What is GNU?

GNU is an operating system that is free software—that is, it respects users' freedom. The GNU operating system consists of GNU packages (programs specifically released by the GNU Project) as well as free software released by third parties. The development of GNU made it possible to use a computer without software that would trample your freedom.

GCC

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain and the standard compiler for most projects related to GNU and Linux, including the Linux kernel. The Free Software Foundation (FSF) distributes GCC under the GNU General Public License (GNU GPL).

Directory structure

In Linux and many other operating systems, directories can be structured in a tree-like hierarchy. The Linux directory structure is well defined and documented in the Linux File system Hierarchy Standard (FHS). Referencing those directories when accessing them is accomplished by using the sequentially deeper directory names connected by forward slashes (/) such as /var/log and /var/spool/mail. These are called paths.

The following table provides a very brief list of the standard, well-known, and defined top-level Linux directories and their purposes.

Directory	Description
/ (root filesystem)	The root filesystem is the top-level directory of the filesystem. It must contain all of the files required to boot the Linux system before other filesystems are mounted. It must include all of the required executables and libraries required to boot the remaining filesystems. After the system is booted, all other filesystems are mounted on standard, well-defined mount points as subdirectories of the root filesystem.
/bin	The /bin directory contains user executable files.

Directory	Description
/boot	Contains the static bootloader and kernel executable and configuration files required to boot a Linux computer.
/dev	This directory contains the device files for every hardware device attached to the system. These are not device drivers, rather they are files that represent each device on the computer and facilitate access to those devices.
/etc	Contains the local system configuration files for the host computer.
/home	Home directory storage for user files. Each user has a subdirectory in /home.
/lib	Contains shared library files that are required to boot the system.
/media	A place to mount external removable media devices such as USB thumb drives that may be connected to the host.
/mnt	A temporary mountpoint for regular filesystems (as in not removable media) that can be used while the administrator is repairing or working on a filesystem.
/opt	Optional files such as vendor supplied application programs should be located here.
/root	This is not the root (/) filesystem. It is the home directory for the root user.
/sbin	System binary files. These are executables used for system administration.

Directory	Description
/tmp	Temporary directory. Used by the operating system and many programs to store temporary files. Users may also store files here temporarily. Note that files stored here may be deleted at any time without prior notice.
/usr	These are shareable, read-only files, including executable binaries and libraries, man files, and other types of documentation.
/var	Variable data files are stored here. This can include things like log files, MySQL, and other database files, web server data files, email inboxes, and much more.

What Do Shells Do?

When you sign in at the command line or launch a terminal window on Linux, the system launches the shell program. Shells offer a standard way of extending the command line environment. You can swap out the default shell for another one, if you like.

Difference between absolute path and relative path in Linux:

When working in a Linux system, navigating through files and directories is a common task. To access a file or directory, you need to specify its path. Linux supports two types of paths: absolute path and relative path.

An absolute path is a complete path to a file or directory from the root directory. The root directory is the top-level directory of the file system and is represented by a forward slash (/). Absolute paths always start with the root directory and provide the full path to the file or directory.

For example, the absolute path to the home directory of a user named "test" would be "/home/test".

On the other hand, a relative path is a path to a file or directory that is relative to the current directory. It specifies the location of the file or directory in relation to the current directory. Relative paths do not start with the root directory and are usually shorter than absolute paths.

For example, if you are currently in the home directory of "test" and want to access a file named "example.txt" in a subdirectory called "documents", the relative path would be "documents/example.txt".

The main difference between absolute and relative paths is how they are interpreted. Absolute paths always point to the same location regardless of the current directory, whereas relative paths may point to different locations depending on the current directory.

Absolute paths are useful when you need to refer to a file or directory from anywhere in the system. For example, if you want to create a symbolic link to a file that is located in a different directory, you would use an absolute path to specify the location of the file.

Relative paths are more convenient when you are working within a specific directory or subdirectory. They allow you to specify the location of a file or directory relative to your current location, which can save you time and typing.

In conclusion, both absolute and relative paths have their advantages and are useful in different situations. Understanding the difference between the two can help you navigate through the file system more efficiently and work more effectively in a Linux environment.

Advantages of absolute path:

1. Unambiguous: Absolute paths are unambiguous and provide the exact location of a file or directory in the file system. This means that regardless of the current directory you are in, an absolute path will always point to the same location. This can be useful when working with scripts, as you can be sure that the correct file or directory is being accessed.

2. Easy to Use: Absolute paths are easy to use once you understand the syntax. They start with the root directory, which is represented by a forward slash (/), and then provide the full path to the file or directory. For example, "/home/user/Documents" is an absolute path to the "Documents" directory in the home directory of the user "user".

3. Useful for Remote Access: Absolute paths are useful when accessing files or directories remotely. For example, if you are accessing a file or directory on a remote server, you can use an absolute path to ensure that you are accessing the correct file or directory.

4. Useful for Symbolic Links: Symbolic links are files that act as shortcuts to other files or directories. Absolute paths are useful when creating symbolic links because they ensure that the link points to the correct location, regardless of the current directory. For example, if you want to create a symbolic link to a file in a different directory, you would use an absolute path to specify the location of the file.

