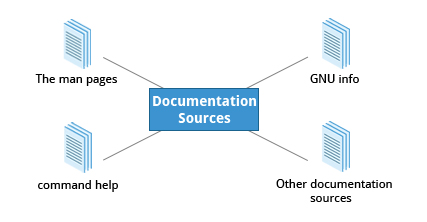
**Chapter 7: Finding Linux Documentation**

**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.



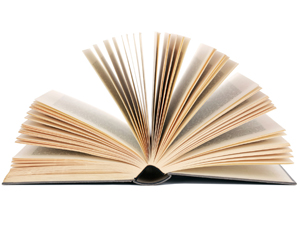
Whether you are an inexperienced user or a veteran, you won’t always know how to use various Linux programs and utilities, or what to type at the command line. You will need to consult the help documentation regularly. Because Linux-based systems draw from a large variety of sources, there are numerous 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/>

**The man pages**



The **man pages** are the most often-used source of Linux documentation. They provide in-depth documentation about many programs and utilties 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**

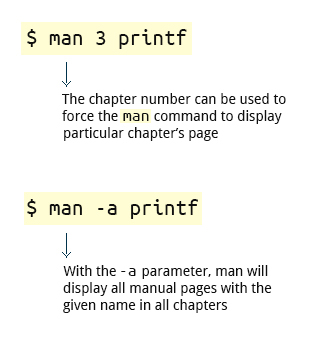
The **man** program searches, formats, and displays the information contained in the **man pages**. Because many topics have a lot of information, output is piped through a **terminal pager** program such as **less** to be viewed one page at a time; at the same time the information is formatted for a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

**GNU Info System**

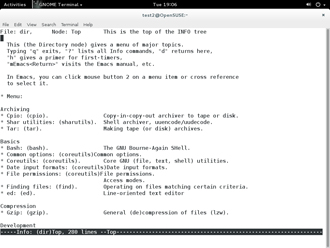


The next source of Linux documentation is the **GNU Info System**.

This is the **GNU** project's standard documentation format (which it prefers it 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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch07_screen13.jpg)

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.

Click the image to view an enlarged version.

**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 |

**GNU Info System**

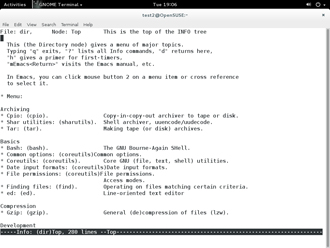


The next source of Linux documentation is the **GNU Info System**.

This is the **GNU** project's standard documentation format (which it prefers it 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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch07_screen13.jpg)

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**

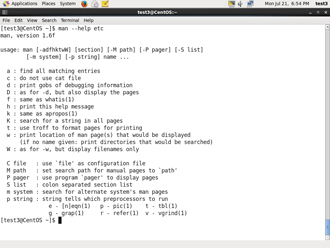
The topic which you view in the **info** page is called a **node**.

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 |

**Introduction to the help Option**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch07_screen18.jpg)

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

Every command has an available short description which can be viewed using the --help or -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

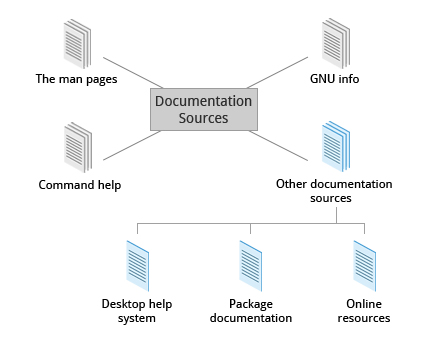
The --help option is useful as a quick reference and it displays information faster than the **man** or **info**pages.

**About the help Command**

Some popular commands (such as **echo**) when run in a **bash** command shellsilently run their own **built-in** versions of system programs or utilties, 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**.

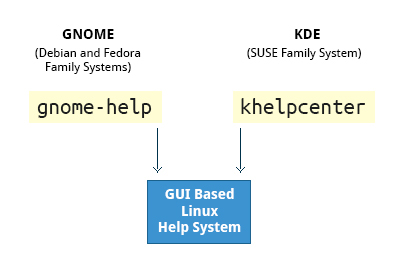
For these built-in commands, **help**performs the same **basic** function as the -h and --help arguments (which we will discuss shortly) perform for stand-alone programs.

