Unix / Linux - File Permission / Access Modes

we will discuss in detail about file permission and access modes in Unix. File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

- **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators

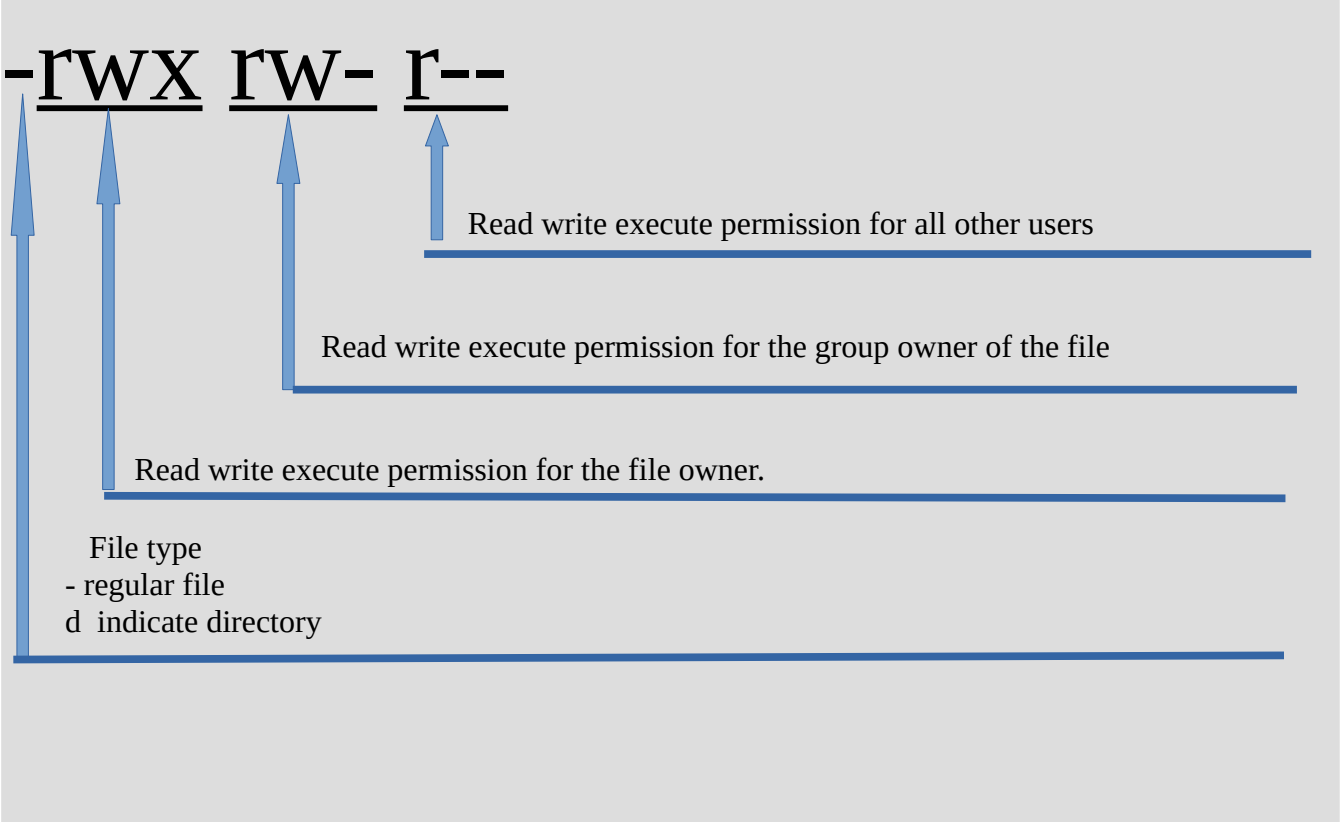
While using **ls -l** command, it displays various information related to file permission as follows

```
[ali@fedora29 ~]$ ls -l
total 100
drwxr-xr-x. 2 ali    ali    4096 Feb 19 19:11 Desktop
drwxrwxr-x. 2 ali    ali    4096 Feb 22 16:30 dir1
drwxrwxr-x. 2 ali    ali    4096 Feb 22 16:30 dir11

drwxrwxr-x. 2 ali    ali    4096 Feb 22 16:30 dir33
-rw-rw-r--. 1 ali    ali      18 Feb 25 18:57 doc1.txt
-rw-rw-r--. 1 ali    ali       0 Feb 22 16:32 doc2.txt
-rw-rw-r--. 1 ali    ali       0 Feb 22 16:32 doc3.txt
drwxr-xr-x. 2 ali    ali    4096 Feb 19 19:11 Documents
drwxr-xr-x. 2 ali    ali    4096 Feb 19 19:11 Downloads
-rw-rw-r--. 1 ali    ali      40 Feb 25 20:25 error.txt

-rwxrw-r--. 1 ali    ali      55 Feb 22 17:19 filetest1.txt
-rw-rw-r--. 1 ali    ali      15 Feb 25 20:14 foo.txt
drwxr-xr-x. 2 ali    ali    4096 Feb 19 19:11 Music
lrwxrwxrwx. 1 ali    ali       6 Feb 28 18:59 mydir -> /dir1/
-rw-rw-r--. 1 ali    ali      40 Feb 25 20:31 output.txt
```

