Introduction to Linux

LFS101x

2014

Chapter 1: The Linux Foundation

Introduction/Learning Objectives

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

- Understand the role of the Linux Foundation.
- Appreciate the learning opportunities provided by the Linux Foundation's training program.
- Describe the software environment required for this course.
- Describe the three major Linux distribution families.

Section 1: Linux Foundation

About the Linux Foundation

Since its inception in 1991, Linux has grown to become a major force in computing - powering everything from the New York Stock Exchange, to mobile phones, supercomputers, and consumer devices.

The **Linux Foundation** is a nonprofit organization that sponsors the work of Linux creator Linus Torvalds. It was founded in 2000 and its mission is to promote, protect, and advance Linux. The Linux Foundation is supported by leading technology companies and thousands of individual members from around the world and marshalls the resources of its members and the open source development community to ensure that Linux remains free and technically advanced.

The Linux Foundation is active on many fronts. In addition to its basic missions of protecting, promoting and advancing Linux, the Foundation:

- Produces technical events throughout the world (more on this on the following screen).
- Develops and delivers training programs.
- Hosts major collaborative projects and industry initiatives (Click here to view a video).
- Manages kernel.org where the official versions of the Linux kernel are released.
- Runs the popular website linux.com.

Linux Foundation Events

The Linux Foundation hosts conferences and other events throughout the world which bring community members together in person. These events:

- Provide an open forum for development of the next kernel (the actual operating system) release.
- Bring together developers and system administrators to solve problems in a real-time environment.
- Host workgroups and community groups for active discussions.
- Connect end users, system administrators, and kernel developers in order to grow Linux use in the enterprise.
- Encourage collaboration among the entire community.
- Provide an atmosphere that is unmatched in its ability to further the platform.

What are the objectives of the Linux Foundation?

- [x] To promote Linux and provide neutral collaboration and education.
- [x] To improve Linux as a technical platform.
- [x] To sponsor the work of Linus Torvalds.
- [ ] To manage Linux and UNIX servers on a commercial basis.

Section 2: Linux Foundation Training

Training That Builds Skills

Linux Foundation training is for the community and is designed by members of the community. The System Administration courses focus on Enterprise Linux environments and target system administrators, technical support, and architects. The developer courses feature instructors and content straight from the leaders of the Linux developer community.

Linux Foundation training is distribution-flexible, technically advanced, and created with the actual leaders of the Linux development community themselves. Most courses are more than 50% focused on hands on labs and activities to build real world skills.

Copyright

The Linux Foundation has a copyright on all the materials in this course.

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

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

Section 3: Course Linux Requirements

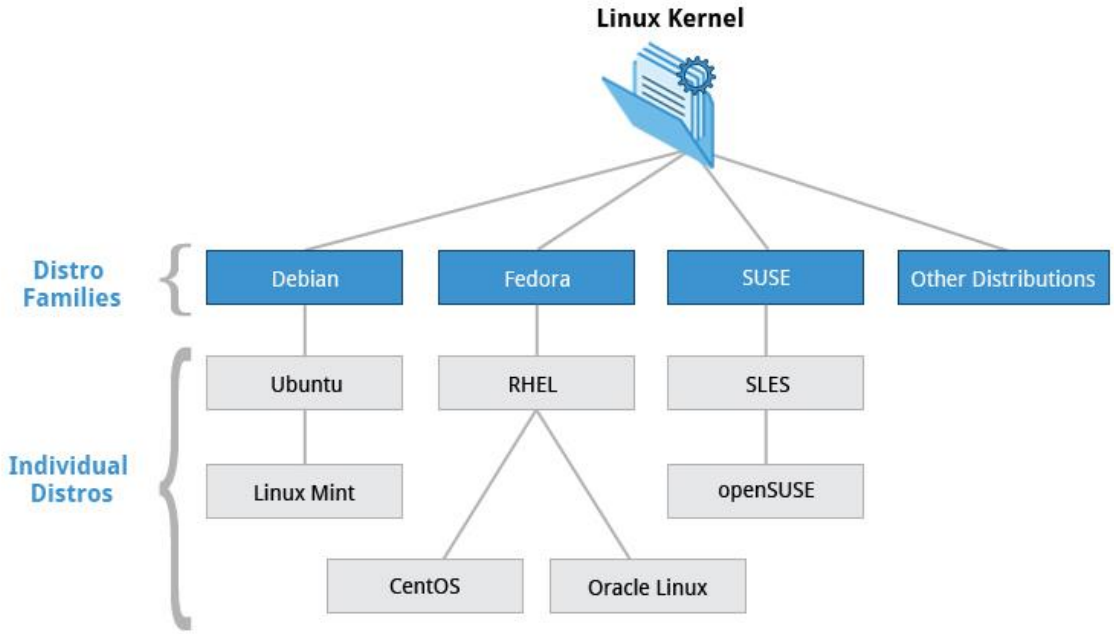
Course Software Requirements

In order to fully benefit from this course, you'll need to have at least one **distribution** of Linux installed (if you're not already familiar with the term "distribution" as it relates to Linux you soon will be!) In the next screen, you will learn some more details about the many available Linux distributions and distribution families. Because there are literally hundreds of distributions, we couldn't possibly cover them all in this course. Instead we have decided to focus on the three major distribution families, and we've chosen one specific distribution from within each family to use for all illustrations, examples, and exercises. This is not meant to suggest that we endorse these specific distributions; they were simply chosen because they are fairly widely used and are broadly representative of their respective family.

The families and representative distributions we are using are:

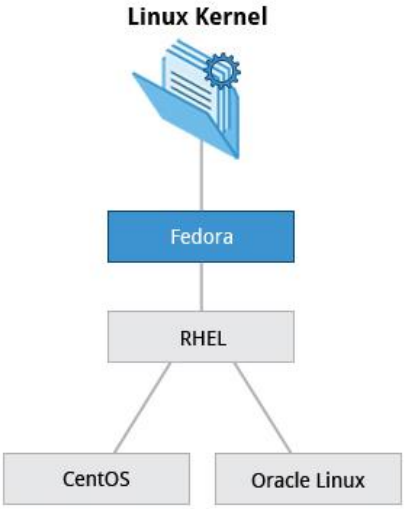
1. **Debian Family Systems (such as Ubuntu)**
2. **SUSE Family Systems (such as openSUSE)**
3. **Fedora Family Systems (such as CentOS)**

Focus on Three Major Linux Distribution Families



In the next chapter you will learn about the components that make up a Linux distribution. For now, what you need to know is that this course focuses on the three major Linux distribution families that currently exist. However, as long as there are talented contributors, the families of distributions and the distributions within these families will continue to change and grow. People see a need and develop special configurations and utilities to respond to that need. Sometimes that effort creates a whole new distribution of Linux. Sometimes that effort will leverage an existing distribution to expand the members of an existing family. There is an active discussion thread on Linux distributions on the linux.com website.

Fedora Family



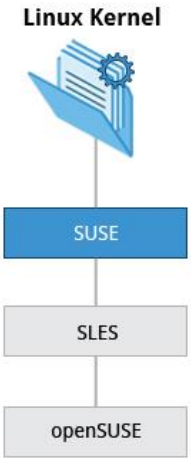
Fedora is the community distribution that forms the basis of **Red Hat Enterprise Linux (RHEL)**, **CentOS**, **Scientific Linux**, and **Oracle Linux**. **Fedora** contains significantly more software than Red Hat's enterprise version. One reason for this is that a diverse community is involved in building **Fedora**; it is not just a company. In this course, **CentOS** is used for activities, demos, and labs because it is available at no cost to the end user and has a much longer release cycle than **Fedora** (which typically releases a new version every six months or so).

For this reason, we have standardized the **Fedora** part of this course material on **CentOS** 6.5. Once installed, **CentOS** is also virtually identical to **Red Hat Enterprise Linux**, which is the most popular Linux distribution in enterprise environments.

Key Facts About the Fedora Family

- The **Fedora** family is upstream for **CentOS**, **RHEL**, and **Oracle Linux**.
- The Linux kernel 2.6.32 is used in **RHEL/CentOS 6.x**
- It supports hardware platforms such as x86, x86-64, Itanium, PowerPC, and IBM System z.
- It uses the **RPM**-based **yum** package manager (we cover in more detail later) to install, update, and remove packages in the system.
- **RHEL** is widely used by enterprises which host their own systems.

SUSE Family

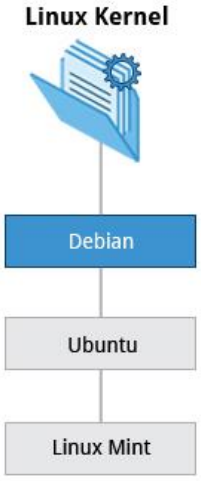


The relationship between **SUSE, SUSE Linux Enterprise Server (SLES),** and **openSUSE** is similar to the one described between **Fedora, Red Hat Enterprise Linux,** and **CentOS**. In this case, we decided to use **openSUSE** 12.3 as the reference distribution for the **SUSE** family as it is available to end users at no cost. The two products are extremely similar, and material that covers **openSUSE** can typically be applied to **SLES** with no problem.

Key Facts About the SUSE Family

- **SUSE Linux Enterprise Server (SLES)** is upstream for **openSUSE**.
- The Linux kernel 3.11 is used in **openSUSE 12.3**.
- It uses the **RPM**-based **zypper** package manager (we cover in more detail later) to install, update, and remove packages in the system.
- It includes the **YaST** (Yet another System Tool) application for system administration purposes.
- **SUSE** is widely used in the retail sector.

Debian Family



The **Debian** distribution is upstream for several other distributions including **Ubuntu**, and **Ubuntu** is upstream for **Linux Mint** and others. It is commonly used on both servers and desktop computers. **Debian** is a pure open source project and focuses on one key aspect, that is, 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** stable branch, **Ubuntu** also has access to a very large software repository. For those reasons, we decided to use **Ubuntu** 14.04 LTS (Long Term Support) as the reference **Debian family** distribution for this course. **Ubuntu** is a registered trademark of Canonical Ltd. and is used throughout this course with their permission.

Key Facts About the Debian Family

- The **Debian** family is upstream for **Ubuntu**, and **Ubuntu** is upstream for **Linux Mint** and others.
- The Linux kernel 3.13 is used in **Ubuntu 14.04**.
- It uses the **DPKG**-based **apt-get** package manager (we cover in more detail later) to install, update, and remove packages in the system.
- **Ubuntu** has been widely used for cloud deployments.
- While **Ubuntu** is built on top of **Debian**, it uses the **Unity** graphical interface, is **GNOME-**based and differs quite a bit visually from the interface on standard **Debian** as well as other distributions.

More About the Software Environment

The material produced by the Linux Foundation is distribution-flexible. This means that technical explanations, labs, and procedures should work on most modern distributions. While choosing a Linux distribution system, you will notice that the 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.

The desktop environment used for this course is **GNOME**. As you'll see in Chapter 4, there are different environments, but we selected **GNOME** as it is the most popular

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- The Linux Foundation is a nonprofit consortium dedicated to fostering the growth of Linux.
- The Linux Foundation training is for the community and by the community. Linux training is distribution-flexible, technically advanced, and created with the leaders of the Linux development community.
- There are three major distribution families within Linux: **Fedora, SUSE** and **Debian**. In this course we will work with representative members of all of these families throughout.

Chapter 2: Linux Philosophy & Concepts

Learning Objectives

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

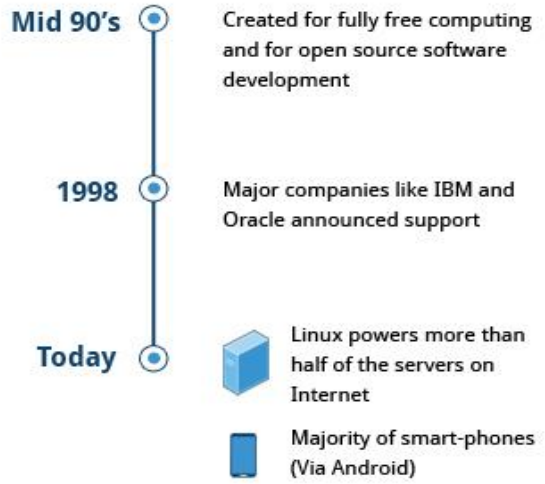
- Understand the history and philosophy of Linux.
- Describe the Linux community.
- Define the common terms associated with Linux.
- Understand the components of a Linux distribution.

Section 1: Linux History

About Linux History

The Linux distributions created in the mid-90's provided the basis for fully free computing and became a driving force in the open source software movement. In 1998, major companies like **IBM** and **Oracle** announced support for the Linux platform and began major development efforts as well.

Today, Linux powers more than half of the servers on the Internet, the majority of smart-phones (via the **Android** system which is built on top of Linux), and nearly all of the world's most powerful supercomputers.



Section 2: Linux Philosophy

Every organization or project has a philosophy that works as a guide while framing its objectives and delineating its growth path. This section contains a description of the Linux philosophy and how this philosophy has impacted its development.

Linux is constantly enhanced and maintained by a network of developers from all over the world collaborating over the Internet, with Linus Torvalds at the head. Technical skill and a desire to contribute are the only qualifications for participating.

Section 3: Linux Community

Suppose as part of your job you need to configure a Linux file server, and you run into some difficulties. If you're not able to figure out the answer yourself or get help from a co-worker, the Linux community might just save the day! There are many ways to engage with the Linux community: you can post queries on relevant discussion forums, subscribe to discussion threads, and even join local Linux groups that meet in your area.

The Linux community is a far-reaching ecosystem consisting of developers, system administrators, users and vendors, who use many different forums to connect with one another. Among the most popular are:

- Linux User Groups (both local and online)
- **I**nternet **R**elay **C**hat (**IRC**) software (such as **Pidgin** and **XChat**)
- Online communities and discussion boards
- Newsgroups and mailing lists
- Community events (such as LinuxCon and ApacheCon)

One of the most powerful online user communities is linux.com. This site is hosted by the **Linux Foundation** and serves over one million unique visitors every month. It has active sections on:

- News
- Community discussion threads
- Free tutorials and user tips

Section 4: Linux Terminology

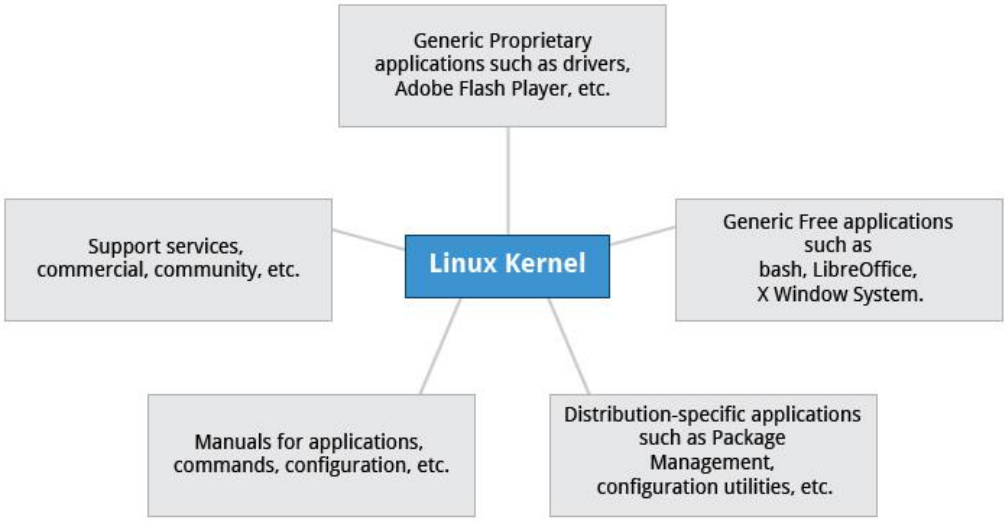
When you start exploring Linux, you'll soon come across some unfamiliar terms like distribution, boot loader, desktop environment, etc. So let's stop and take a look at some basic terminology used in Linux to help you get up to speed before we proceed further.

What is a kernel?

- (•) The glue between hardware and applications.
- ( ) A collection of software making up a Linux-based OS.
- ( ) A program that runs as a background process.
- ( ) A graphical subsystem on nearly all Linux systems.

Section 5: Linux Distributions

Suppose you have been assigned to a project building a product for a Linux platform. Project requirements include making sure the project works properly on the most widely used Linux distributions. To accomplish this you need to learn about the different components, services and configurations associated with each distribution. We're about to look at how you'd go about doing exactly that.



So, what is a Linux distribution and how does it relate to the Linux kernel?

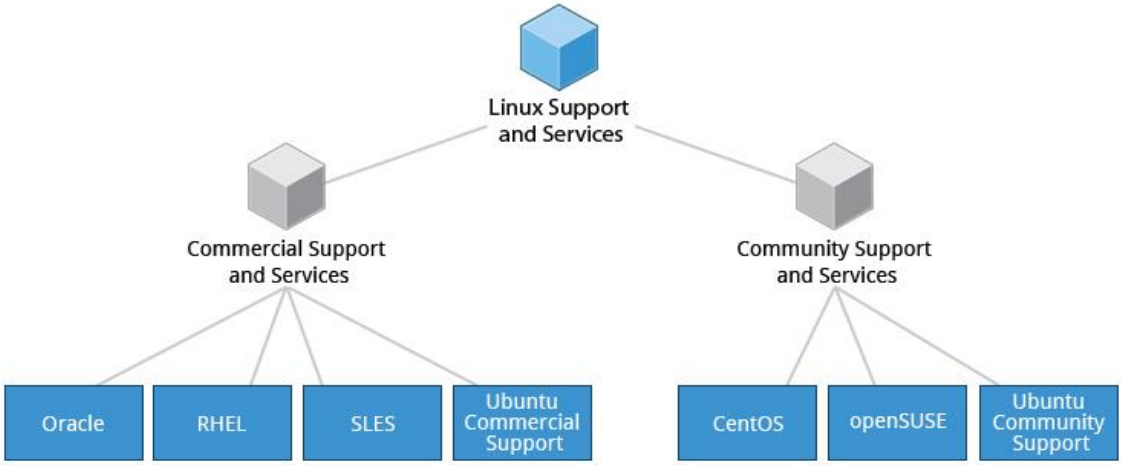
As illustrated above, the Linux kernel is the core of a computer operating system. A full **Linux distribution** consists of the kernel plus a number of other software tools for file-related operations, user management, and software package management. Each of these tools provides a small part of the complete system. Each tool is often its own separate project, with its own developers working to perfect that piece of the system.

As mentioned earlier, the current Linux kernel, along with past Linux kernels (as well as earlier release versions) can be found at the www.kernel.org web site. The various Linux distributions may be based on different kernel versions. For example, the very popular **RHEL 6** distribution is based on the 2.6.32 version of

the Linux kernel, which is rather old but extremely stable. Other distributions may move more quickly in adopting the latest kernel releases. It is important to note that the kernel is not an all or nothing proposition, for example, **RHEL 6** has incorporated many of the more recent kernel improvements into their version of 2.6.32.

Examples of other essential tools and ingredients provided by distributions include the **C/C++** compiler, the **gdb** debugger, the core system libraries applications need to link with in order to run, the low-level interface for drawing graphics on the screen as well as the higher-level desktop environment, and the system for installing and updating the various components including the kernel itself.

Services Associated with Distributions



A vast variety of Linux distributions cater to different audiences and organizations depending on their specific needs. Large commercial organizations tend to favor the commercially supported distributions from **Red Hat, SUSE** and Canonical **(Ubuntu)**.

CentOS is a popular free alternative to **Red Hat Enterprise Linux (RHEL)**. **Ubuntu** and **Fedora** are popular in the educational realm. **Scientific Linux** is favored by the scientific research community for its compatibility with scientific and mathematical software packages. Both **CentOS** and **Scientific Linux** are binary-compatible with **RHEL**; i.e., binary software packages in most cases will install properly across the distributions.

Many commercial distributors, including **Red Hat**, **Ubuntu**, **SUSE**, and **Oracle**, provide long term fee-based support for their distributions, as well as hardware and software certification. All major distributors provide update services for keeping your system primed with the latest security and bug fixes, and performance enhancements, as well as provide online support resources.

In addition to the kernel, what is the purpose of the other software tools required for developing a full Linux distribution?

- [x] Perform file-related operations
- [ ] Cater to different audiences
- [x] Maintain user management
- [x] Software package management

Which of the following are binary-compatible Linux distributions which are free alternatives to Red Hat Enterprise Linux (RHEL). Choose all that apply.

- [x] CentOS
- [ ] Ubuntu
- [x] Scientific Linux
- [ ] openSUSE

Summary

You have completed this chapter. Let's summarize the key concepts covered.

- Linux borrows heavily from the **UNIX** operating system, with which its creators were well versed.
- Linux accesses many features and services through files and file-like objects.
- Linux is a fully multitasking, multiuser operating system, with built-in networking and service processes known as daemons.
- Linux is developed by a loose confederation of developers from all over the world, collaborating over the Internet, with Linus Torvalds at the head. Technical skill and a desire to contribute are the only qualifications for participating.
- The Linux community is a far reaching ecosystem of developers, vendors, and users that supports and advances the Linux operating system.
- Some of the common terms used in Linux are: Kernel, Distribution, Boot loader, Service, Filesystem, X Window system, desktop environment, and command line.
- A full Linux distribution consists of the kernel plus a number of other software tools for file-related operations, user management, and software package management.

Chapter 3: Linux Structure and Installation

Learning Objectives

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

- Identify Linux filesystems.
- Identify the differences between partitions and filesystems.
- Describe the boot process.
- Know how to install Linux on a computer.

Section 1: Linux Filesystem Basics

Linux Filesystems

Think of a refrigerator that has multiple shelves that can be used for storing various items. These shelves help you organize the grocery items by shape, size, type, etc. The same concept applies to a **filesystem**, which is the embodiment of a method of storing and organizing arbitrary collections of data in a human-usable form.

Different Types of Filesystems Supported by Linux:

- Conventional disk filesystems: ext2, ext3, ext4, XFS, Btrfs, JFS, NTFS, etc.
- Flash storage filesystems: ubifs, JFFS2, YAFFS, etc.
- Database filesystems
- Special purpose filesystems: procfs, sysfs, tmpfs, debugfs, etc.

This section will describe the standard filesystem layout shared by most Linux distributions.

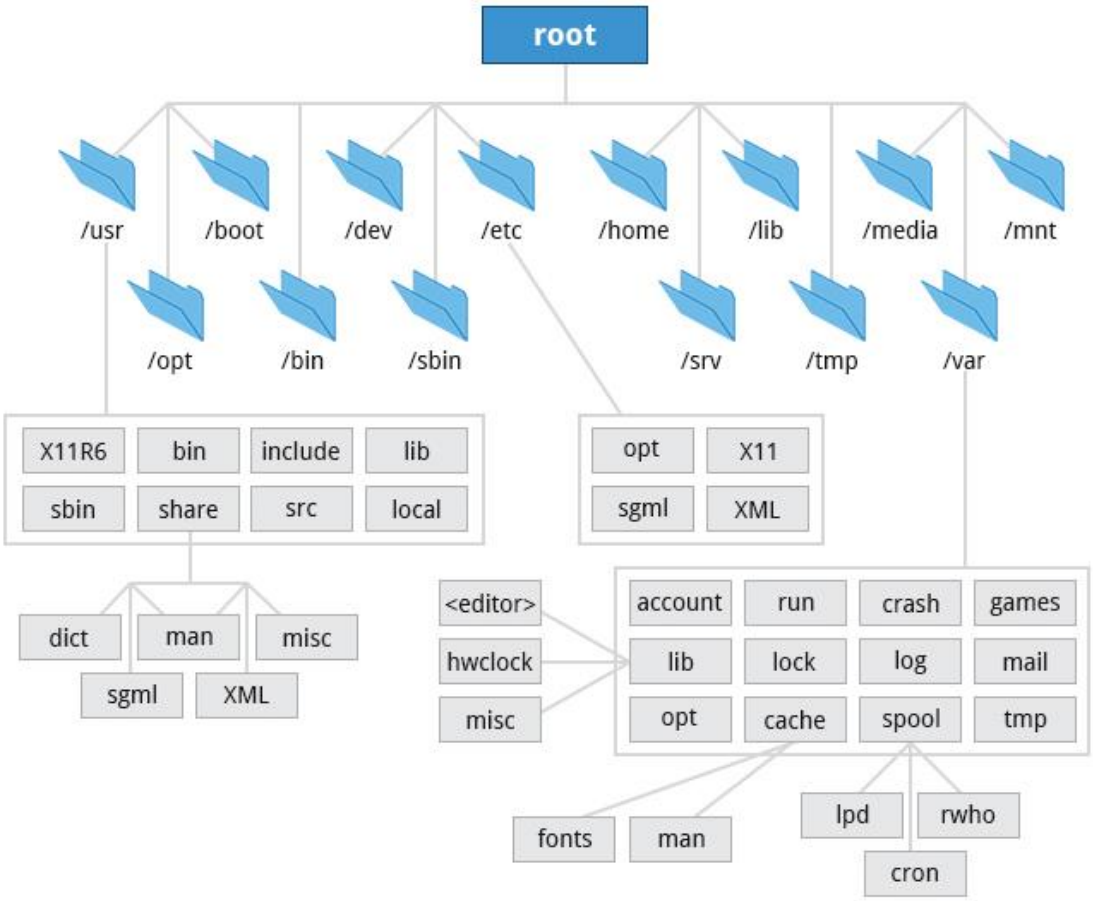
Partitions and Filesystems

A **partition** is a logical part of the disk, whereas a **filesystem** is a method of storing/finding files on a hard disk (usually in a partition). By way of analogy, you can think of filesystems as being like family trees that show descendants and their relationships, while the partitions are like different families (each of which has its own tree).

A comparison between filesystems in Windows and Linux is given in the following table:

	Windows	Linux
Partition	Disk1	/dev/sda1
Filesystem type	NTFS/FAT32	EXT3/EXT4/XFS...
Mounting Parameters	DriveLetter	MountPoint
Base Folder where OS is stored	C drive	/

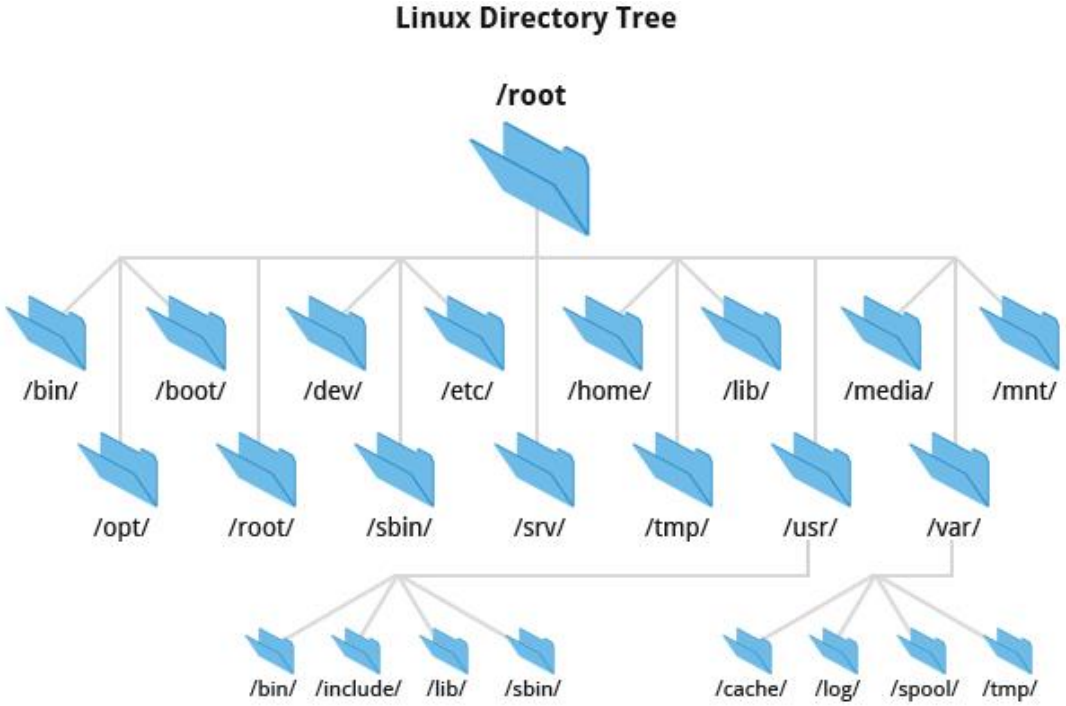
The Filesystem Hierarchy Standard



Linux systems store their important files according to a standard layout called the **Filesystem Hierarchy Standard**, or **FHS**. You can download a document that provides much greater detail here, or look up the original source documents on the Linux Foundation website. 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 have drive letters. New drives are mounted as directories in the single filesystem, often under /media (so, for example, a CD-ROM disc labeled **FEDORA** might end up being found at /media/FEDORA, and a file **README.txt** on that disc would be at /media/FEDORA/README.txt).

More About the Filesystem Hierarchy Standard



All Linux filesystem names are case-sensitive, so /boot, /Boot, and /BOOT represent three different directories (or folders). Many distributions distinguish between core utilities needed for proper system operation and other programs, and place the latter in directories under /usr (think "**user**"). To get a sense for how the other programs are organized, find the /usr directory in the diagram above and compare the subdirectories with those that exist directly under the system root directory (/).

Note: The next few screens cover the demonstrations and Try-It-Yourself activities for all three Linux distributions. You can view a demonstration for the distribution type of your choice and practice the procedure through the relevant Try-It-Yourself activity.

https://www.youtube.com/watch?v=4HZIRqah8bk

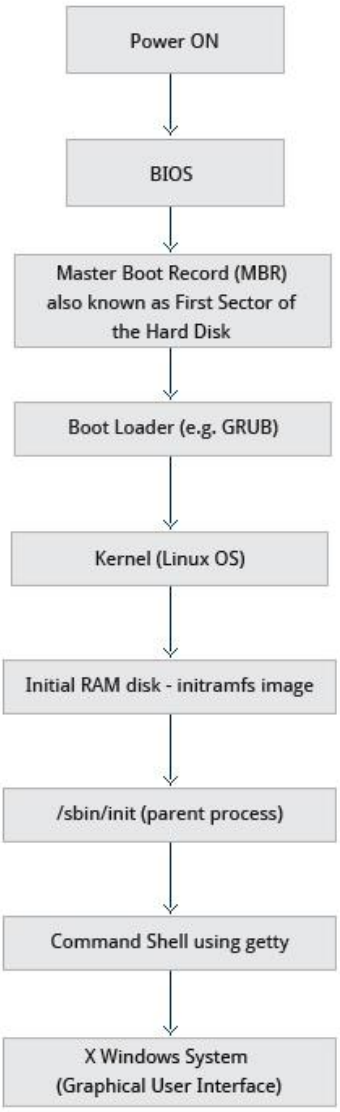
Section 2: Boot Process

The Boot Process

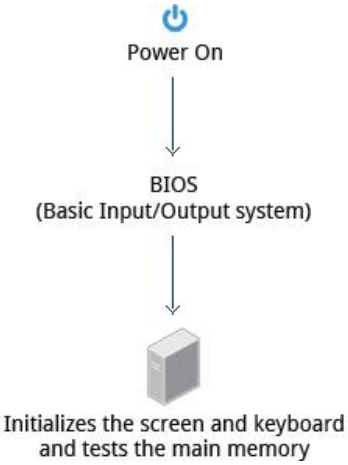
Have you ever wondered what happens in the background from the time you press the **Power** button until the Linux login prompt appears?

The Linux **boot process** is the procedure for initializing the system. It consists of everything that happens from when the computer power is first switched on until the user interface is fully operational.

Once you start using Linux, you will find that having a good understanding of the steps in the boot process will help you with troubleshooting problems as well as with tailoring the computer's performance to your needs.



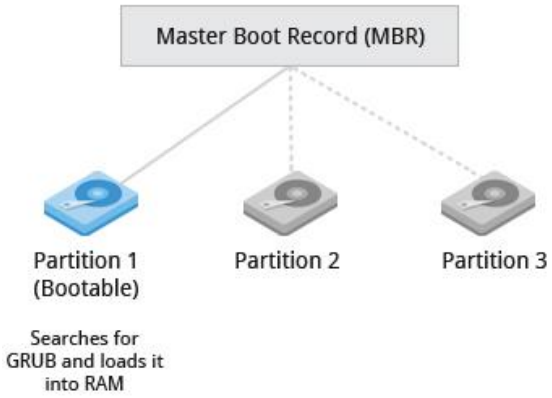
BIOS - The First Step



Starting an **x86-**based Linux system involves a number of steps. When the computer is powered on, the **Basic Input/Output System** (**BIOS)** initializes the hardware, including the screen and keyboard, and tests the main memory. This process is also called **POST** (**Power On Self Test**).

The BIOS software is stored on a ROM chip on the motherboard. After this, the remainder of the boot process is completely controlled by the operating system

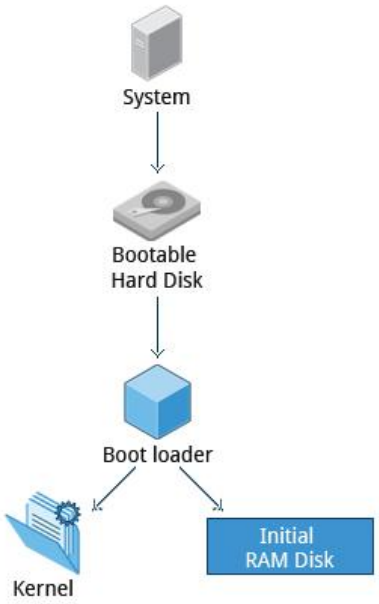
Master Boot Records (MBR) and Boot Loader



Once the **POST** is completed, the system control passes from the **BIOS** to the boot loader. 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 more recent **(Unified) Extensible Firmware Interface** or **EFI/UEFI** systems). Up to this stage, the machine does not access any mass storage media. Thereafter, information on the date, time, and the most important peripherals are loaded from the **CMOS values** (after a technology used for the battery-powered memory store - which allows the system to keep track of the date and time even when it is powered off).

A number of boot loaders exist for Linux; the most common ones are **GRUB** (for **GRand Unified Boot loader**) and **ISOLINUX** (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 (which contains some critical files and device drivers needed to start the system) into memory.

Boot Loader in Action



The boot loader has two distinct stages:

First Stage:

For systems using the BIOS/MBR method, the boot loader resides at the first sector of the hard disk also known as the **Master Boot Record** (**MBR**). The size of the MBR is just 512 bytes. In this stage, the boot loader examines the partition table and finds a bootable partition. Once it finds a bootable partition, it then searches for the second stage boot loader e.g, **GRUB**, and loads it into **RAM** (**Random Access Memory).**

For systems using the **EFI/UEFI** method, **UEFI firmware** reads its **Boot Manager** data to determine which **UEFI** application is to be launched and from where (i.e., from which disk and partition the **EFI** partition can be found). The firmware then launches the **UEFI** application, for example, **GRUB**, as defined in the boot entry in the firmware's boot manager. This procedure is more complicated but more versatile than the older MBR methods.

Second Stage:

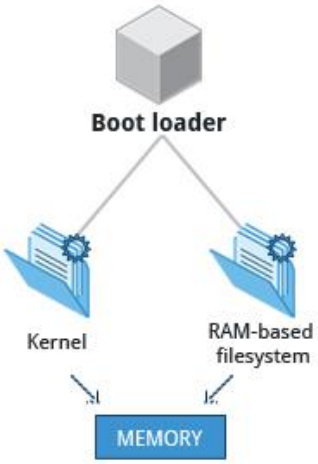
The second stage boot loader resides under /boot. A **splash screen** is displayed which allows us to choose which Operating System (OS) to boot. After choosing the OS, the boot loader loads the kernel of the selected operating system into RAM and passes control to it.

The boot loader loads the selected kernel image (in the case of Linux) and passes control to it. Kernels are almost always compressed, so its first job is to uncompress itself. After this, it will check and analyze the system hardware and initialize any hardware device drivers built into the kernel.

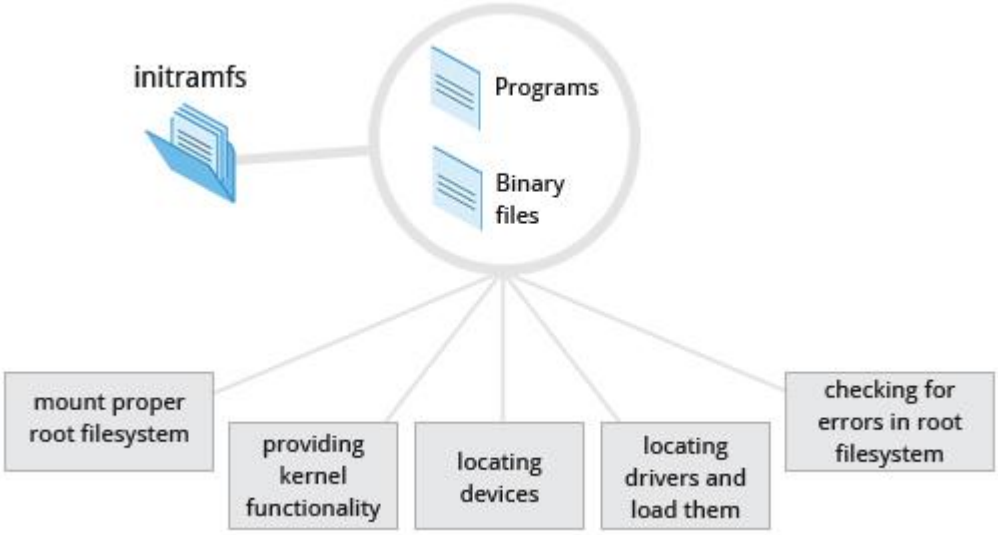
The Linux Kernel

The boot loader loads both the kernel and an initial RAM–based file system (**initramfs**) into memory so it can be used directly by the kernel.

When the kernel is loaded in RAM, it immediately initializes and configures the computer's memory and also configures all the hardware attached to the system. This includes all processors, I/O subsystems, storage devices, etc. The kernel also loads some necessary user space applications.



The Initial RAM Disk

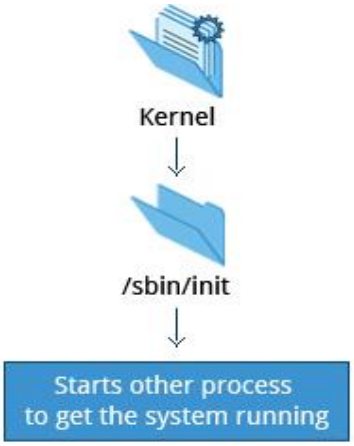


The **initramfs** filesystem image contains programs and binary files that perform all actions needed to mount the proper root filesystem, like providing kernel functionality for the needed filesystem and device drivers for mass storage controllers with a facility called **udev** (for **U**ser **Dev**ice) which is responsible for figuring out which devices are present, locating the **drivers** they need to operate properly, and loading them. After the root filesystem has been found, it is checked for errors and mounted.

The **mount** program instructs the operating system that a filesystem is ready for use, and associates it with a particular point in the overall hierarchy of the filesystem (the **mount point**). If this is successful, the **initramfs** is cleared from RAM and the **init** program on the root filesystem (/sbin/init) is executed.

init handles the mounting and pivoting over to the final real root filesystem. If special hardware drivers are needed before the mass storage can be accessed, they must be in the **initramfs** image.

/sbin/init and Services

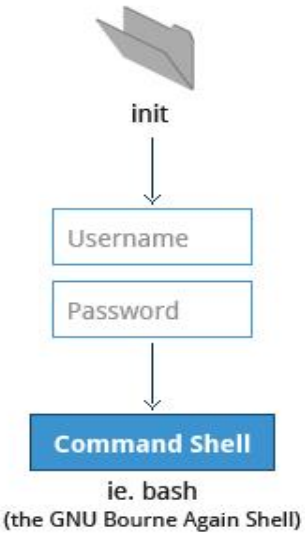


Once the kernel has set up all its hardware and mounted the root filesystem, the kernel runs the /sbin/init program. This then becomes the initial process, which then starts other processes to get the system running. Most other processes on the system trace their origin ultimately to **init**; the exceptions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

Traditionally, this process startup was done using conventions that date back to **System V UNIX**, with the system passing through a sequence of **runlevels** containing collections of scripts that start and stop services. Each runlevel supports a different mode of running the system. Within each runlevel, individual services can be set to run, or to be shut down if running. Newer distributions are moving away from the System V standard, but usually support the System V conventions for compatibility purposes.

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.

Text-Mode Login

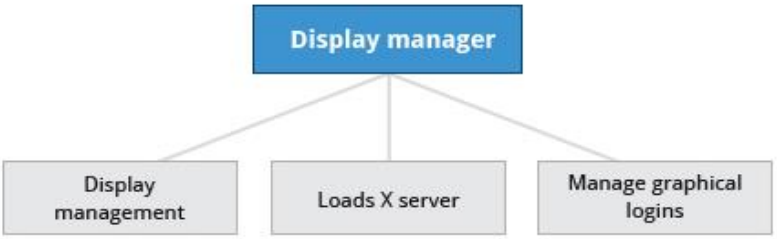


Near the end of the boot process, **init** starts a number of text-mode login prompts (done by a program called **getty**). These enable you to type your username, followed by your password, and to eventually get a command shell.

Usually, 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.

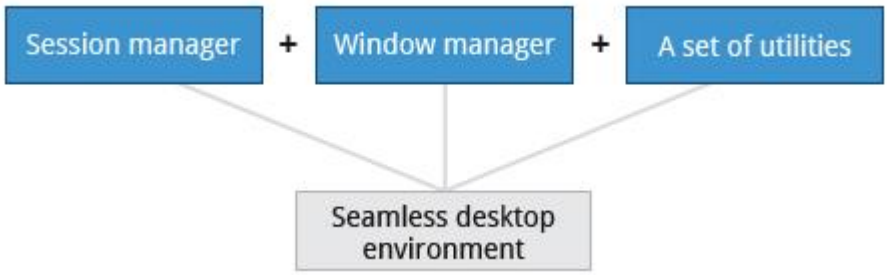
As you'll learn in the chapter 'Command Line Operations', the terminals which run the command shells can be accessed using the **ALT** key plus a **function** key. Most distributions start six text terminals and one graphics terminal starting with **F1** or **F2**. If the graphical environment is also started, switching to a text console requires pressing **CTRL-ALT +** the appropriate function key (with **F7** or **F1** being the GUI). As you'll see shortly, you may need to run the **startx** command in order to start or restart your graphical desktop after you have been in pure text mode.

X Window System



Generally, in a Linux desktop system, 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 applications, 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.

_____ is responsible for starting system and network services at boot time.

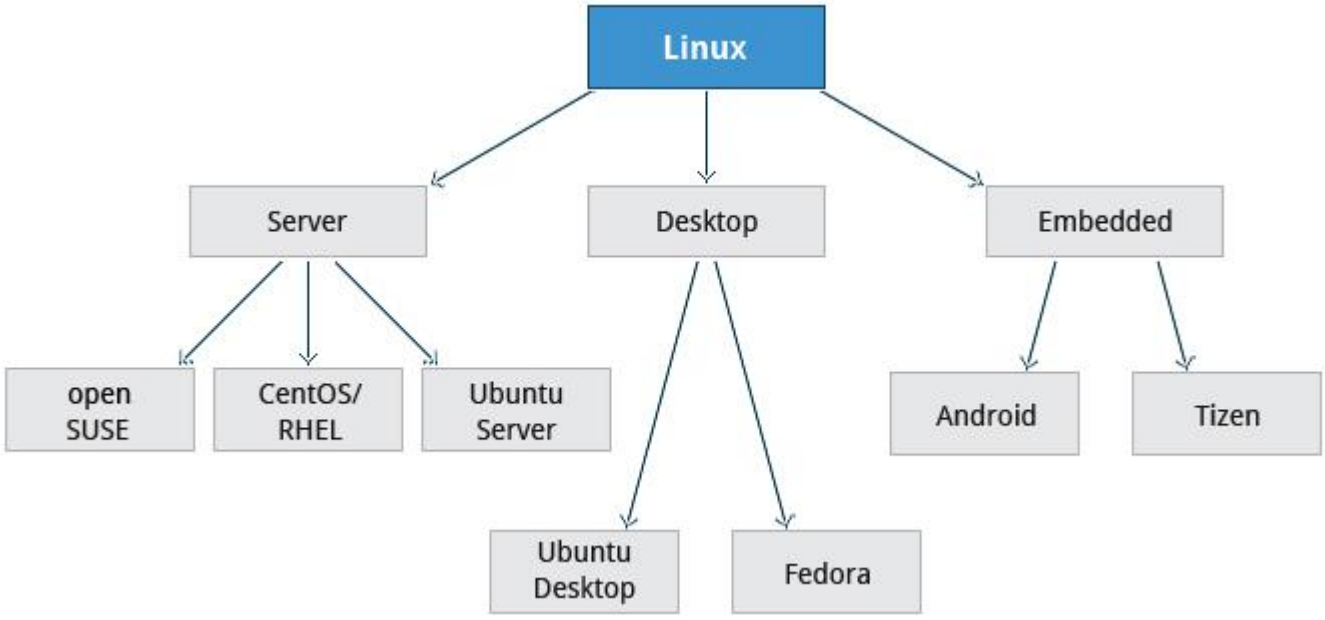
kernel boot loader **init** GRUB

Which one of the following component actually loads Linux?

- Boot loader Boot loader
- **init**
- X Window System
- BIOS

Section 3: Linux Distribution Installation

Choosing a Linux Distribution



Your family is planning to buy its first car. What are the factors you need to consider while purchasing a car? Your planning depends a lot on your requirements. For instance, your budget, available finances, size of the car, type of engine, after-sales services, etc.

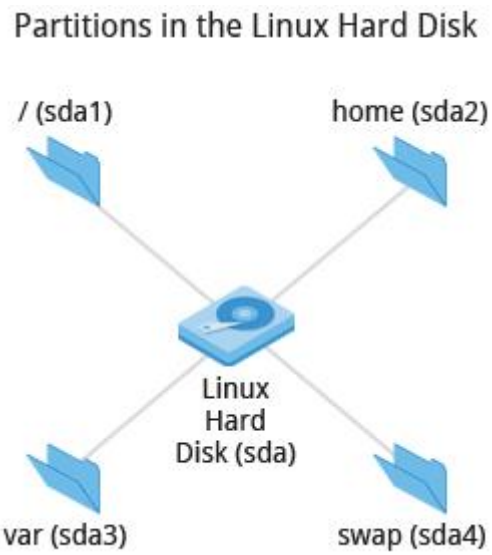
Similarly, determining which distribution to deploy also requires some planning. The figure shows some but not all choices, as there are other choices for distributions and standard embedded Linux systems are mostly neither **Android** or **Tizen**, but are slimmed down standard distributions.

Some questions worth thinking about before deciding on a distribution include:

- What is the main function of the system? (server or desktop)
- What types of packages are important to the organization? For example, web server, word processing, etc.
- How much hard disk space is available? For example, when installing Linux on an embedded device, there will be space limitations.
- How often are packages updated?
- How long is the support cycle for each release? For example, LTS releases have long term support.
- Do you need kernel customization from the vendor?
- What hardware are you running the Linux distribution on? For example, **X86, ARM, PPC**, etc.
- Do you need long-term stability or short-term experimental software?

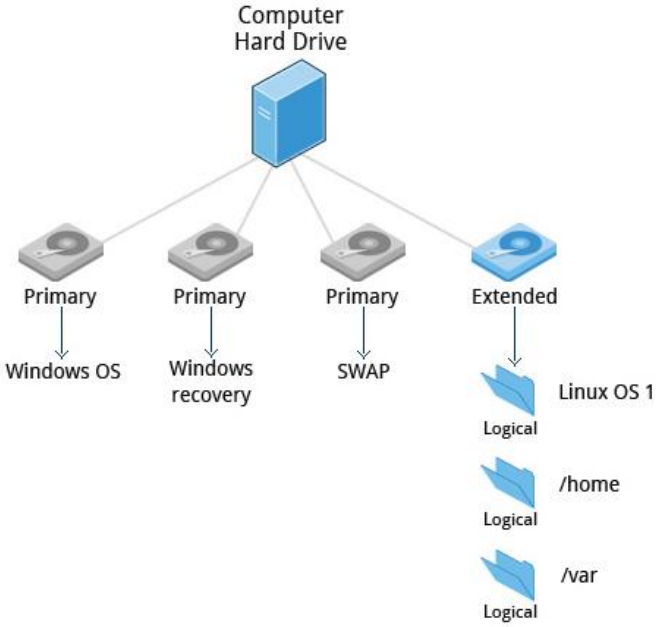
Linux Installation: Planning

A **partition** layout needs to be decided at the time of installation because Linux systems handle partitions by mounting them at specific points in the filesystem. You can always modify the design later, but it always easier to try and get it right to begin with.



Nearly all installers provide a reasonable filesystem layout by default, with either all space dedicated to normal files on one big partition and a smaller **swap** partitition, or with separate partitions for some space-sensitive areas like /home and /var. You may need to override the defaults and do something different if you have special needs, or if you want to use more than one disk.

All installations include the bare minimum software for running a Linux distribution.



Most installers also provide options for adding categories of software. Common applications (such as the **Firefox** web browser and **LibreOffice** office suite), developer tools (like the **vi** and **emacs** text editors which we will explore later in this course), and other popular services, (such as the **Apache** web server tools or **MySQL** database) are usually included. In addition, a desktop environment is installed by default.

All installers secure the system being installed as part of the installation. Usually, this consists of setting the password for the superuser (**root**) and setting up an initial user. In some cases (such as **Ubuntu**), only an initial user is set up; direct root login is disabled and root access requires logging in first as a normal user and then using **sudo** as we will describe later. Some distributions will also install more advanced security frameworks, such as **SELinux** or **AppArmor.**

You have just brought home a new set top box for your television that includes a digital video recorder (DVR). It is probably running Linux like most currently popular multi-media appliances.

Which variety of Linux would you expect to have been installed on it?

- ○ A Desktop version with a friendly graphical interface
- ○ A Server version, such as Enterprise Linux
- ◉ A specialized version with only the minimal set of software needed to accomplish its task

What type of Linux configuration file(s) is/are used to automate the installation process?

Kickstart, AutoYaST, - correct

- ☑ Kickstart
- ☑ AutoYaST
- ☐ CD/DVD
- ☐ USB

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- A **partition** is a logical part of the disk.
- A **filesystem** is a method of storing/finding files on a hard disk.
- By dividing the hard disk into partitions, data can be grouped and separated as needed. When a failure or mistake occurs, only the data in the affected partition will be damaged, while the data on the other partitions will likely survive.
- The boot process has multiple steps, starting with **BIOS**, which triggers the **boot loader** to start up the Linux kernel. From there the **initramfs** filesystem is invoked, which triggers the **init** program to complete the startup process.
- Determining the appropriate distribution to deploy requires that you match your specific system needs to the capabilities of the different distributions.

Chapter 4: Graphical Interface

Learning Objectives

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

- Manage graphical interface sessions.
- Perform basic operations using the graphical interface.
- Change the graphical desktop to suit your needs.

Section 1: Session Management

You can use either a Command Line Interface (**CLI**) or a Graphical User Interface (**GUI**) when using Linux. To work at the CLI, you have to remember which programs and commands are used to perform tasks, and how to quickly and accurately obtain more information about their use and options. On the other hand, using the GUI is often quick and easy. It allows you to interact with your system through graphical icons and screens. For repetitive tasks the CLI is often more efficient, while the GUI is easier to navigate if you don't remember all the details or do something only rarely.

In this section you will learn how to manage sessions using the GUI for the three Linux distribution families that we explicitly cover in this course: **CentOS** (**Fedora** family), **openSUSE** (**SUSE** family) and **Ubuntu** (**Debian** family). As you'll see shortly, **openSUSE** uses **KDE** instead of **GNOME** as the default desktop manager. However, since in many cases we use just a single distro for illustration, we've used **GNOME** for the openSUSE visuals throughout this course. If you are using **KDE** your experience will vary somewhat from what is shown.

GNOME Desktop Environment

GNOME is a popular desktop environment with an easy to use graphical user interface. It is bundled as the default desktop environment for many distributions including **Red Hat Enterprise Linux**, **Fedora**, **CentOS**, **SUSE Linux Enterprise**, and **Debian**. **GNOME** has menu-based navigation and is sometimes an easy transition for at least some **Windows** users. However, as you'll see, the look and feel can be quite different across distributions, even if they are all using **GNOME**.

Another common desktop environment very important in the history of Linux and also widely used is **KDE**, which is used by default in **openSUSE**.

Other alternatives for a desktop environment include **Unity** (from **Ubuntu**, based on **GNOME**), **Xfce**, and **LXDE**. Most desktop environments follow a similar structure to **GNOME**.

GUI Startup

When you install a desktop environment, the **X** display manager starts at the end of the boot process. This **X** display manager is responsible for starting the graphics system, logging in the user, and starting the user's desktop environment. You can often select from a choice of desktop environments when logging in to the system.

The default display manager for **GNOME** is called **gdm**. Other popular display managers include **lightdm** (used on **Ubuntu**) and **kdm** (associated with **KDE**).

Shutting Down and Restarting

Besides normal daily starting and stopping of the computer, a system restart may be required as part of certain major system updates, generally only those involving installing a new Linux **kernel**.

The **init** process is responsible for implementing both restarts and shut downs. On systems using **System V init**, run level 0 is usually used for shutting down, and run level 6 is used to reboot the system. (We will discuss system run levels later.)

Next we will discuss how to shut down and restart the system in the three different reference distributions.

Section 2: Basic Operations

Default Directories

Every user with an account on the system will have a **home** directory, usually created under /home and named the same as the username (such as /home/student). By default, files the user saves will be placed in a directory tree starting there. Account creation, whether during system installation or at a later time when a new user is added, also induces default directories to be created under the user's home directory, such as **Documents**, **Desktop**, and **Downloads**.

Default Directories in Ubuntu

To access your **home** directory from the graphical user interface in **GNOME** on **Ubuntu**, click the **File Cabinet** icon on the left of the screen.

The **File Manager** (**Files** in the case of **Ubuntu**) will open a window with your **Home** directory displayed. The left panel of the **File Manager** window holds a list of commonly used directories, such as **Computer**, **Home**, **Desktop**, **Documents**, **Downloads,** and **Trash**.

You can also click the magnifying glass icon on the top-right of the **File Manager** window to search for files or directories that exist inside your **home** directory.

Viewing Files

Nautilus (the name of the File Manager or file browser) allows you to view files and directories in several different formats.

To view files in the **Icons**, **List**, or **Compact** formats, click the **View** drop-down and select your view, or press **CTRL-1** , **CTRL-2** and **CTRL-3** respectively.

In addition you can also arrange the files and directories by **Name**, **Size**, **Type**, or **Modification Date** for further sorting. To do so, click **View** and select **Arrange Items**.

Searching for Files

Nautilus allows you to refine your search beyond the initial keyword by providing drop-down menus to further filter the search.

1. Based on **Location** or **File Type**, select additional criteria from the drop-down.
2. To regenerate the search, click the **Reload** button.
3. To add multiple search criteria, click the **+** button and select **additional search criteria**.

For example, if you want to find a PDF file containing the word Linux in your **home** directory, navigate to your **home** directory and search for the word “Linux” . You should see that the default search criterion limits the search to your **home** directory already. To finish the job, click the + button to add another search criterion, select **File Type** for the type of criterion, and select **PDF** under the **File Type** drop-down.

Editing a File

Editing any text file through the graphical interface is easy in the **GNOME** desktop environment. Simply double-click the file on the desktop or in the **Nautilus** file browser window to open the file with the default text editor.

The default text editor in **GNOME** is **gedit**. It is simple yet powerful, ideal for editing documents, making quick notes, and programming. Although **gedit** is designed as a general purpose text editor, it offers additional features for spell checking, highlighting, file listings, and statistics.

You'll learn much more about using text editors in a later chapter.

Removing a File

Deleting a file in **Nautilus** will automatically move the deleted files to the .local/share/Trash/files/ directory (a trash can of sorts) under the user's **HOME** directory. There are several ways to delete files and directories using Nautilus.

1. Select all the files and directories that you want to delete.
2. Press **Delete** (in Unity/KDE) or **CTRL-Delete** (in GNOME) on your keyboard. Or, Right-click the file.
3. Select **Move to Trash**. Or, Highlight the file.
4. Click **Edit** and **Move to Trash** through the graphical interface.

Which of these are default directories available under the **home** directory?

- [x] Desktop
- [x] Documents
- [ ] Root
- [x] Downloads

What is the default Text Editor in GNOME?

- (•) gedit gedit
- ( ) Notepad
- ( ) Nautilus
- ( ) redit

Section 3: Graphical Desktop

Graphical Desktop

Each Linux distribution comes with its own set of desktop backgrounds. You can change the default by choosing a new wallpaper or selecting a custom picture to be set as the desktop background. If you do not want to use an image as the background, you can select a color to be displayed on the desktop instead.

In addition, you can also change the desktop theme, which changes the look and feel of the Linux system. The theme also defines the appearance of application windows.

Desktop Background

If you do not like any of the installed wallpapers, you can use different shades of color as the background using the **Colors and Gradients** drop-down in the **Appearance** window.

There are three types of color: solid, horizontal gradient, and vertical gradient. Click the box at the bottom and pick the effect between solid and the two gradients. In addition, you can also install packages that contain wallpapers by searching for packages using “wallpaper” as a keyword.

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- **GNOME** is a popular desktop environment and graphical user interface that runs on top of the Linux operating system.
- The default display manager for **GNOME** is called **gdm**.
- The **gdm** display manager presents the user with the login screen which prompts for the login username and password.
- Logging out through the desktop environment kills all processes in your current **X** session and returns to the display manager login screen.

- Linux enables users to switch between logged in sessions.
- Suspending puts the computer into sleep mode.
- For each key task, there is generally a default application installed.
- Every user created in the system will have a **home** directory.
- The **Places** menu contains entries that allow you to access different parts of the computer and the network.

- **Nautilus** gives three formats to view files.
- Most text editors are located in the **Accessories** submenu.
- Each Linux distribution comes with its own set of desktop backgrounds.
- **GNOME** comes with a set of different themes which can change the way your applications look.

Chapter 5: System Configuration from the Graphical Interface

Learning Objectives

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

- 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.
- Install and update software in Linux from a graphical interface.

Note that we will revisit all these tasks later when we discuss how to accomplish them from the command line interface.

Section 1: System, Display, Date and Time Settings.

System Settings

The **System Settings** panel allows you to control most of the basic configuration options and desktop settings such as specifying the screen resolution, managing network connections, or changing the date and time of the system.

As we mentioned in Chapter 4, we use the **GNOME Desktop Manager** for the visuals in this course as it is the default for **CentOS** and **Ubuntu** and readily available on **openSUSE** (for which the default is the **KDE Desktop Manager**).