**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**

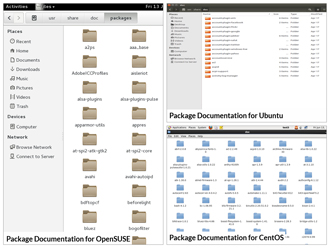
**Package Documentation**

**Summary**

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

* + 1. 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.
    2. 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.

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch07_screen25.jpg)

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>

Moreover you can use online search sites to locate helpful resources from all over the Internet, including blog posts, forum and mailing list posts, news articles, and so on.

**Chapter 8: File Operations**

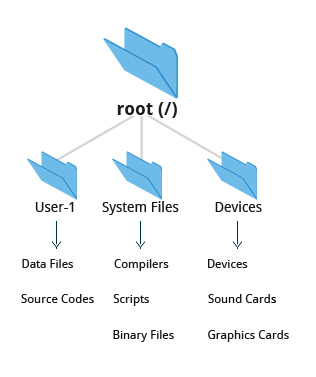
**Learning Objectives**

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.

**Introduction to Filesystems**

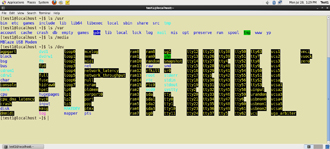


In Linux (and all UNIX-like operating systems) it is often said “Everything is a file”, or at least it is treated as such. This means whether you are dealing with normal data files and documents, or with devices such as sound cards and printers, you interact with them through the same kind of Input/Output (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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_Ch08_screen04.jpg)

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.

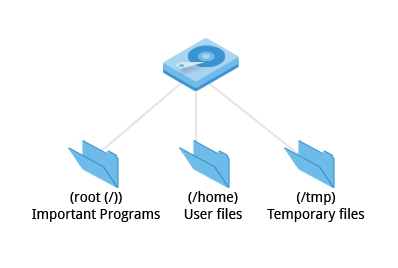
Linux supports various filesystem types created for Linux, along with compatible filesystems from other operating systems such as **Windows** and **MacOS**. Many older, legacy filesystems, such as **FAT**, are supported.

Some examples of filesystem types that Linux supports are:

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

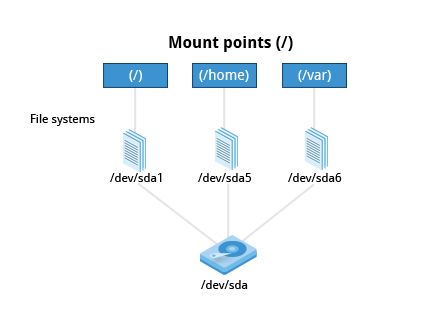
**vfat**, **ntfs, hfs** (filesystems from other operating systems)

**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**

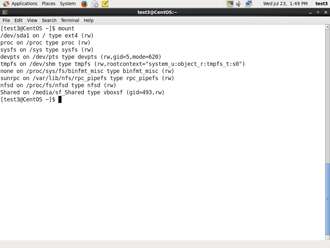
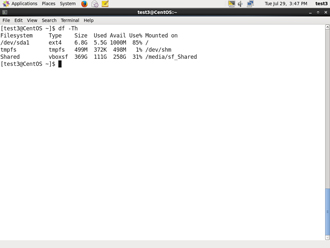
The **mount** command is used to attach a filesystem (which can be local to the computer or, as we shall discuss, on a network) somewhere within the filesystem tree. Arguments include the **device node** and **mount point**. For example,

$ mount /dev/sda5 /home

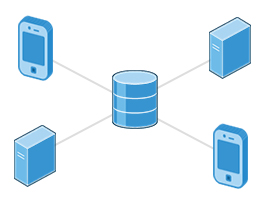
will attach the filesystem contained in the disk partition associated with the /dev/sda5 device node, into the filesystem tree at the /home mount point. (Note that unless the system is otherwise configured only the root user has permission to run **mount**.) If you want it to be automatically available 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.