In the diagram below, we see how the first portion of the listing is interpreted. It consists of a character indicating the file type, followed by three sets of three characters that convey the reading, writing and execution permission for the owner, group, and everybody else



Unix security

Unix security model is based on the discretionary access control (DAC) model, which enables users to configure who can access the resources that they "own". Each user can control which other users can access the files that they create. This enables users to grant permissions, without involving a system admin. This is the type of security that has traditionally been built into most consumer OSs such as Windows and Unix

File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

Read

Grants the capability to read, i.e., view the contents of the file.

Write

Grants the capability to modify, or remove the content of the file.

Execute

User with execute permissions can run a file as a program.

Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −

Read

Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.

Write

Access means that the user can add or delete files from the directory.

Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

No.	Sample	Chmod operator & Description
1	+	Adds the designated permission(s) to a file or directory.
2	-	Removes the designated permission(s) from a file or directory.
3	=	Sets the designated permission(s).

Syntax

```
$ chmod options permissions file name
```

Using symbolic values to add, remove the file permission

u for user , g for group , o for others a for all ; r for read , w for write , x for execute , + , – & = for adding , removing and assigning r w x permissions

```
chmod o+wx testfile

chmod u-x testfile

chmod g = rx testfile

chmod o+wx,u-x,g = rx testfile

chmod u=rwx,g=rx,o=r testfile
```

```
[ali@fedora29 dir1]$ ls -l
total 0
-rw-rw-r--. 1 ali ali 0 Mar 10 21:01 file1
[ali@fedora29 dir1]$ chmod o+wx file1
[ali@fedora29 dir1]$ ls -l
total 0
-rw-rw-rwx. 1 ali ali 0 Mar 10 21:01 file1
[ali@fedora29 dir1]$ chmod g=x file1
[ali@fedora29 dir1]$ ls -l
total 0
-rw---xrwx. 1 ali ali 0 Mar 10 21:01 file1
[ali@fedora29 dir1]$ chmod g=rx file1
[ali@fedora29 dir1]$ ls -l
total 0
-rw-r-xrwx. 1 ali ali 0 Mar 10 21:01 file1
[ali@fedora29 dir1]$ touch file2
[ali@fedora29 dir1]$ ls-l
bash: ls-l: command not found...
[ali@fedora29 dir1]$ ls -l
total 0
-rw-r-xrwx. 1 ali ali 0 Mar 10 21:01 file1
-rw-rw-r--. 1 ali ali 0 Mar 10 21:04 file2
[ali@fedora29 dir1]$ chmod o+wx , u+x, g=rwx file2                    ## don't  use spaces
chmod: cannot access ',u+x,g=rwx': No such file or directory
[ali@fedora29 dir1]$ chmod o+wx ,u+x, g=rwx file
[ali@fedora29 dir1]$ chmod  u+x,o+wx,g=rwx file2
[ali@fedora29 dir1]$ ls -l
total 0
-rw-r-xrwx. 1 ali ali 0 Mar 10 21:01 file1
-rwxrwxrwx. 1 ali ali 0 Mar 10 21:04 file2
[ali@fedora29 dir1]$ touch file3
[ali@fedora29 dir1]$ chmod u=rwx,g=rw,o=r file3
[ali@fedora29 dir1]$ ls -l
total 0
-rw-r-xrwx. 1 ali ali 0 Mar 10 21:01 file1
-rwxrwxrwx. 1 ali ali 0 Mar 10 21:04 file2
-rwxrw-r--. 1 ali ali 0 Mar 10 21:08 file3
```

Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Example, rwx = binary 111 = (4 + 2 + 1) = 7
Likewise, r-x = binary 101 = (4 + 1) = 5
Therefore, "-rwxr-xr-x" = 755.

```
[ali@fedora29 dir1]$ chmod u=r,g=,o= file3
[ali@fedora29 dir1]$ ls -l
total 0
-rw-r-xrwx. 1 ali ali 0 Mar 10 21:01 file1
-rwxrwxrwx. 1 ali ali 0 Mar 10 21:04 file2
-r--------. 1 ali ali 0 Mar 10 21:08 file3
[ali@fedora29 dir1]$ chmod 644 file3
[ali@fedora29 dir1]$ ls -l
total 0
-rw-r-xrwx. 1 ali ali 0 Mar 10 21:01 file1
-rwxrwxrwx. 1 ali ali 0 Mar 10 21:04 file2
-rw-r--r--. 1 ali ali 0 Mar 10 21:08 file3
[ali@fedora29 dir1]$ chmod 644 file3 file1 file2
[ali@fedora29 dir1]$ ls -l
total 0
-rw-r--r--. 1 ali ali 0 Mar 10 21:01 file1
-rw-r--r--. 1 ali ali 0 Mar 10 21:04 file2
-rw-r--r--. 1 ali ali 0 Mar 10 21:08 file3
[ali@fedora29 dir1]$ chmod 750  file3 file1 file2
[ali@fedora29 dir1]$ ls -l
total 0
-rwxr-x---. 1 ali ali 0 Mar 10 21:01 file1
-rwxr-x---. 1 ali ali 0 Mar 10 21:04 file2
-rwxr-x---. 1 ali ali 0 Mar 10 21:08 file3
[ali@fedora29 dir1]$
[ali@fedora29 dir1]$ chmod 777  file3 file1 file2
[ali@fedora29 dir1]$ ls -l
total 0
-rwxrwxrwx. 1 ali ali 0 Mar 10 21:01 file1
-rwxrwxrwx. 1 ali ali 0 Mar 10 21:04 file2
-rwxrwxrwx. 1 ali ali 0 Mar 10 21:08 file3
[ali@fedora29 dir1]$
```

Changing Owners and Groups

While creating an account on Unix, it assigns a **owner ID** and a **group ID** to each user. All the permissions mentioned above are also assigned based on the Owner and the Groups.

Two commands are available to change the owner and the group of files −

- **chown** − The **chown** command stands for **"change owner"** and is used to change the owner of a file.
- **chgrp** − The **chgrp** command stands for **"change group"** and is used to change the group of a file.

Changing Ownership

The **chown** command changes the ownership of a file. The basic syntax is as follows

```
[hayder@fedora29 sharefolder]$ touch file1 file2 file3
[hayder@fedora29 sharefolder]$ ls -l
total 0
-rw-rw-r--. 1 hayder hayder 0 Mar 10 21:33 file1
-rw-rw-r--. 1 hayder hayder 0 Mar 10 21:33 file2
-rw-rw-r--. 1 hayder hayder 0 Mar 10 21:33 file3
[hayder@fedora29 sharefolder]$ chown ali file1
chown: changing ownership of 'file1': Operation not permitted
[hayder@fedora29 sharefolder]$ sudo chown ali file1
[hayder@fedora29 sharefolder]$ ls -l
total 0
-rw-rw-r--. 1 ali    hayder 0 Mar 10 21:33 file1
-rw-rw-r--. 1 hayder hayder 0 Mar 10 21:33 file2
-rw-rw-r--. 1 hayder hayder 0 Mar 10 21:33 file3
[hayder@fedora29 sharefolder]$ sudo chown ali file1 file2 file3
[hayder@fedora29 sharefolder]$ ls -l
total 0
-rw-rw-r--. 1 ali hayder 0 Mar 10 21:33 file1
-rw-rw-r--. 1 ali hayder 0 Mar 10 21:33 file2
-rw-rw-r--. 1 ali hayder 0 Mar 10 21:33 file3
```

