-3525
Michael:555-999-0011
Sonny:555-212-2120
```

join

- Enhanced version of **paste**
- Joins lines of two files on common field
- Works only if files share common field
- To combine two files on common field:

```
$ join file1 file2
```

For example, our common field is the name:

```
$ cat phonebook
```

```
Connie        555-123-4567
Fredo        555-231-3325
Michael        555-999-0011
Sonny        555-212-2120
```

```
$ cat towns
```

```
Connie        Brooklyn
Fredo        Las Vegas
Michael        Manhattan
Sonny        Westchester
```

```
$ join phonebook towns
```

```
Connie 555-123-4567 Brooklyn
Fredo 555-231-3325 Las Vegas
Michael 555-999-0011 Manhattan
Sonny 555-212-2120 Westchester
```

cut

- Separates columns from file
- Default delimiter is tab
- Display third column delimited by space:

```
$ ls -l | cut -d" " -f3
```

```
$ cat phonebook
```

```
Connie        555-123-4567
Fredo        555-231-3325
Michael        555-999-0011
Sonny        555-212-2120
```

```
$ cut -f 1 phonebook
```

```
Connie
Fredo
Michael
Sonny
```

```
$ cut -f 2 phonebook
```

```
555-123-4567
555-231-3325
555-999-0011
555-212-2120
```

split

- Splits a large file into equal-sized segments
- The original remains unchanged
  - Each file created begins with a `PREFIX` and has appended characters
    - `PREFIX` is `x` by default
- Default splits into individual 1000 line files
- To split file into segments:

```
$ split infile
```

- To split file into segments using a `PREFIX`:

```
$ split infile PREFIX
```

```
$ wc JUNK
```

```
  2856 JUNK
```

```
$ split JUNK
$ wc JUNK x*
```

```
  2856 JUNK
  1000 xaa
  1000 xab
   856 xac
  5712 total
```

```
$ split JUNK SPLIT
$ wc JUNK SPLIT*
```

```
  2856 JUNK
  1000 SPLITaa
  1000 SPLITab
   856 SPLITac
  5712 total
```

6.7 Regular Expressions and grep

Regular Expressions

- Sequence of characters forming a search pattern
- Used for string matching
- Often abbreviated as **regex** or **regexp**
- Contain Normal characters and Metacharacters
- Supported in text-processing utilities:
  - **vi**
  - **sed**
  - **awk**
  - **grep**
  - **find**
- Regular expressions are different from shell filename matching metacharacters!

Regular Expressions are used by many text editors, utilities and programming languages to search and manipulate text based on patterns.

Some of the popular languages that have built-in regular expression capabilities include **Perl**, **JavaScript** and **Ruby**. Others languages have a standard library for regular expression functionality, including **Java**, **Python** and **C++**.

Regular expressions can be very complicated, and resemble a comic book curse, and thus we only intend to touch the surface here; there are entire books written about them. As an example:

```
^[ \t\]+|[ \t]+$
```

will match the excess whitespace at the beginning or an end of a line!

Search Patterns

Some of the more common regex search pattern elements are:

- Match any single character
  `. (dot)`
- Matches a or z
  `a|z`
- Matches end of string
  `$`
- Matches preceding item zero or more times
  `*`

Search patterns are at the heart of regular expressions, regardless if we are doing a search and replace or just a search regular expression search patterns are required. Consider the common regex search patterns listed here to the rules how search patterns work.

Consider the sentence:

> The quick brown fox jumped over the lazy dog

- `a..`
  matches `azy`
- `b.|j.`
  matches both `br` and `ju`
- `..$`
  matches `og`
- `l.*`
  matches `lazy dog`
- `l.*y`
  matches `lazy`
- `the.*`
  matches the whole sentence!

Using Regular Expressions

Some additional examples of using **regex** with commands like sed and grep.

- With **sed**:
  `$ sed 's/...$//'`
- With **grep**:
  `$ find / | grep .pg$`
- In **vi**:
  `/lastword$`

Some examples of regular expressions:

```
$ sed 's/...$//'
```

When **sed** runs with the displayed arguments **sed** processes the regular expression as:

- s=search
- dot-dot-dot look for three of ”any character”
- $ indicates the characters must be at the end of the line
- replace the three characters with nothing

The next command says:
Starting at the root directory, find all the files and directories, then display only the files and directories that end in ”.pg”.

```
$ find / | grep .pg$
```

Displays all filenames that end with `pg`

```
/lastword$
```

Search for `lastword` only at the end of a line.

grep

- Primary text searching tool
- Searches for patterns in text files
- Can be used with regular expressions

The command **grep** is one of the most commonly used text commands in Linux. grep searches standard in or a named file for the entered search string. The **grep** command has several command line options that make sorting through text files easier. Here are some examples of common **grep** options:

- -V will display the version of grep

```
$ grep -V
```

```
grep (GNU grep) 3.7
```

- -v inverts the search, show all that do not match

```
$ grep -v /sbin/nologin  /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
gnome-initial-setup:x:125:65534::/run/gnome-initial-setup/:/bin/false
hplip:x:126:7:HPLIP system user,,,:/run/hplip:/bin/false
gdm:x:127:133:Gnome Display Manager:/var/lib/gdm3:/bin/false
student:x:1000:1000:LF student,,,:/home/student:/bin/bash
```

- Sometimes we need to see the lines before and after our search.

```
$ grep hplip -A2 -B1 /etc/passwd
```

```
gnome-initial-setup:x:125:65534::/run/gnome-initial-setup/:/bin/false
hplip:x:126:7:HPLIP system user,,,:/run/hplip:/bin/false
gdm:x:127:133:Gnome Display Manager:/var/lib/gdm3:/bin/false
student:x:1000:1000:LF student,,,:/home/student:/bin/bash
```

Searching Text with grep

- Search pattern in a file and print matching lines:
  `$ grep pattern filename`
- Print all lines that do **not** match pattern:
  `$ grep -v pattern filename`
- Print the lines that contain numbers:
  `$ grep "[0-9]" filename`
- Print context of lines for matching pattern:
  `$ grep -C 3 pattern filename`
- Search for multiple patterns:
  `$ grep -e pattern1 -e pattern2 filename | grep -v -e dog -e cat`

```
$ grep nologin /etc/passwd
```

```
bin:x:1:1:bin:/bin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
....
```

```
$ grep -v nologin$ /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
coop:x:500:500:Jerry Cooperstein:/home/coop:/bin/bash
```

```
$ grep -C 2 ftp /etc/passwd
```

```
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
```

6.8 Labs

Video Demonstration Resources

using_cat.mp4
using_sed.mp4

Exercise 6.1: tee-ing a File

The **tee** utility is very useful for saving a copy of your output while you are watching it generated.

Execute a command such as doing a directory listing of the `/etc` directory:

```
$ ls -l /etc
```

while both saving the output in a file and displaying it at your terminal.

Solution 6.1

```
$ ls -l /etc | tee /tmp/ls-output
$ less /tmp/ls-output
```

Exercise 6.2: Parsing files with awk (and sort and uniq)

Generate a column containing a unique list of all the shells used for users in `/etc/passwd`. You may need to consult the manual page for `/etc/passwd` as in:

```
$ man 5 passwd
```

Which field in `/etc/passwd` holds the account's default shell (user command interpreter)? How do you make a list of **unique** entries (with no repeats).

Solution 6.2

The field in `/etc/passwd` that holds the shell is #7. To display the field holding the shell in `/etc/passwd` using **awk** and produce a unique list:

```
$ awk -F: '{print $7}' /etc/passwd | sort -u
```

or

```
$ awk -F: '{print $7}' /etc/passwd | sort | uniq
```

For example:

```
$ awk -F: '{print $7}' /etc/passwd | sort -u
```

```
/bin/bash
/bin/sync
/sbin/halt
/sbin/nologin
/sbin/shutdown
```

Exercise 6.3: Using wc (Word Count)

Find out how many lines, words and characters there are in all files in `/var/log` that have the `.log` extension.

Solution 6.3

```
$ sudo wc /var/log/*.log
```

```
  376    2675   21570 /var/log/boot.log
  805    4988  126977 /var/log/dnf.librepo.log
 8839   71774  760826 /var/log/dnf.log
 7164   32605  516106 /var/log/dnf.rpm.log
    1       6      60 /var/log/hawkey.log
   30     265    3195 /var/log/kdump.log
    3      15     102 /var/log/vbox-setup.log
  544    5836   42262 /var/lo
```

Note you have to do this with **sudo** to get every file counted; try without it!

Exercise 6.4: Searching and substituting with sed

Search for all instances of the user command interpreter (shell) equal to `/sbin/nologin` in `/etc/passwd` and replace them with `/bin/bash` using **sed**. **Do not** overwrite `/etc/passwd`!

Solution 6.4

To get output on standard out (terminal screen):

```
$ sed  s/'\/sbin\/nologin'/'\/bin\/bash'/g /etc/passwd
```

or to direct to a file:

```
$ sed  s/'\/sbin\/nologin'/'\/bin\/bash'/g /etc/passwd > passwd_new
```

Note this is kind of painful and obscure because we are trying to use the forward slash (/) as both a string and a delimiter between fields. One can do instead:

```
$ sed s:'/sbin/nologin':'/bin/bash':g /etc/passwd
```

where we have used the colon (:) as the delimiter instead. (You are free to choose your delimiting character!) In fact when doing this we don't even need the single quotes:

```
$ sed s:/sbin/nologin:/bin/bash:g /etc/passwd
```

works just fine

Exercise 6.5: Simple Searching with grep

Search for your username in file `/etc/passwd`.

Solution 6.5

```
$ grep your-username /etc/passwd
```

Exercise 6.6: Using grep in More Detail

In the following we give some examples of things you can do with the **grep** command; your task is to experiment with these examples and extend them.

1. Find all entries in `/etc/services` that include the string **ftp**:
2. Restrict to those that use the **tcp** protocol:
3. Now restrict to those that do not use the **tcp** protocol, while printing out the line number:
4. get all strings that start with **ts** or end with **st**:

Solution 6.6

1. `$ grep ftp /etc/services`

2. `$ grep ftp /etc/services | grep tcp`

3. `$ grep -n ftp /etc/services | grep -v tcp`

4. `$ grep ^ts /etc/services`
   `$ grep st$ /etc/services`

Chapter 7

File Operations

Learning Objectives

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

- Describe the basic layout of **Linux** filesystems and the purpose of the Filesystem Hierarchy Standard (**FHS**).
- Understand the roles of partitions and mount points.
- Explain the purpose and use of **NFS**.
- Use **rsync** to synchronize and backup between machines.
- Know how to view files with basic utilities such as **cat**, **less**, **tail** and **head**.
- Know how to remove directories and files.
- Use **dd** for low level data transfer and transformations.
- Compare files with **diff** and patch files with **patch**.
- Know how to investigate the type of a file.
- Compress data with **xz**, **bzip2** and **gzip** and how to view them.
- Use **tar** to create and extract archives.

7.1 Filesystems

Linux Filesystem: Basics

Table 7.1: **Standard Single Hierarchy Layout**

/bin,/usr/bin	Programs (binaries and scripts)
/sbin,/usr/sbin	Programs for system maintenance
/lib,/usr/lib	Shared libraries
/usr/share	Static device-independent data
/var	Shared variable data
/etc	System-wide configuration
/home	User home directories
/media, /run/media	Removable storage
/boot	Files needed to start the system
/proc,/sys	System information from the kernel
/opt	Third party packages kept in isolation

Linux systems store their important files according to a standard layout called the **Filesystem Hierarchy Standard** (**FHS**). This standard ensures that users can move between distributions without having to re-learn how the system is organized.

Linux uses the "/" character to separate paths (unlike **Windows**, which uses "\") and does not use drive letters, such as `C:`. New drives are mounted as directories (at **mount points**) in the single filesystem.

Removable media are usually mounted under `/media` (so, for example, a **CD-ROM** named `FEDORA` might end up being found at `/media/FEDORA`, and a file `README.txt` on that disc would be at `/media/FEDORA/README.txt`).

/run

On modern distributions this behavior is no longer true; the mount point will be under `/run` and may also include the name of the user that mounted the media.

All **Linux** filesystem paths are case-sensitive; thus `/boot` and `/BOOT` are different directories. Many distributions distinguish between core utilities needed for proper system operation and other programs, and place the latter in directories under `/usr` (think **user**).

Filesystem Hierarchy Standard

- Defines main directories and their contents in **Linux** systems
  https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf
- Makes it possible to prepare software packages to be installed and run on all **FHS**-compliant distributions in a unified manner
- Maintained by the **Linux Foundation**
- Gradually accommodates changes in practices
- Most **Linux** distributions follow the **FHS**

The **Filesystem Hierarchy Standard** grew out of historical standards from previous versions of **UNIX** such as the Berkeley Software Distribution (**BSD**) and others.

The **FHS** provides **Linux** developers and system administrators a standard directory structure for the file system which provides consistency between systems and distributions.

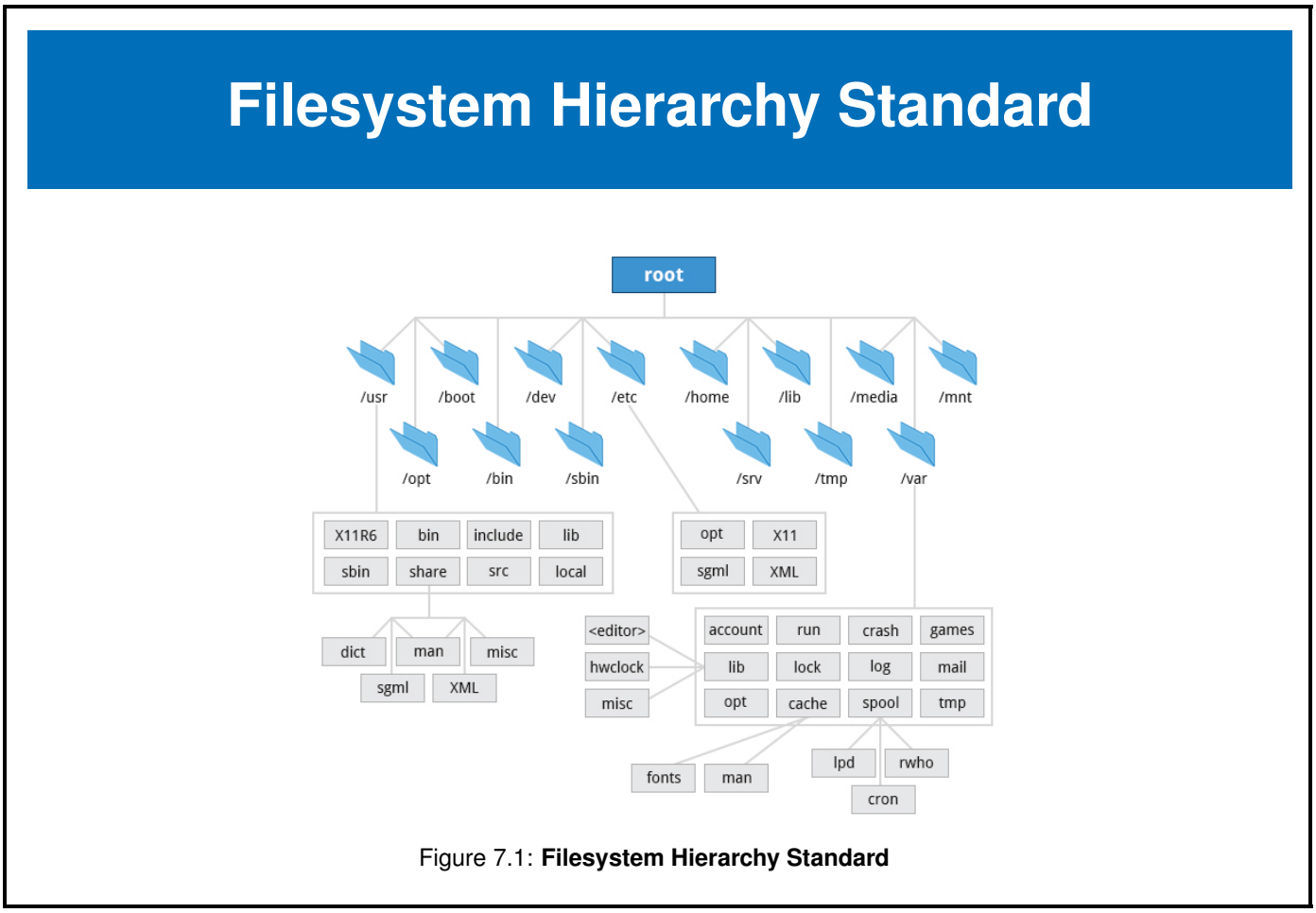


Figure 7.1: **Filesystem Hierarchy Standard**

Everything is a File

- On a **UNIX**-like systems (including **Linux**):

  Everything is a File

  including:
  - programs
  - services
  - texts
  - images
  - devices
  - processes
- Tree-like structure filesystem on hard disk, starting at the **trunk**, or /
- See

  ```
  $ man hier
  ```

  for inline documentation

A very core concept in **Linux** (and all **UNIX**-like operating systems) is that everything is treated as if it were a file. This makes I/O a simpler concept within **Linux**: you open a file and perform normal operations like reads and writes to that file. You will appreciate this as your use of **Linux** expands.

Tree-like structure filesystems are common on many systems. The tree starts at what is often called the **root** directory (or trunk, or, simply, /), but do not confuse this with what is called the **root** user; this is just the start of the hierarchical filesystem. Note that elements in the path are separated by forward slashes (/) as in `/usr/bin/awk`.

7.2 Partitions and Mount Points

Partitions

- Used to separate different forms and functions of data and files into different containers
- Two major kinds of partitions
  - Data: normal **Linux** system data, including root partition
  - Swap: expansion of the computer's physical memory
- Most systems contain a root partition, data partitions and swap partitions
- **Linux** supports many filesystem types:
  - **ext3, ext4, btrfs, xfs**
  - **vfat, ntfs**

Partitions help to separate different forms and functions of data.

For example, often programs required for the proper running of the system are on a separate partition (root (`/`)) from the files owned by regular users of that system (`/home`).

Data which is variable (and volatile) is often isolated in a partition dedicated to holding `/var`.

Temporary files created and destroyed during the normal operation of **Linux** are often located on a separate partition (e.g., `/tmp`).

Isolating different types of material in separate partitions can be helpful: using all of the space of one particular partition may not fatally affect the normal operation of the system.

Linux supports various native **Linux** filesystem types along with compatible filesystems found on other operating systems. Many older, legacy filesystems, (such as **FAT**) are also supported.

Mount Points

- Partitions are **mounted** at **mount points** (usually empty subdirectories) in the one big filesystem
- Show with **mount** or **df**:

```
$ mount

....
/dev/sda2 on /var type ext4 (rw,relatime,data=ordered)
....

$ df -Th
```

```
Filesystem     Type    Size  Used Avail Use% Mounted on
....
/dev/sda2      ext4     20G  7.7G   11G  43% /var
```

- Persistent mounts configuration in `/etc/fstab`

```
....  /dev/sda2 /var ext4 defaults 1 2
```

The name of the persistent mounts file, /etc/fstab, is short for **F**ile**s**ystem **T**able. There is one line per filesystem in this file, for each mounted filesystem, which contains information about partition (or **label** or **UUID**), type of filesystem, and options for how it is mounted and checked.

/etc/fstab

```
UUID=7f7efgb8-80c6-40b7-e623-dfcb99927c37 /           ext4  defaults 1 1
....
LABEL=local                               /usr/local  ext4  defaults  1 2
/dev/sdb3                                 /usr/src    ext4  defaults  1 2
....
/usr/src/KERNELS.sqfs                     /usr/src/KERNELS squashfs loop 0 0
....
LABEL=SWAP                                swap        swap  defaults  0 0
....
```

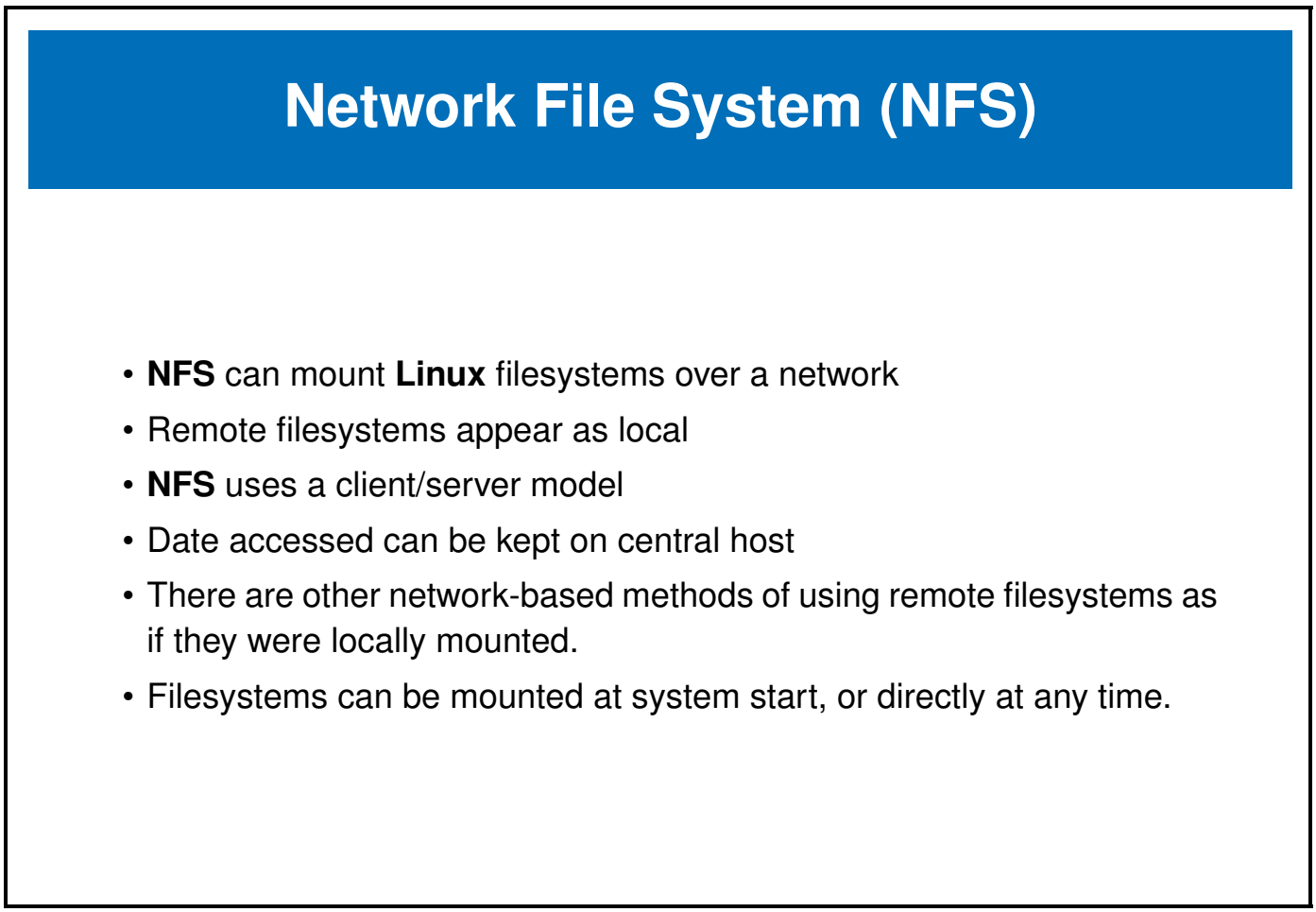
See **man fstab** for the format and details of this file. Typing **mount** without any arguments will display a list of all of the filesystems mounted on your **Linux** system:

```
$ mount
```

```
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
...
```

df with the -Th options will also display information on mounted filesystems along with some usage statistics.

7.3 Network File System (NFS)



The **N**etwork **F**ile **S**ystem (**NFS**) is one of several methods for sharing data across physical systems. Implementations have existed since early days of **UNIX**.

For example, on many multi-user, multi-node organizational installations, the user's home directory may reside on a central server. Thus, users gain access to the same files and configurations across multiple systems. They can also login to different computers at different times, but still have access to the same files in their login directory.

On the internals level, **NFS** has to be rather sophisticated to avoid race conditions, such as when multiple users modify the same files at once, as well as to handle interruptions in network services, such as when a link goes down. Security is also a significant issue any time there is remote logging in and much traffic.

NFS Clients, Servers and Mounting

- **Clients** can mount **NFS** filesystems as in:

  ```
  $ sudo mount server84:/home /home
  ```

  for a one time mount, or for a persistent mount, put

  ```
  server84:/home /home nfs default 0 0
  ```

  in `/etc/fstab`:

- **Servers** must configure `/etc/exports` with lines like:

  ```
  /VIRTUAL        192.168.1.0/24(rw,no_root_squash)
  /usr/local/    *(ro,insecure,all_squash)
  ```

  and start the **nfs** service as in:

  ```
  $ sudo systemctl start nfs-server
  ```

- If `/etc/exports` is revised one should run

  ```
  $ sudo exportfs -av
  ```

  to read in the changed configuration information

On the client machine, /etc/fstab should include the remotely mounted filesystem, the local mount point and various options for a persistent mount across reboots. An entry in our client **Linux** /etc/fstab file might look like:

/etc/fstab

```
servername:/projects /mnt/nfs/projects nfs default 0 0
```

One can also mount the remote filesystem without a reboot or as a one-time mount by using **mount**.

On the server machine, **NFS** daemons (and other system servers) are typically started with **systemctl** as in:

```
$ systemctl start nfs-server
```

(Depending on **Linux** distribution and version the service may be called either **nfs-server**, **nfs** or **nfsd**).

/etc/exports contains the directories and permissions that a host is willing to share with other systems over NFS. An entry may look like:

/etc/exports

```
/projects     *.example.com(rw)
```

This entry will allow the directory /projects to be mounted using NFS with read and write permissions on other hosts only in the example.com domain. After modifying the /etc/exports file, use exportfs -av to notify **Linux** of the directories you are allowing to be remotely mounted using NFS (i.e., exported).

7.4 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.

7.5 Working with Files

Viewing Files

- View an entire file:

```
$ cat filename
$ tac filename
```

(**tac** views in reverse)

- View an entire file with paging forward and back:

```
$ less filename
$ more filename
```

(**more** is older and less powerful than **less**)

- View the last ten (or `-n`) lines of a file:

```
$ tail [-f] [-n] filename
```

- View the first ten (or `-n`) lines of a file:

```
$ head [-n ] filename
```

cat is most suitable to view shorter files as it does not provide any paging facility. The name is short for **concatenate** and is often used to combine files as in

```
$ cat file1 file2 file3 > file3
```

less is an excellent choice to view large files because it does not read the entire input file before starting. It also provides paging, searching, and extensive navigating capabilities, can display line numbers, etc. (Use / to search for a pattern in the forward direction and ? for a pattern in the backward direction.) **more** is an older less capable paging program.

tail by default prints the last ten lines of a file; `-n` option allows one to specify the number of lines to view. Using the `-f` option lets one see all new lines as they appear. **head** is the opposite of **tail**, as it prints the first ten lines of a file, by default.

```
$ cat /etc/shells
```

```
/bin/sh
/bin/bash
....
/bin/zsh
/bin/csh
/bin/tcsh
/usr/bin/csh
/usr/bin/tcsh
```

```
$ head -2 /etc/shells
```

```
/bin/sh
/bin/bash
```

```
$ tail -2 /etc/shells
```

```
/usr/bin/csh
/usr/bin/tcsh
```

Touching a File

- Create or timestamp a file:

  ```
  $ touch filename
  ```

  - Triggers an error if no write permission
  - Creates an empty file if the file does not already exist
  - Otherwise, updates the access and modification timestamps
- Can test directory permissions with `-c`
- Can set a specific timestamp with `-t`

The **touch** command is used to set or update the access, change, and modify times of files. By default **touch** command would change the file timestamp to the current time. To set the file-stamp to a specific time:

```
$ touch -t 03201600 filename
```

This sets a file's timestamp to 4 p.m., March 20th. When `filename` does not exist, **touch** will create an empty file. This is normally done to create empty files with filenames as a placeholder.

```
$ ls -l
    total 0
```

```
$ touch newfile ; ls -l
    total 0
    -rw-rw-r--. 1 bjmoose bjmoose 0 Mar 16 18:30 newfile
```

```
$ touch -t 03141600 myfile ; ls -l
    total 0
    -rw-rw-r--. 1 bjmoose bjmoose 0 Mar 14 16:00 myfile
    -rw-rw-r--. 1 bjmoose bjmoose 0 Mar 16 18:30 newfile
```

```
$ touch -c /bin
    touch: setting times of `/bin': Permission denied
```

Removing a File

- Rename or move (mv) a file:

```
$ mv oldfilename newfilename
```

- Remove a file:

```
$ rm filename
```

- Forcefully remove a file:

```
$ rm -f filename
```

- Interactively remove files:

```
$ rm -i filename
```

rm works with pattern matching characters which makes removing files on the command line much faster. If you aren't certain of removing files that match your pattern, it is always good to run **rm** interactively to prompt before every removal.

```
$ ls
```

```
file-1  file-2  myfile  newfile  non-writable-file
```

```
$ rm newfile ; ls
```

```
file-1  file-2  non-writable-file  myfile
```

```
$ rm non-writable-file
```

```
rm: remove write-protected regular empty file `non-writable-file'? n
```

```
$ rm -f non-writable-file  ; ls
```

```
file-1  file-2  myfile
```

```
$ rm -i file*
```

```
rm: remove regular empty file `file-1'? y
rm: remove regular empty file `file-2'? n
```

```
$ ls
```

```
file-2  myfile
```

Removing a Directory

- Rename a directory:

```
$ mv old-directory new-directory
```

- Remove an empty directory:

```
$ rmdir directory
```

- Forcefully remove a directory recursively:
  - Use with care!

```
$ rm -rf directory
```

Here are some examples moving directories:
rmdir will not work on directories unless they are empty.

```
$ rm -rf
```

is a fast and easy way to remove a whole file system tree recursively, but it should be used with care, especially as root.

```
$ ls
```

```
directory-a  directory-b  directory-c  file-1  file-2
```

```
$ mv directory-c temp ; $ ls
```

```
directory-a  directory-b  file-1  file-2  temp
```

```
$ ls directory-a directory-b
```

```
directory-a:
directory-b:
file-42
```

```
$ rmdir directory-a directory-b
```

```
rmdir: failed to remove `directory-b': Directory not empty
```

```
$ rm -rf directory-b ; ls
```

```
file-1  file-2  temp
```

dd

- Makes copy of input file (`if`)
- Sends results to output file (`of`)
- Useful for converting between raw physical devices

```
$ dd if=/dev/sda of=/dev/sdb
```

- If `-if` not given, input is taken from standard in
- If `-of` not given, output goes to standard out
- **dd** is often used in piped command sequences

dd is very useful for making raw copies of raw partitions and data. For example, a way to back up your **MBR** (the first sector (512 bytes) on the disk that contains the partition table, which when it gets corrupted can cause many headaches):

```
$ dd if=/dev/sda of=sda.mbr bs=512 count=1
```

To make an exact copy of the first disk on the system and overwrite the second with it:

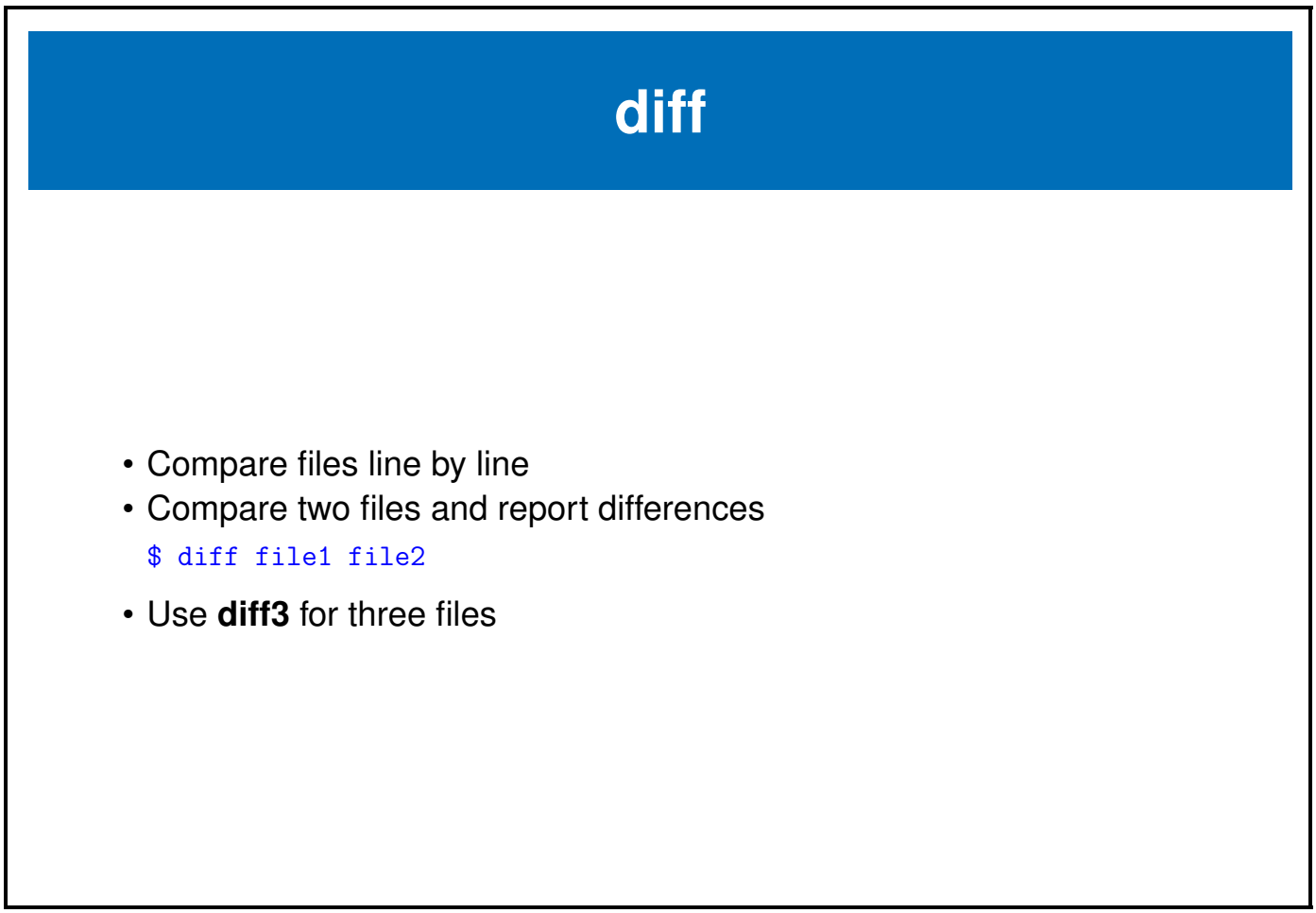
```
$ dd if=/dev/sda of=/dev/sdb
```

This works best if both disks are of the same size.

dd can do many other things and parameters such as `-bs` which controls the block size, the number of bytes copied at once, can have a large effect on the speed and memory usage.

dd can also do other things such as **byte-swapping** and other data conversions.

7.6 Comparing Files



diff has many useful options (see **man diff**). Some options that you may find useful include:

- `-c`
  produce a listing of differences that includes lines around those that differ
- `-r`
  recursively work down directory trees
- `-i`
  ignore the case of letters so that upper and lower case letters will match
- `-w`
  ignore spaces and tabs (white space) so that you are only looking at the text

diff3 is used to compare three files, one of which is the basis for the other two files. For example, if you and a co-worker both modify the same file to come up with your own versions of a new file, **diff3** would be useful to show you the differences based on the common file you both started with. The syntax is:

```
$ diff3 MY-FILE COMMON-FILE YOUR-FILE
```

patch

- Apply patches in a diff file to an original

```
$ patch originalfile patchfile
$ patch -p1 < patchfile
```

- Useful for patches between two directory trees
- Patch file is difference produced by **diff**

patch is a very useful tool in **Linux**. Many modifications to source code and configuration files are distributed with **patch** files as they contain the deltas or changes to go from an old version of a file to the new version of a file.

Often this is more concise and efficient than distributing the entire file. For example, if only one line needs to change in a 1,000 line file, you can see where a **patch** file will be much smaller than distributing the updated 1,000 line file.

7.7 File Types

file Utility

- In **Linux** file extensions rarely play a defining role;
- The file type is determined by examining contents rather than extension

```
$ file *
acpitool:      ELF 64-bit LSB executable, x86-64, version 1
               (SYSV), dynamically linked (uses shared libs),
               for GNU/Linux 2.6.9, not stripped
addkernel.sh:  Bourne-Again shell script, ASCII text executable
archmbox:      Perl script, ASCII text executable
atob:          C shell script, ASCII text executable
back_w7.sh:    POSIX shell script, ASCII text executable
blank.jpg:     JPEG image data, JFIF standard 1.01
do_ent:        ASCII text
rotatepics.sh: symbolic link to `rotate_7.sh'
```

File extensions in **Linux** are not how a file type is determined, the file type is determined by examining the contents of the file.

For example, a file named `file.txt` might in fact be a binary program. It is not a good practice to do something like this, but in **Linux**, a file name is more meaningful to the user of the system than the system itself.

file reads the arguments given to it and determines from the contents and certain characteristics of the files themselves whether the files are ASCII text, images, shared libraries, binary executable programs, scripts of various kinds, source code, etc.

7.8 Compressing Data

Compression Utilities

- There are many reasons to compress files and archives:
  - Free up storage space
  - When combined with archive programs (such as **tar**) convenient way to package data
  - Faster network transmission of data
- Major compression programs:
  - **gzip**
  - **xz**
  - **bzip2** (no longer maintained, better to migrate away from)
  - **zip**
- Utilities differ in compression speed, decompression speed, and compression ratio
- **gzip** is fastest while **xz** compresses best

Compression utilities need to be fast and they need to compress well. For data which is compressed rarely (say to be archived) but decompressed more often, a slow compression speed will be worthwhile if it results in a smaller compressed file.

gzip is the most prevalent compression utility used in **Linux**. It provides excellent and very fast compression.

bzip2 has also been very widely used. While it is somewhat slower than **gzip** for compression, it tends to lead to significantly smaller files.

xz is newer and it is rapidly taking over in some spheres. It offers by far the best compression, but compression can be slow, although decompression is fast.

bzip2 is fading from use

bzip2 has become deprecated due to lack of maintenance and the superior compression ratios of **xz** which is actively maintained.

zip is rarely used in the **Linux** world, but is offered for compatibility with other operating systems. It is slow and inefficient.

The Windows default decompressor is often defective

The default **Windows** decompression programs are often defective and do not follow standards, so if you need to decompress files on **Windows** machines and encounter errors, try one of the numerous external applications that are offered.

gzip

- **GNU** zip
- Compress files
- **gzip** replaces each file with a compressed file
  - Uses `.gz` extension

```
$ gzip file
```

- gzip recursively:

```
$ gzip -r directory
```

- Use **gunzip** to decompress files

Some usage examples:

```
$ gzip *
```

compress all of the files in the current directory; each file is compressed and renamed its name with a `.gz` extension.