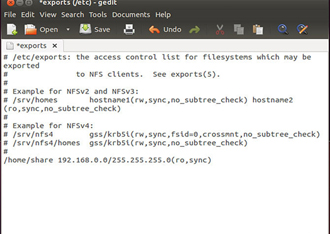
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch08_screen07A.jpg) [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch08_screen07B.jpg)

**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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_Ch08_screen10.jpg)

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

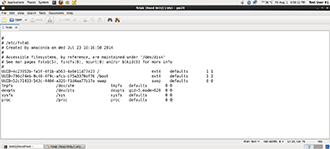
The text file /etc/exports contains the directories and permissions that a host is willing to share with other systems over NFS. An entry in this file may look like the following:

/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).

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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_Ch08_screen11.jpg)

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:

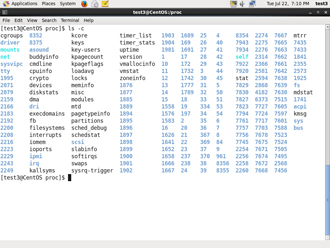
servername:/projects /mnt/nfs/projects nfs default 0 0

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.

**proc Filesystem**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch08_screen12.jpg)

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:

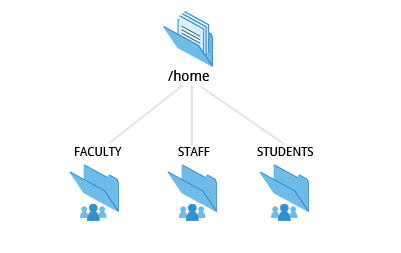
/proc/cpuinfo /proc/interrupts /proc/meminfo /proc/mounts /proc/partitions /proc/version

/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.

**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/

**The /bin and /sbin Directories**

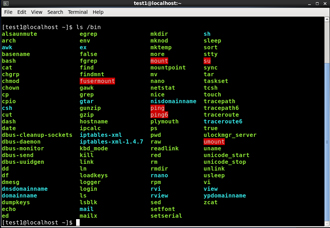
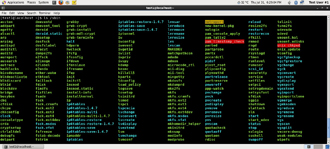
The /bin directory contains executable binaries, essential commands used in single-user mode, and essential commands required by all system users, such as:

|  |  |
| --- | --- |
| **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

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_Ch08_screen16a.jpg) [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_Ch08_screen16b.jpg)

**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:

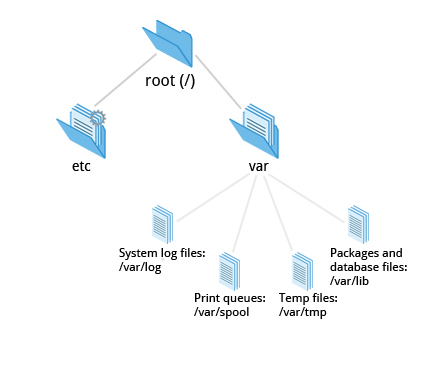
* Empty on the disk partition when it is not mounted
* Contains entries which are created by the **udev** system, which creates and manages device nodes on Linux, creating them dynamically when devices are found. The /dev directory contains items such as:

/dev/sda1 (first partition on the first hard disk)

/dev/lp1 (second printer)

/dev/dvd1 (first DVD drive)

**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**

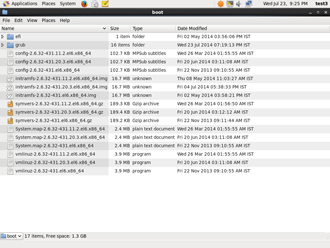
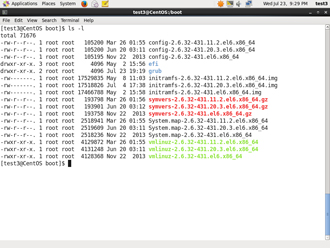
The /boot directory contains the few essential files needed to boot the system. For every alternative kernel installed on the system there are four files:

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

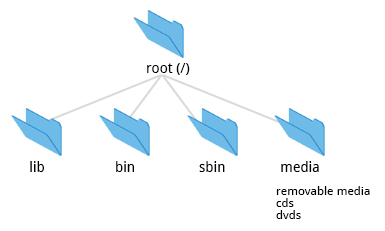
Each of these files has a kernel version appended to its name.

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.