The procedure to access **System Settings** varies according to distribution:

- **CentOS:** click System ⟶ Preferences.

- **openSUSE:** click **Activities**, type **Settings** in the **Search** box.

- **Ubuntu**: they are located in the panel on the left of the screen.

Display Settings

The **Displays** panel under **System Settings** (or **Display and Monitor** panel under **Configure Desktop**) contains the most common settings for changing the desktop appearance. These settings function independently of the specific display drivers you are running.

If your system uses a proprietary driver such as those from **nVidia** or **AMD**, you will probably have a configuration program for that driver that is not included in **System Settings**. This program may give more configuration options, but may also be more complicated, and might require sysadmin (root) access. If possible, you should configure the settings in the **Displays** panel rather than the proprietary configuration program.

The **X** server, which actually provides the GUI, uses the /etc/X11/xorg.conf file as its configuration file *if it exists*. In modern Linux distributions, this file is usually present only in unusual circumstances, such as when certain less common graphic drivers are in use. Changing this configuration file directly is usually for more advanced users.

Setting Resolution and Configuring Multiple Screens

While your system will usually figure out the best resolution for your screen automatically, it may get this wrong in some cases, or you might want to change the resolution to meet your needs.

You can accomplish this using the **Displays** panel. The switch to the new resolution will be effective when you click **Apply**, and then confirm that the resolution is working. In case the selected resolution fails to work or you are just not happy with the appearance, the system will switch back to the original resolution after a short timeout.

In most cases the configuration for multiple displays is set up automatically as one big screen spanning all monitors, using a reasonable guess for screen layout. If the screen layout is not as desired, a check box can turn on mirrored mode, where the same display is seen on all monitors.

Date and Time Settings

Linux always uses **Coordinated 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. UTC is similar to, but more accurate than, Greenwich Mean Time (GMT).

The **Date and Time Settings** window can be accessed from the **System Settings** window. Alternatively, you can right-click **Date and Time** on the top panel to access the **Date and Time Settings** window

Network Time Protocol

The **Network Time Protocol (NTP)** is the most popular and reliable protocol for setting the local time via Internet servers. Most Linux distributions include a working NTP setup which refers to specific time servers run by the distribution. This means that no setup, beyond "on or off", is required for network time synchronization. If desired, more detailed configuration is possible by editing the standard NTP configuration file (/etc/ntp.conf) for Linux NTP utilities.

Which protocol is used to set the local time via Internet servers?

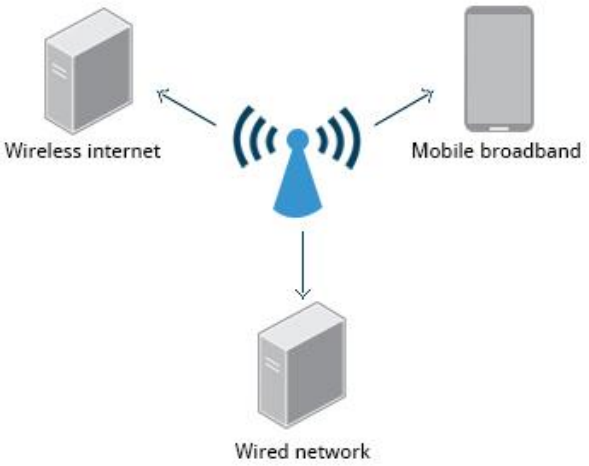
- ○ Network Date Time Protocol
- ○ Network Time Date Protocol
- ◉ Network Time Protocol
- ○ Network Date Protocol

Which of the following settings can you configure using the **Display** panel or the **Display and Monitor** panel?

- ☐ Preferred Applications
- ☑ Screen Resolution
- ☐ Suspend
- ☐ User Name

Section 2: Network Configuration

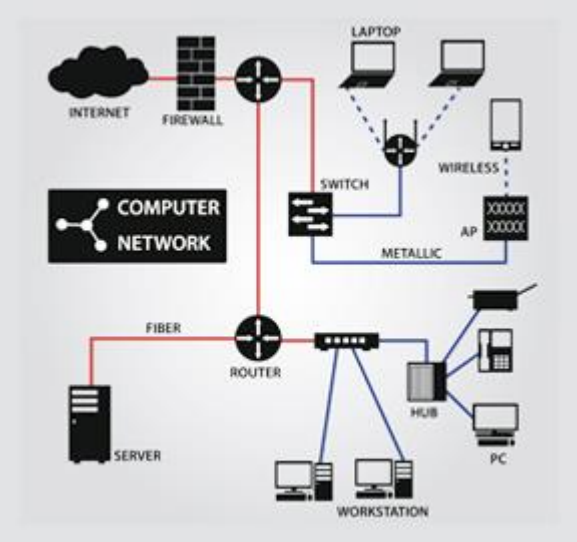
Network Configuration



All Linux distributions have network configuration files, but file formats and locations can differ from one distribution to another. Hand editing of these files can handle quite complicated setups, but is not very dynamic or easy to learn and use. The **Network Manager** utility was developed to make things easier and more uniform across distributions. It can list all available networks (both wired and wireless), allow the choice of a wired, wireless or mobile broadband network, handle passwords, and set up **Virtual Private Networks (VPNs)**. Except for unusual situations, it's generally best to let the **Network Manager** establish your connections and keep track of your settings.

In this section, you will learn how to manage network connections, including wired and wireless connections, and mobile broadband and VPN connections.

Wired and Wireless Connections



Wired connections usually do not require complicated or manual configuration. The hardware interface and signal presence are automatically detected, and then **Network Manager** sets the actual network settings via **DHCP** (Dynamic Host Control Protocol).

For **static** configurations that don't use **DHCP**, manual setup can also be done easily through **Network Manager**. You can also change the **Ethernet Media Access Control (MAC) address** if your hardware supports it. (The MAC address is a unique hexadecimal number of your network card.)

Wireless networks are not connected to the machine by default. You can view the list of available wireless networks and see which one you are connected to by using **Network Manager.** You can then add, edit, or remove known wireless networks, and also specify which ones you want connected by default when present.

Configuring Wireless Connections in Ubuntu

To configure **Wireless Network** in **Ubuntu**:

1. In top panel, click **Network Manager**.
2. Click **Enable Wi-Fi** - to display a list available **Wireless Networks**.
3. Click the desired **Wireless Network**.
4. For a secured network, enter the password.
5. To modify saved wireless network settings, click **Edit Connections**.

Configuring Wireless Connections in openSUSE

openSUSE looks different from **CentOS** or **Ubuntu**, but Wired, Wireless, Mobile Broadband, VPN, and DSL are all available from the **Network Connections** dialog box as you will see in the upcoming demonstration.

Configuring Wireless Connections in Ubuntu

Mobile Broadband and VPN Connections

You can set up a mobile broadband connection with **Network Manager**, which will launch a wizard to set up the connection details for each connection.

Once the configuration is done, the network is configured automatically each time the broadband network is attached.

Network Manager can also manage your VPN connections.

It supports many VPN technologies, such as native **IPSec**, **Cisco OpenConnect** (via either the Cisco client or a native open-source client), **Microsoft PPTP**, and **OpenVPN**.

You might get support for VPN as a separate package from your distributor. You need to install this package if your preferred VPN is not supported.

Which VPN technologies does the Network Manager support?

- ○ OpenVPN
- ○ Cisco OpenConnect
- ○ IPSec
- ◉ All of the above

Which of the following network connections are usually automatically configured?

- ○ Wireless
- ◉ Wired Wired
- ○ Broadband Connection
- ○ VPN

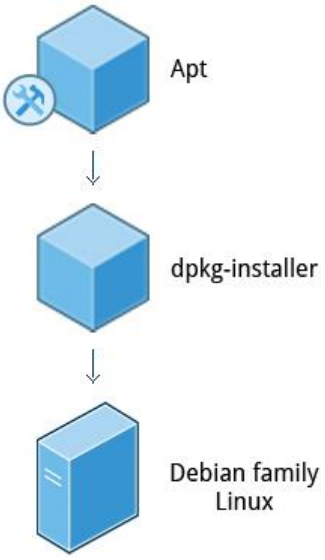
Section 3: Installing and Updating sw

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

Packages often depend on each other; for example, because **Firefox** can communicate using SSL/TLS, it will depend on a package which provides the ability to encrypt and decrypt SSL and TLS communication, and will not install unless that package is also installed at the same time.

One utility handles the low-level details of unpacking a package and putting the pieces in the right places. Most of the time, you will be working with a higher-level utility which knows how to download packages from the Internet and can manage dependencies and groups for you.

Debian Family System



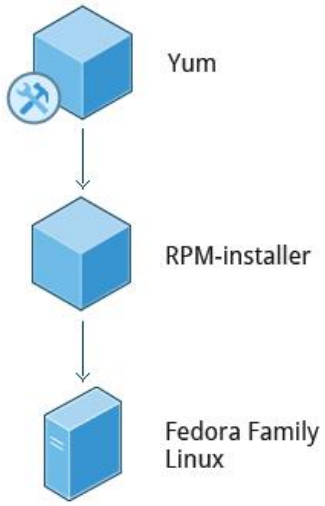
Let's look at **Package Management** in the **Debian** Family System.

dpkg is the underlying package manager for these systems; it can install, remove, and build packages. Unlike higher-level package management systems, it does not automatically download and install packages and satisfy their dependencies.

For Debian-based systems, the higher-level package management system is the **apt** (**A**dvanced **P**ackage **T**ool) system of utilities. Generally, while each distribution within the **Debian** family uses **apt**, it creates its own user interface on top of it (for example, **apt-get**, **aptitude**, **synaptic**, **Ubuntu Software Center**, **Update Manager**, etc). Although **apt** repositories are generally compatible with each other, the software they contain generally isn't. Therefore, most **apt** repositories target a particular distribution (like **Ubuntu**), and often software distributors ship with multiple repositories to support multiple distributions. The demonstration using the **Ubuntu Software Center** is shown later in this section.

https://www.youtube.com/watch?v=iOJWFEdXcao#t=14

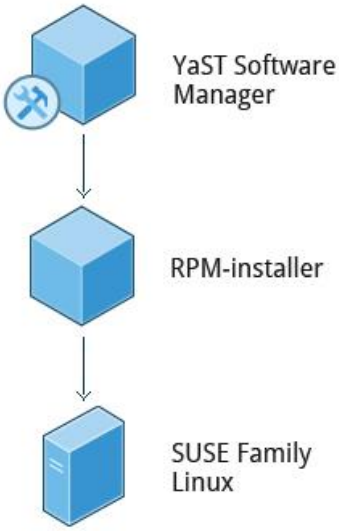
Red Hat Packet Manager (RPM)



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

The high-level package manager differs between distributions; most use the basic repository format used in **yum** (Yellowdog Updater, Modified - the package manager used by **Fedora** and **Red Hat Enterprise Linux**), but with enhancements and changes to fit the features they support. Recently, the **GNOME** project has been developing **PackageKit** as a unified interface; this is now the default interface for **Fedora**.

openSUSE's YaST Software Management



Before **openSUSE 13.1**, **Apper** was used for Software Management. Now it has been replaced by the **YaST** (**Y**et **a**nother **S**ystem **T**ool) **Software Manager**.

The **YaST Software Manager** is similar to other graphical package managers. It is an **RPM-**based application. You can add, remove, or update packages using this application very easily. To access the **YaST Software Manager**:

1. Click **Activities**
2. In the **Search** box type **YaST**
3. Click the **YaST** icon
4. Click **Software Management**

openSUSE's YaST Software Management application is similar to the graphical package managers in other distributions. The demonstration of the **YaST Software Manager** is shown later in this section.

Ubuntu uses which two of the following package management utilities?

- [ ] up2date
- [x] dpkg
- [ ] yum
- [x] Ubuntu Software Center

OpenSUSE uses _____ to add and remove software packages.

- ( ) YaST Network Settings
- (•) YaST Software Management
- ( ) YaST Configuration Management
- ( ) YaST Printer

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- You can control basic configuration options and desktop settings through the **System Settings** panel
- Linux always uses **Coordinated Universal Time (UTC)** for its own internal time-keeping . You can set **Date and Time Settings** from the **System Settings** window.
- The **Network Time Protocol** is the most popular and reliable protocol for setting the local time via Internet servers.
- The **Displays** panel allows you to change the resolution of your display and configure multiple screens.
- **Network Manager** can present available wireless networks, allow the choice of a wireless or mobile broadband network, handle passwords, and set up VPNs.
- **dpkg** and **RPM** are the most popular package management systems used on Linux distributions.
- **Debian** distributions use **dpkg** and **apt**-based utilities for package management.
- **RPM** was developed by **Red Hat**, and adopted by a number of other distributions, including the **openSUSE**, **Mandriva**, **CentOS**, **Oracle Linux**, and others.

Chapter 6: Command Line Operations

Learning Objectives

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

- Use the command line to perform operations in Linux.
- Search for files.
- Create and manage files.
- Install and update software.

Section 1: Command Line Mode Options

Introduction to the Command Line

Linux system administrators spend a significant amount of their time at a **command line** prompt. They often automate and troubleshoot tasks in this text environment. There is a saying, "*graphical user interfaces make easy tasks easier, while command line interfaces make difficult tasks possible*." Linux relies heavily on the abundance of command line tools. The command line interface provides the following advantages:

- No GUI overhead.
- Virtually every task can be accomplished using the command line.
- You can script tasks and series of procedures.
- You can log on remotely to networked machines anywhere on the Internet.
- You can initiate graphical apps directly from the command line.

Using a Text Terminal on the Graphical Desktop

A **terminal emulator** program emulates (simulates) a stand alone terminal within a window on the desktop. By this we mean it behaves essentially as if you were logging into the machine at a pure text terminal with no running graphical interface. Most terminal emulator programs support multiple terminal sessions by opening additional tabs or windows.

By default, on **GNOME** desktop environments, the **gnome-terminal** application is used to emulate a text-mode terminal in a window. Other available terminal programs include:

- **xterm**
- **rxvt**
- **konsole**
- **terminator**

Launching Terminal Windows

To open a terminal in **CentOS**:

1. On the **CentOS** desktop, in the upper-left corner, click **Applications**.
2. From the **System Tools** menu, select **Terminal**.

To open a terminal in **openSUSE**:

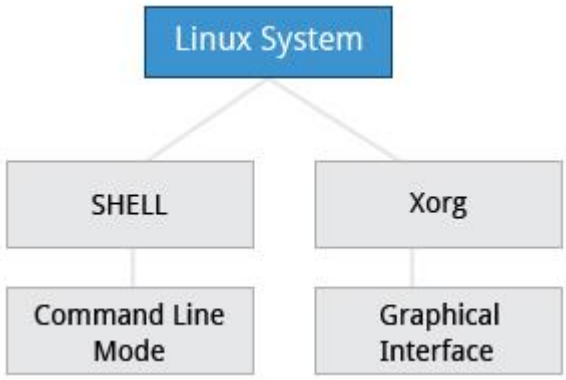
1. On the **openSUSE** desktop, in the upper-left corner of the screen, click **Activities**.
2. From the left pane, click **Show Applications**.
3. Scroll-down and select the required terminal.

To open a terminal in **Ubuntu**:

1. In the left panel, click the **Ubuntu** icon.
2. Type **terminal** in the **Search** box.

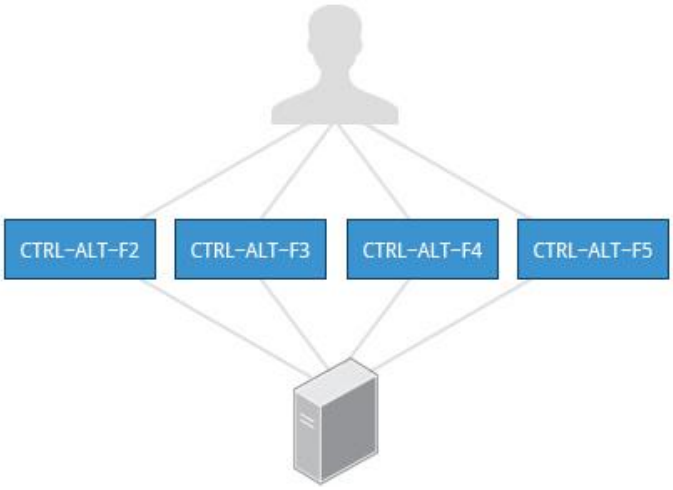
If the **nautilus-open-terminal** package is installed on any of these distributions, you can always open a terminal by right clicking anywhere on the desktop background and selecting **Open in Terminal.**

The X Window System



The customizable nature of Linux allows you to drop (temporarily or permanently) the **X Window** graphical interface, or to start it up after the system has been running. Certain Linux distributions distinguish versions of the install media between desktop (with **X**) and server (usually without **X**); Linux production servers are usually installed without **X** and even if it is installed, usually do not launch it during system start up. Removing **X** from a production server can be very helpful in maintaining a lean system which can be easier to support and keep secure.

Virtual Terminals



Virtual Terminals (VT) are **console** sessions that use the entire display and keyboard outside of a graphical environment. Such terminals are considered "virtual" because although there can be multiple active terminals, only one terminal remains visible at a time. A VT is not quite the same as a command line terminal window; you can have many of those visible at once on a graphical desktop.

One virtual terminal (usually number one or seven) is reserved for the graphical environment, and text logins are enabled on the unused VTs. **Ubuntu** uses VT 7, but **CentOS/RHEL** and **openSUSE** use VT 1 for the graphical display.

An example of a situation where using the VTs is helpful when you run into problems with the graphical desktop. In this situation, you can switch to one of the text VTs and troubleshoot.

To switch between the VTs, press **CTRL-ALT-corresponding function key** for the VT. For example, press **CTRL-ALT-F6** for VT 6. (Actually you only have to press **ALT-F6** key combination if you are in a VT not running **X** and want to switch to another VT.)

The Command Line

Most input lines entered at the shell prompt have three basic elements:

- Command
- Options
- Arguments

The **command** is the name of the program you are executing. It may be followed by one or more **options** (or switches) that modify what the command may do. Options usually start with one or two dashes, for example,-p or--print, in order to differentiate them from **arguments**, which represent what the command operates on.

However, plenty of commands have no options, no arguments, or neither. You can also type other things at the command line besides issuing commands, such as setting environment variables.

Turning off the Graphical Desktop

Linux distributions can start and stop the graphical desktop in various ways. For **Debian**-based systems, the **Desktop Manager** runs as a service which can be simply stopped. For RPM-based systems, the **Desktop Manager** is run directly by **init** when set to run level 5; switching to a different runlevel stops the desktop.

Use the sudo service gdm stop or sudo service lightdm stop commands, to stop the graphical user interface in **Debian**-based systems.

On **RPM**-based systems typing sudo telinit 3 may have the same effect of killing the GUI.

sudo

All the demonstrations created have a user configured with **sudo** capabilities to provide the user with administrative (admin) privileges when required. **sudo** allows users to run programs using the security privileges of another user, generally root (superuser). The functionality of **sudo** is similar to that of **run as** in **Windows**.

On your own systems, you may need to set up and enable **sudo** to work correctly. To do this, you need to follow some steps that we won't explain in much detail now, but you will learn about later in this course. When running on **Ubuntu**, **sudo** is already always set up for you during installation. If you are running something in the **Fedora** or **openSUSE** families of distributions, you will likely need to set up **sudo** to work properly for you after initial installation.

Steps for Setting up and Running sudo

If your system does not already have **sudo** set up and enabled, you need to do the following steps:

1. You will need to make modifications as the administrative or super user, root. While **sudo** will become the preferred method of doing this, we don't have it set up yet, so we will use **su** (which we will discuss later in detail) instead. At the command line prompt, type **su** and press **Enter.** You will then be prompted for the root password, so enter it and press **Enter**. You will notice that nothing is printed; this is so others cannot see the password on the screen. You should end up with a different looking prompt, often ending with '**#**'. For example: $ su Password: #
2. Now you need to create a configuration file to enable your user account to use **sudo**. Typically, this file is created in the /etc/sudoers.d/ directory with the name of the file the same as your username. For example, for this demo, let's say your username is "student". After doing step 1, you would then create the configuration file for "student" by doing this: # echo "student ALL=(ALL) ALL" > /etc/sudoers.d/student
3. Finally, some Linux distributions will complain if you don't also change permissions on the file by doing: # chmod 440 /etc/sudoers.d/student

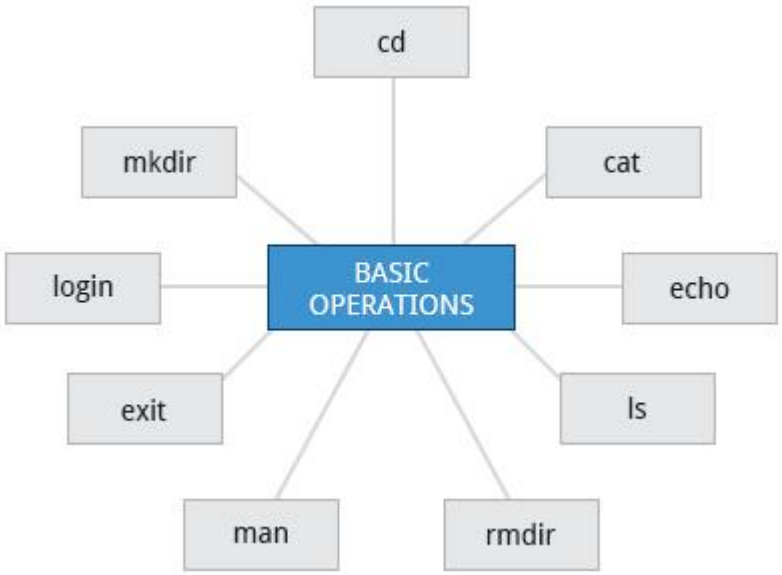
That should be it. For the rest of this course, if you use **sudo** you should be properly set up. When using **sudo,** by default you will be prompted to give a password (your own user password) at least the first time you do it within a specified time interval. It is possible (though very insecure) to configure **sudo** to not require a password or change the time window in which the password does not have to be repeated with every **sudo** command.

Assuming you are not the root user, choose the correct command that turns off the graphical desktop on an RPM-based system.

- (•) sudo telinit 3
- ( ) sudo teleinit 3
- ( ) init 3
- ( ) initel 3

Section 2: Basic Operations

Basic Operations



In this section we will discuss how to accomplish basic operations from the command line. These include how to log in and log out from the system, restart or shutdown the system, locate applications, access directories, identify the absolute and relative paths, and explore the filesystem.

Logging In and Out

An available **text terminal** will prompt for a username (with the string login:) and password. When typing your password, nothing is displayed on the terminal (not even a * to indicate that you typed in something) to prevent others from seeing your password. After you have logged in to the system, you can perform basic operations.

Once your session is started (either by logging in to a text terminal or via a graphical terminal program) you can also connect and log in to remote systems via the **Secure Shell (SSH)** utility. For example, by typing

ssh username@remote-server.com

SSH would connect securely to the remote machine and give you a command line terminal window, using passwords (as with regular logins) or cryptographic keys (a topic we won't discuss) to prove your identity.

Rebooting and Shutting Down

The preferred method to shut down or reboot the system is to use the **shutdown** command. This sends a warning message and then prevents further users from logging in. The **init** process will then control 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 and/or loss of data.

The **halt** and **poweroff** commands issue shutdown -h to halt the system; **reboot** issues shutdown -r and causes the machine to reboot instead of just shutting down. Both rebooting and shutting down from the command line requires superuser (root) access.

When administering a multiuser system, you have the option of notifying all users prior to shutdown as in:

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

Locating Applications

Depending on the specifics of your particular distribution's policy, programs and software packages can be installed in various directories. In general, executable programs should live in the /bin, /usr/bin,/sbin,/usr/sbin directories or under /opt.

One way to locate programs is to employ the **which** utility. For example, to find out exactly where the **diff** program resides on the filesystem:

```
$ which diff
```

If **which** does not find the program, **whereis** is a good alternative because it looks for packages in a broader range of system directories:

```
$ whereis diff
```

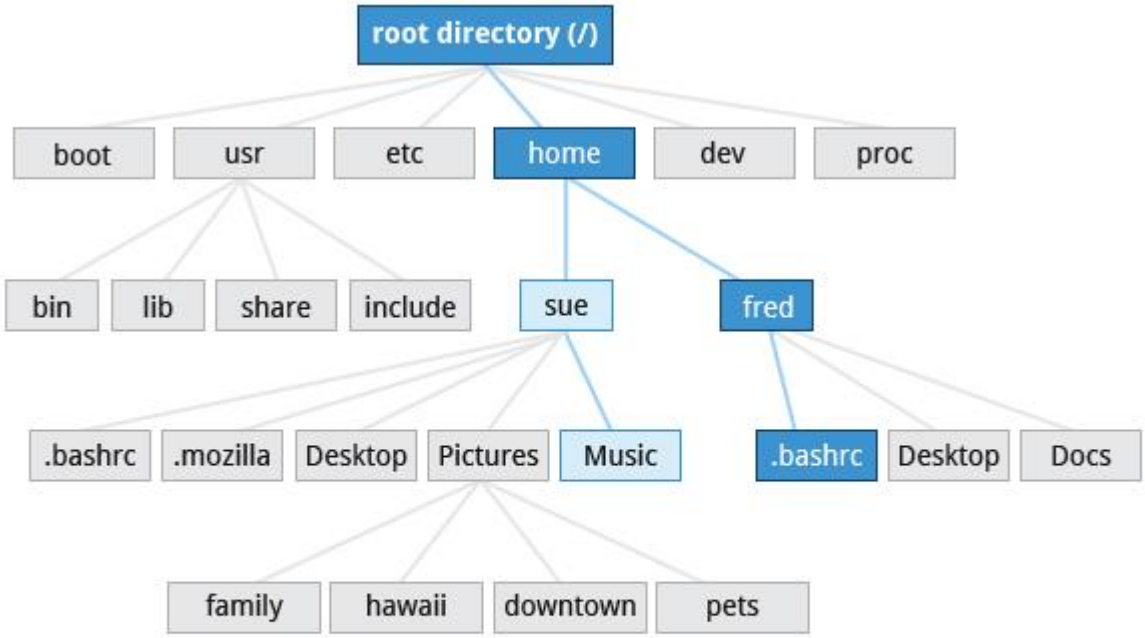
Accessing Directories

When you first log into a system or open a terminal, the default directory should be your **home directory**; you can print the exact path of this by typing echo $HOME. (Note that some Linux distributions actually open new **graphical** terminals in $HOME/Desktop.) The following commands are useful for directory navigation:

Command	Result
pwd	Displays the present working directory
cd ~ or cd	Change to your home directory (short-cut name is ~ (tilde))
cd ..	Change to parent directory (..)
cd -	Change to previous directory (- (minus))

Accessing Directories

Understanding Absolute and Relative Paths



1.In the above example, we use the relative path method to list the files under Music from your current working directory (sue)
$ ls ../sue/Music

2.In the above example, we use the Absolute pathname method to edit the .bashrc file:
$ gedit /home/fred/.bashrc

There are two ways to identify paths:

1. **Absolute pathname**: An absolute pathname begins with the root directory and follows the tree, branch by branch, until it reaches the desired directory or file. Absolute paths always start with /.
2. **Relative pathname**: A relative pathname starts from the present working directory. Relative paths never start with /.

Multiple slashes (/) between directories and files are allowed, but all but one slash between elements in the pathname is ignored by the system. ////usr//bin is valid, but seen as /usr/bin by the system.

Most of the time it is most convenient to use relative paths, which require less typing. Usually you take advantage of the shortcuts provided by: . (present directory), .. (parent directory) and ~ (your home directory).

For example, suppose you are currently working in your home directory and wish to move to the /usr/bin directory. The following two ways will bring you to the same directory from your home directory:

1. Absolute pathname method: $ cd /usr/bin
2. Relative pathname method: $ cd ../../usr/bin

In this case, the absolute pathname method is less typing.

Exploring the Filesystem

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

The following commands can help in exploring the filesystem:

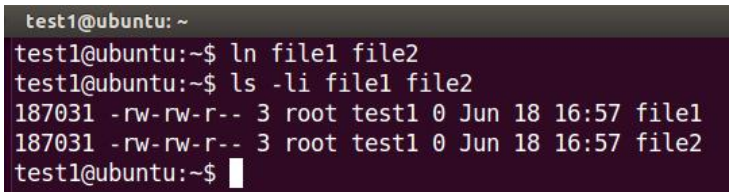
Command	Usage
cd /	**C**hanges your current **d**irectory to the root (/) directory (or path you supply)
ls	**L**i**s**t the contents of the present working directory
ls –a	**L**i**s**t **all** files including **hidden** files and directories (those whose name start with .)
tree	Displays a **tree** view of the filesystem

Hard and Soft (Symbolic) Links

ln can be used to create **hard links** and (with the -s option) **soft links**, also known as **symbolic links** or **symlinks**. These two kinds of links are very useful in UNIX-based operating systems. The advantages of symbolic links are discusssed on the following screen.

Suppose that file1 already exists. A **hard** link, called file2, is created with the command:

$ ln file1 file2



Note that two files now appear to exist. However, a closer inspection of the file listing shows that this is not quite true.

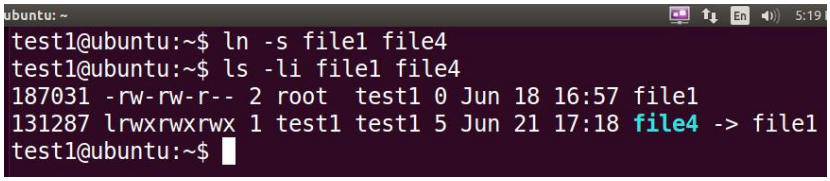
$ ls -li file1 file2

The -i option to **ls** prints out in the first column the **inode** number, which is a unique quantity for each file object. This field is the same for both of these files; what is really going on here is that it is only **one** file but it has more than one name associated with it, as is indicated by the **3** that appears in the **ls** output. Thus, there already was another object linked to file1 before the command was executed.

Symbolic Links

Symbolic (or **Soft**) links are created with the -s option as in:

$ ln -s file1 file4
$ ls -li file1 file4



Notice file4 no longer appears to be a regular file, and it clearly points to file1 and has a different inode number.

Symbolic links take no extra space on the filesystem (unless their names are very long). They are extremely convenient as they can easily be modified to point to different places. An easy way to create a shortcut from your **home** directory to long pathnames is to create a symbolic link.

Unlike hard links, soft links can point to objects even on different filesystems (or partitions) which may or may not be currently available or even exist. In the case where the link does not point to a currently available or existing object, you obtain a **dangling** link.

Hard links are very useful and they save space, but you have to be careful with their use, sometimes in subtle ways. For one thing if you remove either file1 or file2 in the example on the previous screen, the **inode object** (and the remaining file name) will remain, which might be undesirable as it may lead to subtle errors later if you recreate a file of that name.

If you edit one of the files, exactly what happens depends on your editor; most editors including **vi** and **gedit** will retain the link by default but it is possible that modifying one of the names may break the link and result in the creation of two objects.

Exploring the Filesystem

Navigating the Directory History

The **cd** command remembers where you were last, and lets you get back there with cd -.

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

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

```
[test3@CentOS ~]$ mkdir /tmp/dir1
[test3@CentOS ~]$ mkdir /tmp/dir2
[test3@CentOS ~]$ pushd
~ ~
[test3@CentOS ~]$ pushd /tmp/dir1
/tmp/dir1 ~ ~
[test3@CentOS dir1]$ pushd /tmp/dir2
/tmp/dir2 /tmp/dir1 ~ ~
[test3@CentOS dir2]$ popd
/tmp/dir1 ~ ~
[test3@CentOS dir1]$ pwd
/tmp/dir1
[test3@CentOS dir1]$ popd
~ ~
[test3@CentOS ~]$ pwd
/home/test3
[test3@CentOS ~]$ popd
~
[test3@CentOS ~]$
```

Navigating Directory History

Assuming you are the root user, which commands allow you to shut down your system without rebooting?

- [x] halt
- [ ] shutdown -r now
- [x] poweroff
- [x] shutdown -h now

Which commands allow you to locate programs?

- [ ] whichis
- [x] which
- [ ] whichever
- [x] whereis

Which of the following are examples of an absolute path?

- [x] /etc/passwd
- [ ] ../passwd
- [x] //etc/passwd
- [ ] \\passwd

Section 3: Searching for Files

Standard File Streams

When commands are executed, by default there are three standard **file streams** (or **descriptors**) always open for use: **standard input** (standard in or **stdin**), **standard output** (standard out or **stdout**) and **standard error** (or **stderr**). Usually, **stdin** is your keyboard, **stdout** and **stderr** are printed on your terminal; often **stderr** is redirected to an error logging file. **stdin** is often supplied by directing input to come from a file or from the output of a previous command through a **pipe**. **stdout** is also often redirected into a file. Since **stderr** is where error messages are written, often nothing will go there.

In Linux, all open files are represented internally by what are called **file descriptors**. Simply put, these are represented by numbers starting at zero. **stdin** is file descriptor 0, **stdout** is file descriptor 1, and **stderr** is file descriptor 2. Typically, if other files are opened in addition to these three, which are opened by default, they will start at file descriptor 3 and increase from there.

On the next screen and in chapters ahead, you will see examples which alter where a running command gets its input, where it writes its output, or where it prints diagnostic (error) messages.

I/O Redirection

Through the command **shell** we can **redirect** the three standard filestreams so that we can get input from either a file or another command instead of from our keyboard, and we can write output and errors to files or send them as input for subsequent commands.

For example, if we have a program called **do_something** that reads from **stdin** and writes to **stdout** and **stderr**, we can change its input source by using the less-than sign (<) followed by the name of the file to be consumed for input data:

```
$ do_something < input-file
```

If you want to send the output to a file, use the greater-than sign (>) as in:

```
$ do_something > output-file
```

Because **stderr** is **not** the same as **stdout**, error messages will still be seen on the terminal windows in the above example.

If you want to redirect **stderr** to a separate file, you use **stderr's** file descriptor number (2), the greater-than sign (>), followed by the name of the file you want to hold everything the running command writes to **stderr**:

```
$ do_something 2> error-file
```

A special shorthand notation can be used to put anything written to file descriptor 2 (**stderr**) in the same place as file descriptor 1 (**stdout**): 2>&1

```
$ do_something > all-output-file 2>&1
```

bash permits an easier syntax for the above:

```
$ do_something >& all-output-file
```

Pipes

The UNIX/Linux philosophy is to have many simple and short programs (or commands) cooperate together to produce quite complex results, rather than have one complex program with many possible options and modes of operation. In order to accomplish this, extensive use of **pipes** is made; you can pipe the output of one command or program into another as its input.

In order to do this we use the vertical-bar, **| (pipe symbol)** between commands as in:
$ command1 | command2 | command3

The above represents what we often call a **pipeline** and allows Linux to combine the actions of several commands into one. This is extraordinarily efficient because **command2** and **command3** do not have to wait for the previous pipeline commands to complete before they can begin hacking at the data in their input streams; on multiple CPU or core systems the available computing power is much better utilized and things get done quicker. In addition there is no need to save output in (temporary) files between the stages in the pipeline, which saves disk space and reduces reading and writing from disk, which is often the slowest bottleneck in getting something done.

Searching for Files

Being able to quickly find the files you are looking for will make you a much happier Linux user! You can search for files in your parent directory or any other directory on the system as needed.

In this section, you will learn how to use the **locate** and **find** utilities, and how to use **wildcards** in **bash**.

locate

The **locate** utility program performs a search through a previously constructed database of files and directories on your system, matching all entries that contain a specified character string. This can sometimes result in a very long list.

To get a shorter more relevant list we can use the **grep** program as a filter; **grep** will print only the lines that contain one or more specified strings as in:

$ locate zip | grep bin

which will list all files and directories with both "zip" and "bin" in their name . (We will cover **grep** in much more detail later.) Notice the use of **|** to pipe the two commands together.

locate utilizes the database created by another program, **updatedb.** Most Linux systems run this automatically once a day. However, you can update it at any time by just running **updatedb** from the command line as the root user.

Locating Files

Wildcards and Matching File Names

You can search for a filename containing specific characters using **wildcards**.

Wildcard	Result
?	Matches any single character
*	Matches any string of characters
[set], [a-s]	Matches any character in the set of characters, for example [adf] will match any occurrence of "a", "d", or "f"
[!set],[!a-s]	Matches any character not in the set of characters

To search for files using the ? wildcard, replace each unknown **character** with ?, e.g. if you know only the first 2 letters are 'ba' of a 3-letter filename with an extension of .out, type ls ba?.out .

To search for files using the * wildcard, replace the unknown **string** with *, e.g. if you remember only that the extension was .out, type ls *.out

Using bash Wildcards to Search Files

Finding Files in a Directory

find is extremely useful and often-used utility program in the daily life of a Linux system administrator. It recurses down the file system tree from any particular directory (or set of directories) and locates files that match specified conditions. The default pathname is always the present working directory.

```
test1@ubuntu:~$ cd Documents
test1@ubuntu:~/Documents$ find . -name pro\*
./programs.txt~
./programs.txt
./programs_list.txt
./programs_list.txt~
test1@ubuntu:~/Documents$ which firefox
/usr/bin/firefox
test1@ubuntu:~/Documents$ whereis firefox
firefox: /usr/bin/firefox /etc/firefox /usr/lib/firefox /usr/bin/X11/firefox /usr/share/man/man
1/firefox.1.gz
test1@ubuntu:~/Documents$
```

For example, administrators sometimes scan for large **core files** (which contain diagnostic information after a program fails) that are more than several weeks old in order to remove them. It is also common to remove files in **/tmp** (and other temporary directories, such as those containing cached files) that have not been accessed recently. Many distros use automated scripts that run periodically to accomplish such house cleaning.

Using find

When no arguments are given, **find** lists all files in the current directory and all of its subdirectories.
Commonly used options to shorten the list include -name (only list files with a certain pattern in their name), -iname (also ignore the case of file names), and -type (which will restrict the results to files of a certain specified type, such as **d** for directory, **l** for symbolic link or **f** for a regular file, etc).

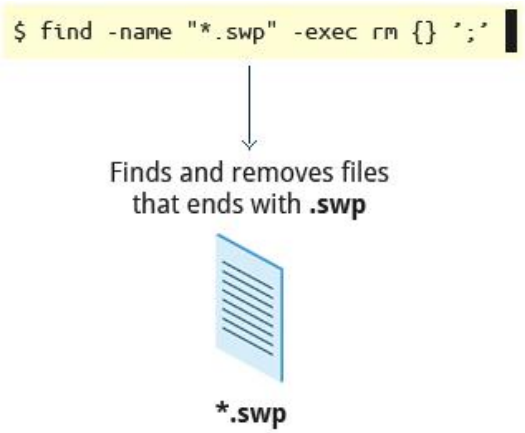
```
test1@ubuntu:~$ find /usr -name gcc
/usr/bin/gcc
/usr/lib/ccache/gcc
/usr/lib/gcc
/usr/share/doc/gcc
/usr/share/doc/gcc-4.8-base/gcc
/usr/share/bash-completion/completions/gcc
test1@ubuntu:~$ find /usr -type d -name gcc
/usr/lib/gcc
/usr/share/doc/gcc-4.8-base/gcc
test1@ubuntu:~$
```

Searching for files and directories named "gcc":
$ find /usr -name gcc

Searching only for directories named "gcc":
$ find /usr -type d -name gcc

Searching only for regular files named "test1":
$ find /usr -type f -name test1

Using Advanced find Options



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 {} (squiggly brackets) is a place holder that will be filled with all the file names that result from the **find** expression, and the preceding command will be run on each one individually.

Note: you have to end the command with either ';' (including the single-quotes) or \; Both forms are fine.

One can also use the -ok option which 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 your results before blindly executing any potentially dangerous commands.

Finding Files Based on Time and Size

It is sometimes the case that you wish to find files according to attributes such as when they were created, last used, etc, or based on their size. Both are easy to accomplish.

```
File  Edit  View  Search  Terminal  Help
[root@CentOS ~]# find / -size +10M
/boot/initramfs-2.6.32-431.el6.x86_64.img
/boot/initramfs-2.6.32-431.20.3.el6.x86_64.img
/boot/initramfs-2.6.32-431.11.2.el6.x86_64.img
/boot/initramfs-2.6.32-431.20.5.el6.x86_64.img
/usr/lib/locale/locale-archive
/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.65.x86_64/jre/lib/amd64/server/libjvm.so
/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.65.x86_64/jre/lib/amd64/server/classes.jsa
/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.65.x86_64/jre/lib/rt.jar
/usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0.x86_64/jre/lib/amd64/server/libjvm.so
/usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0.x86_64/jre/lib/amd64/server/classes.jsa
/usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0.x86_64/jre/lib/rt.jar
/usr/share/anthy/anthy.dic
/usr/share/mythes/th_en_US_v2.dat
/usr/share/icons/hicolor/icon-theme.cache
/usr/share/fonts/cjkuni-uming/uming.ttc
/usr/share/fonts/wqy-zenhei/wqy-zenhei.ttc
                .
                .
                .
                .
                .
                .
                .
/usr/lib64/valgrind/exp-sgcheck-amd64-linux
/usr/lib64/valgrind/none-amd64-linux
/usr/lib64/libLLVM-3.3-mesa.so
/usr/lib64/libicudata.so.42.1
/usr/lib64/valgrind/exp-sgcheck-amd64-linux
/usr/lib64/valgrind/none-amd64-linux
```

Finding based on time:
$ find / -ctime 3

Here, -ctime is when the inode meta-data (i.e., file ownership, permissions, etc) last changed; it is often, but not necessarily when the file was first created. You can also search for accessed/last read (-atime) or modified/last written (-mtime) times. The number is the number of days and can be expressed as either a number (n) that means exactly that value, +n which means greater than that number, or -n which means less than that number. There are similar options for times in minutes (as in -cmin, -amin, and -mmin).

Finding based on sizes:

$ find / -size 0

Note the size here is in 512-byte blocks, by default; you can also specify bytes (**c**), kilobytes (**k**), megabytes (**M**), gigabytes (**G**), etc. As with the time numbers above, file sizes can also be exact numbers (n), +n or -n. For details consult the **man** page for **find**.

For example, to find files greater than 10 MB in size and running a command on those files:
$ find / -size +10M -exec command {} ';'

Finding Files In a Directory

Tasks to be performed:

1. Search for a file with name gcc in the /usr folder using find command.
2. Search for a directory with filename gcc, in the /usr folder using find command.
3. Search for files in the current directory which were modified today.
4. Search for files with size 0 bytes.

```
[test1@CentOS bin]$ find /usr -name gcc

/usr/bin/gcc
/usr/libexec/gcc
find: `/usr/lib64/audit': Permission denied
/usr/lib/gcc


[test1@CentOS bin]$ find /usr -type d -name gcc
/usr/libexec/gcc
find: `/usr/lib64/audit': Permission denied
/usr/lib/gcc

[test1@CentOS ~]$ find -type f -mtime 0
./.ICEauthority
./.esd_auth
./.gstreamer-0.10/registry.x86_64.bin
./.gconfd/saved_state
…

[test1@CentOS ~]$ find -type f -size 0
./.local/share/.converted-launchers
./.gnupg/pubring.gpg
…
[test1@CentOS ~]$
```

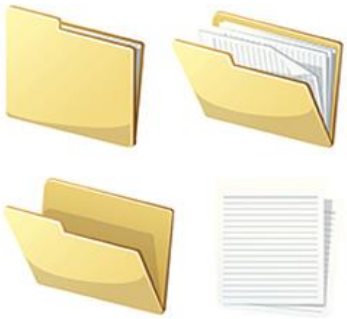
Which command is used to perform a database search of pathnames given the substring that is provided as a parameter?

Locate

Which **bash** wildcard is used to match any single character?

?

Section 4: Working with Files



Working with Files

Linux provides many commands that help you in viewing the contents of a file, creating a new file or an empty file, changing the **timestamp** of a file, and removing and renaming a file or directory. These commands help you in managing your data and files and in ensuring that the correct data is available at the correct location.

In this section, you will learn how to manage files.

Viewing Files

You can use the following utilities to view files:

Command	Usage
cat	Used for viewing files that are not very long; it does not provide any scroll-back.
tac	Used to look at a file backwards, one line at a time.
less	Used to view larger files because it is a paging program; it pauses at each screenful of text, provides scroll-back capabilities, and lets you search and navigate within the file. Note: Use / to search for a pattern in the forward direction and ? for a pattern in the backward direction.
tail	Used to print the last 10 lines of a file by default. You can change the number of lines by doing -n 15 or just -15 if you wanted to look at the last 15 lines instead of the default.
head	The opposite of **tail**; by default it prints the first 10 lines of a file.

More on Viewing Files

touch and mkdir

touch is often used to set or update the access, change, and modify times of files. By default it resets a file's time stamp to match the current time.

However, you can also create an **empty** file using touch:
$ touch <filename>

This is normally done to create an empty file as a placeholder for a later purpose.

touch provides several options, but here is one of interest:

- The -t option allows you to set the date and time stamp of the file.

To set the time stamp to a specific time:
$ touch -t 03201600 myfile

This sets the file, myfile's, time stamp to 4 p.m., March 20th (03 20 1600).

mkdir is used to create a directory.

- To create a sample directory named sampdir under the current directory, type mkdir sampdir.
- To create a sample directory called sampdir under /usr, type mkdir /usr/sampdir.

Removing a directory is simply done with **rmdir.** The directory must be empty or it will fail. To remove a directory and all of its contents you have to do rm -rf as we shall discuss.

Removing a File

Command	Usage
mv	Rename a file
rm	Remove a file
rm –f	Forcefully remove a file
rm –i	Interactively remove a file

If you are not certain about removing files that match a pattern you supply, it is always good to run **rm** interactively (rm –i) to prompt before every removal.

Modifying the Command Line Prompt

The **PS1** variable is the character string that is displayed as the prompt on the command line. Most distributions set **PS1** to a known default value, which is suitable in most cases. However, users may want custom information to show on the command line. For example, some system administrators require the user and the host system name to show up on the command line as in:

student@quad32 $

This could prove useful if you are working in multiple roles and want to be always reminded of who you are and what machine you are on. The prompt above could be implemented by setting the PS1 variable to: \u@\h \$

For example:

```
$ echo $PS1
\$
$ PS1="\u@\h \$ "
coop@quad64 $ echo $PS1
\u@\h \$
coop@quad64 $
```

Working With Files and Directories at the Command Prompt

Tasks to be performed:

1. Create two empty files test1 & test2 with timestamp: 14 March 2018 2:00 PM using touch command.
2. Check for the existence of test1 and test2 files using ls -l command.
3. Rename the test1 file to new_test1 using mv command.
4. Remove test2 using rm option without any options.
5. Remove the new_test1 file using rm command without any options.
6. Create a directory dir1, using mkdir command.
7. Remove the dir1 directory using rmdir command without any options.

```
[test1@CentOS ~]$ touch -t 1803141400 test1 test2
[test1@CentOS ~]$ ls -l test1 test2
-rw-r--r--.  1  test1  test1  0  Mar 14 2018  test1
-rw-r--r--.  1  test1  test1  0  Mar 14 2018  test2
[test1@CentOS ~]$ mv test1 new_test1
[test1@CentOS ~]$ rm test2
[test1@CentOS ~]$ rm new_test1
[test1@CentOS ~]$ mkdir dir1
```

```
[test1@CentOS ~]$ rmdir dir1
[test1@CentOS ~]$
```

Using which command can you view an entire file with scroll-back?

- tail
- cat
- (•) less
- head

Using which command you forcefully remove a directory recursively?

- rmdir
- (•) rm –rf
- rmdir –f
- mv –rf

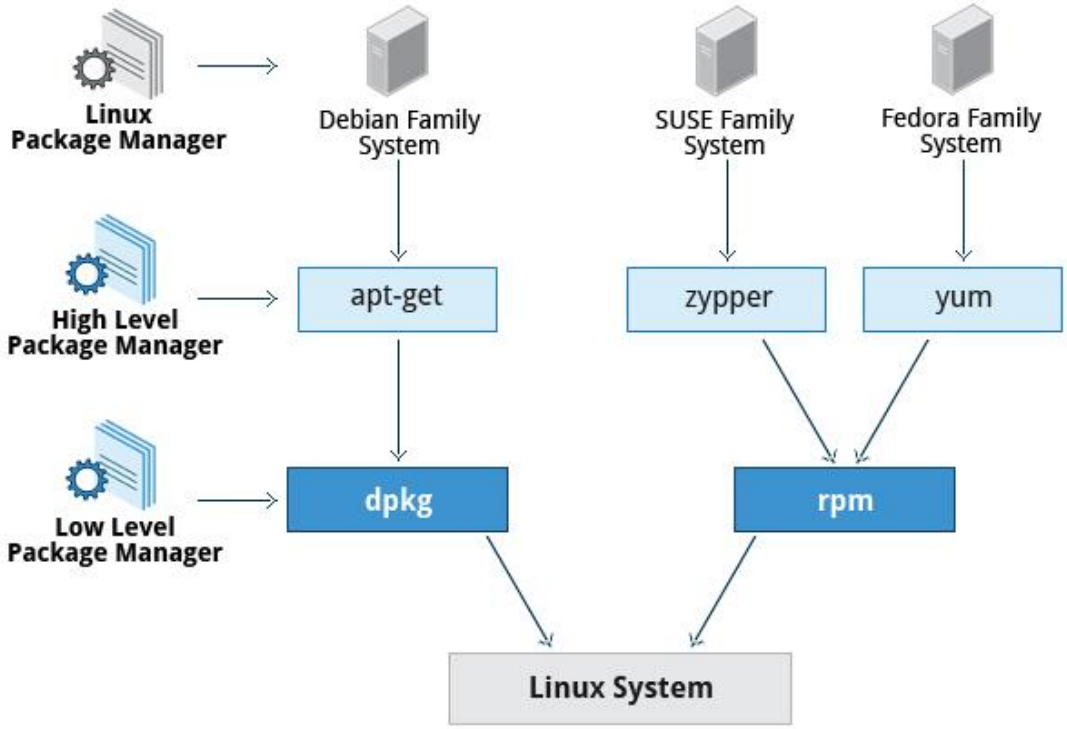
Section 4: Working with Files

Package Management Systems on Linux

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, but provide the same features at a broad level.

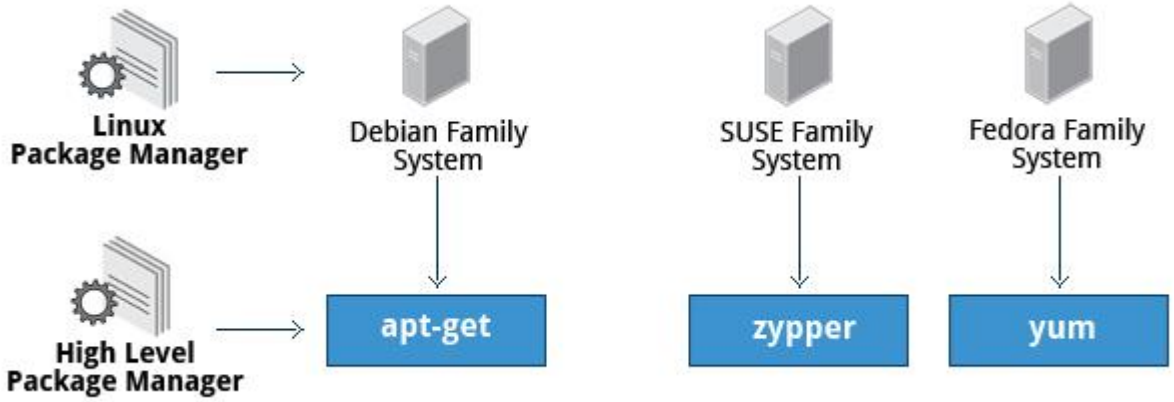
Package Managers: Two Levels



Both package management systems provide two tool levels: a low-level tool (such as **dpkg** or **rpm**), takes care of the details of unpacking individual packages, running scripts, getting the software installed correctly, while a high-level tool (such as **apt-get**, **yum**, or **zypper**) works with groups of packages, downloads packages from the vendor, and figures out dependencies.

Most of the time users need work only 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 you. Be careful, however, as installing a single package could result in many dozens or even hundreds of dependent packages being installed.

Working With Different Package Management Systems



- The **Advanced Packaging Tool** (apt) is the underlying package management system that manages software on Debian-based systems. While it forms the backend for graphical package managers, such as the **Ubuntu Software Center** and **synaptic**, its native user interface is at the command line, with programs that include apt-get and apt-cache.
- **Yellowdog Updater Modified** (**yum**) is an open-source command-line package-management utility for RPM-compatible Linux systems, basically what we have called the **Fedora** family. **yum** has both command line and graphical user interfaces.
- **zypper** is a package management system for **openSUSE** that is based on RPM. **zypper** also allows you to manage repositories from the command line. **zypper** is fairly straightforward to use and resembles **yum** quite closely.

To learn the basic packaging commands, click the link below:
Basic Packaging Commands

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

Managing Software Packages on Ubuntu

Which of the following are high level package managers?

- [ ] dpkg
- [x] apt
- [x] yum
- [x] zypper

Which of the following are low level package managers?

- [x] dpkg
- [ ] apt
- [ ] yum
- [x] rpm

Labs

Click the link to download a PDF for the Lab activity.

Command Line Operations_Labs

Command Line Operations_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered.

- Virtual terminals (VT) in Linux are consoles, or command line terminals that use the connected monitor and keyboard.
- Different Linux distributions start and stop the graphical desktop in different ways.
- A terminal emulator program on the graphical desktop works by emulating a terminal within a window on the desktop.
- The Linux system allows you to either log in via text terminal or remotely via the console.
- When typing your password, nothing is printed to the terminal, not even a generic symbol to indicate that you typed.
- The preferred method to shut down or reboot the system is to use the **shutdown** command.
- There are two types of **pathnames:** absolute and relative.
- 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 **hard** and **soft** (**symbolic**) links is extremely useful in Linux.
- **cd** remembers where you were last, and lets you get back there with cd -.
- **locate** performs a database search to find all file names that match a given pattern.
- **find** locates files recursively from a given directory or set of directories.
- **find** is able to run commands on the files that it lists, when used with the -exec option.
- **touch** is used to set the access, change, and edit times of files as well as to create empty files.
- The **Advanced Packaging Tool** (apt) package management system is used to manage installed software on Debian-based systems.
- You can use the **Yellowdog Updater Modified** (yum) open-source command-line package-management utility for **RPM**-compatible Linux operating systems.
- The **zypper** package management system is based on RPM and used for openSUSE.

Chapter 7: Finding Linux Documentation

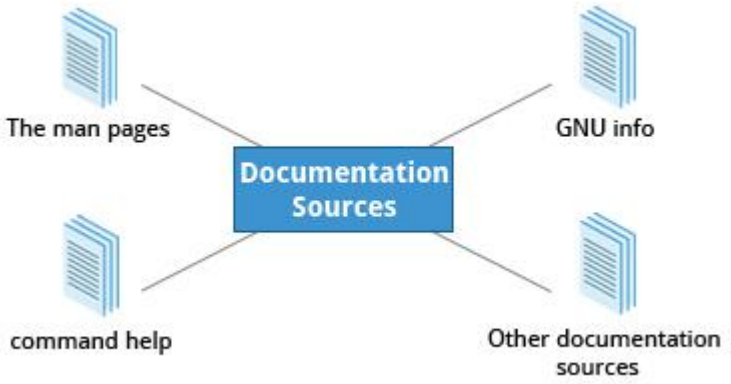
Learning Objectives

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

- Use different sources of documentation.
- Use the **man pages**.
- Access the GNU **info** system.
- Use the **help** command and --help option.
- Use other documentation sources.

Section 1: Documentation Sources

Introduction to Linux Documentation Sources



Whether you are an inexperienced user or a veteran, you won't always know how to use various Linux programs and utilities, or what to type at the command line. You will need to consult the help documentation regularly. Because Linux-based systems draw from a large variety of sources, there are numerous reservoirs of documentation and ways of getting help. Distributors consolidate this material and present it in a comprehensive and easy-to-use manner.

Important Linux documentation sources include:

- The **man pages** (short for manual pages)
- GNU **Info**
- The **help** command and **--help** option
- Other Documentation Sources, e.g. https://www.gentoo.org/doc/en/

Section 2: Man pages

The **man pages** are the most often-used source of Linux documentation. They provide in-depth documentation about many programs and utilities as well as other topics, including configuration files, system calls, library routines, and the kernel.

Typing **man** with a topic name as an argument retrieves the information stored in the topic's **man pages**. Some Linux distributions require every installed program to have a corresponding **man** page, which explains the depth of coverage. (Note: **man** is actually an abbreviation for **manual**.) The **man pages** structure were first introduced in the early UNIX versions of the early 1970s.

The **man pages** are often converted to:

- Web pages
- Published books
- Graphical help
- Other formats

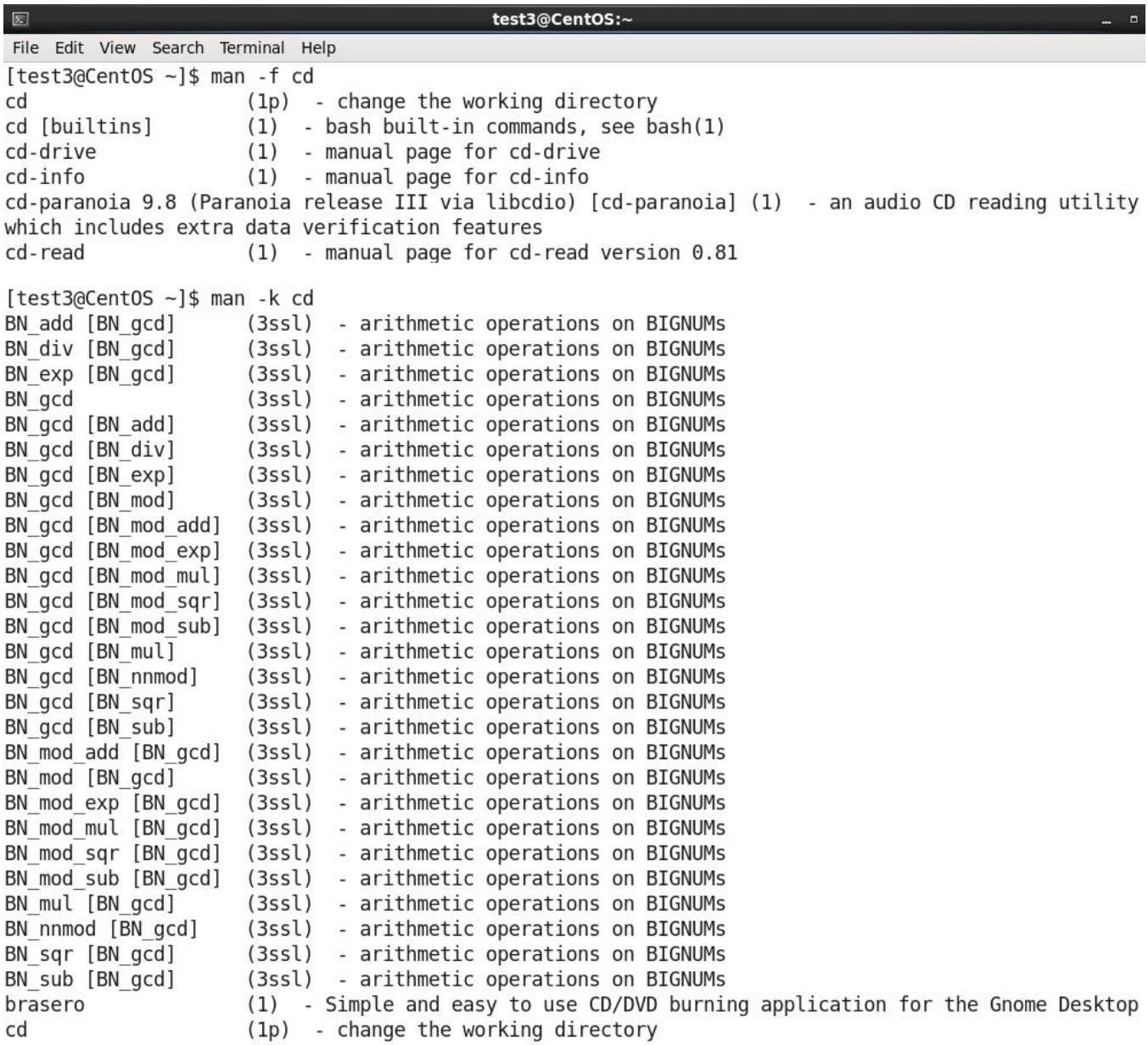
man

The **man** program searches, formats, and displays the information contained in the **man pages**. Because many topics have a lot of information, output is piped through a **terminal pager** program such as **less** to be viewed one page at a time; at the same time the information is formatted for a good visual display.

