SUMMARY

[**Chapter 1 - Introduction** 3](#_Toc179876813)

[**Role of the Kernel** 3](#_Toc179876814)

[**Applications** 4](#_Toc179876815)

[**Role of Open Source** 4](#_Toc179876816)

[**Linux Distributions** 5](#_Toc179876817)

[**Hardware Platforms** 6](#_Toc179876818)

[**Shell** 6](#_Toc179876819)

[**Bash Shell** 7](#_Toc179876820)

[**Accessing the Shell** 8](#_Toc179876821)

[**FileSystems** 9](#_Toc179876822)

[**Chapter 2 - Command Line Interface** 10](#_Toc179876823)

[**Commands** 10](#_Toc179876824)

[**External Commands** 10](#_Toc179876825)

[**Aliases** 11](#_Toc179876826)

[**Basic Command Syntax** 13](#_Toc179876827)

[**Specifying Arguments** 13](#_Toc179876828)

[**Quoting** 15](#_Toc179876829)

[**Options** 16](#_Toc179876830)

[**Scripts** 19](#_Toc179876831)

[**Displaying System Information** 19](#_Toc179876832)

[**Current Directory** 20](#_Toc179876833)

[**Command Completion** 22](#_Toc179876834)

[**Getting Help** 23](#_Toc179876835)

[**Viewing Man Pages** 23](#_Toc179876836)

[**Controlling the Man Page Display** 24](#_Toc179876837)

[**Sections Within Man Pages** 25](#_Toc179876838)

[**Searching Within a Man Page** 27](#_Toc179876839)

[**Man Page Sections** 27](#_Toc179876840)

[**Determining Which Section** 28](#_Toc179876841)

[**Specifying a Section** 28](#_Toc179876842)

[**Searching by Name** 29](#_Toc179876843)

[**Searching by Keyword** 29](#_Toc179876844)

[**Chapter 2 - LAB 2** 30](#_Toc179876845)

[**Chapter 6 - Filesystem Hierarchy Standard (FHS)** 31](#_Toc179876846)

[**Filesystem Hierarchy Standard** 31](#_Toc179876847)

[**Finding Files and Commands** 34](#_Toc179876848)

[locate Command 34](#_Toc179876849)

[find Command 36](#_Toc179876850)

[whereis Command 41](#_Toc179876851)

[which Command 43](#_Toc179876852)

[type Command 44](#_Toc179876853)

[**Chapter 6 - LAB 6** 46](#_Toc179876854)

[**Chapter 3** 47](#_Toc179876855)

# **Chapter 1 - Introduction**

The definition of the word *Linux* depends on the context in which it is used.

Technically speaking, Linux is the *kernel* of the system, which is the central controller of everything that happens on the computer. People that say their computer “runs Linux” are usually referring to the kernel and suite of tools that come with it (called a *distribution*). If someone says they have Linux experience, it might refer to configuring systems, running web servers, or any number of other services and programs that operate on top of Linux. Over time, Linux administration has evolved to encompass just about every task that a modern business, educational, or government institution might use in their daily operations.

What about *UNIX*? UNIX was originally an operating system developed at AT&T Bell Labs in the 1970s. It has been modified and *forked* (that is, people modified it, and those modifications served as the basis for other systems) such that now there are many different variants of UNIX. However, UNIX is now both a trademark and a specification, owned by an industry consortium called the Open Group. Only software that has been certified by the Open Group may call itself UNIX. Despite adopting most if not all of the requirements of the UNIX specification, Linux has not been certified, so Linux really isn’t UNIX! It’s just… UNIX-like.

## **Role of the Kernel**

The three main components of an operating system are the kernel, shell, and filesystem. The kernel of the operating system is like an air traffic controller at an airport. The kernel dictates which program gets which pieces of memory, it starts and kills programs, it interprets instructions given to it by the user, and it handles more common and simple tasks such as displaying text on a monitor. When an application needs to write to disk, it must ask the kernel to complete the write operation.

The kernel also handles the switching of applications. A computer will have one or more CPUs and a finite amount of memory. The kernel takes care of unloading tasks and loading new tasks, and can manage multiple tasks across multiple CPUs. When the current task has run a sufficient amount of time, the CPU pauses the task so that another may run. This is called *preemptive multitasking*. Multitasking means that the computer is doing several tasks at once, and preemptive means that the kernel is deciding when to switch focus between tasks. With the tasks so rapidly switching, it appears that the computer is doing many things at once.

Each application may think it has a large block of memory on the system, but it is the kernel that maintains this illusion; remapping smaller blocks of memory, sharing blocks of memory with other applications, or even swapping out blocks that haven’t been used in a while to the disk.

‌⁠​​⁠​ When the computer starts up, it loads a small piece of code called a *bootloader*. The bootloader’s job is to give you a choice (if configured) of options to load one or more versions of Linux, or even other operating systems, and then to load the kernel of the chosen option and get it started. If you are more familiar with operating systems such as Microsoft Windows or Apple’s OS X, you probably never see the bootloader, but in the UNIX world, it’s usually visible so that you can adjust the way your computer boots.

The bootloader loads the Linux kernel and then transfers control. Linux then continues with running the programs necessary to make the computer useful, such as connecting to the network or starting a web server.

## **Applications**

Like an air traffic controller, the kernel is not useful without something to control. If the kernel is the tower, the applications are the airplanes. Applications make requests to the kernel and receive resources, such as memory, CPU, and disk, in return. The kernel also abstracts the complicated details away from the application. The application doesn’t know if the block of a disk is on a solid-state drive from manufacturer A, a spinning metal hard drive from manufacturer B, or even a network file share. Applications just follow the kernel’s *Application Programming Interface (API)* and in return don’t have to worry about the implementation details.

When we, as users, think of applications, we tend to think of word processors, web browsers, and email clients. The kernel doesn’t care if it is running something that’s user-facing, a network service that talks to a remote computer, or an internal task. So, from this, we get an abstraction called a *process*. A process is just one task that is loaded and tracked by the kernel. An application may even need multiple processes to function, so the kernel takes care of running the processes, starting and stopping them as requested, and handing out system resources.

## **Role of Open Source**

Software projects take the form of *source code*, which is a human-readable set of computer instructions. The source code may be written in any of hundreds of different programming languages. The Linux kernel is mostly written in C, which is a language that shares history with the original UNIX.

Source code is not understood directly by the computer, so it must be compiled into machine instructions by a *compiler*. The compiler gathers all of the source files and generates something that can be run on the computer, such as the Linux kernel.

Historically, most software has been issued under a *closed-source license*, meaning that you get the right to use the machine code, but cannot see the source code. Often the license specifically says that you will not attempt to reverse engineer the machine code back to source code to figure out what it does!

*Open source* takes a source-centric view of software. The open-source philosophy is that you have a right to obtain the software and to modify it for your own use. Linux adopted this philosophy to great success.

In 1991, Linux started as a hobby project by Linus Torvalds. He made the source freely available, allowing others to join in and shape this fledgling operating system. It was not the first system to be developed by a volunteer group, but since it was built from scratch, early adopters could influence the project’s direction. People took the source, made changes, and shared them back with the rest of the group, greatly accelerating the pace of development, and ensuring mistakes from other operating systems were not repeated.

The Linux kernel is licensed under the GNU Public License (GPL) which requires you to make changes available. This guarantees that those who use the code will also contribute to the greater good by making those changes available to anyone.

Alongside this, was the *GNU project* (GNU’s, not UNIX). While GNU (pronounced "guh-noo”) was building their own operating system, they were far more successful at building the tools that go along with a UNIX operating system, such as the compilers and user interfaces. The source was all freely available, so Linux was able to target their tools and provide a complete system. As such, most of the tools that are part of the Linux system come from these GNU tools.

There are many different variants on open source. However, all agree that you should have access to the source code, but they differ in how you can, or in some cases, must, redistribute changes.

## **Linux Distributions**

Take the Linux kernel and the GNU tools, add some more user-facing applications like an email client, word processors and other programs and you have a full Linux system. People started bundling all this software into a *distribution* almost as soon as Linux became usable. The distribution takes care of setting up the storage, installing the kernel, and installing the rest of the software. The full-featured distributions also include tools to manage the system and a *package manager* to help you add and remove software after the installation is complete.