```
$ gzip -r projectX
```

compress all of the files in the `projectX` directory along with all of the files in all of the directories under `projectX`.

```
$ gunzip foo
```

de-compress foo found in the file `foo.gz`. Under the hood, **gunzip** is the same as **gzip -d**.

xz

- Similar in usage to **gzip**, but uses different compression algorithm
- **xz** replaces each file with a compressed file
  - Uses `.xz` extension

    ```
    $ xz file
    ```
  - Unlike **gzip** and **bzip2** there is no `-r` (recursive) option. One could do something like:

    ```
    $ find . -type f -exec xz {} \;
    ```
- Use **unxz** (or **xz -d**) to decompress

The command syntax for **xz** is very similar to that of **gzip**.

```
$ xz *
```

compress all of the files in the current directory and replaces each file with a file named `<original-file-name>.xz`.

```
$ unxz *.xz
```

decompress all of the files with an extension of `.xz` in the current directory. Under the hood, **unxz** is calling **xz -d**.

bzip2

- Stands for **Block** zip
- Similar in usage to **gzip**, but uses different compression algorithm
- **bzip2** replaces each file with a compressed file
  - Uses `.bz2` extension

```
$ bzip2 file
$ bzip2 -r directory
```

- Use **bunzip2** to decompress

Please Note

bzip2 is now deprecated due to lack of maintenance

The command syntax for **bzip2** is very similar to that of **gzip**. Examples of common usage are also similar to **gzip**:

```
$ bzip2 *
```

compress all of the files in the current directory and replaces each file with a file named `<original-file-name>.bz2`.

```
$ bunzip2 *.bz2
```

decompress all of the files with an extension of `.bz2` in the current directory. Under the hood, **bunzip2** is calling **bzip2 -d**.

zip

- Package and compress files

```
$ zip file.zip file
```

- zip a directory recursively:

```
$ zip -r directory.zip directory
```

- Use **unzip** to decompress files

Please Note

zip is rarely used on **Linux** systems to create files.

However, **unzip** is used to deal with files from other operating systems.

Some usage examples:

```
$ zip archive *
```

compress all of the files in the current directory and put them into the file `archive.zip`.

```
$ zip -r backup.zip ~
```

archive your login directory and all files and directories under it in the file `backup.zip`.

```
$ unzip archive.zip
```

extract all of the files in the file `archive.zip` and put them in the current directory.

Working with Compressed Text Files

- Many text commands can be prefixed with **z**
- Reads and works with **gzip**-compressed files
  - **zcat**
  - **zless**
  - **zmore**
  - **zgrep**
- View a zipped file and search for string:

```
$ zcat file.gz | grep string
$ zgrep string file.gz
```

```
c8:/tmp>gzip -c $HOME/.bashrc > bashrc.gz
c8:/tmp>zgrep PATH bashrc.gz
```

```
if [ "$OLD_PATH" == "" ] ; then
    export OLD_PATH=$PATH
        if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
              PATH=$PATH:$1
              PATH=$1:$PATH
export CDPATH=./:$HOME
export CDPATH=$CDPATH:/usr/src/linux:/usr/local
export CDPATH=$CDPATH:/$HOME/LF
export CDPATH=$CDPATH:/
#export CDPATH=$CDPATH:/tmp
PATH="/home/coop/perl5/bin${PATH:+:${PATH}}"; export PATH;
```

```
c8:/tmp>zless bashrc.gz | head
```

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

set -o allexport
```

tar

- Name originally stood for **T**ape **AR**chive
- Place all files in `mydir` in an archive

```
$ tar -cvf mydir.tar mydir
```

- Expand the archive, perhaps somewhere else

```
$ tar xvf mydir.tar
```

- Compress at the same time

```
$ tar zcvf mydir.tar.gz  mydir
$ tar jcvf mydir.tar.bz2 mydir
$ tar Jcvf mydir.tar.xz  mydir
```

- Equivalent to doing in two steps, or in a pipe as in:

```
$ tar cvf mydir.tar mydir ; gzip -v mydir
$ tar cvf - mydir | gzip -v > mydir.tar.gz
```

- In the above commands, a dash (-) may precede the options

Historically, **tar** stood for **t**ape **ar**chive and was used to archive files to magnetic tape. **tar** allows you to create an archive, or **tarball**, or to extract files from it.

Some typical uses might be:

```
$ tar cvf project-X.tar project-X
```

copy all the files and subdirectories in the `project-X` directory into the file named `project-X.tar`.

```
$ tar xvf project-X.tar
```

extract all of the files and subdirectories in the file `project-X.tar` and put them in the current directory.

tar is most often used with one of the compression utilities, unless the files being archived are already compressed; i.e., they consist of compressed images such as picture files, or other tarballs.

One can simply decompress and extract such a compressed tarball as in:

```
$ tar xvf mydir.tar.{gz,bz2,xz}
```

Note you do not have to specify which kind of compression was used (with `z`, `j`, or `J`).

7.9 Labs

Exercise 7.1: Exploring Mounted Filesystems

Issue the command:

```
$ cat /etc/fstab
```

Now type:

```
$ mount
```

Compare the results generated by the two commands.

What are the differences?

Find another way to see a list of the mounted filesystems, by examining the `/proc` pseudo-filesystem.

Solution 7.1

Typically, **mount** will show more filesystems mounted than are shown in `/etc/fstab`, which only lists those which are explicitly requested.

The system, however, will mount additional **special** filesystems required for normal operation, which are not enumerated in `/etc/fstab`.

Another way to show mounted filesystems is to type:

```
$ cat /proc/mounts
```

which is essentially how the **mount** utility gets its information.

Exercise 7.2: Setting up an NFS Share and Mounting It

Ensure NFS Server Software is Installed First

You may have to make sure the proper **nfs** server software is installed on your machine. This will depend on your distribution.

For example, **Ubuntu** machines may need to install **nfs-common,libnfs13 and nfs-kernel-server**.

1. Edit `/etc/exports` to open up access to `/usr/local` through **NFS**. There are a variety of options you can use, but a simple but insecure option is to just add the line:

   ```
   /usr/local/ *(ro,insecure,all_squash)
   ```

   to the file.

2. Start the **nfs** service, with **systemctl**.

3. Type

   ```
   $ exportfs
   ```

   to make sure **nfs** is properly configured.

4. Mount `/usr/local` on `/mnt` and verify by looking at the contents.

5. If possible, work with another VM,system or student to attempt a mount on the other's machine instead of on the local machine.

Solution 7.2

1. Add the line

   ```
   /usr/local/ *(ro,insecure,all_squash)
   ```

   to `/etc/exports`. (You will have to use **sudo** or be root to modify this system file.)

2. ```
   $ sudo systemctl start nfs
   ```

   or

   ```
   $ sudo systemctl start nfs-server
   ```

3. ```
   $ sudo exportfs
   /usr/local            <world>
   ```

4. ```
   $ sudo mount localhost:/usr/local /mnt
   $ ls -l /mnt
   ```

   ```
   total 40
   drwxr-xr-x 2 root root 4096 Apr  3 09:19 bin
   drwxr-xr-x 2 root root 4096 Nov 18 06:33 etc
   drwxr-xr-x 2 root root 4096 Nov 18 06:33 games
   drwxr-xr-x 2 root root 4096 Nov 18 06:33 include
   drwxr-xr-x 2 root root 4096 Nov 18 06:33 lib
   drwxr-xr-x 2 root root 4096 Nov 18 06:33 lib64
   drwxr-xr-x 2 root root 4096 Nov 18 06:33 libexec
   drwxr-xr-x 2 root root 4096 Nov 18 06:33 sbin
   drwxr-xr-x 5 root root 4096 Nov 18 06:33 share
   drwxr-xr-x 2 root root 4096 Nov 18 06:33 src
   ```

5. First find out your **IP** address:

   ```
   $ ip addr show
   ```

   ```
   1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   ...    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
   ↪  1000
       link/ether 08:62:66:45:4d:16 brd ff:ff:ff:ff:ff:ff
       inet 192.168.1.9/24 brd 192.168.1.255 scope global noprefixroute eno1
          valid_lft forever preferred_lft forever
   ```

   or

   ```
   $ ifconfig
   ```

   ```
   eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
           inet 192.168.1.9  netmask 255.255.255.0  broadcast 192.168.1.255
   ```

   Then give it to your neighbor, who should be able to mount your **NFS** share with:

   ```
   $ sudo mount 192.168.1.9:/usr/local /mnt
   ```

   Warning: Corporate or other firewalls may get in the way!

Exercise 7.3: Archiving (Backing Up) the Home Directory

Archiving (or backing up) your files from time to time is essential good hygiene. You might type a command and thereby unintentionally clobber files you need and did not mean to alter.
   ```

Furthermore, while your hardware may be deemed fairly reliable, all devices do fail in some fashion eventually (even if it is just an unexpected power failure). Often this happens at the worst possible time. Periodically backing up files is a good habit to get into.

It is, of course, important to do backups to external systems through a network, or onto external storage, such as an external drive or **USB** stick. Here we will be making a backup archive on the same system, which is very useful, but won't help you if the drive fails catastrophically, or your computer is stolen or the building gets zapped by an asteroid or a fire.

First, using **tar**, back up all files and subdirectories under your **home directory**. Place the resulting **tarball** file in the `/tmp` directory, giving it the name `backup.tar`.

Second, accomplish the same task with **gzip** compression using the `-z` option to **tar**, creating `/tmp/backup.tar.gz`

Compare the size of the two files (with `ls -l`).

For additional experience make backups using the `-j` option using **bzip2** compression, and `-J` option for using xz compression.

Solution 7.3

To construct a tarball archive of your home directory you can do:

```
$ tar -cvf /tmp/backup.tar ~
```

or equivalently

```
$ tar -cvf /tmp/backup.tar /home/student
```

Back up your login directory using **tar**. Note you can have omit the - in the options with no change. In the following we will not bother using the `-v` option for verbose.

To create archives with all three compression utilities:

```
$ tar zcf /tmp/backup.tar.gz ~
$ tar jcf /tmp/backup.tar.bz2 ~
$ tar Jcf /tmp/backup.tar.xz ~
```

Comparing the sizes (first using the `-h` option to **ls** to make it **human**-readable):

```
student@ubuntu:~$ ls -lh /tmp/backup*
```

```
-rw-rw-r-- 1 student student 8.3M Apr 17 10:14 /tmp/backup2.tar.gz
-rw-rw-r-- 1 student student  12M Apr 17 10:13 /tmp/backup.tar
-rw-rw-r-- 1 student student 8.4M Apr 17 10:15 /tmp/backup.tar.bz2
-rw-rw-r-- 1 student student 8.3M Apr 17 10:14 /tmp/backup.tar.gz
-rw-rw-r-- 1 student student 8.2M Apr 17 10:15 /tmp/backup.tar.xz
```

and then without it:

```
student@ubuntu:~$ ls -l /tmp/backup*
```

```
-rw-rw-r-- 1 student student  8686942 Apr 17 10:14 /tmp/backup2.tar.gz
-rw-rw-r-- 1 student student 12226560 Apr 17 10:13 /tmp/backup.tar
-rw-rw-r-- 1 student student  8720491 Apr 17 10:15 /tmp/backup.tar.bz2
-rw-rw-r-- 1 student student  8686929 Apr 17 10:14 /tmp/backup.tar.gz
-rw-rw-r-- 1 student student  8551064 Apr 17 10:15 /tmp/backup.tar.xz
student@ubuntu:~$
```

Note in this case there is not much difference in the different archiving methods, but this particular directory was a bad choice because it already contained a lot of compressed files. A somewhat better example involving more text files:

```
$ tar cf  /tmp/doc.tar     /usr/share/doc
$ tar zcf /tmp/doc.tar.gz  /usr/share/doc
$ tar jcf /tmp/doc.tar.bz2 /usr/share/doc
$ tar Jcf /tmp/doc.tar.xz  /usr/share/doc

$ ls -lh /tmp/doc.tar*
```

```
-rw-rw-r-- 1 student student 85M Apr 17 10:34 /tmp/doc.tar
-rw-rw-r-- 1 student student 31M Apr 17 10:35 /tmp/doc.tar.bz2
-rw-rw-r-- 1 student student 34M Apr 17 10:34 /tmp/doc.tar.gz
-rw-rw-r-- 1 student student 28M Apr 17 10:36 /tmp/doc.tar.xz
```

which shows **xz** did best, followed by **bz2** and then **gz**. You may have noticed, however, the inverse relationship between the speed of the compression and how long it took!

Exercise 7.4: diff and patch

Linux and other open source communities often use the **patch** utility to disseminate modifications and updates. Here, we will give a practical introduction to using **diff** and **patch**.

It would be a good idea to read the **man pages** for both **patch** and **diff** to learn more about advanced options and techniques, that will help one to work more effectively with **patch**. In particular, the form of patches has a lot to do with whether they can be accepted in their submitted form.

1. Change to the `/tmp` directory.
2. Copy a text file to `/tmp`. For example, copy `/etc/group` to `/tmp`.
3. **dd** can not only copy directly from raw disk devices, but from regular files as well. (Remember, in **Linux**, everything is pretty much treated as a file.)

   dd can also perform various conversions; the `conv=ucase` option will convert all of the characters to upper-case characters. We will use **dd** to copy the text file to a new file in `/tmp` while converting characters to upper-case, as in

   ```
   $ dd if=/tmp/group of=/tmp/GROUP conv=ucase
   ```

4. According to **man** page for **patch**, the preferred options for preparing a patch with **diff** are `-Naur` when comparing two directory trees recursively. We will ignore the `-a` option which means treat all files as text, since **patch** and **diff** should only be used on text files anyway.

   Since we are just comparing two files, we do not need to use the `N` or `r` options to `diff`, but we could use them anyway as it will not make a difference.

   Compare `group` and `GROUP` using **diff**, and prepare a proper patch file.
5. Use **patch** to patch the original file, `/tmp/group`, so its contents now match those of the modified file, `/tmp/GROUP`. (You might try with the `--dry-run` option first!)
6. Finally, to prove that your original file is now patched to be the same as the one with all upper-case characters, use **diff** on those two files. The files should be the same and you won't get any output from **diff**.

Solution 7.4

For this exercise, you could use any text file, but we will use `/etc/group` as described.

1. `$ cd /tmp`
2. `$ cp /etc/group /tmp`
3. `$ dd if=/tmp/group of=/tmp/GROUP conv=ucase`

```
1+1 records in
1+1 records out
963 bytes (963 B) copied, 0.000456456 s, 2.1 MB/s
```

4. `$ diff -Nur group GROUP > patchfile`
   `$ cat patchfile`

```
--- group       2015-04-17 11:03:26.710813740 -0500
+++ GROUP       2015-04-17 11:15:14.602813740 -0500
@@ -1,68 +1,68 @@
-root:x:0:
-daemon:x:1:
-bin:x:2:
-sys:x:3:
....
-libvirtd:x:127:student
-vboxsf:x:999:
+ROOT:X:0:
+DAEMON:X:1:
+BIN:X:2:
+SYS:X:3:
.....
```

5. `$ patch --dry-run group patchfile`

```
checking file group
```

`$ patch  group patchfile`

```
patching file group
```

Note you could have also done either of these two commands:

```
$ patch group < patchfile
$ patch < patchfile
```

6. `$ diff group GROUP`

Exercise 7.5: 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 7.5

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
```

Exercise 7.6: 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
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.

Chapter 8

Bash Shell Scripting

Learning Objectives

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

- Explain the features and capabilities of **bash** shell scripting.
- Know the basic syntax of scripting statements and be familiar with various methods and constructs used, including the use of functions.
- Use command substitutions to insert the output of one command into the input of another.
- Perform arithmetic operations using scripting language.
- Use conditional statements, such as `if-then-else` blocks.
- Manipulate strings to perform actions such as comparison and sorting.
- Use Boolean expressions when working with multiple data types, including strings or numbers, as well as files.
- Use `case` statements to handle command line options.
- Use looping constructs to execute one or more lines of code repetitively.
- Debug scripts using `set -x` and `set +x`.
- Create temporary files and directories.
- Create and use random numbers.

8.1 Scripts

What are Shell Scripts?

- Text files that contain commands
  - Parsed by a **shell interpreter** such as **bash**
  - First line should always start with a reference to the interpreter:

```
#!/bin/bash
```

- Executed on the fly:
  - Performance loss (as compared to compiled program)
  - Large potential productivity gain
- Executable: can be invoked directly on the command line:

```
$ chmod +x script.sh
$ ./script.sh
```

Any sequence of commands that could be issued from the command line may be placed in a **shell script**, which is just a simple text file.

The text file can be passed to the shell interpreter manually:

```
$ bash script.sh
```

or can be executed directly as in:

```
$ ./script.sh
```

Note that this last example requires permissions on the script to be set properly in advance:

```
$ chmod +x ./script.sh
```

Furthermore, the first line of the file needs to point to the shell interpreter, e.g.:

```
#!/bin/bash
```

While we are going to concentrate on **bash** scripts, the above discussion is equally applicable to scripts for other command shells such as **csh** or **ksh**, as well as for interpreted languages such as **Perl** or **Python**.

Why Use Shell Scripts?

- **Combine** long and/or difficult to type and/or repetitive commands, into one simple command
- **Create** new commands using combinations of utilities
- **Share** procedures among multiple users
- **Automate** tasks and thus reduce the risk of errors
- **Control** the user interface
- **Prototype** quickly with no need to compile

Using the **Linux** command line interface can be complicated as well as time-consuming.

However, when coupled with shell scripts, using the command line interface becomes an efficient and quick way to launch complex sequences of commands.

Shell scripts are typically used to combine commands. They provide us with the glue necessary to chain multiple commands together and automate repetitive tasks.

The fact that shell scripts are saved in a file also makes it easy to edit them, or create new commands, and also facilitates sharing standard procedures among multiple users and tasks.

Typical Shell Scripts

- Help implement boot and system startup
- Automate maintenance and administration tasks
- Execute well-defined operations when packages are installed
- Give reproducible results
- Make debugging easier
- Make applications easier to start (startup scripts)

Shell scripts are commonly incorporated in (and automatically executed by) software packages.
For example, scripts associated with the **kernel** package on **Red Hat**-based distributions can be displayed with:

```
$ rpm -q --scripts kernel
....
```

Similarly, packages under **Debian**-based distributions also contain control files that are implemented as shell scripts:

```
$ dpkg-query --control-list ipcalc
md5sums
```

```
$ dpkg-query --control-show ipcalc md5sums
....
```

Why Use bash?

- Default on most systems
- Available on all distributions
- Widely documented
- Incorporates features that enhance interactive usage, such as tab completion.
- Consistent and reliable across all recent versions
- Startup files make it easy to modify the shell environment

bash is not only widely available, but it is the default shell on most distributions. It is a powerful, yet simple, scripting language with all the features of a programming language. Users can learn useful, standard operations quickly, but advanced features allows access to complex operations.

bash Scripts

- A simple **bash** shell script: follows:

```
#!/bin/bash

echo "Hello World. This is our first shell script."
```

- Make sure you make the script executable:

```
$ chmod a+x script.sh
```

- Execute it:

```
$ ./script.sh
```

"Hello World" is the standard first program for lots of languages. While simple enough, it does make sure that you can edit the code, run it and produce output.

If you can do that, you have a template to begin to add features, such as reading user input, manipulate it and provide output; i.e, set up an interactive session.

bash Scripts

- A slightly more interactive script:

```sh
#!/bin/bash
echo "Hello World. Enter a number:"
read num
echo "Number entered was  \"$num\""
```

- Note the use of \" so the quotation marks are taken literally

This example adds reading input from a user, showing how the basic interactive session works.

script.sh

```
#!/bin/bash
 echo "Hello World. Enter a number:"
 read num
 echo "Number entered was \"$num\""
$ ls -l script.sh
 -rwxrwxr-x. 1 zelda zelda 88 Mar  8 18:11 script.sh
$ ./script.sh
 Hello World. Enter a number:
 42
 Number entered was "42"
```

Return Value

- All shell scripts generate a return value
- Success is `0`, failure is non-zero
- Can explicitly supply a default value with `exit` as in:


```
#!/bin/bash
...
if [[ sometest ]] ; then
# success
   exit 0
else
# failure
   exit -1
fi
```

- Also applies to **functions** within scripts
- Value visible immediately after by examining `$?` variable

Well-behaved applications and shell scripts generate a **return** value when they exit. This allows the parent process (which can be a shell) to monitor the exit state of the application using the `$?` special variable.

An easy way to compare success versus failure is to execute the **ls** command on a file that exists, and one that does not:

```
$ ls /etc/passwd
```

```
/etc/passwd
```

```
$ echo $?
```

```
0
```

```
$ ls nofile
```

```
ls: cannot access nofile: No such file or directory
```

```
$ echo $?
```

```
2
```

Return values from standard applications can usually be converted into meaningful messages. In this case, if you do **man ls** and search for `Exit status`, you can uncover what a return value of `2` means.

Built-in Commands

- Commands can be of several types:
  - Compiled applications
  - Other scripts
  - **bash** built-in commands
- Entering `help` will get a list of the built-in **bash** commands
- Built-in commands are more efficient because they do not start a separate process in order to run.
- Built-in commands, however, can differ slightly in behavior

Compiled applications are any binary executable file (or script) that the shell script has access to, such as: **rm, ls, df, touch, vi**, and **zip**.

bash built-in commands are implemented directly by the shell script interpreter, such as **cd, pwd, echo, read, logout, printf, let,** and **ulimit**. For some of these built-in commands, programs exist with the same name on the filesystem.

To see the difference compare

```
$ /usr/bin/echo --help
```

with

```
$ help echo
```

A list of **bash** built-in commands can be obtained by simply typing

```
$ help
```

and about a particular command as in

```
$ help time
```

On some recent systems you may even find things like:

```
$ cat /bin/cd
```

```
#!/bin/sh
builtin cd "$@"
```

8.2 Features

Script Parameters

- Script behavior is modified by arguments (parameters)
- Arguments are strings, may be interpreted as numbers

```
$ ./script.sh /tmp
$ ./script.sh 100 200 300
```

- Parameters automatically assigned variable names:
  - $0 : script name (./script.sh)
  - $1 : first parameter (100)
  - $2 : second parameter (200)
  - $* : all parameters (100 200 300)
  - $# : number of arguments (3)

Passing arguments (parameters) to scripts permits running them under a variety of conditions or to accomplish different desired tasks. **bash** designates each parameter with a $ followed by a number, so $3 is the third parameter etc. The script name is stored in $0. The script environment variable $* stores all the arguments. In addition, $# stores the number of arguments.

SH testargs.sh

```
#!/bin/bash
echo The name of this program is: $0
echo The first argument passed from the command line is: $1
echo The second argument passed from the command line is: $2
echo All of the arguments passed from the command line are: $*
echo
echo All done with $0
```

```
$ ./testargs.sh one two three
```

```
The name of this program is: ./testargs.sh
The first argument passed from the command line is: one
The second argument passed from the command line is: two
All of the arguments passed from the command line are: one two three

All done with ./testargs.sh
```

Redirection

- Many scripts have an input stream and/or an output stream
  - Input is usually the keyboard (**standard in**)
  - Output is usually the terminal (**standard out**)
- Both can be redirected:

```
$ foobar < filein > fileout
$ echo 1 | tr 1 2
```

- These three command lines are functionally the same:

```
$ less file
$ less < file
$ cat file | less
```

- To append to a file:

```
$ echo another line >> file
```

```
$ free -mt > /tmp/free.out
```

```
$ cat /tmp/free.out
```

```
              total        used        free      shared  buff/cache   available
Mem:          15902        1440        9104         487        5357       13689
Swap:          8191           0        8191
Total:        24094        1440       17296
```

```
$ echo 1
```

```
1
```

```
$ echo 1 | tr 1 2
```

```
2
```

/dev/null and /dev/zero

- Anything written to `/dev/null` is discarded
- Often called the **bit bucket** or **black hole**
- Writes always succeed, reads always fail
- Often used to ignore stdout:

```
$ find / > /dev/null
```

- Or to ignore both stdout and stderr:

```
$ find / -iname "*.tmp" -exec rm '{}' ';' >& /dev/null
```

- `/dev/zero` is useful for creating empty files and other purposes:

```
$ dd if=/dev/zero of=/tmp/imagefile bs=1M count=200
```

Linux commands and scripts may generate a considerable amount of output, and perhaps you are only interested in seeing if any errors are warnings (what comes out on standard error) and not the normal output (on standard output).

A simple command such as `find /`, for instance, can quickly overload the console with messages.

One can ignore output by redirecting it to the special file, `/dev/null`. This file (also called the **bit bucket** or **black hole**) discards all data that gets written to it and never returns a failure on write operations.

Using the right redirection operator, it can be coupled with commands that would normally generate output to both stdout and stderr:

```
$ find / -name "*.swp" -exec rm '{}' ';' > /dev/null 2>&1
$ find / -name "*.swp" -exec rm '{}' ';' >& /dev/null
```

(Note both redirection forms work with **bash**).

`/dev/zero` just generates an infinite stream of zeros. This can have a variety of uses and is a good trick to have in your pocket.

Generating Random Data

- Generate a 16-bit random number with `$RANDOM`

```
$ echo $RANDOM
```

- Read from `/dev/urandom` or `/dev/random`

```
$ dd if=/dev/urandom of=/tmp/urandom.bin bs=1M count=100
$ dd if=/dev/random  of=/tmp/random.bin  bs=1M count=100
```

- `/dev/random` is slower than `/dev/urandom`, but of much higher "quality"
- Rely on the **openSSL** library for more control in generating one random number:

```
$ openssl rand -base64 128
```

Linux provides multiple ways to generate random data, which can be useful to reinitialize storage devices, erase existing data or simply to generate payload for tests.

The **bash** shell provides the `$RANDOM` environment variable for this purpose: it is guaranteed to contain a sequence of numbers that lack any pattern.

For more extensive random numbers, use the **Linux** kernel's built-in random number generator. This allows scripts and applications to read from a device special file, either `/dev/urandom` or `/dev/random` (the former being faster and the later being stronger) in order to continuously generate random data.

The **OpenSSL** library, finally, also provides hooks into a random number generator that is typically used for cryptographic key generation. The generator used by **OpenSSL** uses a **FIPS 140** approved algorithm and has been widely tested and reviewed. For instance, to generate 1024 bytes of binary random data to a file:

```
$ openssl rand -out random-data.bin 1024
```

Special Characters

Table 8.1: **Special Characters**

Character	Description
#	Comment
\	Logically joins the next line
;	Interpret what follows as a new line
$	Used to expand environment variables
&&	Logical and
||	Logical or
<, >, > >	Redirection
!	Piping

Comments in `bash` are prefixed with a '#' character. If inserted at the beginning of a line, the interpreter ignores the whole line:

```
# This line will not get executed.
```

The '\' character makes it possible to wrap long commands over multiple lines and generally makes them easier to read. The following command would thus be interpreted as one line:

```
$ rm /tmp/file1 \
     /tmp/file2 \
     /tmp file3
```

Technically, the '\' character escapes the special meaning of the next character (like a newline); we saw this in our slightly more interactive script where we escaped the special meaning of the quotation marks when we displayed the entered number.

It is also common, especially in interactive mode, to group several commands on a single line with the ';' character and have them executed logically in a separate manner:

```
$ cd /; ls ; cd /home/op/
```

The logical **and** (&&) and **or** (||) will be discussed when we consider the use of **conditionals** in scripts.

8.3 Functions

Functions

- Used to group commands logically for later execution
- A single **function** name is used for the group
- Declared with curly braces as in:

SH **Shell code**

```
showpath () {
echo $PATH
}
```

- Must be defined before being used
- Can take arguments

Executed like a regular command within the script:

```
showpath
```

Complex shell scripts often have to repeat the same operations multiple times. A good way to avoid duplication of code is to group operations together for later execution.

Such groups of commands are called **functions** or **routines**. In order to avoid collisions you must be careful to make sure that a script has only one function with a given name.

SH **addpath.sh**

```
#!/bin/bash

addtopath () {
# here $1 is the argument to the function,
#         need not be the same
  ADDEDPATH=$1
  echo Original Path= $PATH
  PATH=$PATH:$ADDEDPATH
  echo New Path=$PATH
   }
# here $1 is the argument to the script
addtopath $1

$ ./addpath.sh /tmp/bin

Original Path= /usr/local/bin:/usr/bin:/bin
New Path=/usr/local/bin:/usr/bin:/bin:/tmp/bin
```

8.4 Command Substitutions and Arithmetic

Command Substitution

The constructs for executing a command recycling the output as input into another command (**command substitution**) provide a very powerful and useful capability.

- Two ways to substitute the result of a command **into** another command:

```
$ cd /lib/modules/$(uname -r)/
$ cd /lib/modules/`uname -r`/
```

- The first form allows nesting, the second does not
- Note in the second form, the delimiters are “backticks” (`) not apostrophes (')

The ability to take the output of a command and use that as a part of another command is very powerful. This allows commands to be created dynamically, using one command to gather information for another command.

The form with backticks is limited because the left and right delimiters are the same so you can not nest operations, as you can with the other form. Here is an example of nested substitution:

First generate a list of files with the **find** command. Then pass that list to the **ls** command and display the output.

```
$ echo $(ls $(find /usr/local/images/ -name "*"))
```

Example:

```
$ cd /lib/modules/$(uname -r)
$ pwd
```

```
/lib/modules/3.19.2
```

```
$ cd /lib/modules/`uname -r`
$ pwd
```

```
/lib/modules/3.19.2
```

Arithmetic Expressions

- Arithmetic expressions can be evaluated in three ways:
- Using the `$((...))` syntax, a shell built-in:

```
$ echo $((x+1))
```

- Using **let**, also a shell built-in:

```
$ let x=( 1 + 2 ); echo $x
```

- Using the **expr** utility:

```
$ expr 8 + 8
$ echo $( expr 8 + 8 )
```

- Use of **expr** is discouraged because:
  - It is an external program and thus has more overhead
  - It is very sensitive to spacing, you must have white space

Arithmetic expansion allows the evaluation of an arithmetic expression and the substitution of the result.

The built-in **bash** shell syntax `$(( expression ))` tends to make complex arithmetic expressions a lot easier to implement.

The expression within the brackets is escaped, making it possible to use special characters without worrying about shell expansion.

While **expr** is available as a stand-alone executable, its use tends to be complex due to the need to escape special characters such as *, and because it is picky about white space:

```
$ echo $( 1 + 2 )
```

```
3
```

```
$ echo ( 1+2 )
```

```
1+2
```

Strings and Basic Operations

- Any character, number, punctuation
- Double quote if space
- Can be compared
- Can be manipulated
- Comparing sorting order:

```
[[ string1 > string2 ]]
```

- Comparing two strings:

```
[[ string1 == string2 ]]
```

- Getting the length of a string:

```
myLen1=${#mystring1}
```

The greater and less than operators (> and <) consider the **sorting order** of the strings. The equals operators == and **!=**) compares the strings character by character. This enables, for example, the comparison of an input string with an existing variable, like yes or no.

```
$ if [ "abc" > "def" ]; then
> echo "First string comes alphabetically before second string."
> fi
```

```
First string comes alphabetically before second string.
```

```
$ pwd
```

```
/home/beaver
```

```
$ if [ `pwd` == $HOME ]; then
```

```
> echo "I am in my HOME directory."
> fi
I am in my HOME directory.
```

```
$ export NAME="Beaver"
$ echo "My name is $NAME. My name is ${#NAME} characters long."
```

```
My name is Beaver. My name is 6 characters long.
```

8.5 If Conditions and Tests

The if Statement

- Take different actions based on success or failure of a command or conditional test which can be:
  - Numerical or string comparison test
  - Return value of a command (0 for success)
  - File tests such as existence, type, etc
- Example:

```
if [ -f /etc/passwd ]
then
    echo "/etc/passwd exists"
else
    echo "/etc/passwd does not exist"
fi
```

- Can use `[[ -f /etc/passwd ]]` as well

Decision making using `if` statement constructs is important when different possible courses of action can be taken, depending on evaluated conditions. All real programming languages (and interpreters) have this elementary capability.

`if -> then -> else -> fi` constructs are often used in scripts, but can also be used at the command line, usually in a more compact form:

```
if [ TEST ]; then SOME_COMMANDS; fi
```

The test condition can be phrased with double brackets as in

```
if [[ TEST ]]; then SOME_COMMANDS; fi
```

Note however, not all script interpreters are compatible with this, in particular the original Bourne shell **sh** is not. One advantage of double brackets is when there is a case where you are testing a string that is not defined. Suppose `STR` has never been set.

```
[ $STR == "TEST" ]
```

will fail (with a syntax error) while either of the following expressions:

```
[[ $STR == "TEST" ]]
[ "$STR" == "TEST" ]
```

will work as intended.

Boolean Values and Operators

- Values:
  - TRUE
  - FALSE
- Operators:
  - AND &&
  - OR ||
  - NOT !

Boolean operations concern logic, not math. The operators do not produce numerical values, just evaluate TRUE or FALSE, based on the comparison. For example if A **and** B are true, then do something, but only if both are true. If one is not true, then do something else. Another example, if A **or** B are true, then do something. It does not matter which one is true, as long as one of them is true, otherwise if both are false, then do something else.

Tests

Testing is a fundamental requirement for any script or program. The ability to examine a data element and make a decision is key.

Some examples are below:

- File exists or not:

  `[ -e filename ]`

  Result is `TRUE` or `FALSE`

- `number1` is greater than `number2`:

  `[ $number1 -gt $number2 ]`

  Result is `TRUE` or `FALSE`

Some examples of tests in the `if -> then -> else` construct.

```
$ ls
```

```
   def  file-1  file-2  temp
```

```
$ if [ -e file-2 ]; then
> echo "File 'file-2' exists. Continue processing."
> else
> echo "File 'file-2' does not exist. Stop processing."
> exit 1
> fi
```

```
   File 'file-2' exists. Continue processing.
```

```
$ export number1=42; export number2=$((number1-25*2))
$ if [ $number1 -gt $number2 ]; then
> echo "$number1 is greater than $number2"
> else
> echo "$number1 is less than or equal to $number2"
> fi
```

```
   42 is greater than -8
```

Using Logical Operators

Using logical && (and) and || (or).

- Conditions can be combined with && (logical and):

```
if [ test1 ] && [ test2 ] && [ test3 ] ; then ... ; fi
```

  Processing stops at the first test that fails, returning false

- Conditions can be combined with || (logical or):

```
if [ test1 ] || [ test2 ] || [ test3 ] ; then ... ; fi
```

  Processing stops at the first test that succeeds, returning true

- Compact notation can drop explicit if-then-else construct:

```
$ [ test1 ] && [ test2 ] && some_command
```

- Any condition can be negated:

```
if [ ! test1 ] ; then do something ; fi
```

Multiple tests can be performed at once in evaluating conditional branches.

With the logical and &&, processing continues until one of the tests fails, and only succeeds if all tests answer with true.