When no options are given, by default one sees only the dedicated page specifically about the topic. You can broaden this to view all **man pages** containing a string in their name by using the -f option. You can also view all **man pages** that discuss a specified subject (even if the specified subject is not present in the name) by using the –k option.

man –f generates the same result as typing **whatis**.

man –k generates the same result as typing **apropos.**



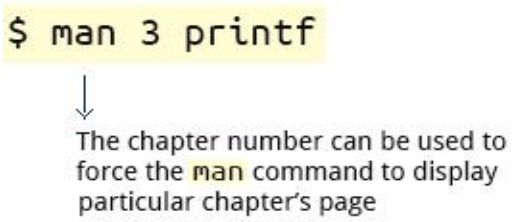
Manual Chapters

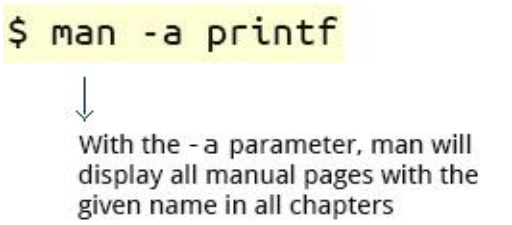
The **man pages** are divided into nine numbered chapters (1 through 9). Sometimes, a letter is appended to the chapter number to identify a specific topic. For example, many pages describing part of the **X Window** API are in chapter 3X.

The chapter number can be used to force **man** to display the page from a particular chapter; it is common to have multiple pages across multiple chapters with the same name, especially for names of library functions or system calls.

With the -a parameter, **man** will display all pages with the given name in all chapters, one after the other.

$ man 3 printf
$ man -a printf





What is displayed when running man with no options other than the topic as an argument?

- All man pages in sequence with the given name in all chapters.
- The manual page for the given topic. The manual page for the given topic.
- The man pages with a word in the name.
- The man pages discussing the subject.

What does the man –k command display?

- All man pages in sequence with the given name in all chapters.
- The man pages with a word in the name.
- The manual page for the given topic.
- A list of the man pages discussing the subject.

What does the man –a command display?

- All man pages in sequence with the given name in all chapters.
- The man pages with a word in the name.
- The manual page for the given topic.
- The man pages discussing the subject.

Section 3: GNU info

GNU Info System

The next source of Linux documentation is the **GNU Info System**.

This is the **GNU** project's standard documentation format (**info**) which it prefers as an alternative to **man**. The **info** system is more free-form and supports linked sub-sections.

Functionally, the **GNU Info System** resembles **man** in many ways. However, topics are connected using links (even though its design predates the World Wide Web). Information can be viewed through either a command line interface, a graphical help utility, printed or viewed online.

Command Line Info Browser

Typing **info** with no arguments in a terminal window displays an index of available topics. You can browse through the topic list using the regular movement keys: **arrows**, **Page Up**, and **Page Down**.

You can view help for a particular topic by typing

info <topic name>.

The system then searches for the topic in all available **info** files.

Some useful keys are: **q** to quit, **h** for help, and **Enter** to select a menu item.

info Page Structure

The topic which you view in the **info** page is called a **node**.

Nodes are similar to sections and subsections in written documentation. You can move between nodes or view each node sequentially. Each node may contain **menus** and linked subtopics, or **items**.

Items can be compared to Internet hyperlinks. They are identified by an asterisk (*) at the beginning of the item name. Named items (outside a menu) are identified with double-colons (::) at the end of the item name. Items can refer to other nodes within the file or to other files. The table lists the basic keystrokes for moving between nodes.

Key	Function
n	Go to the next node
p	Go to the previous node
u	Move one node up in the index

What does info topic show?

- Shows the index of topics.
- Quits the info page.
- Shows the info page for the specified topic.
- Selects a menu item.

What are the topics in info pages called?

- ○ Items
- ◉ Nodes
- ○ Sections
- ○ Menus

What does the info command (with no arguments) show?

- ◉ Shows the index of topics.
- ○ Selects a menu item.
- ○ Shows the info page for the specified topic.
- ○ Quits the info page.

Section 4: Help command

Introduction to the help Option

The third source of Linux documentation is use of the **help** option.

Most commands have an available short description which can be viewed using the --help or the -h option along with the command or application. For example, to learn more about the **man** command, you can run the following command:
$ man --help

The --help option is useful as a quick reference and it displays information faster than the **man** or **info** pages.

About the help Command

Some popular commands (such as **echo**) when run in a **bash** command shell silently run their own **built-in** versions of system programs or utilities, because it is more efficient to do so. (We will discuss command shells in great detail later.) To view a synopsis of these built-in commands, you can simply type **help**.

For these built-in commands, **help** performs the same **basic** function as the -h and --help arguments (which we will discuss shortly) perform for stand-alone programs.

The next screen covers a demonstration on how to use **man**, **info** and the **help** option.

Using man, info and the help Option

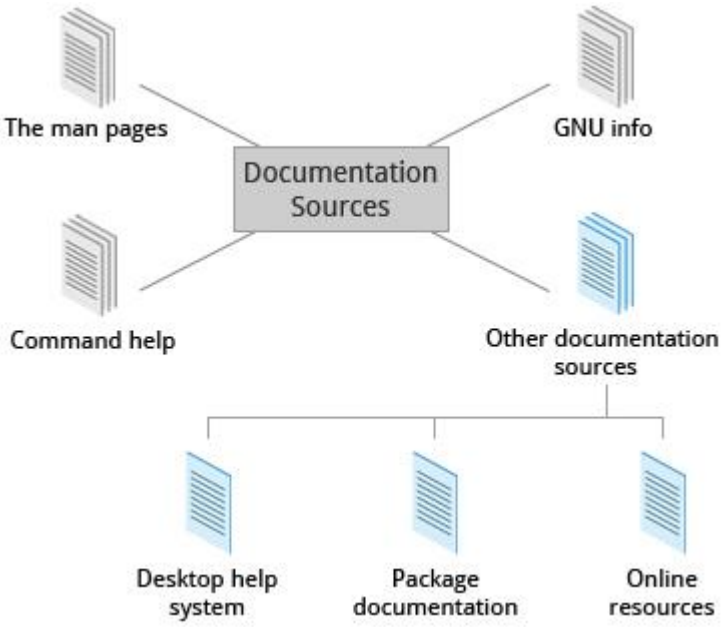
Which of the following are the built-in help options that can be used for quick references?

- ☑ -h
- ☑ --help
- ☐ /help
- ☐ help –

Which command displays a short synopsis of built-in shell commands?
help

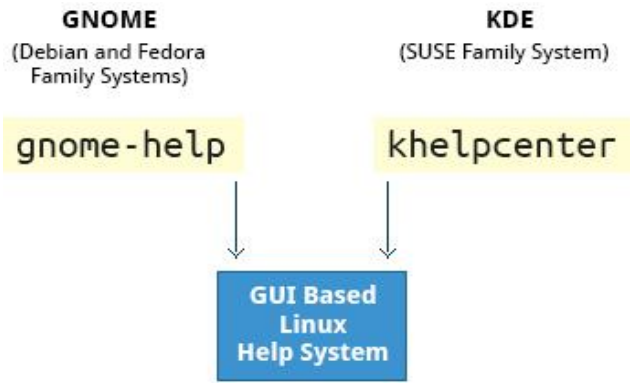
Section 5: Other Documentation sources

In addition to the **man pages**, the **GNU Info System**, and the **help** command, there are other sources of Linux documentation, some examples of which are shown here.



Desktop Help Systems

All Linux desktop systems have a graphical help application. This application is usually displayed as a question-mark icon or an image of a ship's life-preserver. These programs usually contain custom help for the desktop itself and some of its applications, and will often also include graphically rendered **info** and **man pages**.



You can also start the graphical help system from a graphical terminal using the following commands:

- **GNOME**: **gnome-help**
- **KDE: khelpcenter**

Package Documentation

Linux documentation is also available as part of the package management system. Usually this documentation is directly pulled from the upstream source code, but it can also contain information about how the distribution packaged and set up the software.

Such information is placed under the /usr/share/doc directory in a subdirectory named after the package, perhaps including the version number in the name.

Online Resources

There are many places to access online Linux documentation, and a little bit of searching will get you buried in it.

You can also find very helpful documentation for each distribution. Each distribution has its own user-generated forums and wiki sections. Here are just a few links to such sources:

Ubuntu: https://help.ubuntu.com/

CentOS: https://www.centos.org/docs/

OpenSUSE: http://en.opensuse.org/Portal:Documentation

GENTOO: http://www.gentoo.org/doc/en

Labs

Click the links below to download your lab assignments. You can save or print the files.

Finding Linux Documentation_Labs

Finding Linux Documentation_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- The main sources of Linux documentation are the **man pages**, **GNU Info**, the **help** options and command, and a rich variety of online documentation sources.
- The **man** utility searches, formats, and displays **man pages**.
- The **man pages** provide in-depth documentation about programs and other topics about the system including configuration files, system calls, library routines, and the kernel.
- The **GNU Info System** was created by the **GNU** project as its standard documentation. It is robust and is accessible via command line, web, and graphical tools using **info**.
- Short descriptions for commands are usually displayed with the -h or --help argument.
- You can type **help** at the command line to display a synopsis of built-in commands.
- There are many other help resources both on your system and on the Internet.

Chapter 8: File Operations

Learning Objectives

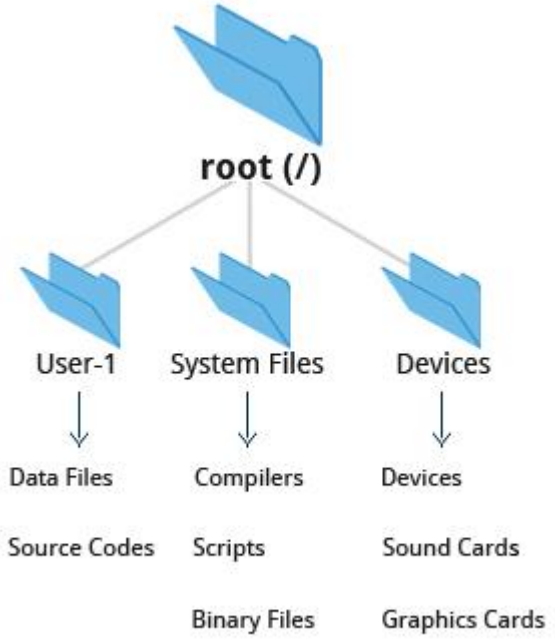
By the end of this chapter, you should be able to:

- Explore the filesystem and its hierarchy.
- Explain the filesystem architecture.
- Compare files and identify different file types.
- Back up and compress data.

Section 1: Filesystems

Introduction to Filesystems

In Linux (and all UNIX-like operating systems) it is often said "Everything is a file", or at least it is treated as such. This means whether you are dealing with normal data files and documents, or with devices such as sound cards and printers, you interact with them through the same kind of Input/Output (I/O) operations. This simplifies things: you open a "file" and perform normal operations like reading the file and writing on it (which is one reason why text editors, which you will learn about in an upcoming section, are so important.)



On many systems (including Linux), the **filesystem** is structured like a tree. The tree is usually portrayed as inverted, and starts at what is most often called the **root directory,** which marks the beginning of the hierarchical filesystem and is also some times referred to as the **trunk,** or simply denoted by */*. The root directory is **not** the same as the root user. The hierarchical filesystem also contains other elements in the path (directory names) which are separated by forward slashes (*/*) as in /usr/bin/awk, where the last element is the actual file name.

In this section, you will learn about some basic concepts including the filesystem hierarchy as well as about **disk partitions**.

Filesystem Hierarchy Standard

The **Filesystem Hierarchy Standard** (**FHS**) grew out of historical standards from early versions of UNIX, such as the **Berkeley Software Distribution** (**BSD**) and others. The FHS provides Linux developers and system administrators with a standard directory structure for the filesystem, which provides consistency between systems and distributions.

Visit http://www.pathname.com/fhs/ for a list of the main directories and their contents in Linux systems.

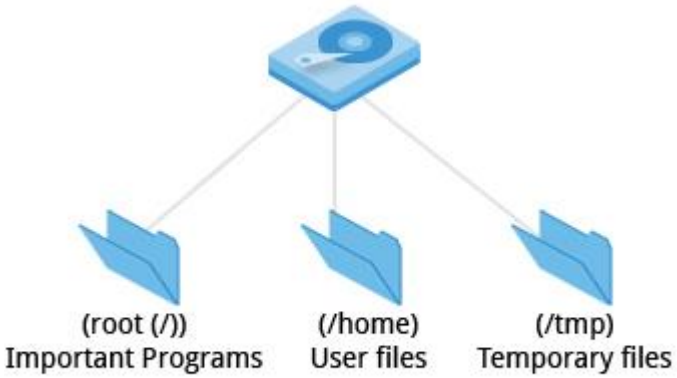
Linux supports various filesystem types created for Linux, along with compatible filesystems from other operating systems such as **Windows** and **MacOS**. Many older, legacy filesystems, such as **FAT**, are supported.

Some examples of filesystem types that Linux supports are:

1. **ext3**, **ext4**, **btrfs**, **xfs** (native Linux filesystems)

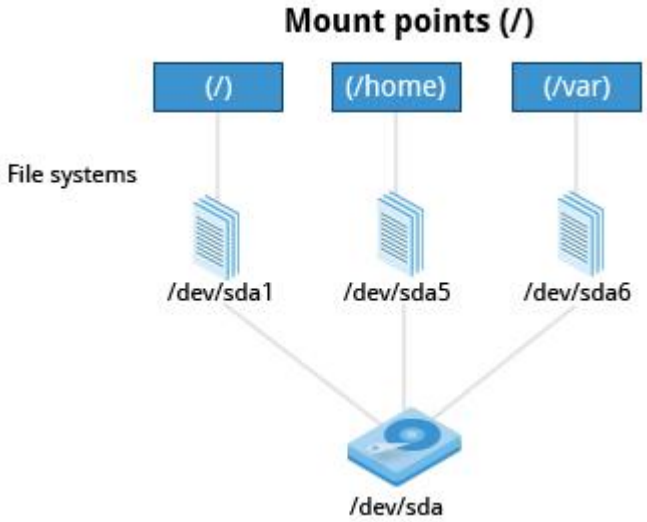
2. **vfat**, **ntfs, hfs** (filesystems from other operating systems)

Partitions in Linux



Each filesystem resides on a hard disk **partition**. Partitions help to organize the contents of disks according to the kind of data contained and how it is used. For example, important programs required to run the system are often kept on a separate partition (known as root or /) than the one that contains files owned by regular users of that system (/home). In addition, temporary files created and destroyed during the normal operation of Linux are often located on a separate partition; in this way, using all available space on a particular partition may not fatally affect the normal operation of the system.

Mount Points



Before you can start using a filesystem, you need to **mount** it to the filesystem tree at a **mount point.** This is simply a directory (which may or may not be empty) where the filesystem is to be attached (mounted). Sometimes you may need to create the directory if it doesn't already exist.

Warning: If you mount a filesystem on a non-empty directory, the former contents of that directory are covered-up and not accessible until the filesystem is unmounted. Thus mount points are usually empty directories.

More About Mount Points

The **mount** command is used to attach a filesystem (which can be local to the computer or, as we shall discuss, on a network) somewhere within the filesystem tree. Arguments include the **device node** and **mount point**. For example,

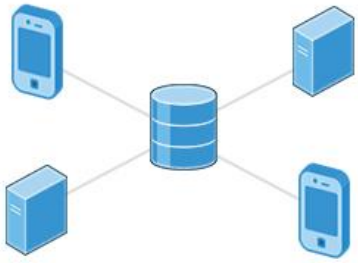
```
$ mount /dev/sda5 /home
```

will attach the filesystem contained in the disk partition associated with the /dev/sda5 device node, into the filesystem tree at the /home mount point. (Note that unless the system is otherwise configured only the root user has permission to run **mount**.) If you want it to be automatically available every time the system starts up, you need to edit the file /etc/fstab accordingly (the name is short for **Filesystem Table**). Looking at this file will show you the configuration of all pre-configured filesystems. man fstab will display how this file is used and how to configure it.

Typing **mount** without any arguments will show all presently mounted filesystems.

The command df -Th (**disk-free**) will display information about mounted filesystems including usage statistics about currently used and available space.

```
[test3@CentOS ~]$ df -Th
Filesystem     Type    Size  Used Avail Use% Mounted on
/dev/sda1      ext4    6.8G  5.5G 1000M  85% /
tmpfs          tmpfs   499M  372K  498M   1% /dev/shm
Shared         vboxsf  369G  111G  258G  31% /media/sf_Shared
[test3@CentOS ~]$ mount
/dev/sda1 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw,rootcontext="system_u:object_r:tmpfs_t:s0")
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
nfsd on /proc/fs/nfsd type nfsd (rw)
Shared on /media/sf_Shared type vboxsf (gid=493,rw)
[test3@CentOS ~]$
```



The Network Filesystem

Using **NFS** (the **Network Filesystem**) is one of the methods used for sharing data across physical systems. Many system administrators mount remote users' home directories on a **server** in order to give them access to the same files and configuration files across multiple **client** systems. This allows the users to log in to different computers yet still have access to the same files and resources.

NFS on the Server

We will now look in detail at how to use NFS on the server machine.

On the server machine, NFS daemons (built-in networking and service processes in Linux) and other system servers are typically started with the following command: sudo service nfs start

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

/projects *.example.com(rw)

This entry allows the directory /projects to be mounted using NFS with read and write (rw) permissions and shared with other hosts in the example.com domain. As we will detail in the next chapter, every file in Linux has 3 possible permissions: **read** (r), **write** (w) and **execute** (x).

*exports ×

```
# /etc/exports: the access control list for filesystems which may be
exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2
(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/home/share 192.168.0.0/255.255.255.0(ro,sync)
```

After modifying the /etc/exports file, you can use the exportfs -av command to notify Linux about the directories you are allowing to be remotely mounted using NFS (restarting NFS with sudo service nfs restart will also work, but is heavier as it halts NFS for a short while before starting it up again).

NFS on the Client

On the client machine, if it is desired to have the remote filesystem mounted automatically upon system boot, the /etc/fstab file is modified to accomplish this. For example, an entry in the client's /etc/fstab file might look like the following:

servername:/projects /mnt/nfs/projects nfs defaults 0 0

You can also mount the remote filesystem without a reboot or as a one-time mount by directly using the mount command:

$ mount servername:/projects /mnt/nfs/projects

Remember, if /etc/fstab is not modified, this remote mount will not be present the next time the system is restarted.

```
[kcs@kcs ~]$ cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Sat May 26 12:09:46 2012
#
# 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
#
/dev/mapper/vg_kcs-lv_root /                       ext4    defaults        1 1
UUID=bc8ca0fd-3a23-4237-9c51-713319ad4191 /boot    ext4    defaults        1 2
/dev/mapper/vg_kcs-lv_home /home                   ext4    defaults        1 2
/dev/mapper/vg_kcs-lv_swap swap                    swap    defaults        0 0
bagend.hobbiton.org:/data  /data                   nfs     defaults        0 0
tmpfs                      /dev/shm                tmpfs   defaults        0 0
devpts                     /dev/pts                devpts  gid=5,mode=620  0 0
sysfs                      /sys                    sysfs   defaults        0 0
proc                       /proc                   proc    defaults        0 0
[kcs@kcs ~]$
```

proc Filesystem

Certain filesystems like the one mounted at /proc are called **pseudo filesystems** because they have no permanent presence anywhere on disk.

The /proc filesystem contains virtual files (files that exist only in memory) that permit viewing constantly varying kernel data. This filesystem contains files and directories that mimic kernel structures and configuration information. It doesn't contain *real* files but runtime system information (e.g. system memory, devices mounted, hardware configuration, etc). Some important files in /proc are:

/proc/cpuinfo
/proc/interrupts
/proc/meminfo
/proc/mounts
/proc/partitions
/proc/version

/proc has subdirectories as well, including:

/proc/<Process-ID-#>
/proc/sys

The first example shows there is a directory for every **process** running on the system which contains vital information about it. The second example shows a virtual directory that contains a lot of information about the

entire system, in particular its hardware and configuration. The /proc filesystem is very useful because the information it reports is gathered only as needed and never needs storage on disk.

Select the command that you could use to verify whether a filesystem is mounted as read only or writable.

- df -Th
- cat /etc/fstab
- mount
- exportfs -av

Filesystems listed in the _____ configuration file will be automatically mounted when the system is brought up into multi-user mode.

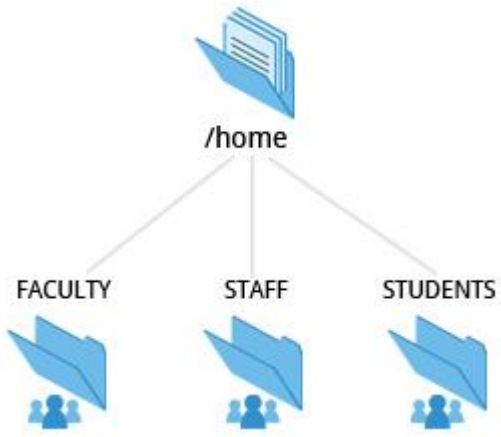
- /etc/nfs
- /etc/fstab
- /mnt/dev
- /dev

Which of the following commands will ensure that the shared files are available over a network?

- exportfs
- exportnfs
- exportfiles
- exportnetwork

Section 2: Filesystem Architecture

Overview of Home Directories



Now that you know about the basics of filesystems, let's learn about the filesystem architecture and directory structure in Linux.

Each user has a **home directory**, usually placed under /home. The /root (slash-root) directory on modern Linux systems is no more than the root user's home directory.

The /home directory is often mounted as a separate filesystem on its own partition, or even exported (shared) remotely on a network through NFS.

Sometimes you may group users based on their department or function. You can then create subdirectories under the /home directory for each of these groups. For example, a school may organize /home with something like the following:

/home/faculty/
/home/staff/
/home/students/

In this section, you will learn to identify and differentiate between the different directories available in Linux.

The /bin and /sbin Directories

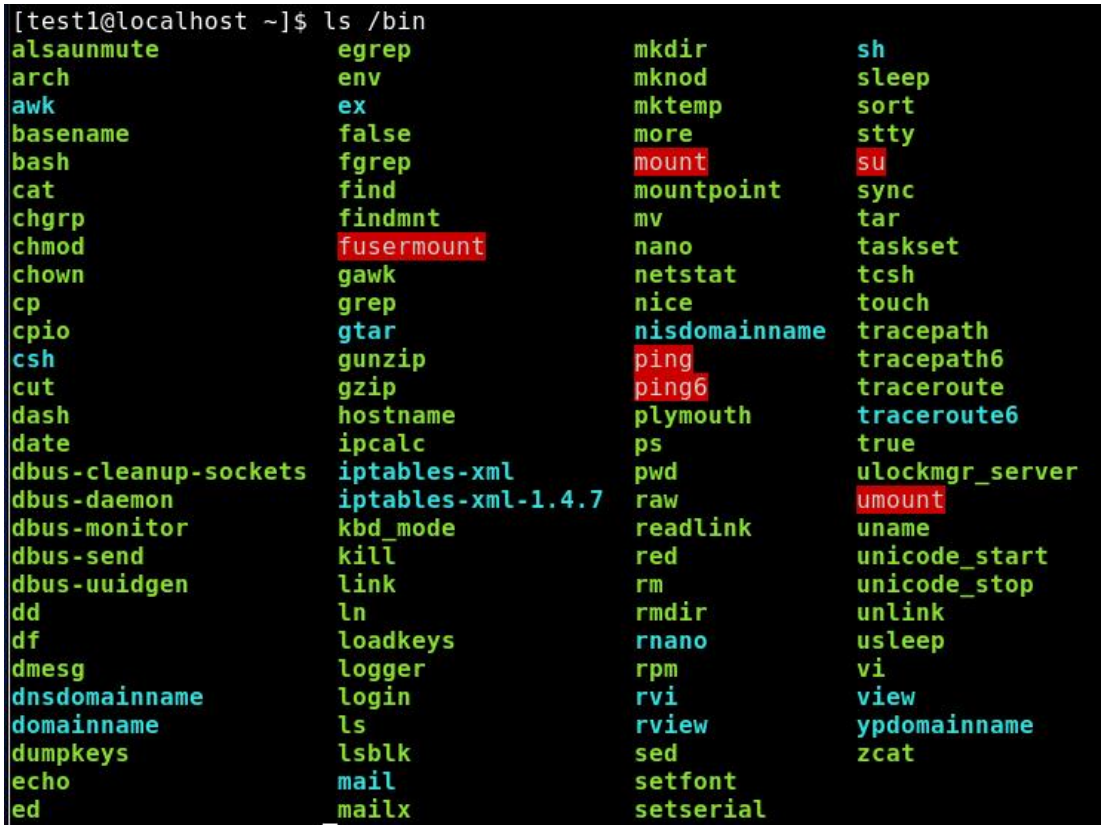
The /bin directory contains executable binaries, essential commands used in single-user mode, and essential commands required by all system users, such as:

Command	Usage
ps	Produces a list of processes along with status information for the system.
ls	Produces a listing of the contents of a directory.
cp	Used to copy files.

To view a list of programs in the /bin directory, type: ls /bin

Commands that are not essential for the system in single-user mode are placed in the /usr/bin directory, while the /sbin directory is used for essential binaries related to system administration, such as **ifconfig** and **shutdown.** There is also a /usr/sbin directory for less essential system administration programs.

Sometimes /usr is a separate filesystem that may not be available/mounted in single-user mode. This was why essential commands were separated from non-essential commands. However, in some of the most modern



Linux systems this distinction is considered obsolete, and /usr/bin and /bin are actually just linked together as are /usr/sbin and /sbin

The /dev Directory

The /dev directory contains **device nodes**, a type of pseudo-file used by most hardware and software devices, except for network devices. This directory is:

- Empty on the disk partition when it is not mounted
- Contains entries which are created by the **udev** system, which creates and manages device nodes on Linux, creating them dynamically when devices are found. The /dev directory contains items such as:
    - /dev/sda1 (first partition on the first hard disk)

- /dev/lp1 (second printer)
- /dev/dvd1 (first DVD drive)

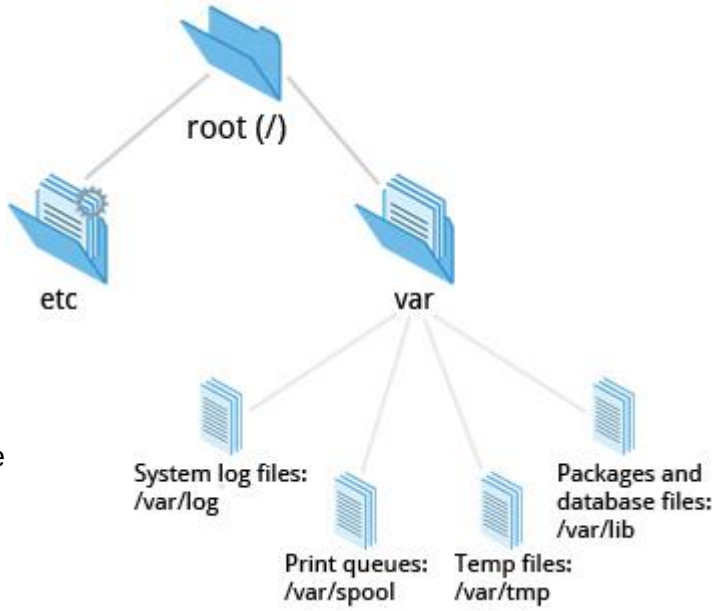
```
[test1@localhost ~]$ ls /dev
agpgart          dvdrw      loop2    net                 ram12   root      stdin   tty19  tty32  tty46  tty6     usbmon1  vcsa6
block            fb         loop3    network_latency     ram13   rtc       stdout  tty2   tty33  tty47  tty60    usbmon2  vga_arbiter
bsg              fb0        loop4    network_throughput  ram14   rtc0      systty  tty20  tty34  tty48  tty61    vcs      vmci
bus              fd         loop5    null                ram15   scd0      tty     tty21  tty35  tty49  tty62    vcs1     vsock
cdrom            full       loop6    nvram               ram2    sda       tty0    tty22  tty36  tty5   tty63    vcs2     zero
cdrw             fuse       loop7    oldmem              ram3    sda1      tty1    tty23  tty37  tty50  tty7     vcs3
char             hidraw0    lp0      parport0            ram4    sda2      tty10   tty24  tty38  tty51  tty8     vcs4
console          hpet       lp1      port                ram5    sda3      tty11   tty25  tty39  tty52  tty9     vcs5
core             hugepages  lp2      ppp                 ram6    sg0       tty12   tty26  tty4   tty53  ttyS0    vcs6
cpu              hvc0       lp3      ptmx                ram7    sg1       tty13   tty27  tty40  tty54  ttyS1    vcsa
cpu_dma_latency  input      MAKEDEV  pts                 ram8    shm       tty14   tty28  tty41  tty55  ttyS2    vcsa1
crash            kmsg       mapper   ram0                ram9    snapshot  tty15   tty29  tty42  tty56  ttyS3    vcsa2
disk             log        mcelog   ram1                random  snd       tty16   tty3   tty43  tty57  uinput   vcsa3
dmmidi           loop0      mem      ram10               raw     sr0       tty17   tty30  tty44  tty58  urandom  vcsa4
dvd              loop1      midi     ram11               rfkill  stderr    tty18   tty31  tty45  tty59  usbmon0  vcsa5
[test1@localhost ~]$
```

The /var and /etc Directories

The /var directory contains files that are expected to change in size and content as the system is running (**var** stands for **variable**) such as the entries in the following directories:

- System log files: /var/log
- Packages and database files: /var/lib
- Print queues: /var/spool
- Temp files: /var/tmp

The /var directory may be put in its own filesystem so that growth of the files can be accommodated and the file sizes do not fatally affect the system. Network services directories such as /var/ftp (the FTP service) and /var/www (the HTTP web service) are also found under /var.



The /etc directory is the home for system configuration files. It contains no binary programs, although there are some executable scripts. For example, the file resolv.conf tells the system where to go on the network to obtain host name to IP address mappings (DNS). Files like passwd,shadow and group for managing user accounts are found in the /etc directory. System run level scripts are found in subdirectories of /etc. For example, /etc/rc2.d contains links to scripts for entering and leaving run level 2. The rc directory historically stood for *Run Commands*. Some distros extend the contents of /etc. For example, **Red Hat** adds the sysconfig subdirectory that contains more configuration files.

The /boot Directory

The /boot directory contains the few essential files needed to boot the system. For every alternative kernel installed on the system there are four files:

1. vmlinuz: the compressed Linux kernel, required for booting
2. initramfs: the initial ram filesystem, required for booting, sometimes called initrd, not initramfs
3. config: the kernel configuration file, only used for debugging and bookkeeping
4. System.map: kernel symbol table, only used for debugging

Each of these files has a kernel version appended to its name.

The **Grand Unified Bootloader** (**GRUB**) files (such as /boot/grub/grub.conf or /boot/grub2/grub2.cfg) are also found under the /boot directory.

The images show an example listing of the /boot directory, taken from a **CentOS** system that has three installed kernels. Names would vary and things would look somewhat different on a different distribution.

```
[test3@CentOS boot]$ ls -l
total 71676
-rw-r--r--. 1 root root   105200 Mar 26 01:55 config-2.6.32-431.11.2.el6.x86_64
-rw-r--r--. 1 root root   105200 Jun 20 03:11 config-2.6.32-431.20.3.el6.x86_64
-rw-r--r--. 1 root root   105195 Nov 22  2013 config-2.6.32-431.el6.x86_64
drwxr-xr-x. 3 root root     4096 May  2 15:56 efi
drwxr-xr-x. 2 root root     4096 Jul 23 19:19 grub
-rw-------. 1 root root 17529835 May  8 11:03 initramfs-2.6.32-431.11.2.el6.x86_64.img
-rw-------. 1 root root 17518826 Jul  4 17:38 initramfs-2.6.32-431.20.3.el6.x86_64.img
-rw-------. 1 root root 17466788 May  2 15:58 initramfs-2.6.32-431.el6.x86_64.img
-rw-r--r--. 1 root root   193798 Mar 26 01:56 symvers-2.6.32-431.11.2.el6.x86_64.gz
-rw-r--r--. 1 root root   193901 Jun 20 03:12 symvers-2.6.32-431.20.3.el6.x86_64.gz
-rw-r--r--. 1 root root   193758 Nov 22  2013 symvers-2.6.32-431.el6.x86_64.gz
-rw-r--r--. 1 root root  2518941 Mar 26 01:55 System.map-2.6.32-431.11.2.el6.x86_64
-rw-r--r--. 1 root root  2519609 Jun 20 03:11 System.map-2.6.32-431.20.3.el6.x86_64
-rw-r--r--. 1 root root  2518236 Nov 22  2013 System.map-2.6.32-431.el6.x86_64
-rwxr-xr-x. 1 root root  4129872 Mar 26 01:55 vmlinuz-2.6.32-431.11.2.el6.x86_64
-rwxr-xr-x. 1 root root  4131248 Jun 20 03:11 vmlinuz-2.6.32-431.20.3.el6.x86_64
-rwxr-xr-x. 1 root root  4128368 Nov 22  2013 vmlinuz-2.6.32-431.el6.x86_64
```

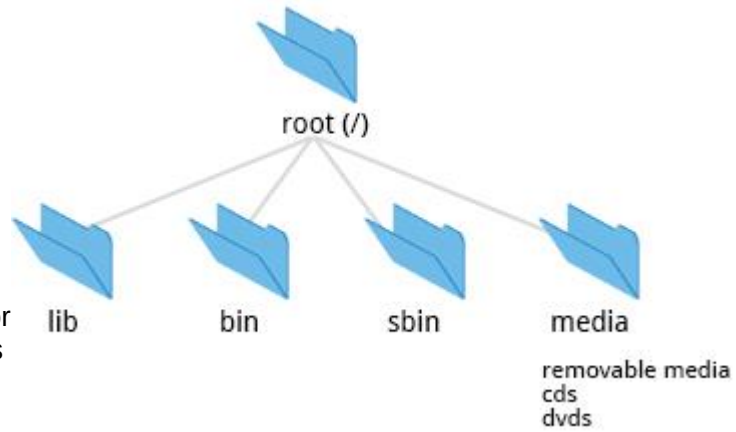
boot

File Edit View Places Help

Name	Size	Type	Date Modified
efi	1 item	folder	Fri 02 May 2014 03:56:06 PM IST
grub	16 items	folder	Wed 23 Jul 2014 07:19:13 PM IST
config-2.6.32-431.11.2.el6.x86_64	102.7 KB	MPSub subtitles	Wed 26 Mar 2014 01:55:55 AM IST
config-2.6.32-431.20.3.el6.x86_64	102.7 KB	MPSub subtitles	Fri 20 Jun 2014 03:11:08 AM IST
config-2.6.32-431.el6.x86_64	102.7 KB	MPSub subtitles	Fri 22 Nov 2013 09:10:55 AM IST
initramfs-2.6.32-431.11.2.el6.x86_64.img	16.7 MB	unknown	Thu 08 May 2014 11:03:27 AM IST
initramfs-2.6.32-431.20.3.el6.x86_64.img	16.7 MB	unknown	Fri 04 Jul 2014 05:38:33 PM IST
initramfs-2.6.32-431.el6.x86_64.img	16.7 MB	unknown	Fri 02 May 2014 03:58:21 PM IST
symvers-2.6.32-431.11.2.el6.x86_64.gz	189.3 KB	Gzip archive	Wed 26 Mar 2014 01:56:50 AM IST
symvers-2.6.32-431.20.3.el6.x86_64.gz	189.4 KB	Gzip archive	Fri 20 Jun 2014 03:12:12 AM IST
symvers-2.6.32-431.el6.x86_64.gz	189.2 KB	Gzip archive	Fri 22 Nov 2013 09:11:44 AM IST
System.map-2.6.32-431.11.2.el6.x86_64	2.4 MB	plain text document	Wed 26 Mar 2014 01:55:55 AM IST
System.map-2.6.32-431.20.3.el6.x86_64	2.4 MB	plain text document	Fri 20 Jun 2014 03:11:08 AM IST
System.map-2.6.32-431.el6.x86_64	2.4 MB	plain text document	Fri 22 Nov 2013 09:10:55 AM IST
vmlinuz-2.6.32-431.11.2.el6.x86_64	3.9 MB	program	Wed 26 Mar 2014 01:55:55 AM IST
vmlinuz-2.6.32-431.20.3.el6.x86_64	3.9 MB	program	Fri 20 Jun 2014 03:11:08 AM IST
vmlinuz-2.6.32-431.el6.x86_64	3.9 MB	program	Fri 22 Nov 2013 09:10:55 AM IST

The /lib and /media Directories

/lib contains libraries (common code shared by applications and needed for them to run) for the essential programs in /bin and /sbin. These library filenames either start with ld or lib, for example, /lib/libncurses.so.5.7.



Most of these are what are known as **dynamically loaded libraries** (also known as **shared libraries** or **Shared Objects (SO)**). On some Linux distributions there exists a /lib64 directory containing 64-bit libraries, while /lib contains 32-bit versions.

Kernel **modules** (kernel code, often device drivers, that can be loaded and unloaded without re-starting the system) are located in /lib/modules/<kernel-version-number>.

The /media directory is typically located where removable media, such as CDs, DVDs and USB drives are mounted. Unless configuration prohibits it, Linux automatically mounts the removable media in the /media directory when they are detected.

Additional Directories Under /:

The following is a list of additional directories under /and their use:

Directory name	Usage
/opt	Optional application software packages.
/sys	Virtual pseudo-filesystem giving information about the system and the hardware. Can be used to alter system parameters and for debugging purposes.
/srv	Site-specific data served up by the system. Seldom used.
/tmp	Temporary files; on some distributions erased across a reboot and/or may actually be a ramdisk in memory.
/usr	Multi-user applications, utilities and data.

Subdirectories under /usr

The /usr directory contains non-essential programs and scripts (in the sense that they should not be needed to initially boot the system) and has at least the following sub-directories:

Directory name	Usage
/usr/include	Header files used to compile applications.
/usr/lib	Libraries for programs in /usr/bin and /usr/sbin.
/usr/lib64	64-bit libraries for 64-bit programs in /usr/bin and /usr/sbin.
/usr/sbin	Non-essential system binaries, such as system daemons.
/usr/share	Shared data used by applications, generally architecture-independent.
/usr/src	Source code, usually for the Linux kernel.
/usr/X11R6	**X Window** configuration files; generally obsolete.
/usr/local	Data and programs specific to the local machine. Subdirectories include bin, sbin, lib, share, include, etc.
/usr/bin	This is the primary directory of executable commands on the system.

Which is the home directory of the *super user*?

- ○ /bin
- ◉ /root
- ○ /dev
- ○ /boot

Where can you find the device nodes of a Linux system?

- ○ /etc
- ○ /mnt
- ◉ /dev
- ○ /var

Section 3: Comparing Files and Files Types

Comparing Files

Now that you know about the filesystem and its structure, let's learn how to manage files and directories.

diff is used to compare files and directories. This often-used utiility program has many useful options (see man diff) including:

diff Option	Usage
-c	Provides a listing of differences that include 3 lines of **context** before and after the lines differing in content
-r	Used to **recursively** compare subdirectories as well as the current directory
-i	**Ignore** the case of letters
-w	Ignore differences in spaces and tabs (**white space**)

To compare two files, at the command prompt, type diff <filename1> <filename2>

In this section, you will learn additional methods for comparing files and how to apply **patches** to files.

Using diff3 and patch

You can compare three files at once using **diff3**, which uses one file as the reference basis for the other two. For example, suppose you and a co-worker both have made modifications to the same file working at the same time independently. **diff3** can show the differences based on the common file you both started with. The syntax for **diff3** is as follows:

```
$ diff3 MY-FILE COMMON-FILE YOUR-FILE
```

Many modifications to source code and configuration files are distributed utilizing **patches**, which are applied, not suprisingly, with the **patch** program. A patch file contains the **deltas** (changes) required to update an older version of a file to the new one. The patch files are actually produced by running **diff** with the correct options, as in:

```
$ diff -Nur originalfile newfile > patchfile
```

Distributing just the patch is more concise and efficient than distributing the entire file. For example, if only one line needs to change in a file that contains 1,000 lines, the **patch** file will be just a few lines long.

To apply a patch you can just do either of the two methods below:

```
$ patch -p1 < patchfile
$ patch originalfile patchfile
```

The first usage is more common as it is often used to apply changes to an entire directory tree, rather than just one file as in the second example. To understand the use of the -p1 option and many others, see the **man** page for **patch**.

The graphic shows a patch file produced by **diff**.

```
[test3@CentOS ~]$ cat myfile.txt
Hello
This is your myfile
[test3@CentOS ~]$ cat commonfile.txt
Hello
This is your commonfile
[test3@CentOS ~]$ cat yourfile.txt
Hello
This is your yourfile
[test3@CentOS ~]$ diff3 myfile.txt commonfile.txt yourfile.txt
====
1:2c
  This is your myfile
2:2c
  This is your commonfile
3:2c
  This is your yourfile
[test3@CentOS ~]$
```

Using the 'file' utility

In Linux, a file's extension often does not categorize it the way it might in other operating systems. One can not assume that a file named file.txt is a text file and not an executable program. In Linux a file name is generally more meaningful to the user of the system than the system itself; in fact most applications directly examine a file's contents to see what kind of object it is rather than relying on an extension. This is very different from the way **Windows** handles filenames, where a filename ending with .exe, for example, represents an executable binary file.

The real nature of a file can be ascertained by using the **file** utility. For the file names given as arguments, it examines the contents and certain characteristics to determine whether the files are plain text, shared libraries, executable programs, scripts, or something else.

```
[test1@localhost ~]$ ls
c15s3.sh    c15s9.sh   Documents  func-call-ex.sh   func-ex.sh~  messages~  Public   sam1.sh~  sam3.sh~  Templates
c15s3.sh~   c15s9.sh~  Downloads  func-call-ex.sh~  if-demo.sh   Music      sam1     sam2.sh   sam4.sh   title
c15s9b.sh   def        example1~  func-ex           if-demo.sh~  new-test1  sam1~    sam2.sh~  sam4.sh~  Videos
c15s9b.sh~  Desktop    example2   func-ex.sh        messages     Pictures   sam1.sh  sam3.sh   sample
[test1@localhost ~]$ file if-demo.sh~
if-demo.sh~: Bourne-Again shell script text executable
[test1@localhost ~]$ file if-demo.sh
if-demo.sh: Bourne-Again shell script text executable
[test1@localhost ~]$ file messages
messages: ASCII English text
[test1@localhost ~]$ file sample
sample: ASCII text
[test1@localhost ~]$ file Videos
Videos: directory
[test1@localhost ~]$ file /usr/bin/perl
/usr/bin/perl: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.18, stripped
[test1@localhost ~]$ file /dev
/dev: directory
[test1@localhost ~]$
```

Try-It-Yourself: Comparing Files

To practice, click the link provided below.

Comparing Files

In a production environment, you have created a file (**MyFile1**) with reference to an existing file (**MyFile2**). Your colleague has also created a file (**MyFile3**) based on the same existing file. Tasks to be performed:

1. Compare the contents of your file and the reference file, using diff.

2. Compare the differences between the files created by you and your friend with reference to the existing file, using diff3.

```
[test3@CentOS ~]$ diff MyFile1 MyFile2
1c1
< This data is available in MyFile1
---
> This data is available in MyFile2
[test3@CentOS ~]$ diff3 MyFile1 MyFile2 MyFile3
1:1c
This data is available in MyFile1
2:1c
This data is available in MyFile2
3:1c
This data is available in MyFile3
[test3@CentOS ~]$
```

Using file

In a production environment, you have created the following items:

1. A shell script named **sam1.**
2. An empty file named **new-test1**.
3. A directory named **videos**.

Tasks to be performed: Find the type of each of the above files/directories, using file.

```
[test3@CentOS ~]$ file sam1
sam1: a /bin/bash script text executable
[test3@CentOS ~]$ file new-test1
new-test1: empty
[test3@CentOS ~]$ file videos
videos: directory
[test3@CentOS ~]$
```

Which command is used to compare three files?

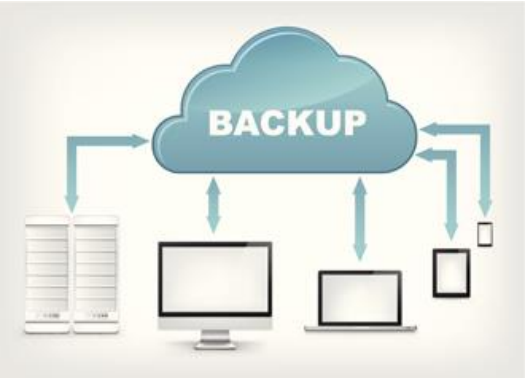
diff3

Which command will show you the type of the file named **data**?

- diff3 data
- patch data
- file data
- diff data

Section 4: Backing Up and Compressing Data

Backing Up Data



There are many ways you can back up data or even your entire system. Basic ways to do so include use of simple copying with **cp** and use of the more robust **rsync**.

Both can be used to synchronize entire directory trees. However, **rsync** is more efficient because it checks if the file being copied already exists. If the file exists and there is no change in size or modification time, **rsync** will avoid an unnecessary copy and save time. Furthermore, because **rsync** copies only the parts of files that have actually changed, it can be very fast.

cp can only copy files to and from destinations on the local machine (unless you are copying to or from a filesystem mounted using NFS), but **rsync** can also be used to copy files from one machine to another. Locations are designated in the target:path form 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.

rsync is very efficient when recursively copying one directory tree to another, because only the differences are transmitted over the network. One often synchronizes the destination directory tree with the origin, using the -r option to recursively walk down the directory tree copying all files and directories below the one listed as the source.

Using rsync

rsync is a very powerful utility. For example, a very useful way to back up a project directory might be to use the following command:

```
$ rsync -r project-X archive-machine:archives/project-X
```

Note that **rsync** can be very destructive! Accidental misuse can do a lot of harm to data and programs by inadvertently copying changes to where they are not wanted. Take care to specify the correct options and paths. It is highly recommended that you first test your **rsync** command using the -dry-run option to ensure that it provides the results that you want.

To use **rsync** at the command prompt, type rsync sourcefile destinationfile, where either file can be on the local machine or on a networked machine.

The contents of sourcefile are copied to destinationfile.

Compressing Data

File data is often compressed to save disk space and reduce the time it takes to transmit files over networks.

Linux uses a number of methods to perform this compression including:

Command	Usage
gzip	The most frequently used Linux compression utility
bzip2	Produces files significantly smaller than those produced by **gzip**
xz	The most space efficient compression utility used in Linux
zip	Is often required to examine and decompress archives from other operating systems

These techniques vary in the efficiency of the compression (how much space is saved) and in how long they take to compress; generally the more efficient techniques take longer. Decompression time doesn't vary as much across different methods.

In addition the **tar** utility is often used to group files in an **archive** and then compress the whole archive at once.

Compressing Data Using gzip

gzip is the most oftenly used Linux compression utility. It compresses very well and is very fast. The following table provides some usage examples:

Command	Usage
gzip *	Compresses all files in the current directory; each file is compressed and renamed with a .gz extension.
gzip -r projectX	Compresses all files in the projectX directory along with all files in all of the directories under projectX.
gunzip foo	De-compresses foo found in the file foo.gz. Under the hood, gunzip command is actually the same as gzip –d.

Compressing Data Using bzip2

bzip2 has syntax that is similar to **gzip** but it uses a different compression algorithm and produces significantly smaller files, at the price of taking a longer time to do its work. Thus, It is more likely to be used to compress larger files.

Examples of common usage are also similar to **gzip**:

Command	Usage
bzip2 *	Compress all of the files in the current directory and replaces each file with a file renamed with a .bz2 extension.
bunzip2 *.bz2	Decompress all of the files with an extension of .bz2 in the current directory. Under the hood, bunzip2 is the same as calling bzip2 -d.

Compress Data Using xz

xz is the most space efficient compression utility used in Linux and is now used by www.kernel.org to store archives of the Linux kernel. Again it trades a slower compression speed for an even higher compression ratio.

Command	Usage
$ xz *	Compress all of the files in the current directory and replace each file with one with a .xz extension.
xz foo	Compress the file foo into foo.xz using the default compression level (-6), and remove foo if compression succeeds.
xz -dk bar.xz	Decompress bar.xz into bar and don't remove bar.xz even if decompression is successful.
xz -dcf a.txt b.txt.xz > abcd.txt	Decompress a mix of compressed and uncompressed files to standard output, using a single command.
$ xz -d *.xz	Decompress the files compressed using xz.

Compressed files are stored with a .xz extension.

Handling Files Using zip

The **zip** program is not often used to compress files in Linux, but is often required to examine and decompress archives from other operating systems. It is only used in Linux when you get a zipped file from a **Windows** user. It is a legacy program.

Command	Usage
zip backup *	Compresses all files in the current directory and places them in the file backup.zip.
zip -r backup.zip ~	Archives your login directory (~) and all files and directories under it in the file backup.zip.
unzip backup.zip	Extracts all files in the file backup.zip and places them in the current directory.

Archiving and Compressing Data Using tar

Historically, **tar** stood for "tape archive" and was used to archive files to a magnetic tape. It allows you to create or extract files from an archive file, often called a **tarball**. At the same time you can optionally compress while creating the archive, and decompress while extracting its contents.

Here are some examples of the use of **tar**:

Command	Usage
$ tar xvf mydir.tar	Extract all the files in mydir.tar into the mydir directory
$ tar zcvf mydir.tar.gz mydir	Create the archive and compress with gzip
$ tar jcvf mydir.tar.bz2 mydir	Create the archive and compress with bz2
$ tar Jcvf mydir.tar.xz mydir	Create the archive and compress with xz
$ tar xvf mydir.tar.gz	Extract all the files in mydir.tar.gz into the mydir directory. Note you do **not** have to tell tar it is in gzip format.

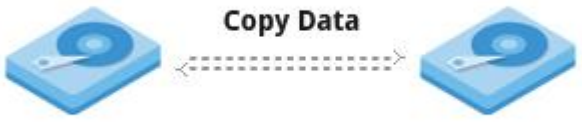
You can separate out the archiving and compression stages, as in:

```
$ tar mydir.tar mydir ; gzip mydir.tar
$ gunzip mydir.tar.gz ; tar xvf mydir.tar
```

but this is slower and wastes space by creating an unneeded intermediary .tar file.

Disk-to-Disk Copying

The **dd** program is very useful for making copies of raw disk space. For example, to back up your **Master Boot Record** (**MBR**) (the first 512 byte sector on the disk that contains a table describing the partitions on that disk), you might type:




```
dd if=/dev/sda of=sda.mbr bs=512 count=1
```

To use **dd** to make a copy of one disk onto another, (**WARNING!**) **deleting everything that previously existed on the second disk**, type:

```
dd if=/dev/sda of=/dev/sdb
```

An exact copy of the first disk device is created on the second disk device.

Do not experiment with this command as written above as it can erase a hard disk!

Exactly what the name **dd** stands for is an often-argued item. The words **data definition** is the most popular theory and has roots in early **IBM** history. Often people joke that it means **disk destroyer** and other variants such as **delete data**!

While copying one directory to a similar directory on another system over the network, **rsync** copies ____.

- all of the files
- all of the sub-directories
- all of the files and sub-directories
- only the differences between the directories

The command zip -r backup.zip ~ will archive which of the following in the file backup.zip?

- The user's home directory
- The user's entire home directory tree
- The entire Hard Disk
- The entire festival directory

The ____ utility compresses all of the files in the projectX directory along with all of the files in all of the directories under projectX.

- $ gzip *
- $ gzip -r projectX
- $ gzip -d projectX
- $ gunzip projectX

The two commands gunzip foo and gzip -d foo do the same thing.

- True
- False

Labs

Click the link to download a PDF for the Lab activity.

File Operations_Labs

File Operations_Solutions

LAB2

sudo tar cvf /tmp/backup.tar ~

sudo tar zcvf /tmp/backup.tar.gz ~

ls -lah /tmp/

sudo rm -i /tmp/backup.tar /tmp/backup.tar.gz

LAB3

cd /tmp

cp /etc/group /tmp

dd if=/tmp/group of=/tmp/GROUP conv=ucase

-a, --text treat all files as text

--strip-trailing-cr strip trailing carriage return on input

-u, -U NUM, --unified[=NUM] output NUM (default 3) lines of unified context

diff -au /tmp/group /tmp/GROUP > /tmp/group.diff

patch /tmp/group /tmp/group.diff

diff group GROUP

(we should get no output after we are done)

Summary

The key concepts covered in this chapter are:

- The filesystem tree starts at what is often called the root directory (or trunk, or /).
- The **Filesystem Hierarchy Standard** (FHS) provides Linux developers and system administrators a standard directory structure for the filesystem.
- Partitions help to segregate files according to usage, ownership and type.
- Filesystems can be **mounted** anywhere on the main filesystem tree at a **mount point**. Automatic filesystem mounting can be set up by editing /etc/fstab.
- **NFS** (The **Network Filesystem**) is a useful method for sharing files and data through the network systems.
- Filesystems like /proc are called pseudo filesystems because they exist only in memory.
- /root (slash-root) is the home directory for the root user.

- /var may be put in its own filesystem so that growth can be contained and not fatally affect the system.
- /boot contains the basic files needed to boot the system
- **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.
- File extensions in Linux do not necessarily mean that a file is of a certain type.
- **cp** is used to copy files on the local machine while **rsync** can also be used to copy files from one machine to another as well as synchronize contents.
- **gzip**, **bzip2**, **xz** and **zip** are used to compress files.
- **tar** allows you to create or extract files from an archive file, often called a tarball. You can optionally compress while creating the archive, and decompress while extracting its contents
- **dd** can be used to make large exact copies even of entire disk partitions efficiently.

Chapter 9: User Environment

Learning Objectives

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

- Use and configure user accounts and user groups.
- Use and set environment variables.
- Use the previous shell command history.
- Use keyboard shortcuts.
- Use and define aliases.
- Use and set file permissions and ownership.

Section 1: Accounts

Identifying the Current User

As you know, Linux is a multiuser operating system; i.e., more than one user can log on at the same time.

- To list the currently logged-on users, type who
- To identify the current user, type whoami

Giving **who** the -a option will give more detailed information.

Basics of Users and Groups

```
File  Edit  View  Search  Terminal  Help
[test1@localhost ~]$ tail -10 /etc/group
pulse-access:x:495:
fuse:x:494:
stapusr:x:156:
stapsys:x:157:
stapdev:x:158:
sshd:x:74:
tcpdump:x:72:
slocate:x:21:
mgnanda:x:500:
test1:x:501:
[test1@localhost ~]$
[test1@localhost ~]$
[test1@localhost ~]$
[test1@localhost ~]$ tail -10 /etc/passwd
gdm:x:42:42::/var/lib/gdm:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
saslauth:x:498:76:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
pulse:x:497:496:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
mgnanda:x:500:500:Nandakumar:/home/mgnanda:/bin/bash
test1:x:501:501:Test User #1:/home/test1:/bin/bash
[test1@localhost ~]$
```

Linux uses **groups** for organizing users. Groups are collections of accounts with certain shared permissions. Control of group membership is administered through the /etc/group file, which shows a list of groups and their members. By default, every user belongs to a default or primary group. When a user logs in, the group membership is set for their primary group and all the members enjoy the same level of access and privilege. Permissions on various files and directories can be modified at the group level.

All Linux users are assigned a unique user ID (**uid**), which is just an integer, as well as one or more group ID's (**gid**), including a default one which is the same as the user ID.

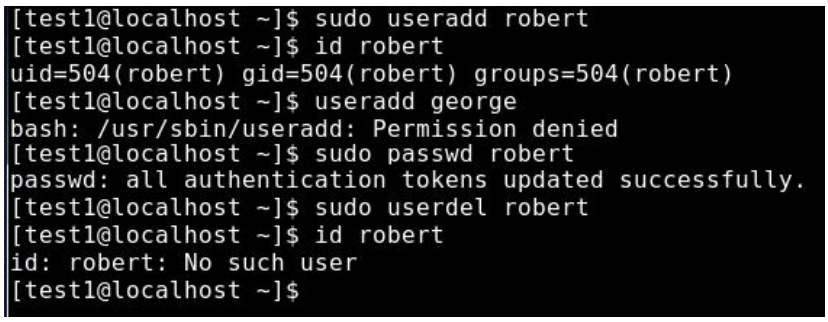
Historically **Fedora**-family systems start **uid**'s at 500; other distributions begin at 1000.

These numbers are associated with names through the files /etc/passwd and /etc/group.

For example, the first file might contain:
george:x:1002:1002:George Metesky:/home/george:/bin/bash
and the second george:x:1002

Groups are used to establish a set of users who have common interests for the purposes of access rights, privileges, and security considerations. Access rights to files (and devices) are granted on the basis of the user and the group they belong to.