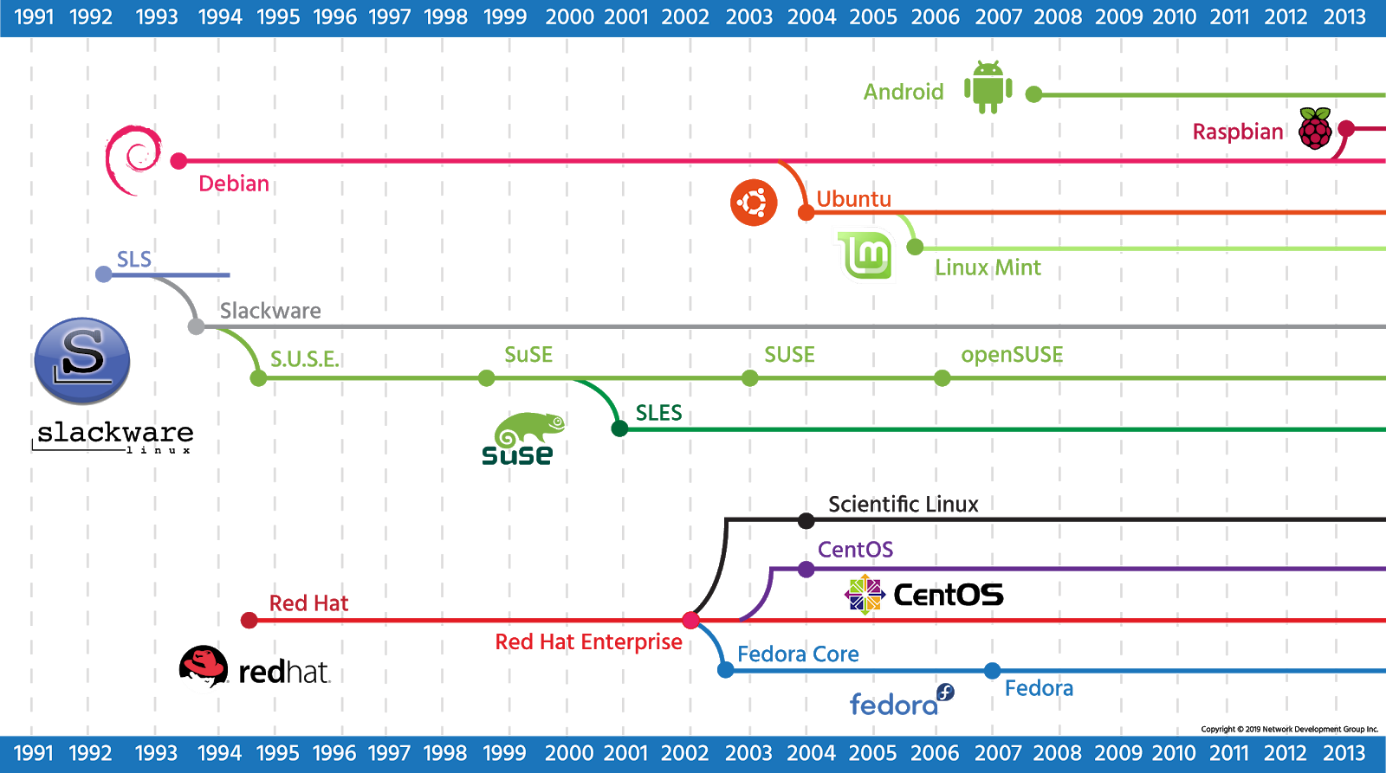
Like UNIX, there are many different flavors of distributions. These days, there are distributions that focus on running servers, desktops, or even industry-specific tools like electronics design or statistical computing. The major players in the market can be traced back to either **Red Hat** or **Debian**. The most visible difference is the software package manager, though you will find other differences on everything from file locations to political philosophies.

Red Hat started out as a simple distribution that introduced the Red Hat Package Manager (RPM) based on the .rpm file format. The developer eventually formed a company around it, which tried to commercialize a Linux desktop for business. Over time, Red Hat started to focus more on the server applications such as web and file serving and released Red Hat Enterprise Linux, which was a paid service on a long *release cycle*. The release cycle dictates how often software is upgraded. A business may value stability and want long release cycles, while a hobbyist or a startup may want the latest software and opt for a shorter release cycle. To satisfy the latter group, Red Hat sponsors the **Fedora Project** which makes a personal desktop comprising the latest software but still built on the same foundations as the enterprise version.

Because everything in Red Hat Enterprise Linux is open source, a project called **CentOS** came to be, that recompiled all the RHEL packages and gave them away for free. CentOS and others like it (such as **Scientific Linux**) are largely compatible with RHEL and integrate some newer software, but do not offer the paid support that Red Hat does.

**Debian** is more of a community effort, and as such, also promotes the use of open-source software and adherence to standards. Debian came up with its own package management system based on the .deb file format. While Red Hat leaves non-Intel and AMD platform support to derivative projects, Debian supports many of these platforms directly.

**Ubuntu** is the most popular Debian-derived distribution. It is the creation of **Canonical**, a company that was made to further the growth of Ubuntu and makes money by providing support.



# **Hardware Platforms**

Linux started out as something that would only run on a computer like Linus’: a 386 with a specific hard drive controller. The range of support grew, as support for other hardware was built. Eventually, Linux started supporting other chipsets, including hardware that was made to run competitive operating systems!

The types of hardware grew from the humble Intel chip up to supercomputers. Smaller sized Linux-supported chips were eventually developed to fit in consumer devices (called embedded devices). The support for Linux became ubiquitous such that it is often easier to build hardware to support Linux and then use Linux as a springboard for your custom software than it is to build the custom hardware and software from scratch.

Eventually, cellular phones and tablets adopted Linux. A company, later bought by Google, came up with the Android platform which is a bundle of Linux and the software necessary to run a phone or tablet. This means that the effort to get a phone to market is significantly less. Instead of long developments on a new operating system, companies can spend their time innovating on the user-facing software. Android is now one of the market leaders in the phone and tablet space.

Aside from phones and tablets, Linux can be found in many consumer devices. Wireless routers often run Linux because it has a rich set of network features. TiVo is a consumer digital video recorder built on Linux. Even though these devices have Linux at the core, the end users don’t have to know how to use Linux. The custom software interacts with the user and Linux provides a stable platform.

# **Shell**

An operating system provides at least one *shell* that allows the user to communicate with the operating system. A shell is sometimes called an *interpreter* because it takes the commands that a user issues and interprets them into a form that the *kernel* can then execute on the hardware of the computer. The two most common types of shells are the Graphical User Interface (GUI) and Command Line Interface (CLI).

Microsoft Windows™ typically uses a GUI shell, primarily using the mouse to indicate what you want to accomplish. While using an operating system in this way might be considered easy, there are many advantages to using a CLI, including:

* **Command Repetition**: In a GUI shell, there is no easy way to repeat a previous command. In a CLI there is an easy way to repeat (and modify) a previous command.
* **Command Flexibility**: The GUI shell provides limited flexibility in the way the command executes. In a CLI, *options* are specified with commands to provide a more flexible and powerful interface.
* **Resources**: A GUI shell typically uses a relatively large number of resources (RAM, CPU, etc.). This is because a great deal of processing power and memory is needed to display graphics. By contrast, CLI uses very little system resources, allowing more of these resources to be available to other programs.
* **Scripting**: In a GUI shell, completing multiple tasks often requires multiple mouse clicks. With a CLI, a *script* can be created to execute many complex operations by just typing the name of the script. A script is a series of commands placed into a single file. When executed, the script runs all of the commands in the file.
* **Remote Access**: While it is possible to execute commands in a GUI shell remotely, this feature isn't typically set up by default. With a CLI shell, gaining access to a remote machine is easy and typically available by default.
* **Development**: Normally a GUI-based program takes more time for the developers to create when compared to CLI-based programs. As a result, there are usually thousands of CLI programs on a typical Linux OS compared to only a couple hundred programs in a primarily GUI-based OS like Microsoft Windows. More programs mean more power and flexibility.

The Microsoft Windows operating system was designed to primarily use the GUI interface because of its simplicity, although there are several CLI interfaces available, too. For simple commands, there is the Run dialog box, where you can type or browse to the commands that you want to execute. If you want to type multiple commands or if you want to see the output of the command, you can use Command Prompt, also called the DOS shell. Recently, Microsoft realized how important it is to have a powerful command line environment and introduced PowerShell.