```
$ make && make modules_install && make install
```

With the logical or ||, processing continues until one of the tests succeeds, and then answers true. If none succeed, false is returned.

```
$ [[ -f /etc/foo.conf ]] || echo 'default config' > /etc/foo.conf
```

Testing for Files

It is a normal function to examine a file for many conditions, like does the file exist. The **test** command has a series of common tests in a short form. Some of the most common conditional tests are:

Table 8.2: **File Conditional Tests**

Test	Meaning
`-e file`	`file` exists?
`-d file`	`file` is a directory?
`-f file`	`file` is a regular file?
`-s file`	`file` has non-zero size?
`-g file`	`file` has `sgid` set?
`-u file`	`file` has `suid` set?
`-r file`	`file` is readable?
`-w file`	`file` is writeable?
`-x file`	`file` is executable?

```
if [ -e $source ] ; then echo $source exists ; fi
if [ ! -x $prog ] ; then echo $prog is not executable ; fi
```

The file tests come in several varieties. One mistake is to use `-e` when you really should use `-f`, depending on intent of course.

```
$ [[ -e /dev/null ]] && echo /dev/null exists
```

```
/dev/null exists
```

```
$ [[ -f /dev/null ]] || echo /dev/null is not a normal file
```

```
/dev/null is not a normal file
```

Testing for Strings

Table 8.3: **String Comparisons**

Test	Meaning
`string`	`string` not empty?
`string1 == string2`	strings same?
`string1 != string2`	strings differ?
`-n string`	`string` not null?
`-z string`	`string` null?

```
if [ string1 == string2 ] ; then
    ACTION
fi
```

Strings can be tested for and compared in a number of ways. Does the string exist? Is it equal to another string? For sorting purposes, is one string greater than another?

Here is an example where the user is asked a yes or no question and then subsequent processing depends on the answer.

somescript.sh

```
#!/bin/bash
echo "Enter yes or no:"
read RESPONSE
if [ $RESPONSE == "yes" ]; then
        echo "The answer was yes!"
else
        echo "The answer was not yes!"
fi
```

```
$ ./somescript.sh
```

```
Enter yes or no:
yes
The answer was yes!
```

```
$ ./somescript.sh
```

```
Enter yes or no:
garbage
The answer was not yes!
```

Testing for Numbers

Numbers can be tested by comparison, is the value of number1 bigger than the value of number2. There are several comparison operators (or just operators) available.

- General format is:

  `exp1 -op exp2`

- `-op` can be:

```
-eq : equal
-ne : not equal
-gt : greater than
-lt : less than
-ge : greater than or equal
-le : less than or equal
```

One example of this is when you try to take more money from the ATM than you have in your account.

```
$ cat atm.sh
```

SH **atm.sh**

```
#!/bin/bash
LIMIT=250
echo "How much money do you want from your account?"
read INPUT
if [ $INPUT -ge $LIMIT ]
then
        echo "You do not have enough money in your account."
else
        echo "Please take your money from the tray."
fi

$ ./atm.sh
How much money do you want from your account?
300
You do not have enough money in your account.
$ ./script5.sh
How much money do you want from your account?
200
Please take your money from the tray.
```

8.6 Looping Structures

for Loops

bash supports several looping structures that are best illustrated by examples.

- `for` loop in words:
  for the variable file in the list generated by find
  do something
  until all done

- `for` loop in code

```
for file in $(find . -name "*.o")
do
    echo "I am removing file: $file"
    rm -f $file
done
```

There are many ways to do the tasks often assigned to for loops, such as the **find** and **xargs**. Here are some alternatives for the example shown above.

```
$ find . -name "*.o" -exec rm {} ';'
```

or showing use of the **xargs** utility.

```
$ find . -name "*.o" | xargs rm
```

Everyday operations might include compact commands such as:

```
$ for dir in LF???? ; do echo $dir ; pushd $dir ; make ; popd ; done
```

Do this often enough and you will get muscle memory for it in your fingers.

while Loops

The **while** loops perform a command(s) as long as the tested condition remains true.

- **while** loop in words:
  - set value of **a** to 7
  - test, a is less than 4, no
  - decrement a
  - back to the test
- bash code

```
#!/usr/bin/bash
a=7
while [ $a -gt 4 ]
do
    echo $a
    ((a--))
done
echo "Out of the loop"
```

A more complex example using **if** and **while** loops.

SH testwhile.sh

```
#!/bin/sh
ntry_max=4 ; ntry=0 ; password=' '
while [[ $ntry -lt $ntry_max ]] ; do
   ntry=$(( $ntry + 1 ))
   echo -n 'Give password:  '
   read password
   if  [[ $password == "linux" ]] ; then
     echo "Congrats: You gave the right password on try $ntry!"
     exit 0
   fi
   echo "You failed on try $ntry; try again!"
done
echo "you failed $ntry_max times; giving up"
exit -1
```

until Loops

while loops are similar to **until** loops. **while** loops continue when the condition is true, **until** loops trigger on a negative condition.

```
#!/bin/bash
echo "until loop"
i=10
until [ $i == 1 ]
do
    echo "$i is not equal to 1";
    i=$((i-1))
done
echo "i value is $i"
echo "loop terminated"
```

```
#
   # starting number for i
# the until loop stops at 1
 # top of loop

     # decrement i
   # end of loop
```

SH **testuntil.sh**

```
#!/bin/sh
ntry_max=4 ; ntry=0 ; password=' '
until [[ $ntry -ge $ntry_max ]] ; do
  ntry=$(( $ntry + 1 ))
  echo -n 'Give password:  '
  read password
  if [[ $password == "linux" ]] ; then
    echo "Congrats You gave the right password on try $ntry!"
    exit 0
  fi
  echo "You failed on try $ntry; try again!"
done
echo "you failed $ntry_max times; giving up"
exit -1
```

8.7 Case Structure

case Statements

- Useful for conditions with several possible results
- Helps avoid nested `if` statements, which can get ugly
- Basic structure:

```
case EXPRESSION in
        CASE1)
           command
           ;;
        CASE2)
           command
           ;;
        *)
           command
           ;;
esac
```

`case` statements are often used to handle command-line options, such as in initialization scripts. If a known choice is not given, a default action is taken

```
case "$1" in
  start)
    start
    ;;
  stop)
    stop
    ;;
  restart)
    stop
    start
    ;;
  force-reload|reload)
       reload
    ;;
  *)
    echo $"Usage: $0 {start|stop|restart|force-reload|reload|help}"
esac
```

8.8 Debugging

Debugging Script Options

- Run a script in **debug mode** with

  ```
  $ bash -x  script.sh
  ```

  – Traces and prefixes each command with a +
  – Prints arguments before commands are executed
- Debug parts of a script with:

  ```
  set -x
  ...
  set +x
  ```

Scripts are a list of shell commands acted upon line by line, there is an option to have bash display each line before executing it, this is done by enabling the **set -x** option to bash.

The **set -x** output will show the script lines with the variables expanded.

SH **script.sh**

```
#! /bin/bash
set -x
progname=$0; argument=$1
case $argument in
        start)
           echo "Start up the application."
                ;;
        *)
           echo "Usage: $progname {start|help}"
                exit 2
                ;;
esac
set +x
        . . .
```

```
$ ./script.sh start
```

```
+ progname=./script.sh
+ argument=start
+ case $argument in
+ echo 'Start up the application.'
Start up the application.
+ set +x
```

Logging Errors Output

- Redirect errors to a file:

```
$ ./script.sh 2>> /tmp/scriptlogfile
```

- Redirect errors both to a file and to the screen:

```
$ ./script.sh 2> >(tee stderr.log >&2)
```

`$ ./script.sh 2>> /tmp/scriptlogfile:`

The

`2>>`

takes standard-error (`stderr`) (file descriptor 2), from the running of the **./script.sh** program and appends it to the file `/tmp/scriptlogfile`.

`$ ./script.sh 2>>(tee stderr.log >2):`

2> redirects `stderr` (file descriptor 2) to the same place as standard-output (`stdout`).

Here we are using the **process substitution** feature in **bash**, where the input stream (`stdin`, due to the '>' character) is connected to the running of **./script.sh** in this example via a **named pipe** (also called a **fifo**). In this case of `>(tee stderr.log >&2)`, the **tee** program has its `stdin` coming in from the stdout of **./script.sh**.

Remember, the 2> put `stderr` in the same stream as `stdout`, so both are becoming the `stdin` for **tee**, which will write its `stdin` to `stderr.log`.

The `>&2` part will take `stdout` from **tee** and write it to the stream used for `stderr` (file descriptor 2). So, as stated in the slide, this will write the `stderr` to both the file `stderr.log` and to the terminal screen.

8.9 Creating Temporary Files and Directories

Temporary Files and Directories

- Scripts often need to create or use temporary files
  - Typically used to store data not used after the task is accomplished
- Creating a **random** filename is the best practice
  - Only accessible to user who creates the file (600)
- Use **mktemp** with XXXXXX to define the random field:

```
$ mktemp /tmp/temp.XXXXXX
```

- Creating temporary directories can be done with the -d option:

```
$ TEMP=$(mktemp -d /tmp/tempdir.XXXXXXXX)
$ mv file $TEMP
```

Just using a simple file name to create temporary files can open your script to attacks. If someone with access to the /tmp directory reads that your script does something like this:

```
$ echo $VAR > /tmp/tempfile
```

The attacker can then do something like:

```
$ ln -s /etc/passwd /tmp/tempfile
```

When the script will be executed by the root user, the password file will thus be overwritten. In order to avoid this situation make sure you randomize your temporary filenames by replacing the line above with:

```
$ TEMP=$(mktemp /tmp/tempfile.XXXXXXXX)
$ echo $VAR > \TEMP
```

8.10 Labs

Exercise 8.1: Exit Status Codes

Write a script which:

1. Does **ls** for a non-existent file, and then displays the resulting exit status.
2. Creates a file and does **ls** for it, and then once again displays the resulting exit status.

Solution 8.1

testls.sh

SH testls.sh

```
#!/bin/bash
#
# check for non-existent file, exit status will be 2
#
ls SoMeFiLe.ext
echo "status: $?"

# create file, and do again, exit status will be 0
touch SoMeFiLe.ext
ls SoMeFiLe.ext
echo "status: $?"

# remove the file to clean up
rm SoMeFiLe.ext
```

Exercise 8.2: Working with Files and Directories in a Script

Write a script that:

1. Prompts the user for a directory name and then creates it with **mkdir**.
2. Changes to the new directory and prints out where it is using **pwd**.
3. Using **touch**, creates several empty files and runs **ls** on them to verify they are empty.
4. Puts some content in them using **echo** and redirection.
5. Displays their content using **cat**.
6. Says goodbye to the user and cleans up after itself.

Solution 8.2

testfile.sh

SH testfile.sh

```
#!/bin/bash

# Prompts the user for a directory name and then creates it  with mkdir.
```

SH

```
echo "Give a directory name to create:"
read NEW_DIR

# Save original directory so we can return to it (could also just use pushd, popd)

ORIG_DIR=$(pwd)

# check to make sure it doesn't already exist!

[[ -d $NEW_DIR ]] && echo $NEW_DIR already exists, aborting && exit
mkdir $NEW_DIR

# Changes to the new directory and prints out where it is using pwd.

cd $NEW_DIR
pwd

# Using touch, creates several empty files and runs ls on them to verify they are empty.

for n in 1 2 3 4
do
    touch file$n
done

ls file?

# (Could have just done touch file1 file2 file3 file4, just want to show do loop!)

# Puts some content in them using echo and redirection.

for names in file?
do
    echo This file is named $names
done

# Displays their content using cat

cat file?

# Says goodbye to the user and cleans up after itself

cd $ORIG_DIR
rm -rf $NEW_DIR
echo "Goodbye My Friend!"
```

Make it executable and then run it:

```
$ chmod +x testfile.sh
```

```
$ testfile.sh
```

```
Give a directory name to create:
SOME_DIR
/tmp/SOME_DIR
/tmp/SOME_DIR
file1  file2  file3  file4
This file is named file1
This file is named file2
This file is named file3
This file is named file4
```

```
Goodbye My Friend!
```

Exercise 8.3: Environment variables

Write a script which:

1. Asks the user for a number, which should be `1` or `2`. Any other input should lead to an error report.
2. Sets an environmental variable to be `Yes` if it is `1`, and `No` if it is `0`.
3. Exports the environmental variable and displays it.

Solution 8.3

testenv.sh

testenv.sh

```
#!/bin/bash

echo "Enter 1 or 2, to set the environmental variable EVAR to Yes or No"
read ans

# Set up a return code
RC=0

if [ $ans -eq 1 ]
then
    export EVAR="Yes"
else
    if [ $ans -eq 2 ]
    then
        export EVAR="No"
    else
# can only reach here with a bad answer
        export EVAR="Unknown"
        RC=1
    fi
fi
echo "The value of EVAR is: $EVAR"
exit $RC
```

Make it executable and then run it:

```
$ chmod +x testenv.sh
```

```
$ ./testenv.sh
```

```
Enter 1 or 2, to set the environmental variable EVAR to Yes or No
1
The value of EVAR is: Yes
```

```
$ ./testenv.sh
```

```
Enter 1 or 2, to set the environmental variable EVAR to Yes or No
2
The value of EVAR is: No
```

```
$ ./testenv.sh
Enter 1 or 2, to set the environmental variable EVAR to Yes or No
somestring
```

```
./testenv.sh: line 9: [: string: integer expression expected
./testenv.sh: line 13: [: string: integer expression expected
The value of EVAR is: Unknown
```

Exercise 8.4: Functions

Write a script which:

1. Asks the user for a number (1, 2 or 3).
2. Calls a function with that number in its name. The function should display a message with its name included.

Solution 8.4

testfun.sh

testfun.sh

```
#!/bin/bash

# Functions (must be defined before use)
func1() {
echo " This message is from function 1"
}
func2() {
echo " This message is from function 2"
}
func3() {
echo " This message is from function 3"
}

# Beginning of the main script

# prompt the user to get their choice
echo "Enter a number from 1 to 3"
read n

# Call the chosen function
func$n
```

Make it executable and then run it:

```
$ chmod +x testfun.sh
```

```
$ ./testfun.sh
```

```
Enter a number from 1 to 3
2
 This message is from function 2
```

```
$ ./testfun.sh
```

```
Enter a number from 1 to 3
7
./testfun.sh: line 21: func7: command not found
```

Exercise 8.5: Arithmetic and Functions

Write a script that will act as a simple calculator for add, subtract, multiply and divide.

1. Each operation should have its own function.
2. Any of the three methods for **bash** arithmetic, (`$((..))`, `let`, or `expr`) may be used.
3. The user should give 3 arguments when executing the script.
   The first should be one of the letters `a`, `s`, `m` or `d` to specify which math operation.
   The second and the third arguments should be the numbers that are being operated on.
4. The script should detect for bad or missing input values and display the results when done.

Solution 8.5

testmath.sh

SH testmath.sh

```
#!/bin/bash

# Functions.  must be before the main part of the script

# in each case method 1 uses $((..))
#              method 2 uses let
#              method 3 uses expr

add() {
    answer1=$(($1 + $2))
    let answer2=($1 + $2)
    answer3=`expr $1 + $2`
}
sub() {
    answer1=$(($1 - $2))
    let answer2=($1 - $2)
    answer3=`expr $1 - $2`
}
mult() {
    answer1=$(($1 * $2))
    let answer2=($1 * $2)
    answer3=`expr $1 \* $2`
}
div() {
    answer1=$(($1 / $2))
    let answer2=($1 / $2)
    answer3=`expr $1 / $2`
}
# End of functions
#

# Main part of the script

# need 3 arguments, and parse to make sure they are valid types
```

SH

```
op=$1 ; arg1=$2 ; arg2=$3

[[ $# -lt 3 ]] && \
    echo "Usage: Provide an operation (a,s,m,d) and two numbers" && exit 1

[[ $op != a ]] && [[ $op != s ]] && [[ $op != d ]] && [[ $op != m ]] && \
    echo operator must be a, s, m, or d, not $op as supplied

# ok, do the work!

case $op in
    a) add $arg1 $arg2 ;;
    s) sub $arg1 $arg2 ;;
    m) mult $arg1 $arg2 ;;
    d) div $arg1 $arg2 ;;
    *) echo $op is not a, s, m, or d, aborting
        exit 2 ;;
esac

# Show the answers
echo $arg1 $op $arg2 :
echo 'Method 1, $((..)),' Answer is  $answer1
echo 'Method 2, let,    ' Answer is  $answer2
echo 'Method 3, expr,   ' Answer is  $answer3
```

Make it executable and then run it:

```
$ chmod +x testmath.sh

$ for n in a s m d  x ; do ./testmath.sh $n 21 7 ; done
```

```
21 a 7 :
Method 1, $((..)), Answer is 28
Method 2, let,     Answer is 28
Method 3, expr,    Answer is 28
21 s 7 :
Method 1, $((..)), Answer is 14
Method 2, let,     Answer is 14
Method 3, expr,    Answer is 14
21 m 7 :
Method 1, $((..)), Answer is 147
Method 2, let,     Answer is 147
Method 3, expr,    Answer is 147
21 d 7 :
Method 1, $((..)), Answer is 3
Method 2, let,     Answer is 3
Method 3, expr,    Answer is 3
operator must be a, s, m, or d, not x as supplied
x is not a, s, m, or d, aborting
$
```

Exercise 8.6: Using the case Statement

Write a script that takes as an argument a month in numerical form (i.e., between 1 and 12), and translates this to the month name and displays the result on standard out (the terminal).

If no argument is given, or a bad number is given, the script should report the error and exit.

Solution 8.6

testmonths.sh

testmonths.sh

```
#!/bin/bash

# Accept a number between 1 and 12 as
# an argument to this script, then return the
# the name of the month that corresponds to that number.

# Check to see if the user passed a parameter.
if [ $# -eq 0 ]
then
  echo "Error. Give as an argument a number between 1 and 12."
  exit 1
fi

# set month equal to argument passed for use in the script
month=$1

################################################
# The example of a case statement:

case $month in

  1)  echo "January"   ;;
  2)  echo "February"  ;;
  3)  echo "March"     ;;
  4)  echo "April"     ;;
  5)  echo "May"       ;;
  6)  echo "June"      ;;
  7)  echo "July"      ;;
  8)  echo "August"    ;;
  9)  echo "September" ;;
  10) echo "October"   ;;
  11) echo "November"  ;;
  12) echo "December"  ;;
  *)
     echo "Error. No month matches: $numb"
     echo "Please pass a number between 1 and 12."
     exit 2
     ;;
esac
exit 0
```

Make it executable and then run it:

```
$ chmod +x testmonths.sh
$ ./testmonths.sh 7
```

```
July
```

```
$ ./testmonths.sh 17
```

```
Error. No month matches: 17
Please pass a number between 1 and 12.
```

```
$ ./testmonths.sh notanumber
```

```
Error. No month matches: notanumber
Please pass a number between 1 and 12.
```

Exercise 8.7: Using Random Numbers

Create a script which:

1. Takes a word as an argument
2. Appends a random number to it
3. Displays the answer

Solution 8.7

testrandom.sh

Notice the use of $0 in the usage message, which evaluates to the name of the script.

testrandom.sh

```
#!/bin/bash
##
# check to see if the user supplied in the parameter.

[[ $# -eq 0 ]] && echo "Usage: $0 word" && exit 1

echo "$1-$RANDOM"
exit 0
```

Make it executable and then run it:

```
$ chmod +x testrandom.sh
```

```
$ ./testrandom.sh stringA
```

```
stringA-21658
```

```
$ ./testrandom.sh stringB
```

```
stringB-4879
```

```
$ ./testrandom.sh stringC
```

```
stringC-25791
```

Exercise 8.8: String Tests and Operations

Write a script which reads two strings as arguments and then:

1. Tests to see if the first string is of zero length, and if the other is of non-zero length, telling the user of both results.

2. Determines the length of each string, and reports on which one is longer or they are of equal length.
3. Compares the strings to see if they are the same, and reports on the result.

Solution 8.8

teststring.sh

SH teststring.sh

```
#!/bin/bash

# check two string arguments were given
[[ $# -lt 2 ]] && \
    echo "Usage: Give two strings as arguments" && exit 1

str1=$1
str2=$2

#-------------------------------------
## test command

echo "Is string 1 zero length? Value of 1 means FALSE"
[ -z "$str1" ]
echo $?
# note if $str1 is empty, the test [ -z $str1 ] would fail
#                               but [[ -z $str1 ]] succeeds
#         i.e., with [[ ]] it works even without the quotes

echo "Is string 2 nonzero length? Value of 0 means TRUE;"
[ -n $str2 ]
echo $?

## comparing the lengths of two string

len1=${#str1}
len2=${#str2}
echo length of string1 = $len1, length of string2 = $len2

if [ $len1 -gt $len2 ]
then
    echo "String 1 is longer than string 2"
else
    if [ $len2 -gt $len1 ]
    then
        echo "String 2 is longer than string 1"
    else
        echo "String 1 is the same length as string 2"
    fi
fi

## compare the two strings to see if they are the same

if [[ $str1 == $str2 ]]
then
    echo "String 1 is the same as string 2"
else
    if [[ $str1 != $str2 ]]
    then
        echo "String 1 is not the same as string 2"
    fi
```

SH

```
fi
```

Make it executable and then run it:

```
$ chmod +x teststring.sh
```

```
$ ./teststring.sh dog cat
```

```
Is string 1 zero length? Value of 1 means FALSE
1
Is string 2 nonzero length? Value of 0 means TRUE;
0
length of string1 = 3, length of string2 = 3
String 1 is the same length as string 2
String 1 is not the same as string 2
```

```
$ ./teststring.sh dog ""
```

```
Is string 1 zero length? Value of 1 means FALSE
1
Is string 2 nonzero length? Value of 0 means TRUE;
0
length of string1 = 3, length of string2 = 0
String 1 is longer than string 2
String 1 is not the same as string 2
```

Chapter 9

Networking

Learning Objectives

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

- Explain the use of **IP** addresses and the differences between **IPv4** and **IPv6**.
- Decode **IPv4** addresses and Network Classes.
- Obtain local addresses through the use of **dhclient** and **NetworkManager**, and show the **hostname**.
- Use information utilities such as **nslookup** and **dig**.
- Deploy **ip** and **ifconfig** to examine and configure network interfaces on **Linux** systems.
- Use networking diagnostic utilities, including **ping**, **route**, **traceroute**, **mtr**, **ethtool**, and **netstat**.

9.1 Addressing

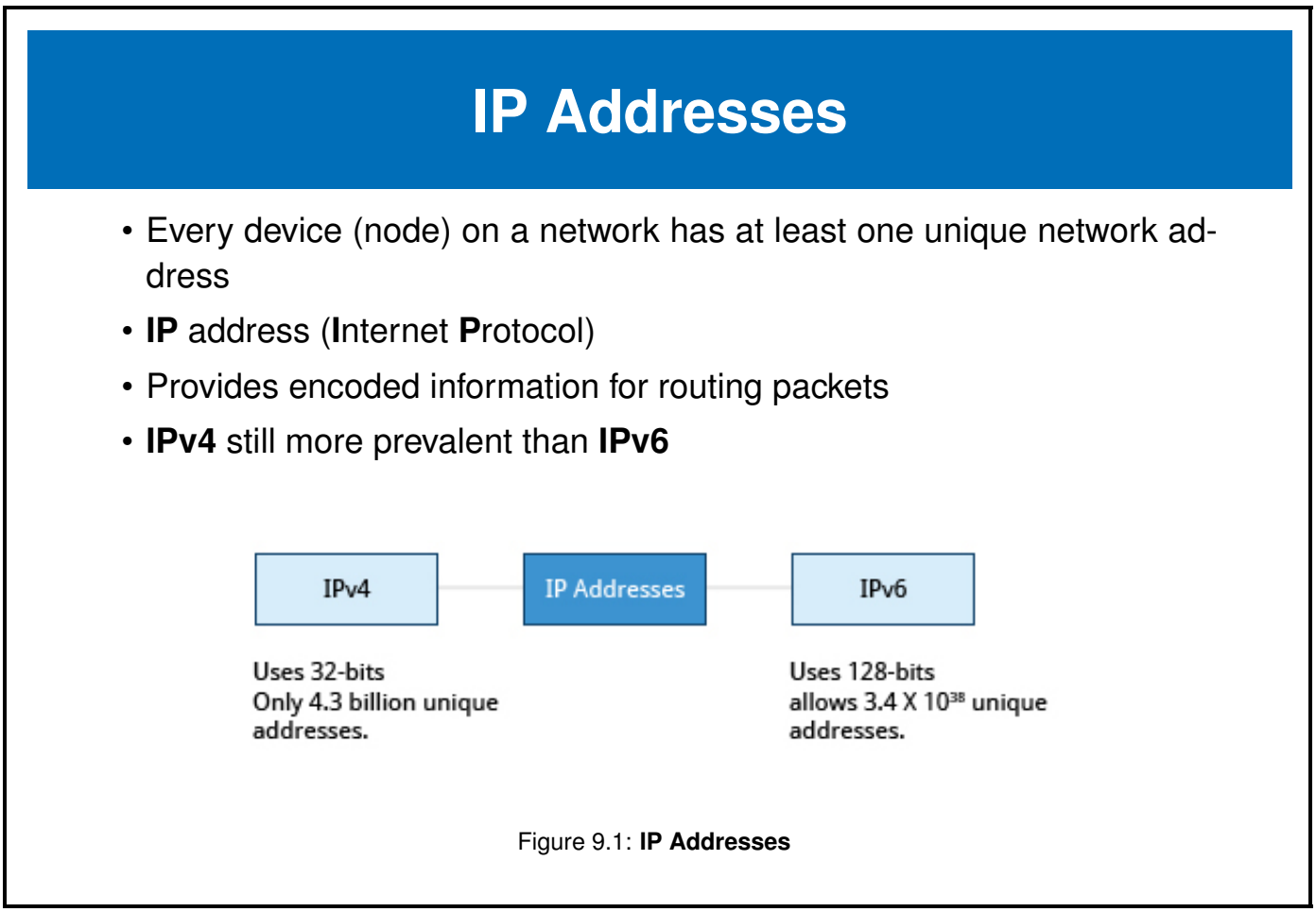


Figure 9.1: **IP Addresses**

IPv4 (version 4) addresses use 32-bits, so there are *only* 4.3 billion unique addresses available. This sounds like a lot, but many of the addresses are not actually available for use. Furthermore, the number of devices available on the Internet has exponentially increased.

IPv6 (version 6) uses 128-bits for addresses. This allows for $3.4x10^{38}$ unique addresses!

Migration to **Ipv6** has been slower than expected, partly because of the complexity of having the two protocol versions interoperate.

Furthermore, many local networks of organizations, companies etc., have been able to handle having many **IP** addresses by skillfully using **NAT** (**N**etwork **A**ddress **T**ranslation.)

The emergence of the **Internet of Things** (**IOT**) has required many more addresses, and requires full eventual adoption of **IPv6**.

Decoding IPv4 Addresses

- IP addresses divided into two parts:
  - **Network** (Class A, B and C) and **Host**
- Addresses are 32 bits in length (four **octets**)
- Sample IP address: 172.16.31.46

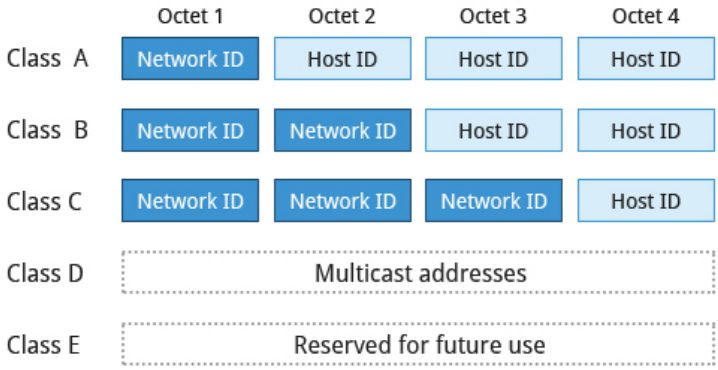


Figure 9.2: **Decoding IPv4 Addresses**

Three **classes** of network addresses are in use: **A**, **B** and **C**. The class is defined by how many octets (bytes) are used for the network part of the IP address. Note, there is also Class **D** (used for special multicast applications) and **E** (reserved for future use) networks; we do not need to cover them in this course.

- Class **A** networks use only the first octet for the network part of their addresses. The first bit of the first octet is set to zero, so only 7-bits are used for unique network numbers. Because of this, there are a maximum of 127 (7-bits - 1 to 127) Class **A** networks available. Each Class **A** network can have up to 16.7 million unique hosts on their network; the host part uses the last three octets (or 24-bits). For example: `10.233.45.23`.
- Class **B** networks use the first two octets for the network part of the IP address and the last two octets for the host part of the IP address. The first two bits of the first octet must be set to binary 10. This means that there are a maximum of 16,384 (14-bits) Class **B** networks. The first octet of a Class **B** network will have values from 128 to 191. Each Class **B** network can support a maximum of 65,536 hosts. For example: `135.201.18.1`.
- Class **C** networks use the first three octets of the IP address for the network number and the last octet for the host. The first three bits of the first octet must be set to binary 110. There are a maximum of almost 2.1 million (21-bits) Class **C** networks. The first octet of a Class **C** network will be a value from 192 to 223. Each Class **C** network can support up to 256 (8-bits) unique hosts. For example: `192.33.84.156`.

IP Address Allocation

- Usually done automatically through **DHCP** (**D**ynamic **H**ost **C**onfiguration **P**rotocol)
  - Usually administered through **NetworkManager** GUI
  - Or, from the command line:

```
$ sudo dhclient eth0
```

- Can also be done **statically**
  - Fixed address
  - Assigned through GUI, or **ip** or the older **ifconfig** utility

```
$ sudo ip addr add 192.168.1.147/24 dev eth0
$ sudo ifconfig eth0 up 192.168.1.147
```

- Display system **hostname**:

```
$ hostname
student7
```

- **Note**: `127.0.0.1` translates to `localhost`

The configuration of the network adapters is normally done at system startup time and for most distributions done by the network administrator tool **NetworkManager**.

There are a few elements required to configure a **static network configuration**.

- An IP address is required to identify our machine on the network.
- A related netmask is needed to define the number of bits in the ipaddress for hosts and the network component.
- A default router IP address is required so we can send traffic to other machines on the Internet.
- A **DNS** server is required to translate IP addresses to names.

Most machines use **DHCP** (**Dynamic Host Configuration Protocol**) to assign an available IP address.

We will not discuss how to configure a **DHCP** server in this course, only how to employ such an address.

Domain Name System (DNS)

- Associates IP Addresses with names through a **DNS** server
- Local name resolution configuration takes precedence over **DNS** results:

  `/etc/hosts`
- Name server configuration:

  `/etc/resolv.conf`
- Look up hostnames using **DNS**:

```
$ host linuxfoundation.org
$ nslookup linuxfoundation.org
$ dig linuxfoundation.org
```

The **D**omain **N**ame **Service** is a network service that associates domain names with IP Addresses.

Before DNS was used local `/etc/hosts` files would contain all the addresses, this did not scale as there are millions of domain names in use. The DNS solved the scalability issue.

Since name resolution started as a local file most systems still support local files with name and IP Addresses. The local file is typically called `/etc/hosts`. The local files are normally queried before the network DNS.

The `/etc/hosts` contains the local name server information and the file `/etc/resolv.conf` contains the IP Address of a DNS server and optionally the domain name to search. The resolv.conf file is generally automatically created at install time, or by **NetworkManager**

Often the `/etc/hosts` file is very minimal with just `localhost` defined:

```
$ cat /etc/hosts
  127.0.0.1 localhost
```

Your `/etc/resolv.conf` file probably points at a system (your ISP or, perhaps, itself) running the Domain Name Service:

```
$ cat /etc/resolv.conf
  nameserver 192.168.53.1
```

host,nslookup and dig

- three common tools used to request information from the **DNS**. **host**, **nslookup** and **dig**.
- **host** command is commonly used in scripts, minimal output
- **nslookup** command
  - most popular
  - available on almost any system
  - medium amount of output
- **dig** command
  - displays the most information
  - shows the question and answer
  - format of data similar to DNS database

Note: The output from the following commands has been shortened, see the lab exercise for more verbose output.

```
$ host -t a linux.com
```

```
linux.com has address 23.185.0.3
```

```
$ nslookup linux.com
```

```
Server:               192.168.1.1
Address:        192.168.1.1#53

Non-authoritative answer:
Name:        linux.com
Address: 23.185.0.3
Name:        linux.com
Address: 2620:12a:8000::3
```

```
$ dig linux.com
```

```
; <<>> DiG 9.11.4-P2-RedHat-9.11.4-26.P2.el8 <<>> linux.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27448
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;linux.com.                     IN        A

;; ANSWER SECTION:
linux.com.              300        IN        A        23.185.0.3

;; Query time: 34 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
```

9.2 Networking Interfaces and Configuration

Network Interfaces

- Point of interconnection between device and network
- Software based configuration
- Can be activated or deactivated
- Display current active network interfaces:

```
$ ip link show
$ ifconfig
```

- Historically have been named:
  - Wired: `eth0`, `eth1`, `eth2` etc.
  - Wireless: `wlan0`, `wlan1`, `wlan2` etc.
- Modern **systemd**-based distributions create names based on other characteristics such as how the hardware is connected:

  Examples: `enp0s25`, `eno1`, `wlp3s0`

```
$ ip link show
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode  DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether 08:62:66:45:4d:16 brd ff:ff:ff:ff:ff:ff
```

```
$ ifconfig
```

```
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.200  netmask 255.255.255.0  broadcast 192.168.1.255
        ether 08:62:66:45:4d:16  txqueuelen 1000  (Ethernet)
        RX packets 10191534  bytes 5037288749 (4.6 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 63960583  bytes 91355390756 (85.0 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 20  memory 0xf7d00000-f7d20000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 0  (Local Loopback)
        RX packets 15757  bytes 621212 (606.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 15757  bytes 621212 (606.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Network Configuration Files

- Configuration Files
  - **Debian**-based systems
    `/etc/network/interfaces`
  - **Red Hat** and SUSE-based systems
    `/etc/sysconfig/network`, `/etc/sysconfig/network-scripts`
- On modern systems using **systemd**, better to use **Network Manager** rather than directly edit these files
- Start/stop/restart networking configuration:

  ```
  $ sudo systemctl [start][stop|restart] NetworkManager
  ```

  The name might be `networkmanager`, all lowercase on some systems
- May also have to start the **network** service first.
- Exact name of service will depend on **Linux** distribution

The exact names and locations of **Linux** network configuration files tend to be somewhat dependent on **Linux** distribution as well as version; you may have to do some hunting in the `/etc` directory tree.

Starting, stopping and restarting network services is done just like for any other system service. Thus, for **systemd**-based systems, the relevant utility is **systemctl**.

The exact name of the services also varies from distribution to distribution. Modern systems permit most of the work of configuring and turning network interfaces off to be done from the graphical desktop using **Network Manager**, with normal users often being able to do much of it without special privilege.

9.3 Networking Utilities and Tools

ip and ifconfig

- Two multi-capability utilities often used to configure interfaces:
  - **ifconfig**: **UNIX**-based utility with long history
  - **ip**: More recent utility which has mostly replaced **ifconfig**
- Some **Linux** distributions no longer install **ifconfig** by default.
  - Learn to use **ip** !

The **ip** and **ifconfig** utilities are used to observe and modify various aspects of the network and its interfaces. When used interactively, both commands can configure the network and its devices.

ifconfig is available an almost any system although its name may change on some non-Linux systems. The **ifconfig** command has a long history but recently **ifconfig** cannot display some configuration values created by changes in the network operation.

ip is the newcomer and it is supported on most Linux systems. **ip**does not suffer from the same issues as **ifconfig**. Although many distributions still make **ifconfig** available, it is not recommended. Use **ip**.

```
$ ip --help
```

```
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] -batch filename
where
   OBJECT := { link | addr | addrlabel | route | rule | neigh | ntable |
               tunnel | tuntap | maddr | mroute | mrule | monitor | xfrm |
               netns | l2tp | tcp_metrics | token }
   OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                -f[amily] { inet | inet6 | ipx | dnet | bridge | link } |
                -4 | -6 | -I | -D | -B | -0 |
                -l[oops] { maximum-addr-flush-attempts } |
                -o[neline] | -t[imestamp] | -b[atch] [filename] |
                -rc[vbuf] [size]}
```

ping

- Confirms remote system is online and responding
- Network testing, measurement and management
- May impose load on network
- To check if a system is online and visible:

```
$ ping some_system.com
```

The **ping** program is used to confirm remote system is online and responding. It is available on almost every system that has network connection.

ping sends an **ICMP echo request** via the `Internet Control Message Protocol`(ICMP) to a remote host. The remote host responds with a **ICMP echo reply** message. The host that sends the ICMPecho request collects sequencing and timing information to display on the originator's system to calculate the return time for the request.