Adding and Removing Users



Distributions have straightforward graphical interfaces for creating and removing users and groups and manipulating group membership. However, it is often useful to do it from the command line or from within shell scripts. Only the root user can add and remove users and groups.

Adding a new user is done with **useradd** and removing an existing user is done with **userdel**. In the simplest form an account for the new user turkey would be done with:

$ sudo useradd turkey

which by default sets the home directory to /home/turkey, populates it with some basic files (copied from /etc/skel) and adds a line to /etc/passwd such as:

turkey:x:502:502::/home/turkey:/bin/bash

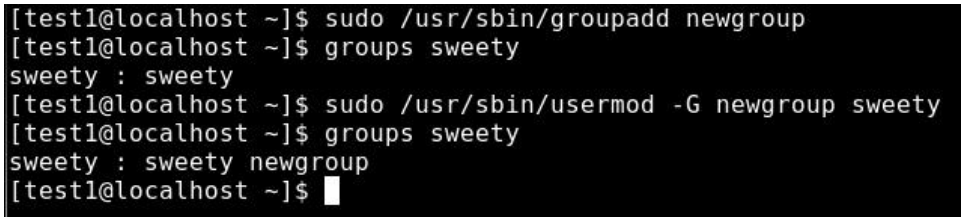
and sets the default shell to /bin/bash. Removing a user account is as easy as typing userdel turkey However, this will leave the /home/turkey directory intact. This might be useful if it is a temporary inactivation. To remove the home directory while removing the account one needs to use the **-r** option to **userdel**.

Typing **id** with no argument gives information about the current user, as in:

$ id
uid=500(george) gid=500(george) groups=106(fuse),500(george)

If given the name of another user as an argument, **id** will report information about that other user.

Adding and Removing Groups



Adding a new group is done with **groupadd**:

$ sudo /usr/sbin/groupadd anewgroup

The group can be removed with

$ sudo /usr/sbin/groupdel anewgroup

Adding a user to an already existing group is done with **usermod**. For example, you would first look at what groups the user already belongs to:

```
$ groups turkey
turkey : turkey
```

and then add the new group:

```
$ sudo /usr/sbin/usermod -G anewgroup turkey
$ groups turkey
turkey: turkey anewgroup
```

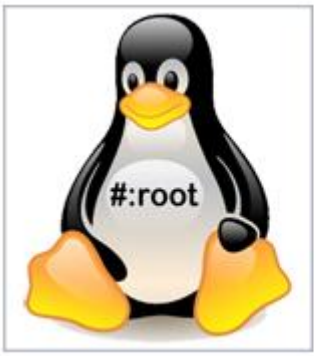
These utilities update /etc/group as necessary. **groupmod** can be used to change group properties such as the Group ID (gid) with the -g option or its name with the -n option.

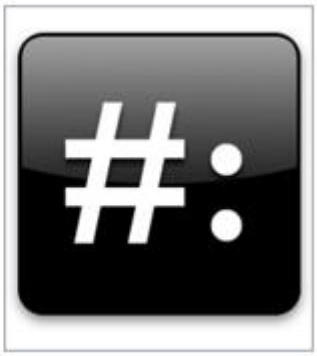
Removing a user from the group is a somewhat trickier. The **-G** option to **usermod** must give a complete list of groups. Thus if you do:

```
$ sudo /usr/sbin/usermod -G turkey turkey
$ groups turkey
turkey : turkey
```

only the **turkey** group will be left.

The root Account





The **root** account is very powerful and has full access to the system. Other operating systems often call this the **administrator** account; in Linux it is often called the **superuser** account. You must be extremely cautious before granting full root access to a user; it is rarely if ever justified. External attacks often consist of tricks used to elevate to the root account.

However, you can use the **sudo** feature to assign more limited privileges to user accounts:

- on only a temporary basis.
- only for a specific subset of commands.

su and sudo

When assigning elevated privileges, you can use the command **su** (switch or substitute user) to launch a new shell running as another user (you must type the password of the user you are becoming). Most often this other user is root, and the new shell allows the use of elevated privileges until it is exited. It is almost always a bad (dangerous for both security and stability) practice to use **su** to become root. Resulting errors can include deletion of vital files from the system and security breaches.

Granting privileges using **sudo** is less dangerous and is preferred. By default, **sudo** must be enabled on a per-user basis. However, some distributions (such as **Ubuntu**) enable it by default for at least one main user, or give this as an installation option.

In the chapter on Security that follows shortly, we will describe and compare **su** and **sudo** in detail.

Elevating to root Account

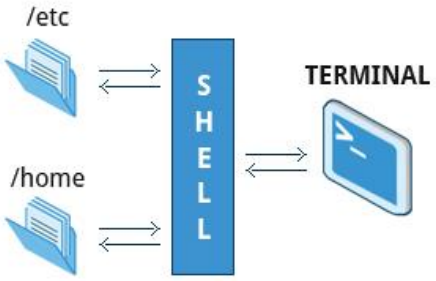
```
[test1@localhost ~]$ useradd gnk
bash: /usr/sbin/useradd: Permission denied
[test1@localhost ~]$ su useradd gnk
su: user useradd does not exist
[test1@localhost ~]$ sudo useradd gnk
[sudo] password for test1:
[test1@localhost ~]$ id gnk
uid=506(gnk) gid=508(gnk) groups=508(gnk)
[test1@localhost ~]$
```

To fully become root, one merely types **su** and then is prompted for the root password.

To execute just one command with root privilege type sudo <command>. When the command is complete you will return to being a normal unprivileged user.

sudo configuration files are stored in the /etc/sudoers file and in the /etc/sudoers.d/ directory. By default, the sudoers.d directory is empty.

Startup Files



In Linux, the command shell program (generally **bash)** uses one or more startup files to configure the environment. Files in the /etc directory define global settings for all users while Initialization files in the user's home directory can include and/or override the global settings.

The startup files can do anything the user would like to do in every command shell, such as:

- Customizing the user's prompt
- Defining command-line shortcuts and aliases
- Setting the default text editor
- Setting the **path** for where to find executable programs

Order of the Startup Files

```
[root@localhost test1]# cat ~/.bashrc
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```

```
[root@localhost test1]# cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH
[root@localhost test1]#
```

When you first login to Linux, /etc/profile is read and evaluated, after which the following files are searched (if they exist) in the listed order:

1. ~/.bash_profile
2. ~/.bash_login
3. ~/.profile

The Linux login shell evaluates whatever startup file that it comes across first and ignores the rest. This means that if it finds ~/.bash_profile, it ignores ~/.bash_login and ~/.profile. Different distributions may use different startup files.

However, every time you create a new shell, or terminal window, etc., you do not perform a full system login; only the ~/.bashrc file is read and evaluated. Although this file is not read and evaluated along with the login shell, most distributions and/or users include the ~/.bashrc file from within one of the three user-owned startup files. In the **Ubuntu**, **openSuse**, and **CentOS** distros, the user must make appropriate changes in the ~/.bash_profile file to include the ~/.bashrc file.

The .bash_profile will have certain extra lines, which in turn will collect the required customization parameters from .bashrc.

To practice, click the link provided below.

Identify the Currently Logged-In Users and Current User

```
ngeo@Dell-desk:~$ who

ngeo     :0          2014-12-18 13:31 (:0)

ngeo     pts/2       2014-12-18 13:46 (:0)

ngeo@Dell-desk:~$ whoami

ngeo

ngeo@Dell-desk:~$
```

Which command allows users to temporarily elevate to the root account?

- ( ) whoami
- ( ) who
- (•) sudo
- ( ) /etc/profile

What are the advantages of startup files?

- [ ] They help the users change to the root account.
- [x] They can set command-line shortcuts and aliases.
- [ ] They can provide full access to the system.
- [x] They can set the default text editor.

Section 2: Environment Variables

Environment Variables

Environment variables are simply named quantities that have specific values and are understood by the command shell, such as **bash**. Some of these are pre-set (built-in) by the system, and others are set by the user either at the command line or within startup and other scripts. An environment variable is actually no more than a character string that contains information used by one or more applications.

There are a number of ways to view the values of currently set environment variables; one can type **set**, **env**, or **export.** Depending on the state of your system, **set** may print out many more lines than the other two methods.

```
$ set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extglob:extquote:force_fignore
BASH_ALIASES=()
...

$ env
SSH_AGENT_PID=1892
GPG_AGENT_INFO=/run/user/me/keyring-Ilf3vt/gpg:0:1
TERM=xterm
SHELL=/bin/bash
...

$ export
declare -x COLORTERM=gnome-terminal
declare -x COMPIZ_BIN_PATH=/usr/bin /
declare -x COMPIZ_CONFIG_PROFILE=ubuntu
...
```

```
[test1@localhost ~]$ set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extquote:force_fignore:hostcomplete:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="1" [2]="2" [3]="1" [4]="release" [5]="x86_64-redhat-linux-gnu")
BASH_VERSION='4.1.2(1)-release'
COLORS=/etc/DIR_COLORS
COLORTERM=gnome-terminal
COLUMNS=157
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-8qTHXWpv8v,guid=abf8043f18a1ff21e8003da500000069
DESKTOP_SESSION=gnome
DIRSTACK=()
DISPLAY=:0.0
EUID=501
GDMSESSION=gnome
GDM_KEYBOARD_LAYOUT=us
GDM_LANG=en_US.UTF-8
GIO_LAUNCHED_DESKTOP_FILE=/home/test1/Desktop/gnome-terminal.desktop
GIO_LAUNCHED_DESKTOP_FILE_PID=2539
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_KEYRING_PID=2271
GNOME_KEYRING_SOCKET=/tmp/keyring-ePEhFj/socket
GROUPS=()
GTK_RC_FILES=/etc/gtk/gtkrc:/home/test1/.gtkrc-1.2-gnome2
G_BROKEN_FILENAMES=1
HISTCONTROL=ignoredups
HISTFILE=/home/test1/.bash_history
```

```
[test1@localhost ~]$ export
declare -x COLORTERM="gnome-terminal"
declare -x DBUS_SESSION_BUS_ADDRESS="unix:abstract=/tmp/dbus-8qTHXWpv8v,guid=abf
8043f18a1ff21e8003da500000069"
declare -x DESKTOP_SESSION="gnome"
declare -x DISPLAY=":0.0"
declare -x GDMSESSION="gnome"
declare -x GDM_KEYBOARD_LAYOUT="us"
declare -x GDM_LANG="en_US.UTF-8"
declare -x GIO_LAUNCHED_DESKTOP_FILE="/home/test1/Desktop/gnome-terminal.desktop
"
declare -x GIO_LAUNCHED_DESKTOP_FILE_PID="2615"
declare -x GNOME_DESKTOP_SESSION_ID="this-is-deprecated"
declare -x GNOME_KEYRING_PID="2271"
declare -x GNOME_KEYRING_SOCKET="/tmp/keyring-ePEhFj/socket"
declare -x GTK_RC_FILES="/etc/gtk/gtkrc:/home/test1/.gtkrc-1.2-gnome2"
declare -x G_BROKEN_FILENAMES="1"
declare -x HISTCONTROL="ignoredups"
                .
                .
                .
                .
                .
                .
declare -x USERNAME="test1"
declare -x WINDOWID="65011715"
declare -x WINDOWPATH="1"
declare -x XAUTHORITY="/var/run/gdm/auth-for-test1-IMUT4K/database"
declare -x XDG_SESSION_COOKIE="8a22b47d44cf2807b6bd9e8c00000029-1406629599.94246
4-1895940425"

[test1@localhost ~]$ env
ORBIT_SOCKETDIR=/tmp/orbit-test1
HOSTNAME=localhost.localdomain
GIO_LAUNCHED_DESKTOP_FILE_PID=2539
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=8a22b47d44cf2807b6bd9e8c00000029-1406629599.942464-1895940425
HISTSIZE=1000
GTK_RC_FILES=/etc/gtk/gtkrc:/home/test1/.gtkrc-1.2-gnome2
WINDOWID=65011715
USER=test1
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=01;05;37;41:s
4;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;3
01;31:*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:*.tbz=01;31:*.tbz2=01;31:*.bz=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01
:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;
*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01
5:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;
      .
      .
      .
      .
      .
      .
WINDOWPATH=1
DISPLAY=:0.0
G_BROKEN_FILENAMES=1                                    in:/sbin:/home/test1/bin
XAUTHORITY=/var/run/gdm/auth-for-test1-IMUT4K/database
COLORTERM=gnome-terminal
 =/usr/bin/env
GNOME_KEYRING_PID=2271
```

Setting Environment Variables

By default, variables created within a script are only available to the current shell; child processes (sub-shells) will not have access to values that have been set or modified. Allowing child processes to see the values, requires use of the **export** command.

Task	Command
Show the value of a specific variable	echo $SHELL
Export a new variable value	export VARIABLE=value (or VARIABLE=value; export VARIABLE)
Add a variable permanently	1. Edit ~/.bashrc and add the line export VARIABLE=value 2. Type source ~/.bashrc or just . ~/.bashrc (dot ~/.bashrc); or just start a new shell by typing bash

The HOME Variable

HOME is an environment variable that represents the home (or login) directory of the user. **cd** without arguments will change the current working directory to the value of HOME. Note the tilde character (~) is often used as an abbreviation for $HOME. Thus cd $HOME and cd ~ are completely equivalent statements.

```
test1@ubuntu:~$ echo $HOME
/home/test1
test1@ubuntu:~$ cd /bin
test1@ubuntu:/bin$ pwd
/bin
test1@ubuntu:/bin$ cd
test1@ubuntu:~$ pwd
/home/test1
test1@ubuntu:~$
```

Command	Explanation
$ echo $HOME /home/me $ cd /bin	Show the value of the HOME environment variable then change directory (cd) to /bin
$ pwd /bin	Where are we? Use print (or present) working directory (pwd) to find out. As expected /bin
$ cd	Change directory without an argument . . .
$ pwd /home/me	. . . takes us back to HOME as you can now see

The PATH Variable

```
test2@OpenSUSE:~> echo $PATH
/home/test2/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games
test2@OpenSUSE:~>
```

PATH is an ordered list of directories (the **path**) which is scanned when a command is given to find the appropriate program or script to run. Each directory in the path is separated by colons (:). A null (empty) directory name (or ./) indicates the current directory at any given time.

- :path1:path2
- path1::path2

In the example :path1:path2, there is null directory before the first colon (:). Similarly, for path1::path2 there is null directory between path1 and path2.

To prefix a private bin directory to your path:

$ export PATH=$HOME/bin:$PATH
$ echo $PATH
/home/me/bin:/usr/local/bin:/usr/bin:/bin/usr

The PS1 Variable

Prompt Statement (PS) is used to customize your **prompt** string in your terminal windows to display the information you want.

```
[test1@localhost: ~]$export PS1='\u@\h\d: \w$'
test1@localhostThu Jul 24: ~$export PS1='[\u@\h: \w]$'
[test1@localhost: ~]$export PS1=$
$export PS1='[\u@\h: \w]$'
[test1@localhost: ~]$
```

PS1 is the primary prompt variable which controls what your command line prompt looks like. The following special characters can be included in PS1 :

\u - User name
\h - Host name
\w - Current working directory

\! - History number of this command
\d - Date

They must be surrounded in single quotes when they are used as in the following example:
$ echo $PS1
$
$ export PS1='\u@\h:\w$ '
me@example.com:~$ # new prompt
me@example.com:~$

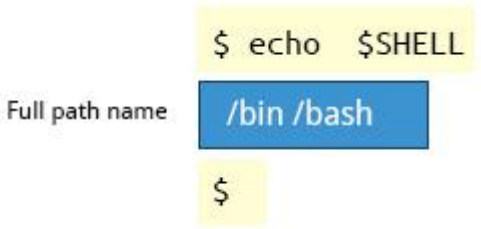
To revert the changes:
me@example.com:~$ export PS1='$ '
$

Even better practice would be to save the old prompt first and then restore, as in:
$ OLD_PS1=$PS1

change the prompt, and eventually change it back with:
$ PS1=$OLD_PS1
$

The SHELL Variable

The environment variable SHELL points to the user's default command shell (the program that is handling whatever you type in a command window, usually **bash**) and contains the full pathname to the shell:



$ echo $SHELL
/bin/bash
$

Which of the following is a commonly used environment variable?

- echo
- bin
- cd
- HOME

Which of the following environment variables is used to specify the prompt string?

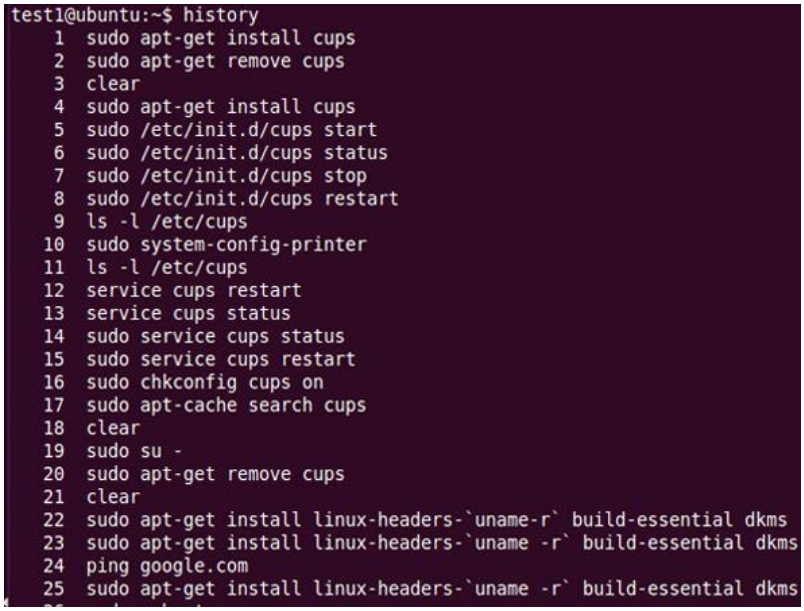
- EXPORT
- PS1
- ENV
- SHELL

Which of the following will set your **bash** prompt to include the current working directory?

- export PS1='\w$ '
- export PS1='\u@\h:\d$ '
- export PS1='$ '
- echo $PS1

Section 3: Recalling Previous Commands

Recalling Previous Commands



bash keeps track of previously entered commands and statements in a **history** buffer; you can recall previously used commands simply by using the **Up** and **Down** cursor keys. To view the list of previously executed commands, you can just type history at the command line.

The list of commands is displayed with the most recent command appearing last in the list. This information is stored in ~/.bash_history.

Using History Environment Variables

```
[test3@CentOS ~]$ echo "$HISTFILESIZE"
1000
[test3@CentOS ~]$ echo "$HISTSIZE"
1000
[test3@CentOS ~]$
```

Several associated environment variables can be used to get information about the history file.

HISTFILE stores the location of the history file.

HISTFILESIZE stores the maximum number of lines in the history file.

HISTSIZE stores the maximum number of lines in the history file for the current session.

Finding and Using Previous Commands

Specific keys to perform various tasks:

Key	Usage
Up/Down arrow key	Browse through the list of commands previously executed
!! (Pronounced as **bang-bang**)	Execute the previous command
CTRL-R	Search previously used commands

If you want to recall a command in the history list, but do not want to press the arrow key repeatedly, you can press **CTRL-R** to do a reverse intelligent search.

As you start typing the search goes back in reverse order to the first command that matches the letters you've typed. By typing more successive letters you make the match more and more specific.

The following is an example of how you can use the CTRL-R command to search through the command history:

```
$ ^R                                    # This all happens on 1 line
(reverse-i-search)'s': sleep 1000   # Searched for 's'; matched "sleep"
$ sleep 1000                            # Pressed Enter to execute the searched command
$
```

Executing Previous Commands

The table describes the syntax used to execute previously used commands.

Syntax	Task
!	Start a history substitution
!$	Refer to the last argument in a line
!n	Refer to the nth command line
!string	Refer to the most recent command starting with string

All history substitutions start with !. In the line $ ls -l /bin /etc /var !$ refers to /var, which is the last argument in the line.

Here are more examples:

```
$ 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 the command beginning with "sl"
sleep 1000
$
```

Keyboard Shortcuts

Keyboard Shortcut	Task
CTRL-L	Clears the screen
CTRL-D	Exits the current shell
CTRL-Z	Puts the current process into suspended background
CTRL-C	Kills the current process
CTRL-H	Works the same as backspace
CTRL-A	Goes to the beginning of the line
CTRL-W	Deletes the word before the cursor
CTRL-U	Deletes from beginning of line to cursor position
CTRL-E	Goes to the end of the line
Tab	Auto-completes files, directories, and binaries

To practice, click the link provided below.

Using History Command Variables

How would you find the name of a previously used command and easily re-run that command without having to type it at the prompt?

- Press the **Down** arrow key until you locate the command, and then press **Enter** to rerun the command.
- Press the **Up** arrow key until you locate the command, and then press **Enter** to rerun the command.
- Use the **history** command to locate the command, and then press **Enter** to rerun the command.
- Press the **Right** arrow key until you locate the command, and then press **Enter** to rerun the command.

How would you run command 70 from your previous history in one step?
!70

Where does the shell store the command history?

- bash
- ~/.bash_history
- /bin/bash
- /bin /etc /var

Section 4: Command Aliases

Creating Aliases

You can create customized commands or modify the behavior of already existing ones by creating **aliases**. Most often these aliases are placed in your ~/.bashrc file so they are available to any command shells you create.

```
[test3@CentOS ~]$ alias
alias l.='ls -d .* --color=auto'
alias l1='ls -l'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[test3@CentOS ~]$ alias l2='ls -l /etc'
[test3@CentOS ~]$ alias
alias l.='ls -d .* --color=auto'
alias l1='ls -l'
alias l2='ls -l /etc'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[test3@CentOS ~]$ l2
total 2176
-rw-r--r--.  1 root root     20 Jun 23 12:45 1.pdf
-rw-r--r--.  1 root root     42 Jun 23 12:45 2.pdf
drwxr-xr-x.  3 root root   4096 May  2 15:49 abrt
drwxr-xr-x.  4 root root   4096 May  2 15:55 acpi
-rw-r--r--.  1 root root     44 Jul 29 21:19 adjtime
-rw-r--r--.  1 root root   1512 Jan 12  2010 aliases
-rw-r--r--.  1 root root  12288 May  2 16:13 aliases.db
drwxr-xr-x.  2 root root   4096 May  2 15:53 alsa
drwxr-xr-x.  2 root root   4096 Jul 28 19:39 alternatives
-rw-------.  1 root root    541 Nov 23  2013 anacrontab
-rw-r--r--.  1 root root    245 Nov 11  2010 anthy-conf
drwxr-xr-x.  3 root root   4096 May  8 10:18 apt
```

Typing **alias** with no arguments will list currently defined aliases.

Please note there should not be any spaces on either side of the equal sign and the alias definition needs to be placed within either single or double quotes if it contains any spaces.

Which command is used to create an alias, projx, for cd /home/staff/RandD/src?

alias projx='cd /home/staff/RandD/src'

Explanation

The alias projx='cd /home/staff/RandD/src' command is used to create an alias projx for the cd /home/staff/RandD/src command. You can use either double or single quotation marks in this case.

Section 5: File Permissions

File Ownership

In Linux and other UNIX-based operating systems, every file is associated with a user who is the **owner**. Every file is also associated with a **group** (a subset of all users) which has an interest in the file and certain rights, or permissions: read, write, and execute.

The following utility programs involve user and group ownership and permission setting.

Command	Usage
chown	Used to change user ownership of a file or directory
chgrp	Used to change group ownership
chmod	Used to change the permissions on the file which can be done separately for **owner**, **group** and the rest of the world (often named as **other**.)

File Permission Modes and chmod

Files have three kinds of permissions: read (r), write (w), execute (x). These are generally represented as in rwx. These permissions affect three groups of owners: user/owner (u), group (g), and others (o).

As a result, you have the following three groups of three permissions:

rwx: rwx: rwx
u: g: o

```
[test3@CentOS ~]$ ls -l test1
-rw-rw-r--. 1 test3 test3 98 Jul 23 22:11 test1
[test3@CentOS ~]$ chmod uo+x,g-w test1
[test3@CentOS ~]$ ls -l test1
-rwxr--r-x. 1 test3 test3 98 Jul 23 22:11 test1
[test3@CentOS ~]$ chmod 755 test1
[test3@CentOS ~]$ ls -l test1
-rwxr-xr-x. 1 test3 test3 98 Jul 23 22:11 test1
[test3@CentOS ~]$
```

There are a number of different ways to use **chmod**. For instance, to give the owner and others execute permission and remove the group write permission:

```
$ ls -l a_file
-rw-rw-r-- 1 coop coop 1601 Mar 9 15:04 a_file
$ chmod uo+x,g-w a_file
$ ls -l a_file
-rwxr--r-x 1 coop coop 1601 Mar 9 15:04 a_file
```

where u stands for user (owner), o stands for other (world), and g stands for group.

This kind of syntax can be difficult to type and remember, so one often uses a shorthand which lets you set all the permissions in one step. This is done with a simple algorithm, and a single digit suffices to specify all three permission bits for each entity. This digit is the sum of:

- 4 if read permission is desired.
- 2 if write permission is desired.
- 1 if execute permission is desired.

Thus 7 means read/write/execute, 6 means read/write, and 5 means read/execute.

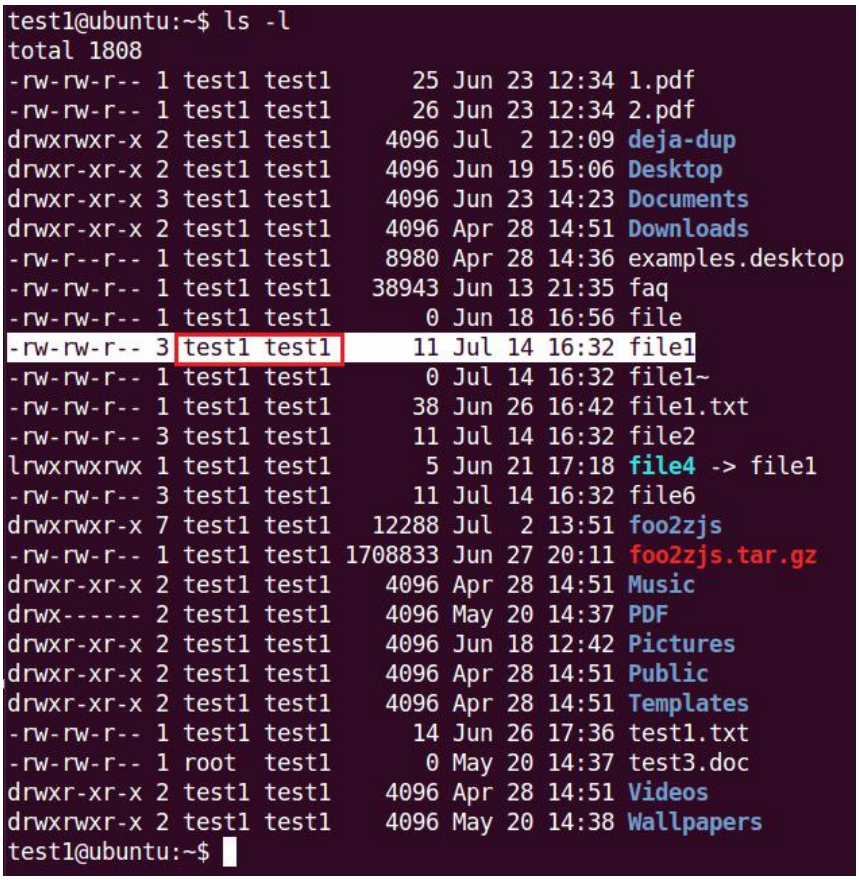
When you apply this to the **chmod** command you have to give three digits for each degree of freedom, such as in

```
$ chmod 755 a_file
$ ls -l a_file
-rwxr-xr-x 1 coop coop 1601 Mar 9 15:04 a_file
```

Example of chown

Let's see an example of changing file ownership using **chown:**

The first image shows the permissions for owners/groups/all users on 'file1'. The second image shows the change in permissions for the different users on "file1"




```
$ ls -l
total 4
-rw-rw-r--. 1 bob bob 0 Mar 16 19:04 file-1
-rw-rw-r--. 1 bob bob 0 Mar 16 19:04 file-2
drwxrwxr-x. 2 bob bob 4096 Mar 16 19:04 temp

$ sudo chown root file-1
[sudo] password for bob:

$ ls -l
total 4
-rw-rw-r--. 1 root bob 0 Mar 16 19:04 file-1
```

-rw-rw-r--. 1 bob bob 0 Mar 16 19:04 file-2
drwxrwxr-x. 2 bob bob 4096 Mar 16 19:04 temp

```
test1@ubuntu:~$ sudo chown root file1
test1@ubuntu:~$ ls -l
total 1808
-rw-rw-r-- 1 test1 test1      25 Jun 23 12:34 1.pdf
-rw-rw-r-- 1 test1 test1      26 Jun 23 12:34 2.pdf
drwxrwxr-x 2 test1 test1    4096 Jul  2 12:09 deja-dup
drwxr-xr-x 2 test1 test1    4096 Jun 19 15:06 Desktop
drwxr-xr-x 3 test1 test1    4096 Jun 23 14:23 Documents
drwxr-xr-x 2 test1 test1    4096 Apr 28 14:51 Downloads
-rw-r--r-- 1 test1 test1    8980 Apr 28 14:36 examples.desktop
-rw-rw-r-- 1 test1 test1   38943 Jun 13 21:35 faq
-rw-rw-r-- 1 test1 test1       0 Jun 18 16:56 file
-rw-rw-r-- 3 root  test1      11 Jul 14 16:32 file1
-rw-rw-r-- 1 test1 test1       0 Jul 14 16:32 file1~
-rw-rw-r-- 1 test1 test1      38 Jun 26 16:42 file1.txt
-rw-rw-r-- 3 root  test1      11 Jul 14 16:32 file2
lrwxrwxrwx 1 test1 test1       5 Jun 21 17:18 file4 -> file1
-rw-rw-r-- 3 root  test1      11 Jul 14 16:32 file6
drwxrwxr-x 7 test1 test1   12288 Jul  2 13:51 foo2zjs
-rw-rw-r-- 1 test1 test1 1708833 Jun 27 20:11 foo2zjs.tar.gz
drwxr-xr-x 2 test1 test1    4096 Apr 28 14:51 Music
drwx------ 2 test1 test1    4096 May 20 14:37 PDF
drwxr-xr-x 2 test1 test1    4096 Jun 18 12:42 Pictures
drwxr-xr-x 2 test1 test1    4096 Apr 28 14:51 Public
drwxr-xr-x 2 test1 test1    4096 Apr 28 14:51 Templates
-rw-rw-r-- 1 test1 test1      14 Jun 26 17:36 test1.txt
-rw-rw-r-- 1 root  test1       0 May 20 14:37 test3.doc
drwxr-xr-x 2 test1 test1    4096 Apr 28 14:51 Videos
drwxrwxr-x 2 test1 test1    4096 May 20 14:38 Wallpapers
test1@ubuntu:~$
```

Example of chgrp

Now let's see an example of changing group ownership using **chgrp**:

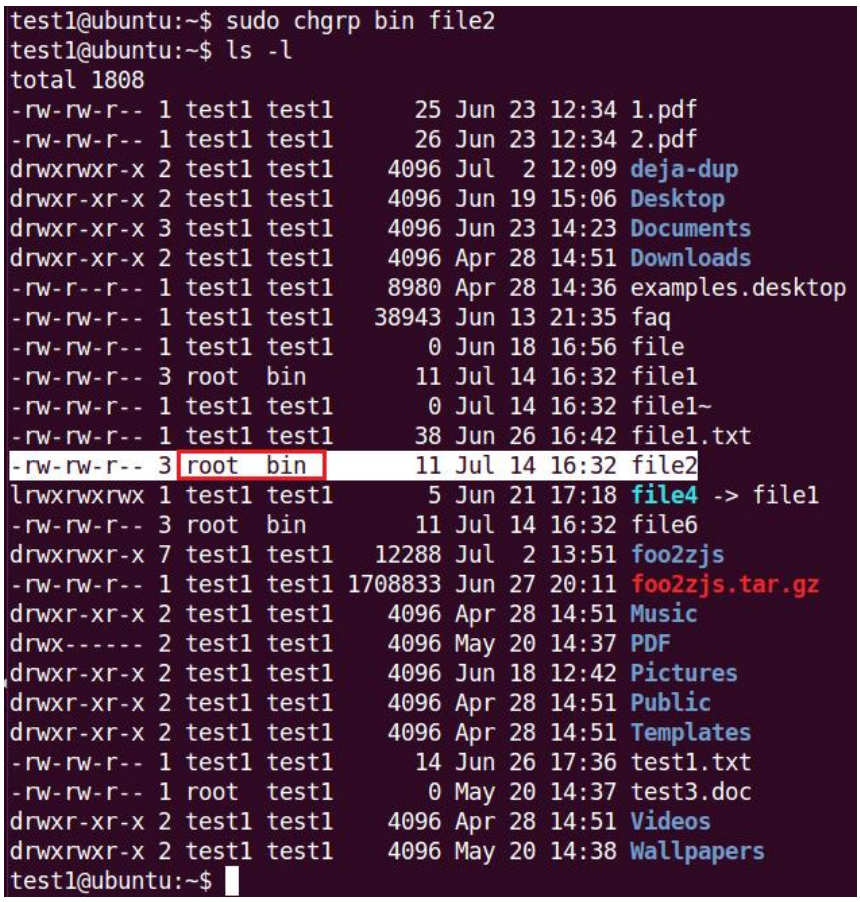
The image on LHS shows the group with their permissions on 'file1'.

The image on RHS shows the change in groups and thier permissions on "file1"

$ sudo chgrp bin file-2
$ ls -l
total 4
-rw-rw-r--. 1 root bob 0 Mar 16 19:04 file-1

```
test1@ubuntu:~$ ls -l
total 1808
-rw-rw-r-- 1 test1 test1      25 Jun 23 12:34 1.pdf
-rw-rw-r-- 1 test1 test1      26 Jun 23 12:34 2.pdf
drwxrwxr-x 2 test1 test1    4096 Jul  2 12:09 deja-dup
drwxr-xr-x 2 test1 test1    4096 Jun 19 15:06 Desktop
drwxr-xr-x 3 test1 test1    4096 Jun 23 14:23 Documents
drwxr-xr-x 2 test1 test1    4096 Apr 28 14:51 Downloads
-rw-r--r-- 1 test1 test1    8980 Apr 28 14:36 examples.desktop
-rw-rw-r-- 1 test1 test1   38943 Jun 13 21:35 faq
-rw-rw-r-- 1 test1 test1       0 Jun 18 16:56 file
-rw-rw-r-- 3 root  test1      11 Jul 14 16:32 file1
-rw-rw-r-- 1 test1 test1       0 Jul 14 16:32 file1~
-rw-rw-r-- 1 test1 test1      38 Jun 26 16:42 file1.txt
-rw-rw-r-- 3 root  test1      11 Jul 14 16:32 file2
lrwxrwxrwx 1 test1 test1       5 Jun 21 17:18 file4 -> file1
-rw-rw-r-- 3 root  test1      11 Jul 14 16:32 file6
drwxrwxr-x 7 test1 test1   12288 Jul  2 13:51 foo2zjs
-rw-rw-r-- 1 test1 test1 1708833 Jun 27 20:11 foo2zjs.tar.gz
```

-rw-rw-r--. 1 bob bin 0 Mar 16 19:04 file-2
drwxrwxr-x. 2 bob bob 4096 Mar 16 19:04 temp



To practice, click the link provided below.

Using File Permission Commands

Which of the following permissions provides read and execute access?

- rwx
- r—
- rw-
- r-x

Labs

Click the link to download a PDF for the Lab activity.

User Environment_Labs

User Environment_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered.

- Linux is a multiuser system.
- To find the currently logged on users, you can use the **who** command.
- To find the current user ID, you can use the **whoami** command.
- The root account has full access to the system. It is never sensible to grant full root access to a user.
- You can assign root privileges to regular user accounts on a temporary basis using the **sudo** command.
- The shell program (**bash**) uses multiple startup files to create the user environment. Each file affects the interactive environment in a different way. /etc/profile provides the global settings.
- Advantages of startup files include that they customize the user's prompt, set the user's terminal type, set the command-line shortcuts and aliases, and set the default text editor, etc.

- An **environment variable** is a character string that contains data used by one or more applications. The built-in shell variables can be customized to suit your requirements.
- The **history** command recalls a list of previous commands which can be edited and recycled.
- In Linux, various keyboard shortcuts can be used at the command prompt instead of long actual commands.
- You can customize commands by creating aliases. Adding an alias to ~/.bashrc will make it available for other shells.
- File permissions can be changed by typing chmod permissions filename.
- File ownership is changed by typing chown owner filename.
- File group ownership is changed by typing chgrp group filename.

Chapter 10: Text Editors

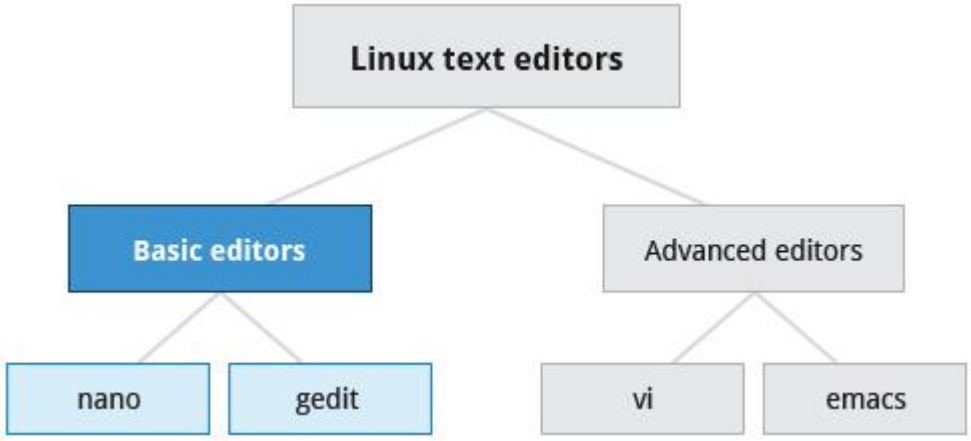
Learning Objectives

By the end of this chapter, you should be familiar with:

- How to create and edit files using the available Linux text editors.
- **nano**, a simple text-based editor.
- **gedit**, a simple graphical editor
- **vi** and **emacs,** two advanced editors with both text-based and graphical interfaces.

Section 1: Basic Editors – nano & gedit

Overview of Text Editors in Linux



At some point you will need to manually edit **text files**. You might be composing an email off-line, writing a script to be used for **bash** or other command interpreters, altering a system or application configuration file, or developing source code for a programming language such as **C** or **Java**.

Linux Administrators quite often sidestep the text editors, by using graphical utilities for creating and modifying system configuration files. However, this can be far more laborious than directly using a text editor. Note that word processing applications such as **Notepad** or the applications that are part of office suites are not really basic text editors because they add a lot of extra (usually invisible) formatting information that will probably render system administration configuration files unusable for their intended purpose. So using text editors really is essential in Linux.

By now you have certainly realized Linux is packed with choices; when it comes to text editors, there are many choices ranging from quite simple to very complex, including:

- **nano**
- **gedit**
- **vi**
- **emacs**

In this section, we will learn about **nano** and **gedit**; editors which are relatively simple and easy to learn. Before we start, let's take a look at some cases where an editor is not needed.

Creating Files Without Using an Editor

Sometimes you may want to create a short file and don't want to bother invoking a full text editor. In addition, doing so can be quite useful when used from within scripts, even when creating longer files. You'll no doubt find yourself using this method when you start on the later chapters that cover **bash** scripting!

If you want to create a file without using an editor there are two standard ways to create one from the command line and fill it with content.

The first is to use **echo** repeatedly:

```
$ echo line one > myfile
$ echo line two >> myfile
$ echo line three >> myfile
```

Earlier we learned that a single greater-than sign (>) will send the output of a command to a file. Two greater-than signs (>>) will **append** new output to an existing file.

The second way is to use **cat** combined with redirection:

```
$ cat << EOF > myfile
> line one
> line two
> line three
> EOF
$
```

Both the above techniques produce a file with the following lines in it:

```
line one
line two
line three
```

and are extremely useful when employed by scripts.

```
test2@0penSUSE:~> cat << EOF > numberfile
> line one
> line two
> line three
> EOF
test2@0penSUSE:~> cat numberfile
line one
line two
line three
test2@0penSUSE:~>
```

```
test2@0penSUSE:~> echo line one > myfile
test2@0penSUSE:~> echo line two >> myfile
test2@0penSUSE:~> echo line three >> myfile
test2@0penSUSE:~> cat myfile
line one
line two
line three
test2@0penSUSE:~>
```

nano and gedit

There are some text editors that are pretty obvious; they require no particular experience to learn and are actually quite capable if not robust. One particularly easy one to use is the text-terminal based editor **nano**. Just invoke **nano** by giving a file name as an argument. All the help you need is displayed at the bottom of the screen, and you should be able to proceed without any problem.

As a graphical editor, **gedit** is part of the **GNOME** desktop system (**kwrite** is associated with **KDE)**. The **gedit** and **kwrite** editors are very easy to use and are extremely capable. They are also very configurable. They look a lot like **Notepad** in **Windows**. Other variants such as **kedit** and **kate** are also supported by **KDE**.

nano

nano is easy to use, and requires very little effort to learn. To open a file in **nano**, type nano <filename> and press **Enter**. If the file doesn't exist, it will be created.

nano provides a two line “shortcut bar” at the bottom of the screen that lists the available commands. Some of these commands are:

- CTRL-G: Display the help screen
- CTRL-O: Write to a file
- CTRL-X: Exit a file

- CTRL-R: Insert contents from another file to the current buffer
- CTRL-C: Cancels previous commands

gedit

gedit (pronounced 'g-edit') is a simple-to-use graphical editor that can only be run within a Graphical Desktop environment. It is visually quite similar to the **Notepad** text editor in **Windows**, but is actually far more capable and very configurable and has a wealth of plugins available to extend its capabilities further.

To open a new file in **gedit**, find the program in your desktop's menu system, or from the command line type gedit <filename>. If the file doesn't exist it will be created.

Using **gedit** is pretty straight-forward and doesn't require much training. Its interface is composed of quite familiar elements.

Labs

Click the link to download a PDF for the Lab activity.

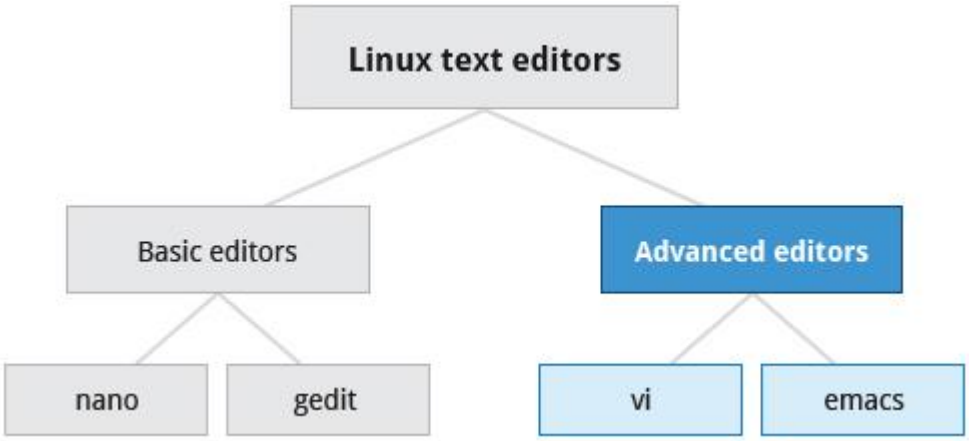
Basic Editors: nano and gedit_Labs

Basic Editors: nano and gedit_Solutions

Section 2: More advanced editors – vi & emacs

vi and emacs

Developers and administrators experienced in working on UNIX-like systems almost always use one of the two venerable editing options; **vi** and **emacs**. Both are present or easily available on all distributions and are completely compatible with the versions available on other operating systems.



Both **vi** and **emacs** have a basic purely text-based form that can run in a non-graphical environment. They also have one or more **X**-based graphical forms with extended capabilities; these may be friendlier for a less experienced user. While **vi** and **emacs** can have significantly steep learning curves for new users, they are extremely efficient when one has learned how to use them.

You need to be aware that fights among seasoned users over which editor is better can be quite intense and are often described as a holy war.

Introduction to vi

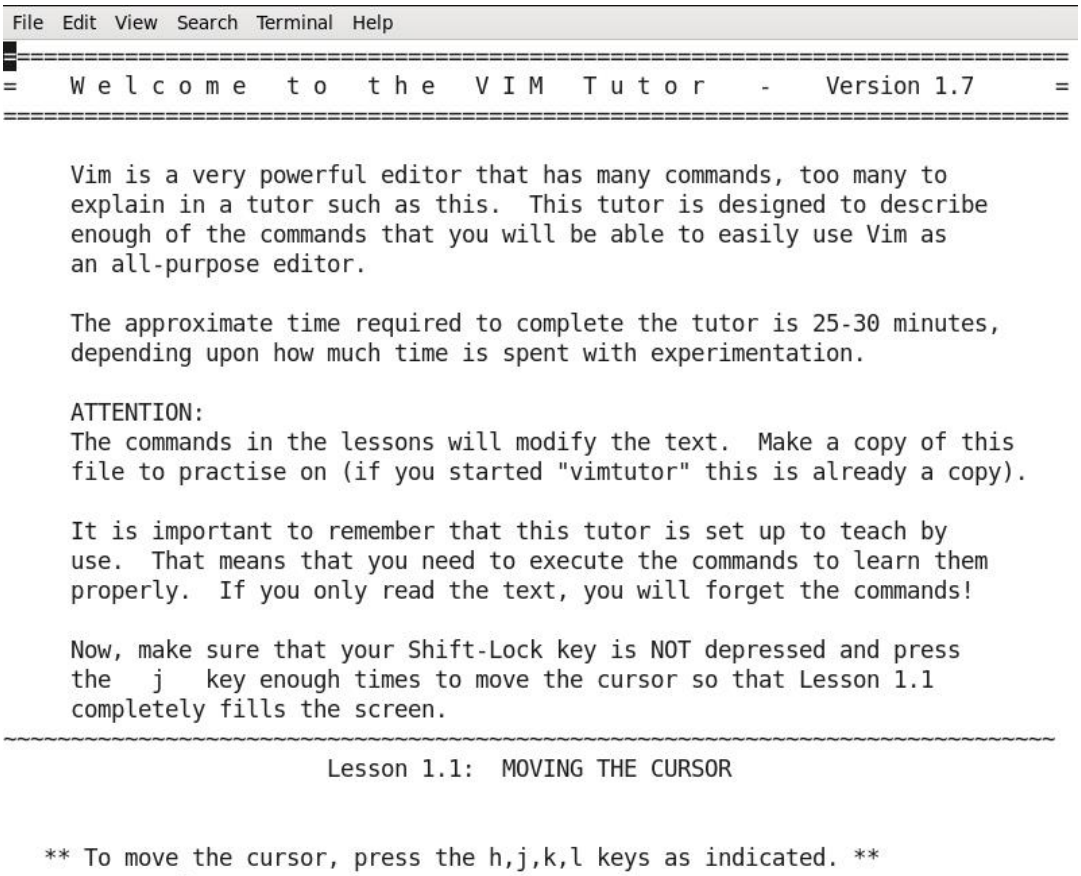
Usually the actual program installed on your system is **vim** which stands for **vi Improved**, and is aliased to the name **vi**. The name is pronounced as “vee-eye”.

Even if you don’t want to use **vi,** it is good to gain some familiarity with it: it is a standard tool installed on virtually all Linux distributions. Indeed, there may be times where there is no other editor available on the system.

GNOME extends **vi** with a very graphical interface known as **gvim** and **KDE** offers **kvim**. Either of these may be easier to use at first.

When using **vi**, all commands are entered through the keyboard; you don’t need to keep moving your hands to use a pointer device such as a mouse or touchpad, unless you want to do so when using one of the graphical versions of the editor.

vimtutor



Typing vimtutor launches a short but very comprehensive tutorial for those who want to learn their first **vi** commands. This tutorial is a good place to start learning **vi**. Even though it provides only an introduction and just seven lessons, it has enough material to make you a very proficient **vi** user because it covers a large number of commands. After learning these basic ones, you can look up new tricks to incorporate into your list of **vi** commands because there are always more optimal ways to do things in **vi** with less typing.

Modes in vi

vi provides three **modes** as described in the table below. It is vital to not lose track of which mode you are in. Many keystrokes and commands behave quite differently in different modes.

Mode	Feature
Command	• By default, **vi** starts in Command mode. • Each key is an editor command. • Keyboard strokes are interpreted as commands that can modify file contents.
Insert	• Type i to switch to Insert mode from Command mode. • Insert mode is used to enter (insert) text into a file. • Insert mode is indicated by an "**? INSERT ?**" indicator at the bottom of the screen. • Press Esc to exit Insert mode and return to Command mode.
Line	• Type: to switch to the Line mode from Command mode. Each key is an external command, including operations such as writing the file contents to disk or exiting. • Uses line editing commands inherited from older line editors. Most of these commands are actually no longer used. Some line editing commands are very powerful. • Press Esc to exit Line mode and return to Command mode.

Working with Files in vi

The table describes the most important commands used to start, exit, read, and write files in **vi**. The **ENTER** key needs to be pressed after all of these commands.

Command	Usage
vi myfile	Start the **vi** editor and edit the **myfile** file
vi -r myfile	Start **vi** and edit **myfile** in recovery mode from a system crash
:r file2	Read in **file2** and insert at current position
:w	Write to the file
:w myfile	Write out the file to **myfile**
:w! file2	Overwrite **file2**
:x or :wq	Exit **vi** and write out modified file
:q	Quit **vi**
:q!	Quit **vi** even though modifications have not been saved

Changing Cursor Positions in vi

The table describes the most important keystrokes used when changing cursor position in **vi.** Line mode commands (those following colon (:)) require the **ENTER** key to be pressed after the command is typed.

Key	Usage
arrow keys	To move up, down, left and right
j or <ret>	To move one line down
k	To move one line up
h or Backspace	To move one character left
l or Space	To move one character right
0	To move to beginning of line
$	To move to end of line
w	To move to beginning of next word
:0 or 1G	To move to beginning of file
:n or nG	To move to line n
:$ or G	To move to last line in file
CTRL-F or Page Down	To move forward one page
CTRL-B or Page Up	To move backward one page
^l	To refresh and center screen

Using Modes and Cursor Movements in vi

Searching for Text in vi

The table describes the most important commands used when searching for text in **vi**. The **ENTER** key should be pressed after typing the search pattern.

Command	Usage
/pattern	Search forward for pattern
?pattern	Search backward for pattern

The table describes the most important keystrokes used when searching for text in **vi**.

Key	Usage
n	Move to next occurrence of search pattern
N	Move to previous occurrence of search pattern

Working with Text in vi

The table describes the most important keystrokes used when changing, adding, and deleting text in **vi.**

Click the link to download a consolidated PDF file with commands for **vi.**

commands for vi

Key	Usage
a	Append text after cursor; stop upon Escape key
A	Append text at end of current line; stop upon Escape key
i	Insert text before cursor; stop upon Escape key
I	Insert text at beginning of current line; stop upon Escape key
o	Start a new line below current line, insert text there; stop upon Escape key
O	Start a new line above current line, insert text there; stop upon Escape key
r	Replace character at current position
R	Replace text starting with current position; stop upon Escape key
x	Delete character at current position
Nx	Delete N characters, starting at current position
dw	Delete the word at the current position
D	Delete the rest of the current line
dd	Delete the current line
Ndd or dNd	Delete N lines
u	Undo the previous operation
yy	Yank (copy) the current line and put it in buffer
Nyy or yNy	Yank (copy) N lines and put it in buffer
p	Paste at the current position the yanked line or lines from the buffer.

Using External Commands, Saving, and Closing in the vi Editor

Using External Commands

Typing :sh command opens an external command shell. When you exit the shell, you will resume your **vi** editing session.

Typing :!executes a command from within **vi**. The command follows the exclamation point. This technique best suited for non-interactive commands such as:
:! wc %

Typing this will run the wc (word count) command on the file; the character % represents the file currently being edited.

The **fmt** command does simple formatting of text. If you are editing a file and want the file to look nice, you can run the file through **fmt**. One way to do this while editing is by using:%!fmt, which runs the entire file (the % part) through **fmt** and replaces the file with the results.

```
File  Edit  View  Search  Terminal  Help
#!/bin/bash
# Check if file-3 exists
if [ -e test1.txt ]; then
   echo "test1.txt exists. Continue processing."
else

 # else print the second statement and check the next validat
ion.
   echo "test1.txt does not exist. Check temp directory."

# Check if the temp directory exists and  print first stateme
nt if TRUE
   if [ -d temp ]; then  .
      echo "temp directory exists. Create test1.txt in temp."

# create temporary file named file-3
      touch temp /test1.txt
   else

 # else print second statement and perform next validation.
      echo "temp directory does not exist. Is sym link there"
      if [ -s sym-link ]; then
         echo "sym-link exists. All is good."
      else
         echo "sym-link does not exist. Bail out."
         exit 2
      fi
   fi
fi
~
~
:! wc %
```

```
File  Edit  View  Search  Terminal  Help
[test1@localhost ~]$ ls
date2.out  Desktop    Downloads  newfile.txt  Public   sc11.s
date.out   Documents  Music      Pictures     sc11.sh  sc1.sh
[test1@localhost ~]$ vi sc11.sh

 27 119 792 sc11.sh

Press ENTER or type command to continue
```

Introduction to emacs

The **emacs** editor is a popular competitor for **vi**. Unlike **vi**, it does not work with modes. **emacs** is highly customizable and includes a large number of features. It was initially designed for use on a console, but was soon adapted to work with a GUI as well. **emacs** has many other capabilities other than simple text editing; it can be used for email, debugging, etc.

Rather than having different modes for command and insert, like **vi**, **emacs** uses the **CTRL** and **Esc** keys for special commands.

Working with emacs

The table lists some of the most important key combinations that are used when starting, exiting, reading, and writing files in **emacs**.

Key	Usage
emacs myfile	Start emacs and edit myfile
CTRL-x i	Insert prompted for file at current position
CTRL-x s	Save all files
CTRL-x CTRL-w	Write to the file giving a new name when prompted
CTRL-x CTRL-s	Saves the current file
CTRL-x CTRL-c	Exit after being prompted to save any modified files

The **emacs** tutorial is a good place to start learning basic **emacs** commands. It is available any time when in **emacs** by simply typing CTRL-h (for help) and then the letter t for tutorial.

Changing Cursor Positions in emacs

The table lists some of the keys and key combinations that are used for changing cursor positions in emacs.

Key	Usage
arrow keys	Use the arrow keys for up, down, left and right
CTRL-n	One line down
CTRL-p	One line up
CTRL-f	One character forward/right
CTRL-b	One character back/left
CTRL-a	Move to beginning of line
CTRL-e	Move to end of line
Esc-f	Move to beginning of next word
Esc-b	Move back to beginning of preceding word
Esc-<	Move to beginning of file
Esc-x	Goto-line n move to line n
Esc->	Move to end of file
CTRL-v or Page Down	Move forward one page
Esc-v or Page Up	Move backward one page
CTRL-l	Refresh and center screen

Searching for Text in emacs

The table lists the key combinations that are used for searching for text in **emacs**.

Key	Usage
CTRL-s	Search forward for prompted pattern, or for next pattern
CTRL-r	Search backwards for prompted pattern, or for next pattern

Working with Text in emacs

The table lists some of the key combinations used for changing, adding, and deleting text in **emacs**:

Key	Usage
CTRL-o	Insert a blank line
CTRL-d	Delete character at current position
CTRL-k	Delete the rest of the current line
CTRL-_	Undo the previous operation
CTRL- (space or CTRL-@)	Mark the beginning of the selected region. The end will be at the cursor position
CTRL-w	Delete the current marked text and write it to the buffer
CTRL-y	Insert at current cursor location whatever was most recently deleted

Click the link to download a consolidated PDF file with commands for **emacs**.

commands for emacs

In the **vi** editor, which key is used to move the cursor to the top of the screen?

- (•) H
- ( ) M
- ( ) L
- ( ) $

In the **vim** editor, which command is used to launch the tutorial?
vimtutor

Which key combination is used to move the cursor to the beginning of the next word in **emacs**?

- (•) Esc-f
- ( ) Esc-b
- ( ) Esc-<
- ( ) Esc-x

Which of the following is used to exit **emacs** after prompting to save all modified files?

- ( ) CTRL-x c
- ( ) CTRL-x s
- ( ) CTRL-x CTRL-s
- (•) CTRL-x CTRL-c

Labs

Click the link to download a PDF for the Lab activity.

More Advanced Editors: vi and emacs_Labs

More Advanced Editors: vi and emacs_Solution

Summary

You have completed this chapter. Let's summarize the key concepts covered.

- **Text editors** (rather than word processing programs) are used quite often in Linux, for tasks such as for creating or modifying system configuration files, writing scripts, developing source code, etc.
- **nano** is an easy-to-use text-based editor that utilizes on-screen prompts.
- **gedit** is a graphical editor very similar to **Notepad** in **Windows**.
- The **vi** editor is available on all Linux systems and is very widely used. Graphical extension versions of **vi** are widely available as well.
- **emacs** is available on all Linux systems as a popular alternative to **vi**. **emacs** can support both a graphical user interface and a text mode interface.
- To access the **vi** tutorial, type **vimtutor** at a command line window.
- To access the **emacs** tutorial type **Ctl-h** and then **t** from within **emacs.**
- **vi** has three modes: **Command**, **Insert**, and **Line; emacs** has only one but requires use of special keys such as Control and Escape.
- Both editors use various combinations of keystrokes to accomplish tasks; the learning curve to master these can be long but once mastered using either editor is extremely efficient.

Chapter 11: Local Security Principles

Learning Objectives

By the end of this chapter, you should:

- Have a good grasp of best practices and tools for making Linux systems as secure as possible.
- Understand the powers and dangers of using the **root** (**superuser**) account.
- Know how to use the **sudo** command to perform privileged operations while restricting enhanced powers as much as feasible.
- Be able to explain the importance of process isolation and hardware access.
- Know how to work with passwords, including how to set and change them.
- Describe how to secure the boot process and hardware resources.

Section 1: Understanding Linux Security

User Accounts

The Linux kernel allows properly authenticated users to access files and applications. While each user is identified by a unique integer (the user id or **UID),** a separate database associates a **username** with each UID. Upon account creation, new user information is added to the user database and the user's home directory must be created and populated with some essential files. Command line programs such as **useradd** and **userdel** as well as GUI tools are used for creating and removing accounts.

For each user, the following seven fields are maintained in the /etc/passwd file:

Field Name	Details	Remarks
Username	User login name	Should be between 1 and 32 characters long
Password	User password (or the character **x** if the password is stored in the /etc/shadow file) in encrypted format	Is never shown in Linux when it is being typed; this stops prying eyes
User ID (UID)	Every user must have a user id (UID)	• UID 0 is reserved for root user • UID's ranging from 1-99 are reserved for other predefined accounts • UID's ranging from 100-999 are reserved for system accounts and groups (except for RHEL, which reserves only up to 499) • Normal users have UID's of 1000 or greater, except on RHEL where they start at 500
Group ID (GID)	The primary Group ID (GID); Group Identification Number stored in the /etc/group file	Will be covered in detail in the chapter on Processes
User Info	This field is optional and allows insertion of extra information about the user such as their name	For example: Rufus T. Firefly
Home Directory	The absolute path location of user's home directory	For example: /home/rtfirefly
Shell	The absolute location of a user's default shell	For example: /bin/bash

Types of Accounts

By default, Linux distinguishes between several account types in order to isolate processes and workloads. Linux has four types of accounts:

- root
- System
- Normal
- Network

For a safe working environment, it is advised to grant the minimum privileges possible and necessary to accounts, and remove inactive accounts. The **last** utility, which shows the last time each user logged into the system, can be used to help identify potentially inactive accounts which are candidates for system removal.

Keep in mind that practices you use on multi-user business systems are more strict than practices you can use on personal desktop systems that only affect the casual user. This is especially true with security. We hope to show you practices applicable to enterprise servers that you can use on all systems, but understand that you may choose to relax these rules on your own personal system.