Like Windows, Linux also has both a CLI and GUI. Unlike Windows, Linux lets you easily change the GUI shell (also called the desktop environment) that you want to use. The two most common desktop environments for Linux are GNOME and KDE; however, there are many other GUI shells available.

To access the CLI from within the GUI on a Linux operating system, the user can open a software program called a *terminal*. Linux can also be configured only to run the CLI without the GUI; this is typically done on servers that don't require a GUI, primarily to free up system resources.

## **Bash Shell**

Not only does the Linux operating system provide multiple GUI shells, but also multiple CLI shells are available. Normally, these shells are derived from one of two older UNIX shells: the Bourne Shell and the C Shell. In fact, the Bash shell, a default

CLI shell used in modern Linux operating systems, derives its name from the Bourne Shell: **B**ourne **A**gain **Sh**ell. In this course, you will focus upon learning how to use the CLI for Linux with the Bash shell, arguably the most popular CLI in Linux.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteUsers interact with a system by executing *commands* which are interpreted by the shell and transformed into actions by the kernel. These actions may or may not return information to the command line depending on the command issued and its result. For example, when the ls command is typed into the console, it will return the contents of whichever directory the user is currently in.

Commands entered are considered standard input, (stdin) whether they are typed by an operator, entered by a script, or as the result of another command. Text returned to the console can be either standard output (stdout), or standard error (stderr).

This deceptively simple method of communicating with the Linux kernel is the basis for almost every interaction a Linux administrator has with their systems. It can be confusing at first for users who have only experienced GUI interfaces, but ultimately it gives the experienced operator far more power than any graphical interface can.

The Bash shell has numerous built-in commands and features that you will learn including:

* **Aliases**: Give a command a different or shorter name to make working with the shell more efficient.
* **Re-Executing Commands**: To save retyping long command lines.
* **Wildcard Matching**: Uses special characters like ?, \*, and [] to select one or more files as a group for processing.
* **Input/Output Redirection**: Uses special characters for redirecting input, < or <<, and output, >.
* **Pipes**: Used to connect one or more simple commands to perform more complex operations.
* **Background Processing**: Enables programs and commands to run in the background while the user continues to interact with the shell to complete other tasks.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteThe shell that your user account uses by default is set at the time your user account was created. By default, many Linux distributions use Bash for a new user's shell. Typically, a user learns one shell and sticks with that shell; however, after you have learned the basics of Linux, you may want to explore the features of other shells.

## **Accessing the Shell**

How you access the command line shell depends on whether your system provides a GUI login or CLI login:

* **GUI-based systems**: If the system is configured to present a GUI, then you will need to find a software application called a *terminal*. In the GNOME desktop environment, the terminal application can be started by clicking the Applications menu, then the System Tools menu and Terminal icon.
* **CLI-based systems**: Many Linux systems, especially servers, are not configured to provide a GUI by default, so they present a CLI instead. If the system is configured to present a CLI, then the system runs a terminal application automatically after you log in.

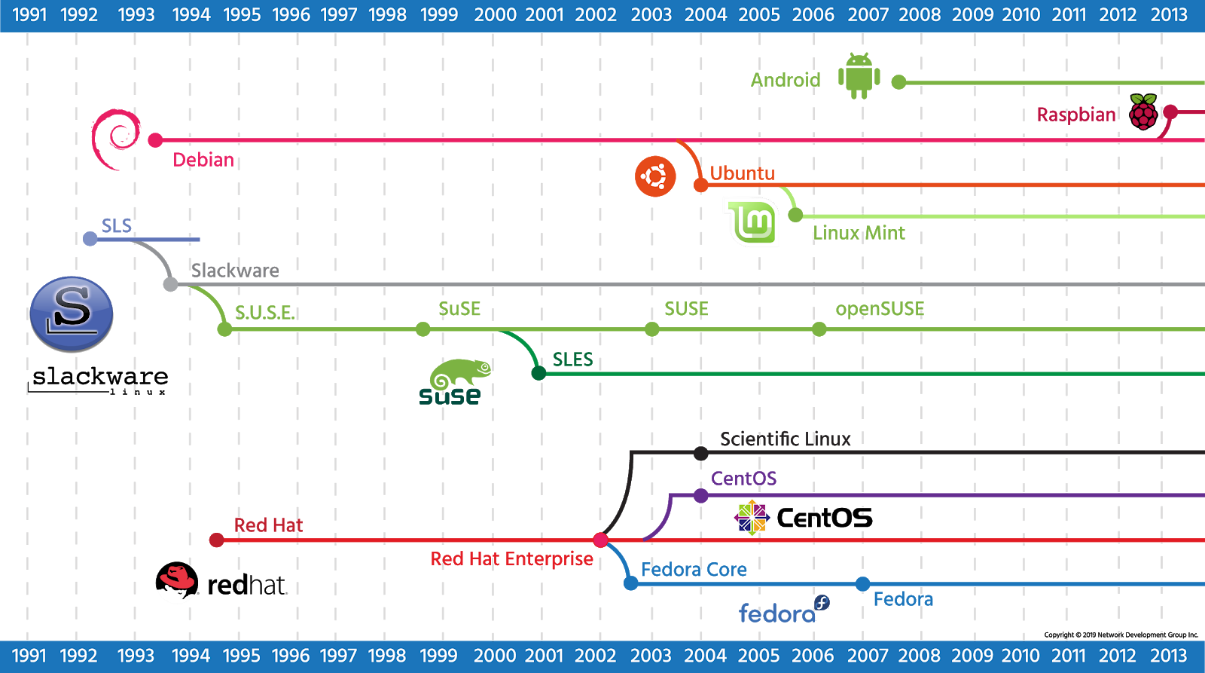
In the early days of computing, terminal devices were large machines that allowed users to provide input through a keyboard and displayed output by printing on paper. Over time, terminals evolved and their size shrank down into something that looked similar to a desktop computer with a video display monitor for output and a keyboard for input.

Ultimately, with the introduction of personal computers, terminals became *software emulators* of the actual hardware. Whatever you type in the terminal is interpreted by your shell and translated into a form that can then be executed by the kernel of the operating system.

If you are in a remote location, then *pseudo-terminal* connections can also be made across the network using several techniques. Insecure connections could be made using protocols such as telnet and programs such as rlogin, while secure connections can be established using programs like putty and protocols such as ssh.

# **FileSystems**

In addition to the kernel and the shell, the other major component of any operating system is the filesystem. To the user, a filesystem is a hierarchy of directories and files with the root / directory at the top of the directory tree. To the operating system, a filesystem is a structure created on a disk partition consisting of tables defining the locations of directories and files.



In this course, you will learn about the different Linux filesystems, filesystem benefits and how to create and manage filesystems using commands.

# **Chapter 2 - Command Line Interface**

Most consumer operating systems are designed to shield the user from the “ins and outs” of the CLI. The Linux community is different in that it positively celebrates the CLI for its power, speed, and ability to accomplish a vast array of tasks with a single command line instruction.

When a user first encounters the CLI, they can find it challenging because it requires memorizing a dizzying amount of commands and their options. However, once a user has learned the structure of how commands are used, where the necessary files and directories are located, and how to navigate the hierarchy of a file system, they can be immensely productive. This capability provides more precise control, greater speed, and the ability to automate tasks more easily through scripting.

Furthermore, by learning the CLI, a user can easily be productive almost instantly on ANY flavor or distribution of Linux, reducing the amount of time needed to familiarize themselves with a system because of variations in a GUI.

# **Commands**

What is a command? The simplest answer is that a command is a software program that when executed on the command line, performs an action on the computer.

When you consider a command using this definition, you are really considering what happens when you execute a command. When you type in a command, a process is run by the operating system that can read input, manipulate data, and produce output. From this perspective, a command runs a process on the operating system, which then causes the computer to perform a *job*.

‌⁠​However, there is another way of looking at what a command is: look at its *source*. The source is where the command "comes from" and there are several different sources of commands within the shell of your CLI:

* **Internal Commands**: Also called *built-in commands*, these commands are built-in to the shell itself. A good example is the cd (change directory) command as it is part of the Bash shell. When a user types the cd command, the Bash shell is already executing and knows how to interpret that command, requiring no additional programs to be started.
* **External Commands**: These commands are stored in files that are searched by the shell. If you type the ls command, then the shell searches through a predetermined list of directories to try to find a file named ls that it can execute. These commands can also be executed by typing the complete path to the command.
* **Aliases**: An alias can override a built-in command, function, or a command that is found in a file. Aliases can be useful for creating new commands built from existing functions and commands.
* **Functions**: Functions can also be built using existing commands to either create new commands, override commands built-in to the shell or commands stored in files. Aliases and functions are normally loaded from the initialization files when the shell first starts, discussed later in this section.