It is possible for the **ping** utility to generate sufficient traffic as to interface with normal operations of either or both hosts.

```
$ sudo ping -n -c 6 google.com
```

```
PING google.com (216.58.192.174) 56(84) bytes of data.
64 bytes from 216.58.192.174: icmp_seq=1 ttl=50 time=29.8 ms
64 bytes from 216.58.192.174: icmp_seq=2 ttl=50 time=29.1 ms
64 bytes from 216.58.192.174: icmp_seq=3 ttl=50 time=27.9 ms
64 bytes from 216.58.192.174: icmp_seq=4 ttl=50 time=28.2 ms
64 bytes from 216.58.192.174: icmp_seq=5 ttl=50 time=30.8 ms
64 bytes from 216.58.192.174: icmp_seq=6 ttl=50 time=28.9 ms

--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 13ms
rtt min/avg/max/mdev = 27.935/29.127/30.784/0.976 ms
```

ping is interrupted with `CTRL-C`; then the summary is printed. Or you can limit the number of packets sent with the `-c` option. In the example we have also used the `-n` option to make each line shorter.

route

- Show current routing table:

```
$ route
```

- Add static route:

```
$ sudo route add -net address
```

- Delete static route:

```
$ sudo route del -net address
```

- **ip** utility also does routing, more modern:

```
$ ip route show
$ sudo ip route add 10.5.0.0/16 via 192.168.1.100
```

The **route** and **ip route** commands will display and alter the IP routing table. The IP routing table is a network component that describes how IP is to route packets, specifically what is the next system to send the packet to. Routing is required when systems are on different sub-nets and wish to communicate. To get from one network to another there must be a system/appliance/hardware that has both sub nets available and has the ability to forward packets enabled. The routing table contains **default** and **static** routes. Default routes are used if there is no specific or static route defined to reach the desired network.

Show the routing table:

```
$ ip route show
```

```
default via 192.168.1.1 dev eno1 proto dhcp metric 100
172.16.203.0/24 dev vmnet1 proto kernel scope link src 172.16.203.1
172.16.249.0/24 dev vmnet8 proto kernel scope link src 172.16.249.1
192.168.1.0/24 dev eno1 proto kernel scope link src 192.168.1.200 metric 100
```

```
$ route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         gateway         0.0.0.0         UG    100    0        0 eno1
172.16.203.0    0.0.0.0         255.255.255.0   U     0      0        0 vmnet1
172.16.249.0    0.0.0.0         255.255.255.0   U     0      0        0 vmnet8
192.168.1.0     0.0.0.0         255.255.255.0   U     100    0        0 eno1
```

traceroute and mtr

- **traceroute**:
  - Checks packet route to destination machine
  - Used for network troubleshooting
  - Isolate connectivity issue between hops
  - To print route to networked system:

    ```
    $ traceroute some_system.some_domain.com
    ```

- **mtr**
  - Combines functionality of **traceroute** and **ping**
  - Updates periodically, like **top**

    ```
    $ mtr some_system.some_domain.com
    ```

traceroute tracks the route packets take through the network to another host. It uses ICMP messages with a short `time-to-live`(TTL) on the packet in an attempt to prompt each gateway along the way to reply with a **ICMP TIME_EXCEEDED** message. Traceroute sends 3 probe packets per host to measure the **Round Trip Time**(RTT) to each gateway. Variations in the RTT could indicate network issues.

mtr combines functionality of **traceroute** and **ping**. Traceroute, like ping runs until terminated unless an option is passed to mtr. To get mtr to run for 3 cycles, print a report and terminates the options **-c 3 -r** to mtr. An **ICMP ECHO** is sent every second, this can be changed with the **-i** option. mtr collects more data than **ping** or **traceroute** that is readily available in many formats.

```
$ traceroute google.com
```

```
traceroute to google.com (172.217.0.14), 30 hops max, 60 byte packets
1  router (192.168.1.1)  0.373 ms  0.461 ms  0.530 ms
2  * * *
3  96-34-31-180.static.unas.mn.charter.com (96.34.31.180)  30.816 ms  32.368 ms  32.773 ms
4  acr06ftbgwi-gbe-8-46.ftbg.wi.charter.com (96.34.30.244)  32.508 ms  32.596 ms  32.672 ms
    *** output shortened ***
```

```
$ mtr -c 3 -r linuxfoundation.org
```

```
Start: 2020-04-06T10:52:43-0500
HOST: c8                          Loss%   Snt   Last   Avg  Best  Wrst StDev
1.|-- router                      0.0%     3    0.3   0.3   0.3   0.4   0.0
2.|-- ???                       100.0      3    0.0   0.0   0.0   0.0   0.0
3.|-- 96-34-31-180.static.unas.  0.0%     3    9.4  10.5   9.4  11.5   1.1
4.|-- acr06ftbgwi-gbe-8-46.ftbg  0.0%     3    9.8   9.6   8.7  10.2   0.8
5.|-- crr01onlswi-bue-402.onls.  0.0%     3   13.9  13.8  13.3  14.1   0.4
*** output shortened ***
```

ethtool

- Gets and sets network interfaces properties and parameters
- Sets link speed and duplex
- Tune many other interface properties
- Examples:

```
$ ethtool -S eno1  # show statistics about interface eno1
$ sudo ethtool -s eno1 speed 1000 duplex full autoneg on  # set speed
$ sudo ethtool -K eno1 tso on # turn on tcp-segmentation-offload
```

```
$ sudo ethtool eno1
```

Executing the **ethtool** with only the adapter of interest, **ethtool** reports the current values of the parameters supported by the selected adapter.

```
Settings for eno1:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 1
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: off (auto)
        Supports Wake-on: pumbg
        Wake-on: g
        Current message level: 0x00000007 (7) drv probe link
        Link detected: yes
```

netstat

- Displays all active connections
  - **Internet** and local **Unix Domain Sockets** tabulated separately
- Displays routing tables

```
$ netstat -r
```

```
Kernel IP routing table
Destination Gateway     Genmask         Flags MSS Window irtt Iface
default     192.168.1.1 0.0.0.0         UG      0 0      0    eno1
172.16.2.0  0.0.0.0     255.255.255.0   U       0 0      0    vmnet8
172.16.43.0 0.0.0.0     255.255.255.0   U       0 0      0    vmnet1
192.168.1.0 0.0.0.0     255.255.255.0   U       0 0      0    eno1
```

netstat can check network configuration and activity. Most network attributes can be observed with netstat. For more information consult the `netstat(8)` man page.

```
$ netstat
```

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 c7:49292               129.33.139.56:https     ESTABLISHED
tcp        0      0 c7:51610               a23-10-240-19.depl:http ESTABLISHED
tcp        0      0 c7:47494               iy-in-f108.1e100.:imaps ESTABLISHED
tcp        0      0 c7:53868               129.33.138.31:https     ESTABLISHED
tcp        0      0 c7:33344               a184-25-208-245.de:http ESTABLISHED
tcp        0      0 c7:47072               151.101.148.134:https   ESTABLISHED
tcp        0      0 c7:35502               104.16.80.166:https     ESTABLISHED
tcp        0      0 c7:43196               167.127.154.104.b:tripe ESTABLISHED
tcp        0      0 c7:33378               o2.ycpi.vip.ne1.y:https ESTABLISHED
.....
tcp6       0      0 c7:41287               40.83.21.197:https      ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  3      [ ]         SEQPACKET  CONNECTED     55454    @000b4
unix  3      [ ]         SEQPACKET  CONNECTED     56437    @000b5
unix  3      [ ]         DGRAM                    15368    /run/systemd/notify
unix  2      [ ]         DGRAM                    15370    /run/systemd/cgroups-agent
unix  5      [ ]         DGRAM                    15384    /run/systemd/journal/socket
unix  30     [ ]         DGRAM                    15386    /dev/log
....
```

More Network Tools

- **nmap** scans open ports on a network

```
$ nmap -sP linuxfoundation.org
```

- **tcpdump** dumps network traffic for analysis

```
$ tcpdump -i eth0
```

- **iptraf** monitors network traffic in text mode graphical screen

```
$ sudo iptraf
$ sudo iptraf -i all
```

- On some newer systems the program is now called **iptraf-ng**

The nmap command can scan ports on designated host. It defaults to a very light scan of the commonly used ports. Deeper scans, inclusion of more ports and repetitive scans may be viewed as a network attack, please seek permission from stakeholders before attempting more complex scans.

```
$ nmap -v -A scanme.nmap.org
```

```
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-06 11:11 CDT
NSE: Loaded 148 scripts for scanning.
...

Scanning Scanning scanme.nmap.org (45.33.32.156) [2 ports]
Completed Ping Scan at 11:12, 0.07s elapsed (1 total hosts)
....
Scanning scanme.nmap.org (45.33.32.156) [1000 ports]
Discovered open port 22/tcp on 45.33.32.156
Discovered open port 80/tcp on 45.33.32.156
Discovered open port 31337/tcp on 45.33.32.156
Discovered open port 9929/tcp on 45.33.32.156
....
Nmap done: 1 IP address (1 host up) scanned in 18.62 seconds
```

tcpdump dumps the actual packet data, some more intense understanding of network protocols is required to make sense of the data. The enhanced network analysis is beyond the scope of this class.

```
$ sudo tcpdump -i eno1  host google.com
```

```
10:19:43.550139 IP c7.38422 > ord31s22-in-f14.1e100.net.https: Flags
 [P.], seq 3824458009:3824458115, ack 3261553128, win 953,
 options [nop,nop,TS val 12030719 ecr 2875604567], length 106
.....
^C                                  <-- Typed CTRL-C to end
18 packets captured
22 packets received by filter
0 packets dropped by kernel
```

9.4 Labs

Video Demonstration Resources

dns_demo.mp4

Exercise 9.1: Network Troubleshooting

Overview

Troubleshooting network problems is something that you will often encounter if you haven't already. We are going to practice some of the previously discussed tools that can help you isolate, troubleshoot and fix problems in your network.

Suppose you need to perform an Internet search, but your web browser can not find google.com, saying the host is unknown. Let's proceed step by step to fix this.

1. First make certain your network is properly configured. If your Ethernet device is up and running you can run the command **ip** and/or **ifconfig** and they should display similar information.

   You can show your **IP** address with:

   ```
   $ ip addr show
   ```

   ```
   1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
       link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
       inet 127.0.0.1/8 scope host lo
          valid_lft forever preferred_lft forever
       inet6 ::1/128 scope host
          valid_lft forever preferred_lft forever
   2: eno16777736: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state \
       UP group default qlen 1000
       link/ether 00:0c:29:bb:92:c2 brd ff:ff:ff:ff:ff:ff
       inet 192.168.1.14/24 brd 192.168.1.255 scope global dynamic eno16777736
          valid_lft 84941sec preferred_lft 84941sec
       inet 192.168.1.15/24 brd 192.168.1.255 scope global secondary dynamic eno16777736
          valid_lft 85909sec preferred_lft 85909sec
       inet6 fe80::20c:29ff:febb:92c2/64 scope link
          valid_lft forever preferred_lft forever
   ```

   or with **ifconfig**

   ```
   $ /sbin/ifconfig
   ```

   ```
   eno167777 Link encap:Ethernet  HWaddr 00:0C:29:BB:92:C2
             inet addr:192.168.1.14  Bcast:192.168.1.255  Mask:255.255.255.0
             inet6 addr: fe80::20c:29ff:febb:92c2/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
             RX packets:3244 errors:0 dropped:0 overruns:0 frame:0
             TX packets:2006 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:4343606 (4.1 Mb)  TX bytes:169082 (165.1 Kb)

   lo        Link encap:Local Loopback
             inet addr:127.0.0.1  Mask:255.0.0.0
             inet6 addr: ::1/128 Scope:Host
             UP LOOPBACK RUNNING  MTU:65536  Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
   ```

```
            RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

On older systems you probably will see a less cryptic name than `eno167777`, like `eth0`, or for a wireless connection, you might see something like `wlan0` or `wlp3s0`.

Does the IP address look valid? Depending on where you are using this from, it is most likely a Class **C** IP address; in the above this is `192.168.1.14`

If it does not show a device with an **IP** address, you may need to start or restart the network and/or **NetworkManager**. Exactly how you do this depends on your system. For most distributions one of these commands will accomplish this:

```
$ sudo systemctl restart NetworkManager
$ sudo systemctl restart network
```

If your device was up but had no **IP** address, the above should have helped fix it, but you can try to get a fresh address with:

```
$ sudo dhclient eth0
```

substituting the right name for the Ethernet device.

2. If your interface is up and running with an assigned **IP** address and you still can not reach google.com, we should make sure you have a valid hostname assigned to your machine, with **hostname**:

```
$ hostname
```

```
openSUSE
```

It is rare you would have a problem here, as there is probably always at least a default hostname, such as `localhost`.

3. When you type in a name of a site such as google.com, that name needs to be connected to a known IP address. This is usually done by employing a **DNS** server (**D**omain **N**ame **S**ystem)

First, see if the site is up and reachable with **ping**:

```
$ sudo ping -c 3 google.com
```

```
PING google.com (216.58.216.238) 56(84) bytes of data.
64 bytes from ord31s22-in-f14.1e100.net (216.58.216.238): icmp_seq=1 ttl=51 time=21.7 ms
64 bytes from ord31s22-in-f14.1e100.net (216.58.216.238): icmp_seq=2 ttl=51 time=23.8 ms
64 bytes from ord31s22-in-f14.1e100.net (216.58.216.238): icmp_seq=3 ttl=51 time=21.3 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 21.388/22.331/23.813/1.074 ms
```

Please Note

We have used **sudo** for **ping**; recent **Linux** distributions have required this to avoid clueless or malicious users from flooding systems with such queries.

4. We have used `-c 3` to limit to 3 packets; otherwise **ping** would run forever until forcibly terminated, say with `CTRL-C`.

If the result was:

```
ping: unknown host google.com
```

It is likely that something is wrong with your DNS setup.

Please Note

On some systems you will never see the unknown host message, but you will get a suspicious result like:

```
$ sudo ping l89xl28vkjs.com
```

```
PING l89xl28vkjs.com.site (127.0.53.53) 56(84) bytes of data.
64 bytes from 127.0.53.53: icmp_seq=1 ttl=64 time=0.016 ms
...
```

where the `127.0.x.x` address is a loop feeding back to the host machine you are on. You can eliminate this as being a valid address by doing:

```
$ host l89xl28vkjs.com
```

```
Host l89xl28vkjs.com not found: 3(NXDOMAIN)
```

whereas a correct result would look like:

```
$ host google.com
```

```
google.com has address 216.58.216.206
google.com has IPv6 address 2607:f8b0:4009:80b::200e
google.com mail is handled by 20 alt1.aspmx.l.google.com.
google.com mail is handled by 10 aspmx.l.google.com.
google.com mail is handled by 30 alt2.aspmx.l.google.com.
google.com mail is handled by 40 alt3.aspmx.l.google.com.
google.com mail is handled by 50 alt4.aspmx.l.google.com.
```

The above command utilizes the **DNS** server configured in `/etc/resolv.conf` on your machine. If you wanted to override that you could do:

```
host 8.8.8.8
```

```
8.8.8.8.in-addr.arpa domain name pointer google-public-dns-a.google.com.
```

```
student@linux:~> host google.com 8.8.8.8
```

```
Using domain server:
Name: 8.8.8.8
Address: 8.8.8.8#53
Aliases:

google.com has address 216.58.216.110
google.com has IPv6 address 2607:f8b0:4009:804::1002
...\
```

where we have used the publicly available **DNS** server provided by **Google** itself. (Using this or another public server can be a good trick sometimes if your network is up but **DNS** is ill; in that case you can also enter it in `resolv.conf`.

/etc/hosts

There is another file, /etc/hosts, where you can associate names with **IP** addresses, which is used **before** the **DNS** server is consulted. This is most useful for specifying nodes on your local network.

You could also use the **dig** utility if you prefer:

```
$ dig google.com
```

```
; <<>> DiG 9.9.5-rpz2+rl.14038.05-P1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29613
;; flags: qr rd ra; QUERY: 1, ANSWER: 11, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; MBZ: 1c20 , udp: 1280
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             244     IN      A       173.194.46.67
google.com.             244     IN      A       173.194.46.65
google.com.             244     IN      A       173.194.46.71
google.com.             244     IN      A       173.194.46.73
google.com.             244     IN      A       173.194.46.69
google.com.             244     IN      A       173.194.46.68
google.com.             244     IN      A       173.194.46.64
google.com.             244     IN      A       173.194.46.72
google.com.             244     IN      A       173.194.46.70
google.com.             244     IN      A       173.194.46.66
google.com.             244     IN      A       173.194.46.78

;; Query time: 22 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Mon Apr 20 08:58:58 CDT 2015
;; MSG SIZE  rcvd: 215
```

5. Suppose **host** or **dig** fail to connect the name to an **IP** address. There are many reasons **DNS** can fail, some of which are:

   - The **DNS** server is down. In this case try **ping**ing it to see if it is alive (you should have the **IP** address in /etc/resolv.conf.
   - The server can be up and running, but **DNS** may not be currently available on the machine.
   - Your **route** to the **DNS** server may not be correct.

   How can we test the route? Tracing the route to one of the public name servers we mentioned before:

```
student@linux:~> sudo traceroute 8.8.8.8
```

```
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  192.168.1.1 (192.168.1.1)  0.405 ms  0.494 ms  0.556 ms
 2  10.132.4.1 (10.132.4.1)  15.127 ms  15.107 ms  15.185 ms
 3  dtr02ftbgwi-tge-0-6-0-3.ftbg.wi.charter.com (96.34.24.122)
                                                    15.243 ms  15.327 ms  17.878 ms
 4  crr02ftbgwi-bue-3.ftbg.wi.charter.com (96.34.18.116)  17.667 ms  17.734 ms  20.016 ms
 5  crr01ftbgwi-bue-4.ftbg.wi.charter.com (96.34.18.108)  22.017 ms  22.359 ms  22.052 ms
 6  crr01euclwi-bue-1.eucl.wi.charter.com (96.34.16.77)  29.430 ms  22.705 ms  22.076 ms
 7  bbr01euclwi-bue-4.eucl.wi.charter.com (96.34.2.4)  17.795 ms  25.542 ms  25.600 ms
 8  bbr02euclwi-bue-5.eucl.wi.charter.com (96.34.0.7)  28.227 ms  28.270 ms  28.303 ms
```

```
 9  bbr01chcgil-bue-1.chcg.il.charter.com (96.34.0.9)  33.114 ms  33.072 ms  33.175 ms
10  prr01chcgil-bue-2.chcg.il.charter.com (96.34.3.9)  36.882 ms  36.794 ms  36.895 ms
11  96-34-152-30.static.unas.mo.charter.com (96.34.152.30)  42.585 ms  42.326 ms  42.401 ms
12  216.239.43.111 (216.239.43.111)  28.737 ms 216.239.43.113 (216.239.43.113)
                                                  24.558 ms  23.941 ms
13  209.85.243.115 (209.85.243.115)  24.269 ms 209.85.247.17 (209.85.247.17)
   25.758 ms 216.239.50.123 (216.239.50.123)  25.433 ms
14  google-public-dns-a.google.com (8.8.8.8)  25.239 ms  24.003 ms  23.795 ms
```

Again, this should likely work for you, but what if you only got the first line in the **traceroute** output?

If this happened, most likely your default route is wrong. Try:

```
$ ip route show
```

```
efault via 192.168.1.1 dev eno16777736  proto static  metric 1024
192.168.1.0/24 dev eno16777736  proto kernel  scope link  src 192.168.1.14
```

Most likely this is set to your network interface and the IP address of your router, **DSL**, or Cable Modem. Let's say that it is blank or simply points to your own machine. Here's your problem! At this point, you would need to add a proper default route and run some of the same tests we just did.

Note, an enhanced version of **traceroute** is supplied by **mtr**, which runs continuously (like **top**). Running it with the `--report-cycles` option to limit how long it runs:

```
$ sudo mtr --report-cycles 3 8.8.8.8
```

```
                             My traceroute  [v0.85]
c7 (0.0.0.0)                                        Mon Apr 20 09:30:41 2015
Unable to allocate IPv6 socket for nameserver communication: Address family not supported
             by protocol                 Packets               Pings
 Host                                 Loss%   Snt   Last   Avg  Best  Wrst StDev
                                       0.0%     3    0.3   0.3   0.2   0.3   0.0
 2. 10.132.4.1                         0.0%     3    6.3   7.1   6.3   8.4   0.7
 3. dtr02ftbgwi-tge-0-6-0-3.ftbg.wi.   0.0%     3    6.2   7.5   6.2  10.0   2.1
 4. dtr01ftbgwi-bue-1.ftbg.wi.charte   0.0%     3    8.9   8.5   6.2  10.4   2.0
 5. crr01ftbgwi-bue-4.ftbg.wi.charte   0.0%     3    8.9   9.7   8.9  10.4   0.0
 6. crr01euclwi-bue-1.eucl.wi.charte   0.0%     3   16.5  17.4  14.2  21.5   3.7
 7. bbr01euclwi-bue-4.eucl.wi.charte   0.0%     3   23.5  22.0  18.2  24.2   3.2
 8. bbr02euclwi-bue-5.eucl.wi.charte   0.0%     3   18.9  22.7  18.1  31.1   7.2
 9. bbr01chcgil-bue-1.chcg.il.charte   0.0%     3   22.9  23.0  22.9  23.1   0.0
10. prr01chcgil-bue-2.chcg.il.charte   0.0%     3   21.4  24.1  20.8  30.2   5.2
11. 96-34-152-30.static.unas.mo.char   0.0%     3   22.6  21.9  20.0  23.3   1.6
12. 216.239.43.111                     0.0%     3   21.2  21.7  21.2  22.0   0.0
13. 72.14.237.35                       0.0%     3   21.2  21.0  19.8  21.9   1.0
14. google-public-dns-a.google.com     0.0%     3   26.7  23.0  21.0  26.7   3.2
```

Hopefully, running through some of these commands helped. It actually helps to see what the correct output for your system looks like. Practice using these commands; it is very likely that you will need them someday.

Exercise 9.2: Static Configuration of a Network Interface

Please Note

You may have to use a different network interface name than `eth0`. You can most easily do this exercise with **nmtui** or your system's graphical interface. We will present a command line solution, but beware details may not exactly fit your distribution flavor or fashion.

1. Show your current IP address, default route and **DNS** settings for `eth0`. Keep a copy of them for resetting later.
2. Bring down `eth0` and reconfigure to use a static address instead of **DHCP**, using the information you just recorded.
3. Bring the interface back up, and configure the nameserver resolver with the information that you noted before. Verify your hostname and then **ping** it.
4. Make sure your configuration works after a reboot.

You will probably want to restore your configuration when you are done.

Solution 9.2

1. ```
   $ ip addr show eth0
   $ ip route
   $ cp /etc/resolv.conf resolv.conf.keep
   ```

   or

   ```
   $ ifconfig eth0
   $ route -n
   $ cp /etc/resolv.conf resolv.conf.keep
   ```

2. ```
   $ sudo ip link set eth0 down
   ```

   or

   ```
   $ sudo ifconfig eth0 down
   ```

On Red Hat / CentOS

Make sure the following is in `/etc/sysconfig/network-scripts/ifcfg-eth0` on **Red Hat**-based systems:

in /etc/sysconfig/network-scripts/ifcfg-eth0

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
IPADDR=noted from step 1
NETMASK=noted from step 1
GATEWAY=noted from step 1
```

On openSUSE , SLES, and Debian-based systems

On **SUSE**-based systems edit the file in `/etc/sysconfig/network` in the same way, and on **Debian**-based systems edit `/etc/networking/interfaces` to include:

in /etc/sysconfig/network or /etc/networking/interfaces

```
iface eth0 inet static
address noted from step 1
netmask noted from step 1
gateway noted from step 1
```

3. ```
   $ sudo ip link set eth0 up
   ```
   ```

or

```
$ sudo ifconfig eth0 up
```

```
$ sudo cp resolv.conf.keep /etc/resolv.conf
$ cat /etc/sysconfig/network
$ cat /etc/hosts
$ ping yourhostname
```

4. ```
   $ sudo reboot
   $ ping hostname
   ```

Exercise 9.3: Adding a Static Hostname

In this exercise we will add entries to the local `/etc/hosts` 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 9.3

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"
   ```

Chapter 10

Working With Linux Filesystems

Learning Objectives

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

- Explain what a filesystem is and the use of a virtual filesystem layer (**VFS**).
- Use both hard and soft (symbolic) links.
- Enumerate the most important filesystem varieties available on **Linux** systems, including **ext4**, **xfs**, **btrfs**, **squashfs** etc.
- Create and format filesystems with **mkfs**.
- Check and repair filesystems with **fsck**.
- Analyze filesystem usage with **df** and **du**.
- Mount and unmount filesystems with **mount** and **umount**.
- Work with **NFS**.
- Understand and configure `/etc/fstab` and control mounting at system boot.

10.1 Filesystem Basics

Filesystem Basics

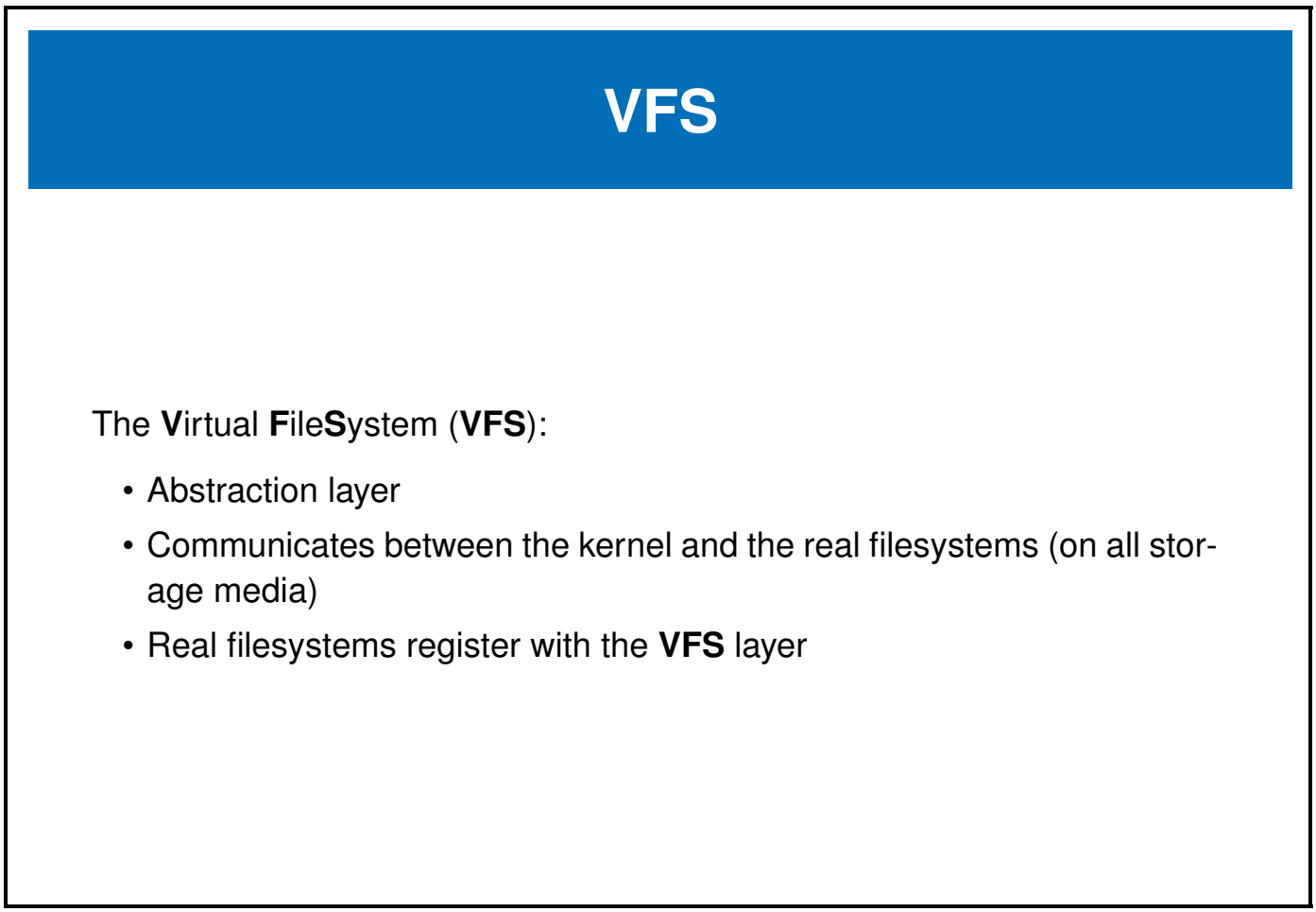
- **Linux** programs read and write files, not disk sectors
- A **file** is actually an abstraction camouflaging the physical I/O layer
- **Filesystems** create a usable format on a physical partition
- A **UNIX**-like filesystem uses a tree hierarchy
  - Directories contain files and/or other directories
  - 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.

10.2 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.

10.3 Hard and Soft Links

Hard and Soft Links

- **Hard** links point to an **inode**, a data structure on disk describing all attributes, including location.
  - 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

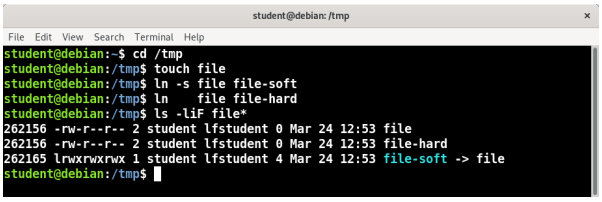


Figure 10.1: **Hard and Soft Links**

A **directory file** is a particular type of file that is used to associate file names and inodes. There are two ways to associate (or **link**) a file name with an inode:

- **Hard** links point to an inode. They are made by using **ln** without an option.
- **Soft** (or **symbolic**) links point to a file name which has an associated inode. They are made with using **ln** with the `-s` option.

Each association of a directory file contents and an inode is known as a link. Additional links can be created using **ln**.

Because it is possible (and quite common) for two or more directory entries to point to the same inode (hard links), a file can be known by multiple names, each of which has its own place in the directory structure. However, it can have only one inode no matter which name is being used.

When a process refers to a pathname, the kernel searches directories to find the corresponding inode number. After the name has been converted to an inode number, the inode is loaded into memory and is used by subsequent requests.

10.4 Available Filesystems

Available Filesystems

- **Linux** works with more filesystem varieties than any other operating system.
- This democratic flexibility has been a large factor in its success.
- Most filesystems have full read/write access, while a few have read only access.
- Commonly used filesystems include **ext4, xfs, btrfs, squashfs, nfs** and **vfat**
- A list of currently supported filesystems can be seen at `/proc/filesystems`

We can see a list of the supported filesystems in the `/proc/filesystems` file.

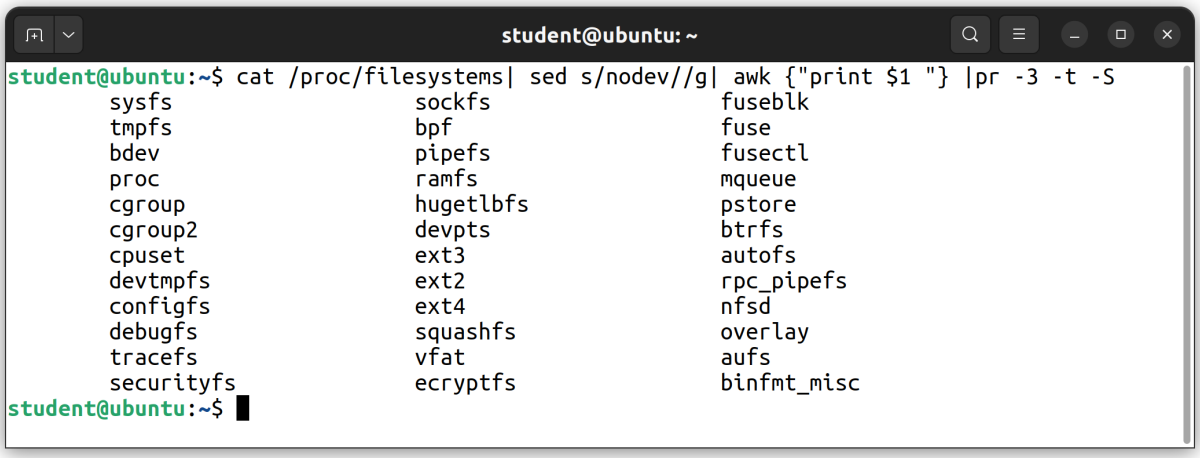


Figure 10.2: **supported filesystems**

10.5 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:

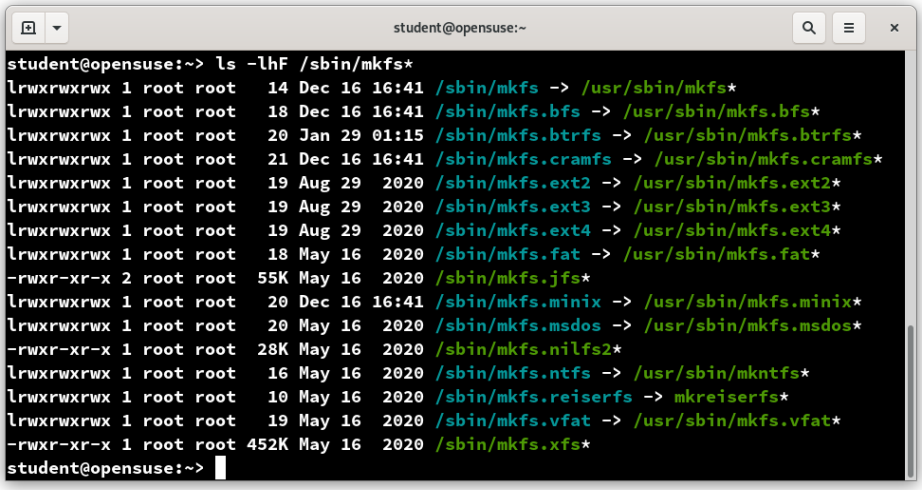


Figure 10.3: **mkfs**

One should look at the **man** page for each of the **mkfs.*** programs to see details.

10.6 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:

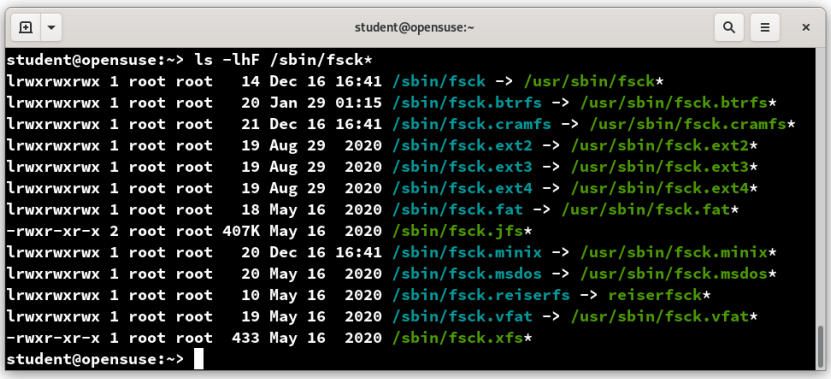


Figure 10.4: **fsck**

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.

10.7 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
```

The **df** command provides an essential function of determining the size, use and amount of free space on filesystems.

```
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 10.5: **Using df**

10.8 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`

The **du** command is file orientated, it checks for how much disk space is consumed by files.
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 10.6: **Using du**

10.9 Mounting filesystems

Mounting Filesystems

- Each filesystem is mounted under a specific directory:

```
$ mount -t ext4 /dev/sdb4 /home
```

  - Mounts an **ext4** filesystem
  - Usually not necessary to specify the type with `-t` option
  - The filesystem is located on a specific partition of a hard drive (`/dev/sdb4`)
  - The filesystem is mounted at the position `/home` in the current directory tree
  - Any files residing in the original `/home` directory are hidden until the partition is unmounted

All accessible files in **Linux** are organized into one large hierarchical tree structure with the head of the tree being the root directory (`/`). However, it is common to have more than one partition (each of which can have its own filesystem type) joined together in the same filesystem tree. These partitions can also be on different physical devices, even on a network.

The **mount** program allows attaching at any point in the tree structure; **umount** allows detaching them.

The **mount point** is the directory where the filesystem is attached. It must exist before **mount** can use it; **mkdir** can be used to create an empty directory. If a pre-existing directory is used and it contains files prior to being used as a mount point, they will be hidden after mounting. These files are not deleted and will again be visible when the filesystem is unmounted.

Only the superuser can mount and unmount filesystems.

mount

- General form for **mount**

```
mount [options] <source> \
                <directory>
```

- Can mount using: **Device Node**, **Label** or **UUID**
- Each filesystem has its own set of options. A common example would be:

```
$ sudo mount -o remount,ro /myfs
```

  which remounts a filesystem with a read-only attribute.

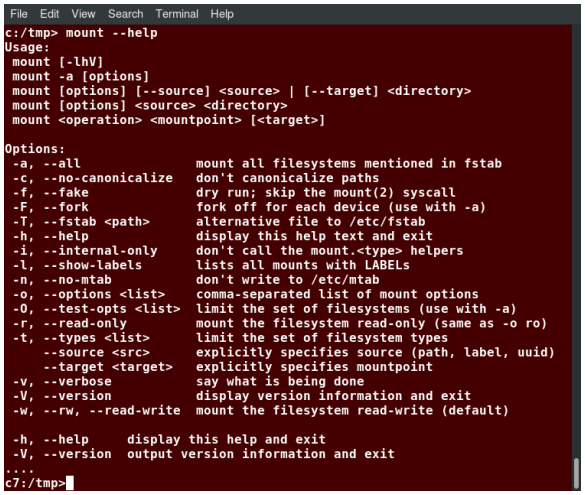


Figure 10.7: **mount**

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.

Currently Mounted Filesystems

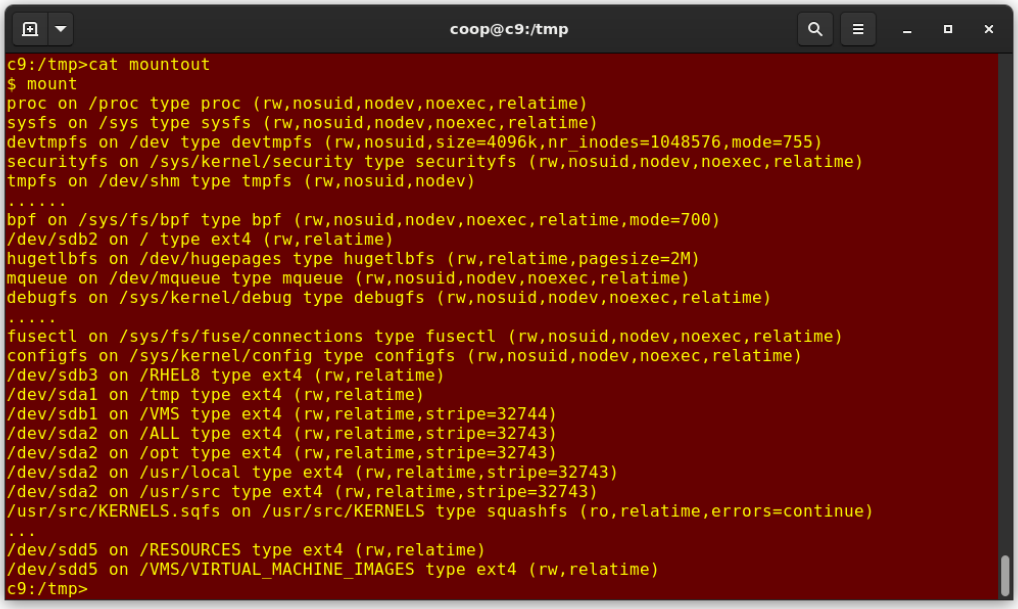


Figure 10.8: **Currently Mounted Filesystems**

Note that the **mount** run without arguments lists all currently mounted filesystems.

Many of them are “special” filesystems such as `hugetlbfs` or `debugfs` rather than ones directly created by users and mounted in the normal filesystem hard disk on storage media.

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.

10.10 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`:

/etc/fstab

```
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**.