```
[root@localhost test1]# last
test1    pts/0        :0.0             Thu Jul  3 07:47   still logged in
test1    tty1         :0               Thu Jul  3 07:47   still logged in
reboot   system boot  2.6.32-431.el6.x Thu Jul  3 07:33 - 07:51  (00:18)
mgnanda  pts/0        :0.0             Mon Jun 30 18:00 - down   (00:30)
mgnanda  tty1         :0               Mon Jun 30 18:00 - down   (00:31)
reboot   system boot  2.6.32-431.el6.x Mon Jun 30 17:53 - 18:31  (00:38)
reboot   system boot  2.6.32-431.el6.x Sun Jun 29 22:39 - 18:31  (19:52)
test1    pts/1        :1.0             Sat Jun 28 07:42 - down   (00:23)
test1    pts/0        :1.0             Sat Jun 28 07:42 - 07:42  (00:00)
test1    tty7         :1               Sat Jun 28 07:41 - down   (00:23)
mgnanda  pts/0        :0.0             Sat Jun 28 07:05 - 07:41  (00:35)
mgnanda  pts/0        :0.0             Sat Jun 28 07:04 - 07:05  (00:01)
mgnanda  pts/0        :0.0             Sat Jun 28 02:48 - 03:02  (00:14)
mgnanda  tty1         :0               Sat Jun 28 02:46 - down   (05:19)
reboot   system boot  2.6.32-431.el6.x Sat Jun 28 02:09 - 08:05  (05:56)
```

Understanding the root Account

root is the most privileged account on a Linux/UNIX system. This account has the ability to carry out all facets of system administration, including adding accounts, changing user passwords, examining log files, installing software, etc. Utmost care must be taken when using this account. It has no security restrictions imposed upon it.

When you are signed in as, or acting as **root**, the shell prompt displays **'#'** (if you are using **bash** and you haven't customized the prompt as we discuss elsewhere in this course). This convention is intended to serve as a warning to you of the absolute power of this account.

Which command can be used to identify how long a user account has remained inactive?

last

Which account has authority over the entire system?

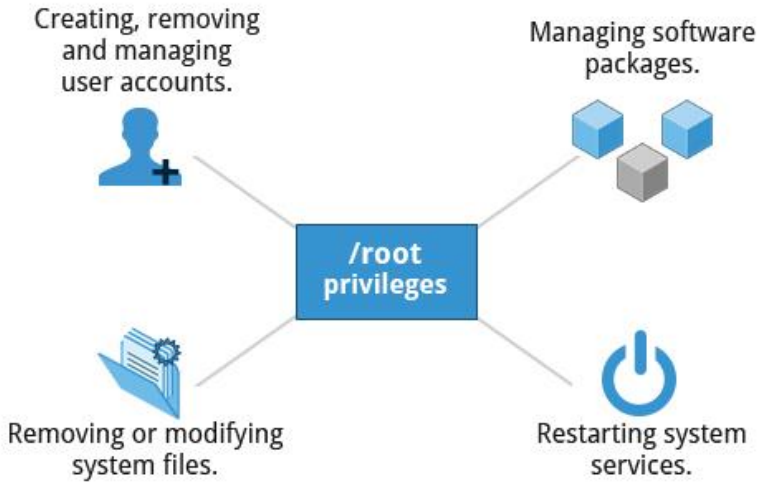
- User account
- Root account Root account
- Master account
- Personal account

Section 2: Understanding the usage of the root account

Operations that Require root Privileges

root privileges are required to perform operations such as:

- Creating, removing and managing user accounts.
- Managing software packages.
- Removing or modifying system files.
- Restarting system services.



Regular account users of Linux distributions may be allowed to install software packages, update some settings, and apply various kinds of changes to the system. However, **root** privilege is required for performing administration tasks such as restarting services, manually installing packages and managing parts of the filesystem that are outside the normal user's directories.

Creating a New User in Linux

To create a new user account:

1. At the command prompt, as root type useradd <username> and press the **ENTER** key.
2. To set the initial password, type passwd <username> and press the **ENTER** key.
   The **New password**: prompt is displayed.
3. Enter the password and press the **ENTER** key.
   To confirm the password, the prompt **Retype new password**: is displayed.
4. Enter the password again and press the **ENTER** key.
   The message **passwd: all authentication tokens updated successfully**. is displayed.

```
File  Edit  View  Search  Terminal  Help
[root@CentOS ~]# useradd kevin
[root@CentOS ~]# passwd kevin
Changing password for user kevin.
New password:
BAD PASSWORD: it is based on a dictionary word
Retype new password:
passwd: all authentication tokens updated successfully.
[root@CentOS ~]#
```

Operations That Do Not Require root Privileges

A regular account user can perform some operations requiring special permissions; however, the system configuration must allow such abilities to be exercised.

SUID (Set owner User ID upon execution—similar to the Windows "run as" feature) is a special kind of file permission given to a file. SUID provides temporary permissions to a user to run a program with the permissions of the file **owner** (which may be root) instead of the permissions held by the user.

The table provides examples of operations which do not require root privileges:

Operations that do not require Root privilege	Examples of this operation
Running a network client	Sharing a file over the network
Using devices such as printers	Printing over the network
Operations on files that the user has proper permissions to access	Accessing files that you have access to or sharing data over the network
Running SUID-root applications	Executing programs such as passwd.

To perform administrative tasks, such as restarting services and manually installing packages, access to _____ privileges is required.

Root

Which of the following tasks does not require **root** privileges?

- [x] Accessing files that the user has proper permissions to access
- [x] Accessing specific devices such as printers
- [ ] Managing software packages
- [ ] Removing or modifying system files

Section 3: Using sudo, the Importance of Process Isolation, Limiting Hardware Access and Keeping Systems Current

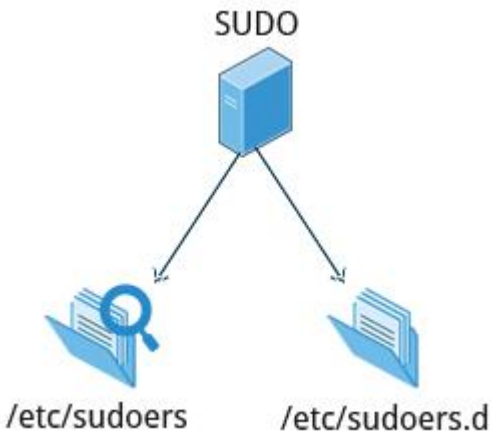
Comparing sudo and su

In Linux you can use either **su** or **sudo** to temporarily grant root access to a normal user; these methods are actually quite different. Listed below are the differences between the two commands.

su	sudo
When elevating privilege, you need to enter the **root** password. Giving the root password to a normal user should **never**, **ever** be done.	When elevating privilege, you need to enter the user's password and not the **root** password.
Once a user elevates to the **root** account using **su,** the user can do **anything** that the **root** user can do for as long as the user wants, without being asked again for a password.	Offers more features and is considered more secure and more configurable. Exactly what the user is allowed to do can be precisely controlled and limited. By default the user will either always have to keep giving their password to do further operations with **sudo**, or can avoid doing so for a configurable time interval.
The command has limited logging features.	The command has detailed logging features.

sudo Features

sudo has the ability to keep track of unsuccessful attempts at gaining root access. Users' authorization for using **sudo** is based on configuration information stored in the /etc/sudoers file and in the /etc/sudoers.d directory.



A message such as the following would appear in a system log file (usually /var/log/secure) when trying to execute **sudo bash** without successfully authenticating the user:

```
authentication failure; logname=op uid=0 euid=0 tty=/dev/pts/6
ruser=op rhost= user=op
conversation failed
auth could not identify password for [op]
op : 1 incorrect password attempt ;
TTY=pts/6 ; PWD=/var/log ; USER=root ; COMMAND=/bin/bash
```

The sudoers File

```
File  Edit  View  Search  Terminal  Help
## Sudoers allows particular users to run various commands as
## the root user, without needing the root password.
##
## Examples are provided at the bottom of the file for collections
## of related commands, which can then be delegated out to particular
## users or groups.
##
## This file must be edited with the 'visudo' command.

## Host Aliases
## Groups of machines. You may prefer to use hostnames (perhaps using
## wildcards for entire domains) or IP addresses instead.
# Host_Alias     FILESERVERS = fs1, fs2
# Host_Alias     MAILSERVERS = smtp, smtp2

## User Aliases
## These aren't often necessary, as you can use regular groups
## (ie, from files, LDAP, NIS, etc) in this file - just use %groupname
## rather than USERALIAS
# User_Alias ADMINS = jsmith, mikem
test1 ALL=(ALL:ALL) ALL
root ALL=(ALL:ALL) ALL
## These are groups of related commands...

## Networking
# Cmnd_Alias NETWORKING = /sbin/route, /sbin/ifconfig, /bin/ping, /sbin/dhclient, /usr/bin/net, /sbin/iptables, /usr/bin/rfcomm, /usr/bin/wvdial, /sbin/iwconfig, /sbin/
mii-tool
"/etc/sudoers.tmp" 117L, 4028C
```

Whenever **sudo** is invoked, a trigger will look at **/etc/sudoers** and the files in **/etc/sudoers.d** to determine if the user has the right to use **sudo** and what the scope of their privilege is. Unknown user requests and requests to do operations not allowed to the user even with **sudo** are reported. You can edit the **sudoers** file by using **visudo**, which ensures that only one person is editing the file at a time, has the proper permissions, and refuses to write out the file and exit if there is an error in the changes made.

The basic structure of an entry is:
who where = (as_whom) what

The file has a lot of documentation in it about how to customize. Most Linux distributions now prefer you add a file in the directory **/etc/sudoers.d** with a name the same as the user. This file contains the individual user's **sudo** configuration, and one should leave the master configuration file untouched except for changes that affect all users.

Command Logging

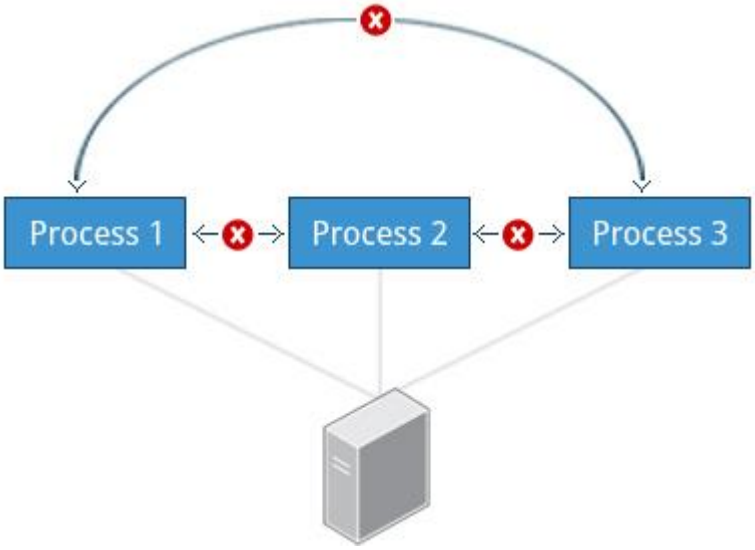
```
File  Edit  View  Search  Terminal  Help
[root@CentOS ~]# sudo tail /var/log/secure
Jul 30 22:33:29 CentOS unix_chkpwd[2630]: password check failed for user (test3)
Jul 30 22:33:29 CentOS pam: gdm-password: pam_unix(gdm-password:auth): authentication failure; logna
me= uid=0 euid=0 tty=:0 ruser= rhost=  user=test3
Jul 30 22:33:38 CentOS pam: gdm-password: pam_unix(gdm-password:session): session opened for user te
st3 by (uid=0)
Jul 30 22:33:38 CentOS polkitd(authority=local): Unregistered Authentication Agent for session /org/
freedesktop/ConsoleKit/Session1 (system bus name :1.26, object path /org/gnome/PolicyKit1/Authentica
tionAgent, locale en_US.UTF-8) (disconnected from bus)
Jul 30 22:33:40 CentOS polkitd(authority=local): Registered Authentication Agent for session /org/fr
eedesktop/ConsoleKit/Session2 (system bus name :1.51 [/usr/libexec/polkit-gnome-authentication-agent
-1], object path /org/gnome/PolicyKit1/AuthenticationAgent, locale en_US.UTF-8)
Jul 30 22:34:04 CentOS sudo:    test3 : user NOT in sudoers ; TTY=pts/0 ; PWD=/home/test3 ; USER=roo
t ; COMMAND=/usr/bin/tail /var/log/secure
Jul 30 22:34:14 CentOS su: pam_unix(su:session): session opened for user root by test3(uid=501)
Jul 30 22:34:25 CentOS sudo:     root : TTY=pts/0 ; PWD=/root ; USER=root ; COMMAND=/usr/bin/tail /v
ar/log/secure
Jul 30 22:34:58 CentOS sudo:     root : TTY=pts/0 ; PWD=/root ; USER=root ; COMMAND=/usr/bin/tail /v
ar/log/secure
Jul 30 22:37:22 CentOS sudo:     root : TTY=pts/0 ; PWD=/root ; USER=root ; COMMAND=/usr/bin/tail /v
ar/log/secure
[root@CentOS ~]#
```

By default, **sudo** commands and any failures are logged in /var/log/auth.log under the **Debian** distribution family, and in /var/log/messages or /var/log/secure on other systems. This is an important safeguard to allow for tracking and accountability of **sudo** use. A typical entry of the message contains:

- Calling username
- Terminal info
- Working directory
- User account invoked
- Command with arguments

Running a command such as sudo whoami results in a log file entry such as:
Dec 8 14:20:47 server1 sudo: op : TTY=pts/6 PWD=/var/log USER=root COMMAND=/usr/bin/whoami

Process Isolation



Linux is considered to be more secure than many other operating systems because processes are naturally **isolated** from each other. One process normally cannot access the resources of another process, even when that process is running with the same user privileges. Linux thus makes it difficult (though certainly not impossible) for viruses and security exploits to access and attack random resources on a system.

Additional security mechanisms that have been recently introduced in order to make risks even smaller are:

- **Control Groups (cgroups)**: Allows system administrators to group processes and associate finite resources to each cgroup.
- **Linux Containers (LXC)**: Makes it possible to run multiple isolated Linux systems (containers) on a single system by relying on **cgroups**.
- **Virtualization**: Hardware is emulated in such a way that not only processes can be isolated, but entire systems are run simultaneously as isolated and insulated guests (virtual machines) on one physical host.

Hardware Device Access

```
root@ubuntu:/home/test1# ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 Jun 10 19:59 /dev/sda
root@ubuntu:/home/test1#
```

Linux limits user access to non-networking hardware devices in a manner that is extremely similar to regular file access. Applications interact by engaging the filesystem layer (which is independent of the actual device or hardware the file resides on). This layer will then opens a **device special file** (often called a **device node**) under the **/dev** directory that corresponds to the device being accessed. Each device special file has standard owner, group and world permission fields. Security is naturally enforced just as it is when standard files are accessed.

Hard disks, for example, are represented as **/dev/sd***. While a root user can read and write to the disk in a **raw** fashion (for example, by doing something like:

```
$ echo hello world > /dev/sda1
```

the standard permissions as shown in the figure make it impossible for regular users to do so. Writing to a device in this fashion can easily obliterate the filesystem stored on it in a way that cannot be repaired without great effort, if at all. The normal reading and writing of files on the hard disk by applications is done at a higher level through the filesystem, and never through direct access to the device node.

Keeping Current

When security problems in either the Linux kernel or applications and libraries are discovered, Linux distributions have a good record of reacting quickly and pushing out fixes to all systems by updating their software repositories and sending notifications to update immediately. The same thing is true with bug fixes and performance improvements that are not security related.

However, it is well known that many systems do not get updated frequently enough and problems which have already been cured are allowed to remain on computers for a long time; this is particularly true with proprietary operating systems where users are either uninformed or distrustful of the vendor's patching policy as sometimes updates can cause new problems and break existing operations. Many of the most successful attack vectors come from exploiting security holes for which fixes are already known but not universally deployed.

So the best practice is to take advantage of your Linux distribution's mechanism for automatic updates and never postpone them. It is extremely rare that such an update will cause new problems.

Which command is used to execute the root commands, without being logged in as the root user?

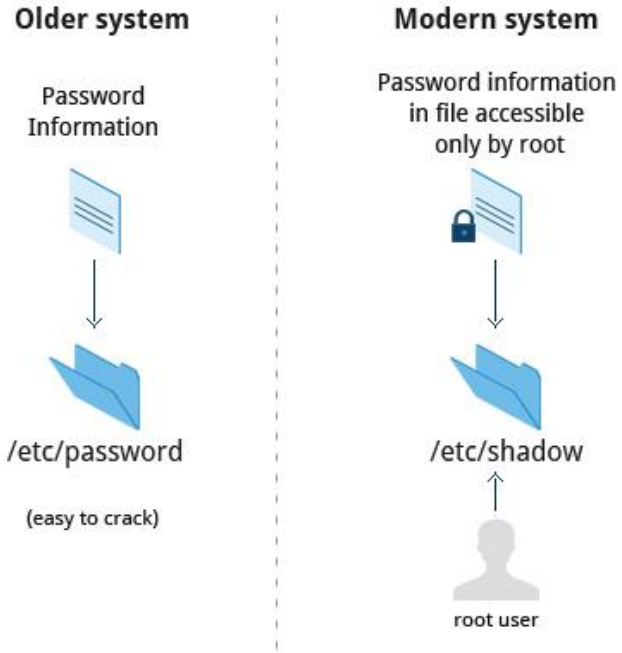
sudo

Identify the log file in which the sudo commands and failures are logged in by default under the **Debian** family?

- (•) **/var/log/auth.log**
- ( ) **/var/messages**
- ( ) **/var/log/USER**
- ( ) **/etc/sudoers**

Section 4: Working with Passwords

How Passwords are Stored



The system verifies authenticity and identity using user credentials. Originally, encrypted passwords were stored in the /etc/passwd file, which was readable by everyone. This made it rather easy for passwords to be cracked. On modern systems, passwords are actually stored in an encrypted format in a secondary file named **/etc/shadow**. Only those with **root access** can modify/read this file.

Password Encryption

```
root@ubuntu:/home/test1# echo -n test | sha512sum
ee26b0dd4af7e749aa1a8ee3c10ae9923f618980772e473f8819a
5d4940e0db27ac185f8a0e1d5f84f88bc887fd67b143732c304cc
5fa9ad8e6f57f50028a8ff  -
root@ubuntu:/home/test1#
```

Protecting passwords has become a crucial element of security. Most Linux distributions rely on a modern password encryption algorithm called **SHA-512** (Secure Hashing Algorithm 512 bits), developed by the U.S. National Security Agency (NSA) to encrypt passwords.

The **SHA-512** algorithm is widely used for security applications and protocols. These security applications and protocols include TLS, SSL, PHP, SSH, S/MIME and IPSec. **SHA-512** is one of the most tested hashing algorithms.

For example, if you wish to experiment with **SHA-512** encoding, the word “test” can be encoded using the program **sha512sum** to produce the **SHA-512** form (see graphic):

Good Password Practices

```
test1@ubuntu:~$ chage --list test1
Last password change                                    : Jul 07, 2014
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
test1@ubuntu:~$
```

IT professionals follow several good practices for securing the data and the password of every user.

1. **Password aging** is a method to ensure that users get prompts that remind them to create a new password after a specific period. This can ensure that passwords, if cracked, will only be usable for a limited amount of time. This feature is implemented using **chage,** which configures the password expiry information for a user.
2. Another method is to force users to set strong passwords using **Pluggable Authentication Modules (PAM)**. **PAM** can be configured to automatically verify that a password created or modified using the **passwd** utility is sufficiently strong. **PAM** configuration is implemented using a library called **pam_cracklib.so**, which can also be replaced by **pam_passwdqc.so** for more options.
3. One can also install password cracking programs, such as **Jack The Ripper**, to secure the password file and detect weak password entries. It is recommended that written authorization be obtained before installing such tools on any system that you do not own.

Shadow passwords used on most modern distributions are stored under which file?

- ○ /etc/passwd
- ◉ /etc/shadow
- ○ /etc/passwd/shadow
- ○ /etc/shadow/password

Identify the Linux utility that ensures that passwords, if cracked, will only be usable for a limited amount of time.
chage

Which of the following mechanisms are used to automatically verify that passwords created or modified using the passwd utility are strong enough?

- ☐ RAM
- ☑ PAM
- ☑ Jack The Ripper
- ☐ Password Aging

The PAM and Jack The Ripper mechanisms are used to automatically verify that passwords created or modified using the passwd utility are strong enough.

Requiring Boot Loader Passwords

```
test1@ubuntu:~$ grub-md5-crypt
Password:
Retype password:
$1$74r8m1$NmkE693AjXre.oF1k0cyK/
test1@ubuntu:~$
```

You can secure the boot process with a secure password to prevent someone from bypassing the user authentication step. For systems using the **GRUB** boot loader, for the older **GRUB** version 1, you can invoke **grub-md5-crypt** which will prompt you for a password and then encrypt as shown on the adjoining screen.

You then must edit /boot/grub/grub.conf by adding the following line below the timeout entry:

password --md5 1Wnvo.1$qz781HRVG4jUnJXmdSCZ30

You can also force passwords for only certain boot choices rather than all.

For the now more common **GRUB** version 2 things are more complicated, and you have more flexibility and can do things like use user-specific passwords, which can be their normal login password. Also you never edit the configuration file, /boot/grub/grub.cfg, directly, rather you edit system configuration files in /etc/grub.d and then run **update-grub**. One explanation of this can be found at https://help.ubuntu.com/community/Grub2/Passwords.

Hardware Vulnerability

When hardware is physically accessible, security can be compromised by:

- Key logging: Recording the real time activity of a computer user including the keys they press. The captured data can either be stored locally or transmitted to remote machines
- Network sniffing: Capturing and viewing the network packet level data on your network
- Booting with a live or rescue disk
- Remounting and modifying disk content

Your IT security policy should start with requirements on how to properly secure physical access to servers and workstations. Physical access to a system makes it possible for attackers to easily leverage several attack vectors, in a way that makes all operating system level recommendations irrelevant.

The guidelines of security are:

- Lock down workstations and servers
- Protect your network links such that it cannot be accessed by people you do not trust
- Protect your keyboards where passwords are entered to ensure the keyboards cannot be tampered with
- Ensure a password protects the BIOS in such a way that the system cannot be booted with a live or rescue DVD or USB key

For single user computers and those in a home environment some of the above features (like preventing booting from removable media) can be excessive, and you can avoid implementing them. However, if sensitive information is on your system that requires careful protection, either it shouldn't be there or it should be better protected by following the above guidelines.

What are the various ways in which security can be compromised if hardware is accessible?

- [x] Key logging
- [x] Network sniffing
- [ ] Locking workstations and servers
- [x] Remounting and modifying disk content

In systems using the GRUB version 1 boot loader, the ____ utility will prompt you for a password, which it will then encrypt.
grub-md5-crypt

Labs

Click the link to download a PDF of the Lab activity.

Local Security Principles_Labs

Local Security Principles_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- The **root** account has authority over the entire system.
- **root** privileges may be required for tasks, such as restarting services, manually installing packages and managing parts of the filesystem that are outside your home directory.
- In order to perform any privileged operations such as system-wide changes, you need to use either **su** or **sudo.**
- Calls to **sudo** trigger a lookup in the **/etc/sudoers** file, or in the **/etc/sudoers.d** directory which first validates that the calling user is allowed to use **sudo** and that it is being used within permitted scope
- One of the most powerful features of **sudo** is its ability to log unsuccessful attempts at gaining root access. By default **sudo** commands and failures are logged in **/var/log/auth.log** under the **Debian** family and **/var/log/messages** in other distribution families.

- One process cannot access another process' resources, even when that process is running with the same user privileges.
- Using the user credentials, the system verifies the authenticity and identity.
- The SHA-512 algorithm is typically used to encode passwords. They can be encrypted but not decrypted.
- Pluggable Authentication Modules (**PAM**) can be configured to automatically verify that passwords created or modified using the **passwd** utility are strong enough (what is considered strong enough can also be configured).
- Your IT security policy should start with requirements on how to properly secure physical access to servers and workstations.
- Keeping your systems updated is an important step in avoiding security attacks.

Chapter 12: Network Operations

Learning Objectives

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

- Explain many basic networking concepts including types of networks and addressing issues.
- Know how to configure network interfaces and use basic networking utilities, such as **ifconfig**, **ip**, **ping**, **route** & **traceroute**.
- Use graphical and non-graphical browsers, such as **Lynx**, **w3m**, **Firefox**, **Chrome** and **Epiphany**.
- Transfer files to and from clients and servers using both graphical and text mode applications, such as **Filezilla**, **ftp**, **sftp**, **curl** and **wget**.

Section 1: Introduction to Networking

Introduction to Networking

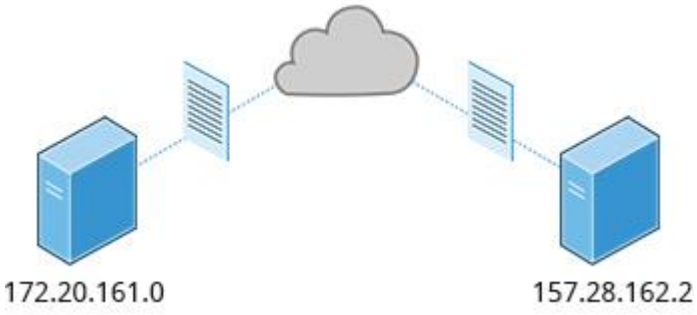
A network is a group of computers and computing devices connected together through communication channels, such as cables or wireless media. The computers connected over a network may be located in the same geographical area or spread across the world.

A network is used to:

- Allow the connected devices to communicate with each other.
- Enable multiple users to share devices over the network, such as printers and scanners.
- Share and manage information across computers easily.

Most organizations have both an internal network and an Internet connection for users to communicate with machines and people outside the organization. The Internet is the largest network in the world and is often called "the network of networks".

IP Addresses



Devices attached to a network must have at least one unique network address identifier known as the IP (**Internet Protocol)** address. The address is essential for routing **packets** of information through the network.

Exchanging information across the network requires using streams of bite-sized packets, each of which contains a piece of the information going from one machine to another. These packets contain **data buffers** together with **headers** which contain information about where the packet is going to and coming from, and where it fits in the sequence of packets that constitute the stream. Networking protocols and software are rather complicated due to the diversity of machines and operating systems they must deal with, as well as the fact that even very old standards must be supported.

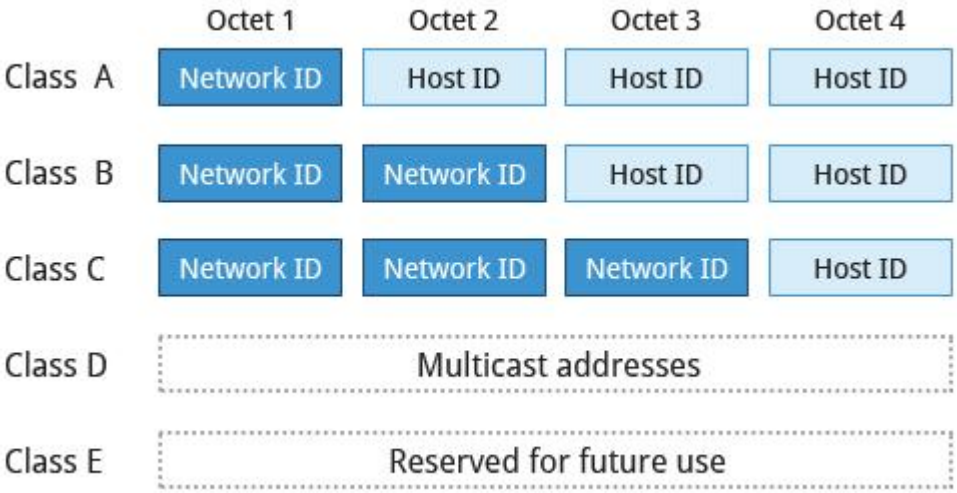
IPv4 and IPv6

There are two different types of IP addresses available: **IPv4** (version 4) and **IPv6** (version 6). **IPv4** is older and by far the more widely used, while **IPv6** is newer and is designed to get past the limitations of the older standard and furnish many more possible addresses.

IPv4 uses 32-bits for addresses; there are ***only*** 4.3 billion unique addresses available. Furthermore, many addresses are allotted and reserved but not actually used. **IPv4** is becoming inadequate because the number of devices available on the global network has significantly increased over the past years.

IPv6 uses 128-bits for addresses; this allows for 3.4 X 10^{38} unique addresses. If you have a larger network of computers and want to add more, you may want to move to **IPv6**, because it provides more unique addresses. However, it is difficult to move to **IPv6** as the two protocols do not inter-operate. Due to this, migrating equipment and addresses to **IPv6** requires significant effort and hasn't been as fast as was originally intended.

Decoding IPv4 Addresses

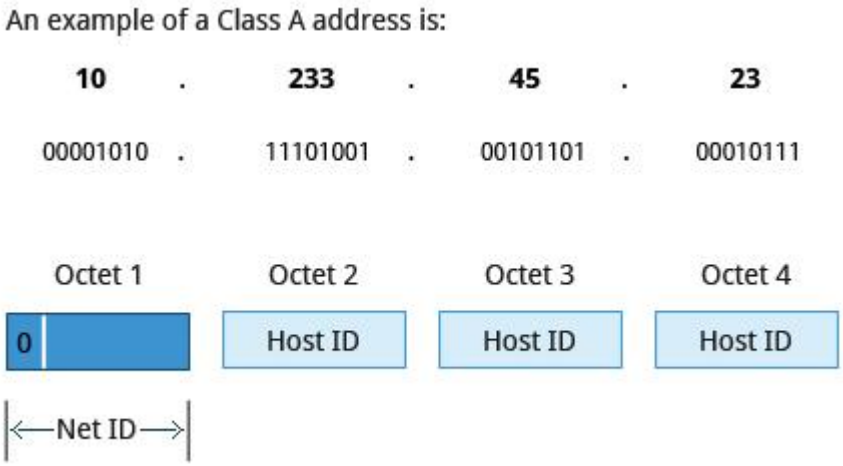


A 32-bit IPv4 address is divided into four 8-bit sections called octets.

Example:
IP address → 172 . 16 . 31 . 46
Bit format → 10101100.00010000.00011111.00101110

Network address are divided into five classes: A, B, C, D, and E. Classes A, B, and C are classified into two parts: **Network addresses (Net ID)** and **Host address (Host ID)**. The Net ID is used to identify the network, while the Host ID is used to identify a host in the network. Class D is used for special multicast applications (information is broadcast to multiple computers simultaneously) and Class E is reserved for future use. In this section you will learn about classes A, B, and C.

Class A Network Addresses

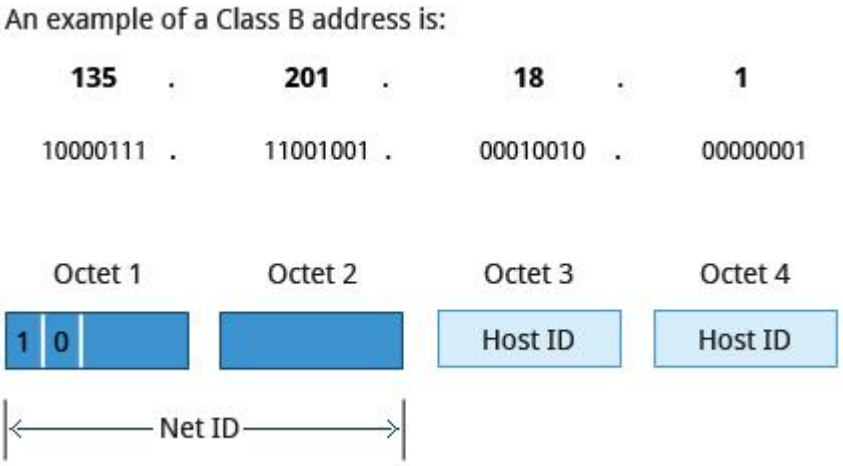


Class A addresses use the first octet of an IP address as their Net ID and use the other three octets as the **Host ID.** The first bit of the first octet is always set to zero. So you can use only 7-bits for unique network numbers. As a result, there are a maximum of 127 Class A networks available. Not surprisingly, this was only feasible when there were very few unique networks with large numbers of hosts. As the use of the Internet expanded, Classes B and C were added in order to accomodate the growing demand for independent networks.

Each Class A network can have up to 16.7 million unique hosts on its network. The range of host address is from 1.0.0.0 to 127.255.255.255.

Note: The value of an octet, or 8-bits, can range from 0 to 255.

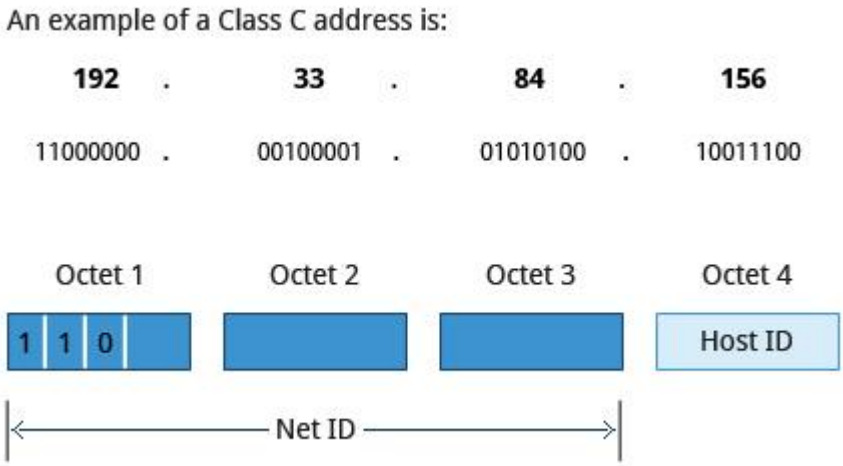
Class B Network Addresses



Class B addresses use the first two octets of the IP address as their Net ID and the last two octets as the Host ID. The first two bits of the first octet are always set to binary 10, so there are a maximum of 16,384 (14-bits) Class B networks. The first octet of a Class B address has values from 128 to 191. The introduction of Class B networks expanded the number of networks but it soon became clear that a further level would be needed.

Each Class B network can support a maximum of 65,536 unique hosts on its network. The range of host address is from 128.0.0.0 to 191.255.255.255.

Class C Network Addresses

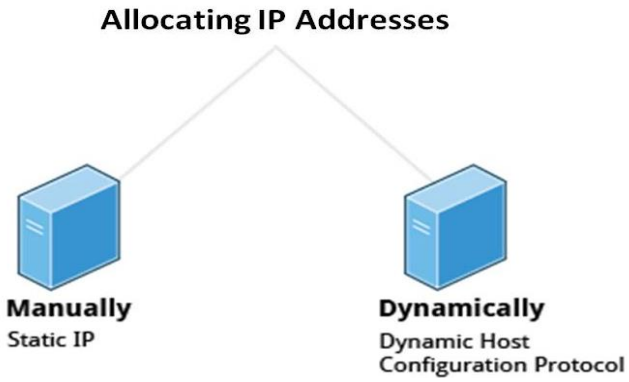


Class C addresses use the first three octets of the IP address as their Net ID and the last octet as their Host ID. The first three bits of the first octet are set to binary 110, so almost 2.1 million (21-bits) Class C networks are available. The first octet of a Class C address has values from 192 to 223. These are most common for smaller networks which don't have many unique hosts.

Each Class C network can support up to 256 (8-bits) unique hosts. The range of host address is from 192.0.0.0 to 223.255.255.255.

IP Address Allocation

Typically, a range of IP addresses are requested from your Internet Service Provider (ISP) by your organization's network administrator. Often your choice of which class of IP address you are given depends on the size of your network and expected growth needs.



You can assign IP addresses to computers over a network manually or dynamically. When you assign IP addresses manually, you add **static** (never changing) addresses to the network. When you assign IP addresses dynamically (they can change every time you reboot or even more often), the **Dynamic Host Configuration Protocol (DHCP)** is used to assign IP addresses.

Manually Allocating an IP Address

Before an IP address can be allocated manually, one must identify the size of the network by determining the host range; this determines which network class (A, B, or C) can be used. The **ipcalc** program can be used to ascertain the host range.

Note: The version of ipcalc supplied in the Fedora family of distributions does not behave as described below, it is really a different program.

Assume that you have a Class C network. The first three octets of the IP address are 192.168.0. As it uses 3 octets (i.e. 24 bits) for the network mask, the shorthand for this type of address is 192.168.0.0/24. To determine the host range of the address you can use for this new host, at the command prompt, type: ipcalc 192.168.0.0/24 and press **Enter**.

From the result, you can check the **HostMin** and **HostMax** values to manually assign a static address available from 1 to 254 (192.168.0.1 to 192.168.0.254).

Name Resolution

Name Resolution is used to convert numerical IP address values into a human-readable format known as the **hostname**. For example, 140.211.169.4 is the numerical IP address that refers to the **linuxfoundation.org** hostname. Hostnames are easier to remember.

Given an IP address, you can obtain its corresponding hostname. Accessing the machine over the network becomes easier when you can type the hostname instead of the IP address.

You can view your system's hostname simply by typing **hostname** with no argument.

Note: If you give an argument, the system will try to change its hostname to match it, however, only root users can do that.

The special hostname **localhost** is associated with the IP address 127.0.0.1**,** and describes the machine you are currently on (which normally has additional network-related IP addresses).

Using Domain Name System (DNS) and Name Resolution Tools

To practice, click the link provided below.

Using Domain Name System (DNS) and Name Resolution Tools

Network Interfaces

```
[test3@CentOS ~]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:4E:9C:9E
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe4e:9c9e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:89 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10618 (10.3 KiB)  TX bytes:11765 (11.4 KiB)

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:16436  Metric:1
          RX packets:44 errors:0 dropped:0 overruns:0 frame:0
          TX packets:44 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3000 (2.9 KiB)  TX bytes:3000 (2.9 KiB)

[test3@CentOS ~]$
```

Network interfaces are a connection channel between a device and a network. Physically, network interfaces can proceed through a **network interface card** (**NIC**) or can be more abstractly implemented as software. You can have multiple network interfaces operating at once. Specific interfaces can be brought up (activated) or brought down (de-activated) at any time.

A list of currently active network interfaces is reported by the **ifconfig** utility which you may have to run as the superuser, or at least, give the full path, i.e., /sbin/ifconfig, on some distributions.

Network Configuration Files

```
test1@ubuntu:~$ cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
test1@ubuntu:~$ /etc/init.d/networking start
test1@ubuntu:~$
```

Network configuration files are essential to ensure that interfaces function correctly.

For **Debian** family configuration, the basic network configuration file is /etc/network/interfaces. You can type /etc/init.d/networking start to start the networking configuration.

For **Fedora** family system configuration, the routing and host information is contained in /etc/sysconfig/network. The network interface configuration script is located at /etc/sysconfig/network-scripts/ifcfg-eth0.

For **SUSE** family system configuration, the routing and host information and network interface configuration scripts are contained in the /etc/sysconfig/network directory.

You can type /etc/init.d/network start to start the networking configuration for **Fedora** and **SUSE** families.

Network Configuration Commands

```
[test3@CentOS ~]$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:4e:9c:9e brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0
    inet6 fe80::a00:27ff:fe4e:9c9e/64 scope link
       valid_lft forever preferred_lft forever
[test3@CentOS ~]$
```

To view the IP address:
$ /sbin/ip addr show

To view the routing information:
$ /sbin/ip route show

ip is a very powerful program that can do many things. Older (and more specific) utilities such as **ifconfig** and **route** are often used to accomplish similar tasks. A look at the relevant **man pages** can tell you much more about these utilities.

ping

ping is used to check whether or not a machine attached to the network can receive and send data; i.e., it confirms that the remote host is online and is responding.

To check the status of the remote host, at the command prompt, type ping <hostname>.

ping is frequently used for network testing and management; however, its usage can increase network load unacceptably. Hence, you can abort the execution of **ping** by typing **CTRL-C**, or by using the **-c** option, which limits the number of packets that **ping** will send before it quits. When execution stops, a summary is displayed.

```
[test3@CentOS Desktop]$ ping google.com
PING google.com (74.125.236.142) 56(84) bytes of data.
64 bytes from bom03s02-in-f14.1e100.net (74.125.236.142
): icmp_seq=1 ttl=56 time=105 ms
64 bytes from bom03s02-in-f14.1e100.net (74.125.236.142
): icmp_seq=2 ttl=56 time=145 ms
64 bytes from bom03s02-in-f14.1e100.net (74.125.236.142
): icmp_seq=3 ttl=56 time=104 ms
64 bytes from bom03s02-in-f14.1e100.net (74.125.236.142
): icmp_seq=4 ttl=56 time=99.8 ms
64 bytes from bom03s02-in-f14.1e100.net (74.125.236.142
): icmp_seq=5 ttl=56 time=100 ms
64 bytes from bom03s02-in-f14.1e100.net (74.125.236.142
): icmp_seq=6 ttl=56 time=104 ms
64 bytes from bom03s02-in-f14.1e100.net (74.125.236.142
): icmp_seq=7 ttl=56 time=107 ms
64 bytes from bom03s02-in-f14.1e100.net (74.125.236.142
): icmp_seq=8 ttl=56 time=103 ms
```

route

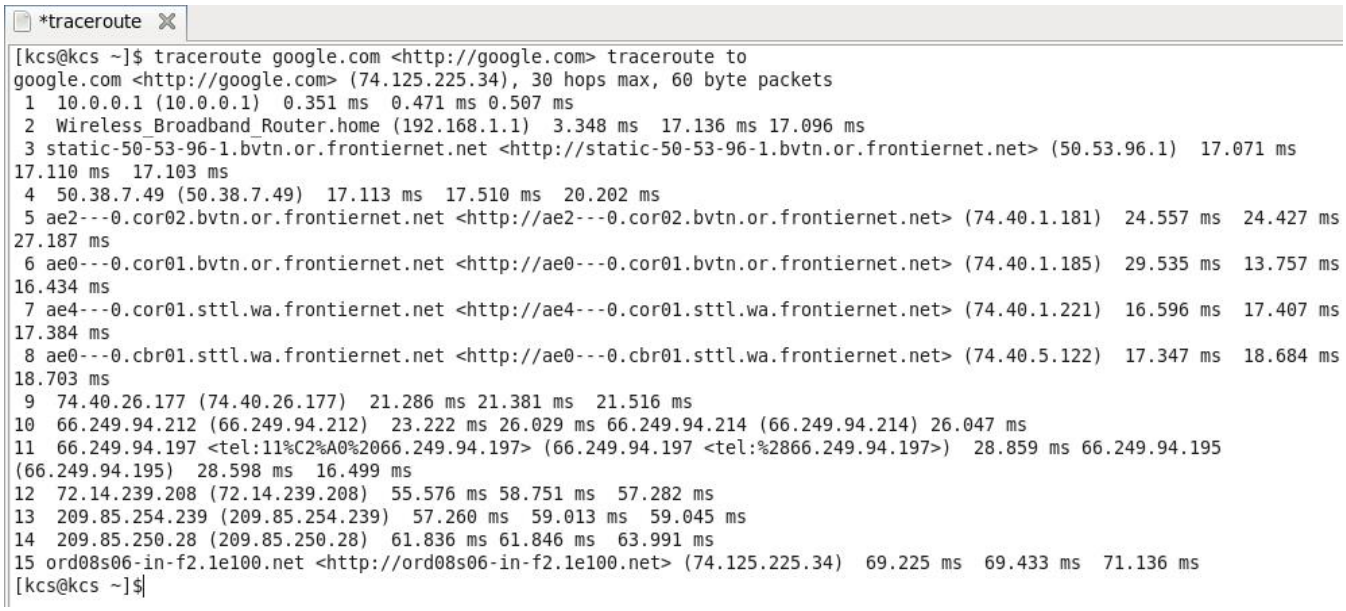
```
[test3@CentOS ~]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1002   0        0 eth0
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
[test3@CentOS ~]$
```

A network requires the connection of many nodes. Data moves from source to destination by passing through a series of routers and potentially across multiple networks. Servers maintain **routing tables** containing the addresses of each node in the network. The **IP Routing protocols** enable routers to build up a forwarding table that correlates final destinations with the next **hop** addresses.

route is used to view or change the IP routing table. You may want to change the IP routing table to add, delete or modify specific (static) routes to specific hosts or networks. The table explains some commands that can be used to manage IP routing.

Task	Command
Show current routing table	$ route –n
Add static route	$ route add -net address
Delete static route	$ route del -net address

traceroute



traceroute is used to inspect the route which the data packet takes to reach the destination host which makes it quite useful for troubleshooting network delays and errors. By using **traceroute** you can isolate connectivity issues between **hops**, which helps resolve them faster.

To print the route taken by the packet to reach the network host, at the command prompt, type traceroute <domain>.

To practice, click the link provided below.

Using ping, route, and traceroute Commands

More Networking Tools

Now, let's learn about some additional networking tools. Networking tools are very useful for monitoring and debugging network problems, such as network connectivity and network traffic.

Networking Tools	Description
ethtool	Queries network interfaces and can also set various parameters such as the speed.
netstat	Displays all active connections and routing tables. Useful for monitoring performance and troubleshooting.
nmap	Scans open **ports** on a network. Important for security analysis
tcpdump	Dumps network traffic for analysis.
iptraf	Monitors network traffic in text mode.

Using More Networking Tools

To practice, click the link provided below.

Using Network Tools

Which command is used to view a system’s hostname?

hostname

Which command checks whether a host is online?

- ping
- route
- traceroute
- netstat

Graphical and Non-Graphical Browsers

Browsers are used to retrieve, transmit, and explore information resources, usually on the **World Wide Web**. Linux users commonly use both graphical and non-graphical browser applications.

The common graphical browsers used in Linux are:

- **Firefox**
- **Google Chrome**
- **Chromium**
- **Epiphany**
- **Opera**

Sometimes you either do not have a graphical environment to work in (or have reasons not to use it) but still need to access web resources. In such a case, you can use non-graphical browsers such as the following:

Non-Graphical Browsers	Description
lynx	Configurable text-based web browser; the earliest such browser and still in use.
links or elinks	Based on **lynx.** It can display tables and frames.
w3m	Newer text-based web browser with many features.

wget

```
test2@OpenSUSE:~> wget linuxfoundation.org
--2014-06-14 01:41:03--  http://linuxfoundation.org/
Resolving linuxfoundation.org (linuxfoundation.org)... 140.211.169.4
Connecting to linuxfoundation.org (linuxfoundation.org)|140.211.169.4|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www.linuxfoundation.org/ [following]
--2014-06-14 01:41:04--  http://www.linuxfoundation.org/
Resolving www.linuxfoundation.org (www.linuxfoundation.org)... 140.211.169.4
Reusing existing connection to linuxfoundation.org:80.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html.1'

    [    <=>                                                                 ] 58,182      71.9KB/s   in 0.8s

2014-06-14 01:41:05 (71.9 KB/s) - 'index.html.1' saved [58182]

test2@OpenSUSE:~>
```

Sometimes you need to download files and information but a browser is not the best choice, either because you want to download multiple files and/or directories, or you want to perform the action from a command line or a script. **wget** is a command line utility that can capably handle the following types of downloads:

- Large file downloads
- Recursive downloads, where a web page refers to other web pages and all are downloaded at once
- Password-required downloads
- Multiple file downloads

To download a webpage, you can simply type wget <url>, and then you can read the downloaded page as a local file using a graphical or non-graphical browser.

curl

Besides downloading you may want to obtain information about a URL, such as the source code being used. **curl** can be used from the command line or a script to read such information. **curl** also allows you to save the contents of a web page to a file as does **wget.**

You can read a URL using curl <URL>. For example, if you want to read http://www.linuxfoundation.org , type curl http://www.linuxfoundation.org.

To get the contents of a web page and store it to a file, type curl -o saved.html http://www.mysite.com. The contents of the main index file at the website will be saved in saved.html.

To practice, click the link provided below.

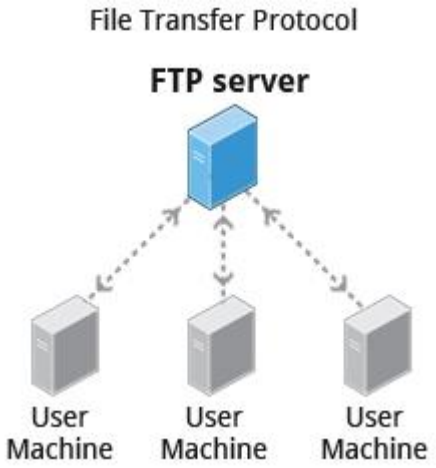
Using wget and curl

Which of the following are graphical browsers used in Linux?

- [ ] Lynx
- [x] Epiphany
- [ ] W3m
- [x] Google Chrome

Which command line utilities are used to download webpages?

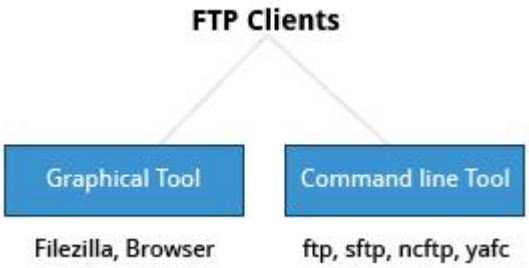
- [x] wget
- [ ] Lynx
- [x] curl
- [ ] Epiphany



FTP (File Transfer Protocol)

When you are connected to a network, you may need to transfer files from one machine to another. **File Transfer Protocol (FTP)** is a well-known and popular method for transferring files between computers using the Internet. This method is built on a **client-server** model. FTP can be used within a browser or with standalone client programs.

FTP Clients



FTP clients enable you to transfer files with remote computers using the FTP protocol. These clients can be either graphical or command line tools. **Filezilla**, for example, allows use of the drag-and-drop approach to transfer files between hosts. All web browsers support FTP, all you have to do is give a URL like : ftp://ftp.kernel.org where the usual http:// becomes ftp://.

Some command line FTP clients are:

- **ftp**
- **sftp**
- **ncftp**
- **yafc** (Yet Another FTP Client)

sftp is a very secure mode of connection, which uses the **Secure Shell** (**ssh)** protocol, which we will discuss shortly. **sftp** encrypts its data and thus sensitive information is transmitted more securely. However, it does not work with so-called **anonymous FTP** (guest user credentials). Both **ncftp** and **yafc** are also powerful FTP clients which work on a wide variety of operating systems including **Windows** and **Linux**.

Connecting to an FTP server

To practice, click the link provided below.

Connecting to an FTP server

SSH: Executing Commands Remotely

Secure Shell (SSH) is a cryptographic network protocol used for secure data communication. It is also used for remote services and other secure services between two devices on the network and is very useful for administering systems which are not easily available to physically work on but to which you have remote access.

To run my_command on a remote system via SSH, at the command prompt, type, ssh <remotesystem> my_command and press **Enter**. **ssh** then prompts you for the remote password. You can also configure **ssh** to securely allow your remote access without typing a password each time.

Copying Files Securely with scp

```
scp <localfile> <user@remotesystem>:/home/user/
```

We can also move files securely using **Secure Copy** (**scp**) between two networked hosts. **scp** uses the SSH protocol for transferring data.

To copy a local file to a remote system, at the command prompt, type scp <localfile> <user@remotesystem>:/home/user/ and press **Enter**.

You will receive a prompt for the remote password. You can also configure **scp** so that it does not prompt for a password for each transfer.

Which of the following protocols are used for transferring files between two servers?

- [x] FTP
- [ ] TCP
- [ ] UDC
- [x] HTTP

Which protocol is used for executing commands remotely?

- ( ) ftp
- (•) ssh
- ( ) scp
- ( ) yafc

Labs

Click the link to download a PDF for the Lab activity.

Network Operations_Labs

Network Operations_Solution

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- The **IP** (Internet Protocol) **address** is a unique logical network address that is assigned to a device on a network.
- **IPv4** uses 32-bits for addresses and **IPv6** uses 128-bits for addresses.
- Every IP address contains both a network and a host address field.
- There are five classes of network addresses available: A, B, C, D & E.
- **DNS** (Domain Name System) is used for converting Internet domain and host names to IP addresses.
- The **ifconfig** program is used to display current active network interfaces.
- The commands ip addr show and ip route show can be used to view IP address and routing information.
- You can use **ping** to check if the remote host is alive and responding.

- You can use the **route** utility program to manage IP routing.
- You can monitor and debug network problems using networking tools.
- **Firefox**, **Google Chrome**, **Chromium**, and **Epiphany** are the main graphical browsers used in Linux.
- Non-graphical or text browsers used in Linux are **Lynx**, **Links**, and **w3m**.
- You can use **wget** to download webpages.
- You can use **curl** to obtain information about URL's.
- **FTP** (File Transfer Protocol) is used to transfer files over a network.
- **ftp**, **sftp**, **ncftp**, and **yafc** are command line FTP clients used in Linux.
- You can use **ssh** to run commands on remote systems.

Chapter 13: Manipulating Text

Learning Objectives

By the end of this chapter, 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 file and text manipulation.

Section 1: cat & echo

Command Line Tools

Irrespective of the role you play with Linux (system administrator, developer, or user) you often need to browse through and parse text files, and/or extract data from them. These are **file manipulation** operations. Thus it is essential for the Linux user to become adept at performing certain operations on files.

Most of the time such file manipulation is done at the **command line** which allows users to perform tasks more efficiently than while using a GUI. Furthermore the command line is more suitable for automating often executed tasks.

Indeed, experienced system administrators write customized scripts to accomplish such repetitive tasks, standardized for each particular environment. We will discuss such scripting later in much detail.

In this section, we will concentrate on command line file and text manipulation related utilities.

cat

cat is short for concatenate and is one of the most frequently used Linux command line utilities. It is often used to read and print files as well as for simply viewing file contents. To view a file, use the following command:
$ cat <filename>

For example, cat readme.txt will display the contents of readme.txt on the terminal. Often the main purpose of **cat,** however, is to combine (concatenate) multiple files together. You can perform the actions listed in the following table using **cat**:

Command	Usage
cat file1 file2	Concatenate multiple files and display the output; i.e., the entire content of the first file is followed by that of the second file.
cat file1 file2 > newfile	Combine multiple files and save the output into a new file.
cat file >> existingfile	Append a file to the end of an existing file.
cat > file	Any subsequent lines typed will go into the file until CTRL-D is typed.
cat >> file	Any subsequent lines are appended to the file until CTRL-D is typed.

The **tac** command (**cat** spelled backwards) prints the lines of a file in reverse order. (Each line remains the same but the order of lines is inverted.) The syntax of **tac** is exactly the same as for **cat** as in:

$ tac file
$ tac file1 file2 > newfile

Using cat Interactively

```
[test1@localhost ~]$ cat > test1.txt
This is my first file
[test1@localhost ~]$ cat > test2.txt
This is my second file
[test1@localhost ~]$ cat > test3.txt
This is my third file
[test1@localhost ~]$ ls
Desktop    Downloads  Pictures  Templates  test2.txt  Videos
Documents  Music      Public    test1.txt  test3.txt
[test1@localhost ~]$ cat test1.txt test2.txt
This is my first file
This is my second file
[test1@localhost ~]$ cat test1.txt test2.txt > newfile.txt
[test1@localhost ~]$ cat newfile.txt
This is my first file
This is my second file
[test1@localhost ~]$ cat test3.txt >> newfile.txt
[test1@localhost ~]$ cat newfile.txt
This is my first file
This is my second file
This is my third file
[test1@localhost ~]$ cat >> newfile.txt
This line is appended to newfile from command line
[test1@localhost ~]$ cat >> newfile.txt
This second line is also appended to newfile from command line
[test1@localhost ~]$ cat newfile.txt
This is my first file
This is my second file
This is my third file
This line is appended to newfile from command line
This second line is also appended to newfile from command line
[test1@localhost ~]$
```

cat can be used to read from standard input (such as the terminal window) if no files are specified. You can use the > operator to create and add lines into a new file, and the >> operator to append lines (or files) to an existing file.

To create a new file, at the command prompt type cat > <filename> and press the **Enter** key.

This command creates a new file and waits for the user to edit/enter the text. After you finish typing the required text, press **CTRL-D** at the beginning of the next line to save and exit the editing.

Another way to create a file at the terminal is cat > <filename> << EOF. A new file is created and you can type the required input. To exit, enter EOF at the beginning of a line.

Note that EOF is case sensitive. (One can also use another word, such as STOP.)

Using cat

To practice, click the link provided below.

Using cat

echo

echo simply displays (echoes) text. It is used simply as in:

$ echo string

echo can be used to display a string on **standard output** (i.e., the terminal) or to place in a new file (using the > operator) or append to an already existing file (using the >> operator).

The –e option along with the following switches is used to enable special character sequences, such as the **newline** character or horizontal **tab**.

- \n represents newline
- \t represents horizontal tab

echo is particularly useful for viewing the values of environment variables (built-in shell variables). For example, echo $USERNAME will print the name of the user who has logged into the current terminal.

The following table lists **echo** commands and their usage:

Command	Usage
echo string > newfile	The specified string is placed in a new file.
echo string >> existingfile	The specified string is appended to the end of an already existing file.
echo $variable	The contents of the specified environment variable are displayed.

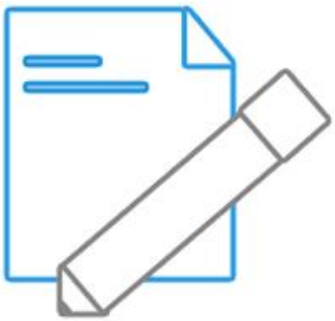
Which command is used to concatenate three files?

- cat file1 file2 file3
- cat file1 file2 > newfile
- cat file >> existingfile
- cat file1 > file 2 > file 3

Which command is used to display a line of text?
echo

Section 2: sed & awk

Introduction to sed and awk



It is very common to create and then repeatedly edit and/or extract contents from a file. Let's learn how to use **sed** and **awk** to easily perform such operations.

Note that many Linux users and administrators will write scripts using more comprehensive language utilities such as **python** and **perl**, rather than use **sed** and **awk** (and some other utilities we'll discuss later.) Using such utilities is certainly fine in most circumstances; one should always feel free to use the tools one is experienced with. However, the utilities that are described here are much lighter; i.e., they use fewer system resources, and execute faster. There are times (such as during booting the system) where a lot of time would be wasted using the more complicated tools, and the system may not even be able to run them. So the simpler tools will always be needed.

sed

sed is a powerful text processing tool and is one of the oldest earliest and most popular UNIX utilities. It is used to modify the contents of a file, usually placing the contents into a new file. Its name is an abbreviation for **stream editor**.

sed can filter text as well as perform substitutions in data streams, working like a churn-mill.

Data from an input source/file (or stream) is taken and moved to a working space. The entire list of operations/modifications is applied over the data in the working space and the final contents are moved to the standard output space (or stream).

sed Command Syntax

You can invoke **sed** using commands like those listed in the following table:

Command	Usage
sed -e command <filename>	Specify editing commands at the command line, operate on file and put the output on standard out (e.g., the terminal)
sed -f scriptfile <filename>	Specify a scriptfile containing sed commands, operate on file and put output on standard out.

The -e command option allows you to specify multiple editing commands simultaneously at the command line.

sed Basic Operations

Now that you know that you can perform multiple editing and filtering operations with **sed**, let's explain some of them in more detail. The table explains some basic operations, where pattern is the current string and replace_string is the new string:

Command	Usage
sed s/pattern/replace_string/ file	Substitute first string occurrence in a line
sed s/pattern/replace_string/g file	Substitute all string occurrences in a line
sed 1,3s/pattern/replace_string/g file	Substitute all string occurrences in a range of lines
sed -i s/pattern/replace_string/g file	Save changes for string substitution in the same file

You must use the -i option with care, because the action is not reversible. It is always safer to use **sed** without the –i option and then replace the file yourself, as shown in the following example:

```
$ sed s/pattern/replace_string/g file > file2
```

The above command will replace all occurrences of pattern with replace_string in file1 and move the contents to file2. The contents of file2 can be viewed with cat file2. If you approve you can then overwrite the original file with mv file2 file1.

Example: To convert 01/02/… to JAN/FEB/…

```
sed -e 's/01/JAN/' -e 's/02/FEB/' -e 's/03/MAR/' -e 's/04/APR/' -e 's/05/MAY/' \
-e 's/06/JUN/' -e 's/07/JUL/' -e 's/08/AUG/' -e 's/09/SEP/' -e 's/10/OCT/' \
-e 's/11/NOV/' -e 's/12/DEC/'
```

Using sed

To practice, click the link provided below.

Using sed

awk

awk is used to extract and then print specific contents of a file and is often used to construct reports. It was created at Bell Labs in the 1970s and derived its name from the last names of its authors: Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan.

awk has the following features:

- It is a powerful utility and interpreted programming language.
- It is used to manipulate data files, retrieving, and processing text.
- It works well with **fields** (containing a single piece of data, essentially a column) and **records** (a collection of fields, essentially a line in a file).

awk is invoked as shown in the following:

Command	Usage
awk 'command' var=value file	Specify a command directly at the command line
awk -f scriptfile var=value file	Specify a file that contains the script to be executed along with f

As with **sed**, short **awk** commands can be specified directly at the command line, but a more complex script can be saved in a file that you can specify using the -f option.

awk Basic Operations

The table explains the basic tasks that can be performed using **awk**. The input file is read one line at a time, and for each line, **awk** matches the given pattern in the given order and performs the requested action. The -F option allows you to specify a particular **field separator** character. For example, the /etc/passwd file uses : to separate the fields, so the -F: option is used with the /etc/passwd file.

The command/action in **awk** needs to be surrounded with apostrophes (or single-quote ('')). awk can be used as follows:

Command	Usage
awk '{ print $0 }' /etc/passwd	Print entire file
awk -F: '{ print $1 }' /etc/passwd	Print first field (column) of every line, separated by a space
awk -F: '{ print $1 $6 }' /etc/passwd	Print first and sixth field of every line

To practice, click the link provided below.

Using awk

___ utility is used as a data extraction and reporting tool.

- awk
- ps
- gnome-help
- sudo

Consider a data file /usr/data/employee that contains the information of 300 employees, in the following format. (emp-name:age:date-of-birth)
Select the correct syntax of the awk command that prints name and date-of-birth.

- $ awk -F: '{ print $1 ; $3 }' /usr/data/employee
- $ awk -F: '{ print $1 & $3 }' /usr/data/employee
- $ awk -F: '{ print $1 $3 }' /usr/data/employee
- $ awk -F: '{ print $1 - $3 }' /usr/data/employee

Section 3: File Manipulation Utilities

File Manipulation Utilities

In managing your files you may need to perform many tasks, such as sorting data and copying data from one location to another. Linux provides several file manipulation utilities that you can use while working with text files. In this section, you will learn about the following file manipulation programs:

- sort
- uniq
- paste
- join
- split

You will also learn about **regular expressions** and **search patterns**.

sort

```
[test3@CentOS Desktop]$ sort -u names
Alice
Bob
Carol
Earth
John
Mars
Mercury
Neptune
Sheik
Ted
Venus
[test3@CentOS Desktop]$ sort -r names
Venus
Venus
Ted
Sheik
Neptune
Mercury
```

```
[test3@CentOS Desktop]$ cat names
Alice
Carol
Bob
Ted
Carol
Alice
Mercury
Mars
Earth
Neptune
Earth
Venus
Mercury
Venus
John
Sheik
John
[test3@CentOS Desktop]$
```

sort is used to rearrange the lines of a text file either in ascending or descending order, according to a sort key. You can also sort by particular fields of a file. The default sort key is the order of the ASCII characters (i.e., essentially alphabetically).

sort can be used as follows:

Syntax	Usage
sort <filename>	Sort the lines in the specified file
cat file1 file2 | sort	Append the two files, then sort the lines and display the output on the terminal
sort -r <filename>	Sort the lines in reverse order

When used with the -u option, **sort** checks for unique values after sorting the records (lines). It is equivalent to running **uniq** (which we shall discuss) on the output of **sort**.

uniq

uniq is used to remove duplicate lines in a text file and is useful for simplifying text display. **uniq** requires that the duplicate entries to be removed are consecutive. Therefore one often runs **sort** first and then pipes the output into **uniq**; if **sort** is passed the -u option it can do all this in one step. In the example shown, the file is called names and was originally Ted, Bob, Alice, Bob, Carol, Alice.