## **External Commands**

Commands that are stored in files can be in several forms that you should be aware of. Most commands are written in the C programming language, which is initially stored in a human-readable text file. These text source files are then *compiled* into computer-readable binary files, which are then distributed as the command files.

Users who are interested in seeing the source code of compiled, GPL licensed software can find it through the sites where it originated. GPL licensed code also compels distributors of the compiled binaries, such as RedHat and Debian, to make the source code available. Often it is found in the distributors’ repositories.

Immagine che contiene testo, elettronica, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata, numero

Descrizione generata automaticamente

## **Aliases**

An *alias* can be used to map longer commands to shorter key sequences. When the shell sees an alias being executed, it substitutes the longer sequence before proceeding to interpret commands.

For example, the command ls -l is commonly aliased to l or ll. Because these smaller commands are easier to type, it becomes faster to run the ls -l command line.

To determine what aliases are set on the current shell use the alias command:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

The aliases from the previous examples were created by initialization files. These files are designed to make the process of creating aliases automatic.

New aliases can be created using the following format, where *name* is the name to be given the alias and *command* is the command to be executed when the alias is run.



For example, the cal 2030 command displays the calendar for the year 2030. Suppose you end up running this command often. Instead of executing the full command each time, you can create an alias called mycal and run the alias, as demonstrated in the following graphic:

Immagine che contiene testo, schermata, Carattere, design

Descrizione generata automaticamente

Aliases created this way only persists while the shell is open. Once the shell is closed, the new aliases are lost. Additionally, each shell has its own aliases, so, aliases created in one shell won’t be available in a new shell that’s opened.

# **Basic Command Syntax**

To execute a command, the first step is to type the name of the command. Click in the terminal on the right. Type ls and hit **Enter**. The result should resemble the example below:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Many commands can be used by themselves with no further input. Some commands require additional input to run correctly. This additional input comes in two forms: *options* and *arguments*. Commands typically follow a simple pattern of syntax:



When typing a command that is to be executed, the first step is to type the name of the command. The name of the command is often based on what the command does or what the developer who created the command thinks will best describe the command's function.

For example, the ls command displays a *listing* of information about files. Associating the name of the command with something mnemonic for what it does may help you to remember commands more easily.

Keep in mind that every part of the command is normally case-sensitive, so LS is incorrect and will fail, but ls is correct and will succeed.

## **Specifying Arguments**



An *argument* can be used to specify something for the command to act upon. Following a command, any desired arguments are allowed or are required depending on the command. For example, the touch command is used to create empty files or update the timestamp of existing files. It requires at least one argument to specify the file name to act upon.

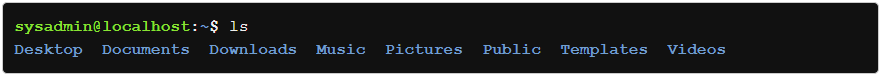
Immagine che contiene testo, schermata

Descrizione generata automaticamente

The ls command, on the other hand, allows for a path and/or file name to be specified as an argument, but it’s not required.



An example of a scenario where an argument is allowed but not required is the use of the ls command. If the ls command is used without an argument, it will list the contents of the current directory:



If the ls command is given the name of a directory as an argument, it will list the contents of that directory. In the following example, the /etc/ppp directory is used as an argument; the resulting output is a list of files contained in the /etc/ppp directory:



The ls command also accepts multiple arguments. To list the contents of both the /etc/ppp and /etc/ssh directories, pass them both as arguments:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Some commands, like the cp command (*copy* file) and the mv command (*move* file), always require at least two arguments: a source file and a destination file.



In the example below, we will copy the public ssh rsa key called ssh\_host\_rsa\_key.pub which resides in the /etc/ssh directory, to the /home/sysadmin/Documents directory and verify that it is there:

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

## **Quoting**

Arguments that contain unusual characters like spaces or non-alphanumeric characters will usually need to be *quoted*, either by enclosing them within double quotes or single quotes. *Double quotes* will prevent the shell from interpreting *some* of these special characters; *single quotes* prevent the shell from interpreting *any* special characters.

In most cases, single quotes are considered safer and should probably be used whenever you have an argument that contains characters that aren't alphanumeric. To understand the importance of quotes, consider the echo command. The echo command displays text to the terminal and is used extensively in shell scripting.



Consider the following scenario in which you want to list the contents of the current directory using the ls command and then use the echo command to display the string hello world!! on the screen.

You might first try the echo command without any quotes, unfortunately without success:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Using no quotes failed because the shell interprets the !! characters as special shell characters; in this case, they mean "replace the !! with the last command that was executed". In this case, the last command was the ls command, so ls replaced !! and then the echo command displayed hello worldls to the screen.

You may want to try the double quote " characters to see if they will block the interpretation (or expansion) of the exclamation !! characters. The double quotes block the expansion of some special characters, but not all of them. Unfortunately, double quotes do not block the expansion of the exclamation !! characters:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Using double quotes preserves the literal value of all characters that they enclose except metacharacters such as the $ dollar sign character, the ` backquote character, the \ backslash character and the ! exclamation point character. These characters, called *wild cards*, are symbol characters that have special meaning to the shell. Wild card characters are used for *globbing* and are interpreted by the shell itself before it attempts to run any command. Glob characters are useful because they allow you to specify patterns that make it easier to match file names in the command line. For example, the command ls e?? would list all files in that directory that start with an e and have any two characters after it. However, because glob characters are interpreted differently by the shell, they need to be enclosed appropriately to be interpreted literally.

If you enclose text within the ' single quote characters, then all characters have their literal meaning:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

## **Options**



*Options* can be used with commands to expand or modify the way a command behaves. If it is necessary to add options, they can be specified after the command name. *Short options* are specified with a hyphen - followed by a single character. Short options are how options were traditionally specified.

In the following example, the -l option is provided to the ls command, which results in a *long display* output:

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

Often the character is chosen to be mnemonic for its purpose, like choosing the letter *l for long* or *r for reverse*. By default, the ls command prints the results in alphabetical order, so adding the -r option prints the results in reverse alphabetical order.



In most cases, options can be used in conjunction with other options. They can be given as separate options like -l -r or combined like -lr. The combination of these two options would result in a long listing output in reverse alphabetical order:

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

Multiple single options can be either given as separate options like -a -l -r or combined like -alr. The output of all of these examples would be the same:



Generally, short options can be combined with other short options in any order. The exception to this is when an option requires an argument.

For example, the -w option to the ls command specifies the *width* of the output desired and therefore requires an argument. If combined with other options, the -w option can be specified last, followed by its argument and still be valid, as in ls -rtw 40, which specifies an output width of 40 characters. Otherwise, the -w option cannot be combined with other options and must be given separately.

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

If you are using multiple options that require arguments, don't combine them. For example, the -T option, which specifies *tab size*, also requires an argument. In order to accommodate both arguments, each option is given separately:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Some commands support additional options that are longer than a single character. *Long options* for commands are preceded by a double hyphen -- and the meaning of the option is typically the name of the option, like the --all option, which lists all files, including hidden ones. For example:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

For commands that support both long and short options, execute the command using the long and short options concurrently:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

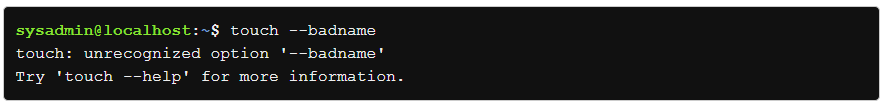
Commands that support long options will often also support arguments that may be specified with or without an equal symbol (the output of both commands is the same):

Immagine che contiene testo, Carattere, software, linea

Descrizione generata automaticamente

A special option exists, the *lone double hyphen* -- option, which can be used to indicate the end of all options for the command. This can be useful in some circumstances where it is unclear whether some text that follows the options should be interpreted as an additional option or as an argument to the command.