Network file systems may be unavailable at any time due to the file system not being mounted or network issues. Systems using network based file systems must be prepared for these conditions.

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`.

10.11 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.

/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 10.9: **/etc/fstab Example**

10.12 Labs

Video Demonstration Resources

using_available_filesystems_demo.mp4
using_filesystems_demo.mp4

Exercise 10.1: 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:

```
$ 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`.

Exercise 10.2: Mounting Options

Fresh Partition or Loopback File

In this exercise we will use loopback files to get some practice with mounting filesystems with various options.

This exercise can also be performed with a physical partition, if you are familiar with the use of **fdisk** to make fresh partitions available. While we do not discuss this utility in this course, we provide an optional solution for those who have learned how to use **fdisk**.

The methods do not differ greatly.

1. Create a file full of zeros to use as a loopback file to simulate a new partition.

   Optionally, you can use **fdisk** to create a new 250 MB partition on your system, probably on `/dev/sda`.

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 10.2

Loopback File Solution

1. `$ sudo dd if=/dev/zero of=/imagefile bs=1M count=250`

2. ```
   $ sudo mkfs -t ext4         -v
   $ sudo mkfs -t ext4 -b 2048 -v /imagefile
   $ sudo mkfs -t ext4 -b 4096 -v /imagefile
   ```

   You will get warned that this is a file and not a partition, just proceed.

   Note the `-v` flag (verbose) will give the requested information; you will see that for a small partition like this the default is 1024 byte blocks.

3. ```
   $ sudo mkdir /mnt/tempdir
   $ sudo mount -o loop /imagefile /mnt/tempdir
   $ mount | grep tempdir
   ```

4. ```
   $ sudo umount /mnt/tempdir
   $ sudo mount -o ro,loop /imagefile /mnt/tempdir
   ```

   If you get an error while unmounting, make sure you are not currently in the directory.

5. `$ sudo touch /mnt/tempdir/afile`

6. `$ sudo umount /mnt/tempdir`

7. Put this line in `/etc/fstab`:

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?

Physical Partition Solution

1. We do not show the detailed steps in **fdisk**; 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** will not 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?

Chapter 11

Virtualization Overview

Learning Objectives

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

- Understand the concepts behind virtualization and how it came to be widely used.
- Understand the roles of hosts and guests.
- Discuss the difference between emulation and virtualization.
- Distinguish between the different types of hypervisors.
- Be familiar with how **Linux** distributions use and depend on **libvirt**.
- Use the **qemu** hypervisor.
- Install, use and manage **KVM**.

11.1 Introduction to Virtualization

What is Virtualization?

- An abstraction layer:
  - A **physical** resource is replaced by a **virtual** one
- Types include:
  - Operating System (e.g., **Virtual Machine, VM**)
  - Network (e.g., **Software Defined Networking, SDN**)
  - Storage (e.g., **Network Attached Storage, NAS**)
  - Application (e.g., **containers**)
- The abstraction layer may replace operating system specific **software**, rather than **hardware**

In this section of the course we will be concerned with the creation, deployment and maintenance of **Virtual Machines** (**VM**s).

These are a virtualized instance of an entire operating system, and may fulfill the role of a **server** or a **desktop**/**workstation**.

The outside world sees the VM as if it were an actual physical machine, present somewhere on the network. Applications running in the VM are generally unaware of their non-physical environment.

Other kinds of virtualization include:

- **Network:** The details of the actual physical network such as the type of hardware, routers etc. are abstracted and need not be known by software running on it and configuring it.
- **Storage:**

  Multiple network storage devices are configured to look like one big storage unit such as a disk.
- **Application:**

  An application is isolated into a stand alone format such as a **container**, which we will discuss later.

There are differences between physical and virtual machines which are not always easy to ignore. For example, performance tuning at a low level needs to be done (separately) on both the VM and the physical machine residing underneath it.

Virtualization History

- Implemented originally on mainframes decades ago:
  - Enables better hardware utilization
  - Operating Systems often progress more quickly than hardware
  - Microcode driven
  - Not particularly user friendly
- Virtualization technology migrated to PCs and workstations:
  - Initially done using **Emulation**
  - CPUs enhanced to support Virtualization led to boost in performance, easier configuration and more flexibility in VM installation and migration

Virtualization has a long proud history. From early **mainframes** to **mini-computers**, virtualization has been used for expanding limits, debugging and administration enhancements.

Today, virtualization can be found everywhere in many forms.

Each of the different forms and implementations available provide particular advantages.

11.2 Hosts and Guests

Hosts and Guests

- **Host:**
  Underlying physical operating system managing one or more virtual machines
- **Guest:**
  The **VM** which is instance of a complete operating system, running one or more applications. Sometimes called a **client**
- Guest does not care or know the specific host
- Guests can be **migrated** to another host
- Performance tuning needs to be done on host as well as guest

In most cases the guest should not care what host it is running on and can be **migrated** from one host to another, sometimes while actually running.

In fact, it is possible to convert physical machines into virtual ones, by copying the entire contents into a VM. Specialized software utilities can make this easier.

Low level performance tuning on areas such as CPU utilization, networking throughput, memory utilization is often best done on the host, as the guest may have only simulated qualities not directly useful.

Application tuning will be done mostly on the guest.

11.3 Emulation

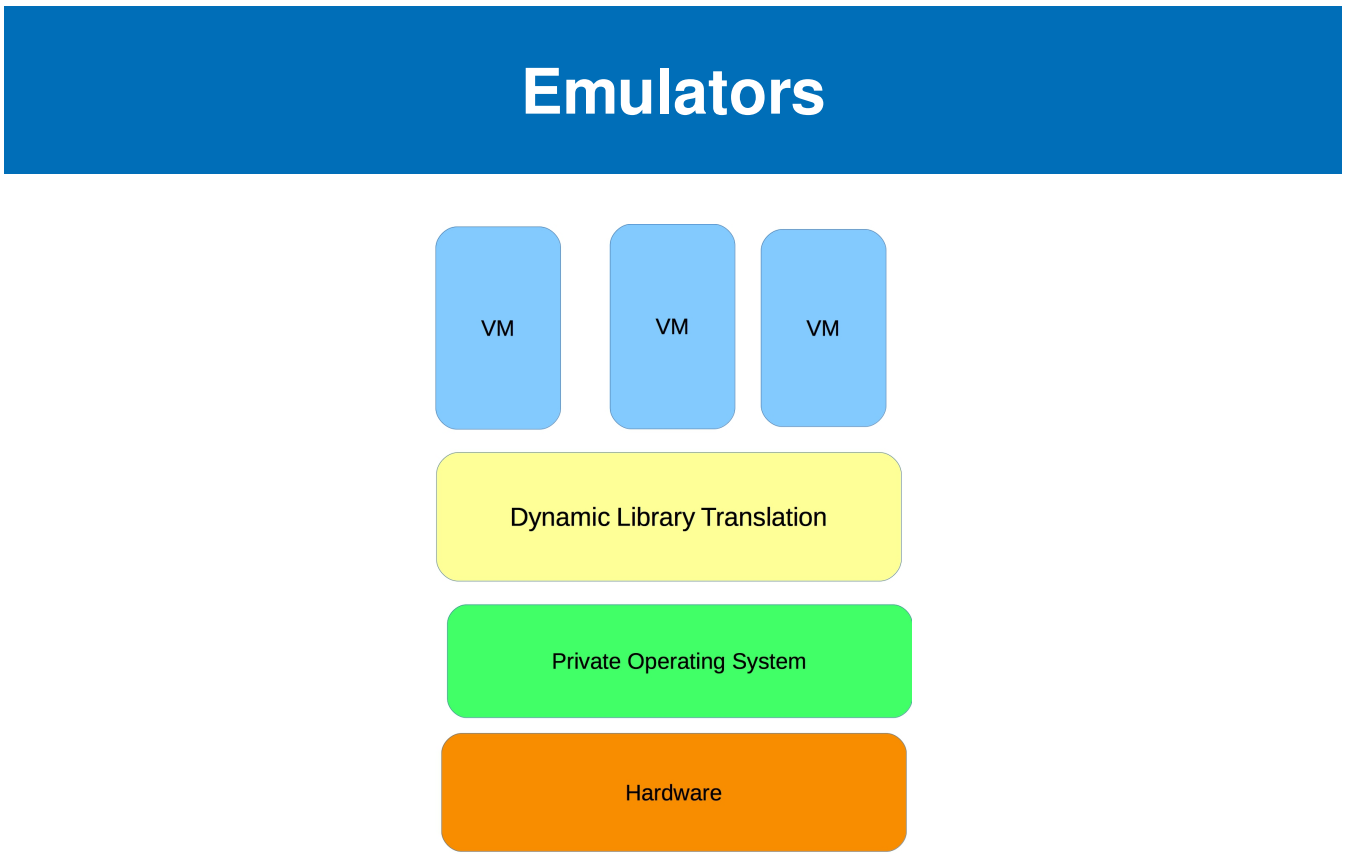


Figure 11.1: **Emulators**

The first implementations of virtualization on the PC architecture were through the use of **Emulators**. An application running on the current operating system would appear to another OS as a specific hardware environment. Emulators generally do not require special hardware to operate.

Qemu is one such emulator.

A common use case is: a new processor chip is being developed and the specifications of the hardware and instructions this new device will use are incorporated into **QEMU** System Emulation. A developer using their Intel `x86_64` based laptop with Linux on it can install the matching **QEMU** System Emulator and create a virtual machine that will function the same as the real hardware when it is released from the manufacturer. This allows the software and Operating Systems to be developed in parallel with the hardware.

For more information on **QEMU System Emulation** see:

https://www.qemu.org/docs/master/system/targets.html

Emulation vs Virtualization

- An **Emulator** runs completely in software
- Hardware constructs are replaced by software
- Useful for running virtual machines on different architectures: e.g., running a pretend **ARM** guest machine on an **x86** host
- Often used for developing operating system for a new CPU even before hardware is available.
- Performance is relatively slow

The usage of **emulation** and **virtualization** are quite different.

emulation

- Speed is not one of the prime considerations, with **emulation**, the main focus accurately emulating the environment.
- All of the physical components are represented by software. These hardware devices are very low level and usually under direct control of the Operating System. An example is the General Purpose Input/Output (GPIO) component that can be used to detect a switch being on or off. This can be emulated in software to test a program that will be using the switch.

Virtualization

- Performance is a very big requirement in the computing environment. An application or service may be required to expand or contract easily and seamlessly. Adding and provisioning systems and applications can be time consuming. Using Virtual resources, typically cloud based, applications and services can easily be changed.
- Virtualization excels in its ability to share the physical environment allowing for maximum utilization of resources.

11.4 Hypervisors

Types of Virtualization Hypervisors

- The layer that runs underneath the virtual machines and manages them
- Also called **Virtual Machine Monitor** (**VMM**)
- Enables multiple VMs to share hardware resources transparently
- Two basic types:
  - **Hardware Virtualization**
    Guest system is unaware it is running as a VM. Also called **Full Virtualization**
  - **Paravirtualization**
    Guest system knows it is running as a VM and has been modified to run better
- Most modern hardware has hardware virtualization abilities (must be turned on in BIOS)

The host system, besides functioning normally with respect to software that it runs, also acts as the **hypervisor** that initiates, terminates and manages guests. There are two basic methods of virtualization:

- **Hardware virtualization:** the guest system runs without being aware it is running as a virtualized guest and does not require modification to be run in this fashion.
- **Para-virtualization:**: the guest system is aware it is running in a virtualized environment and has been modified specifically to work with it.

Widely-used CPUs from **Intel** and **AMD** incorporate virtualization extensions to the **x86** architecture that allow the hypervisor to run fully virtualized (i.e., unmodified) guest operating systems with only minor performance penalties.

The **Intel** extension (**I**ntel **V**irtualization **T**echnology), usually abbreviated as **VT, IVT, VT-32** or **VT-64**, is also known under the development code name **Vanderpool**. It has been available since the spring of 2005. The **AMD** extension is usually called **AMD-V** and is still sometimes referred to by the development code name **Pacifica**. For a detailed explanation and comparison of how these two extensions work see https://lwn.net/Articles/182080/.

You can check directly if your CPU supports hardware virtualization extensions by looking at `/proc/cpuinfo`; if you have an **IVT**-capable chip you will see `vmx` in the flags field, and if you have an **AMD-V** capable chip, you will see `svm` in the same field. You may also have to ensure the virtualization capability is turned on in your **CMOS**.

While the choice of operating systems tends to be more limited for para-virtualized guests, originally they tended to run more efficiently than fully virtualized guests. Advances in virtualization techniques have narrowed or eliminated such advantages, and the wider availability of the hardware support needed for full virtualization has made para-virtualization less advantageous and less popular.

External and in-Kernel Hypervisors

- The hypervisor can be external to the host operating system kernel: **VMWare** or **Xen**
- The hypervisor can be internal to the host operating system kernel: **KVM**
- In this course we will concentrate on **KVM** as it is all open source and requires no external third party hypervisor program.

An **external hypervisor** is a software application that provides virtualization capabilities through the use of a program. **VMWare** is one such application. When **VMWare Player** runs it interprets the configuration information for the virtual client and provides or acquires the resources to run the Operating System as a client. It is considered an **external hypervisor** because the player program `plugs-in` to the existing Operating system. **External hypervisors** can be ported to many different environments such as **Linux**, **macOS** or **Windows**.

An **internal** hypervisor is merged with the Operating system Kernel and works very closely with the Kernel. An **internal hypervisor** is entwined into a specific Operating System and portability of the hypervisor is not possible. At this time the **Linux Kernel Virtual Machine (KVM)** is the only common **Hypervisor/Kernel** pair.

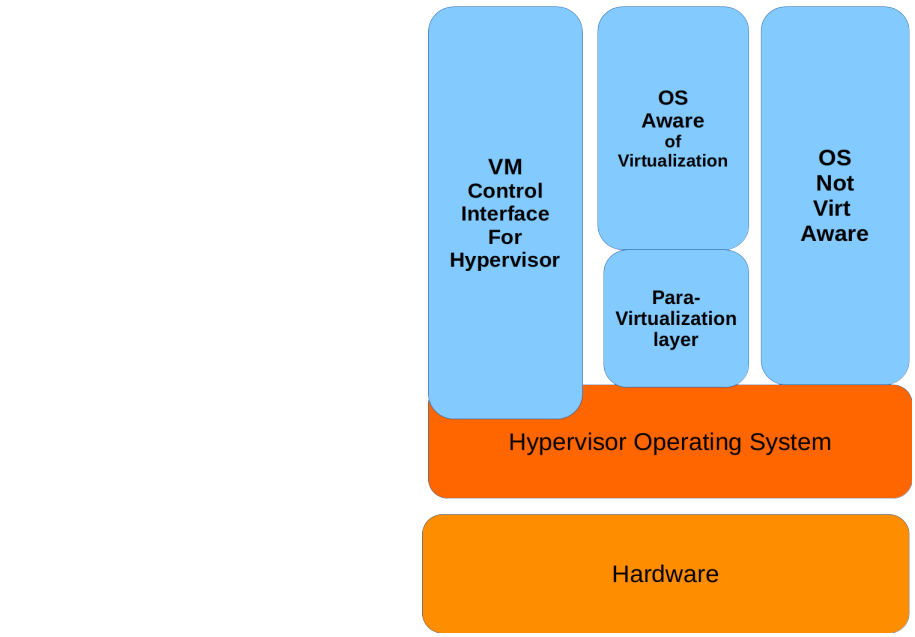


Figure 11.2: **Dedicated Hypervisor**

Going past Emulation, merging of the hypervisor program into a specially designed light weight kernel was the next step in Virtualization deployment.

VMware ESX (and related friends) or **Xen** are examples of a hypervisor embedded into an operating system.

In the case of **Xen** the **hypervisor** is embedded into a very small limited **kernel**. The **Xen** hypervisor and kernel require an external operating system to perform the necessary configuration of the hypervisor. The initialization sequence for a **Xen** based environment is something like:

- Boot the system to run the **Xen** hypervisor/kernel
- Start an operating system that can interface with the **Xen** hypervisor/kernel and allow that operating system to configure the hypervisor and any subsequent Virtual Machines to be run. This initial Virtual Machine is known as **Domain0** and should only be used for configuration and control of the VM's

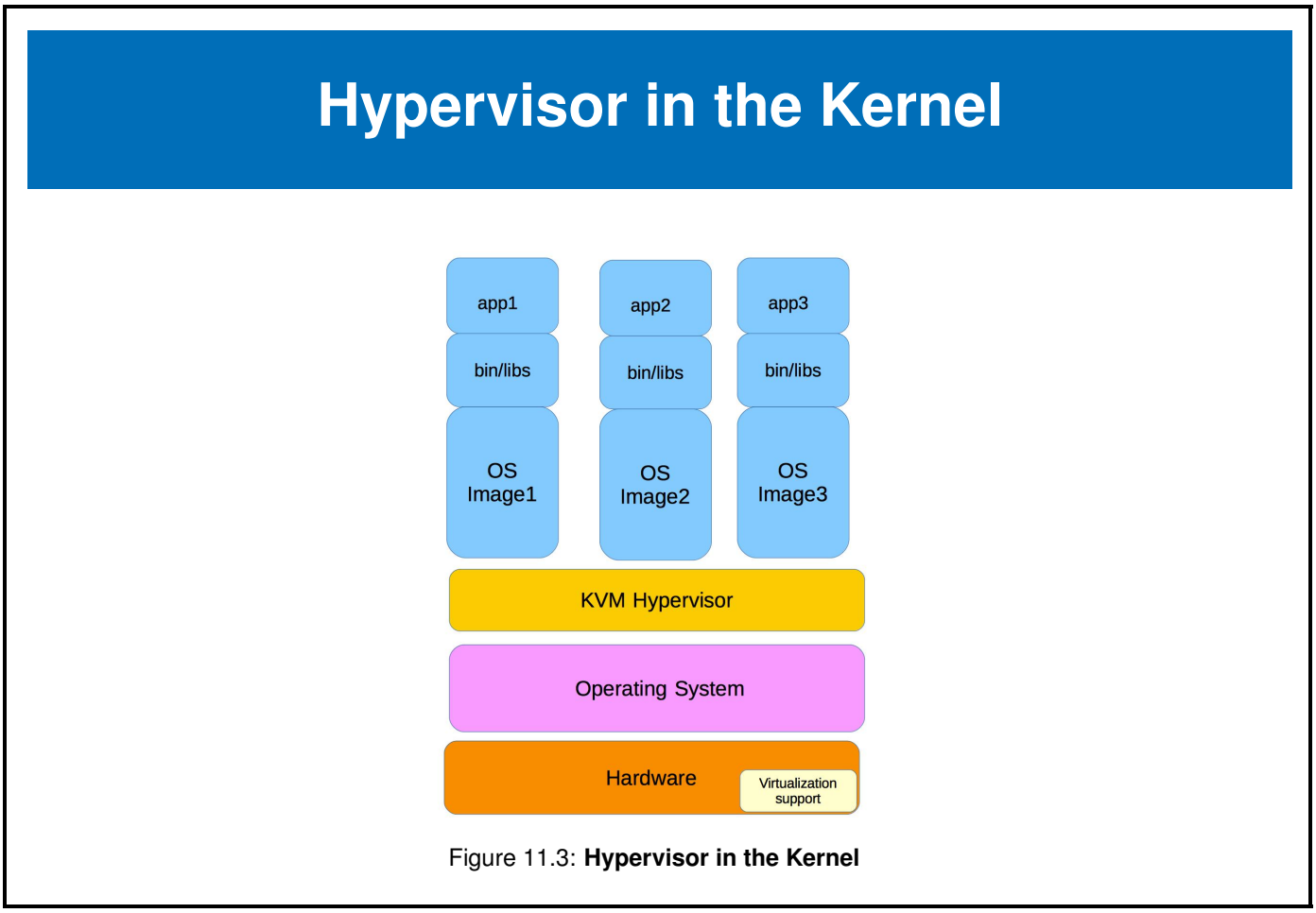


Figure 11.3: **Hypervisor in the Kernel**

The **KVM** project added hypervisor capabilities into the **Linux** kernel. This leveraged the facilities of the kernel and enabled the kernel to be a hypervisor.

As we have discussed, specific CPU chip functions and facilities were required and deployed for this type of virtualization.

11.5 libvirt

libvirt

- Toolkit to interact with virtualization technologies
- Common API interface for many hypervisors
  - **KVM/QEMU**
  - **LXC**
  - **VirtualBox**
  - . . . and many more
- Provides management for
  - virtual machines
  - virtual networks
  - storage
- Available on all enterprise **Linux** distributions

Many application programs interface with **libvirt**, some of the most common are **virt-manager**, **virt-viewer**,**virt-install**,**virsh**. The complete list of currently supported hypervisors at https://www.libvirt.org:

- **QEMU/KVM**
- **Xen**
- **Oracle VirtualBox**
- **VMware ESX**
- **VMware Workstation/Player**
- **Microsoft Hyper-V**
- **IBM PowerVM (phyp)**
- **OpenVZ**
- **UML** (User Mode Linux)
- **LXC** (Linux Containers)
- **Virtuozzo**
- **Bhyve** (The BSD Hypervisor)
- **Test** (Used for testing)

Using the **libvirt** API, the user interface can easily be changed without needing updates to the API that does the actual work. New user interfaces have been created to work with the **libvirt** API. As an example, there is a web based plugin for the **co-pilot** system administration tool that uses the **libvirt** API.

Programs Using libvirt

- Many utilities use **libvirt**:

```
student@ubuntu: ~
student@ubuntu:~$ ls -lF /usr/bin/virt*
-rwxr-xr-x 1 root root  63840 Aug  9 08:50 /usr/bin/virt-admin*
-rwxr-xr-x 1 root root     57 Dec 14  2016 /usr/bin/virt-clone*
-rwxr-xr-x 1 root root     59 Dec 14  2016 /usr/bin/virt-convert*
-rwxr-xr-x 1 root root  51440 May 10 18:25 /usr/bin/virtfs-proxy-helper*
-rwxr-xr-x 1 root root  18480 Aug  9 08:50 /usr/bin/virt-host-validate*
-rwxr-xr-x 1 root root     59 Dec 14  2016 /usr/bin/virt-install*
-rwxr-xr-x 1 root root 908632 Aug  9 08:50 /usr/bin/virt-login-shell*
-rwxr-xr-x 1 root root     59 Dec 14  2016 /usr/bin/virt-manager*
-rwxr-xr-x 1 root root  11191 Aug  9 08:49 /usr/bin/virt-pki-validate*
-rwxr-xr-x 1 root root     55 Dec 14  2016 /usr/bin/virt-xml*
-rwxr-xr-x 1 root root   4700 Aug  9 08:49 /usr/bin/virt-xml-validate*
student@ubuntu:~$ \
```

Figure 11.4: **libvirt-based Utilities**

- The exact list will depend on your **Linux distribution**.
- A full list can be found at: https://www.libvirt.org/apps.html

In this course we will do our work through the use of the robust **GUI**, **virt-manager** rather than make much use of command line utilities, which lead to more flexibility, as well as use on non-graphical servers.

11.6 QEMU

What is QEMU?

- **Q**uick **EMU**lator
- Three levels:
  1. **libvirt**: Common Library and API
  2. **QEMU**: Emulator
  3. **KVM**: Accelerator
- Hypervisor doing hardware emulation
- Host and guest can be different architectures (CPUs)
- Combine with accelerator (e.g., **KVM**) to speed up to native speeds
- Useful for development for new processing chips

QEMU stands for **Q**uick **EMU**lator. It was originally written by Fabrice Bellard in 2002. (Bellard is also known for feats such as holding, at one point, the world record for calculating π, reaching 2.7 trillion digits.)

QEMU is a **hypervisor** that performs hardware **emulation**, or **virtualization**. It emulates CPUs by dynamically translating binary instructions between the host architecture and the emulated one.

The host and emulated architectures may be different or the same. There are numerous choices for both host and guest operating systems.

QEMU can also be used to emulate just particular applications, not entire operating systems.

By itself, **QEMU** is much slower than the host machine. But it can be used together with **KVM** (**K**ernel **V**irtual **M**achine) to reach speeds close to those of the native host.

Guest operating systems do not require rewriting to run under **QEMU**.

QEMU can save, pause, and restore a virtual machine at any time.

QEMU is free software licensed under the **GPL**.

QEMU has the capability of supporting many architectures including:

IA-32 (i386), **x86-64**, **MIPS**, **SPARC**, **ARM**, **SH4**, **PowerPC**, **CRIS**, **MicroBlaze**, **etc.**

The cross-compilation abilities of **QEMU** make it extremely useful when doing development for embedded processors.

In fact, **QEMU** has often been used to develop for processors which have either not yet been physically produced, or released to market.

Third Party Hypervisor Integration

- **QEMU** is slow but can reach almost native speed with an **accelerator** such as:
  - **KVM**
  - **Xen**
  - **Oracle Virtual Box**
- **KVM** is very tightly integrated with **QEMU** and arose from the same projects.

Used by itself, **QEMU** is relatively slow. However, it can be integrated with third party hypervisors, and then reach near native speeds. Note some of these systems are very close cousins of **QEMU**; others are more remote. To mention a few main ones:

- **KVM** offers particularly tight integration with **QEMU**. When host and target are the same architecture, full acceleration and high speed are delivered. **KVM** is native to **Linux**. We will discuss this in detail.
- **Xen**, also native to **Linux**, can run in hardware virtualization mode if the architecture offers it, as does **x86** and some **ARM** variants.
- **Oracle Virtual Box** can use **qcow2** formatted images, and has a very close relationship with **qemu**.

In this course we recommend using (and will use in labs) **virt-manager** to configure and run virtual machines. We will also give instructions for how to run them using **qemu** command line utilities.

Image Formats

- Two main Formats:
  1. **cmd**: Simplest
  2. **qcow2**: **C**opy **O**n **W**rite 2
- Can support many other formats
- Can convert between formats:

```
$ qemu-img convert -O vmdk myvm.qcow2 myvm.vmdk
```

QEMU supports many formats for disk image files. However, only two are used primarily, with the rest being available for historical reasons and for conversion utilities. These two main formats are:

- **cmd** (default)
  This is the simplest and easiest to export to other non-**QEMU** emulators. Empty sectors do not take up space.
- **qcow2**
  COW stands for **C**opy **O**n **W**rite. There are many options. See **man qemu-img**.

To get a list of supported formats:

```
$ qemu-img --help | grep formats:
```

```
Supported formats: blkdebug blklogwrites blkreplay blkverify copy-on-read file ftp ftps gluster
↪ host_cdrom host_device http https iscsi iser luks nbd null-aio null-co nvme qcow2 quorum raw rbd ssh
↪ throttle vhdx vmdk vpc
```

Note in particular:

- **vdi**: Used by **Oracle Virtual Box**
- **vmdk**: Used by **VMware**

Note that **qemu-img** can be used to translate between formats, .e.g.:

```
$ qemu-img convert -O vmdk myvm.qcow2 myvm.vmdk
```

using default options. See the **man** page for full possibilities.

11.7 KVM

KVM and Linux

- **K**ernel-based **V**irtual **M**achine
- Requires hardware virtualization extensions
- Uses kernel for resources in a co-processing relationship
- Kernel emulates guest instructions
- Avi Kivity submitted, quickly merged in 2007
- Has paravirtualization extensions and support

KVM uses the **Linux** kernel for computing resources including memory management, scheduling, synchronization, and more. When running a virtual machine, **KVM** engages in a co-processing relationship with the **Linux** kernel.

In this format, **KVM** runs the virtual machine monitor within one or more of the CPUs, using `VMX` or `SVM` instructions. At the same time the **Linux** kernel is executing on the other CPUs.

The virtual machine monitor runs the guest, which is running at full hardware speed, until it executes an instruction that causes the virtual machine monitor to take over.

At this point the virtual machine monitor can use any **Linux** resource to emulate a guest instruction, restart the guest at the last instruction, or do something else.

By loading the **KVM** modules and starting a guest, you turn **Linux** into a hypervisor. The **Linux** personality is still there, but you also have the hardware virtual machine monitor. You can control the Virtual Machine using standard **Linux** resource and process control tools, such as **cgroups**, **nice**, **numactl**, and so on.

KVM first appeared (pre-merge) as part of a Windows Virtual Desktop product. At the time of its upstream merge in 2007, **KVM** required recent **x86_64** processors. On **x86_64** platforms, **KVM** is primarily (but not always) a driver for the processor's virtualization subsystem.

KVM appeared as a trio of **Linux** kernel modules in 2007. When paired with a modified version of **QEMU** (also provided at the same time) it created a hypervisor that used the **Linux** kernel for most of its run-time services.

Shortly after Avi Kivity, the author of **KVM**, submitted the source code to the **Linux** development community, Linus merged **KVM** into his **Linux** tree. This was surprising to a lot of folks.

Managing KVM

- Both Command Line and Graphical Interfaces
- Command line tools include:
  - **virt-***
  - **qemu-***
- Graphical interfaces include:
  - **virt-manager**
  - **kimchi**
  - **OpenStack**
  - **oVirt**
- We will concentrate on **virt-manager**:
  - Most stable GUI across distributions and versions

There are many low level commands for creating, converting, manipulating, deploying and maintaining virtual machine images.

As you develop more expertise you will become practiced in using them. But for all basic operations, **virt-manager** will suffice and that is what we will use.

11.8 Labs

Exercise 11.1: Making Sure KVM is Properly Set up

Very Important

- The following labs are best run on a physical machine running **Linux** natively.
- It may be possible to run them within a Virtual Machine running under a hypervisor, such as **VMWare** or **Virtual Box**, or even **KVM**. However, this requires **nested virtualization** to be running properly.
- Whether or not this works depends on the particular hypervisor used, the underlying host operating system (i.e., **Windows**, **Mac OS** or **Linux**) as well as the particular variant, such as which **Linux** or **Windows** version as well as the particular kernel.
- Furthermore it also depends on your particular hardware. For example, we have found nested virtualization working with **VMWare** on various **x86_64** machines but with **Oracle Virtual Box** only on some.
- If this works, performance will be poor as compared to running on native hardware, but that is not important for the simple demonstrative exercises we will do.
- Your mileage will vary! If it does not work we cannot be responsible for helping you trying to get it rolling.

1. First check that you have hardware virtualization available and enabled:

   ```
   $ grep -e vmx -e svm /proc/cpuinfo
   ```

   where `vmx` is for **INTEL** CPUs and `svm` for **AMD**. If you do not see either one of these:

   - If you are on a physical machine, maybe you can fix this. Reboot your machine and see if you can turn on virtualization in the BIOS settings. Note that some IT personnel may make this impossible for "security" reasons, so try to get that policy changed.
   - You are on a virtual machine running under a hypervisor, and you do not have nested virtualization operable.

2. If for either of these reasons you do not have hardware virtualization, you **may** be able to run **virt-manager**, but with weak performance.

3. You need all relevant packages installed on your system. One can work hard to construct an exact list. However, exact names and requirements change with time, and most enterprise distributions ship with all (or almost all) of the software you need.

4. The easiest and best procedure is to run the script we have already supplied to you:

   ```
   $ ./ready-for.sh --install LFS303
   ```

   where we have done the hard work.

   Alternatively, on **RPM** systems you can do some overkill with:

   ```
   $ sudo dnf|zypper install kvm* qemu* libvirt*
   ```

   It is not a large amount of storage space to do it this way.

   On **Debian** package based systems including **Ubuntu** you will have to do the equivalent with your favorite package installing procedure.

Very Important

- Do not run **libvirtd** at the same time as another hypervisor as dire consequences are likely to arise. This can easily include crashing your system and doing damage to any virtual machines being used.

- We recommend both **stopping** and **disabling** your other hypervisor as in:

```
$ sudo systemctl stop vmware
$ sudo systemctl disable vmware
```

or

```
$ sudo systemctl stop vboxdrv
$ sudo systemctl disable vboxdrv
```

Exercise 11.2: Using virt-manager with KVM to Install a Virtual Machine and Run it

In this exercise we will use pre-built **iso** images built by **TinyCoreLinux** (http://www.tinycorelinux.net/downloads.html) because they are cooked up very nicely and are quite small.

If you would like, you can substitute any installation **iso** image for another **Linux** distribution, such as **Debian**, **CentOS**, **Ubuntu**, **Fedora**, **openSUSE** etc. The basic steps will be identical and only differ when you get to the installation phase for building your new VM; which is no different than building any fresh installation on an actual physical machine.

We will give step-by-step instructions with screen capture images; If you feel confident, please try to just launch **virt-manager** and see if you can work your way through the necessary steps, as the **GUI** is reasonably clearly constructed.

1. Copy the iso files to the libvirt default directory.

Very Important

There are default directories for the **libvirt** components and **cockpit** and other tools know the defaults. It is best to use the defaults as **libvirt** and friends expect resources in the default directories. Use the directory `/var/lib/libvirt/images` for the installation iso files.