NOTE − The super user, root, has the unrestricted capability to change the ownership of any file but normal users can change the ownership of only those files that they own

Changing Group Ownership

The **chgrp** command changes the group ownership of a file. The basic syntax is as follows

```
[hayder@fedora29 sharefolder]$ ls -l
total 0
-rw-rw-r--. 1 ali hayder 0 Mar 10 21:33 file1
-rw-rw-r--. 1 ali hayder 0 Mar 10 21:33 file2
-rw-rw-r--. 1 ali hayder 0 Mar 10 21:33 file3
[hayder@fedora29 sharefolder]$ sudo chgrp ali file1 file2 file3
[sudo] password for hayder:
[hayder@fedora29 sharefolder]$ ls -l
total 0
-rw-rw-r--. 1 ali ali 0 Mar 10 21:33 file1
-rw-rw-r--. 1 ali ali 0 Mar 10 21:33 file2
-rw-rw-r--. 1 ali ali 0 Mar 10 21:33 file3
[hayder@fedora29 sharefolder]$ sudo chgrp root file1 file2 file3
[hayder@fedora29 sharefolder]$ ls -l
total 0
-rw-rw-r--. 1 ali root 0 Mar 10 21:33 file1
-rw-rw-r--. 1 ali root 0 Mar 10 21:33 file2
-rw-rw-r--. 1 ali root 0 Mar 10 21:33 file3
[hayder@fedora29 sharefolder]$ ls -l
total 0
-rw-rw-r--. 1 ali root 0 Mar 10 21:33 file1
-rw-rw-r--. 1 ali root 0 Mar 10 21:33 file2
-rw-rw-r--. 1 ali root 0 Mar 10 21:33 file3
[hayder@fedora29 sharefolder]$ sudo chown hayder:hayder  file1 file2 file3
[hayder@fedora29 sharefolder]$ ls -l
total 0
-rw-rw-r--. 1 hayder hayder 0 Mar 10 21:33 file1
-rw-rw-r--. 1 hayder hayder 0 Mar 10 21:33 file2
-rw-rw-r--. 1 hayder hayder 0 Mar 10 21:33 file3
```

SUID and SGID File Permission

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the **passwd** command, your new password is stored in the file **/etc/shadow**.

As a regular user, you do not have **read** or **write** access to this file for security reasons, but when you change your password, you need to have the write permission to this file. This means that the **passwd** program has to give you additional permissions so that you can write to the file **/etc/shadow**.

Additional permissions are given to programs via a mechanism known as the **Set User ID (SUID)** and **Set Group ID (SGID)** bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is the case with SGID as well. Normally, programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter **"s"** if the permission is available. The SUID **"s"** bit will be located in the permission bits where the owners' **execute** permission normally resides.

For example, the command −

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x  1   root   bin  19031 Feb 7 13:47  /usr/bin/passwd*
$
```

Shows that the SUID bit is set and that the command is owned by the root. A capital letter **S** in the execute position instead of a lowercase **s** indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users −

- The owner of the sticky directory
- The owner of the file being removed
- The super user, root

To set the SUID and SGID bits for any directory try the following command

```
[root@fedora29 tmp]# mkdir  sharefolder
[root@fedora29 tmp]# ls -ld sharefolder/
drwxr-xr-x. 2 root root 40 Mar 10 22:03 sharefolder/
[root@fedora29 tmp]# chmod ug+s sharefolder/
[root@fedora29 tmp]# ls -ld sharefolder/
drwsr-sr-x. 2 root root 40 Mar 10 22:03 sharefolder/
[root@fedora29 tmp]#
```