For example, if the touch command tries to create a file called --badname:



The command tries to interpret --badname as an option instead of an argument. However, if the lone double hyphen --' option is placed before the file name, indicating that there are no more options, then the file name can successfully be interpreted as an argument:

Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente

A third type of option exists for a select few commands. While the options used in the **AT&T** version of UNIX used a single hyphen and the GNU port of those commands used two hyphens, the **Berkeley Software Distribution (BSD)** version of UNIX used options with no hyphen at all.

This no hyphen syntax is fairly rare in most Linux distributions. A couple of notable commands that support the BSD UNIX style options are the ps and tar commands; both of these commands also support the single and double hyphen style of options.

In the terminal below, there are two similar commands, the first command is executed with a traditional UNIX style option (with single hyphens) and the second command is executed with a BSD style option (no hyphens).

Immagine che contiene testo, schermata, software, multimediale

Descrizione generata automaticamente

# **Scripts**

One exception to the basic command syntax used is the exec command, which takes another command to execute as an argument. What is special about the commands that are executed with exec is that they replace the currently running shell.

A common use of the exec command is in what is known as *wrapper scripts*. If the purpose of a script is to simply configure and launch another program, then it is known as a wrapper script.

A wrapper script often uses the following as the last line of the script to execute another program.



A script written this way avoids having a shell continue to run while the program that it launched is running, the result is that this technique saves resources (like RAM).

Although redirection of input and output to a script are discussed in another section, it should also be mentioned that the exec command can be used to cause redirection for one or more statements in a script.

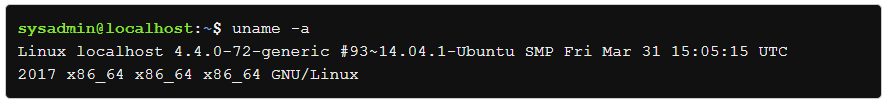
# **Displaying System Information**

The uname command displays system information. This command will output Linux by default when it is executed without any options.



The uname command is useful for several reasons, including when you need to determine the name of the computer as well as the current version of the kernel that is being used.

To display additional information about the system, you can use one of the many options available for the uname command. For example, to display *all* the information about the system, use the -a option with the uname command:



To display information about what kernel version the system is running, use the -r option:

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamenteThe options for the uname command are summarized below:

# **Current Directory**

One of the simplest commands available is the pwd command, which is an acronym for *print working directory*. When executed without any options, the pwd command will display the name of the directory where the user is currently located in the file system. When a user logs into a system, they are normally placed in their *home directory* where files they create and control reside. As you navigate around the file system it is often helpful to know what directory you’re in.



Notice our virtual machines employ a prompt that displays the current working directory, emphasized with the color blue. In the first prompt above, the blue tilde ~ character is equivalent to /home/sysadmin, representing the user's home directory:



After changing directories, the new location can also be confirmed in the new prompt by using the pwd command, and is again shown in blue:



To get back to the home directory after changing to a new location, use the cd *change directory* command without any arguments:



The type command displays information about a command type. For example, if you entered type ls at the command prompt, it will return that the ls command is actually an alias for the ls --color=auto command:



Using the -a option with the type command will return all locations of the files that contain a command; also called an *executable* file:



In the output above, the /bin/ls file path is the file location of the ls command.

This command is helpful for getting information about commands and where they reside on the system. For internal commands, like the pwd command, the type command will identify them as shell builtins:



For external commands like the ip command, the type command will return the location of the command, in this case, the /sbin directory:

Immagine che contiene testo, Carattere, software, Pagina Web

Descrizione generata automaticamente

If a command does not behave as expected or if a command is not accessible, that should be, it can be beneficial to know where the shell is finding the command.

The which command searches for the location of a command in the system by searching the PATH variable.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

# **Command Completion**

A useful tool of the Bash shell is the ability to complete commands and their arguments automatically. Like many command line shells, Bash offers command line completion, where you type a few characters of a command (or its file name argument) and then press the **Tab** key. The Bash shell will complete the command (or its file name argument) automatically for you. For example, if you type ech and press **Tab**, then the shell will automatically complete the command echo for you.

There will be times when you type a character or two and press the **Tab** key, only to discover that Bash does not automatically complete the command. This will happen when you haven't typed enough characters to match only one command. However, pressing the **Tab** key a second time in this situation will display the possible completions (possible commands) available.

A good example of this would be if you typed ca and pressed **Tab**; then nothing would be displayed. If you pressed **Tab** a second time, then the possible ways to complete a command starting with ca would be shown:

Immagine che contiene testo, software, Software multimediale, Carattere

Descrizione generata automaticamente

Another possibility may occur when you have typed too little to match a single command name uniquely. If there are more possible matches to what you've typed than can easily be displayed, then the system will use a *pager* to display the output, which will allow you to scroll through the possible matches.

For example, if you just type c and press the **Tab** key twice, the system will provide you with many matches that you can scroll through:

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

You may not wish to scroll through all these options; in that case, press **Q** to exit the pager. In a situation like this, you should probably continue to type more characters to achieve a more refined match.

A common mistake when typing commands is to misspell the command name. Not only will you type commands faster, but you will type more accurately if you use command completion. Using the **Tab** key to complete the command automatically helps to ensure that the command is typed correctly.

Note that completion also works for arguments to commands when the arguments are file or directory names.

# **Getting Help**

As previously mentioned, UNIX was the operating system from which the foundation of Linux was built. The developers of UNIX created help documents called *man* pages (short for *manual* page).

Referring to the man page for a command will provide you with the basic idea behind the purpose of the command, as well as details regarding the options of the command and a description of its features.

## **Viewing Man Pages**

To view a man page for a command, execute the man command in a terminal window.



For example, the following graphic shows the partial man page for the cal command:

Immagine che contiene testo, elettronica, schermata, software

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

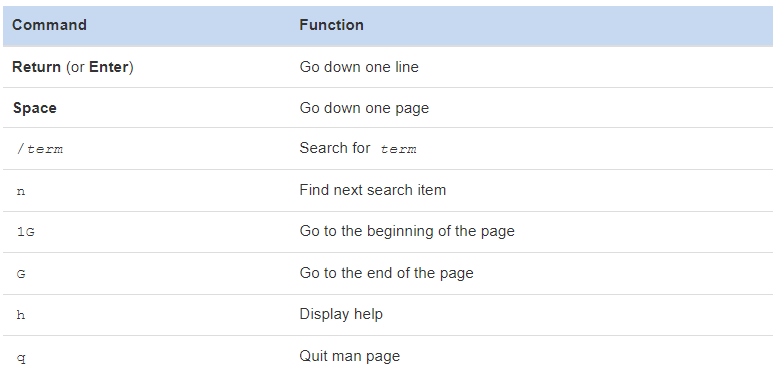
## **Controlling the Man Page Display**

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamenteThe man command uses a pager to display documents. Typically, this pager is the less command, but on some distributions, it may be the more command. Both are very similar in how they perform and will be discussed in more detail in a later chapter. To view the various movement commands that are available, use the **H** key or **Shift+H** while viewing a man page. This will display a help page.

To exit the SUMMARY OF LESS COMMANDS, type **Q**.

If your distribution uses the less command, you might be a bit overwhelmed with the large number of "commands" that are available. The following table provides a summary of the more useful commands:



## **Sections Within Man Pages**

Each man page is broken into sections. Each section is designed to provide specific information about a command. While there are common sections that you will see in most man pages, some developers also create sections that you will only see in a specific man page.

The following table describes some of the more common sections that you will find in man pages:

**NAME**

Provides the name of the command and a very brief description.



**SYNOPSIS**

A brief summary of the command or function's interface. A summary of how the command line syntax of the program looks.



**DESCRIPTION**

Provides a more detailed description of the command.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

**OPTIONS**

Lists the options for the command as well as a description of how they are used. Often this information is found in the DESCRIPTION section and not in a separate OPTIONS section.

Immagine che contiene testo, schermata, software

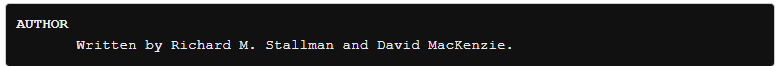
Descrizione generata automaticamente

**FILES**

Lists the files that are associated with the command as well as a description of how they are used. These files may be used to configure the command's more advanced features. Often this information is found in the DESCRIPTION section and not in a separate FILES section.