Copy the installation iso files from their current location to the default directory.

```
$ sudo cp /home/student/Downloads/CorePlus-current.iso /var/lib/libvirt/images/
```

2. Make sure **libvirtd** is running and start **cockpit** by typing:

```
$ sudo systemctl start cockpit
$ firefox http://localhost:9090
```

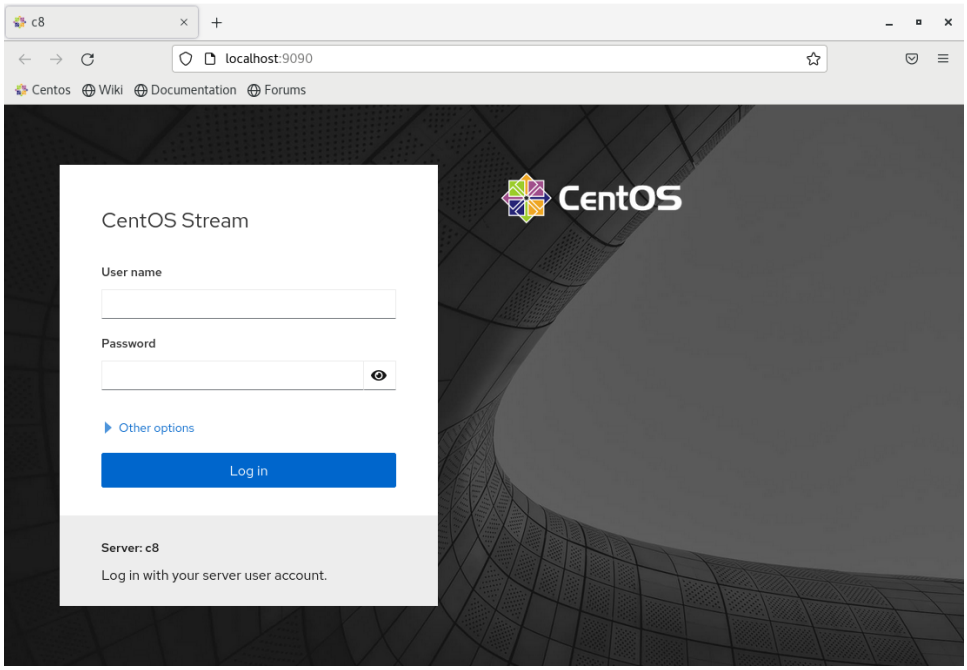


Figure 11.5: **Starting cockpit**

3. **Verify** and if **required**, **enable Administrative Access**

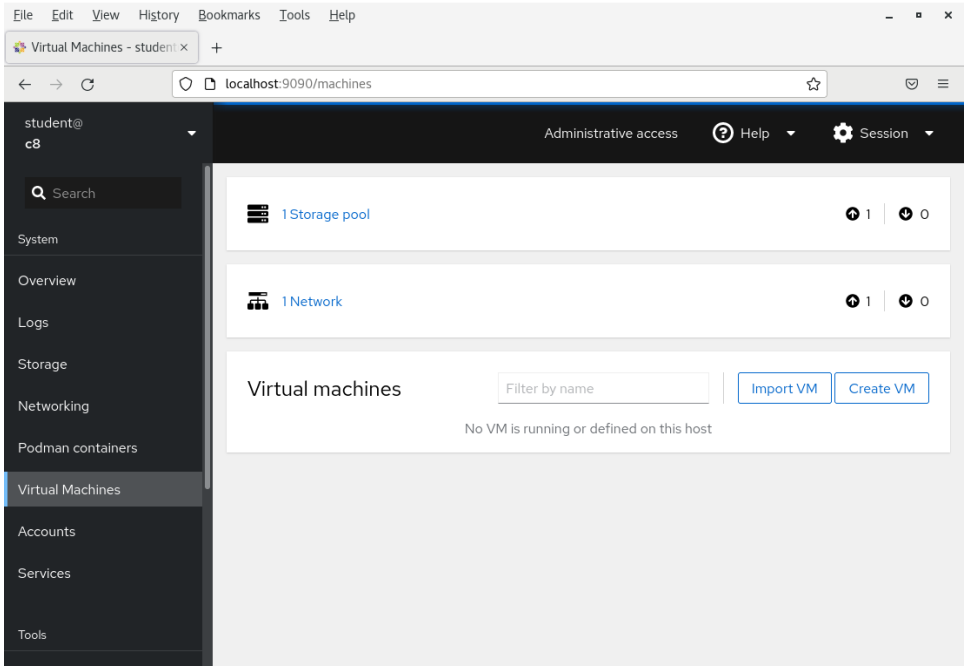


Figure 11.6: **Confirm Administrative Access is enabled**

4. Click on `Virtual Machines` :

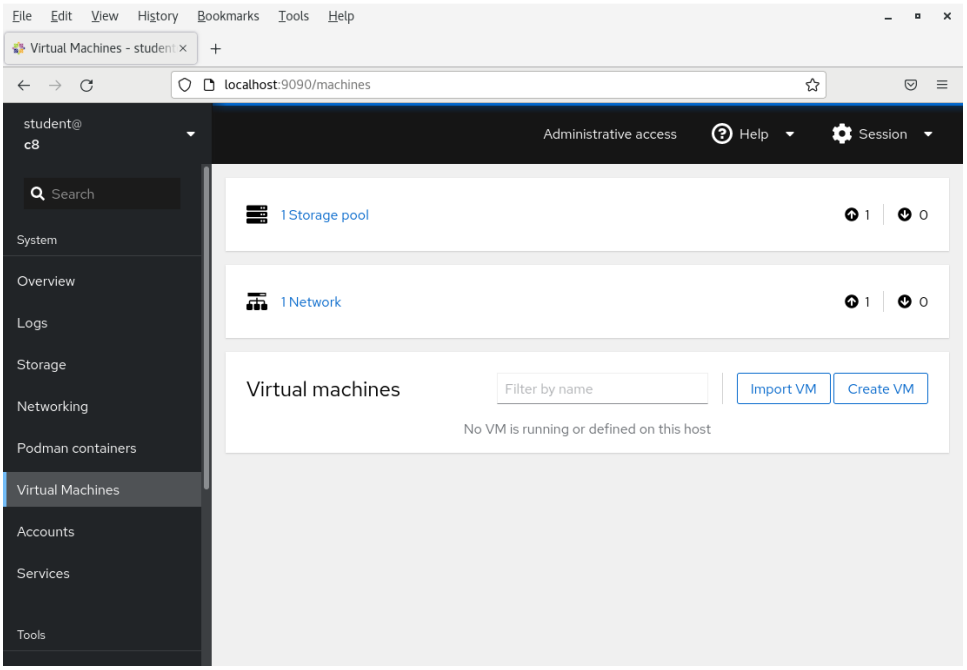


Figure 11.7: **Virtual Machine summary**

5. Click on `Create VM`, this displays the VM configuration form for **cockpit**.

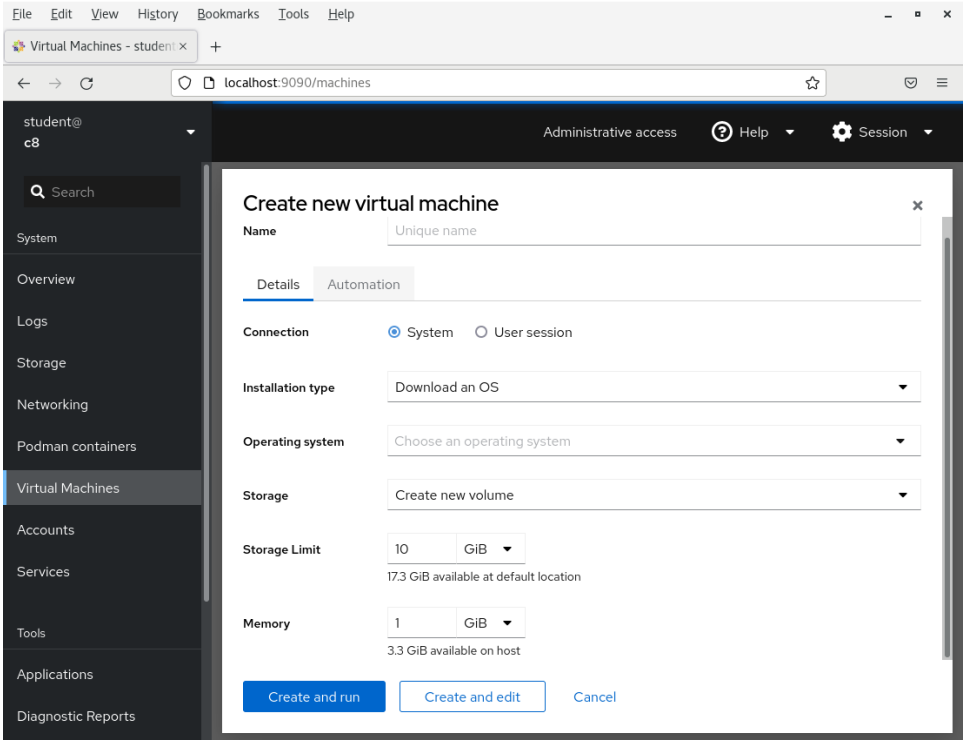


Figure 11.8: **Virtual Machine creation form**

6. We have included three different **iso** install images from **TinyCoreLinux** in the `RESOURCES/s_11` directory:

   `Core-current.iso`
   `CorePlus-current.iso`
   `TinyCore-current.iso`

Copy the **iso** files to: `/var/lib/libvirt/images/` directory.

(You can check and see if there are newer versions upstream at http://www.tinycorelinux.net but these should be fine.)

CorePlus-current is largest and robust and we will use this as it will install with full graphics. The others are considerably quicker to use, however.

Complete the VM creation form as shown, then click the `Create and run` button:

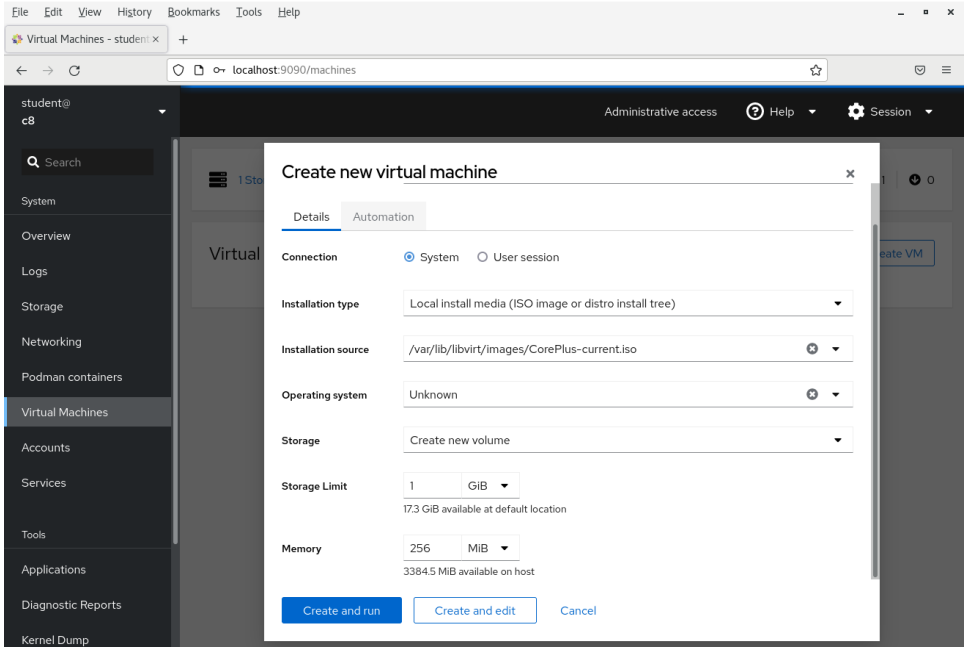


Figure 11.9: **Completed VM creation form in cockpit**

7. The boot list with countdown timer will be shown. Note there are several options for booting, we will select the item at the top of the list. Press enter to begin the initial boot.

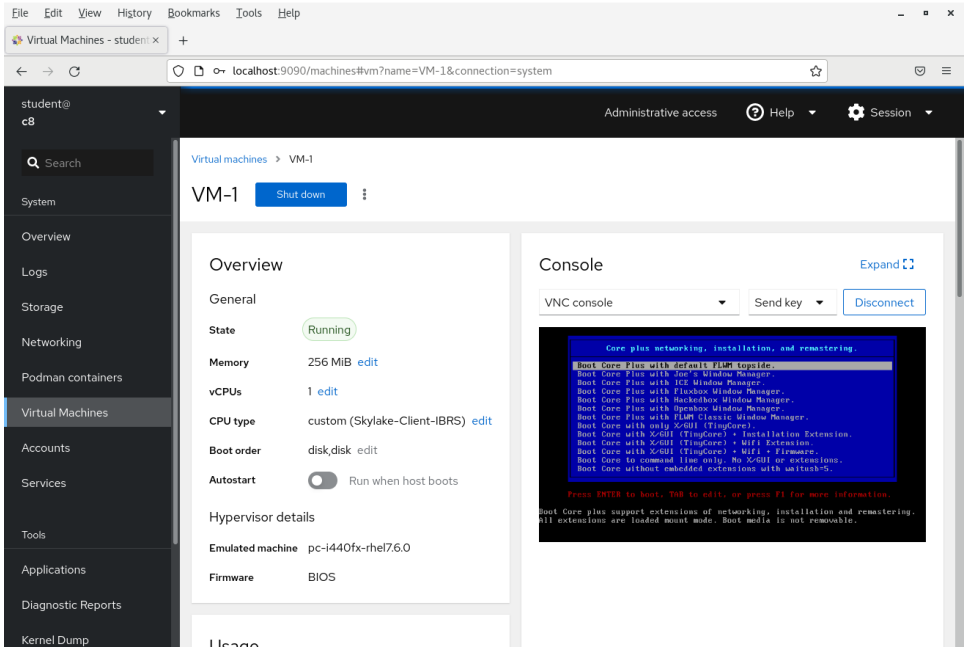


Figure 11.10: **Boot selections screen**

8. This will take a while and eventually you will see the following screen:

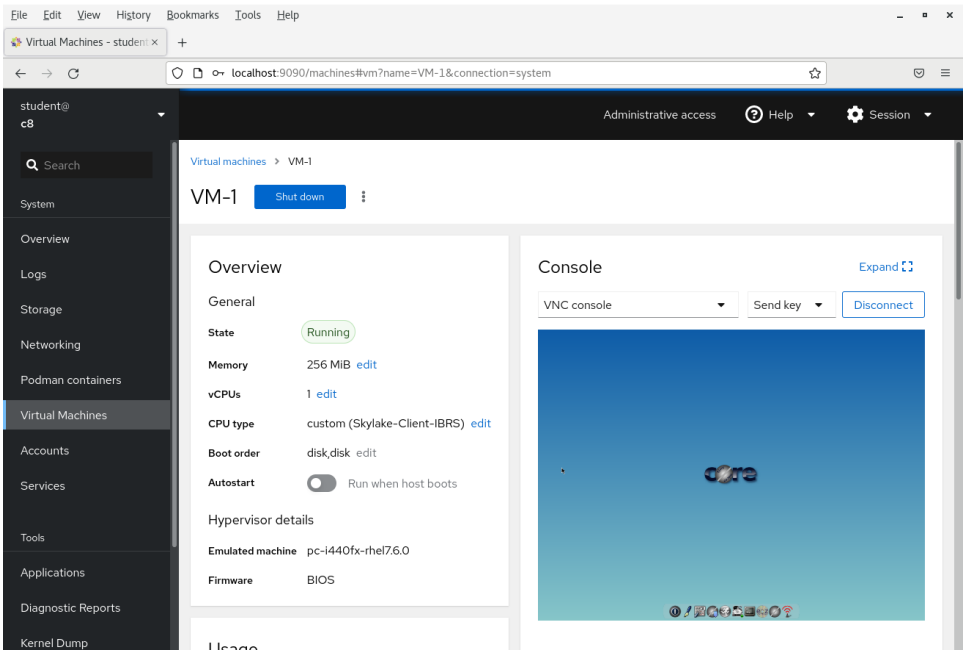


Figure 11.11: **First TinyCoreLinux Screen**

The console has an `[expand]` button that increases the screen size by hiding some of the details. The increased size makes it easier to mouse over the icons at the bottom of the screen.

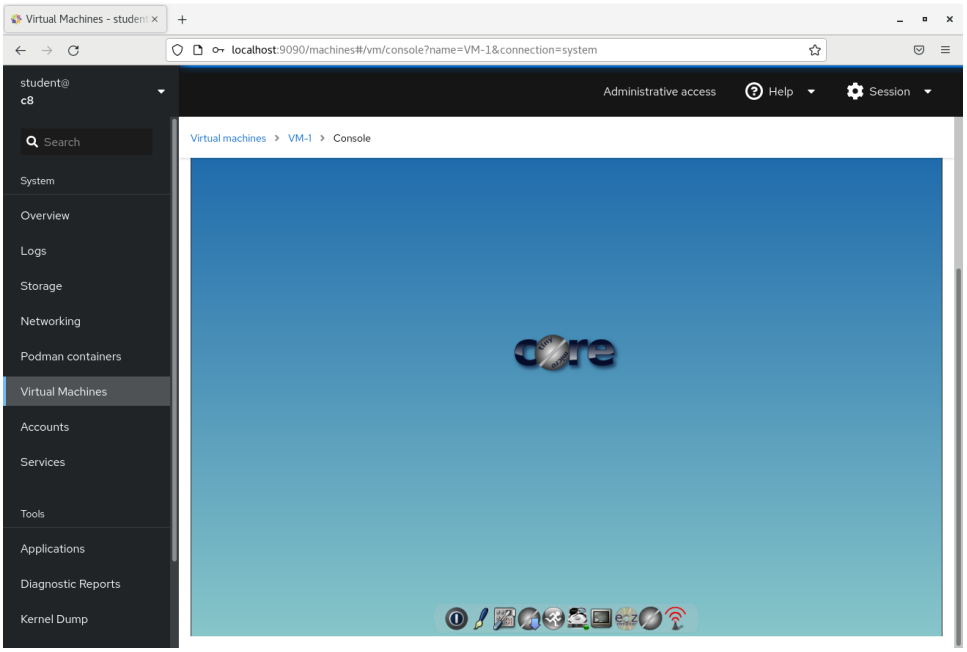


Figure 11.12: **First TinyCoreLinux Screen Expanded**

9. The **tc-install** command is the second icon from the right, looks like a globe. Click the tc-install icon to continue the installation.

10. The tc-install program will prompt for disk configuration information. Select `Whole disk` and `sda` and click the forward arrow:

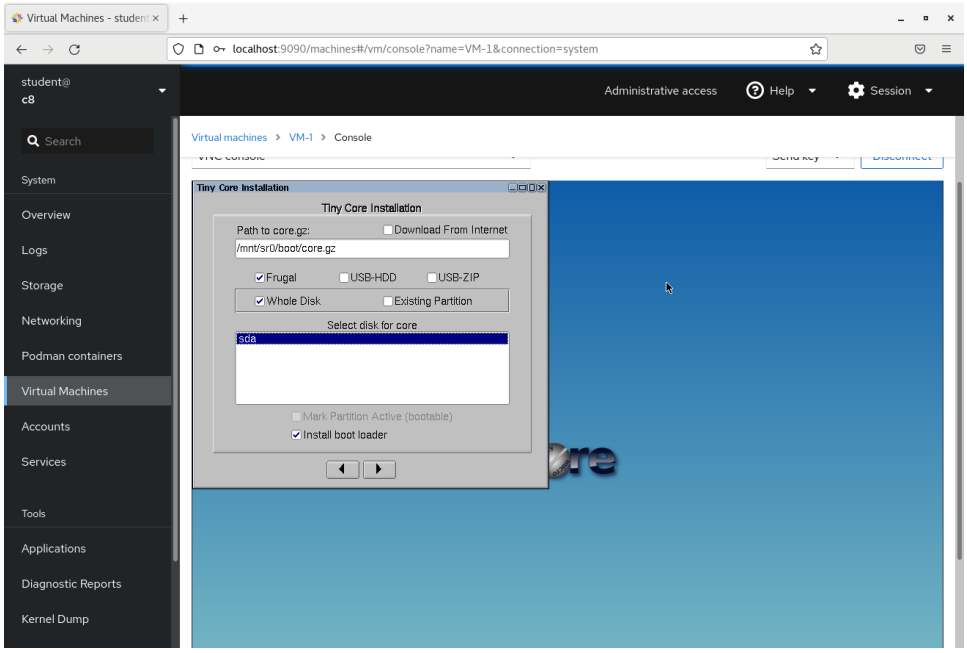


Figure 11.13: **Selecting Disk in VM**

11. Things will crank for a while and each step will be reflected in the output window.

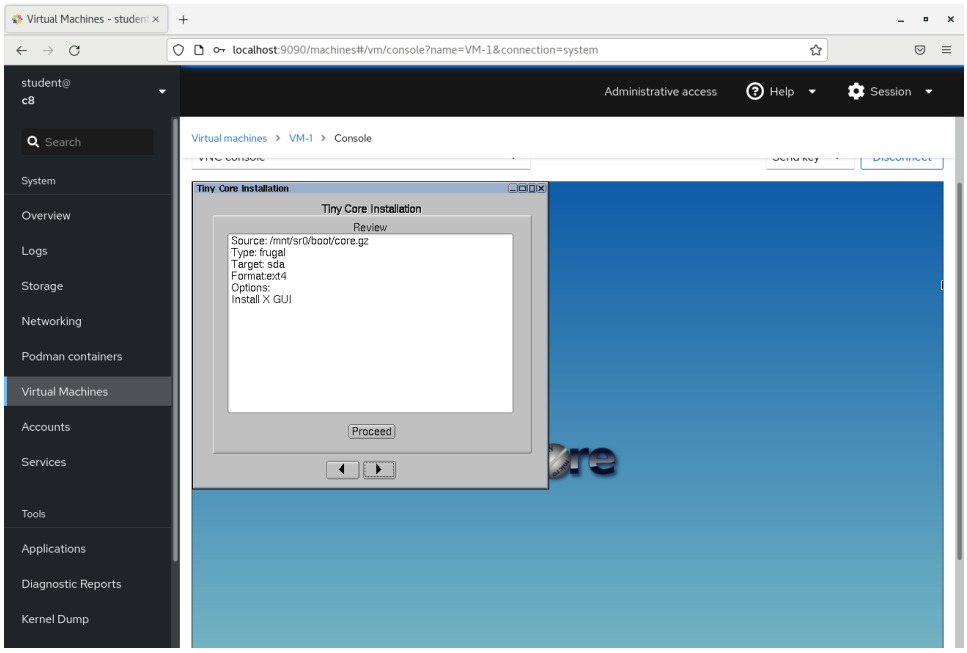


Figure 11.14: **Installation in progress**

12. When completed the tc-install command will have printed a message indicating it is completed.

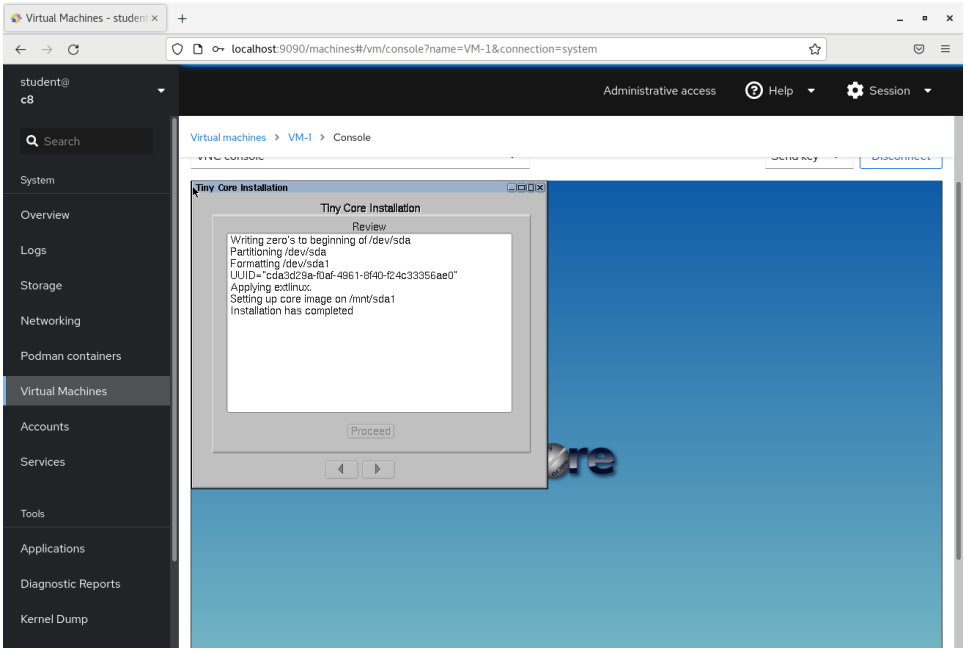


Figure 11.15: **Installation complete**

13. There will be a ”dots” menu on the Virtual Machine beside our VM’s name, in there one can reboot, stop or destroy the VM.

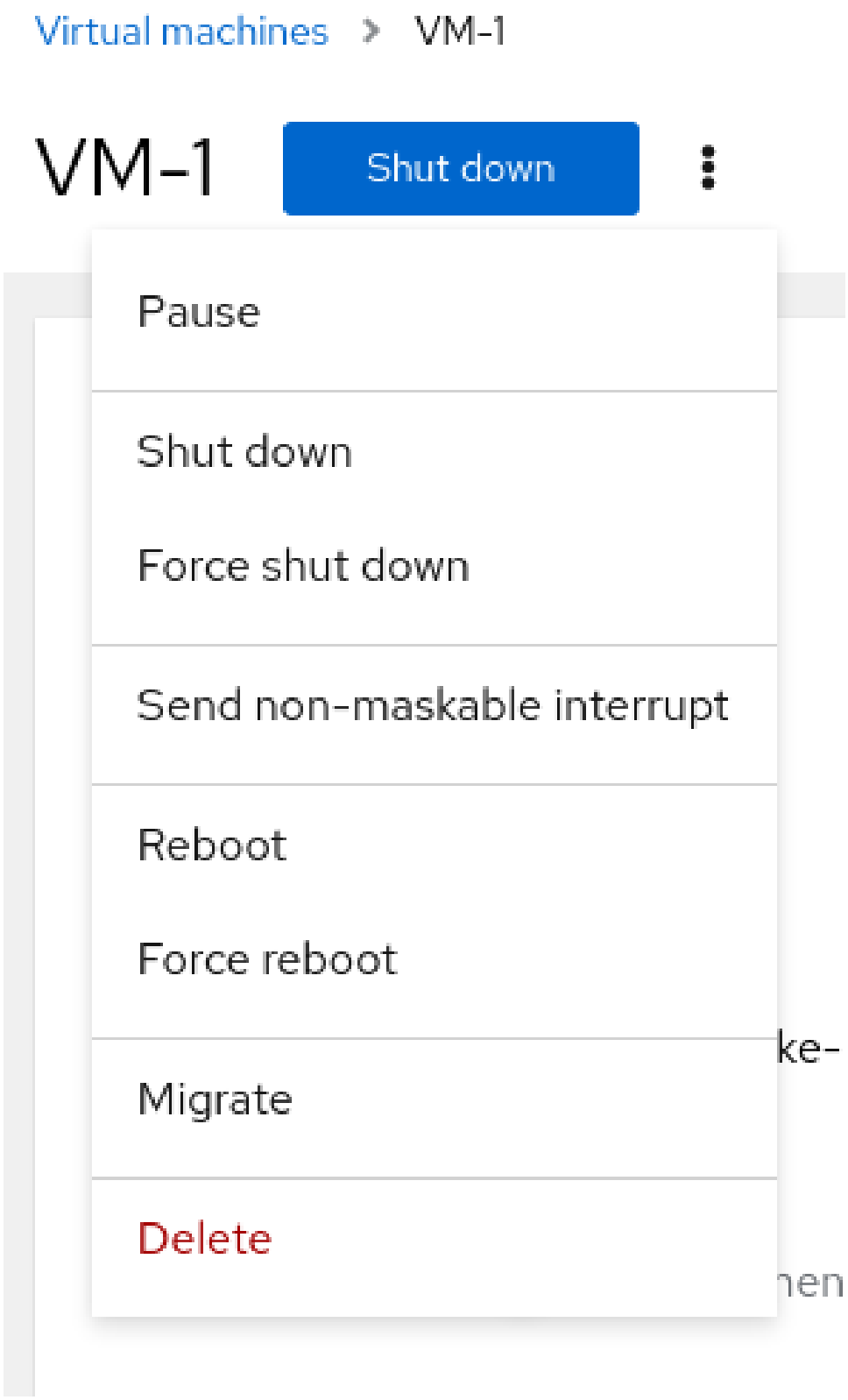


Figure 11.16: **Virtual Machine control menu**

14. On the next boot, the VM will have its boot disk switched from the iso image to the virtual hard disk sda1.

Exercise 11.3: Extra Credit: Doing from the command line

From the command line:

```
$ sudo qemu-img create -f qcow2 /var/lib/libvirtd/myimg.qcow2 24M
$ sudo qemu-system-x86_64 -hda /var/lib/libvirtd/myimg.qcow2  \
      -cdrom /teaching/LFCW/RESOURCES/LFS303/CorePlus-current.iso  -usbdevice tablet
```

Building QEMU from Source

RHEL/CentOS 8 does not ship with **qemu-system-*** binaries. It expects you to use the **kvm-qemu** software we have used in the previous exercises.

On these distributions, or on any system which does not provide these programs you can always compile from source; you may want to do this for various reasons even if your distribution provides the binary in order to get the latest version.

Note that this requires about 3 GB of storage during the build phase and takes quite a bit of time to accomplish. You may also be missing a number of development software packages and have to do some work to get things working. (These might include **ninja*** and **libpulseaudio-devel**.)

The procedure for doing this is:

1. Get the QEMU source code:

   ```
   $ git clone https://git.qemu.org/git/qemu.git
   $ cd qemu
   $ git submodule init
   $ git submodule update --recursive
   ```

2. Configure QEMU to only build the `x86_64` version of `qemu-system`; building only the architecture we plan to use.

   ```
   $ ./configure --target-list=x86_64-softmmu
   ```

3. Build QEMU as a non-root user.

   ```
   $ make -j $(nproc)
   ```

4. Install QEMU into `/usr/local/bin/` as root.

   ```
   $ sudo make install
   ```

Chapter 12

Containers Overview

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.
- Be familiar with **Docker** and know the steps necessary to use it properly.
- Know how **podman** has been replacing **Docker**.
- Use the major commands associated with containers, for both **Docker** and **podman**.

12.1 Containers

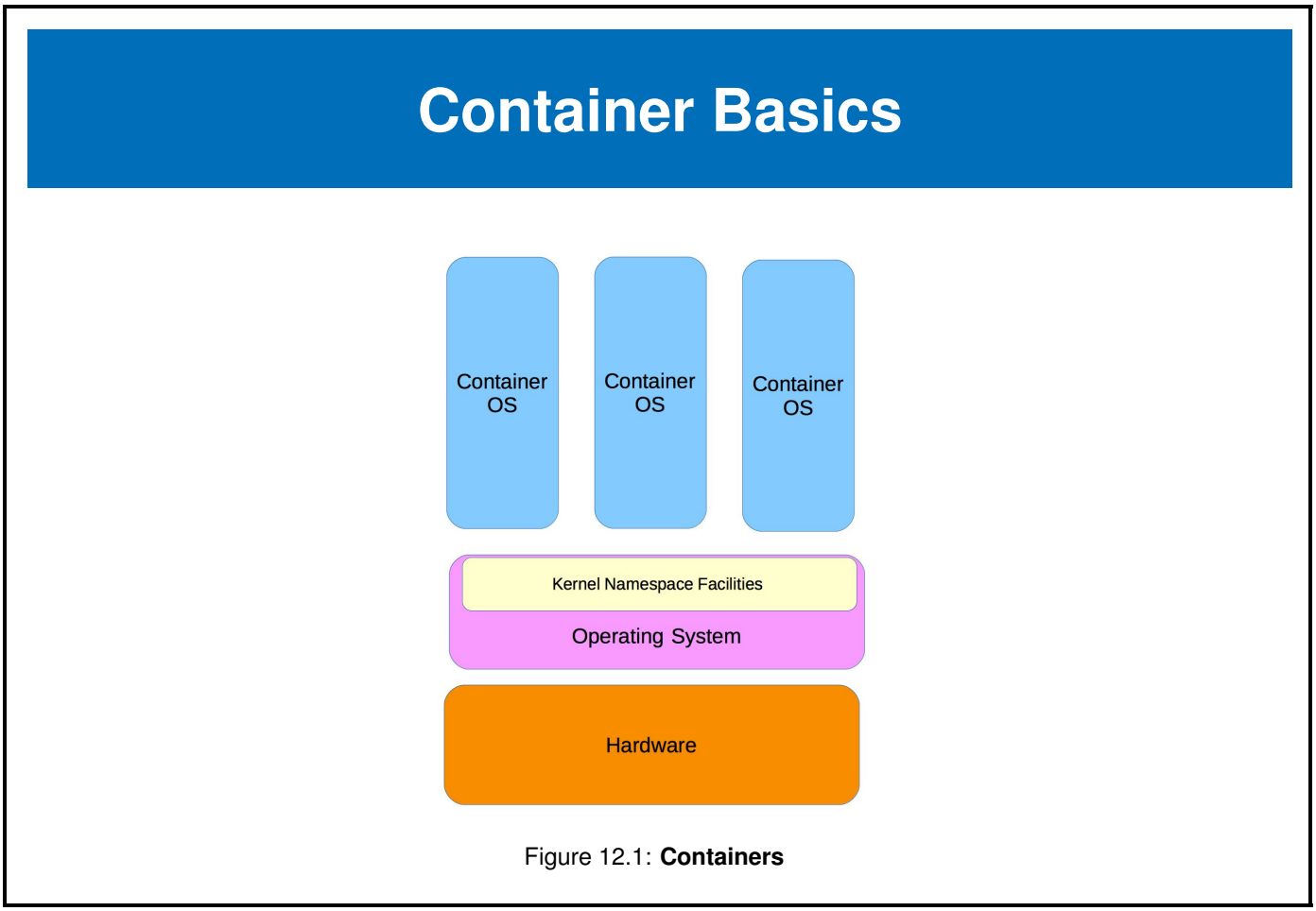


Figure 12.1: **Containers**

Further integration between the hypervisor and the **Linux** kernel allowed the creation of operating system level virtual machines or **Containers**. Containers share many facilities in the **Linux** kernel and make use of some recent kernel additions such as **namespaces** and **cgroups**. Containers are very light weight and reduce the overhead associated with having whole virtual machines.

The first flavor of containers was the **OS container**. This type of container runs an image of an operating system with the ability to run **init** processes and spawn multiple applications.

LXC (**Linux Containers**) is one example.

12.2 Application Virtualization

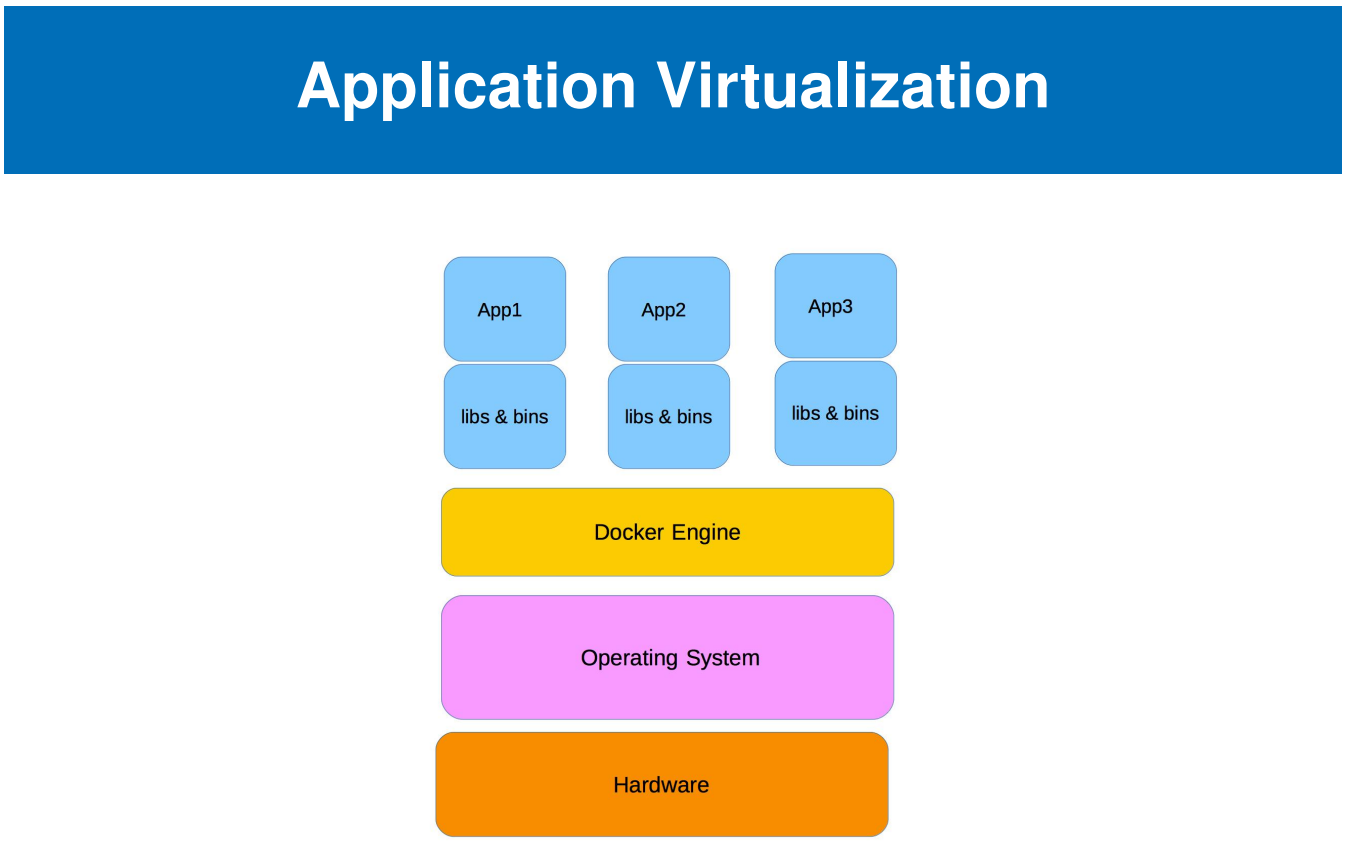


Figure 12.2: **Application Virtualization**

In an effort to further reduce the overhead associated with virtual machines, **application virtualization** is rising in popularity. Application virtualization runs one application for each container. Many single application containers are typically initialized on a single machine. Using smaller components creates a greater flexibility and reduces overhead normally associated with virtualization.