```
[test3@CentOS Desktop]$ sort names | uniq
Alice
Bob
Carol
Ted
[test3@CentOS Desktop]$
```

To remove duplicate entries from some files, use the following command:

sort file1 file2 | uniq > file3 **OR** sort -u file1 file2 > file3

To count the number of duplicate entries, use the following command: uniq -c filename

To practice, click the link provided below.

Using sort and uniq

paste

Suppose you have a file that contains the full name of all employees and another file that lists their phone numbers and Employee IDs. You want to create a new file that contains all the data listed in three columns: name, employee ID, and phone number. How can you do this effectively without investing too much time?

paste can be used to create a single file containing all three columns. The different columns are identified based on delimiters (spacing used to separate two fields). For example, delimiters can be a blank space, a tab, or an **Enter**. In the image provided, a single space is used as the delimiter in all files.

paste accepts the following options:

- -d delimiters, which specify a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, paste begins again at the first delimiter.
- -s, which causes **paste** to append the data in series rather than in parallel; that is, in a horizontal rather than vertical fashion.

Using paste

paste can be used to combine fields (such as name or phone number) from different files as well as combine lines from multiple files. For example, line one from file1 can be combined with line one of file2, line two from file1 can be combined with line two of file2, and so on.

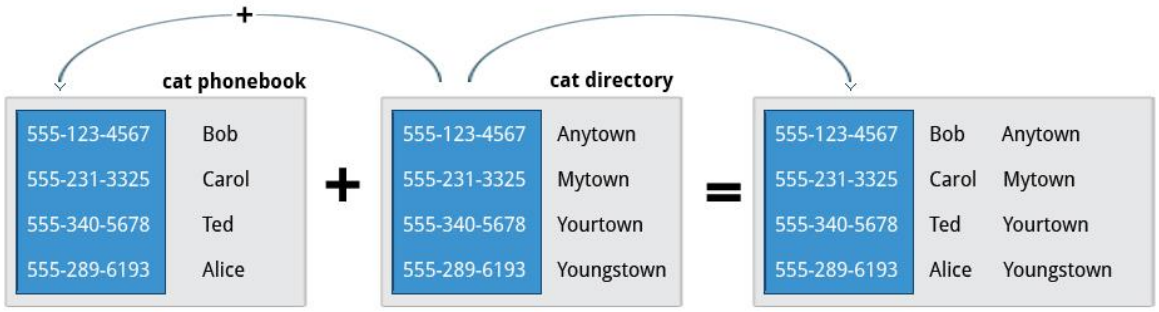
```
[test3@CentOS Desktop]$ cat phone
555-123-4567
555-231-3325
555-340-5678
555-289-6193
[test3@CentOS Desktop]$ cat names
Bob
Carol
Ted
Alice
[test3@CentOS Desktop]$ paste -d ':' names phone
Bob:555-123-4567
Carol:555-231-3325
Ted:555-340-5678
Alice:555-289-6193
[test3@CentOS Desktop]$
```

To paste contents from two files one can do:
$ paste file1 file2

The syntax to use a different delimiter is as follows:
$ paste -d, file1 file2

Common delimiters are 'space', 'tab', '|', 'comma', etc.

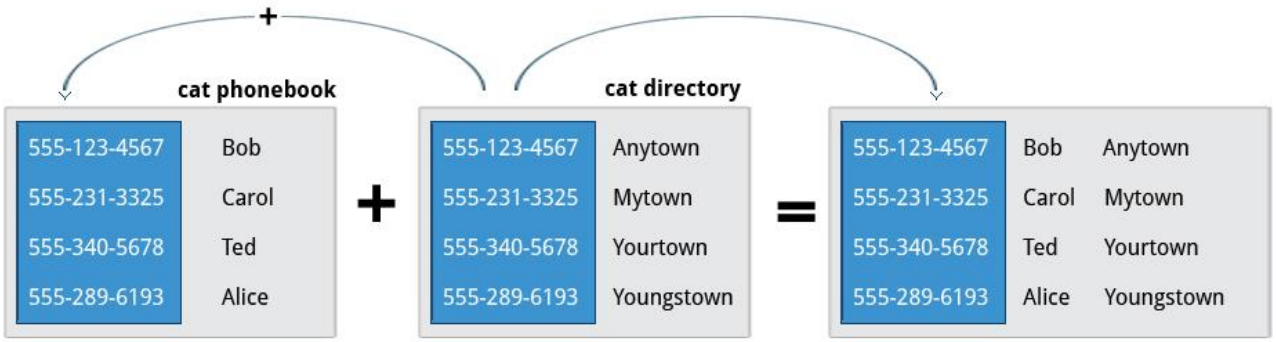
join



Suppose you have two files with some similar columns. You have saved employees' phone numbers in two files, one with their first name and the other with their last name. You want to combine the files without repeating the data of common columns. How do you achieve this?

The above task can be achieved using **join**, which is essentially an enhanced version of **paste**. It first checks whether the files share common fields, such as names or phone numbers, and then joins the lines in two files based on a common field.

Using join



To combine two files on a common field, at the command prompt type join file1 file2 and press the **Enter** key.

For example, the common field (i.e., it contains the same values) among the phonebook and directory files is the phone number, as shown by the output of the following **cat** commands:

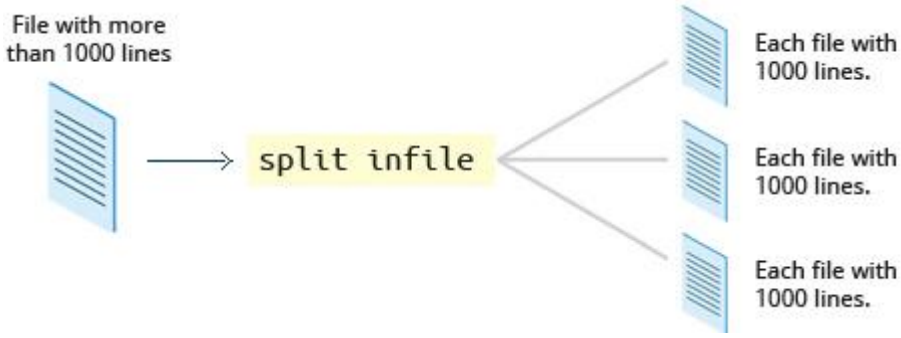
```
$ cat phonebook
555-123-4567 Bob
555-231-3325 Carol
555-340-5678 Ted
555-289-6193 Alice

$ cat directory
555-123-4567 Anytown
555-231-3325 Mytown
555-340-5678 Yourtown
555-289-6193 Youngstown
```

The result of **join**ing these two file is as shown in the output of the following command:

```
$ join phonebook directory
555-123-4567 Bob Anytown
555-231-3325 Carol Mytown
555-340-5678 Ted Yourtown
555-289-6193 Alice Youngstown
```

split

split is used to break up (or split) a file into equal-sized segments for easier viewing and manipulation, and is generally used only on relatively large files. By default **split** breaks up a file into 1,000-line segments. The original file remains unchanged, and set of new files with the same name plus an added prefix is created. By default, the **x** prefix is added. To split a file into segments, use the command split infile.



To split a file into segments using a different prefix, use the command split infile <Prefix>.

Using split

```
[test1@localhost ~]$ wc words
 479829  479829 4953699 words
[test1@localhost ~]$ split words
[test1@localhost ~]$ ls -l words
-rw-rw-r--. 1 test1 test1 4953699 Jul 26 08:12 words
[test1@localhost ~]$ split words words1
[test1@localhost ~]$ ls -l word*
-rw-rw-r--. 1 test1 test1 4953699 Jul 26 08:12 words
-rw-rw-r--. 1 test1 test1    8699 Jul 26 08:13 words1aa
-rw-rw-r--. 1 test1 test1   10091 Jul 26 08:13 words1ab
-rw-rw-r--. 1 test1 test1   10689 Jul 26 08:13 words1ac
-rw-rw-r--. 1 test1 test1   10177 Jul 26 08:13 words1ad
-rw-rw-r--. 1 test1 test1    9850 Jul 26 08:13 words1ae
-rw-rw-r--. 1 test1 test1    9847 Jul 26 08:13 words1af
-rw-rw-r--. 1 test1 test1    9994 Jul 26 08:13 words1ag
-rw-rw-r--. 1 test1 test1    9052 Jul 26 08:13 words1ah
-rw-rw-r--. 1 test1 test1    8403 Jul 26 08:13 words1ai
-rw-rw-r--. 1 test1 test1    9363 Jul 26 08:13 words1aj
-rw-rw-r--. 1 test1 test1   10271 Jul 26 08:13 words1ak
-rw-rw-r--. 1 test1 test1   10036 Jul 26 08:13 words1al
-rw-rw-r--. 1 test1 test1    9462 Jul 26 08:13 words1am
-rw-rw-r--. 1 test1 test1    9850 Jul 26 08:13 words1an
-rw-rw-r--. 1 test1 test1   10447 Jul 26 08:13 words1ao
```

To demonstrate the use of **split,** we'll apply it to an american-english dictionary file of over 99,000 lines:

```
$ wc -l american-english
99171 american-english
```

where we have used the **wc** program (soon to be discussed) to report on the number of lines in the file. Then typing:

```
$ split american-english dictionary
```

will split the american-english file into equal-sized segments named 'dictionary'.

```
$ ls -l dictionary*
-rw-rw-r 1 me me 8552 Mar 23 20:19 dictionaryab
-rw-rw-r 1 me me 8653 Mar 23 20:19 dictionaryaa
. . .
```

Regular Expressions and Search Patterns

Regular expressions are text strings used for matching a specific **pattern**, or to search for a specific location, such as the start or end of a line or a word. Regular expressions can contain both normal characters or so-called **metacharacters**, such as * and $.

Many text editors and utilities such as **vi**, **sed**, **awk**, **find** and **grep** work extensively with regular expressions. Some of the popular computer languages that use regular expressions include **Perl**, **Python** and **Ruby**. It can get rather complicated and there are whole books written about regular expressions; we'll only skim the surface here.

These regular expressions are different from the wildcards (or "metacharacters") used in filename matching in command shells such as **bash** (which were covered in the earlier Chapter on Command Line Operations)**.** The table lists search patterns and their usage.

Search Patterns	Usage
.(dot)	Match any single character
a|z	Match a or z
$	Match end of string
*	Match preceding item 0 or more times

Using Regular Expressions and Search Patterns

For example, Consider the following sentence:

the quick brown fox jumped over the lazy dog

Some of the patterns that can be applied to this sentence are as follows:

Command	Usage
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

Identify the command option that should be present in the paste command to combine two files with a specified delimiter?

- (•) -d
- ( ) -n
- ( ) -t
- ( ) -r

Which command will join two files that share a common field?

- ( ) merge
- ( ) paste
- (•) join
- ( ) cat

In search patterns, what is the syntax to match the end of the string?

- [ ] .
- [ ] a|z
- [x] $
- [ ] *

Section 4: grep

grep

grep is extensively used as a primary text searching tool. It scans files for specified patterns and can be used with regular expressions as well as simple strings as shown in the table.

Command	Usage
grep [pattern] <filename>	Search for a pattern in a file and print all matching lines
grep -v [pattern] <filename>	Print all lines that do **not** match the pattern
grep [0-9] <filename>	Print the lines that contain the numbers 0 through 9
grep -C 3 [pattern] <filename>	Print context of lines (specified number of lines above and below the pattern) for matching the pattern. Here the number of lines is specified as 3.

Which commands can be used to print the lines that contain the numbers 0–5 in a file?

- [x] grep [0,1,2,3,4,5] filename
- [x] grep [0-5] filename
- [ ] grep [-e0 -e1 -e2 -e3 -e4 -e5] filename
- [ ] grep {0-5} filename

Which of the following grep commands can be used to print out all of the lines that don't match the specified pattern?

- [ ] grep -C 3 [pattern] <filename>
- [x] grep -v [pattern] <filename>
- [ ] grep [pattern] <filename>
- [ ] grep -w [pattern] <filename>

Section 5: Miscellaneous Text Utilities

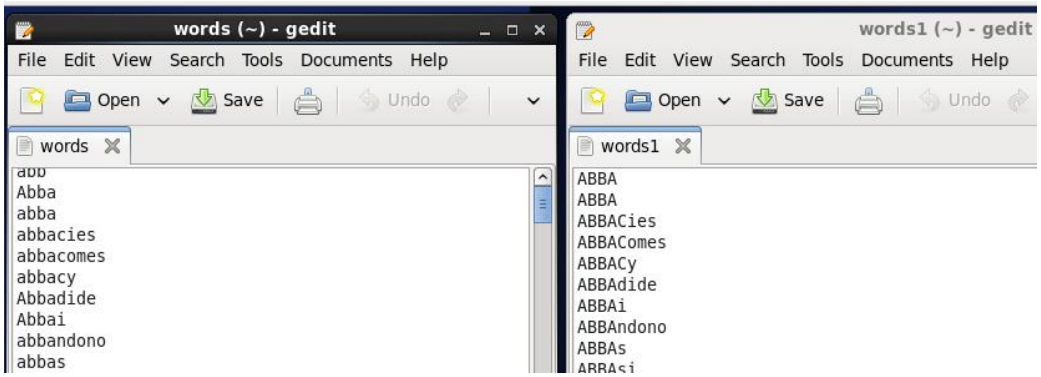
tr

In this section, you will learn about some additional text utilities that you can use for performing various actions on your Linux files, such as changing the case of letters or determining the count of words, lines, and characters in a file.

The **tr** utility is used to **translate** specified characters into other characters or to delete them. The general syntax is as follows:

```
$ tr [options] set1 [set2]
```

```
[test1@localhost ~]$ cat words | tr 'abc' 'ABC' > words1
[test1@localhost ~]$
```



The items in the square brackets are optional. **tr** requires at least one argument and accepts a maximum of two. The first, designated set1 in the example, lists the characters in the text to be replaced or removed. The second, set2, lists the characters that are to be substituted for the characters listed in the first argument. Sometimes these sets need to be surrounded by apostrophes (or single-quotes (')) in order to have the shell ignore that they mean something special to the shell. It is usually safe (and may be required) to use the single-quotes around each of the sets as you will see in the examples below.

For example, suppose you have a file named city containing several lines of text in mixed case. To translate all lower case characters to upper case, at the command prompt type cat city | tr a-z A-Z and press the **Enter** key.

Command	Usage
$ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ	Convert lower case to upper case
$ tr '{}' '()' < inputfile > outputfile	Translate braces into parenthesis
$ echo "This is for testing" | tr [:space:] '\t'	Translate white-space to tabs
$ echo "This is for testing" | tr -s [:space:]	Squeeze repetition of characters using -s
$ echo "the geek stuff" | tr -d 't'	Delete specified characters using -d option
$ echo "my username is 432234" | tr -cd [:digit:]	Complement the sets using -c option
$ tr -cd [:print:] < file.txt	Remove all non-printable character from a file
$ tr -s '\n' ' ' < file.txt	Join all the lines in a file into a single line

tee

```
test3@CentOS lm]$ ls -l | tee newfile
otal 0
rw-rw-r--. 1 test3 test3 0 Jul 30 13:53 newfile
test3@CentOS lm]$ cat newfile
otal 0
rw-rw-r--. 1 test3 test3 0 Jul 30 13:53 newfile
test3@CentOS lm]$
```

tee takes the output from any command, and while sending it to standard output, it also saves it to a file. In other words, it "tees" the output stream from the command: one stream is displayed on the standard output and the other is saved to a file.

For example, to list the contents of a directory on the screen and save the output to a file, at the command prompt type ls -l | tee newfile and press the **Enter** key.

Typing cat newfile will then display the output of ls –l.

wc

```
[test3@CentOS Desktop]$ wc newtest
 2  9 44 newtest
[test3@CentOS Desktop]$ wc -l newtest
2 newtest
[test3@CentOS Desktop]$ wc -c newtest
44 newtest
[test3@CentOS Desktop]$ wc -w newtest
9 newtest
[test3@CentOS Desktop]$
```

wc (word count) counts the number of lines, words, and characters in a file or list of files. Options are given in the table below.

By default all three of these options are active.

For example, to print the number of lines contained in a file, at the command prompt type wc -l filename and press the **Enter** key.

Option	Description
–l	display the number of lines.
-c	display the number of bytes.
-w	display the number of words.

cut

```
[test3@CentOS Desktop]$ ls -l | cut -d" " -f3

test3
test3
test3
test3
test3
test3
test3
test3
[test3@CentOS Desktop]$
```

cut is used for manipulating column-based files and is designed to extract specific columns. The default column separator is the **tab** character. A different delimiter can be given as a command option.

For example, to display the third column delimited by a blank space, at the command prompt type ls -l | cut -d" " -f3 and press the **Enter** key.

To practice, click the link provided below.

Using tr

To practice, click the link provided below.

Using wc

Which command is used to extract columns from a file to work on them later?

- tr
- tee
- wc
- cut

```
$ less filename
$ cat filename | less
```

Searches content piece by piece in large files.

Section 6: Dealing with Large Files and Text-related Utilities

Working with Large Files

System administrators need to work with configuration files, text files, documentation files, and log files. Some of these files may be large or become quite large as they accumulate data with time. These files will require both viewing and administrative updating. In this section, you will learn how to manage such large files.

For example, a banking system might maintain one simple large log file to record details of all of one day's ATM transactions. Due to a security attack or a malfunction, the administrator might be forced to check for some data by navigating within the file. In such cases, directly opening the file in an editor

will cause issues, due to high memory utilization, as an editor will usually try to read the whole file into memory first. However, one can use **less** to view the contents of such a large file, scrolling up and down page by page without the system having to place the entire file in memory before starting. This is much faster than using a text editor.

Viewing the file can be done by typing either of the two following commands:

$ less <filename>
$ cat <filename> | less

By default, manual (i.e., the **man** command) pages are sent through the **less** command.

head

head reads the first few lines of each named file (10 by default) and displays it on standard output. You can give a different number of lines in an option.

```
[test3@CentOS Desktop]$ head -n 2 newtest
This is Sample File1
This is Sample File2
[test3@CentOS Desktop]$
```

For example, If you want to print the first 5 lines from atmtrans.txt, use the following command:

$ head –n 5 atmtrans.txt

(You can also just say head -5 atmtrans.txt.)

tail

```
[test3@CentOS ~]$ tail -n 15 test1.txt
apache:x:48:48:Apache:/var/www:/sbin/nologin
saslauth:x:498:76:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
pulse:x:497:496:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
vboxadd:x:496:1::/var/run/vboxadd:/bin/false
test3:x:501:501:test3:/home/test3:/bin/bash
user3:x:502:502:user3:/home/user3:/bin/bash
share:x:503:503::/home/share:/bin/bash
rpc:x:32:32:Rpcbind Daemon:/var/cache/rpcbind:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
jerry:x:504:504::/home/jerry:/bin/bash
[test3@CentOS ~]$ tail -f test1.txt
tcpdump:x:72:72::/:/sbin/nologin
vboxadd:x:496:1::/var/run/vboxadd:/bin/false
test3:x:501:501:test3:/home/test3:/bin/bash
user3:x:502:502:user3:/home/user3:/bin/bash
share:x:503:503::/home/share:/bin/bash
rpc:x:32:32:Rpcbind Daemon:/var/cache/rpcbind:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
jerry:x:504:504::/home/jerry:/bin/bash
```

tail prints the last few lines of each named file and displays it on standard output. By default, it displays the last 10 lines. You can give a different number of lines as an option. **tail** is especially useful when you are troubleshooting any issue using log files as you probably want to see the most recent lines of output.

For example, to display the last 15 lines of atmtrans.txt, use the following command:

$ tail -n 15 atmtrans.txt

(You can also just say tail -15 atmtrans.txt.) To continually monitor new output in a growing log file:

$ tail -f atmtrans.txt

This command will continuously display any new lines of output in **atmtrans.txt** as soon as they appear. Thus it enables you to monitor any current activity that is being reported and recorded.

strings

strings is used to extract all printable character strings found in the file or files given as arguments. It is useful in locating human readable content embedded in binary files: for text files one can just use **grep**.

```
[test3@CentOS Desktop]$ strings /bin/cp | grep GPL
License GPLv3+: GNU GPL version 3 or later <http://gnu.
org/licenses/gpl.html>.
[test3@CentOS Desktop]$
```

For example, to search for the string **my_string** in a spreadsheet:
$ strings book1.xls | grep my_string

The z Command Family

When working with compressed files many standard commands can not be used directly. For many commonly-used file and text manipulation programs there is also a version especially designed to work directly with compressed files. These associated utilities have the letter **z** prefixed to their name. For example, we have utility programs such as **zcat**, **zless**, **zdiff,** and **zgrep**.

Here is a table listing some z family commands:

Command	Description
$ zcat compressed-file.txt.gz	To view a compressed file

$ zless <filename>.gz
or
$ zmore <filename>.gz
	To page through a compressed file

$ zgrep -i less test-file.txt.gz	To search inside a compressed file
$ zdiff filename1.txt.gz filename2.txt.gz	To compare two compressed files

Note that if you run **zless** on an uncompressed file, it will still work and ignore the decompression stage. There are also equivalent utility programs for other compression methods besides **gzip**; i.e, we have **bzcat** and **bzless** associated with **bzip2,** and **xzcat** and **xzless** associated with **xz**.

To practice, click the link provided below.

Using head and tail

To practice, click the link provided below.

Using the z Command Family

Which command is used to print strings of printable characters found in files?

strings

Which command is used to view the last few lines of a file?

- ztail
- end
- tail
- head

Labs

Click the link to download a PDF for the Lab activity.

Manipulating Text_Labs

Manipulating Text_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- The **command line** often allows the users to perform tasks more efficiently than the GUI.
- **cat**, short for **concatenate**, is used to read, print and combine files.
- **echo** displays a line of text either on standard output or to place in a file.
- **sed** is a popular **stream editor** often used to filter and perform substitutions on files and text data streams.
- **awk** is a interpreted programming language typically used as a data extraction and reporting tool.
- **sort** is used to sort text files and output streams in either ascending or descending order.

- **uniq** eliminates duplicate entries in a text file.
- **paste** combines fields from different files and can also extract and combine lines from multiple sources.
- **join** combines lines from two files based on a common field. It works only if files share a common field.
- **split** breaks up a large file into equal-sized segments.
- **Regular expressions** are text strings used for **pattern matching**. The pattern can be used to search for a specific location, such as the start or end of a line or a word.
- **grep** searches text files and data streams for patterns and can be used with regular expressions.
- **tr** translates characters, copies standard input to standard output, and handles special characters.

- **tee** accepts saves a copy of standard output to a file while still displaying at the terminal.
- **wc** (**word count**) displays the number of lines, words and characters in a file or group of files.
- **cut** extracts columns from a file.
- **less** views files a page at a time and allows scrolling in both directions.
- **head** displays the first few lines of a file or data stream on standard output. By default it displays 10 lines.
- **tail** displays the last few lines of a file or data stream on standard output. By default it displays 10 lines.
- **strings** extracts printable character strings from binary files.
- The **z** command family is used to read and work with compressed files.

Chapter 14: Printing

Learning Objectives

By the end of this chapter, you should know how to:

- Configure a printer on a Linux machine
- Print documents.
- Manipulate postscript and pdf files using command line utilities.

Section 1: Configuration

Introduction to Printing

To manage printers and print directly from a computer or across a networked environment, you need to know how to configure and install a printer. Printing itself requires software that converts information from the application you are using to a language your printer can understand. The Linux standard for printing software is the **Common UNIX Printing System (CUPS)**.

CUPS Overview

CUPS is the software that is used behind the scenes to print from applications like a web browser or **LibreOffice**. It converts page descriptions produced by your application (put a paragraph here, draw a line there, and so forth) and then sends the information to the printer. It acts as a **print server** for local as well as network printers.

Printers manufactured by different companies may use their own particular print languages and formats. **CUPS** uses a modular printing system which accommodates a wide variety of printers and also processes various data formats. This makes the printing process simpler; you can concentrate more on printing and less on how to print.

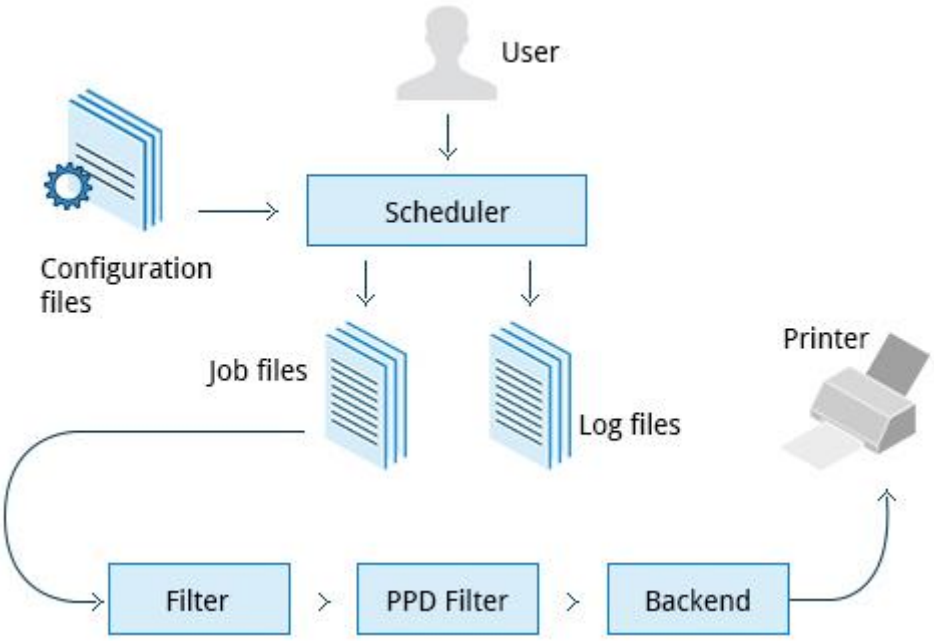
Generally, the only time you should need to configure your printer is when you use it for the first time. In fact, **CUPS** often figures things out on its own by detecting and configuring any printers it locates.

How Does CUPS Work?

CUPS carries out the printing process with the help of its various components:

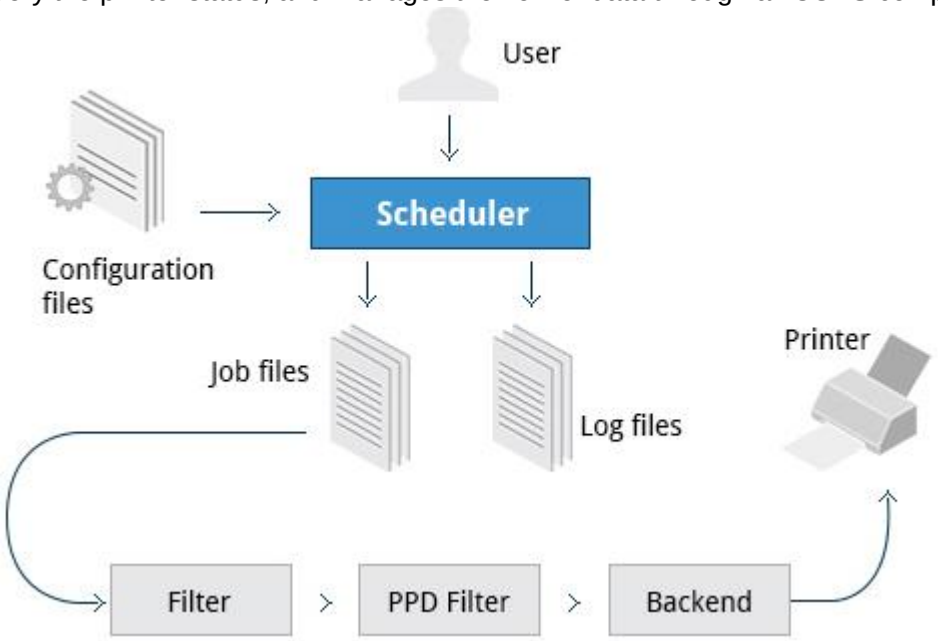
- Configuration Files
- Scheduler
- Job Files
- Log Files
- Filter
- Printer Drivers
- Backend

You will learn about each of these components in detail in the next few screens.



Scheduler

CUPS is designed around a **print scheduler** that manages print jobs, handles administrative commands, allows users to query the printer status, and manages the flow of data through all **CUPS** components.



As you'll see shortly, CUPS has a browser-based interface which allows you to view and manipulate the order and status of pending print jobs.

Configuration Files

The print scheduler reads server settings from several configuration files, the two most important of which are cupsd.conf and printers.conf. These and all other **CUPS** related configuration files are stored under the /etc/cups/ directory.

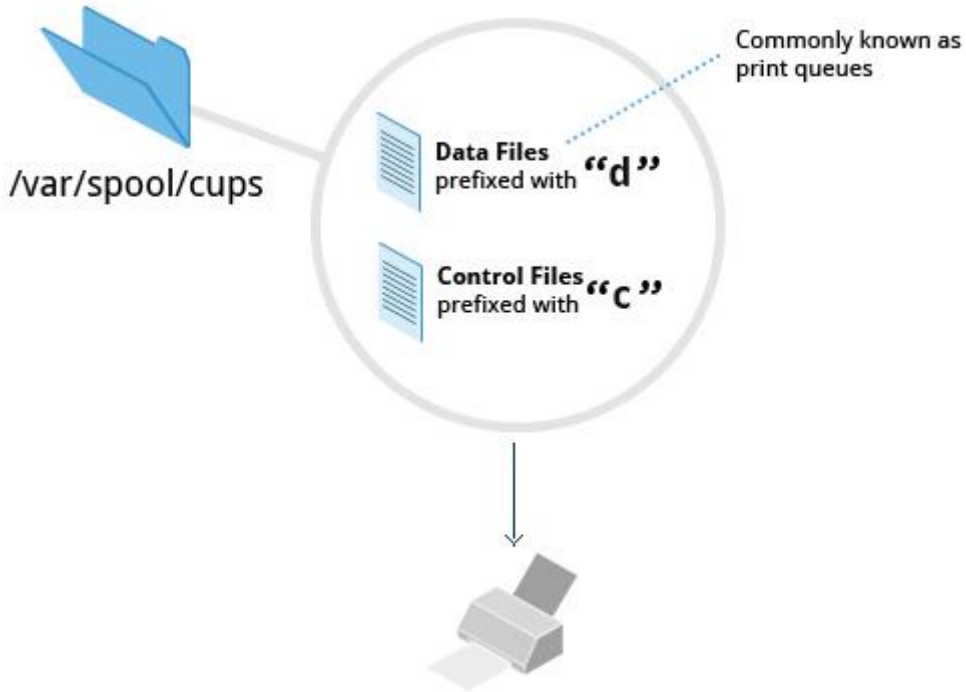
cupsd.conf is where most system-wide settings are located; it does not contain any printer-specific details. Most of the settings available in this file relate to network security, i.e. which systems can access **CUPS** network capabilities, how printers are advertised on the local network, what management features are offered, and so on.

printers.conf is where you will find the printer-specific settings. For every printer connected to the system, a corresponding section describes the printer's status and capabilities. This file is generated only after adding a printer to the system and should not be modified by hand.

You can view the full list of configuration files by typing: ls -l /etc/cups/

```
[test3@CentOS ~]$ ls -l /etc/cups/
total 64
-rw-------. 1 root lp    128 Jun 21 18:05 classes.conf
-rw-------. 1 root lp      0 Aug 17  2013 classes.conf.O
-rw-r--r--. 1 root lp      0 Aug 17  2013 client.conf
-rw-r-----. 1 root lp   3037 Jul 24 19:53 cupsd.conf
-rw-r-----. 1 root lp   4064 Aug 17  2013 cupsd.conf.default
-rw-r-----. 1 root lp   4127 Jun 21 18:02 cupsd.conf.O
drwxr-xr-x. 2 root root 4096 Aug 17  2013 interfaces
-rw-r--r--. 1 root lp     20 Jun 30 21:42 lpoptions
-rw-r--r--. 1 root root  319 Dec  9  2009 paps.convs
drwxr-xr-x. 2 root lp   4096 Jul 24 20:43 ppd
-rw-------. 1 root lp   1604 Jul 24 20:44 printers.conf
-rw-------. 1 root lp   1120 Jul 24 19:58 printers.conf.O
-rw-r--r--. 1 root lp    186 Aug 17  2013 snmp.conf
drwx------. 2 root lp   4096 Aug 17  2013 ssl
-rw-r-----. 1 root lp    394 Jul 24 20:55 subscriptions.conf
-rw-r-----. 1 root lp    621 Jul 24 20:44 subscriptions.conf.O
-rw-r-----. 1 root lp    110 May  9 19:37 subscriptions.conf.rpmsave
[test3@CentOS ~]$
```

Job Files



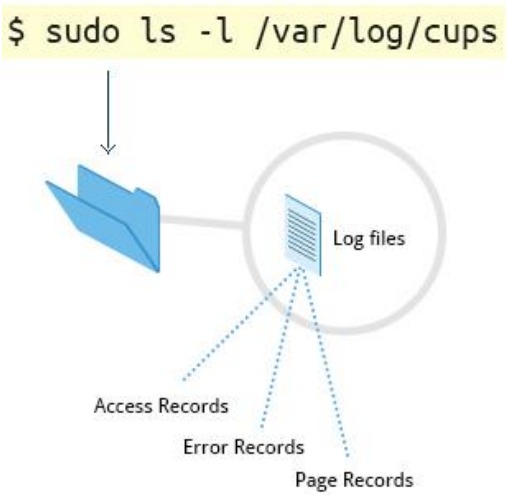
CUPS stores print requests as files under the /var/spool/cups directory (these can actually be accessed before a document is sent to a printer). Data files are prefixed with the letter **d** while control files are prefixed with the letter **c**. After a printer successfully handles a job, data files are automatically removed. These data files belong to what is commonly known as the **print queue**.

Log Files

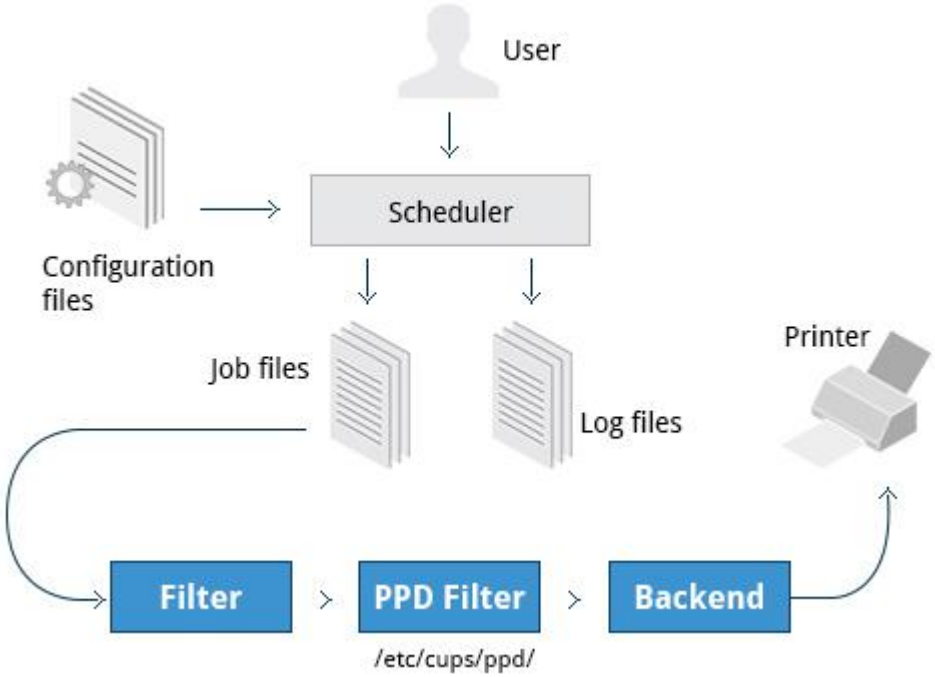
Log files are placed in /var/log/cups and are used by the scheduler to record activities that have taken place. These files include access, error, and page records.

To view what log files exist, type:
sudo ls -l /var/log/cups

(Note on some distributions permissions are set such that you don't need the **sudo**.) You can view the log files with the usual tools.



Filters, Printer Drivers, and Backends



CUPS uses **filters** to convert job file formats to printable formats. Printer **drivers** contain descriptions for currently connected and configured printers, and are usually stored under /etc/cups/ppd/. The print data is then sent to the printer through a filter and via a **backend** that helps to locate devices connected to the system.

So In short, when you execute a print command, the scheduler validates the command and processes the print job creating job files according to the settings specified in configuration files. Simultaneously, the scheduler records activities in the log files. Job files are processed with the help of the filter, printer driver, and backend, and then sent to the printer.

Installing CUPS

Due to printing being a relatively important and fundamental feature of any Linux distribution, most Linux systems come with **CUPS** preinstalled. In some cases, especially for Linux server setups, **CUPS** may have been left uninstalled. This may be fixed by installing the corresponding package manually. To install **CUPS**, please ensure that your system is connected to the Internet.

Demonstration of Installing CUPS

You can use the commands shown below to manually install **CUPS**:

- **CentOS:** $ sudo yum install cups
- **OpenSUSE:** $ sudo zypper install cups
- **Ubuntu:** $ sudo apt-get install cups

The video below demonstrates this procedure for **Ubuntu**, the other two are similar, once the correct install command is provided.

Note: CUPS features are also supported by other packages such as cups-common and libcups2, which contains the core **CUPS** libraries. The above install command will make sure any needed packages are also installed.

Managing CUPS

After installing **CUPS**, you'll need to start and manage the **CUPS** daemon so that **CUPS** is ready for configuring a printer. Managing the **CUPS** daemon is simple; all management features are wrapped around the **cups** init script, which can be easily started, stopped, and restarted.

Managing the CUPS daemon in Ubuntu

Configuring a Printer from the GUI

Each Linux distribution has a GUI application that lets you add, remove, and configure local or remote printers. Using this application, you can easily set up the system to use both local and network printers. The following screens show how to find and use the appropriate application in each of the distribution families covered in this course.

When configuring a printer, make sure the device is currently turned on and connected to the system; if so it should show up in the printer selection menu. If the printer is not visible, you may want to troubleshoot using tools that will determine if the printer is connected. For common USB printers, for example, the **lsusb** utility will show a line for the printer. Some printer manufacturers also require some extra software to be installed in order to make the printer visible to **CUPS**, however, due to the standardization these days, this is rarely required.

Configuring a Printer in Ubuntu

Adding Printers from the CUPS Web Interface

A fact that few people know is that **CUPS** also comes with its own web server, which makes a configuration interface available via a set of CGI scripts.

This web interface allows you to:

- Add and remove local/remote printers
- Configure printers:
  - Local/remote printers
  - Share a printer as a CUPS server
- Control print jobs:
  - Monitor jobs
  - Show completed or pending jobs
  - Cancel or move jobs

The **CUPS** web interface is available on your browser at: http://localhost:631

Some pages require a username and password to perform certain actions, for example to add a printer. For most Linux distributions, you must use the root password to add, modify, or delete printers or classes.

To practice configuring a printer, click the link provided below.

Configuring a Printer

Which two things can **CUPS** do?

- [x] Connect a local printer to a Linux system and share it over a network
- [x] Connect a network printer to a Linux system
- [ ] Configure a network IP to the system
- [ ] Configure a local printer on Windows machine

Printing from the Graphical Interface

Many graphical applications allow users to access printing features using the **CTRL-P** shortcut. To print a file, you first need to specify the printer (or a file name and location if you are printing to a file instead) you want to use; and then select the page setup, quality, and color options. After selecting the required options, you can submit the document for printing. The document is then submitted to **CUPS**. You can use your browser to access the **CUPS** web interface at http://localhost:631/ to monitor the status of the printing job. Now that you have configured the printer, you can print using either the Graphical or Command Line interfaces.

The screenshots below show the GUI interfaces for **CTRL-P** for (from left to right) **CentOS**, **Ubuntu** and **openSUSE**.

Ubuntu printers

Printing from the Command-Line Interface

```
[root@CentOS ~]# lp test1.txt
request id is HP-LaserJet-23 (1 file(s))
[root@CentOS ~]#
```

CUPS provides two command-line interfaces, descended from the **System V** and **BSD** flavors of UNIX. This means that you can use either **lp** (System V) or **lpr** (BSD) to print. You can use these commands to print text, PostScript, PDF, and image files.

These commands are useful in cases where printing operations must be automated (from shell scripts, for instance, which contain multiple commands in one file). You will learn more about the shell scripts in the upcoming chapters on **bash** scripts.

lp is just a command line front-end to the **lpr** utility that passes input to **lpr**. Thus, we will discuss only **lp** in detail. In the example shown here, the task is to print the file called test1.txt.

Using lp

lp and **lpr** accept command line options that help you perform all operations that the GUI can accomplish. **lp** is typically used with a file name as an argument.

Some **lp** commands and other printing utilities you can use are listed in the table.

Command	Usage
lp <filename>	To print the file to default printer
lp -d printer <filename>	To print to a specific printer (useful if multiple printers are available)
program | lp echo string | lp	To print the output of a program
lp -n number <filename>	To print multiple copies
lpoptions -d printer	To set the default printer
lpq -a	To show the queue status
lpadmin	To configure printer queues

The **lpoptions** utility can be used to set printer options and defaults. Each printer has a set of **tags** associated with it, such as the default number of copies and authentication requirements. You can execute the command lpoptions help to obtain a list of supported options. **lpoptions** can also be used to set system-wide values, such as the default printer.

Printing Using lp

To practice, click the link provided below. **Printing With the lp Command**

Managing Print Jobs

You send a file to the shared printer. But when you go there to collect the printout, you discover another user has just started a 200 page job that is not time sensitive. Your file cannot be printed until this print job is complete. What do you do now?

In Linux, command line print job management commands allow you to monitor the job state as well as managing the listing of all printers and checking their status, and cancelling or moving print jobs to another printer.

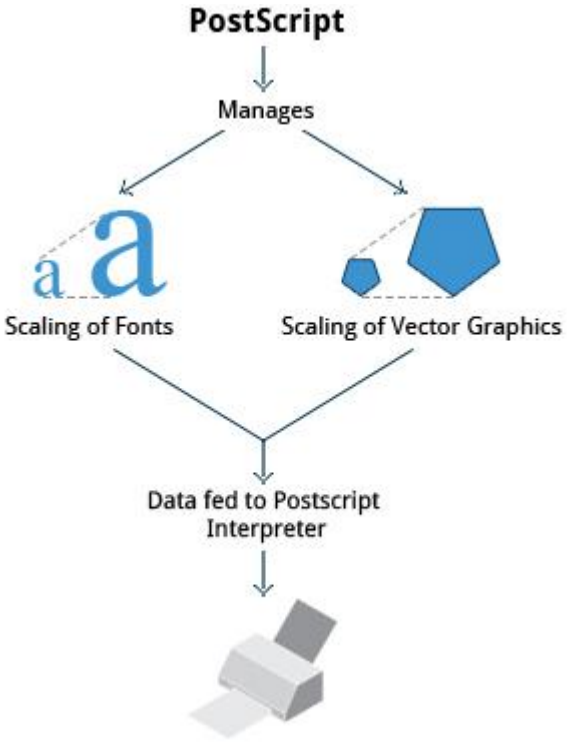
Some of these commands are listed in the table.

Command	Usage
lpstat -p -d	To get a list of available printers, along with their status
lpstat -a	To check the status of all connected printers, including job numbers
cancel job-id OR lprm job-id	To cancel a print job
lpmove job-id newprinter	To move a print job to new printer

To practice, click the link provided below.

Managing Print Jobs

Section 3: Manipulating Postscript and PDF files



Working with PostScript

PostScript is a standard **page description language.** It effectively manages scaling of fonts and vector graphics to provide quality printouts. It is purely a text format that contains the data fed to a PostScript interpreter. The format itself is a language that was developed by **Adobe** in the early 1980s to enable the transfer of data to printers.

Features of PostScript are:

- It can be used on any printer that is PostScript-compatible; i.e., any modern printer
- Any program that understands the PostScript specification can print to it
- Information about page appearance, etc. is embedded in the page

Working with enscript

enscript is a tool that is used to convert a text file to PostScript and other formats. It also supports **Rich Text Format (RTF)** and **HyperText Markup Language (HTML)**. For example, you can convert a text file to two column (-2) formatted **PostScript** using the command: enscript -2 -r -p psfile.ps textfile.txt. This command will also rotate (-r) the output to print so the width of the paper is greater than the height (aka landscape mode) thereby reducing the number of pages required for printing.

The commands that can be used with **enscript** are listed in the table below (for a file called 'textfile.txt').

Command	Usage
enscript -p psfile.ps textfile.txt	Convert a text file to PostScript (saved to psfile.ps)
enscript -n -p psfile.ps textfile.txt	Convert a text file to n columns where n=1-9 (saved in psfile.ps)
enscript textfile.txt	Print a text file directly to the default printer

Viewing PDF Content

Linux has many standard programs that can read PDF files as well as many applications that can easily create them, including all available office suites such as **LibreOffice.**

The most common Linux PDF readers are:

1. **Evince** is available on virtually all distributions and the most widely used program.
2. **Okular** is based on the older **kpdf** and available on any distribution that provides the **KDE** environment.
3. **GhostView** is one of the first open source PDF readers and is universally available.
4. **Xpdf** is one of the oldest open source PDF readers and still has a good user base.

All of these open source PDF readers support and can read files following the PostScript standard unlike the proprietary **Adobe Acrobat Reader**, which was once widely used on Linux systems but with the growth of these excellent programs, very few Linux users use it today.

Modifying PDFs with pdftk

At times, you may want to merge, split, or rotate PDF files; not all of these operations can be achieved while using a PDF viewer. A great way to do this is to use the "PDF Toolkit", **pdftk**, to perform a very large variety of sophisticated tasks. Some of these operations include:

- Merging/Splitting/Rotating PDF documents
- Repairing corrupted PDF pages
- Pulling single pages from a file
- Encrypting and decrypting PDF files
- Adding, updating, and exporting a PDF's **metadata**
- Exporting bookmarks to a text file
- Filling out PDF forms

In short, there's very little **pdftk** cannot do when it comes to working with PDF files; it is indeed the Swiss Army knife of PDF tools.

Installing pdftk on Different Family Systems

To install **pdftk** on **Ubuntu**, use the following command:

```
$ sudo apt-get install pdftk
```

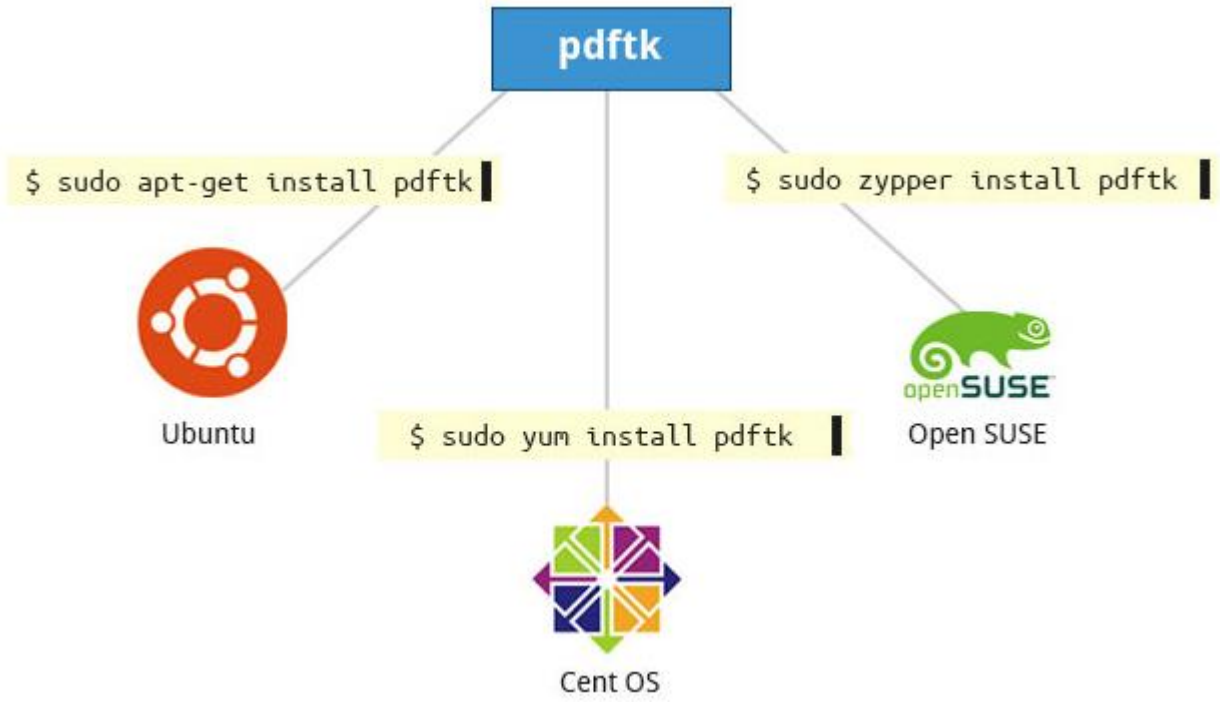
On **CentOS:**

```
$ sudo yum install pdftk
```

On **openSUSE**:

```
$ sudo zypper install pdftk
```

You may find that **CentOS** (and **RHEL**) don't have **pdftk** in their packaging system, but you can obtain the PDF Toolkit directly from the PDF Lab's website by downloading from:
http://www.pdflabs.com/docs/install-pdftk-on-redhat-or-centos/



Using pdftk

You can accomplish a wide variety of tasks using **pdftk** including:

Command	Usage
pdftk 1.pdf 2.pdf cat output 12.pdf	Merge the two documents 1.pdf and 2.pdf. The output will be saved to 12.pdf.
pdftk A=1.pdf cat A1-2 output new.pdf	Write only pages 1 and 2 of 1.pdf. The output will be saved to new.pdf.
pdftk A=1.pdf cat A1-endright output new.pdf	Rotate all pages of 1.pdf 90 degrees clockwise and save result in new.pdf.

Encrypting PDF Files

If you're working with PDF files that contain confidential information and you want to ensure that only certain people can view the PDF file, you can apply a password to it using the user_pw option. One can do this by issuing a command such as:

```
[test3@CentOS ~]$ pdftk public.pdf output private.pdf u
ser_pw PROMPT
Please enter the user password to use on the output PDF
.
   It can be empty, or have a maximum of 32 characters:
test3
[test3@CentOS ~]$
```

$ pdftk public.pdf output private.pdf user_pw PROMPT

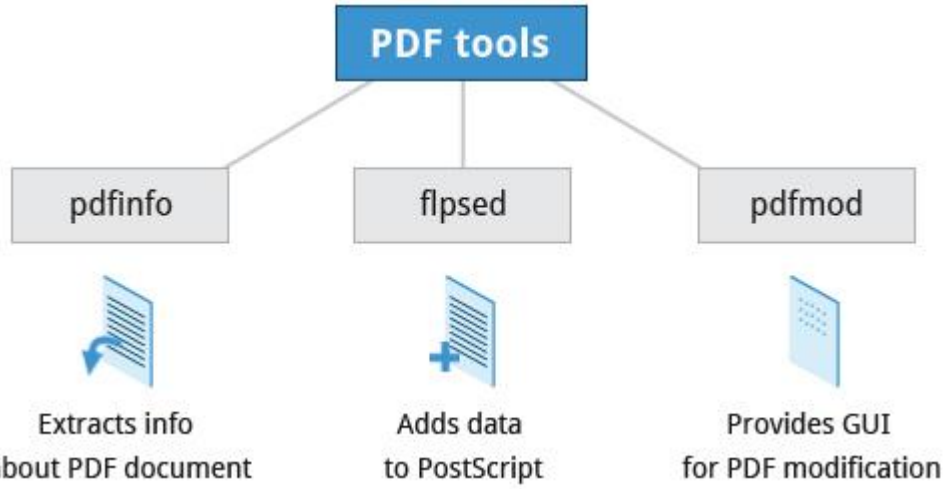
When you run this command, you will receive a prompt to set the required password, which can have a maximum of 32 characters. A new file, private.pdf, will be created with the identical content as public.pdf, but anyone will need to type the password to be able to view it.

Using pdftk

Using Additional Tools

You can use other tools, such as **pdfinfo**, **flpsed**, and **pdfmod** to work with PDF files.

pdfinfo can extract information about PDF files, especially when the files are very large or when a graphical interface is not available.



flpsed can add data to a PostScript document. This tool is specifically useful for filling in forms or adding short comments into the document.

pdfmod is a simple application that provides a graphical interface for modifying PDF documents. Using this tool, you can reorder, rotate, and remove pages; export images from a document; edit the title, subject, and author; add keywords; and combine documents using drag-and-drop action.

For example, to collect the details of a document, you can use the following command:
$ pdfinfo /usr/share/doc/readme.pdf

Converting Between PostScript and PDF

Most users today are far more accustomed to working with files in PDF format, viewing them easily either on the Internet through their browser or locally on their machine. The PostScript format is still important for various technical reasons that the general user will rarely have to deal with.

From time to time you may need to convert files from one format to the other, and there are very simple utilities for accomplishing that task. **ps2pdf** and **pdf2ps** are part of the **ghostscript** package installed on or available on all Linux distributions. As an alternative, there are **pstopdf** and **pdftops** which are usually part of the **poppler** package which may need to be added through your package manager. Unless you are doing a lot of conversions or need some of the fancier options (which you can read about in the **man pages** for these utilities) it really doesn't matter which ones you use.

Some usage examples:

Command	Usage
pdf2ps file.pdf	Converts file.pdf to file.ps
ps2pdf file.ps	Convertsfile.ps to file.pdf
pstopdf input.ps output.pdf	Converts input.ps to output.pdf
pdftops input.pdf output.pdf	Converts input.pdf to output.ps

Labs

Click the link to download a PDF from the Lab activity.

Printing_Labs

Printing_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- **CUPS** provides two command-line interfaces: the **System V** and **BSD** interfaces.
- The CUPS interface is available at http://localhost:631
- **lp** and **lpr** are used to submit a document to **CUPS** directly from the command line.
- **lpoptions** can be used to set printer options and defaults.
- PostScript effectively manages scaling of fonts and vector graphics to provide quality prints.
- **enscript** is used to convert a text file to PostScript and other formats.

- **Portable Document Format (PDF)** is the standard format used to exchange documents while ensuring a certain level of consistency in the way the documents are viewed.
- **pdftk** joins and splits PDFs; pulls single pages from a file; encrypts and decrypts PDF files; adds, updates, and exports a PDF's metadata; exports bookmarks to a text file; adds or removes attachments to a PDF; fixes a damaged PDF; and fills out PDF forms.
- **pdfinfo** can extract information about PDF documents.
- **flpsed** can add data to a PostScript document.
- **pdfmod** is a simple application with a graphical interface that you can use to modify PDF documents.

Chapter 15: Bash Shell Scripting

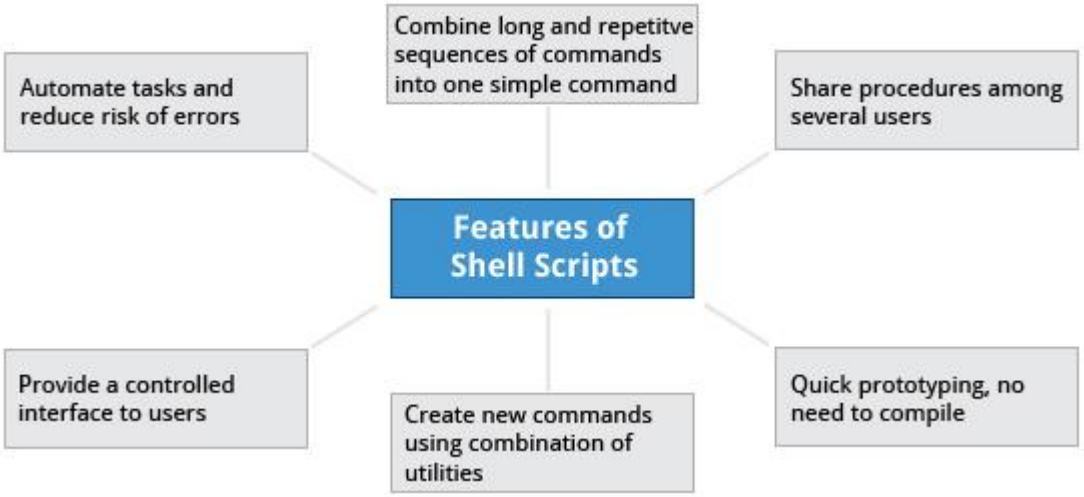
Learning Objectives

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

- Explain the features and capabilities of **bash shell scripting**.
- Know the basic syntax of scripting statements.
- Be familiar with various methods and constructs used.
- Know how to test for properties and existence of files and other objects.
- Use conditional statements, such as **if-then-else** blocks.
- Perform arithmetic operations using scripting language.

Section 1: Features and Capabilities

Introduction to Scripts



Suppose you want to look up a filename, check if the associated file exists, and then respond accordingly, displaying a message confirming or not confirming the file's existence. If you only need to do it once, you can just type a sequence of commands at a terminal. However, if you need to do this multiple times, automation is the way to go. In order to automate sets of commands you'll need to learn how to write **shell scripts**, the most common of which are used with **bash**. The graphic illustrates several of the benefits of deploying scripts.

Introduction to Shell Scripts

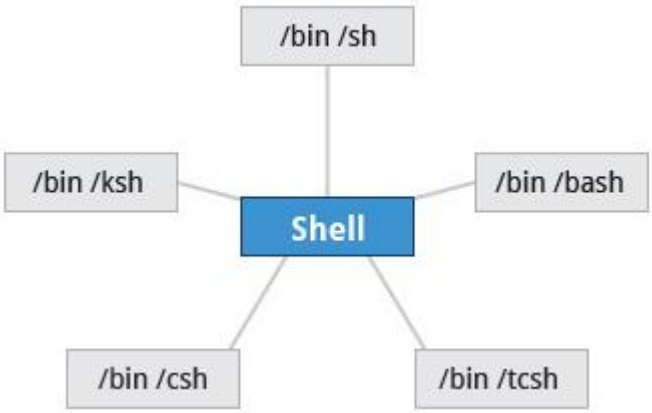
Remember from our earlier discussion, a **shell** is a command line **interpreter** which provides the user interface for terminal windows. It can also be used to run scripts, even in non-interactive sessions without a terminal window, as if the commands were being directly typed in. For example typing: find . -name "*.c" -ls at the command line accomplishes the same thing as executing a script file containing the lines:

```
#!/bin/bash
find . -name "*.c" -ls
```

The #!/bin/bash in the first line should be recognized by anyone who has developed any kind of script in UNIX environments. The first line of the script, that starts with #!, contains the full path of the command interpreter (in this case /bin/bash) that is to be used on the file. As we will see on the next screen, you have a few choices depending upon which scripting language you use.

