hitbullseye

Files and Directory Permissions (chmod)

Linux is a clone of UNIX, the multi-user operating system which can be accessed by many users simultaneously. Linux can also be used in mainframes and servers without any modifications. But this raises security concerns as an unsolicited or malign user can corrupt, change or remove crucial data. For effective security, Linux divides authorization into 2 levels.

1. **Ownership**

2. **Permission**

In this Linux file commands tutorial, you will learn-

- Linux File Ownership

- Linux File Permissions

- Changing file/directory permissions in Linux Using 'chmod' command

- Absolute(Numeric) Mode in Linux

- Symbolic Mode in Linux

- Changing Ownership and Group in Linux

The concept of Linux File **permission** and **ownership** is crucial in Linux. Here, we will explain Linux permissions and ownership and will discuss both of them. Let us start with the **Ownership.**

Linux File Ownership

Every file and directory on your Unix/Linux system is assigned 3 types of owner, given below.

- **User:** A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

- **Group:** A user- group can contain multiple users. All users belonging to a group will have the same Linux group permissions access to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

- **Other:** Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Now, the big question arises how does **Linux distinguish** between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like you do not want your

colleague, who works on your <u>Linux computer</u>, to view your images. This is where **Permissions** set in, and they define **user behavior**.

Let us understand the **Permission system** on Linux.

Linux File Permissions

Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

- **Read:** This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.
- **Write:** The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
- **Execute:** In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to

see/modify the program code(provided read & write permissions are set), but not run it.

Owners assigned Permission On Every File and Directory
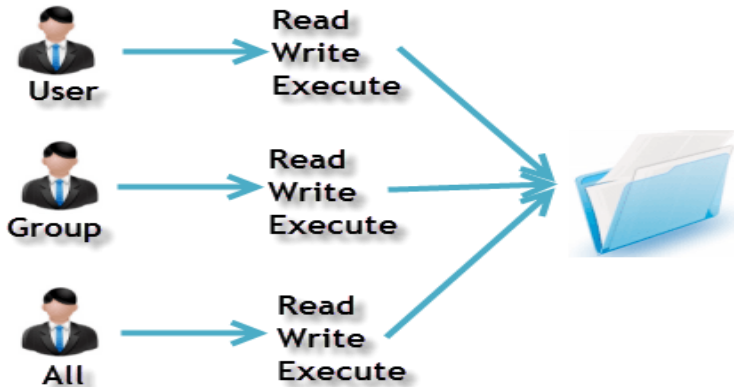
File Permissions in Linux/Unix

Let's see file permissions in Linux with examples:

ls – l on terminal gives

ls - l

File type and Access Permissions.

```
home@VirtualBox: ~
home@VirtualBox:~$ ls -l
-rw-rw-r-- 1 home home     0 2012-08-30 19:06 My File
```

Here, we have highlighted **'-rw-rw-r–'**and this weird looking code is the one that tells us about the Unix permissions given to the owner, user group and the world. Here, the first **'–'** implies that we have selected a file.p>

```
- rw - rw - r - -
```
indicates
file

Else, if it were a directory, **d** would have been shown.

```
            d represents directory
drwxr-xr-x 2 ubuntu ubuntu 80 Sep  6 07:27 Desktop
```

The characters are pretty easy to remember.

r =   read   permission
w =   write   permission
x =   execute   permission
– = no permission

Let us look at it this way. The first part of the code is **'rw-'**. This suggests that the owner 'Home' can:

```
- rw - rw - r - -
```
no execute
permission

- Read the file

- Write or edit the file

- He cannot execute the file since the execute bit is set to '-'.

By design, many Linux distributions like Fedora, CentOS, Ubuntu, etc. will add users to a group of the same group name as the user name. Thus, a user 'tom' is added to a group named 'tom'. The second part is **'rw-'.** It for the user group 'Home' and group-members can:

- Read the file
- Write or edit the file

The third part is for the world which means any user. It says **'r–'.** This means the user can only:

- Read the file

> ➢ **Changing file/directory permissions in Linux Using 'chmod' command**

Say you do not want your colleague to see your personal images. This can be achieved by changing file permissions. We can use the '**chmod'** command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world.

Syntax:

chmod permissions filename

There are 2 ways to use the command −

1. **Absolute mode**
2. **Symbolic mode**

Absolute (Numeric) Mode in Linux

In this mode, file **permissions are not represented as characters but a three-digit octal number**. The table below gives numbers for all for permissions types.