Docker is one such project.

12.3 Containers vs Virtual Machines

Containers vs Virtual Machines

- Running Processes and Services
  - VM runs a complete operating system and can run many services and applications
  - Container usually runs one thing
- VM uses more resources than a container
- Multiple containers share one OS kernel, each VM has its own
- Containers are more portable
- Containers can be run inside a VM
- Containers are harder to secure
- Containers usually start faster
- However, Virtual Machines are still often the best solution

Both Virtual Machines and Containers satisfy important needs. For a while each facility 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.

12.4 Docker

Docker

Docker is an application-level virtualization using many individual images to build up the necessary services to support the target application. These images are packaged into containers.

- Images are components in containers
- Images may contain
  - application code
  - runtime libraries
  - system tools
  - . . . or just about anything required for an application
- Images may reside on a **Docker Hub** or a **registry** server

Docker has extensive documentation at https://docs.docker.com

The **Docker** website https://www.docker.com has documentation, tutorials and training information.

One of the most compelling features of **Docker** is an application can be packaged up with all of its dependent code and services and deployed as a single unit with the minimum of overhead. This deployment can be easily repeated as often as desired. This reduces the requirement of building up a server with layered services to support the end application.

Docker Steps

Starting up a **Docker** containerized application may include only a few steps.

- Install the **Docker** service package with your favorite tool.
- Start the **Docker** service.
- Search for an appropriate image from **Docker Hub** or your private repository.
- Pull the image.
- Run the image
- and finally test the application.

The steps detailed above are just a very minimal example of testing a **Docker** application.

There are of course many, many options that may be used including functions that create an image, set system variables or configuration parameters, and then store the result as a new image. In some cases, a writable image is required, rather than a non-writable one.

Most of the **Docker** commands have individual **man** pages. Examples include: `docker(1)`, `docker-search(1)`, `docker-pull(1)`, `docker-create(1)`, and `docker-run(1)`.

12.5 Docker Commands

docker Command

- The **docker** command has more than 40 sub-commands
  - Some with 50 or more options
- `docker <command> --help` for details

- `run`
- `create`
- `exec`
- `ps`
- `images`
- `network`
- `rm`
- `rmi`
- `save`
- `search`
- `start`
- `stop`
- `top`
- `update`
- `volume`

etc.

There are many **docker** sub-commands and they tend to be somewhat self-documenting. Often confused are `run`, `create` and `exec`. The `ps` command will list running containers, or all containers if you include the **–all** option.

`run` will start a new container and execute a command within. Common options are `-t` to attach to a tty and `-d` to run the container in the background.

The `create` command creates a container. It has many options for configuring container settings and attachments.

If the container is already running, and you want to execute something inside of it you can use the `exec` command. It also accepts the `-t -d` options.

The `images` command will show images in various outputs. The `rmi` command will remove images and delete un-tagged parents by default.

You can also leverage shell functions to operate upon all containers. For example to remove all stopped containers: `docker rm $(docker ps -a -q)`.

12.6 Podman

Podman

- **RHEL8**/**CentOS8** have replaced pure **docker** with **Podman**
- **Podman** uses a child/parent forking model for container creation and management
- **Docker** uses a server/client model with a daemon running in background for management
- Promised benefits:
  - Better security
  - Less overhead

```
$ sudo dnf install podman podman-docker
```

- Emulation layer enables backwards compatibility with **docker** commands

We will show how to run the lab exercise on **RHEL**/**CentOS 8** using the **docker** emulation layer. Other distributions have been adding **podman** to their packaging systems.

On **Ubuntu 20.04** the official repositories already include **podman** so you can just do:

```
$ sudo apt update
$ sudo apt install podman
```

Podman, rootful vs rootless

Most container management utilities have the concept of non-root users running containers. Podman has very distinct differences.

- rootful
  - use **sudo** to be rootful
  - rootful does not show all resources
  - has a network bridge between the container and the host system by default.
  - uses `/etc/containers` directory for configuration
- rootless
  - can see user related resources only (images,networks,containers)
  - has an isolated private network with no bridge to the host system
  - stores configuration information in: `$HOME/.config/containers/containers.conf`

The order of precedence for the configuration files is:

1. distribution default:
   `/usr/share/containers/containers.conf`
2. administrators override file:
   `/etc/containers/containers.conf`
   (This file may not be present.)
3. users preferences:
   `$HOME/.config/containers/containers.conf`

The networking and user level access to the networking is provided by:

- **slirp4netns** via the **podman network** interface
- as of **podman v4**, the **netarvark** package can be used for rootless and rootful containers.

For additional information on podman networking, see:
https://github.com/containers/podman/blob/main/docs/tutorials/basic_networking.md

Podman, repositories

There are several locations for the container image repositories to reside:

- **/etc/containers/registries.conf**
  - system wide configuration file location
  - may also have separate **.conf** files in **/etc/containers/registries.conf.d** (format V2 only)
- `$HOME/.config/containers/registries.conf` will be used first if it exists. In the case of a conflict, the first match is used.

The repository information options, some of the more common items are:

- **registries.search** indicates which repository will be used for the **podman search** command. If there is no search repositories defined, the search will not return any information.
- **registries.insecure** by default registries require **TLS** to retrieve an image from the registries, this option allows unencrypted HTTP traffic
- **registries.block** container images cannot be pulled from the listed locations
- an additional parameter for the repositories file is the:

  unqualified-search-registries option which allows a comma separated list of image registries to be used for the **podman search** command for example:

```
unqualified-search-registries = ["docker.io"]
```

See **man 5 containers-registries.conf** for additional details.

12.7 Labs

Exercise 12.1: Install, test and run a podman container

Overview

This exercise is intended for distributions that have the **podman** package available.

For distributions that do not have the **podman** packages, use **Docker** instead; skip to the next exercise.

Table 12.1: **podman and distribution status**

Distribution	Status	Remedy
CentOS-8-Stream	Some repositories malfunction	reduce repositories to **docker.io**
CentOS-9-Stream	Works	none
Ubuntu-20.04	packages not available	use Docker
Ubuntu-22.04	may be missing repository records	update repository records in repositories.conf

1. Make sure **podman** is installed: pick the appropriate command for your distribution:

```
$ sudo dnf install podman podman-docker slirp4netns     # RHEL/CentOS, Fedora
$ sudo apt install podman podman-docker slirp4netns     # Ubuntu-22.04, Debian
$ sudo zypper install podman podman-docker slirp4netns  # openSUSE
```

2. Start the **podman** socket:

```
$ sudo systemctl start podman.socket
```

You may want to verify that it is running properly with

```
$ sudo systemctl status podman.socket?:
```

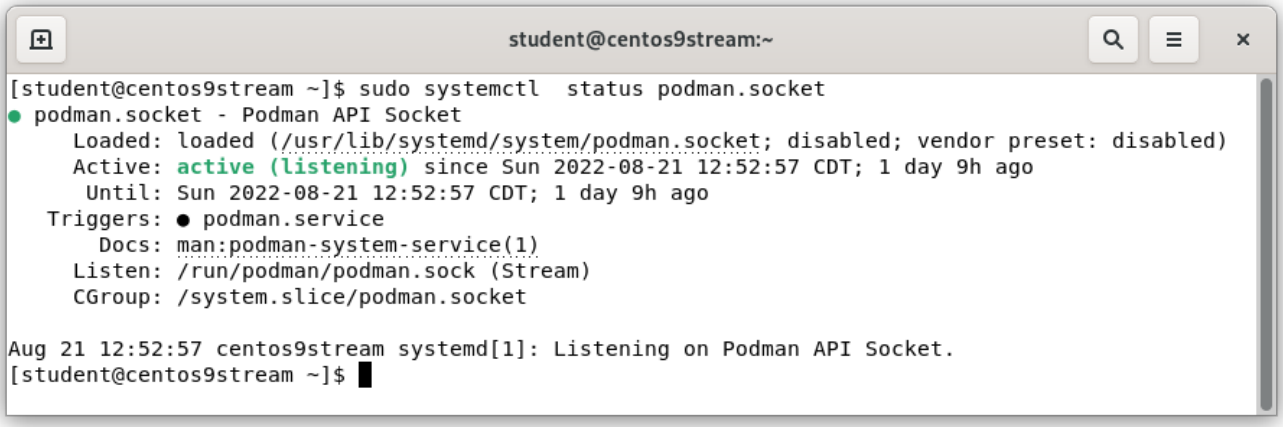


Figure 12.3: **Checking podman socket status**

If you see anything indicating failure, you should inspect `/var/log/messages` or whatever other logging file you have on your system for clues.

Kernel version may matter

If you are running a standard distribution kernel you should be fine. However, if you are running a custom **Linux** kernel, it is likely you have to select the proper configuration options, especially as regards to networking. This is too complicated to go into here, so please stay with a distribution-supplied kernel unless you want a more challenging exercise!

3. The container **registry** is a collection of containers ready to run. These may be public, private or local collections. Some distributions ship with several registries defined and others have nothing defined.
   Search for the **httpd** image container, with:

   ```
   $ sudo podman search apache
   ```

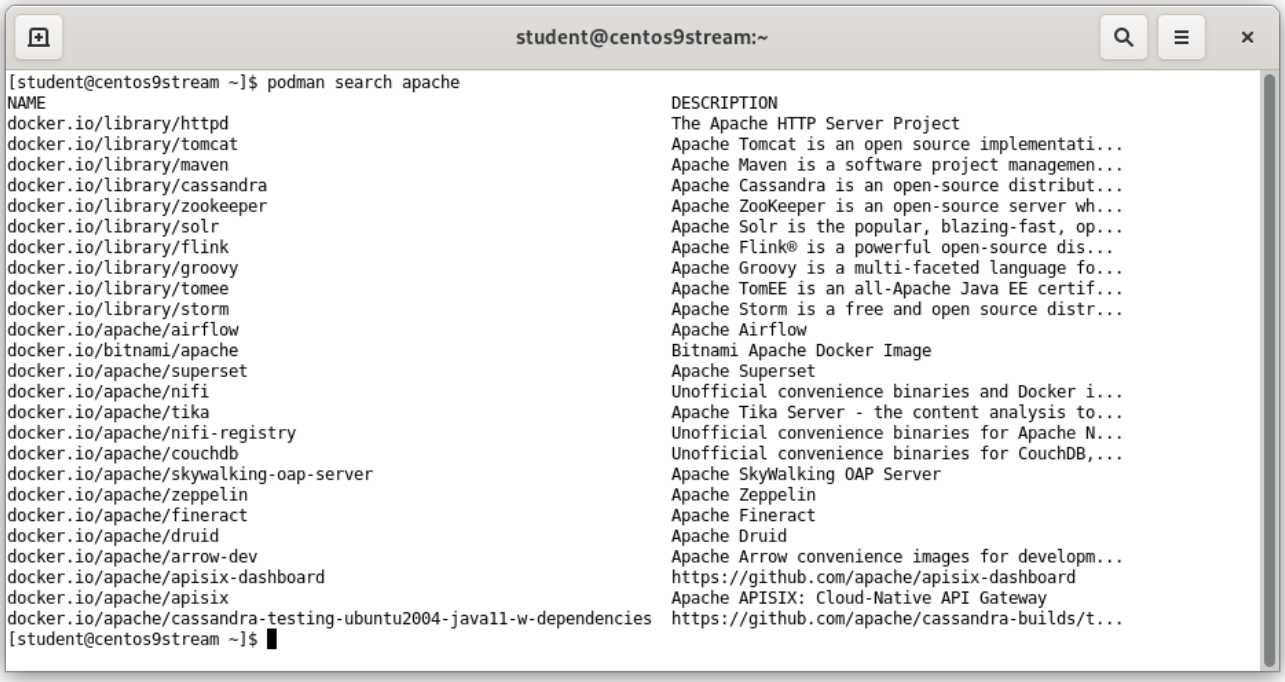


Figure 12.4: **Using podman search**

(You could have used **httpd** instead of **apache** in the above command with very similar results.)

Troubleshooting

A list of container images containing **httpd** or **apache** is expected. If this is not the case, distribution may not have defined a default search entry in the repository information. This can be easily corrected, by entering the following command and retesting the search command:

```
$ sudo sh -c 'echo unqualified-search-registries = ["docker.io"]  >> /etc/containers/registries.conf'
```

From now on we will not show detailed output since if you have gotten this far, things should be fine.

4. Retrieve the container:

   ```
   $ sudo podman pull docker.io/httpd
   ```

   This may take a couple of minutes while all the components download.

5. List the installed containers:

   ```
   $ sudo podman images
   ```

6. List the components associated with the images.

   ```
   $ sudo  podman images --all
   ```

7. Start the **httpd podman** container. The terminal will appear to hang as it is now connected to the **httpd** daemon.

8. You can open a graphical web browser pointing to the IP address in the above output. (Do not use the address shown in the output above!)

   Or you can use a text-based browser (especially if you are not in a graphical environment) by opening up a new terminal window (do not kill the one in which the **httpd** container is running!) and doing one of the following commands:

   ```
   $ lynx   http://10.88.0.33
   $ w3m    http://10.88.0.33
   $ elinks http://10.88.0.33
   ```

   using whichever text-based browser is installed on your system.

9. Stop the container and **podman** containers and clean up.

   ```
   $ sudo podman container list
   ```

   ```
   CONTAINER ID  IMAGE                           COMMAND           CREATED         STATUS             PORTS
   ↪  NAMES
   5eb53be5d5a0  docker.io/library/httpd:latest  httpd-foreground  12 minutes ago  Up 12 minutes ago
   ↪  nice_khorana
   ```

   ```
   $ sudo podman container kill nice_khorana
   ```

   ```
   nice_khorana
   ```

10. This will leave images and their associated storage under either `/var/lib/docker` or `/var/lib/containers` depending on your particular system and distribution. If you do not need to reuse them you can clean up with:

    ```
    sudo podman container prune
    ```

    ```
    WARNING! This will remove all non running containers.
    Are you sure you want to continue? [y/N] y
    00b785df8f14fafff679e759329f327de01d8d0efce645f7d708ef73c168d0f5
    23ad214fbaa9e2b1387c006f364a7e473af7c93f4d36940d001a237ea7e8309d
    27526c4912abdb22e56813d26f19554f9cd5ae3123a789a534c96253496652f2
    27d627aff076f871683cb92e6f89db6efbf47cc150fb5e71f9349c41b03a9ea5
    35e7b44229d056073cb39a718a6c4cc624fc6b5e69c861908c15d9881ff45803
    3e885f9f02a861adf49858516dff15c4432fab96afaf0cc40d62d66832857225
    47c43e3eb2eff9092c82e806f64ce3b1bf720a922715d97a4ba5fa1c0d18de7e
    ```

    ```
    $ sudo podman image list
    ```

    ```
    REPOSITORY               TAG         IMAGE ID      CREATED       SIZE
    docker.io/library/httpd  latest      f2a976f932ec  3 weeks ago   149 MB
    quay.io/libpod/banner    latest      5ba9aec95f0f  2 months ago  12.1 MB
    ```

    ```
    $ sudo podman image rm docker.io/library/httpd
    ```

    ```
    Untagged: docker.io/library/httpd:latest
    Deleted: f2a976f932ec6fe48978c1cdde2c8217a497b1f080c80e49049e02757302cf74
    ```

Exercise 12.2: Install, test and run a Docker container

Overview

This exercise is for distributions that have the **docker** package available.

Table 12.2: **Docker and distribution status**

Distribution	Status	Remedy
CentOS-8-Stream	No distribution supported packages	Use podman
CentOS-9-Stream	No distribution supported packages	use podman-docker
Ubuntu-20.04	Docker packages are available	use Docker
Ubuntu-22.04	Docker packages are available but may be missing repository records	update repository records in repositories.conf

1. Make sure **docker** is installed. Pick the appropriate command for your distribution:

```
$ sudo apt install docker.io     # Ubuntu-20.04
$ sudo zypper install docker     # openSUSE
```

2. Start the **docker** service:

```
$ sudo systemctl start docker
```

You may want to verify that it is running properly with

```
$ sudo systemctl status docker
```

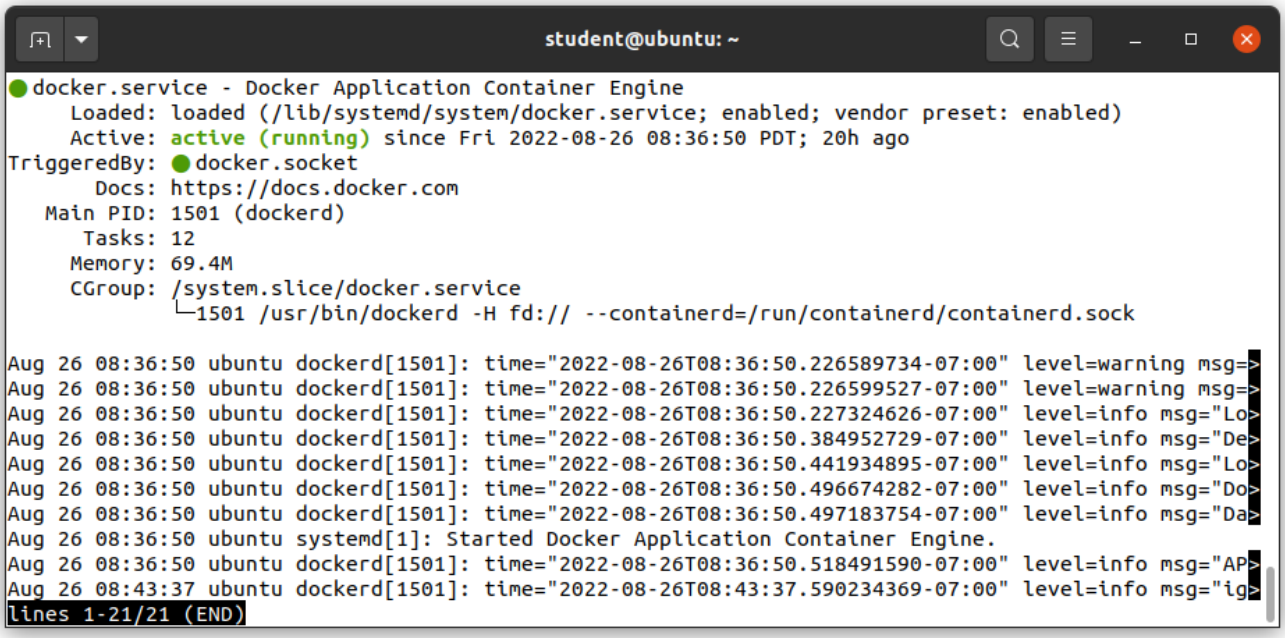


Figure 12.5: **Checking docker service status**

Troubleshooting

If you see anything indicating failure you should inspect `/var/log/messages` or whatever other logging file you have on your system for clues.

Kernel version may matter

If you are running a standard distribution kernel you should be fine. However, if you are running a custom **Linux** kernel, it is likely you have to select the proper configuration options, especially as regards to networking. This is too complicated to go into here, so please stay with a distribution-supplied kernel unless you want a more challenging exercise!

3. The container **registry** is a collection of containers ready to run. These may be public, private or local collections. Some distributions ship with several registries defined and others have nothing defined.
   Search for the **httpd** image container, with:

   ```
   $ sudo docker search apache
   ```

```
student@ubuntu:~$ sudo docker search apache
NAME                                DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
httpd                               The Apache HTTP Server Project                  3424                [OK]
tomcat                              Apache Tomcat is an open source implementati…   2979                [OK]
cassandra                           Apache Cassandra is an open-source distribut…   1251                [OK]
maven                               Apache Maven is a software project managemen…   1174                [OK]
zookeeper                           Apache ZooKeeper is an open-source server wh…   1041                [OK]
solr                                Solr is the popular, blazing-fast, open sour…   819                 [OK]
apache/airflow                      Apache Airflow                                  226
apache/nifi                         Unofficial convenience binaries and Docker i…   206                                     [OK]
eboraas/apache-php                  PHP on Apache (with SSL/TLS support), built …   144                                     [OK]
apache/zeppelin                     Apache Zeppelin                                 144                                     [OK]
eboraas/apache                      Apache (with SSL/TLS support), built on Debi…   92                                      [OK]
apacheignite/ignite                 Apache Ignite - Distributed Database            76                                      [OK]
nimmis/apache-php5                  This is docker images of Ubuntu 14.04 LTS wi…   69                                      [OK]
bitnami/apache                      Bitnami Apache Docker Image                     67                                      [OK]
apachepulsar/pulsar                 Apache Pulsar - Distributed pub/sub messagin…   34
linuxserver/apache                  An Apache container, brought to you by Linux…   27
apache/nutch                        Apache Nutch                                    23                                      [OK]
antage/apache2-php5                 Docker image for running Apache 2.x with PHP…   23                                      [OK]
webdevops/apache                    Apache container                                15                                      [OK]
newdeveloper/apache-php             apache-php7.2                                   8
newdeveloper/apache-php-composer    apache-php-composer                             7
lephare/apache                      Apache container                                6                                       [OK]
secoresearch/apache-varnish         Apache+PHP+Varnish5.0                           2                                       [OK]
apache/arrow-dev                    Apache Arrow convenience images for developm…   1
jelastic/apachephp                  An image of the Apache PHP application serve…   0
student@ubuntu:~$
```

Figure 12.6: **Using docker search**

   (You could have used **httpd** instead of **apache** in the above command with very similar results.)
   From now on we will not show detailed output since if you have gotten this far, things should be fine.

4. Retrieve the container:

   ```
   $ sudo docker pull docker.io/httpd
   ```

   This may take a couple of minutes while all the components download.

5. List the installed images:

   ```
   $ sudo docker images
   ```

6. Start the **httpd docker** container. The terminal will appear to hang as it is now connected to the **httpd** daemon.

   ```
   $ sudo docker run httpd
   ```

   ```
   AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
      using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
   ......
   ```

7. You can open a graphical web browser pointing to the IP address in the above output. Or you can use a text-based browser (especially if you are not in a graphical environment) by opening up a new terminal window (do not kill the one in which the **httpd** container is running!) and doing one of the following commands:

```
$ lynx   http://172.17.0.2
$ lynx   w3m://172.17.0.2
$ lynx   elinks://172.17.0.2
```

using whichever text-based browser is installed on your system.

8. Stop the containers and clean up.

```
$ sudo docker container list --all
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
959a4ab95626 httpd "httpd-foreground" 2 hours ago Exited (0)
2 hours ago pensive_sammet
```

```
$ sudo docker container rm pensive_sammet
```

```
pensive_sammet
```

9. This will leave images and their associated storage under either `/var/lib/docker` or `/var/lib/containers` depending on your particular system and distribution. If you do not need to reuse them you can clean up with:

```
sudo docker container prune
```

```
WARNING! This will remove all non running containers.
Are you sure you want to continue? [y/N] y
00b785df8f14fafff679e759329f327de01d8d0efce645f7d708ef73c168d0f5
23ad214fbaa9e2b1387c006f364a7e473af7c93f4d36940d001a237ea7e8309d
27526c4912abdb22e56813d26f19554f9cd5ae3123a789a534c96253496652f2
27d627aff076f871683cb92e6f89db6efbf47cc150fb5e71f9349c41b03a9ea5
35e7b44229d056073cb39a718a6c4cc624fc6b5e69c861908c15d9881ff45803
3e885f9f02a861adf49858516dff15c4432fab96afaf0cc40d62d66832857225
47c43e3eb2eff9092c82e806f64ce3b1bf720a922715d97a4ba5fa1c0d18de7e
```

```
$ sudo docker  image list --all
```

```
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
httpd        latest    a981c8992512   4 days ago   145MB
```

```
$ sudo docker  image rm httpd
```

```
Untagged: httpd:latest
Untagged: httpd@sha256:70999c4a17c796dd28f86f9c847b30f28abaed6ef1fd72a44282b1c941238804
Deleted: sha256:a981c8992512d65c9b450a9ecabb1cb9d35bb6b03f3640f86471032d5800d825
Deleted: sha256:0fbd49f82b45671985b3275bbe52bb36496047a8893146e7e6fc8b258c0c7274
Deleted: sha256:d4d2d5c5d610b9cd6063e316954eb0bdc0af36cf8c94570658278504f4c6bbf8
Deleted: sha256:1c463a675bc2ad1cd823d7944a0c7cbea87635a248257455c5fd355c98bcfd0a
Deleted: sha256:6b66301e945a6b716b5fdef63c4e2e2dd692ec6aa5a84abb1165dec8a22538c6
```

You may also want to do:

```
$ sudo systemctl stop docker
```

Chapter 13

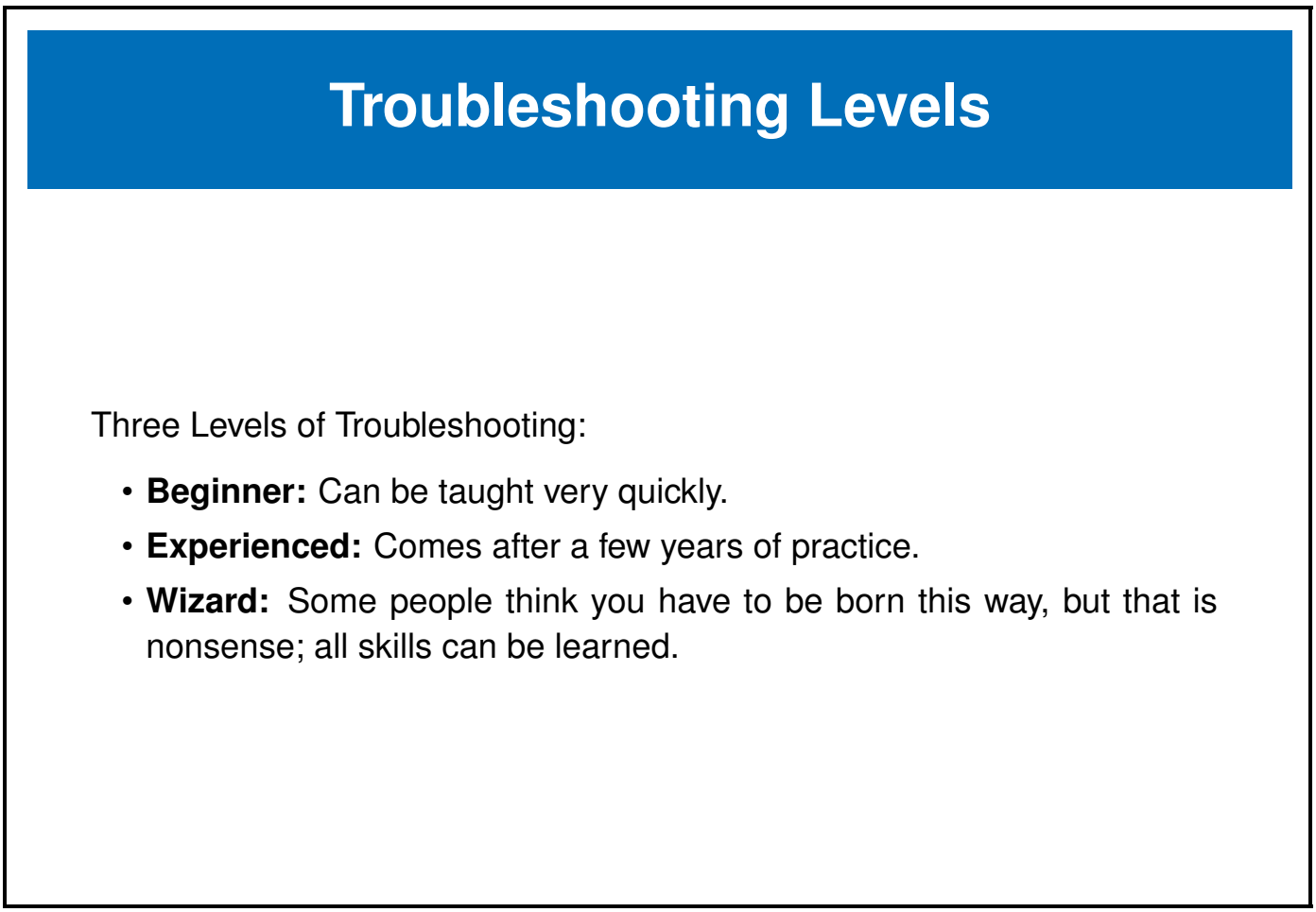
Basic Troubleshooting

Learning Objectives

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

- Grasp the fundamental techniques of troubleshooting **Linux** systems.
- Know what to examine when something goes wrong with networking.
- Know what to examine when there are questions of file integrity and software package corruption.
- Explain what steps to take when the system fails to boot properly.
- Examine and recover from filesystem corruption.
- Use **Virtual Consoles** (**VT**s).

13.1 Troubleshooting Levels



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.

13.2 Troubleshooting Techniques

Basic Troubleshooting Techniques

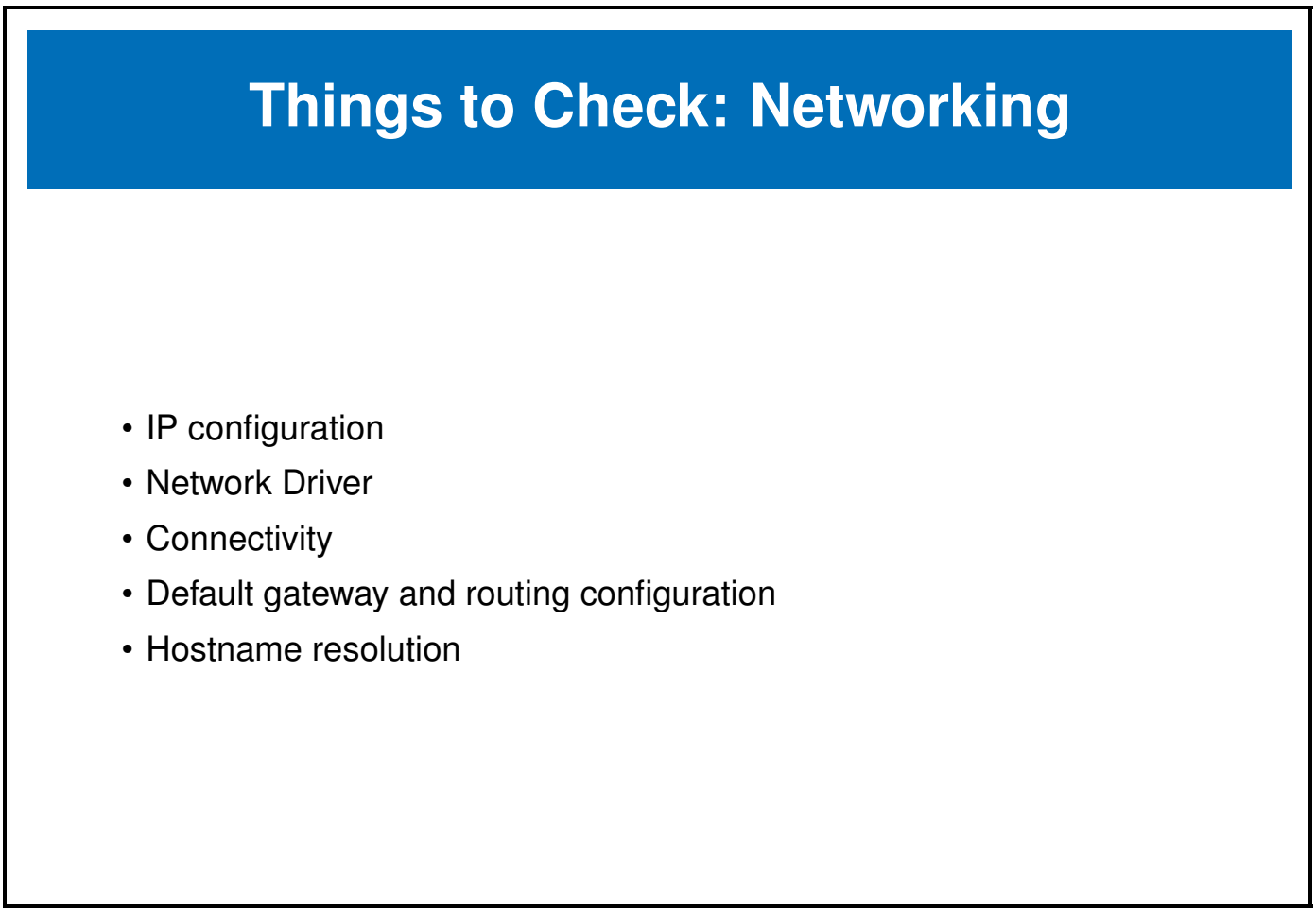
- Characterize the problem.
- Reproduce the problem.
- Always try the easy things first.
- Eliminate possible causes one at a time.
- Change only one thing at at time; if that does not fix the problem, change it back.
- Check the system logs (`/var/log/messages`, `/var/log/secure`, etc.) for further 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.

13.3 Networking



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.

13.4 File Integrity

Things to Check: File Integrity

- Check for corrupt configuration files or binaries
- Can use numerous methods
  - **rpm**

    `$ rpm -V some_package`

    to check a single package

    `$ rpm -Va`

    to check all packages
  - **aide**: intrusion detection

    `$ aide --check`
  - **debsums**: **Debian**

    `$ debsums options some_package`

There are a number of ways to check for corrupt configuration files and binaries. One way is to use the command `rpm -V` to check a particular package. You can also use `rpm -Va` to check the integrity of all packages.

Using **aide** is another way to check for changes in files. The command `aide --check` will run a scan on your files and compare them to the last scan.

In **Debian**, the only way to do integrity checking is with **debsums**. Running `debsums somepackage` will check the checksums on the files in that package. However, not all packages maintain checksums so this might be less than useful.

13.5 Boot Process Failures

Boot Process Failures

- Follow the boot process
  - **BIOS**
  - **GRUB**
  - Kernel
  - **init**

If the system fails to boot properly or fully, being familiar with what happens at each stage is important. Assuming you get through the **BIOS** stage, you may reach a state of:

No bootloader screen:

Check for **GRUB** misconfiguration or a corrupt boot sector. Bootloader re-install needed?

Kernel fails to load:

If the kernel **panics** during the boot process, it is most likely a misconfigured or corrupt kernel, or incorrect kernel boot parameters in the **GRUB** configuration file. If the kernel has booted successfully in the past, either it has corrupted, or bad parameters were supplied. Depending on which, you can re-install the kernel, or enter into the interactive **GRUB** menu at boot and use very minimal command line parameters and try to fix that way. Or you can try booting into a rescue image.

Kernel loads but fails to mount the root filesystem:

The main causes here are

1. Misconfigured **GRUB** configuration file.
2. Misconfigured `/etc/fstab`.
3. No support for the root filesystem type built into the kernel or in the **initramfs** image.

Failure during the init process.

There are many things that can go wrong once **init** starts; look closely at the messages that are displayed before things stop. If things were working before, with a different kernel, that is a big clue. Look out for corrupted filesystems, errors in startup scripts etc. Try booting into a lower runlevel such as `3` (no graphics) or `1` (single user mode).

13.6 Filesystem Corruption and Recovery

Filesystem Corruption and Recovery

- **fsck** may be used to fix corrupted filesystems
- Check `/etc/fstab` for misconfiguration
- May need to run

  ```
  $ mount -o remount,rw /
  ```

  to attempt fixes
- Try manually mounting filesystems

fsck may be used to attempt repair. However, before doing that one should check that `/etc/fstab` has not been misconfigured or corrupted. Note once again you could have a problem with a filesystem type the kernel you are running does not understand.

If the root filesystem has been mounted you can examine this file, but `/` may have been mounted as read-only. so to edit the file and fix it you can run:

```
$ sudo mount -o remount,rw /
```

to remount it with write permission.

If `/etc/fstab` seems to be correct, you can move to **fsck**. First you should try:

```
$ sudo mount -a
```

to try and mount all filesystems. If this does not succeed completely, you can try to manually mount the ones with problems. You should first run **fsck** to just examine; afterwards you can run it again to have it try and fix any errors found.