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_Ch08_screen18a.jpg) [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_Ch08_screen18b.jpg)

**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. |

**Comparing Files**

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

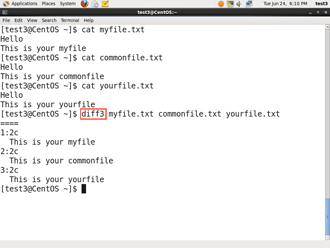
**diff** is used to compare files and directories. This often-used utiility program has many useful options (see man diff) including:

|  |  |
| --- | --- |
| **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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch08_screen28.png)

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**.

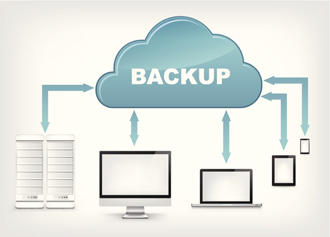
**Using the 'file' utility**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch08_screen29.jpg)

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.

**Backing Up Data**



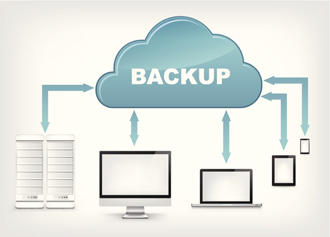
There are many ways you can back up data or even your entire system. Basic ways to do so include use of simple copying with **cp** and use of the more robust **rsync**.

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.

**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.

**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](https://www.kernel.org/) to store archives of the Linux kernel. Once again it trades a slower compression speed for an even higher compression ratio.

Some usage examples:

|  |  |
| --- | --- |
| **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.xz 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**!

**Summary (1 of 2)**



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.

**Summary (2 of 2)**



* /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.

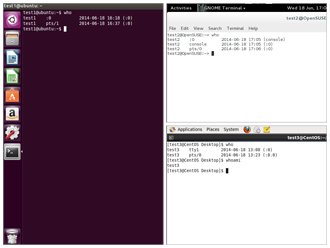
**Chapter 9: User Environment**

**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.

**Identifying the Current User**

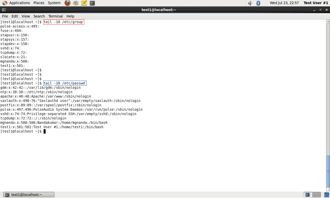
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen03.jpg)

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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen09.jpg)

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.

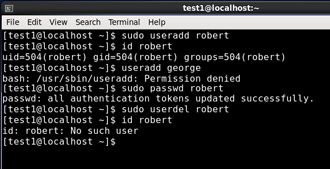
Historically **Fedora**-family systems start **uid**'s at 500; other distributions begin at 1000.

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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen15.jpg)

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

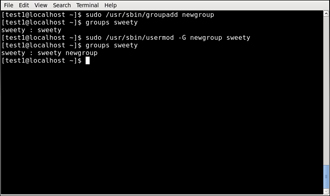
and sets the default shell to /bin/bash. Removing a user account is as easy as typing userdel turkey However, this will leave the /home/turkey directory intact. This might be useful if it is a temporary inactivation. To remove the home directory while removing the account one needs to use the **-r**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

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen12.jpg)

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

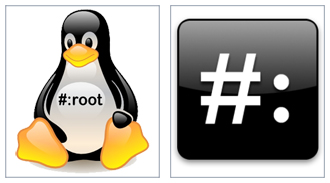
These utilities update /etc/group as necessary. **groupmod** can be used to change group properties such as the Group ID (gid) with the -g option or its name with the -m option.

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**

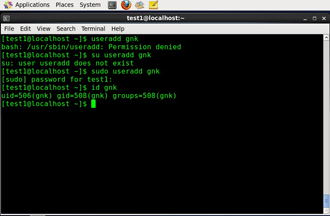
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen05.jpg)

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**

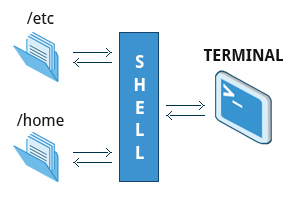
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen11.jpg)

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**

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:

* ~/.bash\_profile
* ~/.bash\_login
* ~/.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.