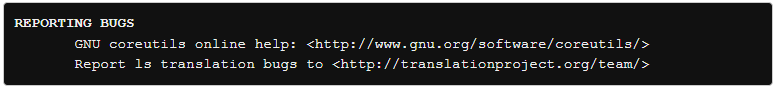
**AUTHOR**

Provides the name of the person who created the man page and (sometimes) how to contact the person.



**REPORTING BUGS**

Provides details on how to report problems with the command.



**COPYRIGHT**

Provides basic copyright information.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

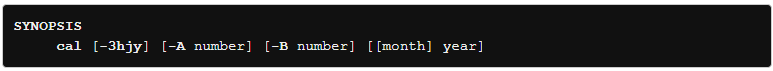
**SEE ALSO**

Provides you with an idea of where you can find additional information. This often includes other commands that are related to this command.



**Man Pages Synopsis**

The SYNOPSIS section of a man page can be difficult to understand, but it is a valuable resource since it provides a concise example of how to use the command. For example, consider an example SYNOPSIS for the cal command:



The square brackets [ ] are used to indicate that this feature is not required to run the command. For example, [-3hjy] means you can use the options -3, -h, -j, -y, but none of these options are required for the cal command to function properly.

The second set of square brackets [-A number] allows you to ‌⁠​​⁠specify the number of months to be added to the end of the display.

The third set of square brackets in the [-B number] allows you to specify the number of months to be added to the beginning of the display.

The fourth set of square brackets in the [[month] year] demonstrates another feature; it means that you can specify a year by itself, but if you specify a month you must also specify a year.

Another component of the SYNOPSIS that might cause some confusion can be seen in the SYNOPSIS of the date command:



In this SYNOPSIS there are two syntaxes for the date command. The first one is used to display the date on the system while the second is used to set the date.

The ellipses ... following [OPTION], indicate that [OPTION] may be repeated.

Additionally, the [-u|--utc|--universal] notation means that you can either use the -u option or the --utc option, or the --universal option. Typically, this means that all three options really do the same thing, but sometimes this format (use of the | character) is used to indicate that the options can't be used in combination, like a logical *or*.

## **Searching Within a Man Page**

In order to search a man page for a term, type the / character followed by the term and hit the **Enter** key. The program will search from the current location down towards the bottom of the page to try to locate and highlight the term.

If the term is not found, or you have reached the end of the matches, then the program will report Pattern not found (press Return). If a match is found and you want to move to the next match of the term, press **N**. To return to a previous match of the term, press **Shift+N**.

## **Man Page Sections**

Until now, we have been displaying man pages for commands. However, sometimes configuration files also have man pages. Configuration files (sometimes called system files) contain information that is used to store information about the operating system or services.

Additionally, there are several different types of commands (user commands, system commands, and administration commands) as well as other features that require documentation, such as libraries and kernel components.

As a result, there are thousands of man pages on a typical Linux distribution. To organize all of these man pages, the pages are categorized by sections, much like each individual man page is broken into sections.

By default, there are nine sections of man pages:

1. Executable programs or shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in /dev)
5. File formats and conventions, e.g. /etc/passwd
6. Games
7. Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
8. System administration commands (usually only for root)
9. Kernel routines [non-standard]

When you use the man command, it searches each of these sections in order until it finds the first match. For example, if you execute the command man cal, the first section (Executable programs or shell commands) is searched for a man page called cal. If not found, then the second section is searched. If no man page is found after searching all sections, you will receive an error message:

Immagine che contiene testo, Carattere, software, Pagina Web

Descrizione generata automaticamente

## **Determining Which Section**

To determine which section a specific man page belongs to, look at the numeric value on the first line of the output of the man page. For example, if you execute the command man cal, you will see that the cal command belongs to the first section of man pages:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

## **Specifying a Section**

In some cases, you will need to specify the section in order to display the correct man page. This is necessary because sometimes there will be man pages with the same name in different sections.

For example, there is a command called passwd that allows you to change your password. There is also a file called passwd that stores account information. Both the command and the file have a man page.

The passwd command is a user command, so the following command will display the man page for the passwd command, located in the first section, by default:

Immagine che contiene testo, software, Software multimediale, schermata

Descrizione generata automaticamente

To specify a different section, provide the number of the section as the first argument of the man command. For example, the command man 5 passwd will look for the passwd man page in section 5 only:

Immagine che contiene testo, Carattere, software, Software multimediale

Descrizione generata automaticamente

## **Searching by Name**

Sometimes it isn't clear which section a man page is stored in. In cases like this, you can search for a man page by name.

The -f option to the man command will display man pages that match, or partially match, a specific name and provide a brief description of each man page:

Immagine che contiene testo, software, Carattere, Software multimediale

Descrizione generata automaticamente

Note that on most Linux distributions, the whatis command does the same thing as man -f. On those distributions, both will produce the same output.

Immagine che contiene testo, software, Software multimediale, schermata

Descrizione generata automaticamente

## **Searching by Keyword**

Unfortunately, you won't always remember the exact name of the man page that you want to view. In these cases, you can search for man pages that match a keyword by using the -k option to the man command.

For example, what if you knew you wanted a man page that displays how to change your password, but you didn't remember the exact name? You could run the command man -k password:

Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente

Recall that there are thousands of man pages, so when you search for a keyword, be as specific as possible. Using a generic word, such as "the" could result in hundreds or even thousands of results.

Note that on most Linux distributions, the apropos command does the same thing as man -k. On those distributions, both will produce the same output.