13.7 Virtual Consoles

Using the Virtual Consoles

- **Linux** has 12 **virtual consoles** by default (also called **virtual terminals**)
  - The first 6 are defined as login text consoles
  - Console 1 is used by most distributions as the system console
  - Graphical console is usually console 7, but some distributions (including **RHEL**) use console 1.
- Use `Ctrl-Alt-FX` to switch between consoles
  - Example: `Ctrl-Alt-F5` goes to console 5.

By default, **Linux** defines 12 virtual consoles to allow local access to the system. The first 6 are usually login text consoles. The seventh is usually the graphical console, if you have one.

You can use `Ctrl-Alt-FX` (where `X` is the number of the console) to go between the console.

13.8 Labs

Please Note

There are no lab exercises for in this chapter. It just summarizes points discussed earlier when considering configuring and monitoring the system, and in addition, sets the stage for the following section on system rescue, which has several labs.

Chapter 14

Introduction to GIT

We explain the concepts of **revision control systems**, and chart the evolution of **git** from its historical antecedents. We talk about the graphical interfaces available for use with **git**. We point to important sources of documentation.

Learning Objectives

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

- Explain why revision control systems are important.
- Understand the role they play in projects with distributed development, such as the use of **git** for the **Linux** kernel.
- Grasp why it is important to know the provenance of every line of code through the use of **CLA**s and **DCO**s.
- Survey the history of source control systems and list currently available open source systems.
- Know the graphical interfaces to **git**.
- Be able to find detailed documentation on the use of **git**.

14.1 Revision Control

Why We Need Revision Control Systems

We need revision control for many reasons some of the top requirements are:

- Changes accumulate with time:
  - Bug and Security Fixes
  - Product Enhancements
- Need to keep track of changes:
  - and the ability to revert as necessary
- A clear revision history is needed

As you work on a project changes accumulate. As you fix bugs, add new features and optimize, your code base will diverge dramatically from where you began.

Sometimes you add a feature or fix a bug and think everything is working perfectly, but then such wishful thinking meets reality when a user (perhaps even yourself) employs your application in a way you did not foresee. Or you hit what was thought to be an impossibly rare race condition when you port to a new platform with faster hardware and multiple CPUs.

At such a point you may have progressed considerably past the point where the problem was introduced, and you will have to locate the point in your development where things went wrong, fix the problem, and bring all the other changes forward to the current state.

Every developer has been sloppy about this at some point, particularly when they are the only one working on the project and code base. While there is no substitute for a clean and intelligent design that has a lot of modular components, a design which is easy to break down and reassemble once pieces have been fixed, there are tools which can be used to greatly facilitate efficient cycles of development.

Distributed Development

- Having many contributors (even thousands as for the **Linux** kernel) complicates matters
- Is there a central authority? If so, whom?
- What if more than one developer works on the same code and conflicts arise? How are they resolved?
- How is the project history maintained, line for line? Who is responsible for the code?
  - For maintenance
  - For authenticity declaration
  - For legal and licensing considerations
- Many revision control systems have existed over time. We will concentrate on **git**

The simplest method is to simply keep regular backup snapshots of the work, and revert to them as problems arise. We have all done this. But it is a manual process, and as the size of the project grows it becomes more and more unwieldy. You will wind up differencing your current code base with earlier ones; why not just store the history of changes and the differences instead of complete snapshots? If nothing else, you will save disk space. More importantly you can track the changes directly instead of inferring them from the endpoint and starting point.

As other developers are added to the project the manual method also becomes difficult to manage:

- One developer has to remain the central authority who handles all actual changes.
- Other developers cannot **commit** changes directly unless you want to confuse the result.

As more developers are added it becomes more and more difficult to manage the simultaneous contributions.

Massively distributed projects (such as the **Linux** kernel) can involve thousands of contributors, and even the notion of a central authoritative repository of all wisdom can become a hindrance.

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.

14.2 Know Where the Code is Coming From: DCO and CLA

DCO and CLA

- Code origins must be documented carefully for both provenance reasons and maintenance
- **CLA** (Contributor License Agreement)
  Older method, all contributors sign an agreement
- **DCO** (Developer Certificate of Origin)
  Introduced by the **Linux Foundation** in 2004. All contributors must **sign off** on every contribution
- **CLA** one time operation; **DCO** for each commit
- See https://wki.linuxfoundation.org/dco

It is very important for any open source project to document and understand where all contributed code is coming from. This is important for multiple reasons, including:

- Knowing who the expert(s) are on the contributions; when there are questions or modifications, generally these people need to either answer them or do the work, or at least review it before acceptance.
- Knowing the legal provenance of the code; if someone either inadvertently or intentionally uses code that violates the license it may have used in another context this can be a big problem. In this case it is important to know who to *blame*.

Two common methods for handling these issues are the use of either a **Contributor License Agreement** (**CLA**) or a **Developer Certificate of Origin** (**DCO**). There can be arguments about which is more appropriate for a given project. However, a number of well-established projects have migrated from **CLA** to **DCO** as their operative integrity documentation.

With a **CLA** (the older method) contributors must sign an agreement which can vary from project to project. For example, they may or may not grant copyright and/or patent licenses.

The **DCO** was introduced by the **Linux Foundation** in 2004 and is simpler and of lighter weight. Every **git** commit must be **signed off** with this approach.

With either approach, the servers hosting the software repositories (such as **GitHub** or **GitLab**) can use checks to ensure all contributions either have a **CLA** in place or are signed off as required by a **DCO**.

Generally speaking a **CLA** is a one-time operation, whereas a **DCO** requires a sign-off with each contribution, but this is trivial (i.e., adding the `-s` option to each commit operation.)

A detailed summary of the use of **DCO**s is given at https://wiki.linuxfoundation.org/dco. When you use the `-s` option, each commit will have a message embedded such as:

```
Signed-off-by: Some Developer
<some-developer@example.com>
```

14.3 Available Revision Control Systems

Available Source Control Systems

- Long history of revision/source control systems. Earliest include:
  - **SCCS**
  - **RCS**
  - **CVS**
- More modern open source systems:
  - **Subversion**
  - **Bazaar**
  - **Monotone**
  - **Mercurial**
  - **PRCS**
  - **git**
- We will only consider **git** which has by far the dominant position today, with literally millions of projects managed within its framework

Revision control systems have been available as long as developers have been working. Probably the first system widely available on **Unix**-like platforms was **SCCS** (**S**ource **C**ode **C**ontrol **S**ystem) which dates back to the early 1970s.

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

Table 14.1: **Available Source Control Systems**

Product	URL
RCS	http://gnu.org/software/rcs/
CVS	http://savannah.nongnu.org/projects/cvs
Subversion	https://subversion.apache.org/
git	https://kernel.org/pub/software/scm/git/
Bazaar	http://bazaar.canonical.com
Monotone	https://monotone.ca
Mercurial	https://mercurial-scm.org
PRCS	http://prcs.sourceforge.net

We will not discuss proprietary products for which there is also a long list.

14.4 Graphical Interfaces

Available GUIs

- Multiple graphical interfaces exist:

Table 14.2: **Available Graphical Interfaces for Git**

Interface	Description
git-gui	To use this you merely have to launch it from within the main repository directory, either by typing `git gui` or `git-gui` . It enables you to browse histories, make changes, commits, work with remote repositories, compare branches etc.
gitk	This repository browser can actually be invoked from within **git-gui** and is a somewhat older interface more dedicated to looking at project history.
cgit	Can be installed in concert with a web server to enable very efficient repository browsing. For a good example of its capabilities try https://kernel.org.
gitweb	An older interface than **cgit** that works in a similar fashion.

- In addition, many users work with the web interface provided by **GitHub**, **GitLab**, etc

There are a number of graphical interfaces which can be used to view or maintain **git** repositories. In this class we will not concentrate on them but we do not mean to discourage their use.

The reason for this emphasis is that while GUIs come and go, and change their user interface, the command line based tools are less likely to undergo major revision. Furthermore, the better you understand what is going on under the hood, the better use you will make of **git** even when you are using a graphical interface.

We may from time to time demonstrate the graphical interfaces, but we will not design the class around them.

git gui

- As an example, here is a snapshot of **git gui** (run by typing `git gui`):

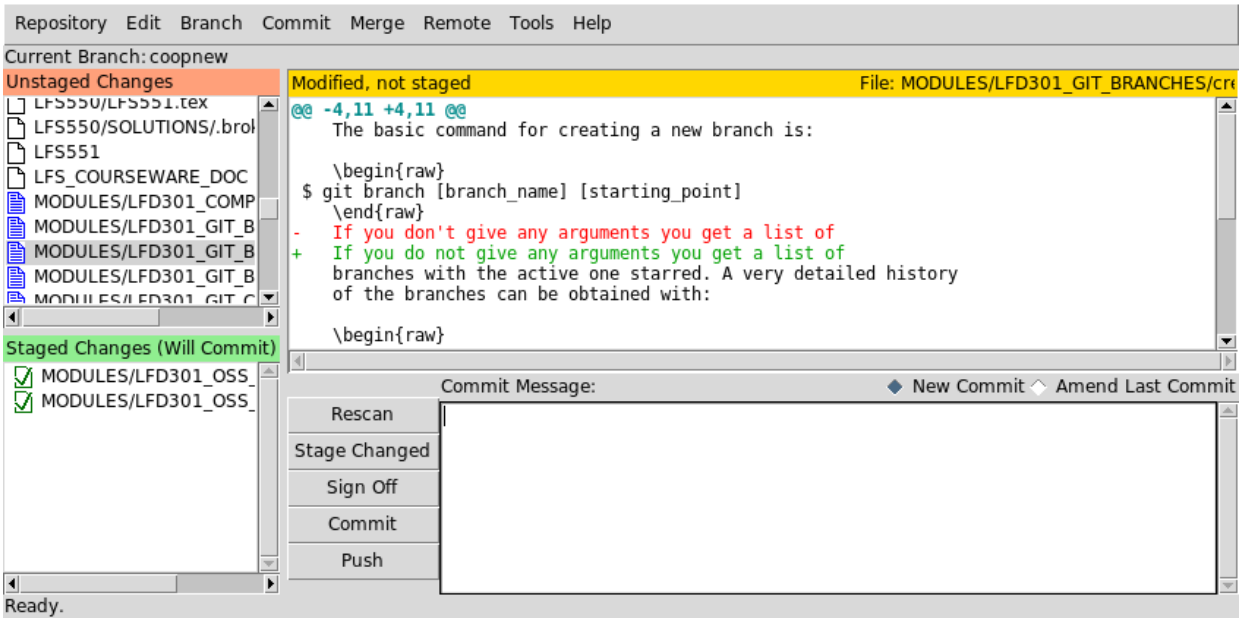


Figure 14.1: **git gui**

- **git-gui** is a graphical user interface focused on the code on your local machine.
- It can be used for committing changes, creating branches, doing merging, fetching and pushing to remote repos.
- **git-gui** is written in the well-known **Tcl/Tk** framework and thus is available on almost any operating system.
- The command `man git-gui` will give details of its operation.

14.5 Documentation

Local documentation

- For help on all commands or a specific one:

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

- For help on a specific command these two commands are equivalent

```
$ git help commit
$ man git-commit
```

Any **git** installation brings with it a lot of documentation on your local machine. Just typing:

```
$ git help
```

gives you the basic commands. Detailed help about any one of them is obtained with:

```
$ git help command
```

One can get help on particular commands in either of two ways:

```
$ git help commit
$ man git-commit
```

the result being a conventional **man** page.

Further Sources of Documentation

- https://kernel.org/pub/software/scm/git/docs/user-manual.html:
  The official **Git User's Manual**.
- https://git-scm.com
  The home page for **git**
- ***Version Control with Git***, by Jon Loeliger, pub. O'Reilly, 2009.
- ***Version Control with Git: Powerful tools and techniques for collaborative software development***, by Jon Loeliger and Matthew McCullough, pub. O'Reilly, 2012.
- ***Introduction to Git*** by James Bottomley,
  https://training.linuxfoundation.org/resources/webinars/introduction-to-git/
- ***GIT Started with Community Software*** by Jerry Cooperstein,
  https://training.linuxfoundation.org/cm/LFS303/ESC.pdf.

Two great sources of documentation are:

- https://kernel.org/pub/software/scm/git/docs/user-manual.html: The official **Git User's Manual**. There are many other useful documents either residing at the same website, or linked to from there.

  This document is terse but authoritative and extremely useful.

- ***Version Control with Git***, by Jon Loeliger, pub. O'Reilly, 2009.

  Version Control with Git: Powerful tools and techniques for collaborative software development, by Jon Loeliger and Matthew McCullough, pub. O'Reilly, 2012.

  This book (and its second edition) is one of several about using **git** but it is the most up to date, and the most thorough. There are many topics in this class that receive only cursory treatment given time constraints; this book goes into great detail with many examples bringing to the fore real life problems and experiences.

The best overall source of documentation can be found at at the **git** home page at https://git-scm.com, which contains a wealth of documentation and examples.

For a slide presentation, ***GIT Started with Community Software*** presented at the Chicago Embedded Systems Conference in 2010, by Jerry Cooperstein, download:

https://training.linuxfoundation.org/cm/LFS303/ESC.pdf.

For a free on-line webinar by James Bottomley, ***Introduction to Git*** go to: https://training.linuxfoundation.org/resources/webinars/introduction-to-git/.

14.6 Labs

Video Demonstration Resources

using_git_gui.mp4

Exercise 14.1: cgit Example at git.kernel.org

- To see a comprehensive implementation of the **cgit** repository browser, go to https://git.kernel.org which serves as the host for many projects growing out of the **Linux** kernel community, including **git** itself.
- Look at a particular repository by scrolling down to the project and clicking on `summary`. See if you can find the main **Linux** kernel repository. Look at the history and patches etc.
- This is not the same as looking at https://kernel.org which is also powered by **cgit**.

Chapter 15

Using Git: an Example

Learning Objectives

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

- Be familiar with the major commands used with **git**, such as **clone**, **add**, **commit**, **merge**, **pull**, **push**, etc.
- Get on-system help with any particular command with **git help**.
- Work through a simple but complete example including creating and initializing a repository, making changes to it and making it available to collaborators.
- Know how to sign off on contributions establishing who is responsible.
- Explain why the primary authoritative branch should be named **main** (or something similar) and not **master**.

15.1 Basic Commands

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

```
$ git --version
```

```
git version 2.27.0
```

Detailed help information in the form of a **man** page can be obtained about any subcommand by doing:

```
$ git help [subcommand]
```

For example the two following statements produce the same result:

```
$ git help status
$ man git-status
```

Basic git commands

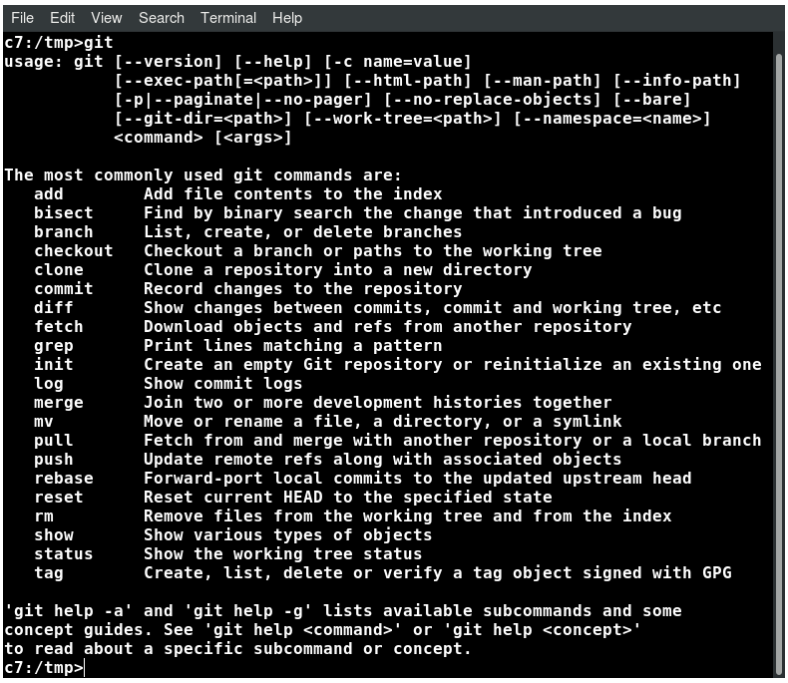


Figure 15.1: **Basic git commands**

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.

More Help

If you can not resist seeing the more complete set of commands do:

```
$ git help --all
```

```
       See 'git help <command>' to read about a specific subcommand

   Main Porcelain Commands
      add                  Add file contents to the index
      am                   Apply a series of patches from a mailbox
      archive              Create an archive of files from a named tree
      bisect               Use binary search to find the commit that introduced a bug
      branch               List, create, or delete branches
      bundle               Move objects and refs by archive
      checkout             Switch branches or restore working tree files
      cherry-pick          Apply the changes introduced by some existing commits
      citool               Graphical alternative to git-commit
      clean                Remove untracked files from the working tree
      clone                Clone a repository into a new directory
      commit               Record changes to the repository
      describe             Give an object a human readable name based on an available ref
      diff                 Show changes between commits, commit and working tree, etc
      fetch                Download objects and refs from another repository
      format-patch         Prepare patches for e-mail submission
      gc                   Cleanup unnecessary files and optimize the local repository
      gitk                 The Git repository browser
      grep                 Print lines matching a pattern
      gui                  A portable graphical interface to Git
      init                 Create an empty Git repository or reinitialize an existing one
      log                  Show commit logs
      merge                Join two or more development histories together
   ....
```

This long list may seem rather intimidating but we some of them are really for expert usage and rarely used, or are more efficiently invoked through shorthand combinatorial commands.

Furthermore, there are several graphical interfaces to **git** which avoid having to be able to name all the plumbing fixtures.

15.2 A Simple Example

Create and Initialize Repository

Create a local project.

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

Creates a `.git` directory which will contain all the version control information.

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

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

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 7 coop coop 4096 Dec 30 13:59 ./
drwxrwxr-x 3 coop coop 4096 Dec 30 13:59 ../
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.

Add Files to the Project

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

$ git status
```

```
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** (added) but not yet **committed**.

Whose Repository is it

Tell **git** who is responsible for this repository:

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

Must be done for each new project unless it is predefined in a global configuration file.

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

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

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

Modifications

Modify a file:

```
$ 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
```

Commit the change:

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

```
Created initial commit eafad66: My initial commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 somejunkfile
```

- 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
```

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

```
$  git add somejunkfile
$  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`.

History

```
$ git log
```

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

    My initial commit
```

- 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 which we will discuss later. **git** cares about these beasts, not file names.

- You are now free to modify the already existing file and add new files with `git add`. But the changes are not actually placed fully in the repository until you do another `git commit`
- Now, that was not so bad. But we have only scratched the surface.

15.3 Signing Off on Commits

Signing Off on Commits

Know who is responsible for all changes to the repository to track history and ensure proper licensing and ownership.

Add a `Signed-off-by` line to the commit with the `-s` option:

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

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

    My initial commit

    Signed-off-by: Another Genius <a_genius@linux.com>
```

Signers need not be the author; can be maintainers or reviewers

It is important to know who is responsible for all changes to the repository, in order to track history and to know who is guaranteeing that the code included is being done so with proper licensing and ownership.

This is done most easily by adding a `Signed-off-by` line to the commit with the `-s` option. So we could have done:

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

and then we would get:

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

    My initial commit

    Signed-off-by: Another Genius <a_genius@linux.com>
```

Note that the person who does the signing need not be the author of the changes; they could be a maintainer or another reviewer, but they do take on any responsibilities required. Furthermore, there can be more than one person signing off on any given change.

15.4 master vs main

master vs main

The name **master** is deprecated: most projects have replaced it with **main**

See https://inclusivenaming.org/word-lists/ created by the **Inclusive Naming Initiative**.

Also deprecated are use of terms like **blacklist**, **whitelist**, **worker** and **slave**

For a new repository:

```
$ git init
$ git checkout -b main
```

From now on we will assume that the name of the branch that plays this role is `main` and that any necessary changes have been made.

- Historically, a new repository was always created with an initial **branch** called `master`. However, many projects have deprecated this name and replaced it with `main`.
- Reasons for this change and other changes of terminology, such as deprecation of the use of the words **blacklist** and **whitelist**, are explained in https://inclusivenaming.org/word-lists/ created by the **Inclusive Naming Initiative**.
- Technically this main branch could have any name, as there is nothing structurally special about this authoritative branch. However, **git** has to have a default name for new repositories, and some web hosts have requirements that are hard to avoid.
- **GitHub** has strongly recommended this naming convention change and has given instructions at https://github.com/github/renaming on how to create new repositories as well as rename old ones to have `main` as the main branch.
- If you are creating a new repository the easiest way to do this from the outset is to do something like:

```
$ git init
$ git checkout -b main
```

Renaming Existing Repos

Rename both local and remote branches:

```
$ git checkout master
$ git branch -m master main     # Change the local name

# Change the remote name
$ git push -u origin main
$ git symbolic-ref refs/remotes/origin/HEAD refs/remotes/origin/main

# Confirm the names!
$ git branch -a
```

Easier approach:

```
$ git checkout master
$ git branch main
$ git checkout main
$ git push -u origin main
```

and then just ignore `master` from now on.

- For existing repositories you can rename both the local branch and the remote branch on the server with

```
$ git checkout master
# Change the local name
$ git branch -m master main

# Change the remote name
$ git push -u origin main
$ git symbolic-ref refs/remotes/origin/HEAD refs/remotes/origin/main

# Confirm the names!
$ git branch -a
```

- Depending on the setup of the remote server (such as **GitHub**) this may or may not work. An easier approach is to not delete the `master` branch but just copy it to `main` and work from there from now on, as in:

```
$ git checkout master
$ git branch main
$ git checkout main
$ git push -u origin main
```

and then just ignore `master` from now on.

15.5 Labs

Exercise 15.1: Setting up a repository, and making changes and commits

Using just the simple subset of commands we have already given you, we are going to set up a simple repository.

1. First initialize the repository with `git init`. Then add author and email information using `git config`.
2. Create a couple of simple text files and add them to the repository and commit them with `git add` and `git commit`.
3. Now modify one of the files, and run `git diff` to see the differences between your working project files and what is in the repository.
4. Add the changed file to the repository again and then run `git diff` again.
5. Finally commit again, and then using `git log` examine your history.

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 15.1

1.
```
# Set up the directory we are going to work in

rm -rf git-test ; mkdir git-test ; cd git-test

# initialize the repository and put our name and email in the .config file

echo -e "\n\n*********   CREATING THE REPOSITORY AND CONFIGURING IT\n\n"

git init
git config user.name "A Smart Guy"
git config user.email "asmartguy@linux.com"
```

2.
```
echo -e "\n\n*********   CREATING A COUPLE OF FILES AND ADDING THEM TO THE PROJECT AND COMMITTING\n\n"

# create a couple of files and add them to the project
# we'll do this as two commits, although we could do it as one

echo file1 > file1
git add file1
git commit file1 -s -m "This is the first commit"

echo file2 > file2
git add file2
git commit . -s -m "This is the second commit"
```

3.
```
# modify one of the files and then see the difference with the repository

echo -e "\n\n*************   MODIFYING ONE OF THE FILES AND THEN DIFFING\n\n"
```

```
echo This is another line for file2 >> file2
git diff
```

4.
```
# now stage it and diff again
git add file2
git diff
```

5.
```
echo -e "\n\n*************   ADDING THE MODIFIED FILE AND THEN DIFFING AGAIN\n\n"

echo -e "\n\n*************   RECOMITTING FOR THE THIRD TIME\n\n"

# now get it all in with another commit

git commit . -s -m "This is the third commit"

echo -e" \n\n*************   LOOKING AT THE HISTORY OF THE PROJECT\n\n"

# look at the history

git log
```

You can download a script with the above steps from

`s_15/lab_gitexample.sh`

in your solutions file.

`lab_gitexample.sh`

Chapter 16

DevOps and GitOps

Learing Objectives

By the end of this session, you should:

- Be familiar with the the evolution of software development management
- Understand the basic concepts of **DevOps**
- Understand the concept of why breaking down the **silos** between development and operations teams is key to **DevOps**
- Be familiar with the difference between culture and tools as it applies to **DevOps**
- Be familiar with **GitOps** and how it relates to **DevOps**

16.1 Introduction

DevOps Introduction

The larger context of **DevOps** and **GitOps** is the discussion of how to manage the Software Development Lifecycle (SDLC).

The typical stages of the SDLC are:

- **Planning:** Gathering user requirements and defining project goals and objectives.
- **Design:** Developing a detailed plan for the software system, including technical specifications, architecture, and user interface design.
- **Implementation:** Coding and building the software to the design.
- **Testing:** Ensuring the software works as expected.
- **Deployment:** The completed software is delivered to the end-users, and the system is put into operation.
- **Maintenance:** The software system is maintained to ensure it continues to meet user needs.

The software development lifecycle (SDLC) is a process used by software development teams to design, develop, test, and deploy software. It is a framework that provides a structured approach to software development projects and helps ensure that the software is of high quality, meets the users' requirements, and is delivered on time and within budget.

SDLC management has gone through many evolutions and continues to evolve. It is this evolution that gave rise to **DevOps** and **GitOps**.

Various cultural shifts and methodologies have arisen throughout software development history in an attempt to manage complexity of the SDLC. We will go through a few of the most prominent of these cultural shifts and methodologies. This is the context in which **DevOps** and **GitOps** are best understood.

16.2 Cultural Philosophies and Methodologies

Cultural Philosophies and Methodologies

The attempts at managing the complexity of software development can generally be placed in two categories - **cultural philosophies** and **methodologies**:

- **Cultural philosophies** (the 'should') are organizational value propositions. They state something about what an organization believes should be done. "We should value people over processes so that we can achieve better software development," is a cultural philosophy. It says nothing about how to achieve valuing people over processes, just that we should.
- **Methodologies** (the 'how') are the answer to, "How do we implement a Cultural Philosophy." In software development, this could be Scrum for Agile or CI/CD for DevOps.

The development of these intertwined subcategories of software development management feed one another. Sometimes a team will stumble upon a methodology and the cultural philosophy that engenders it will be defined later. Other times, someone may notice something wrong and identify a cultural change that needs to happen. New methodologies may then be developed or existent methodologies will be co-opted to implement the new cultural philosophy.

This dynamic interplay between software development management cultural philosophies and methodologies causes untold confusion for anyone who attempts to find clear cut lines and definitions. Que the holy wars. The debates of how to define philosophies and methodologies verge into the nonsensical and very rude as passionate advocates for this or that definition duke it out on IT forums. Various competing definitions fly and the insults fly faster.

It is sufficient to get a general idea and feeling of the landscape of software development management. Concern yourself mostly with how your particular organization defines things. As you grow in the IT industry you will naturally begin to construct a mental image of the landscape that makes sense to you. You will also begin to form your own opinions of the most useful way to define things. While definitions matter, getting work done efficiently matters more.

16.3 Early Software Development Management Practice

Waterfall

Waterfall is one of the earliest software development management practices.

Waterfall is characterized by each step of the SDLC being completed before the next step can begin.

It is known for a lack of flexibility.

Waterfall is a methodology that stems from a business culture in which control is the primary value.

The waterfall method is a sequential and linear project management methodology that was first introduced in the 1970s for software development but has since been used for various types of projects. The approach is named ”waterfall” because it progresses through stages in a downward fashion, like a waterfall.

The waterfall method follows a rigid, sequential approach with distinct stages that must be completed in order.

The waterfall method is often criticized for being inflexible and not allowing for changes or revisions once a stage is completed. It can also result in longer development times and higher costs due to the need for more extensive testing and development documentation. However, it is still used in some industries where a highly structured, documented, and predictable approach is necessary, such as in government projects, aerospace, and medical devices.

16.4 Modern Software Development

Agile

The **Agile** cultural philosophy was created in response to the rigidity of traditional corporate culture and the Waterfall method.

The Agile philosophy was created with the **Agile Manifesto**, a document created by some well known names in the software industry in 2001. The document can be found here - https://agilemanifesto.org/

The Agile cultural philosophy (which was heavily influenced by the Extreme Programming methodology) gave rise to the Agile methodology of software development. Because the Agile philosophy values an iterative approach to software development, Agile methodologies take an iterative approach to the SDLC. Whereas the Waterfall approach to the SDLC is rigid and linear, Agile methodologies are cyclical and circular.

DevOps

Development teams are responsible for the development of software.

Operations teams are responsible for the deployment of software.

In the old days, these teams were silo-ed.

DevOps is a cultural philosophy that states the two teams should be radically cooperative if not merged entirely.

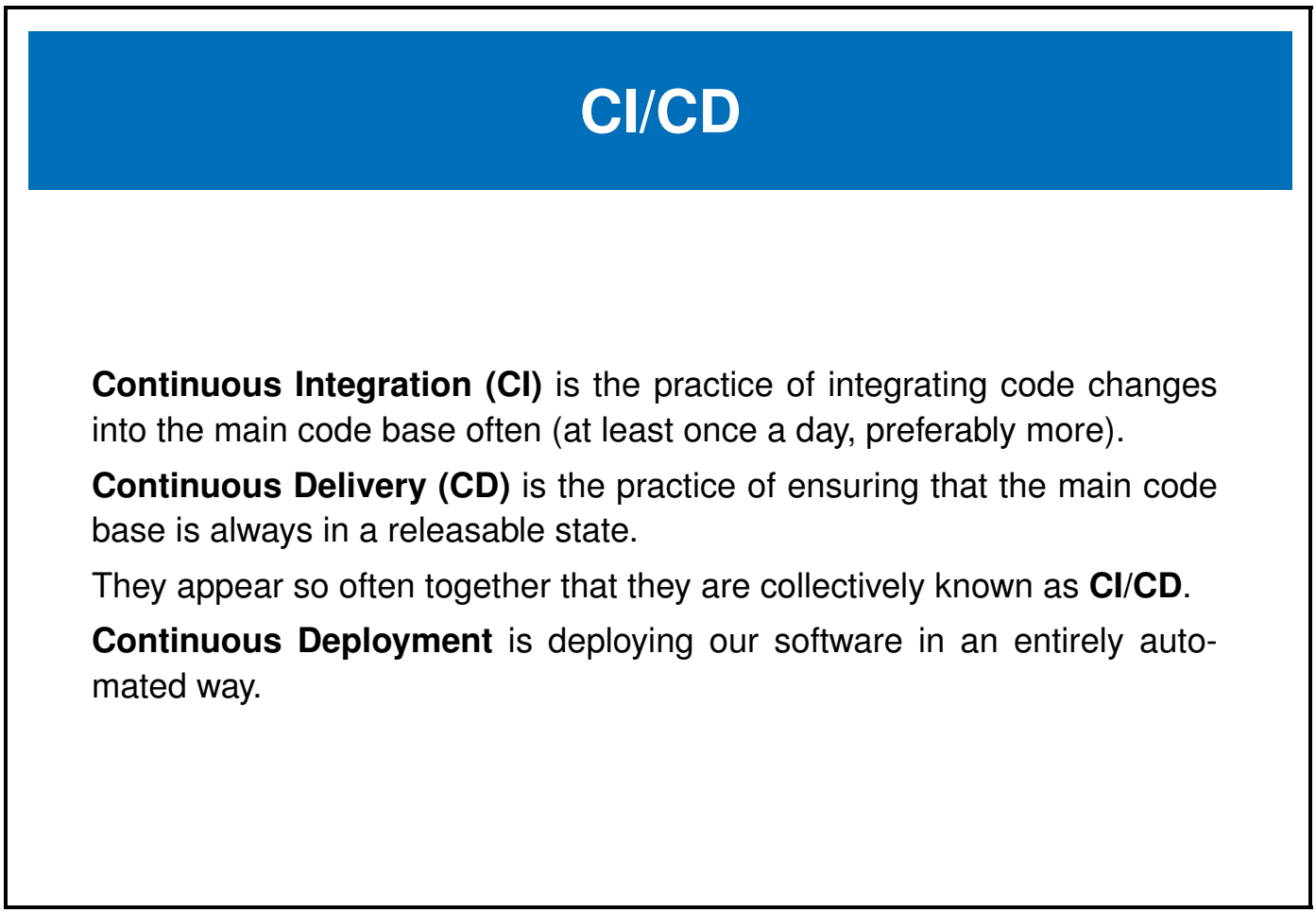
A good, face value definition of DevOps is: A cultural philosophy that primarily values radical cooperation between the development team and the operations team to facilitate rapid and stable product deployment.

DevOps has evolved to become much more and a more complete definition these days would be something like:

A cultural philosophy that values radical cooperation between all entities in an organization to facilitate the rapid and efficient accomplishment of the organization's mission.

There are diverse ways to implement DevOps, and nearly every organization will have its own idiosyncrasies. The universal commonality is that an organization will align the software development process with the organization's mission through radical team cooperation. Organizational missions are as varied as the number of organizations, so there is no one-size-fits-all implementation of DevOps.

16.5 DevOps Methodologies



Automation and high velocity of software development are important to a DevOps culture. CI/CD facilitates automation and allows for safe and reliable yet rapid software development. CI/CD is, therefore, an almost universal practice in a DevOps culture.

When Continuous Integration is used properly, much of the opportunity for bugs to sneak into our code base is removed. This is because one of the goals of Continuous Integration as practiced, is to automate as much of the development process as possible, thereby removing as much opportunity for human error as is possible.

Continuous Delivery requires that our software always be in a releasable state. Conforming to this practice forces us into better software development practices. This is because, in order for our software to always be in a releasable state without devolving into chaos, we must be practicing Continuous Integration and other practices that remove inefficiencies in the software development and delivery process.

Continuous Deployment means that the process of deploying code to users is automated and occurs with no prior review. Once an engineer integrates changes, those changes go through the CI/CD pipeline, and the code is immediately and automatically deployed. Continuous Deployment requires total trust that your processes will catch any issues before the code is deployed and that the code will not deploy until those issues are resolved.

16.6 DevOps tools

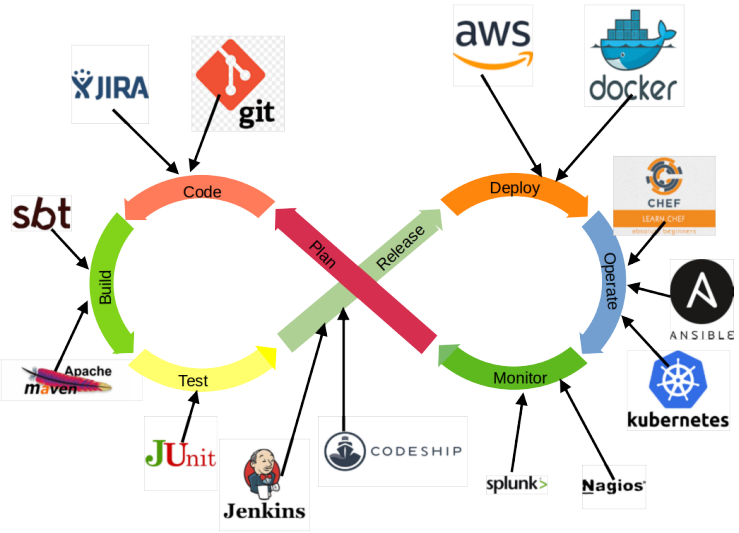


Figure 16.1: **DevOps and tools**

When implementing DevOps, any tool that facilitates the merging of operations and development is a valid option. There are many options for tools that can be used.

The figure shows a small sample of tools that can be useful when implementing DevOps.

GitOps and DevOps

GitOps is **Infrastructure-as-code** using **Git** as the source of truth for a software system's infrastructure.

- Devops vs GitOps
  - DevOps is about the culture shift and collaboration
  - GitOps is about tools and tool framework to support DevOps initiatives
  - DevOps is a culture from which policy flows
  - GitOps focuses on specific practices
    - Infrastructure-as-code (**IaC**)
    - A Git repository is the source of truth for infrastructure
    - Merge Requests are the mechanism of change management (**MR**s)
    - Continuous Integration and Delivery (**CI**/**CD**)

- **Infrastructure-as-code, (IaC)**

  Consists of managing and provisioning compute resources as if they were application code. This means the configuration and initialization of resources using configuration files to automate the deployment function.

- **Merge Requests, (MR's)**

  The merging of requests for the resource configurations to keep the infrastructure up to date with current requirements.

- **Continuous Integration and Continuous Delivery (CI/CD)**

  Change management is implemented in smaller and more frequent elements to avoid large time consuming changes.

DevOps: Additional information

DevOps normally requires a large change to the business process, requiring an understanding of not only the concepts, but also how to select the components that make up a company tailored **DevOps** implementation.

Some additional information and training offerings can be found at:

- https://training.linuxfoundation.org/devops-site-reliability/
- https://www.edx.org/school/linuxfoundationx

Chapter 17

Closing and Evaluation Survey

17.1 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.

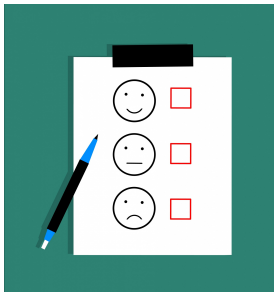


Figure 17.1: **Course Survey**