```
[test3@CentOS ~]$ find . -name "*.c" -ls
163017   20 -rw-rw-r--   1 1000     1000        17890 Jun  3 12:00 ./hplip-3.14.6/prnt/hpcups/jccolo
r.c
163068    8 -rw-rw-r--   1 1000     1000         5925 Jun  3 12:00 ./hplip-3.14.6/prnt/hpcups/jdatad
bf.c
162870   20 -rw-r--r--   1 1000     1000        18573 Jun  3 12:03 ./hplip-3.14.6/prnt/hpijs/jccolor
.c
162850   20 -rw-rw-r--   1 1000     1000        20102 Jun  3 12:01 ./hplip-3.14.6/prnt/hpijs/ijs_ser
ver.c
162939   20 -rw-rw-r--   1 1000     1000        16889 Jun  3 12:01 ./hplip-3.14.6/prnt/hpijs/hpiom.c
162963    8 -rw-r--r--   1 1000     1000         6074 Jun  3 12:03 ./hplip-3.14.6/prnt/hpijs/jdatadb
f.c
162966    8 -rw-rw-r--   1 1000     1000         4619 Jun  3 12:01 ./hplip-3.14.6/prnt/hpijs/ijs.c
163082   60 -rw-rw-r--   1 1000     1000        57952 Jun  3 12:00 ./hplip-3.14.6/prnt/cupsext/cupse
xt.c
163078   28 -rw-rw-r--   1 1000     1000        27939 Jun  3 12:00 ./hplip-3.14.6/prnt/backend/hp.c
162557   24 -rwxrwxr-x   1 1000     1000        22811 Jun  3 12:01 ./hplip-3.14.6/scan/scanext/scane
xt.c
162564   40 -rw-rw-r--   1 1000     1000        37077 Jun  3 12:02 ./hplip-3.14.6/scan/sane/bb_ledm.
c
162565    4 -rwxrwxr-x   1 1000     1000         3960 Jun  3 12:02 ./hplip-3.14.6/scan/sane/io.c
162560   20 -rwxrwxr-x   1 1000     1000        17188 Jun  3 12:02 ./hplip-3.14.6/scan/sane/mfpdtf.c
162581    4 -rw-rw-r--   1 1000     1000         3771 Jun  3 12:02 ./hplip-3.14.6/scan/sane/xml.c
162582   40 -rw-rw-r--   1 1000     1000        37140 Jun  3 12:02 ./hplip-3.14.6/scan/sane/marvell.
c
162573  112 -rw-rw-r--   1 1000     1000       112343 Jun  3 12:02 ./hplip-3.14.6/scan/sane/sclpml.c
162569    4 -rw-rw-r--   1 1000     1000         3038 Jun  3 12:02 ./hplip-3.14.6/scan/sane/sanei_in
it_debug.c
162593   40 -rw-rw-r--   1 1000     1000        37375 Jun  3 12:02 ./hplip-3.14.6/scan/sane/ledm.c
162587    8 -rwxrwxr-x   1 1000     1000         8037 Jun  3 12:02 ./hplip-3.14.6/scan/sane/common.c
162580   20 -rw-rw-r--   1 1000     1000        17685 Jun  3 12:02 ./hplip-3.14.6/scan/sane/http.c
162559   12 -rwxrwxr-x   1 1000     1000        10262 Jun  3 12:02 ./hplip-3.14.6/scan/sane/scl.c
162568   32 -rwxrwxr-x   1 1000     1000        31946 Jun  3 12:02 ./hplip-3.14.6/scan/sane/pml.c
162567   16 -rw-rw-r--   1 1000     1000        15926 Jun  3 12:02 ./hplip-3.14.6/scan/sane/hpaio.c
162595   40 -rw-rw-r--   1 1000     1000        37252 Jun  3 12:02 ./hplip-3.14.6/scan/sane/soap.c
```

Command Shell Choices



The command **interpreter** is tasked with executing statements that follow it in the script. Commonly used interpreters include: /usr/bin/perl, /bin/bash, /bin/csh, /usr/bin/python and /bin/sh.

Typing a long sequence of commands at a terminal window can be complicated, time consuming, and error prone. By deploying shell scripts, using the command-line becomes an efficient and quick way to launch complex sequences of steps. The fact that shell scripts are saved in a file also makes it easy to use them to create new script variations and share standard procedures with several users.

Linux provides a wide choice of shells; exactly what is available on the system is listed in /etc/shells. Typical choices are:

/bin/sh
/bin/bash
/bin/tcsh
/bin/csh
/bin/ksh

Most Linux users use the default **bash** shell, but those with long UNIX backgrounds with other shells may want to override the default.

Click the link to download UNIX Shell.pdf to read more about the UNIX Shell.

bash Scripts

```
[test3@CentOS Desktop]$ cat > exscript.sh
#!/bin/bash
echo "HELLO"
echo "WORLD"
[test3@CentOS Desktop]$ bash exscript.sh
HELLO
WORLD
[test3@CentOS Desktop]$
```

Let's write a simple **bash** script that displays a two-line message on the screen. Either type

```
$ cat > exscript.sh
  #!/bin/bash
  echo "HELLO"
  echo "WORLD"
```

and press **ENTER** and **CTRL-D** to save the file, or just create exscript.sh in your favorite text editor. Then, type chmod +x exscript.sh to make the file executable. (The chmod +x command makes the file executable for all users.) You can then run it by simply typing ./exscript.sh or by doing:

```
$ bash exscript.sh
  HELLO
  WORLD
```

Note if you use the second form, you don't have to make the file executable.

Interactive Example Using bash Scripts

```
[test3@CentOS ~]$ cat ioscript.sh
#!/bin/bash
# Interactive reading of variables
echo "ENTER YOUR NAME"
read sname
# Display of variable values
echo $sname
[test3@CentOS ~]$ chmod +x ioscript.sh
[test3@CentOS ~]$ ./ioscript.sh
ENTER YOUR NAME
Alice
Alice
[test3@CentOS ~]$
```

Now, let's see how to create a more interactive example using a **bash** script. The user will be prompted to enter a value, which is then displayed on the screen. The value is stored in a temporary variable, sname. We can reference the value of a shell variable by using a $ in front of the variable name, such as $sname. To create this script, you need to create a file named ioscript.sh in your favorite editor with the following content:

```
  #!/bin/bash
  # Interactive reading of variables
  echo "ENTER YOUR NAME"
  read sname
  # Display of variable values
  echo $sname
```

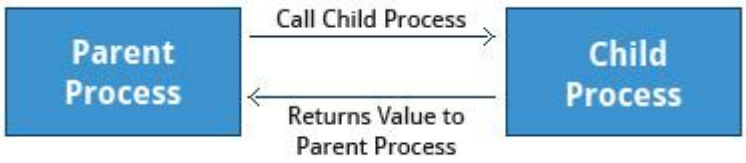
Once again, make it executable by doing chmod +x ioscript.sh.

In the above example, when the script ./ioscript.sh is executed, the user will receive a prompt ENTER YOUR NAME. The user then needs to enter a value and press the **Enter** key. The value will then be printed out.

Additional note: The hash-tag/pound-sign/number-sign (#) is used to start comments in the script and can be placed anywhere in the line (the rest of the line is considered a comment).

Return Values

All shell scripts generate a **return value** upon finishing execution; the value can be set with the exit statement. Return values permit a process to monitor the exit state of another process often in a parent-child relationship. This helps to determine how this process terminated and take any appropriate steps necessary, contingent on success or failure.



Viewing Return Values

```
[test3@CentOS ~]$ ls /etc/passwd
/etc/passwd
[test3@CentOS ~]$ echo $?
0
[test3@CentOS ~]$ ls filethatdoesnotexist
ls: cannot access filethatdoesnotexist: No such file or directory
[test3@CentOS ~]$ echo $?
2
[test3@CentOS ~]$
```

As a script executes, one can check for a specific value or condition and return success or failure as the result. By convention, success is returned as 0, and failure is returned as a non-zero value. An easy way to demonstrate success and failure completion is to execute **ls** on a file that exists and one that doesn't, as shown in the following example, where the return value is stored in the environment variable represented by $?:

$ ls /etc/passwd
/etc/ passwd

$ echo $?
0

In this example, the system is able to locate the file /etc/passwd and returns a value of 0 to indicate success; the return value is always stored in the $? environment variable. Applications often translate these return values into meaningful messages easily understood by the user.

Which command is used to make the script, script1.sh, executable?

- chmod a script1.sh
- chmod +x script1.sh
- ./script1.sh command
- cat ./script1.sh

Which of the following commands is used to manually pass a text file, script.sh, to the shell interpreter?

- bash script.sh
- script.sh bash
- ./script.sh
- .sh ./script

Section 2: Syntax

Basic Syntax and Special Characters

Scripts require you to follow a standard language **syntax**. Rules delineate how to define variables and how to construct and format allowed statements, etc. The table lists some special character usages within **bash** scripts:

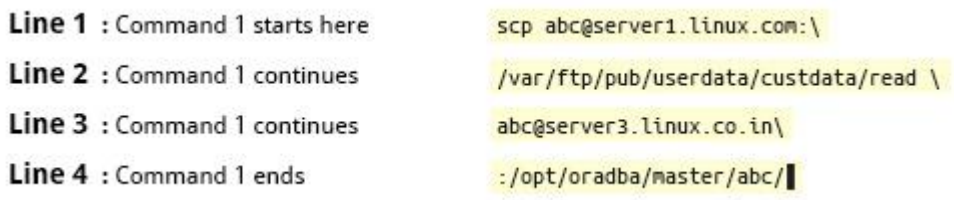
Character	Description
#	Used to add a comment, **except** when used as \#, or as #! when starting a script
\	Used at the end of a line to indicate continuation on to the next line
;	Used to interpret what follows as a new command
$	Indicates what follows is a variable

Note that when # is inserted at the beginning of a line of commentary, the whole line is ignored.

```
# This line will not get executed.
```

Splitting Long Commands Over Multiple Lines

Users sometimes need to combine several commands and statements and even conditionally execute them based on the behaviour of operators used in between them. This method is called **chaining of commands**.



The **concatenation operator** (****) is used to concatenate large commands over several lines in the shell.

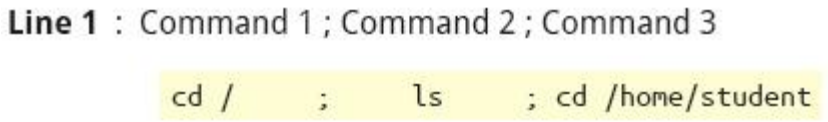
For example, you want to copy the file **/var/ftp/pub/userdata/custdata/read** from **server1.linux.com** to the **/opt/oradba/master/abc** directory on **server3.linux.co.in**. To perform this action, you can write the command using the \ operator as:

```
scp abc@server1.linux.com:\
/var/ftp/pub/userdata/custdata/read \
abc@server3.linux.co.in:\
/opt/oradba/master/abc/
```

The command is divided into multiple lines to make it look readable and easier to understand. The \ operator at the end of each line combines the commands from multiple lines and executes it as one single command.

Putting Multiple Commands on a Single Line

Sometimes you may want to group multiple commands on a single line. The ; (semicolon) character is used to separate these commands and execute them sequentially as if they had been typed on separate lines.



The three commands in the following example will all execute even if the ones preceding them fail:

```
$ make ; make install ; make clean
```

However, you may want to abort subsequent commands if one fails. You can do this using the && (and) operator as in:

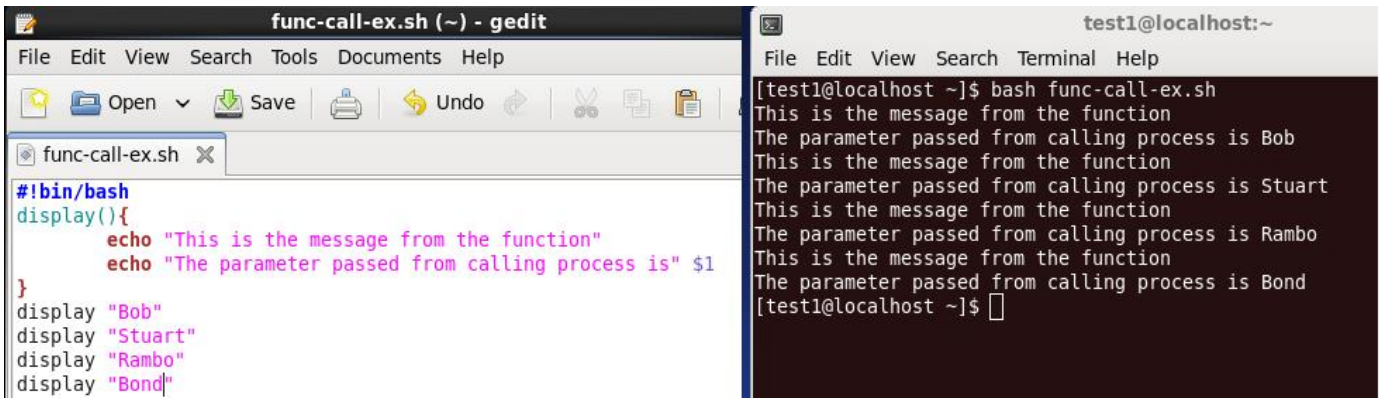
```
$ make && make install && make clean
```

If the first command fails the second one will never be executed. A final refinement is to use the || (or) operator as in:

```
$ cat file1 || cat file2 || cat file3
```

In this case, you proceed until something succeeds and then you stop executing any further steps.

Functions



A **function** is a code block that implements a set of operations. Functions are useful for executing procedures multiple times perhaps with varying input variables. Functions are also often called **subroutines.** Using functions in scripts requires two steps:

1. Declaring a function
2. Calling a function

The function declaration requires a name which is used to invoke it. The proper syntax is:

```
function_name () {
  command...
}
```

For example, the following function is named display:

```
display () {
  echo "This is a sample function"
}
```

The function can be as long as desired and have many statements. Once defined, the function can be called later as many times as necessary. In the full example shown in the figure, we are also showing an often-used refinement: how to pass an argument to the function. The first argument can be referred to as $1, the second as $2, etc.

Built-in Shell Commands

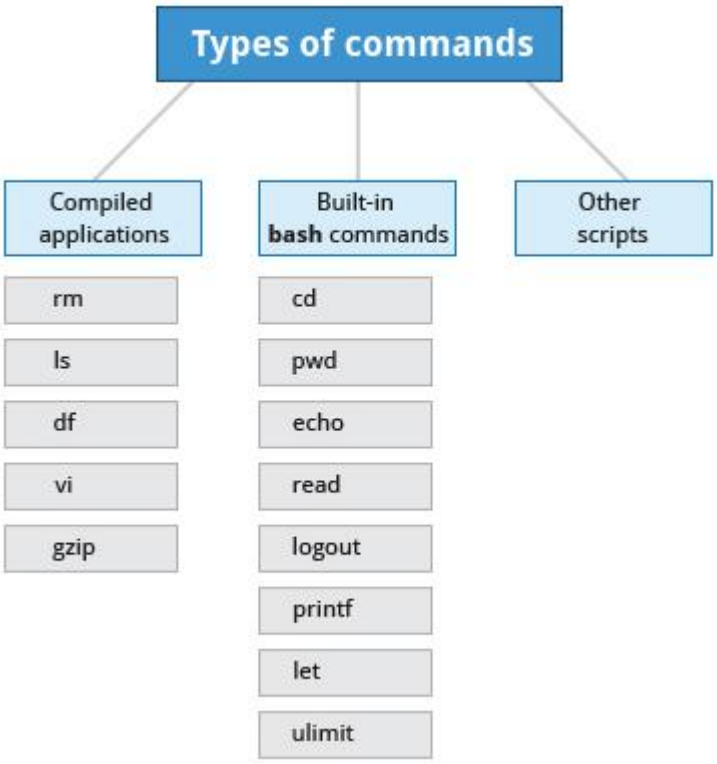
Shell scripts are used to execute sequences of commands and other types of statements. Commands can be divided into the following categories:

- Compiled applications
- Built-in **bash** commands
- Other scripts

Compiled applications are binary executable files that you can find on the filesystem. The shell script always has access to compiled applications such as **rm**, **ls**, **df**, **vi**, and **gzip**.

bash has many **built-in** commands which can only be used to display the output within a terminal shell or shell script. Sometimes these commands have the same name as executable programs on the system, such as **echo** which can lead to subtle problems. **bash** built-in commands include and cd, pwd, echo, read, logout, printf, let, and ulimit.

A complete list of **bash** built-in commands can be found in the **bash man** page, or by simply typing help.



Command Substitution

```
[test3@CentOS ~]$ uname -r
2.6.32-431.20.5.el6.x86_64
[test3@CentOS ~]$ echo cd /lib/modules/`uname -r`/
cd /lib/modules/2.6.32-431.20.5.el6.x86_64/
[test3@CentOS ~]$ echo cd /lib/modules/$(uname -r)/
cd /lib/modules/2.6.32-431.20.5.el6.x86_64/
[test3@CentOS ~]$ pwd
/home/test3
[test3@CentOS ~]$ $(echo cd /lib/modules/$(uname -r)/)
[test3@CentOS 2.6.32-431.20.5.el6.x86_64]$ pwd
/lib/modules/2.6.32-431.20.5.el6.x86_64
[test3@CentOS 2.6.32-431.20.5.el6.x86_64]$
```

At times, you may need to substitute the result of a command as a portion of another command. It can be done in two ways:

- By enclosing the inner command with backticks (`)
- By enclosing the inner command in $( )

No matter the method, the innermost command will be executed in a newly launched shell environment, and the standard output of the shell will be inserted where the command substitution was done.

Virtually any command can be executed this way. Both of these methods enable command substitution; however, the $( ) method allows command nesting. New scripts should always use this more modern method. For example:

$ cd /lib/modules/$(uname -r)/

In the above example, the output of the command "uname –r" becomes the argument for the cd command.

Environment Variables

Almost all scripts use **variables** containing a value, which can be used anywhere in the script. These variables can either be user or system defined. Many applications use such **environment variables** (covered in the "User Environment" chapter) for supplying inputs, validation, and controlling behaviour.

Some examples of standard environment variables are HOME, PATH, and HOST. When referenced, environment variables must be prefixed with the $ symbol as in $HOME. You can view and set the value of environment variables. For example, the following command displays the value stored in the PATH variable:

```
$ echo $PATH
```

However, no prefix is required when setting or modifying the variable value. For example, the following command sets the value of the MYCOLOR variable to blue:

```
$ MYCOLOR=blue
```

You can get a list of environment variables with the **env**, **set**, or **printenv** commands.

Exporting Variables

```
[test3@CentOS ~]$ VERSION=$(uname -r); export VERSION
[test3@CentOS ~]$ export
declare -x COLORTERM="gnome-terminal"
declare -x CVS_RSH="ssh"
declare -x DBUS_SESSION_BUS_ADDRESS="unix:abstract=/tmp/dbus-FNOEnF3C4H,guid=b9f778e49a223a9b58cf1f1e00000018"
declare -x DESKTOP_SESSION="gnome"
declare -x DISPLAY=":0.0"
declare -x GDMSESSION="gnome"
declare -x GDM_KEYBOARD_LAYOUT="us"
declare -x GDM_LANG="en_US.UTF-8"
declare -x GNOME_DESKTOP_SESSION_ID="this-is-deprecated"
declare -x GNOME_KEYRING_PID="2259"
declare -x GNOME_KEYRING_SOCKET="/tmp/keyring-NpgxUc/socket"
declare -x GTK_RC_FILES="/etc/gtk/gtkrc:/home/test3/.gtkrc-1.2-gnome2"
declare -x G_BROKEN_FILENAMES="1"
        .
        .
        .
        .
        .
        .
declare -x VERSION="2.6.32-431.20.3.el6.x86_64"
declare -x WINDOWID="71303172"
declare -x WINDOWPATH="1"
declare -x XAUTHORITY="/var/run/gdm/auth-for-test3-lrdS7v/database"
declare -x XDG_SESSION_COOKIE="f2c88ffc5eeba3db045d1f7c0000000d-1404798425.42575-2021681626"
declare -x XMODIFIERS="@im=none"
[test3@CentOS ~]$
```

By default, the variables created within a script are available only to the subsequent steps of that script. Any child processes (sub-shells) do not have automatic access to the values of these variables. To make them available to child processes, they must be promoted to environment variables using the **export** statement as in:

```
export VAR=value
```

or

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

While child processes are allowed to modify the value of exported variables, the parent will not see any changes; exported variables are not shared, but only copied.

Script Parameters

Users often need to pass parameter values to a script, such as a filename, date, etc. Scripts will take different paths or arrive at different values according to the parameters (command arguments) that are passed to them. These values can be text or numbers as in:

$ ./script.sh /tmp
$ ./script.sh 100 200

Within a script, the parameter or an argument is represented with a $ and a number. The table lists some of these parameters.

Parameter	Meaning
$0	Script name
$1	First parameter
$2, $3, etc.	Second, third parameter, etc.
$*	All parameters
$#	Number of arguments

Script Parameters

Using Script Parameters

```
[test3@CentOS ~]$ vi script3.sh
[test3@CentOS ~]$ bash script3.sh one two three
The name of this program is: script3.sh
The first argument passed from the command line is: one
The second argument passed from the command line is: two
The third argument passed from the command line is: three
All of the arguments passed from the command line are : one two three

All done with script3.sh
[test3@CentOS ~]$
```

Using your favorite text editor, create a new script file named **script3.sh** with the following contents:

```
#!/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 The third argument passed from the command line is: $3
echo All of the arguments passed from the command line are : $*
echo
echo All done with $0
```

Make the script executable with **chmod +x**. Run the script giving it three arguments as in: script3.sh one two three, and the script is processed as follows:

$0 prints the script name: script3.sh
$1 prints the first parameter: one
$2 prints the second parameter: two
$3 prints the third parameter: three
$* prints all parameters: one two three
The final statement becomes: All done with script3.sh

Output Redirection

```
[root@CentOS ~]# free>/tmp/free.out
[root@CentOS ~]# ls -l /tmp/free.out
-rw-rw-r--. 1 root root 230 Jul  7 12:49 /tmp/free.out
[root@CentOS ~]# cat /tmp/free.out
             total       used       free     shared    buffers     cached
Mem:       1020176     922100      98076          0      66608     473240
-/+ buffers/cache:     382252     637924
Swap:      1219576          0    1219576
[root@CentOS ~]#
```

Most operating systems accept input from the keyboard and display the output on the terminal. However, in shell scripting you can send the output to a file. The process of diverting the output to a file is called output **redirection**.

The > character is used to write output to a file. For example, the following command sends the output of **free** to the file /tmp/free.out:

$ free > /tmp/free.out

To check the contents of the /tmp/free.out file, at the command prompt type cat /tmp/free.out.

Two > characters (>>) will append output to a file if it exists, and act just like > if the file does not already exist.

Input Redirection

```
[kcs@kcs temp]$ ./script8.sh
“Line count”
4
[kcs@kcs temp]$
```

Just as the output can be redirected **to** a file, the input of a command can be read **from** a file. The process of reading input from a file is called input redirection and uses the < character. If you create a file called script8.sh with the following contents:

```
#!/bin/bash
echo “Line count”
wc -l < /temp/free.out
```

and then execute it with chmod +x script8.sh ; ./script8.sh, it will count the number of lines from the **/temp/free.out** file and display the results.

Which character is used to represent the continuation of a command over several lines?

- /
- |
- \
- -

What are the different types of commands that can be used in a bash shell script?

- Compiled applications
- Built-in **bash** commands
- Other scripts
- All of the above

If you want to substitute the result of command1 to be the argument for cd, how will you represent them?

- [x] cd `$command1`
- [ ] cd ($command2)
- [ ] cd $command2
- [x] cd $(command1)

Which of the following are the correct syntax to declare the **showpath** function?

- [ ] showpath [echo $PATH]
- [ ] showpath {echo $PATH}
- [x] showpath () {echo $PATH}
- [ ] showpath (echo PATH)

Which of the following parameters contains the name of the script being executed?

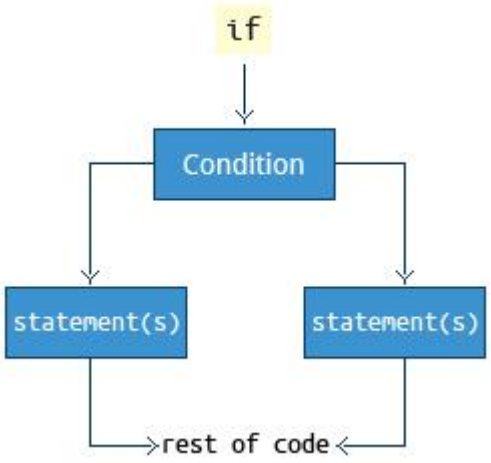
- ( ) $n
- ( ) var
- (•) $0
- ( ) $*

What commands are used to write the output of the free command to the file **/tmp/free.out**?

- [x] free >> /tmp/free.out
- [ ] free < /tmp/free.out
- [x] free > /tmp/free.out
- [ ] free <> /tmp/free.out

Section 3: Constructs

The if Statement



Conditional decision making using an if statement, is a basic construct that any useful programming or scripting language must have.

When an if statement is used, the ensuing actions depend on the evaluation of specified conditions such as:

- Numerical or string comparisons
- Return value of a command (0 for success)
- File existence or permissions

In compact form, the syntax of an if statement is:

if TEST-COMMANDS; then CONSEQUENT-COMMANDS; fi

A more general definition is:

```
if condition
then
      statements
else
      statements
fi
```

The if Statement

Nested if Statements

The elif Statement

Using the if Statement

```
[test3@CentOS ~]$ if [ -f /etc/passwd ]
> then
> echo "/etc/passwd exists."
> fi
/etc/passwd exists.
[test3@CentOS ~]$
```

The following if statement checks for the /etc/passwd file, and if the file is found it displays the message /etc/passwd exists.:

```
if [ -f /etc/passwd ]
then
   echo "/etc/passwd exists."
fi
```

Notice the use of the square brackets ([]) to delineate the test condition. There are many other kinds of tests you can perform, such as checking whether two numbers are equal to, greater than, or less than each other and make a decision accordingly; we will discuss these other tests.

In modern scripts you may see doubled brackets as in[[-f /etc/passwd]]. This is not an error. It is never wrong to do so and it avoids some subtle problems such as referring to an empty environment variable without surrounding it in double quotes; we won't talk about this here.

Testing for Files

You can use the if statement to test for file attributes such as:

- File or directory existence
- Read or write permission
- Executable permission

For example, in the following example:

```
if [ -f /etc/passwd ] ; then
   ACTION
fi
```

the if statement checks if the file /etc/passwd is a regular file.
Note the very common practice of putting **"; then"** on the same line as the **if** statement.

bash provides a set of **file conditionals**, that can used with the if statement, including:

Condition	Meaning
-e file	Check if the file exists.
-d file	Check if the file is a directory.
-f file	Check if the file is a regular file (i.e., not a symbolic link, device node, directory, etc.)
-s file	Check if the file is of non-zero size.
-g file	Check if the file has sgid set.
-u file	Check if the file has suid set.
-r file	Check if the file is readable.
-w file	Check if the file is writable.
-x file	Check if the file is executable.

You can view the full list of file conditions using the command man 1 test.

Example of Testing of Strings

You can use the if statement to compare strings using the operator == (two equal signs). The syntax is as follows:

```
if [ string1 == string2 ] ; then
   ACTION
fi
```

Let's now consider an example of testing strings.

In the example illustrated here, the if statement is used to compare the input provided by the user and accordingly display the result.

```
[test3@CentOS ~]$ vim sam2.sh
[test3@CentOS ~]$ bash sam2.sh
 Enter any color code [R OR Y OR G] :
G
G
MOVE.. IT IS UR TURN TO GO
[test3@CentOS ~]$ bash sam2.sh
 Enter any color code [R OR Y OR G] :
Y
Y
GET READY YOUR WAY WILL BE OPEN SHORTLY
[test3@CentOS ~]$ bash sam2.sh
 Enter any color code [R OR Y OR G] :
R
R
STOP! LEAVE WAY FOR OTHERS
[test3@CentOS ~]$ bash sam2.sh
 Enter any color code [R OR Y OR G] :
T
T
INCORRECT COLOR CODE
[test3@CentOS ~]$
```

```
#!/bin/bash
# Section that reads the input
echo " Enter any color code [R OR Y OR G] :"
read COLOR
echo $COLOR
# Section that compares the entry and display a message
if [ "$COLOR" == "R" ]
then
echo "STOP! LEAVE WAY FOR OTHERS"
elif [ "$COLOR" == "Y" ]
then
echo "GET READY YOUR WAY WILL BE OPEN SHORTLY"
elif [ "$COLOR" == "G" ]
then
echo "MOVE.. IT IS UR TURN TO GO"
else
echo "INCORRECT COLOR CODE"
fi
~
~
~
:wq
```

Numerical Tests

You can use specially defined operators with the if statement to compare numbers. The various operators that are available are listed in the table.

Operator	Meaning
-eq	Equal to
-ne	Not equal to
-gt	Greater than
-lt	Less than
-ge	Greater than or equal to
-le	Less than or equal to

The syntax for comparing numbers is as follows:
exp1 -op exp2

Example of Testing for Numbers

Let us now consider an example of comparing numbers using the various operators:

```
[test3@CentOS ~]$ vim sam3.sh
[test3@CentOS ~]$ chmod a+x sam3.sh
[test3@CentOS ~]$ bash sam3.sh
Please enter your age:
20
You are in your 20s
[test3@CentOS ~]$ bash sam3.sh
Please enter your age:
30
You are in your 30s
[test3@CentOS ~]$ bash sam3.sh
Please enter your age:
40
You are in your 40s
[test3@CentOS ~]$ bash sam3.sh
Please enter your age:
50
Sorry, you are out of the age range.
[test3@CentOS ~]$
```

```
#!/bin/bash
# Prompt for a user name...
echo "Please enter your age:"
read AGE
if [ "$AGE" -lt 20 ] || [ "$AGE" -ge 50 ] ; then
echo "Sorry, you are out of the age range."
elif [ "$AGE" -ge 20 ] && [ "$AGE" -lt 30 ] ; then
echo "You are in your 20s"
elif [ "$AGE" -ge 30 ] && [ "$AGE" -lt 40 ] ; then
echo "You are in your 30s"
elif [ "$AGE" -ge 40 ] && [ "$AGE" -lt 50 ] ; then
echo "You are in your 40s"
fi
~
~
~
~
~
~
~
~
:wq
```

Testing for Numbers (video)

Arithmetic Expressions

```
[test3@CentOS ~]$ expr 8 + 8
16
[test3@CentOS ~]$ echo $(expr 8 + 8)
16
[test3@CentOS ~]$ let x=( 1 + 2 ) ; echo $x
3
[test3@CentOS ~]$ echo $((x+1))
4
[test3@CentOS ~]$
```

Arithmetic expressions can be evaluated in the following three ways (spaces are important!):

- Using the **expr** utility: **expr** is a standard but somewhat deprecated program. The syntax is as follows:

  expr 8 + 8
  echo $(expr 8 + 8)
- Using the $((...)) syntax: This is the built-in shell format. The syntax is as follows:

  echo $((x+1))
- Using the built-in shell command let. The syntax is as follows:

  let x=(1 + 2); echo $x

In modern shell scripts the use of **expr** is better replaced with var=$((...))

Which are the correct commands to add two numbers?

- [x] expr 2 + 3
- [ ] echo expr 2 + 3
- [x] let x=(1 + 2); echo $x
- [ ] echo (expr(2 + 3))

Which of the following syntax is used to compare two strings?

- ( ) str1 = str2
- (•) str1 == str2
- ( ) str1 -eq str2
- ( ) None of the above

Which of the following are the correct conditionals that can be used to test for file attributes?

- [x] -e
- [ ] -z
- [x] -d
- [x] -r

Labs

Click the link to download a PDF from the Lab activity.

Bash Shell Scripting_Labs

Bash Shell Scripting_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- Scripts are a sequence of statements and commands stored in a file that can be executed by a shell. The most commonly used shell in Linux is **bash.**
- Command substitution allows you to substitute the result of a command as a portion of another command.
- Functions or routines are a group of commands that are used for execution.
- Environmental variables are quantities either pre-assigned by the shell or defined and modified by the user.

- To make environment variables visible to child processes, they need to be **exported**.
- Scripts can behave differently based on the parameters (values) passed to them.
- The process of writing the output to a file is called output redirection.
- The process of reading input from a file is called input redirection.
- The if statement is used to select an action based on a condition.
- Arithmetic expressions consist of numbers and arithmetic operators, such as +, -, and *.

Chapter 13: Advanced Bash Scripting

Learning Objectives

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

- 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.

Section 1: String Manipulation

String Manipulation

Let's go deeper and find out how to work with strings in scripts.

A **string variable** contains a sequence of text characters. It can include letters, numbers, symbols and punctuation marks. Some examples: abcde, 123, abcde 123, abcde-123, &acbde=%123

String **operators** include those that do comparison, sorting, and finding the length. The following table demonstrates the use of some basic string operators.

Operator	Meaning
[string1 > string2]	Compares the sorting order of string1 and string2.
[string1 == string2]	Compares the characters in string1 with the characters in string2.
myLen1=${#mystring1}	Saves the length of string1 in the variable myLen1.

Example of String Manipulation

See the screen shot.

In the first example, we compare the first string with the second string and display an appropriate message using the if statement.

In the second example, we pass in a file name and see if that file exists in the current directory or not.

```
[kcs@kcs ifdemo]$ ls
file1.txt  ifdemo1.sh  ifdemo2.sh
[kcs@kcs ifdemo]$ cat ifdemo1.sh
#!/bin/bash
if [ $1 == $2 ] ; then
  echo "The first string, $1, is the same as the second string, $2"
else
  echo "The first string, $1, is not the same as the second string, $2"
fi
[kcs@kcs ifdemo]$ ./ifdemo1.sh Apple Orange
The first string, Apple, is not the same as the second string, Orange
[kcs@kcs ifdemo]$ ./ifdemo1.sh Apple Apple
The first string, Apple, is the same as the second string, Apple
[kcs@kcs ifdemo]$ cat ifdemo2.sh
#!/bin/bash
file=$1
if [ -f "$file" ]
then
  echo File $file exists
else
  echo File $file does not exists
fi
[kcs@kcs ifdemo]$ ./ifdemo2.sh file1.txt
File file1.txt exists
[kcs@kcs ifdemo]$ ./ifdemo2.sh file2.txt
File file2.txt does not exists
[kcs@kcs ifdemo]$
```

Parts of a String

```
[test1@localhost ~]$ export name="bagend.hobbiton.com"
[test1@localhost ~]$ loco=${name:0:6}; echo $loco
bagend
[test1@localhost ~]$ moto=${name#*.}; echo $moto
hobbiton.com
[test1@localhost ~]$
```

At times, you may not need to compare or use an entire string. To extract the first character of a string we can specify:

${string:0:1} Here 0 is the offset in the string (i.e., which character to begin from) where the extraction needs to start and 1 is the number of characters to be extracted.

To extract all characters in a string after a dot (.), use the following expression: ${string#*.}

Testing for Strings

String Operations

Select the correct syntax to find the length of a string named 'abc'.

- $#[abc]
- ${abc#}
- $#{abc}
- ${#abc}

Boolean Expressions

Boolean expressions evaluate to either **TRUE** or **FALSE**, and results are obtained using the various Boolean operators listed in the table.

Operator	Operation	Meaning
&&	AND	The action will be performed only if both the conditions evaluate to true.
||	OR	The action will be performed if any one of the conditions evaluate to true.
!	NOT	The action will be performed only if the condition evaluates to false.

Note that if you have multiple conditions strung together with the && operator processing stops as soon as a condition evaluates to false. For example if you have A && B && C and A is true but B is false, C will never be executed.

Likewise if you are using the || operator, processing stops as soon as anything is true. For example if you have A || B || C and A is false and B is true, you will also never execute C.

Tests in Boolean Expressions

```
[test1@localhost ~]$ ls
date2.out  Desktop    Downloads  newfile.txt  Public   Templates  test1.txt  test3.txt
date.out   Documents  Music      Pictures     scratch  test1      test2.txt  Videos
[test1@localhost ~]$ if [ -e test1.txt ]; then
> echo "The test1.txt file exists. Continue processing"
> else
> echo "The test1.txt file does not exists. Stop processing"
> exit 1
> fi
The test1.txt file exists. Continue processing
[test1@localhost ~]$
```

Boolean expressions return either **TRUE** or **FALSE**. We can use such expressions when working with multiple data types including strings or numbers as well as with files. For example, to check if a file exists, use the following conditional test:

[-e <filename>]

Similarly, to check if the value of number1 is greater than the value of number2, use the following conditional test:

[$number1 -gt $number2]

The operator -gt returns **TRUE** if number1 is greater than number2.

Example of Using Operators

sc11.sh

```
#!/bin/bash
# Check if Test1.txt exists
if [ -e test1.txt ]; then
   echo "test1.txt exists. Continue processing."
else

 # else print the second statement and check the next validation.
   echo "test1.txt does not exist. Check temp directory."

# Check if the temp directory exists and  print first statement if TRUE
   if [ -d temp ]; then
      echo "temp directory exists. Create test1.txt in temp."

# create temporary file named Test1.txt
      touch temp/test1.txt
   else

 # else print second statement and perform next validation.
      echo "temp directory does not exist. Is sym link there"
      if [ -s sym-link ]; then
         echo "sym-link exists. All is good."
      else
         echo "sym-link does not exist. Bail out."
         exit 2
      fi
   fi
fi
```

```
[test1@localhost ~]$ ls
date2.out  Documents  newfile.txt  sc11.sh   Screen11.sh   test1
date.out   Downloads  Pictures     sc11.sh~  Screen11.sh~  test1.txt
Desktop    Music      Public       scratch   Templates     test2.txt
[test1@localhost ~]$
[test1@localhost ~]$ chmod a+x sc11.sh
[test1@localhost ~]$ bash sc11.sh
test1.txt exists. Continue processing.
[test1@localhost ~]$
```

The **NOT** operator is represented as _.

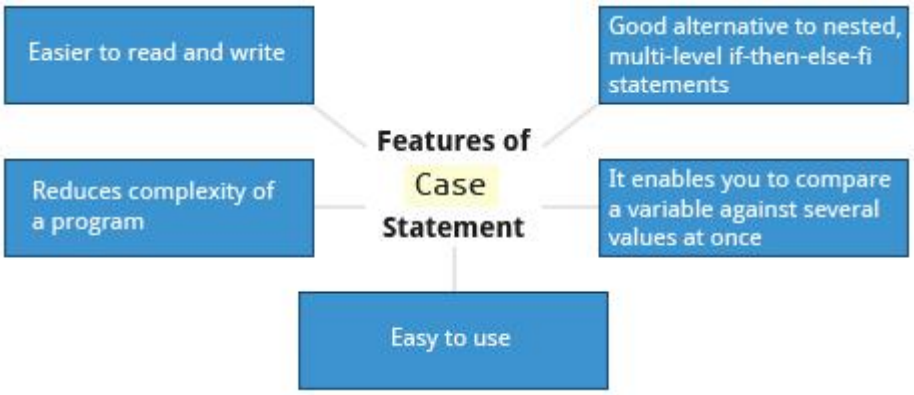
!

In an if statement, the ____ operator is used with a directory name to checking for its existence.

- -e
- -d
- -f
- -s

Section 3: The case Statement

The case Statement

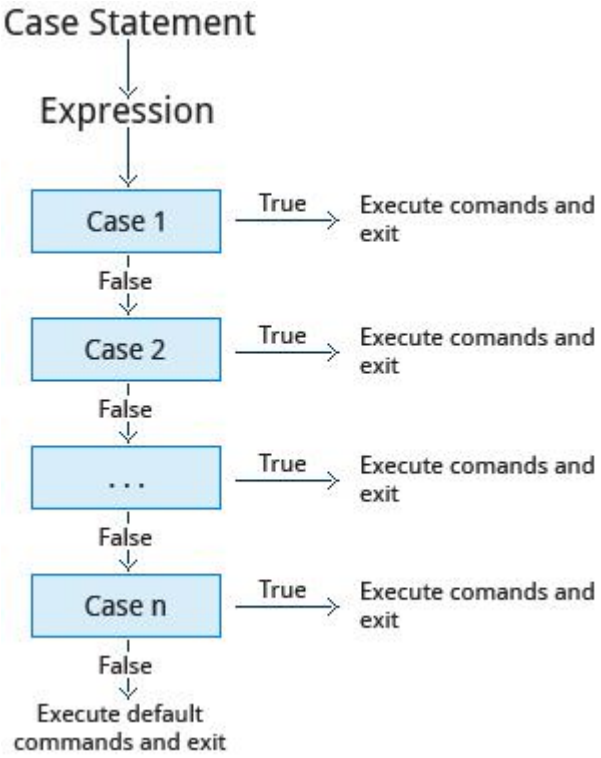


The case statement is used in scenarios where the actual value of a variable can lead to different execution paths. case statements are often used to handle command-line options.

Below are some of the advantages of using the case statement:

- It is easier to read and write.
- It is a good alternative to nested, multi-level if-then-else-fi code blocks.
- It enables you to compare a variable against several values at once.
- It reduces the complexity of a program.

Structure of the case Statement



Here is the basic structure of the case statement:

```
case expression in
  pattern1) execute commands;;
  pattern2) execute commands;;
  pattern3) execute commands;;
  pattern4) execute commands;;
  * )     execute some default commands or nothing ;;
esac
```

Example of the case Statement

Here's an example of a case statement, please click on the image to open it in a new tab.

```
#!/bin/sh
# Prompt user to enter a character
echo "Please enter a  letter:"
read charac
case "$charac" in
    "a"|"A") echo "You have typed a vowel!" ;;
    "e"|"E") echo "You have typed a vowel!" ;;
    "i"|"I") echo "You have typed a vowel!" ;;
    "o"|"O") echo "You have typed a vowel!" ;;
    "u"|"U") echo "You have typed a vowel!" ;;
    *)         echo "You have typed a consonant!" ;;
esac
exit 0

```

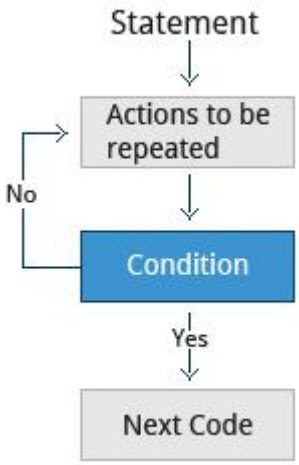
```
[test1@localhost ~]$ ./sam4.sh
Please enter a letter:
e
You have typed a vowel!
[test1@localhost ~]$ ./sam4.sh
Please enter a letter:
t
You have typed a consonant!
[test1@localhost ~]$
```

Which of the following statements are true about the case statement?

- [x] Enables you to match several values against one variable.
- [ ] Ensures that the condition is true for all cases.
- [x] Is a good alternative to nested multilevel if-then-else-fi statements.
- [ ] Indicates that any one of the conditions needs to be true, to perform the specified action.

Section 4: Looping Constructs

Looping Constructs



By using **looping constructs**, you can execute one or more lines of code repetitively. Usually you do this until a conditional test returns either true or false as is required.

Three type of loops are often used in most programming languages:

- for
- while
- until

All these loops are easily used for repeating a set of statements until the exit condition is true.

The 'for' Loop

The for loop operates on each element of a list of items. The syntax for the for loop is:

for *variable-name* in *list*
do
  execute one iteration for each item in the
    list until the *list* is finished
done

In this case, *variable-name* and *list* are substituted by you as appropriate (see examples). As with other looping constructs, the statements that are repeated should be enclosed by do and done.

The screenshots here show an example of the for loop to print the sum of numbers 1 to 4.

```
1 #!/bin/sh
2 #
3 sum=0
4 for i in 1 2 3 4
5 do
6   sum=$(($sum+$i))
7 done
8 echo "The sum of $i numbers is: $sum"
9
```

```
[test1@localhost ~]$ chmod a+x for.sh
[test1@localhost ~]$ ./for.sh
The sum of 4 numbers is: 10
[test1@localhost ~]$
```

Using the for and case Statements Together

The while Loop

The while loop repeats a set of statements as long as the control command returns true. The syntax is:

```
while condition is true
do
   Commands for execution
   ----
done
```

The set of commands that need to be repeated should be enclosed between do and done. You can use any command or operator as the condition. Often it is enclosed within square brackets ([]).

The screenshots here show an example of the **while** loop that calculates the factorial of a number.

```
#!/bin/bash
#
echo "Enter the number"
read no
fact=1
i=1
while [ $i -le $no ]
do
        fact=$(($fact * $i))
        i=$(($i + 1))
done
echo "The factorial of $no is $fact"
```

```
[test1@localhost ~]$ chmod a+x sc2.sh
[test1@localhost ~]$ bash sc2.sh
Enter the number
6
The factorial of 6 is 720
[test1@localhost ~]$ bash sc2.sh
Enter the number
3
The factorial of 3 is 6
[test1@localhost ~]$
```

While Statement

The until loop

The until loop repeats a set of statements as long as the control command is false. Thus it is essentially the opposite of the while loop. The syntax is:

```
until condition is false
do
   Commands for execution
   ----
done
```

Similar to the while loop, the set of commands that need to be repeated should be enclosed between do and done. You can use any command or operator as the condition.

```
[test2@OpenSUSE~]#cat until.sh
#Until Loop - Example-1
echo "NUMBER"
mn=1
mx=10
until [ $mn -gt $mx ]
do
        echo "$mn"
        mn=$(($mn + 2))
done
[test2@OpenSUSE~]#sh until.sh
NUMBER
1
3
5
7
9
[test2@OpenSUSE~]#
```

The screenshot here shows example of the until loop that displays odd numbers between 1 and 10.

The until Loop

Which of the following loops repeats a set of statements as long as the control command returns a true exit status?

- (•) while
- ( ) until
- ( ) for
- ( ) if else

Which of the following looping constructs repeats a set of statements for a known number of iterations?

- ( ) while
- ( ) until
- (•) for
- ( ) do

Section 5: Script Debugging

Introduction to Script Debugging

While working with scripts and commands, you may run into errors. These may be due to an error in the script, such as incorrect syntax, or other ingredients such as a missing file or insufficient permission to do an operation. These errors may be reported with a specific error code, but often just yield incorrect or confusing output. So how do you go about identifying and fixing an error?

Debugging helps you troubleshoot and resolve such errors, and is one of the most important tasks a system administrator performs.

More About Script Debugging

Before fixing an error (or bug), it is vital to know its source.

In **bash** shell scripting, you can run a script in **debug mode** by doing bash –x ./script_file. Debug mode helps identify the error because:

- It traces and prefixes each command with the + character.
- It displays each command before executing it.
- It can debug only selected parts of a script (if desired) with:
  set -x # turns on debugging
  ...
  set +x # turns off debugging

The screenshots shown here demonstrate a scriptfile called sc2 (left image) and the results from running it in debug mode (right image).

```
[test1@localhost ~]$ ./c15s9.sh
Enter your name with title:
Mrs.Mary Ann
~~~~~~~~~~~~~~~~~~~~~~~~~~~
My title is Mrs
My name is Mary Ann
~~~~~~~~~~~~~~~~~~~~~~~~~~~
[test1@localhost ~]$
```

```
1 #!/bin/bash
2 # Section that reads the input
3 echo "Enter your name with title:"
4 read name
5 echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~"
6 echo "My title is ${name:0:3}"
7 echo "My name is ${name#*.}"
8 echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~"
```

```
[test1@localhost ~]$ ./c15s9.sh
Enter your name with title:
+ read name
Mrs.Annie Brown
+ echo '~~~~~~~~~~~~~~~~~~~~~~~~~~~'
~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ echo 'My title is Mrs'
My title is Mrs
+ echo 'My name is Annie Brown'
My name is Annie Brown
+ set +x
~~~~~~~~~~~~~~~~~~~~~~~~~~~
[test1@localhost ~]$
```

```
#!/bin/bash
# Section that reads the input
echo "Enter your name with title:"
set -x
read name
echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~"
echo "My title is ${name:0:3}"
echo "My name is ${name#*.}"
set +x
echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~"
```

Redirecting Errors to File and Screen

In UNIX/Linux, all programs that run are given three open file streams when they are started as listed in the table:

File stream	Description	File Descriptor
stdin	Standard Input, by default the keyboard/terminal for programs run from the command line	0
stdout	Standard output, by default the screen for programs run from the command line	1
stderr	Standard error, where output error messages are shown or saved	2

Using redirection we can save the **stdout** and **stderr** output streams to one file or two separate files for later analysis after a program or command is executed

On the left screen is a buggy shell script. On the right screen the buggy script is executed and the errors are redirected to the file "error.txt". Using "cat" to display the contents of "error.txt" shows the errors of executing the buggy shell script (presumably for further debugging).

```
[test1@localhost ~]$ ls error.txt
ls: cannot access error.txt: No such file or directory
[test1@localhost ~]$ bash sample.sh 2> error.txt
[test1@localhost ~]$ ls error.txt
error.txt
[test1@localhost ~]$ cat error.txt
sample.sh: line 6: syntax error near unexpected token `('
sample.sh: line 6: `   sum=(($sum+$i))'
sample.sh: line 7: syntax error near unexpected token `done'
sample.sh: line 7: `done'
[test1@localhost ~]$
```

```
#!/bin/sh
#
sum=0
for i in 1 2 3 4
do
  sum=(($sum+$i))
done
echo "The sum of $i numbers is: $sum"
ls error
```

Select the command that appends the error output to a temporary log file.

- $ ./script 2>> /tmp/scriptlogfile
- $ ./script 2> /tmp/scriptlogfile
- $ ./script 2<< /tmp/scriptlogfile
- $ ./script 2/>> /tmp/scriptlogfile

If you want to debug the 15th line of a script, we must provide the _____ command on 14th line and the _____command on 16th line of the script.

- ./script 2> >, set +x
- set -x, set +x
- ./script 2>>, set +x
- set +x, ./script 2>>

Section 6: Some additional useful techniques

Creating Temporary Files and Directories

Consider a situation where you want to retrieve 100 records from a file with 10,000 records. You will need a place to store the extracted information, perhaps in a **temporary file**, while you do further processing on it.

Temporary files (and directories) are meant to store data for a short time. Usually one arranges it so that these files disappear when the program using them terminates. While you can also use **touch** to create a temporary file, this may make it easy for hackers to gain access to your data.

The best practice is to create random and unpredictable filenames for temporary storage. One way to do this is with the **mktemp** utility as in these examples:

The XXXXXXXX is replaced by the **mktemp** utility with random characters to ensure the name of the temporary file cannot be easily predicted and is only known within your program.

Command	Usage
TEMP=$(mktemp /tmp/tempfile.XXXXXXXX)	To create a temporary file
TEMPDIR=$(mktemp -d /tmp/tempdir.XXXXXXXX)	To create a temporary directory

Example of Creating a Temporary File and Directory

```
[test1@localhost ~]$ ls /tmp
keyring-3npM6w  keyring-XAo3Kh         pulse-2FJRudfiHYlW      virtual-root.maXSRW   virtual-test1.YuzESF  vmware-root
keyring-F7U13a  ks-script-I7AXzM       pulse-hwIKrjR20J5f      virtual-test1.1ZFJTm  virtual-test1.Z4eDdV  vmware-root-1857424609
keyring-MyhO1f  ks-script-I7AXzM.log   pulse-kmc0ovKppMhv      virtual-test1.Kmx3JX  vmware-config0        vmware-test1
keyring-ojByp5  orbit-gdm              pulse-mPe8bzgDOxVu      virtual-test1.RkhXtl  VMwareDnD             yum.log
keyring-tAVs8b  orbit-test1            virtual-mgnanda.ukM6kD  virtual-test1.xGekGH  vmware-mgnanda
[test1@localhost ~]$
```

First, the danger: If someone creates a symbolic link from a known temporary file used by root to the /etc/passwd file, like this:

$ ln -s /etc/passwd /tmp/tempfile
There could be a big problem if a script run by root has a line in like this:

echo $VAR > /tmp/tempfile

The password file will be overwritten by the temporary file contents.