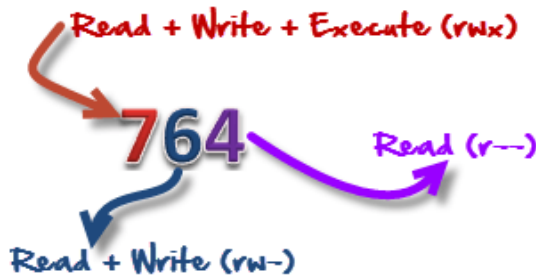| Number	Permission Type	Symbol
0	No Permission	—
1	Execute	−x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r−
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rwx

Let's see the chmod permissions command in action.

Checking Current File Permissions

```
ubuntu@ubuntu:~$ ls -l sample
-rw-rw-r-- 1 ubuntu ubuntu 15 Sep  6 08:00 sample
```

chmod 764 and checking permissions again

```
ubuntu@ubuntu:~$ chmod 764 sample
ubuntu@ubuntu:~$ ls -l sample
-rwxrw-r-- 1 ubuntu ubuntu 15 Sep  6 08:00 sample
```

In the above-given terminal window, we have changed the permissions of the file 'sample to '764'.

Read + Write + Execute (rwx)

764

Read (r--)

Read + Write (rw-)

'764' absolute code says the following:

- Owner can read, write and execute
- Usergroup can read and write
- World can only read

This is shown as '-rwxrw-r−

This is how you can change user permissions in Linux on file by assigning an absolute number. **Symbolic Mode in Linux**

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the Unix file permissions.

Operator	Description
+	Adds a permission to a file or directory
−	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

The various owners are represented as −

User Denotations	
u	user/owner
g	group
o	other
a	all

We will not be using permissions in numbers like 755 but characters like rwx.

Let's look into an example

Current File Permissions
```
home@VirtualBox:~$ ls -l sample
-rw-rw-r-- 1 home home 55 2012-09-10 10:59 sample
```

Setting permissions to the 'other' users
```
home@VirtualBox:~$ chmod o=rwx sample
home@VirtualBox:~$ ls -l sample
-rw-rw-rwx 1 home home 55 2012-09-10 10:59 sample
```

Adding 'execute' permission to the usergroup
```
home@VirtualBox:~$ chmod g+x sample
home@VirtualBox:~$ ls -l sample
-rw-rwxrwx 1 home home 55 2012-09-10 10:59 sample
```

Removing 'read' permission for 'user'
```
home@VirtualBox:~$ chmod u-r sample
home@VirtualBox:~$ ls -l sample
--w-rwxrwx 1 home home 55 2012-09-10 10:59 sample
```

Changing Ownership and Group in Linux

For changing the ownership of a file/directory, you can use the following command:

chown user filename

In case you want to change the user as well as group for a file or directory use the command

chown user:group filename

Let's see this in action

Check the current file ownership using ls -l

```
-rw-rw-r--  1 root n10     18 2012-09-16 18:17 sample.txt
```

Change the file owner to nl00 . You will need sudo

```
n10@N100:~$ sudo chown n100 sample.txt
```

ownership changed to nl00

```
-rw-rw-r--  1 n100 n10     18 2012-09-16 18:17 sample.txt
```

Changing user and group to root 'chown user:group file'

```
n10@N100:~$ sudo chown root:root sample.txt
```

User and Group ownership changed to root

```
-rw-rw-r--  1 root root     18 2012-09-16 18:17 sample.txt
```

In case you want to change group-owner only, use the command

chgrp group_name filename

'**chgrp**' stands for change group.

Check the current file ownership using ls -dl

```
guru99@VirtualBox:~$ ls -dl test1
-rwxrwxrwx 1 root cdrom 0 Oct  6 11:27 test1
```

Change the file owner to root . You will need sudo

```
guru99@VirtualBox:~$ sudo chgrp root test1
```

Group ownership changed to root

```
guru99@VirtualBox:~$ ls -dl test1
-rwxrwxrwx 1 root root 0 Oct  6 11:27 test1
```

Tip

- The file /etc/group contains all the groups defined in the system

- You can use the command "groups" to find all the groups you are a member of

```
guru99@VirtualBox:~$ groups
cdrom guru99 adm sudo dip plugdev lpadmin sambashare
guru99@VirtualBox:~$
```

- You can use the command newgrp to work as a member a group other than your default group

```
guru99@VirtualBox:~$ newgrp cdrom
guru99@VirtualBox:~$ cat > test
this is a test to change group
^C
guru99@VirtualBox:~$ ls -dl test
-rw-rw-r-- 1 guru99 cdrom 31 Oct 11 16:39 test
guru99@VirtualBox:~$
```

- You cannot have 2 groups owning the same file.

- You do not have nested groups in Linux. One group cannot be sub-group of other

- x- eXecuting a directory means Being allowed to "enter" a dir and gain possible access to sub-dirs

- There are other permissions that you can set on Files and Directories which will be covered in a later advanced tutorial