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen08a.jpg) [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen08b.jpg)

**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 ...

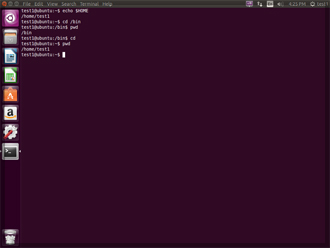
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen13a.jpg) [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen13b.jpg) [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen13c.jpg)

**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 | * Edit ~/.bashrc and add the line export VARIABLE=value * Type source ~/.bashrc or just . ~/.bashrc (dot ~/.bashrc); or just start a new shell by typing  bash |

**The HOME Variable**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen16.jpg)

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.

|  |  |
| --- | --- |
| **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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen17.jpg)

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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen18.jpg)

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

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

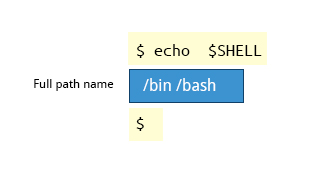
They must be surrounded in single quotes when they are used as in the following example: $ echo $PS1 $ $ export PS1='\ u@\h:\w$ ' me@example.com:~$ # new prompt me@example.com:~$

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 $

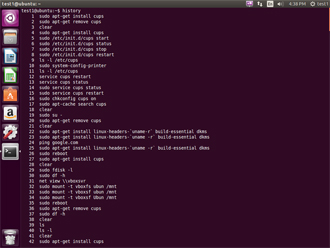
**The SHELL Variable**



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 $

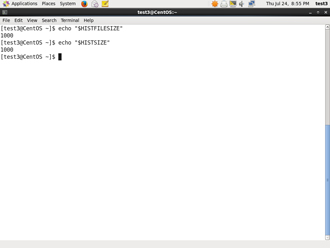
**Recalling Previous Commands**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen23.jpg)

**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 first in the list. This information is stored in ~/.bash\_history.

**Using History Environment Variables**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen24.jpg)

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

* echo $SHELL
* echo $HOME
* echo $PS1
* ls -a
* ls -l /etc/ passwd
* sleep 1000
* history

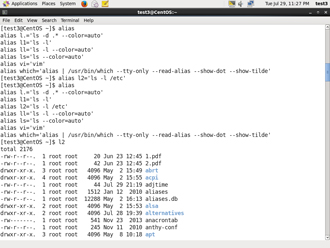
$ !1                             # Execute command #1 above echo $SHELL /bin/bash $ !sl                           # Execute the command beginning with "sl" sleep 1000 $

**Keyboard Shortcuts**

You can use keyboard shortcuts to perform different tasks quickly. The table lists some of these keyboard shortcuts and their uses.

|  |  |
| --- | --- |
| **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 the entire line |
| **Tab** | Auto-completes files, directories, and binaries |

**Creating Aliases**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen32.jpg)

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.

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.

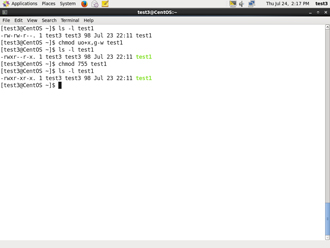
**File Ownership**

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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen37.jpg)

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

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:

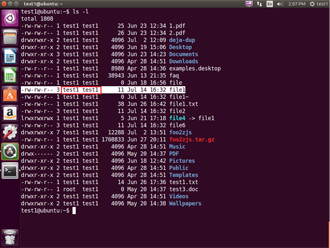
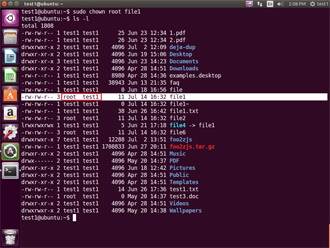
* 4 if read permission is desired.
* 2 if write permission is desired.
* 1 if execute permission is desired.

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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen35a.jpg) [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen35b.jpg)

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

The image on LHS shows the permissions for owners/groups/all users on 'file1'. The image on RHS 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

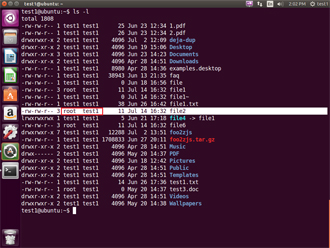
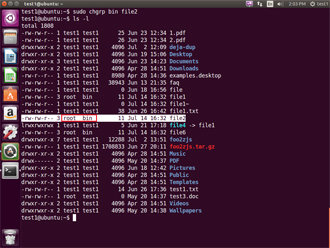
**Example of chgrp**

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

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen36a.jpg) [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch09_screen36b.jpg)

Click the image to view an enlarged version.