5. Useful for System Administration: System administrators often need to work with files and directories located in different parts of the file system. Absolute paths are useful for system administration tasks because they allow administrators to access files and directories from anywhere in the file system. For example, an administrator might need to access log files located in the /var/log directory. An absolute path would allow them to access these files from any directory in the system.

Advantages of relative path:

1. Shorter and Simpler: Relative paths are shorter and simpler than absolute paths. They are specified relative to the current working directory, which means that they do not need to include the entire path to a file or directory. This can be useful when working with long and complex file paths, as it can save time and reduce typing errors.

2. Easier to Move Around: Relative paths are relative to the current working directory, which means that they can be easily moved around within the file system without breaking the path. For example, if you are working in the directory /home/user/documents and want to access a file in the directory /home/user/pictures, you can use the relative path ../pictures/file.jpg to access the file, even if you move to a different directory.

3. Useful in Scripts: Relative paths are useful in scripts because they can be used to reference files and directories that are relative to the script's location. This can be useful when creating portable scripts that can be run from different locations without needing to modify the file paths.

4. Useful in Web Development: Relative paths are commonly used in web development to reference files such as images, scripts, and stylesheets. Using relative paths makes it easier to move a website from one server to another, as the file paths will remain valid as long as the directory structure is maintained.

5. Useful in Collaborative Work: When working collaboratively on a project, relative paths can be useful as they allow team members to reference files and directories relative to their own working directory. This can help to avoid confusion and ensure that everyone is referencing the same files.

ls

The ls command is used without any arguments to list all the files and folders in the terminal's current directory.

ls path

To list all the files and folders in a particular directory, the path of this directory is input as the first argument to the ls command.

ls /

The / argument is used to list all files and folders in the root directory.

ls . .

The .. argument is used to list down the files and folders in the parent directory of the current directory.

ls ~

The ~ argument is used to list all files and folders in the home directory.

ls -R

The -R argument is used to recursively list down all the folders and files in all the directories and sub-directories of the system.

ls -lS

The -lS flag lists down folders and files in sorted order (according to size).

ls -a

The -a argument is used to list down any hidden folders or files.

ls -F

Symbol meaning

- / - directory
- @ - symbolic links
- | - fifos
- = - sockets
- > - doors
- * Executable

Linux File Permissions

At times, even will get confused about the numbers and notations used for setting up file permissions. In this article, we will learn the concepts and commands involved in Linux file permissions from a beginner perspective.

Before diving into commands, you should understand the basic notations used for representing permissions.

Read, Write and Execute

The read, write and execute Permissions are denoted by letters r, w and x

Octal Notation

Read, Write and Execute can also be denoted using Octal.

Read (r) – 4
Write (w) – 2
Execute (1) – 1

Let's say a file has read, write and execute permissions, then you can denote that in a number as 7 (ie, 4+2+1=7). You will understand about this more in the following sections.

User, Group and Others (UGO)

User – The owner of the file. Mostly, one who created the file.
Group – The group which the file belongs to.
Others – Everyone other than the user and the group.

Listing Permissions

Every file has a permission associated with it. To list the assigned permissions for files (also hidden files) in your current directory, use the following command.

```
ls -la
```

Output:

```
[vagrant@centos7 ~]$ ls -l

-rw-rw-r--. 1 vagrant vagrant 0 May  2 02:51 demo.sh

-rw-r--r--. 1 root    root    0 May  2 03:14 rootsfile.txt

drwxrwxr-x. 2 vagrant vagrant 4096 May  2 03:17 demodir
```

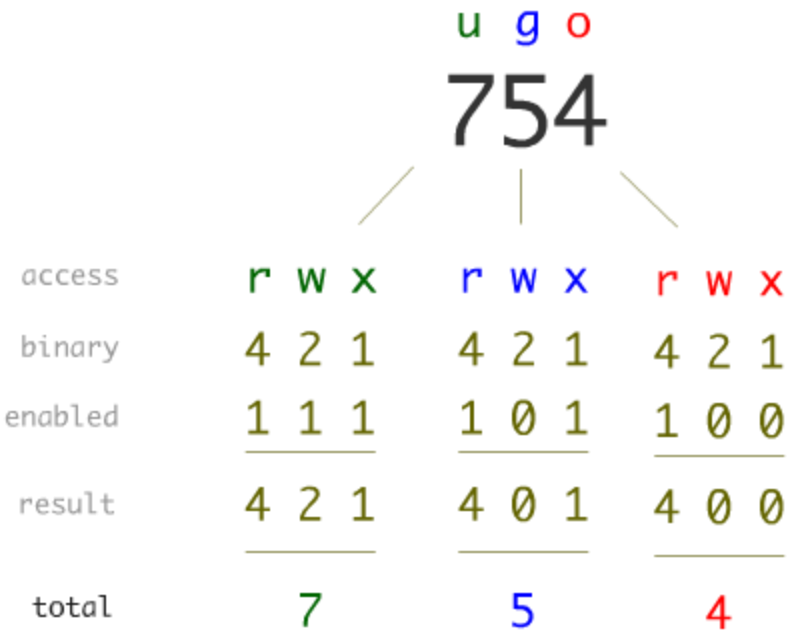
Now, lets dissect the output and see how to understand the permissions of files.