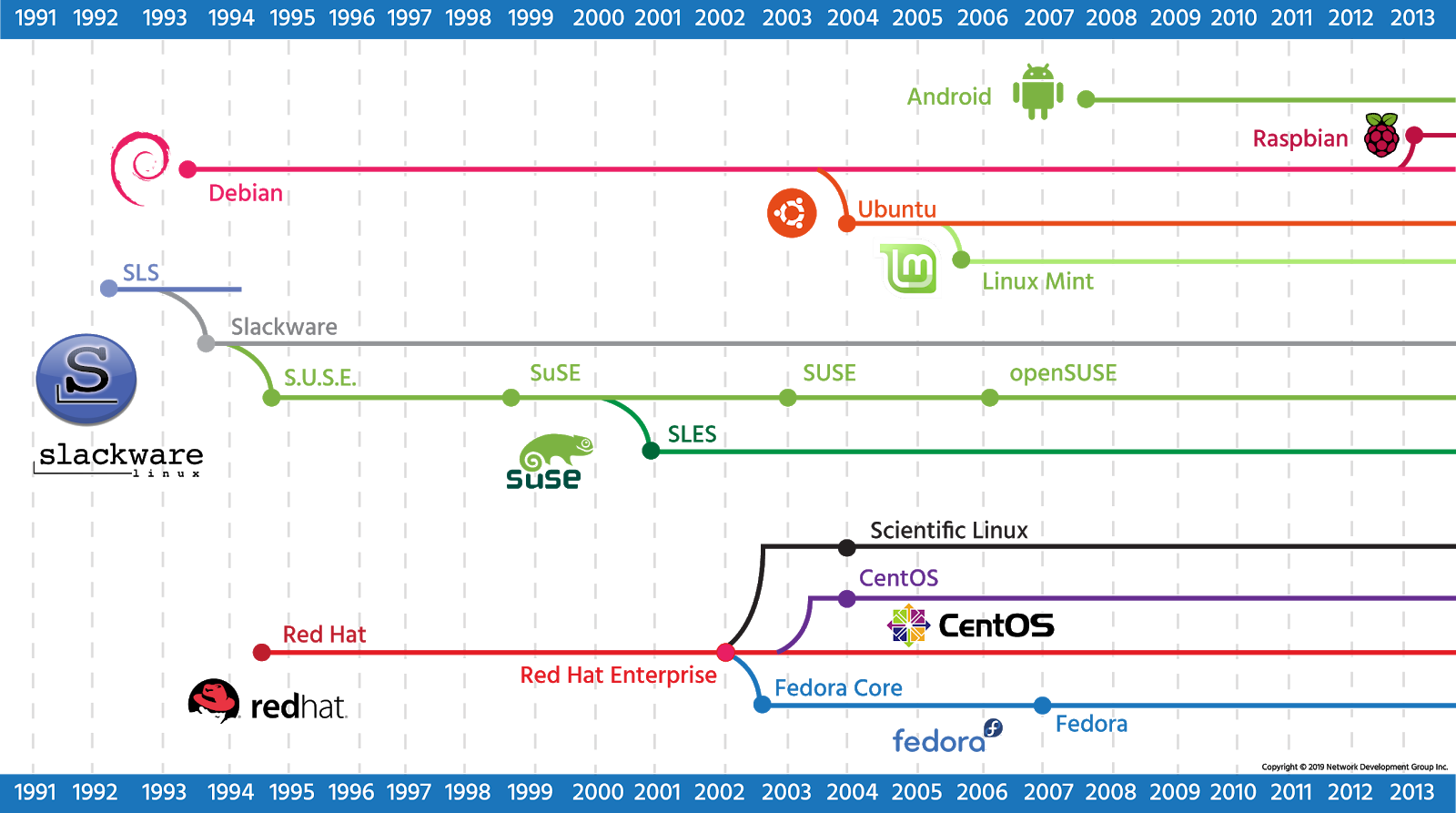
# **Chapter 2 - LAB 2**

# **Chapter 6 - Filesystem Hierarchy Standard (FHS)**

In Linux, everything is considered a file. Files are used to store data such as text, graphics, and programs. Directories are a type of file used to store other files. This chapter will cover how to find files, which requires knowledge about the Linux directory structure, typically called a filesystem and the filesystem standards supported by the Linux Foundation; the **Filesystem Hierarchy Standard (FHS)**. You will also learn about how to find files and commands within the Linux filesystem from the command line.

## **Filesystem Hierarchy Standard**

The open source licensing of many Linux components makes it possible for anyone to create their own distribution. Most people start with a well-known distribution, make modifications, and release a fork of the original.



Since there are so many Linux distributions, it would be expected that numerous people would change the names of the files and folders, eventually making the distributions incompatible. This makes a basic agreement necessary concerning the naming and location of important system files and directories.

The **Filesystem Hierarchy Standard (FHS)** is an agreement to standardize the names and locations of directories and their content for use within most Linux filesystems. It helps to know what directories to expect to find, and what files one should expect to find in them. More importantly, it allows programmers to write programs that will be able to work across a wide variety of systems that conform to this standard.

During the development of the first series of this standard from 1994 to 1995, it was known as the Filesystem Standard (FSSTND). When the second series was started in 1997, it was renamed to the Filesystem Hierarchy Standard (FHS). The final 2.3 version of this second series of this FHS standard was published in 2004 at <http://refspecs.linuxfoundation.org/fhs.shtml>. In 2015, a version of the third series of this standard was published at <http://www.linuxbase.org/betaspecs/fhs/fhs.html>.

The Linux file structure is best visualized as an upside-down tree, with directories and files branching out from the top-level root / directory. While the actual standard details many more directories than listed below, the image and table highlight some of the most important ones to know.

Immagine che contiene schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteImmagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

Vendors of Linux distributions have continued to make some changes, even though a new version of the standard has not been published in over ten years. Two notable new additions include the /run directory and the /sys directory. The /run directory is being considered for use in the forthcoming FHS versions to contain volatile data that changes at runtime. Previously, this data was supposed to be stored under the /var/run directory, but due to the unavailability of this directory at boot time, this data can become scattered in other places, such as hidden files in the /dev directory.

The /sys directory in some traditional UNIX systems was used to hold files related to the kernel. In modern Linux systems, the /sys directory is used to mount the sysfs pseudo-filesystem. This filesystem is used to export information about kernel objects and their relationships to each other. The kernel objects are represented by directories, and the files that they contain are named for the attributes of those objects. The contents of the files represent the value for that attribute. Symbolic links are used to represent relationships between objects.

Another notable change that some Linux distributions are making is the conversion of the /bin, /sbin and /lib directories into symbolic links which point to /usr/bin, /usr/sbin and /usr/lib, respectively. All user executables are now in the /usr/bin directory, administrator executables are now in the /usr/sbin directory, and the libraries to support all these executables are now in the /usr/lib directory.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

The merger of the /bin, /sbin and /lib directories into the /usr/bin, /usr/sbin and /usr/lib directories has been somewhat controversial. Many administrators are comfortable with the long-standing subdivisions of these files into different directories.

Because of the way that UNIX booted, the /bin, /sbin and /lib directories had to be part of the root filesystem as they contain critical boot executables. Some developers now argue that the reason for having them split is no longer valid. In early versions of UNIX, the developers had two filesystems of about 1.5 MiB each on two separate disks for the root filesystem and the /usr filesystem. As the root filesystem started to become full, the developers decided to move some of the executable files that were in the /bin and /sbin directories that were not essential to booting the system into the corresponding directories /usr/bin and /usr/sbin (in the separate /usr filesystem).

The FHS standard categorizes each system directory in a couple of ways for security purposes:

Shareable / Unshareable

* Shareable files can be stored on one host and used on others. For instance, /var/www is often used as the root directory of a web server, which shares files with other hosts. Another example is the user home directories.
* Unshareable files should not be shared between hosts. These include process states in the /var/run directory and the /boot directory.

Variable / Static

* Static files generally do not change, including library files and documentation pages. An example is the info pages located at /usr/share/info.
* Variable files normally change during the execution of programs. The /var/run directory contains files that are both variable and unshareable.

The table below summarizes the main distinctions between file types:

Immagine che contiene testo, linea, Carattere, schermata

Descrizione generata automaticamente

## **Finding Files and Commands**

Most operating systems allow users to search for specific files in the filesystem. A GUI typically provides a search tool that makes it possible to find files and applications. However, there are a few commands available for searching files and commands from the CLI. Both the locate and find commands are useful for searching for a file within the filesystem. While both commands perform similar tasks, each does so by using a different technique, with its own distinct advantages and disadvantages.

### locate Command

Of the two main search commands, the locate command can be described as fast, but always potentially out-of-date. Its speed comes from the fact that the locate command searches a database that contains the location of the files on the filesystem, but that database needs to be updated to be accurate.



In its simplest form, the locate command accepts a search string as an argument. For example, to find a file named passwd, use the following command:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

A few things to consider when using the locate command:

* The locate command will only return results of files that the current user would normally have access to.
* The locate command will display all files that have the search term anywhere in the file name. For example, the search term passwd would match both /etc/passwd and /etc/thishaspasswdinit.
* Like most things in Linux, the locate command is case sensitive. For example, the search term passwd would not match a file named /etc/PASSWD. To have the locate command not be case sensitive, use the -i option.

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

The locate command is dependent on a database. This database can be updated manually by an administrator using the updatedb command, though typically this command is run automatically every day through *cron*, a system scheduling service that runs commands on a particular recurring schedule.

When executed, the updatedb command creates a database of all files that it finds on the computer for quick searching. This command can only be executed by a user with administrative access, which can be achieved using the sudo command:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

The updatedb command can be told not to search a particular name, path, or filesystem by changing the corresponding line in its configuration file, /etc/updatedb.conf. Below is an example of the default file in its entirety:

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

The /etc/updatedb.conf file can be edited as the root user. Any name, path, or filesystem that is listed in the file on the appropriate line will not be added to the database. Any line starting with the # symbol will be ignored since it is commented out.

Since the locate command works with a database, it is able to work very quickly even on a system with hundreds of thousands of files. However, if you want to use the locate command to search for a file that was created very recently, it will fail to find the file if the database hasn't been updated since the file creation.

Likewise, the database may contain outdated information about files that might have existed in the very recent past, so the command will report them incorrectly as still existing.

The following example demonstrates the consequences that arise when the database is not updated in real time:

1. A new file called lostfile isn't initially found by the locate command.



1. After the database is updated by the updatedb command, the locate command can find the lostfile file.

Immagine che contiene testo, Carattere, software, Software multimediale

Descrizione generata automaticamente

1. After the lostfile file has been deleted, the locate command will still report that the file exists.



### find Command

If you want to search for files that are currently in the filesystem, then you should use the find command. The find command is slower than the locate command because it searches directories in real time; however, it doesn't suffer from problems associated with an outdated database.

The find command expects a directory as the first argument. This will be the starting point of the search. The find command will search this directory and all of its subdirectories. If no directory is specified, then the find command will start the search at the current directory.



Note that the period . character refers to the current directory in the following example, but the current directory could also be referred to using the ./ notation. The find command uses the -name option to search for files by name, in this case, the Downloads directory.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

The first search yielded no results because the string must match the exact name of the file, not just part of the name. The third command demonstrates that globbing can be used, and the fourth command demonstrates multiple matches (notice that single quotes were added so that the find command will interpret the glob, rather than the shell).