To prevent such a situation make sure you randomize your temporary filenames by replacing the above line with the following lines:

TEMP=$(mktemp /tmp/tempfile.XXXXXXXX)
echo $VAR > $TEMP

Discarding Output with /dev/null

```
[root@localhost test1]# ls -l /dev/null
crw-rw-rw-. 1 root root 1, 3 Jul 26 13:32 /dev/null
[root@localhost test1]# find / > /dev/null
^Z
[2]+  Stopped                 find / > /dev/null
[root@localhost test1]# cat /dev/null
[root@localhost test1]#
```

Certain commands like **find** will produce voluminous amounts of output which can overwhelm the console. To avoid this, we can redirect the large output to a special file (a device node) called **/dev/null**. This file is also called the **bit bucket** or **black hole**.

It discards all data that gets written to it and never returns a failure on write operations. Using the proper redirection operators, it can make the output disappear from commands that would normally generate output to **stdout** and/or **stderr**:

$ find / > /dev/null
In the above command, the entire standard output stream is ignored, but any errors will still appear on the console.

Random Numbers and Data

```
[test2@OpenSUSE:~]#echo $RANDOM
19561
[test2@OpenSUSE:~]#echo $RANDOM
12887
[test2@OpenSUSE:~]#echo $RANDOM
19622
[test2@OpenSUSE:~]#
```

It is often useful to generate random numbers and other random data when performing tasks such as:

- Performing security-related tasks.
- Reinitializing storage devices.
- Erasing and/or obscuring existing data.
- Generating meaningless data to be used for tests.

Such random numbers can be generated by using the $RANDOM environment variable, which is derived from the Linux kernel's built-in random number generator, or by the **OpenSSL** library function, which uses the FIPS140 algorithm to generate random numbers for encryption

To read more about FIPS140, see http://en.wikipedia.org/wiki/FIPS_140-2

The example shows you how to easily use the environmental variable method to generate random numbers.

How the Kernel Generates Random Numbers

```
[test3@CentOS ~]$ ls -l /dev/*random
crw-rw-rw-. 1 root root 1, 8 Jul 28 11:51 /dev/random
crw-rw-rw-. 1 root root 1, 9 Jul 28 11:51 /dev/urandom
[test3@CentOS ~]$
```

Some servers have hardware random number generators that take as input different types of noise signals, such as thermal noise and photoelectric effect. A **transducer** converts this noise into an electric signal, which is again converted into a digital number by an **A-D converter**. This number is considered random. However, most common computers do not contain such specialized hardware and instead rely on events created during booting to create the raw data needed.

Regardless of which of these two sources is used, the system maintains a so-called **entropy pool** of these digital numbers/random bits. Random numbers are created from this entropy pool.

The Linux kernel offers the **/dev/random** and **/dev/urandom** device nodes which draw on the entropy pool to provide random numbers which are drawn from the estimated number of bits of noise in the entropy pool.

/dev/random is used where very high quality randomness is required, such as one-time pad or key generation, but it is relatively slow to provide vaules. **/dev/urandom** is faster and suitable (good enough) for most cryptographic purposes.

Furthermore, when the entropy pool is empty, **/dev/random** is blocked and does not generate any number until additional environmental noise (network traffic, mouse movement, etc.) is gathered whereas **/dev/urandom** reuses the internal pool to produce more pseudo-random bits.

Which command is used to create a temporary directory?

- (•) mktemp –d
- ( ) mktemp –dir
- ( ) Mktemp
- ( ) mktemp –dr

The **/dev/null** file is also known as the _____ or black hole.
bit bucket or **black hole**

Which of the following Linux device nodes provide random numbers?

- [ ] /dev/rnd
- [x] /dev/urandom
- [ ] /dev/urnd
- [x] /dev/random

Labs

Click the link to download a PDF from the Lab activity.

Advanced Bash Shell Scripting_Labs

Advanced Bash Shell Scripting_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- You can manipulate strings to perform actions such as comparison, sorting, and finding length.
- You can use Boolean expressions when working with multiple data types including strings or numbers as well as files.
- The output of a Boolean expression is either true or false.
- Operators used in Boolean expressions include the && (AND), ||(OR), and ! (NOT) operators.

- We looked at the advantages of using case statement in scenarios where the value of a variable can lead to different execution paths.
- Script debugging methods help troubleshoot and resolve errors.
- The standard and error outputs from a script or shell commands can easily be redirected into the same file or separate files to aid in debugging and saving results
- Linux allows you to create temporary files and directories, which store data for a short duration, both saving space and increasing security.
- Linux provides several different ways of generating random numbers, which are widely used.

Chapter 17: Processes

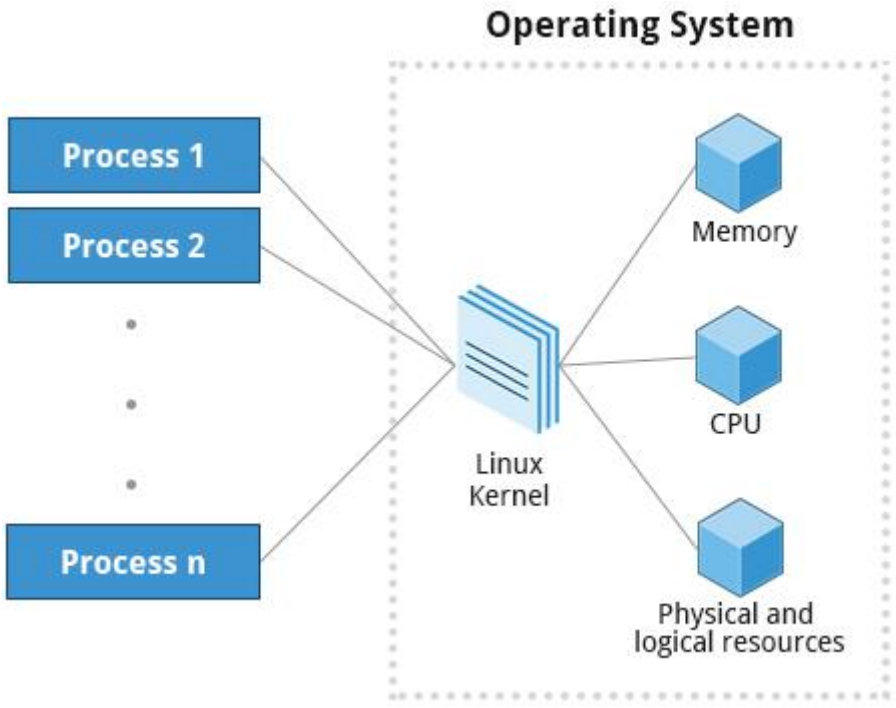
Learning Objectives

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

- Describe what a process is and distinguish between types of processes.
- Enumerate process attributes.
- Manage processes using **ps** and **top**.
- Understand the use of load averages and other process metrics.
- Manipulate processes by putting them in **background** and restoring them to **foreground**.
- Use **at**, **cron**, and **sleep** to schedule processes in the future or pause them.

Section 1: Introduction to Processes and Process Attributes

What Is a Process?



A process is simply an instance of one or more related **tasks (threads)** executing on your computer. It is not the same as a **program** or a **command**; a single program may actually start several processes simultaneously. Some processes are independent of each other and others are related. A failure of one process may or may not affect the others running on the system.

Processes use many system resources, such as memory, CPU (central processing unit) cycles, and peripheral devices such as printers and displays. The operating system (especially the kernel) is responsible for allocating a proper share of these resources to each process and ensuring overall optimum utilization.

Process Types

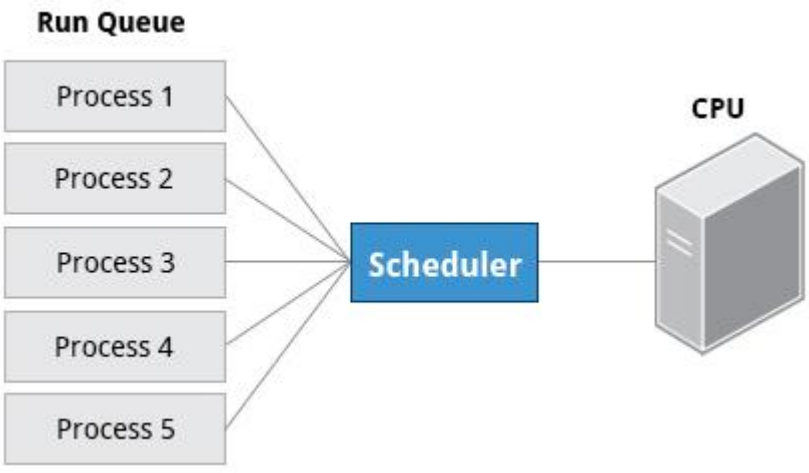
A terminal window (one kind of command shell), is a process that runs as long as needed. It allows users to execute programs and access resources in an interactive environment. You can also run programs in the **background**, which means they become **detached** from the shell.

Processes can be of different types according to the task being performed. Here are some different process types along with their descriptions and examples.

Process Type	Description	Example
Interactive Processes	Need to be started by a user, either at a command line or through a graphical interface such as an icon or a menu selection.	**bash, firefox, top**
Batch Processes	Automatic processes which are scheduled from and then disconnected from the terminal. These tasks are queued and work on a **FIFO** (First In, First Out) basis.	**updatedb**
Daemons	Server processes that run continuously. Many are launched during system startup and then wait for a user or system request indicating that their service is required.	**httpd, xinetd, sshd**
Threads	Lightweight processes. These are **tasks** that run under the umbrella of a main process, sharing memory and other resources, but are scheduled and run by the system on an individual basis. An individual thread can end without terminating the whole process and a process can create new threads at any time. Many non-trivial programs are multi-threaded.	**gnome-terminal, firefox**
Kernel Threads	Kernel tasks that users neither start nor terminate and have little control over. These may perform actions like moving a thread from one CPU to another, or making sure input/output operations to disk are completed.	**kswapd0, migration, ksoftirqd**

Process Scheduling and States

When a process is in a so-called **running** state, it means it is either currently executing instructions on a CPU, or is waiting for a share (or **time slice)** so it can run. A critical kernel routine called the **scheduler** constantly shifts processes in and out of the CPU, sharing time according to relative priority, how much time is needed and how much has already been granted to a task. All processes in this state reside on what is called a **run queue** and on a computer with multiple CPUs, or cores, there is a run queue on each.



However, sometimes processes go into what is called a **sleep** state, generally when they are waiting for something to happen before they can resume, perhaps for the user to type something. In this condition a process is sitting in a **wait** queue.

There are some other less frequent process states, especially when a process is terminating. Sometimes a child process completes but its parent process has not asked about its state. Amusingly such a process is said to be in a **zombie** state; it is not really alive but still shows up in the system's list of processes.

Process and Thread IDs

At any given time there are always multiple processes being executed. The operating system keeps track of them by assigning each a unique **process ID** (**PID**) number. The PID is used to track process state, cpu usage, memory use, precisely where resources are located in memory, and other characteristics.

New PIDs are usually assigned in ascending order as processes are born. Thus PID 1 denotes the **init** process (initialization process), and succeeding processes are gradually assigned higher numbers.

The table explains the PID types and their descriptions:

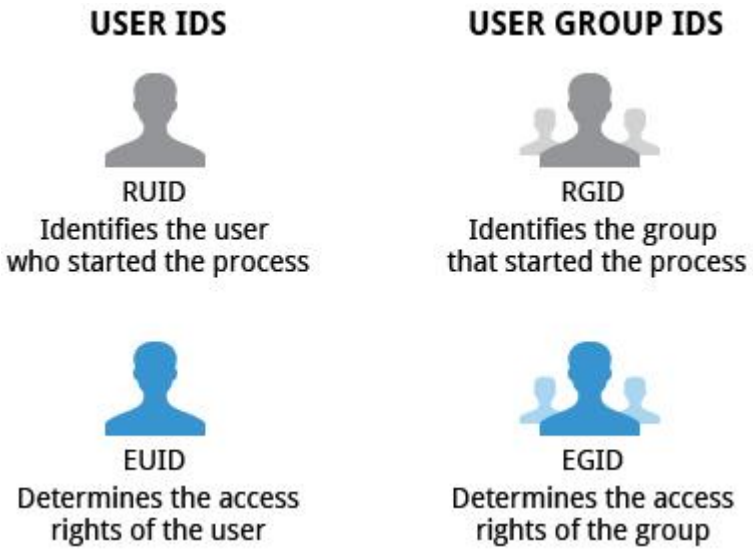
ID Type	Description
Process ID (PID)	Unique Process ID number
Parent Process ID (PPID)	Process (Parent) that started this process
Thread ID (TID)	Thread ID number. This is the same as the PID for single-threaded processes. For a multi-threaded process, each thread shares the same PID but has a unique TID.

User and Group IDs

Many users can access a system simultaneously, and each user can run multiple processes. The operating system identifies the user who starts the process by the Real User ID (**RUID**) assigned to the user.

The user who determines the access rights for the users is identified by the Effective UID (**EUID**). The EUID may or may not be the same as the RUID.

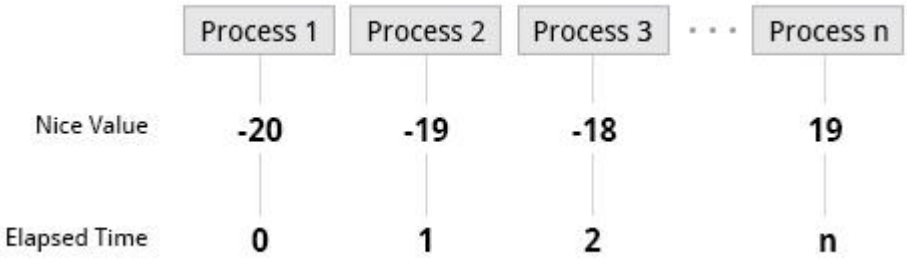
Users can be categorized into various groups. Each group is identified by the Real Group ID, or **RGID**. The access rights of the group are determined by the Effective Group ID, or **EGID**. Each user can be a member of one or more groups.



Most of the time we ignore these details and just talk about the User ID (**UID**).

More About Priorities

At any given time, many processes are running (i.e., in the run queue) on the system. However, a CPU can actually accommodate only one task at a time, just like a car can have only one driver at a time. Some processes are more important than others so Linux allows you to set and manipulate process **priority**. Higher priority processes are granted more time on the CPU.



The priority for a process can be set by specifying a **nice value**, or **niceness**, for the process. The lower the nice value, the higher the priority. Low values are assigned to important processes, while high values are assigned to processes that can wait longer. A process with a high nice value simply allows other processes to be executed first. In Linux, a nice value of -20 represents the highest priority and 19 represents the lowest. (This does sound kind of backwards, but this convention, the nicer the process, the lower the priority, goes back to the earliest days of UNIX.)

You can also assign a so-called **real-time priority** to time-sensitive tasks, such as controlling machines through a computer or collecting incoming data. This is just a very high priority and is not to be confused with what is called **hard real time** which is conceptually different, and has more to do with making sure a job gets completed within a very well-defined time window.

Identify the server processes that get initialized during system startup and then wait for a user or system request indicating that their service is required.

- ○ Batch Processes
- ◉ Daemons Daemons
- ○ Threads
- ○ Kernel Threads

Which of the following is scheduled to run and then disconnected from the terminal?

- ○ interactive process
- ◉ batch process
- ○ kernel threads
- ○ threads

What are possible states of a process?

- ☑ Running
- ☐ Agitated
- ☑ Sleeping (Waiting)
- ☐ Nice

Which of the following is the unique identifier for a process?

- ◉ PID
- ○ PPID
- ○ TID
- ○ UID

Identify the correct statements.

- ☐ High priority jobs have higher nice value.
- ☑ High priority jobs have lower nice value.
- ☑ Low priority jobs have higher nice value.
- ☐ Low priority jobs have lower nice value.

The ps Command (System V Style)

```
[test1@localhost ~]$ ps -u test1
  PID TTY          TIME CMD
 2316 ?        00:00:00 gnome-keyring-d
 2327 ?        00:00:00 gnome-session
 2335 ?        00:00:00 dbus-launch
 2336 ?        00:00:00 dbus-daemon
 2354 ?        00:00:00 gconfd-2
 2362 ?        00:00:02 gnome-settings-
 2364 ?        00:00:00 seahorse-daemon
 2366 ?        00:00:00 gvfsd
 2386 ?        00:00:00 metacity
 2387 ?        00:00:01 gnome-panel
 2388 ?        00:00:04 nautilus
 2391 ?        00:00:00 bonobo-activati
 2395 ?        00:00:00 gnome-volume-co
```

ps provides information about currently running processes, keyed by **PID**. If you want a repetitive update of this status, you can use **top** or commonly installed variants such as **htop** or **atop** from the command line, or invoke your distribution's graphical system monitor application.

ps has many options for specifying exactly which tasks to examine, what information to display about them, and precisely what output format should be used.

Without options **ps** will display all processes running under the current shell. You can use the -u option to display information of processes for a specified username. The command ps -ef displays all the processes in the system in full detail. The command ps -eLf goes one step further and displays one line of information for every **thread** (remember, a process can contain multiple threads).

The ps Command (BSD Style)

ps has another style of option specification which stems from the **BSD** variety of UNIX, where options are specified without preceding dashes. For example, the command ps aux displays all processes of all users. The command ps axo allows you to specify which attributes you want to view.

The following tables shows sample output of **ps** with the aux and axo qualifiers.

Command	Output
ps aux	USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND root 1 0.0 0.0 19356 1292 ? Ss Feb27 0:08 /sbin/init root 2 0.0 0.0 0 0 ? S Feb27 0:00 [kthreadd] root 3 0.0 0.0 0 0 ? S Feb27 0:27 [migration/0] . . .

Command	Output
ps aux stat,priority,pid,pcpu,comm	STAT PRI PID %CPU COMMAND Ss 20 1 0.0 init S 20 2 0.0 kthreadd S -100 3 0.0 migration/0 . . .

Using ps

The Process Tree

```
test2@OpenSUSE:~> pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─3*[VBoxClient───{VBoxClient}]
        ├─VBoxClient───2*[{VBoxClient}]
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─agetty
        ├─at-spi-bus-laun─┬─dbus-daemon
        │                 └─3*[{at-spi-bus-laun}]
        ├─at-spi2-registr───{at-spi2-registr}
        ├─avahi-autoipd───avahi-autoipd
        ├─avahi-daemon
        ├─bluetoothd
        ├─colord───2*[{colord}]
        ├─cron
        ├─cupsd
        ├─2*[dbus-daemon]
        ├─dbus-launch
        ├─dconf-service───2*[{dconf-service}]
        ├─dhclient6
```

At some point one of your applications may stop working properly. How might you terminate it?

pstree displays the processes running on the system in the form of a **tree diagram** showing the relationship between a process and its parent process and any other processes that it created. Repeated entries of a process are not displayed, and threads are displayed in curly braces.

To terminate a process you can type kill -SIGKILL <pid> or kill -9 <pid>. Note however, you can only **kill** your own processes: those belonging to another user are off limits unless you are root.

top

```
top - 16:14:55 up 4 days, 14:40,  3 users,  load average: 0.48, 0.20, 0.11
Tasks: 167 total,   2 running, 164 sleeping,   0 stopped,   1 zombie
%Cpu(s):  2.1 us,  8.0 sy,  0.0 ni, 89.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem:   1020440 total,   946556 used,    73884 free,    20380 buffers
KiB Swap:  1589244 total,   632116 used,   957128 free,   246532 cached

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1987 test2      9 -11  442204   2104   1336 S 12.29 0.206 140:12.33 pulseaudio
 3185 test2     20   0  444216   2924   1176 S 9.634 0.287   7:28.85 sd_espeak
 2229 test2     20   0 2126148 189700  12072 S 0.997 18.59  26:32.41 gnome-shell
 2749 test2     20   0  748692 169028   3252 S 0.997 16.56   6:35.50 orca
 1270 test2     20   0  103428    168    168 S 0.664 0.016  25:28.50 VBoxClient
19783 test2     20   0 1157756 187008  17036 S 0.664 18.33   6:55.73 firefox
18464 test2     20   0 2341924   8284   4144 S 0.332 0.812   5:00.30 soffice.bin
19131 test2     20   0  660772  17676  10336 S 0.332 1.732   0:07.61 gnome-terminal-
    1 root      20   0   47564   2444   1224 S 0.000 0.240   0:02.63 systemd
    2 root      20   0       0      0      0 S 0.000 0.000   0:00.36 kthreadd
    3 root      20   0       0      0      0 S 0.000 0.000   0:20.02 ksoftirqd/0
    5 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S 0.000 0.000   0:00.35 migration/0
    8 root      -2   0       0      0      0 S 0.000 0.000   0:00.00 rcuc/0
    9 root      -2   0       0      0      0 S 0.000 0.000   0:00.00 rcub/0
   10 root      20   0       0      0      0 S 0.000 0.000   0:32.17 rcu_preempt
   11 root      20   0       0      0      0 S 0.000 0.000   0:21.53 rcuop/0
   12 root      20   0       0      0      0 R 0.000 0.000   0:20.93 rcuop/1
```

While a static view of what the system is doing is useful, monitoring the system performance live over time is also valuable. One option would be to run **ps** at regular intervals, say, every two minutes. A better alternative is to use **top** to get constant real-time updates (every two seconds by default) until you exit by typing q. **top** clearly highlights which processes are consuming the most CPU cycles and memory (using appropriate commands from within **top**.)

First Line of the top Output

```
top - 16:14:55 up 4 days, 14:40,  3 users,  load average: 0.48, 0.20, 0.11
Tasks: 167 total,   2 running, 164 sleeping,   0 stopped,   1 zombie
%Cpu(s):  2.1 us,  8.0 sy,  0.0 ni, 89.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem:   1020440 total,   946556 used,    73884 free,    20380 buffers
KiB Swap:  1589244 total,   632116 used,   957128 free,   246532 cached
```

The first line of the **top** output displays a quick summary of what is happening in the system including:

- How long the system has been up
- How many users are logged on
- What is the load average

The **load average** determines how busy the system is. A load average of 1.00 per CPU indicates a fully subscribed, but not overloaded, system. If the load average goes above this value, it indicates that processes are competing for CPU time. If the load average is very high, it might indicate that the system is having a problem, such as a runaway process (a process in a non-responding state).

Second Line of the top Output

```
top - 16:14:55 up 4 days, 14:40,  3 users,  load average: 0.48, 0.20, 0.11
Tasks: 167 total,   2 running, 164 sleeping,   0 stopped,   1 zombie
%Cpu(s):  2.1 us,  8.0 sy,  0.0 ni, 89.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem:   1020440 total,   946556 used,    73884 free,    20380 buffers
KiB Swap:  1589244 total,   632116 used,   957128 free,   246532 cached
```

The second line of the **top** output displays the total number of processes, the number of running, sleeping, stopped and zombie processes. Comparing the number of running processes with the load average helps determine if the system has reached its capacity or perhaps a particular user is running too many processes. The stopped processes should be examined to see if everything is running correctly.

Third Line of the top Output

```
top - 16:14:55 up 4 days, 14:40,  3 users,  load average: 0.48, 0.20, 0.11
Tasks: 167 total,   2 running, 164 sleeping,   0 stopped,   1 zombie
%Cpu(s):  2.1 us,  8.0 sy,  0.0 ni, 89.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem:   1020440 total,   946556 used,    73884 free,    20380 buffers
KiB Swap:  1589244 total,   632116 used,   957128 free,   246532 cached
```

The third line of the **top** output indicates how the CPU time is being divided between the users (**us**) and the kernel (**sy**) by displaying the percentage of CPU time used for each.

The percentage of user jobs running at a lower priority (niceness - **ni**) is then listed. Idle mode (**id**) should be low if the load average is high, and vice versa. The percentage of jobs waiting (**wa**) for I/O is listed. Interrupts include the percentage of hardware (**hi**) vs. software interrupts (**si**). Steal time (**st**) is generally used with virtual machines, which has some of its idle CPU time taken for other uses.

Fourth and Fifth Lines of the top Output

```
top - 16:14:55 up 4 days, 14:40,  3 users,  load average: 0.48, 0.20, 0.11
Tasks: 167 total,   2 running, 164 sleeping,   0 stopped,   1 zombie
%Cpu(s):  2.1 us,  8.0 sy,  0.0 ni, 89.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem:   1020440 total,   946556 used,    73884 free,    20380 buffers
KiB Swap:  1589244 total,   632116 used,   957128 free,   246532 cached
```

The fourth and fifth lines of the **top** output indicate memory usage, which is divided in two categories:

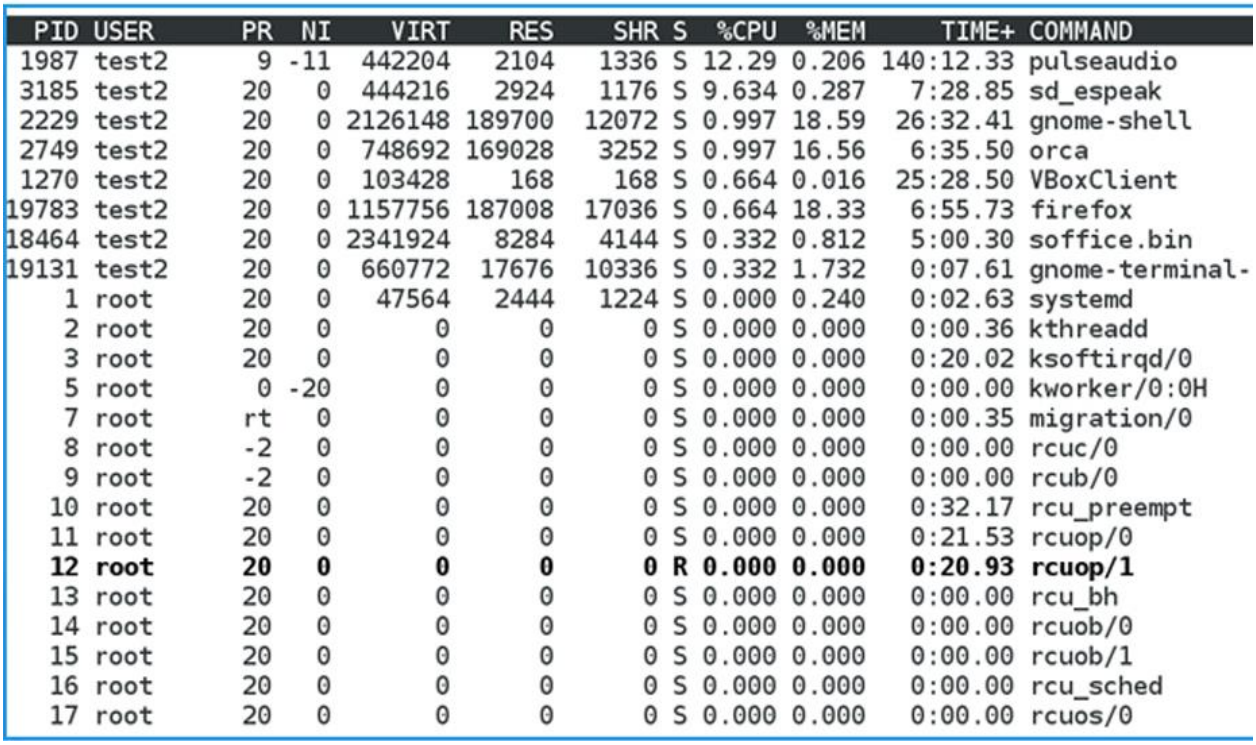
- Physical memory (RAM) – displayed on line 4.
- Swap space – displayed on line 5.

Both categories display total memory, used memory, and free space.

You need to monitor memory usage very carefully to ensure good system performance. Once the physical memory is exhausted, the system starts using **swap** space (temporary storage space on the hard drive) as an extended memory pool, and since accessing disk is much slower than accessing memory, this will negatively affect system performance.

If the system starts using swap often, you can add more swap space. However, adding more physical memory should also be considered.

Process List of the top Output



```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1987 test2      9 -11  442204   2104   1336 S 12.29 0.206 140:12.33 pulseaudio
 3185 test2     20   0  444216   2924   1176 S 9.634 0.287   7:28.85 sd_espeak
 2229 test2     20   0 2126148 189700  12072 S 0.997 18.59  26:32.41 gnome-shell
 2749 test2     20   0  748692 169028   3252 S 0.997 16.56   6:35.50 orca
 1270 test2     20   0  103428    168    168 S 0.664 0.016  25:28.50 VBoxClient
19783 test2     20   0 1157756 187008  17036 S 0.664 18.33   6:55.73 firefox
18464 test2     20   0 2341924   8284   4144 S 0.332 0.812   5:00.30 soffice.bin
19131 test2     20   0  660772  17676  10336 S 0.332 1.732   0:07.61 gnome-terminal-
    1 root      20   0   47564   2444   1224 S 0.000 0.240   0:02.63 systemd
    2 root      20   0       0      0      0 S 0.000 0.000   0:00.36 kthreadd
    3 root      20   0       0      0      0 S 0.000 0.000   0:20.02 ksoftirqd/0
    5 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S 0.000 0.000   0:00.35 migration/0
    8 root      -2   0       0      0      0 S 0.000 0.000   0:00.00 rcuc/0
    9 root      -2   0       0      0      0 S 0.000 0.000   0:00.00 rcub/0
   10 root      20   0       0      0      0 S 0.000 0.000   0:32.17 rcu_preempt
   11 root      20   0       0      0      0 S 0.000 0.000   0:21.53 rcuop/0
   12 root      20   0       0      0      0 R 0.000 0.000   0:20.93 rcuop/1
   13 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcu_bh
   14 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcuob/0
   15 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcuob/1
   16 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcu_sched
   17 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcuos/0
```

Each line in the process list of the **top** output displays information about a process. By default, processes are ordered by highest CPU usage. The following information about each process is displayed:

- Process Identification Number (PID)
- Process owner (USER)
- Priority (PR) and nice values (NI)
- Virtual (VIRT), physical (RES), and shared memory (SHR)
- Status (S)
- Percentage of CPU (%CPU) and memory (%MEM) used
- Execution time (TIME+)
- Command (COMMAND)

Interactive Keys with top

Besides reporting information, **top** can be utilized interactively for monitoring and controlling processes. While **top** is running in a terminal window you can enter single-letter commands to change its behaviour. For example, you can view the top-ranked processes based on CPU or memory usage. If needed, you can alter the priorities of running processes or you can stop/kill a process.

The table lists what happens when pressing various keys when running **top**:

Command	Output
t	Display or hide summary information (rows 2 and 3)
m	Display or hide memory information (rows 4 and 5)
A	Sort the process list by top resource consumers
r	Renice (change the priority of) a specific processes
k	Kill a specific process
f	Enter the top configuration screen
o	Interactively select a new sort order in the process list

Using top

Which of the following can be used to view process information?

- [x] top
- [x] pstree
- [ ] which
- [ ] w

Which of the following describes things **pstree** can display?

- [x] The relationship between parent and child processes
- [ ] Recently terminated processes
- [ ] Repeated processes
- [x] Threads in curly braces

Which of the following are displayed by **top**?

- [x] PID
- [ ] PR
- [ ] UID
- [x] USER

What information does the second line of the **top** output display?

- [ ] user sessions
- [x] stopped processes
- [ ] steal time
- [x] zombie processes

Which key is used to sort the process list by top resource consumers when using **top**?

- ( ) S
- ( ) F
- ( ) K
- (•) A

Load Averages

```
test2@OpenSUSE:~> w
 17:10:20 up 4 days, 15:35,  3 users,  load average: 0.04, 0.08, 0.13
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
test2    :0       console          Wed01   ?xdm?   4:03m  0.07s /usr/lib/gdm/gdm-simple-sl
test2    console  :0               Wed01    4days  0.00s  0.07s /usr/lib/gdm/gdm-simple-sl
test2    pts/0    :0               16:37    4.00s  0.22s  0.00s w
test2@OpenSUSE:~>
```

Load average is the average of the **load number** for a given period of time. It takes into account processes that are:

- Actively running on a CPU.
- Considered runnable, but waiting for a CPU to become available.
- Sleeping: i.e., waiting for some kind of resource (typically, I/O) to become available.

The load average can be obtained by running **w**, **top** or **uptime**.

Interpreting Load Averages

The load average is displayed using three different sets of numbers, as shown in the following example:

The last piece of information is the average load of the system. Assuming our system is a single-CPU system, the 0.25 means that for the past minute, on average, the system has been 25% utilized. 0.12 in the next position means that over the past 5 minutes, on average, the system has been 12% utilized; and 0.15 in the final position means that over the past 15 minutes, on average, the system has been 15% utilized. If we saw a value of 1.00 in the second position, that would imply that the single-CPU system was 100% utilized, on average, over the past 5 minutes; this is good if we want to fully use a system. A value over 1.00 for a single-CPU system implies that the system was over-utilized: there were more processes needing CPU than CPU was available.

If we had more than one CPU, say a quad-CPU system, we would divide the load average numbers by the number of CPUs. In this case, for example, seeing a 1 minute load average of 4.00 implies that the system as a whole was 100% (4.00/4) utilized during the last minute.

Short term increases are usually not a problem. A high peak you see is likely a burst of activity, not a new level. For example, at start up, many processes start and then activity settles down. If a high peak is seen in the 5 and 15 minute load averages, it would probably be cause for concern.

Background and Foreground Processes

```
[root@CentOS test3]# updatedb &
[1] 24660
[root@CentOS test3]# sleep 100
^Z[1]   Done                    updatedb

[2]+  Stopped                 sleep 100
[root@CentOS test3]# bg
[2]+ sleep 100 &
[root@CentOS test3]# fg
sleep 100
^C
[root@CentOS test3]#
```

Linux supports **background** and **foreground** job processing. (A job in this context is just a command launched from a terminal window.) **Foreground** jobs run directly from the shell, and when one foreground job is running, other jobs need to wait for shell access (at least in that terminal window if using the GUI) until it is completed. This is fine when jobs complete quickly. But this can have an adverse effect if the current job is going to take a long time (even several hours) to complete.

In such cases, you can run the job in the **background** and free the shell for other tasks. The background job will be executed at lower priority, which, in turn, will allow smooth execution of the interactive tasks, and you can type other commands in the terminal window while the background job is running. By default all jobs are

executed in the foreground. You can put a job in the background by suffixing & to the command, for example: updatedb &

You can either use **CTRL-Z** to suspend a foreground job or **CTRL-C** to terminate a foreground job and can always use the **bg** and **fg** commands to run a process in the background and foreground, respectively.

Managing Jobs

```
test2@0penSUSE:~> jobs
[1]-  Stopped                 sleep 100
[2]+  Stopped                 sleep 100
test2@0penSUSE:~> jobs -l
[1]- 23375 Stopped                 sleep 100
[2]+ 23392 Stopped                 sleep 100
test2@0penSUSE:~>
```

The **jobs** utility displays all jobs running in background. The display shows the job ID, state, and command name, as shown here.

jobs -l provides a the same information as jobs including the PID of the background jobs.

The background jobs are connected to the terminal window, so if you log off, the **jobs** utility will not show the ones started from that window.

Tick the correct statements related to '**CTRL-Z**'.

- [x] It is used to suspend the foreground processes.
- [ ] It is used to suspend the background processes.
- [ ] It is used to reinitiate the foreground processes.
- [ ] It is used to reinitiate the background processes.

Which command is used to bring the process to the foreground?

- [ ] bg
- [ ] Sleep
- [x] fg
- [ ] &

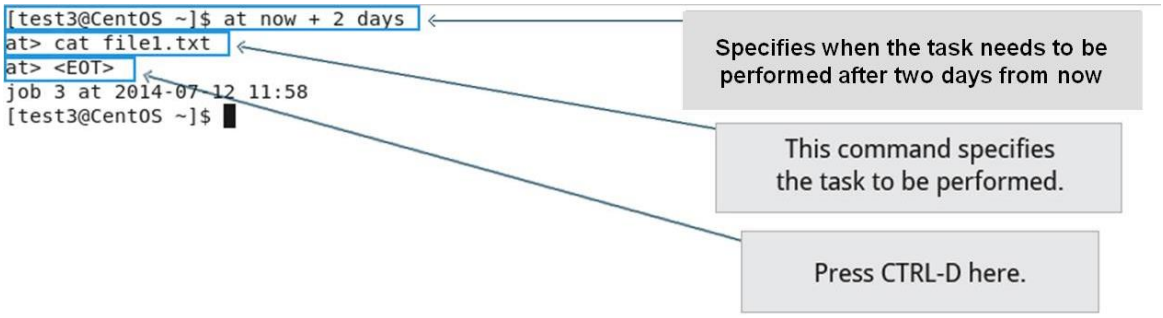
Which command is used to view the background processes in the current terminal?

- [ ] sleep
- [ ] fg
- [x] jobs
- [ ] bg

_____ shortcut is used to terminate a foreground process.

- [x] CTRL-C
- [ ] CTRL-Z
- [ ] ALT-C
- [ ] ALT-Z

Scheduling Future Processes using at



Suppose you need to perform a task on a specific day sometime in the future. However, you know you will be away from the machine on that day. How will you perform the task? You can use the **at** utility program to execute any non-interactive command at a specified time, as illustrated in the diagram:

cron

cron is a time-based scheduling utility program. It can launch routine background jobs at specific times and/or days on an on-going basis. **cron** is driven by a configuration file called /etc/crontab (**cron** table) which contains the various shell commands that need to be run at the properly scheduled times. There are both system-wide crontab files and individual user-based ones. Each line of a crontab file represents a job, and is composed of a so-called CRON expression, followed by a shell command to execute.

The crontab -e command will open the crontab editor to edit existing jobs or to create new jobs. Each line of the crontab file will contain 6 fields:

Field	Description	Values
MIN	Minutes	0 to 59
HOUR	Hour field	0 to 23
DOM	Day of Month	1-31
MON	Month field	1-12
DOW	Day Of Week	0-6 (0 = Sunday)
CMD	Command	Any command to be executed

Examples:
1. The entry "* * * * * /usr/local/bin/execute/this/script.sh" will schedule a job to execute 'script.sh' every minute of every hour of every day of the month, and every month and every day in the week.

2. The entry "30 08 10 06 * /home/sysadmin/full-backup" will schedule a full-backup at 8.30am, 10-June irrespective of the day of the week.

sleep

```
test2@OpenSUSE:~> cat sleep.sh
#!/bin/bash
echo "The system will go to sleep mode for 15 seconds..."
sleep 15
echo "Hi, I am awake."
test2@OpenSUSE:~> ./sleep.sh
The system will go to sleep mode for 15 seconds...
Hi, I am awake.
test2@OpenSUSE:~>
```

Sometimes a command or job must be delayed or suspended. Suppose, for example, an application has read and processed the contents of a data file and then needs to save a report on a backup system. If the backup system is currently busy or not available, the application can be made to **sleep** (wait) until it can complete its work. Such a delay might be to mount the backup device and prepare it for writing.

sleep suspends execution for at least the specified period of time, which can be given as the number of seconds (the default), minutes, hours or days. After that time has passed (or an interrupting signal has been received) execution will resume.

Syntax:
sleep NUMBER[SUFFIX]...
where SUFFIX may be:
1. s for seconds (the default)
2. m for minutes
3. h for hours
4. d for days

sleep and **at** are quite different; **sleep** delays execution for a specific period while **at** starts execution at a later time.

Which facility is used to schedule a periodically performed task?

cron

The **at** utility is used to execute_____.

- interactive programs at a specific time
- non-interactive programs at a specific time
- interactive programs at periodic time intervals
- non-interactive programs at periodic time intervals

Labs

Click the link to download a PDF from the Lab activity.

Processes_Labs

Processes_Solutions

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- Processes are used to perform various tasks on the system.
- Processes can be single-threaded or multi-threaded.
- Processes can be of different types such as interactive and non-interactive.
- Every process has a unique identifier (PID) to enable the operating system to keep track of it.
- The **nice value**, or **niceness**, can be used to set priority.
- **ps** provides information about the currently running processes.

- You can use **top** to get constant real-time updates about overall system performance as well as information about the processes running on the system.
- Load average indicates the amount of utilization the system is under at particular times.
- Linux supports background and foreground processing for a job.
- **at** executes any non-interactive command at a specified time.
- **cron** is used to schedule tasks that need to be performed at regular intervals.

Chapter 18: Common Applications

Learning Objectives

By the end of this chapter, you should be familiar with common Linux applications, including:

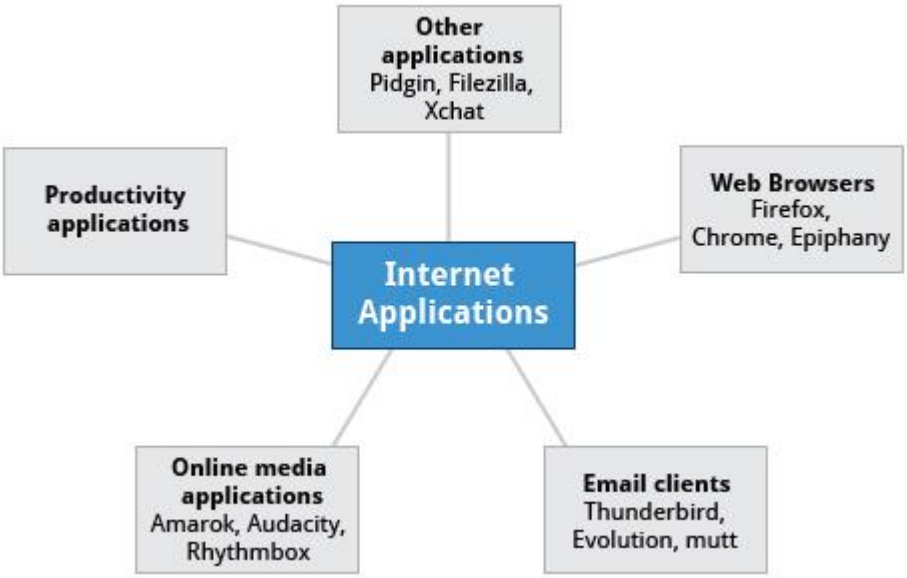
- Internet applications such as browsers, and email programs.
- Office Productivity Suites such as **LibreOffice**.
- Developer tools, such as compilers, debuggers, etc.
- Multimedia applications, such as those for audio and video.
- Graphics editors such as the **GIMP** and other graphics utilities.

Section 1: Internet Applications

Internet Applications

The Internet is a global network that allows users around the world to perform multiple tasks such as searching for data, communicating through emails and online shopping. Obviously, you need to use network-aware applications to take advantage of the Internet. These include:

- Web browsers
- Email clients
- Online media applications
- Other applications



Web Browsers

As discussed in the earlier chapter on Network Operations, Linux offers a wide variety of web browsers, both graphical and text based, including:

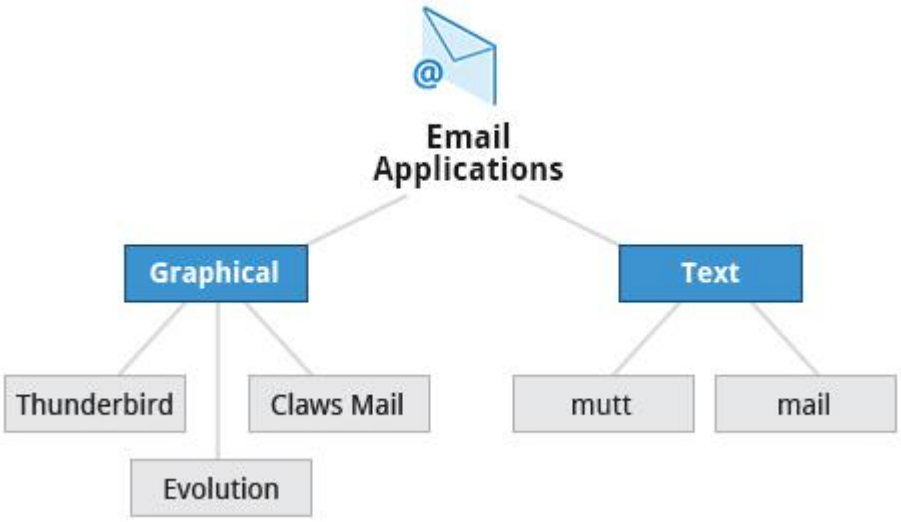
- **Firefox**
- **Google Chrome**
- **Chromium**
- **Epiphany**
- **Konqueror**
- **w3m**
- **lynx**

Email Applications

Email applications allow for sending, receiving, and reading messages over the Internet. Linux systems offer a wide number of **email clients**, both graphical and text-based. In addition many users simply use their browsers to access their email accounts.

Most email clients use the **Internet Message Access Protocol (IMAP)** or the older **Post Office Protocol (POP)** to access emails stored on a remote mail server. Most email applications also display **HTML**

(**HyperText Markup Language**) formatted emails that display objects, such as pictures and hyperlinks. The features of advanced email applications include the ability of importing address books/contact lists, configuration information, and emails from other email applications.



Linux supports the following types of email applications:

- Graphical email clients, such as **Thunderbird** (produced by **Mozilla)**, **Evolution**, and **Claws Mail**
- Text mode email clients such as **mutt** and **mail**

Other Internet Applications

Linux systems provide many other applications for performing Internet-related tasks. These include:

Application	Use
FileZilla	Intuitive graphical **FTP** client that supports **FTP**, **Secure File Transfer Protocol (SFTP)**, and **FTP Secured (FTPS).** Used to transfer files to/from **(FTP)** servers.
Pidgin	To access **GTalk**, **AIM**, **ICQ**, **MSN**, **IRC** and other messaging networks
Ekiga	To connect to **Voice over Internet Protocol (VoIP)** networks
XChat	To access **Internet Relay Chat (IRC)** networks

Which two of the following are common protocols used to access emails stored on a remote mail server?

- [ ] **mutt**
- [x] **POP**
- [ ] **mail**
- [x] **IMAP**

Which two of the following are text mode email clients available in Linux?

- [x] **mutt**
- [ ] **Evolution**
- [x] **mail**
- [ ] **Claws Mail**

What is **Ekiga** used for?

- (•) To connect to **VoIP**
- ( ) To access **IRC**
- ( ) To connect to **FTP** Servers
- ( ) To access **GTalk**

Office Applications

LibreOffice

OpenOffice.org

Most day-to-day computer systems have **productivity applications** (sometimes called **office suites**) available or installed (click here for a list of commonly used suites). Each suite is a collection of closely coupled programs used to create and edit different kinds of files such as:

- Text (articles, books, reports etc.)
- Spreadsheet
- Presentation
- Graphical objects

Most Linux distributions offer **LibreOffice**, an open source office suite that started in 2010 and has evolved from **OpenOffice.org.** While other office suites are available as we have listed, **LibreOffice** is the most mature, widely used and intensely developed.

The component applications included in **LibreOffice** are:

Components of LibreOffice	Usage
Writer	Word processing
Calc	Spreadsheets
Impress	Presentations
Draw	Create and edit graphics and diagrams

Development Applications

Linux distributions come with a complete set of applications and tools that are needed by those developing or maintaining both user applications and the kernel itself.

These tools are tightly integrated and include:

- Advanced editors customized for programmers' needs, such as **vi** and **emacs**.
- Compilers (such as **gcc** for programs in **C** and **C++**) for every computer language that has ever existed.
- Debuggers such as **gdb** and various graphical front ends to it and many other debugging tools (such as **valgrind**).
- Performance measuring and monitoring programs, some with easy to use graphical interfaces, others more arcane and meant to be used only by serious experienced development engineers.
- Complete Integrated Development Environments (IDE's) such as **Eclipse**, that put all these tools together.

On other operating systems these tools have to be obtained and installed separately, often at a high cost, while on Linux they are all available at no cost through standard package installation systems.

What is the use of the **Impress** component in **LibreOffice?**

- Word processing
- Spreadsheets
- Create and edit graphics and diagrams

Sound Players

Multimedia applications are used to listen to music, view videos, etc, as well as to present and view text and graphics. Linux systems offer a number of **sound player** applications including:

Application	Use
Amarok	Mature **MP3** player with a graphical interface, that plays audio and video files, and streams (online audio files). It allows you to create a play list that contains a group of songs, and uses a database to store information about the music collection.
Audacity	Used to record and edit sounds and can be quickly installed through a package manager. **Audacity** has a simple interface to get you started.
Rhythmbox	Supports a large variety of digital music sources including streaming Internet audio and podcasts. The application also enables search of particular audio in a library. It supports 'smart playlists' with an 'automatic update' feature which can revise playlists based on specified selection criteria.

Of course Linux systems can also connect with commercial online music streaming services such as **Pandora** and **Spotify** through web browsers.

Movie Players

Movie (video) **players** can portray input from many different sources, either local to the machine or on the Internet.

Linux systems offer a number of movie players including:

- **VLC**
- **MPlayer**
- **Xine**
- **Totem**

Movie Editors

Movie editors are used to edit videos or movies. Linux systems offer a number of movie editors including:

Application	Use
Kino	Acquire and edit camera streams. **Kino** can merge and separate video clips.
Cinepaint	Frame-by-frame retouching. **Cinepaint** is used for editing images in a video.
Blender	Create 3D animation and design. **Blender** is a professional tool that uses modeling as a starting point. There are complex and powerful tools for camera capture, recording, editing, enhancing and creating video, each having its own focus.
Cinelerra	Capture, compose, and edit audio/video.
FFmpeg	Record, convert, and stream audio/video. **FFmpeg** is a format converter, among other things, and has other tools such as **ffplay** and **ffserver**.

GIMP (GNU Image Manipulation Program)

Graphic editors allow you to create, edit, view, and organize images of various formats like Joint Photographic Experts Group (JPEG or JPG), Portable Network Graphics (PNG), Graphics Interchange Format (GIF), and Tagged Image File Format (TIFF).

GIMP (**GNU Image Manipulation Program**) is a feature-rich image retouching and editing tool similar to **Adobe Photoshop** and is available on all Linux distributions. Some features of the **GIMP** are:

- It can handle any image file format.
- It has many special purpose plugins and filters.
- It provides extensive information about the image, such as layers, channels, and histograms.

Graphics Utilities

In addition to the **GIMP**, there are other graphics utilities that help perform various image-related tasks, including:

Graphic Utility	Use
eog	**Eye of Gnome (eog)** is an image viewer that provides slide show capability and a few image editing tools, such as rotate and resize. It can also step through the images in a directory with just a click.
Inkscape	**Inkscape** is an image editor with lots of editing features. It works with layers and transformations of the image. It is sometimes compared to **Adobe FrameMaker.**
convert	**convert** is a command line tool (part of the **ImageMagick** set of applications) that can modify image files in many ways. The options include file format conversion and numerous image modification options, such as blur, resize, despeckle, etc.
Scribus	**Scribus** is used for creating documents used for publishing and providing a *What You See Is What You Get (WYSIWYG)* environment. It also provides numerous editing tools.

Labs

As a lab activity for this chapter, please install and explore various applications that are of interest to you (you don't need to limit yourself to the programs mentioned in this Chapter).

Summary

You have completed this chapter. Let's summarize the key concepts covered:

- Linux offers a wide variety of Internet applications such as web browsers, email clients, online media applications, and others.
- Web browsers supported by Linux can be either graphical or text-based such as **Firefox**, **Google Chrome**, **Epiphany**, **w3m, lynx** and others.
- Linux supports graphical email clients, such as **Thunderbird**, **Evolution**, and **Claws Mail**, and text mode email clients, such as **mutt** and **mail**.
- Linux systems provide many other applications for performing Internet-related tasks, such as **Filezilla**, **XChat**, **Pidgin**, and others.
- Most Linux distributions offer **LibreOffice** to create and edit different kinds of documents.

- Linux systems offer entire suites of development applications and tools, including compilers and debuggers.
- Linux systems offer a number of **sound players** including **Amarok**, **Audacity**, and **Rhythmbox**.
- Linux systems offer a number of movie players including **VLC**, **MPlayer**, **Xine**, and **Totem**.
- Linux systems offer a number of movie editors including **Kino**, **Cinepaint**, **Blender** among others.

- The **GIMP** (**GNU Image Manipulation Program**) utility is a feature-rich image retouching and editing tool available on all Linux distributions.
- Other graphics utilities that help perform various image-related tasks are **eog**, **Inkscape**, **convert**, and **Scribus**.

Final Exams

Question 1 (1/1 point)

The kernel is defined as:

- The graphical user interface on top of the operating system
- (•) The glue between the hardware and applications
- The software libraries needed to run the system
- All of the above

Question 2 (1/1 point)

Which statements are true about the Linux Boot Loader:

- It is executed before Linux
- It gives us more control on how we boot the system
- It can be used to select operating systems or kernels
- (•) All of the above

Question 3 (1/1 point)

If the display manager is not started by default in the default runlevel, you can start the X-Window System by:

- (•) Running the startx command from the command-line
- Running the xstart command from the command-line
- Running the runX command from the command-line
- Running the X –begin command from the command-line

Question 4 (1/1 point)

To change your desktop background:

- Edit the desktop-background file
- (•) Right-click on the desktop, select Change Desktop Background and select a wallpaper
- Double-click the desktop, select an available wallpaper and click on Save
- Try a different Linux distribution

Question 5 (1/1 point)

For best coordination between multiple Linux systems, you should set the date and time:

- Manually as close as possible to each other
- Manually, but the various systems' time only needs to be synchronized within a few minutes
- (•) Using the Network Time Protocol (NTP) so that machines are automatically very close to the same time

- Manually every time you boot the system

Question 6 (1/1 point)

Display settings allow you to change things like:

- The date and time
- Network settings
- Screen resolution and multiple screens configuration
- Whether or not to use a graphical interface

Question 7 (1/1 point)

One of the reasons you should learn text mode operations is:

- Your server may not have the overhead for a graphical user interface (GUI)
- Linux does not include a graphical user interface (GUI)
- A particular command may not be installed on your system
- You are not competent if you can't do everything from a command line

Question 8 (1/1 point)

To move around the file system tree from one directory to another you can use:

- **ls**
- **cd**
- **ps**
- **mv**

Question 9 (1/1 point)

To remove a directory named **data** and all of its files, including sub-directories, in that directory, use:

- **rmdir data**
- **rm –f data**
- **rm –rf data**
- **remove data**

Question 10 (1/1 point)

The Linux manual pages can be read using:

- **manual**
- **man**
- **doc**
- **command /?**

Question 11 (1/1 point)

Many programs have built-in help; to access this, use:

- (•) The -**h** or --**help** option
- ( ) The --**info** option
- ( ) The -**m** or --**man** option
- ( ) The –**doc** option

Question 12 (1/1 point)

Which of the following compares two files line by line and reports differences?

- ( ) **cp**
- ( ) **rsync**
- ( ) **patch**
- (•) **diff**

Question 13 (1/1 point possible)

Which of the following is useful for creating a backup of a disk Master Boot Record (MBR)?

- ( ) **d2d**
- (•) **dd**
- ( ) **bzip2**
- ( ) **file**

Question 14 (1/1 point)

In order to execute a Linux command as the root user, what should you prefix it with?

- ( ) **su** (as in **su ls /etc**)
- ( ) **time** (as in **time ls /etc**)
- ( ) **strace** (as in **strace ls /etc**)
- (•) **sudo command** (as in **sudo ls /etc**)

Question 15 (1/1 point)

___environment variable lists the directories in which the shell looks for executable programs and scripts.

- (•) $PATH
- ( ) $HOME
- ( ) $PS1
- ( ) $SHELL

Question 16 (1/1 point)

Which command is used to set the Linux file permissions to make the file named **data**, readable and writable by the owner and group owner of the file, but not accessible by any others?

- **chown 660 data**
- **chmod 660 data**
- **chgrp 670 data**
- **chmod 777 data**

Question 17 (1/1 point)

An easy to use text-based editor that utilizes on-screen prompts is:

- **edit**
- **cat**
- **edln**
- **nano**

Question 18 (1/1 point)

The command-mode key stroke(s)_____within **vi** will delete the characters from the cursor to the end of the word.

- D dw
- d$ rl

Question 19 (1/1 point)

Encrypted account passwords are found in the file,_____, and are typically encrypted using the_____algorithm.

- **/etc/passwd;** SHA-512
- **/etc/password;** SSH-512
- **/etc/shadow;** SHA-512
- **/etc/shadow;** HASH-256

Question 20 (1/1 point)

The root account:

- Has authority over the entire system
- Requires extreme caution when using because small mistakes can lead to disasters
- Should be used carefully; think before pressing the Enter key and run complex commands in a safe way, first, to ensure mistakes aren't made
- All of the above

Question 21 (1/1 point)

IP addresses are divided into two parts:

- Address and octets
- TCP and IP
- Network and host Network and host
- Class A and B

Question 22 (1/1 point)

To confirm that a remote host is online and responding and to measure network latency between machines at the same time , one can use:

- **nslookup**
- **ip addr show**
- **dig**
- **ping**

Question 23 (1/1 point)

How would you use **sed** to replace all the occurrences of the word "tiger" with "lion" in the file named **data**?

- **sed 's/tiger/lion/' data**
- **sed 's/lion/tiger/' data**
- **sed -i 's/tiger/lion/g' data**
- **sed –i 'r/lion/tiger/' data**

Question 24 (1/1 point)

The command "**wc –l data**" will print out the number of :

- lines in the file **data**
- words in the file **data**
- characters in the file **data**
- bytes in the file **data**

Question 25 (1/1 point)

Which of following could you use to check the status of all your printers?

- **status –a**
- **lp status –all**
- **lprm my-printers**
- **lpstat –a**

Question 26 (1/1 point)

Why should we use shell scripts?

- They combine long, possibly repetitive commands into one simple command
- They allow you to create new commands and share these procedures among several users
- They automate tasks and reduce risk of errors
- (•) All of the above

Question 27 (1/1 point)

Your shell script uses an "**if**" statement to see if the word "**yes**" was entered. This "**if**" statement looks like:

- **if ($RESPONSE = "yes") ; then**
- **if [RESPONSE = "yes") ; then**
- (•) **if [$RESPONSE == "yes"] ; then**
- **if (RESPONSE = "yes") ; then**

Question 28 (1/1 point)

Your shell script needs to test to see if a directory, **document**, exists; a way to do this is:

- **[-e document]**
- **[document –gt 0]**
- (•) **[-d document]**
- **[-s document]**

Question 29 (1/1point)

Which of the following can be used to show which processes are using most of the CPU or memory resources on your system:

- **most**
- **big**
- **lots**
- (•) **top**

Question 30 (1/1 point)

The ssh command provides a way to:

- Exit your login shell cleanly
- Silently copy files from one directory to another
- (•) Connect securely to remote systems Connect securely to remote systems
- Check the process status of your shell