-rw-rw-r–. represents the file permissions. if the line starts with "-" it means it is a regular file. If it starts with "d" then it is a directory. Followed by that, you have three sets of "rwx".

1. The first set represents the permission for the user (who created or owns the file).
2. The second set represents the permissions for the group the file is associated with.
3. The third set represents anyone other than the user and group.

By default, when a new file is created, it is assigned certain permissions. In Linux, this is usually **666**, which translates to read and write permissions for the owner, group, and others. For directories, the default is typically **777**, which means read, write, and execute permissions for the owner, group, and others.

Changing permissions of a file
“chmod” command is used for changing the permission of a file/directory. You need two parameters for chmod command as shown below.
chmod (permission-to-be-assigned) (path-to-file)
Permissions can be assigned using “+” and “-” symbols. Lets look at some examples.
1. To assign user permissions use “u+”” (eg: u+x, u+xw, u+rwx) with the chmod command as shown below.
chmod u+x demo.sh
chmod u+rw demo.sh
chmod u+rwx demo.sh
2. To revoke the access given to the user, you can use “u-” command as shown below. This will unset all the given permissions.
chmod u-x demo.sh
chmod u-rw demo.sh
chmod u-rwx demo.sh
In the same way, you can replace “u” with “g” and “o” for assigning permissions to groups and others.
3. To assign permissions for ugo at the same time, you can use the following syntax.
chmod ugo+x demo.sh
Changing permissions Using Octals
We have seen how octal can be user to represent permissions. Have a look at the image below to get more ideas about octal representation.



While using octal, we represent the permissions using three numbers. First for the user, second for the group and the third one for others.
1. To give the user all permissions use the following form.
chmod 700 demo.sh

2. To give the user all permissions, the group just read/write and others only read, use the following command.
chmod 764 demo.sh
In this manner, you can assign different permissions to users, groups and others.

List of Linux system Errors

Following is a partial list of possible errors of Linux system:

Error code	Error no	Description
EPERM	1	It is displayed if the operation is not permitted.
ENOENT	2	It is displayed if there is no such file or directory exists.
ESRCH	3	It is displayed if there is no such process exists.
EINTR	4	It is displayed for Interrupted system call
EIO	5	It is displayed for input/output error.
ENXIO	6	It is displayed if there is no such device or address exists.
E2BIG	7	It is displayed if argument list is too long.
ENOEXEC	8	It is displayed if there is an exec format error
EBADF	9	It is displayed in case of bad file descriptor.
ECHILD	10	It is displayed if there is no child process exists.

What is the number between file permission and owner in ls -l command output?

The first number of the `ls -l` output after the permission block is the number of hard links.

It is the same value as the one returned by the `stat` command in "Links".

This number is the hardlink count of the file, when referring to a file, or the number of contained directory entries, when referring to a directory.

A **file** typically has a hard link count of 1 but this changes if hard links are made with the `ln` command.

In your example, adding a hard link for tempFile2 will increase its link count:

```
ln -l
ln tempFile2 tempHardLink
ln -l
```

Both *tempFile2* and *tempHardLink* will have a link count of 2.

If you do the same exercise with a symbolic link (`ln -s tempFile2 tempSymLink`), the count value will not increase.

A **directory** will have a minimum count of 2 for `'.'` (link to itself) and for the entry in its parent's directory .

In your example, if you want to increase the link count of *tempFolder*, create a new directory and the number will go up.

```
ls -l tempFolder
mkdir tempFolder/anotherFolder
ls -l tempFolder
```

The link from *anotherFolder/* to *tempFolder/* (which is `..`) will be added to the count.

25 Basic Linux Commands

1. **Is –** Displays information about files in the current directory.
2. **pwd –** Displays the current working directory.
3. **mkdir –** Creates a directory.
4. **cd –** To navigate between different folders.
5. **rmdir –** Removes empty directories from the directory lists.
6. **cp –** Moves files from one directory to another.
7. **mv –** Rename and Replace the files
8. **rm –** Delete files
9. **uname –** Command to get basic information about the OS
10. **locate–** Find a file in the database.
11. **touch –** Create empty files
12. **ln –** Create shortcuts to other files
13. **cat –** Display file contents on terminal
14. **clear –** Clear terminal
15. **ps-** Display the processes in terminal
16. **man –** Access manual for all Linux commands
17. **grep-** Search for a specific string in an output
18. **echo-** Display active processes on the terminal
19. **wget –** download files from the internet
20. **whoami-** Create or update passwords for existing users
21. **sort-** sort the file content
22. **cal-** View Calendar in terminal
23. **whereis –** View the exact location of any command types after this command
24. **df –** Check the details of the file system
25. **wc –** Check the lines, word count, and characters in a file using different options