If the search for the Downloads directory was conducted from the root / directory, many errors would arise:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

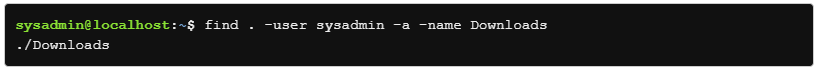
These errors can be ignored for now, but just be aware that errors like these are typical when either a regular user is attempting to search root-only areas, or when the root user is attempting to search areas that are dedicated to the system’s processes.

The find command also offers many options for searching files, unlike the locate command which searches only for files based on file name. The following table illustrates some of the more commonly used criteria:

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

If multiple criteria are specified, then all criteria must match as the find command automatically assumes a logical *AND* condition between criteria, meaning all of the conditions must be met. This could be explicitly stated by using the -a option between criteria. For example, the Downloads directory must also be owned by the sysadmin user in order for the find command to produce a result in the following example:



Logical *OR* conditions can be specified between criteria with the -o option, meaning at least one of the conditions must be true. The following output lists files that are either named Downloads or owned by the sysadmin user.

Immagine che contiene testo, software, schermata

Descrizione generata automaticamente

Logical groupings of criteria can also be specified using parentheses. Be careful to precede the parentheses with a backslash or to use single quotes around them so that the shell doesn't attempt to interpret them as special characters. Also, as mentioned previously, use quotes around any globs that the find command should interpret instead of the shell, as shown in the following example:



In plain text, the find command in the preceding example will return files in the current directory, with the non-case sensitive name starting with desk *OR* files with the exact name Downloads owned by the sysadmin user.

By default, the find command simply prints the names of the files that it finds. To generate output showing additional file details, similar to the ls -l command, you can specify the -ls option:

Immagine che contiene testo, Carattere, software, schermata

Descrizione generata automaticamente

To make the output exactly like the output of the ls -l command, use the -exec option to execute ls -l on each file found. For example:

Immagine che contiene testo, software, Carattere, Software multimediale

Descrizione generata automaticamente

The previous example tells the find command to execute the ls -l command for each file found. The pair of curly braces {} is a placeholder for the name of each file found; note that there is no space between them. The \; is an escaped semicolon that is added between each command so that multiple commands may be executed in series.

In order to make the find command confirm the execution of the command for each file found, use the action option -ok instead of the -exec option:

Immagine che contiene testo, Carattere, software, schermata

Descrizione generata automaticamente

The command above tells the find command to execute the rm command for files that were modified less than two minutes ago. Did you notice how the first file that the rm command tried to remove was the current directory (represented by the period. character)? That is enough reason not to use the -exec option because the rm command would have tried to remove the current directory.

There will be situations where it’s useful to find a file or set of files depending on their age, or when they were last modified, such as all files that have been changed since a given time (such as the last backup). The *modification time* -mtime option to the find command provides the ability to give time as criteria for a search.

First, a word about how the find command uses times is relevant here. When looking for files modified within the period of a day, such as 3 days ago, you would use -mtime 3, but if you are looking for files that have changed anytime between 3 days ago and now, you would use -mtime -3. If you are looking to find files older than 3 days, you would use -mtime +3.

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

To summarize, a number without a + or - means exactly *N* days ago, a number with a - means anytime between *N* days ago and NOW, and a number with a + means *N* days ago or more.

For example, if you backup every seven days, use the following command to backup all files that are 3 days or less old:



The above command starts in the /tmp directory and finds all files that have been modified in the last three 24-hour periods, or 72 hours.

The find command is less exact by default than you might want it to be. When using numbers of days with the -atime, -ctime or -mtime operators, the days refer to 24-hour blocks. Therefore, you may think that you are searching to find all files that occurred between now and your last backup by referring to that day, but depending on the time of day, you may not have found all of the files.

Since the find command’s time operators are effectively tied to 24-hour blocks or days, you can use a reference file to make certain you have found all files that occurred precisely since your last backup.

To find all files that are newer than your last backup, find out the date and time of that backup, and create a reference file with that date/time so you can use it with the find -newer command. For example, if the backup was done at precisely 2300 (11:00 PM) on the 1st of March, 2020, you can create a reference file with that exact date and time with the following command:



Now that we have a file that mirrors precisely the date and time of our last backup, we can search for all files in the /home directory that might have been created or modified since the last backup date with the following command:

Immagine che contiene testo, elettronica, schermata, software

Descrizione generata automaticamente

The find command also allows the use of friendly notation to find files that are larger or smaller than a particular size. Trying to find files that are above 1 Kilobyte on a busy system is like searching for the word “the” on Google; so much output is returned it’s essentially useless. When searching for files of a given size, or those that are smaller or larger than a given size, try to restrict the search to the smallest valid set of directories and files possible.

For example, to find all files that are smaller than 1 Kilobyte in the /etc directory, you would use:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

If the files are smaller than 1 Kilobyte, they will be listed as output. You can also use the -size option to find all files on the system that are 1 Megabyte or more with the command:

Immagine che contiene testo, elettronica, schermata, Sito Web

Descrizione generata automaticamente

The find command can also help you find files by specifying a file type, such as regular file, directory, and more, using the -type option. The table below shows all the different types of files you can find using the -type option:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

To demonstrate, in order to search only for directories under the /home directory, you could use the following command:

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

The command above will search the /home/sysadmin directory tree structure and filter out all results except for files that are the specified directory type.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

### whereis Command

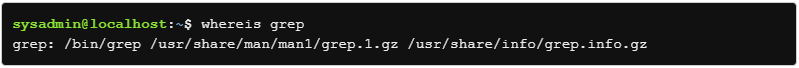
There are many commands available to the operating system, and it is sometimes useful to know where they are. The whereis command can be used to search your PATH for the binary. It is also capable of searching for the man page, and source files for any given command.



The echo command can be used to determine which directories are in the PATH:



To find out where the grep command is located, use whereis without any options:



The output of the whereis command returns three directories. The first is where the grep command is located, in the /bin/grep directory. There is also a path given for the man page of the grep command at /usr/share/man/man1/grep.1.gz and another path for the info page at /usr/share/info/grep.info.gz. Without an option, the whereis command provides all three pieces of information.

To view the location of the binary separate from the man page and info page, use the -b and -m options respectively:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

The -s option can be used to find source code that has been installed for a given command. In the case of grep, only the binary is currently installed:



Notice that no results are returned because the source code for the grep command is not installed.

The -u option can be used in one of two ways. It can identify commands that do not have documentation (i.e. a man page) for a requested attribute. It can also be used to identify commands that have more than one documentation file. For example, to find out which commands in the /bin directory either do not have a man page, or have more than one documentation file, use the following command:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

In the previous example, the asterisk \* character is used to consider every file in the current directory.

The example below identifies a specific command that does not have a man page:



This means that there is no man page for the type command:



If the same example is repeated for the ls command, there will be no output because ls has a man page.



To limit the search to a specific path or paths, capitalized options can be used. The -B option searches for binaries, the -M option for manuals and documentation, and the -S option for sources. The -f option must be used to indicate the end of the path list and the beginning of the search term.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

This type of searching, however, may be best left to the find command, as it allows for more specific criteria to be used in a search.

### which Command

Sometimes the whereis command returns more than one result on a command. In this case, an administrator would want to know which command is being used.



The bash command is an example that normally yields two results:



To find out which bash result is the real command, in other words, the result that is used when executing the command, use the which command:



The -a option can be used with the which command to locate multiple executable files. This would be useful to know if an executable script was inserted maliciously to override an existing command.



By using the which command, an administrator can be fairly certain that the only executable running by the name of the ping command is located in the /bin directory.

### type Command

The type command can be used to determine information about various commands.



Some commands originate from a specific file:



This output would be similar to the output of the which command:



The type command can also identify commands that are built into the Bash (or other) shell:



This output is significantly different than the output of the which command:



Using the -a option, the type command can also reveal the path of another command:



The type command can also identify aliases to other commands:

Immagine che contiene testo, Carattere, software, Software multimediale

Descrizione generata automaticamente

The output of these commands indicates that ll is an alias for ls -alF, and even ls is an alias for ls --color=auto. Again, the output is significantly different than the which command:



The type command supports other options, and can lookup multiple commands simultaneously. To display only a single word describing the echo, ll, and which commands, use the -t option:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

# **Chapter 6 - LAB 6**

# **Chapter 3**