Linux File

Permissions:

A Simple Guide

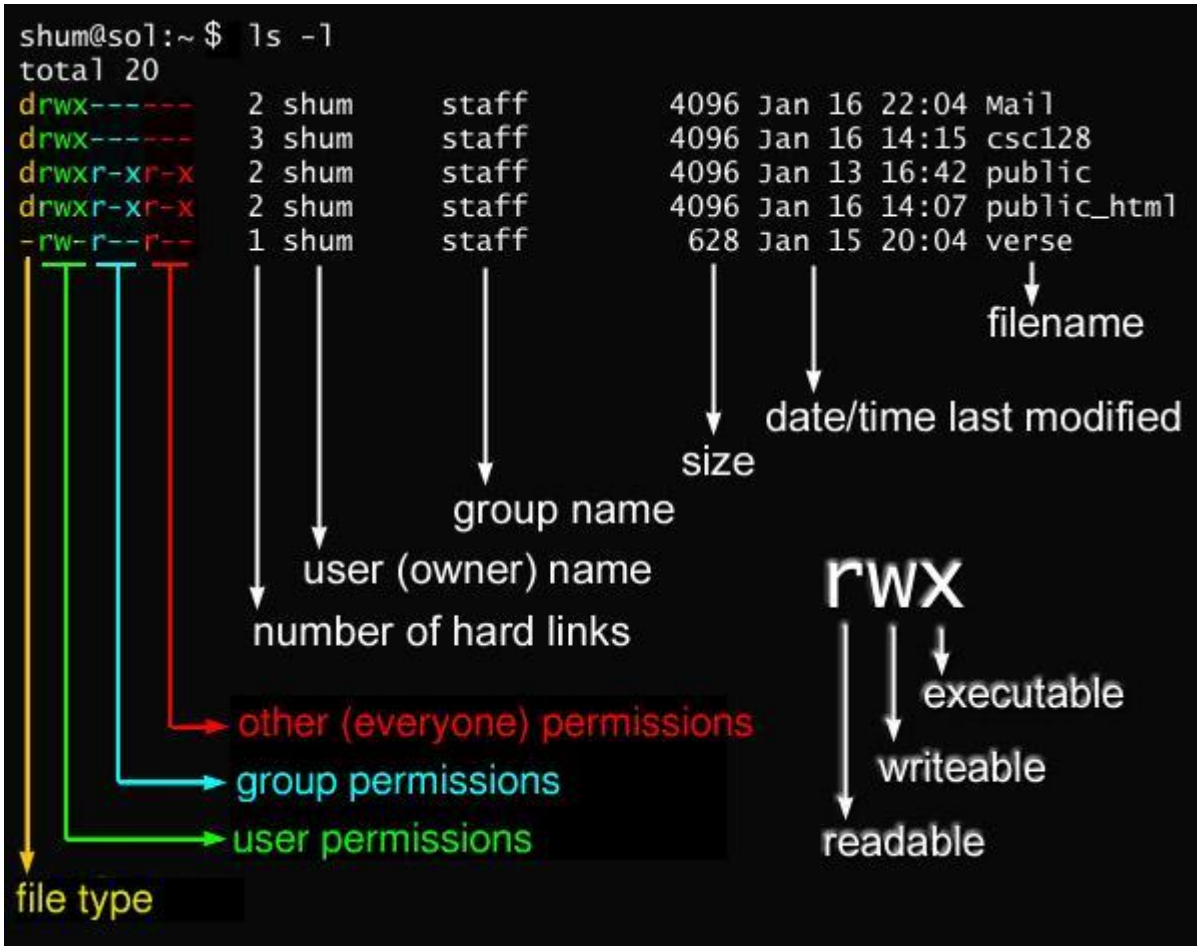
Notes by:- PRASHANT KARNE

We have seen that everything in Linux is file and the statement **"everything in Linux is a file"** is a simplification often used to illustrate a fundamental concept in Linux. In Linux, many system resources, including devices, directories, and processes, are represented and accessed as files. This abstraction allows for a consistent and unified way to interact with various components of the operating system. While not absolutely everything is literally a file in the traditional sense, this concept helps users and developers understand the system's architecture and how to interact with it effectively.

We know that Linux is a multi-user operating system, so it has security to prevent people from accessing each other's confidential files with the help of set of permissions given to the file. Permission of the files can be seen with the help of long list command- `$ls -l`. in below example you can see the permissions given to the file with the help of `$ls -l` command.

```
-rw-r--r--. 1 root root 0 Jun  7 22:52 file.txt
```

here,



- ✓ To read a file or content of the file you need to have read **(r)** permission for that file.
- ✓ To write to a file, to modify a file, or to delete a file, you need to have write **(w)** permission for that file.
- ✓ To run a program or to change to a directory, you need to have execute **(x)** permission for that program or directory.
- ✓ These permissions can be assigned in set of **rwx** to the file owner, group or All other users.

1. **Owner- (u)**: These permissions apply exclusively to the individuals who own the files or directories.
2. **Groups- (g)**: Permissions can be assigned to a specific group of users, impacting only those within that particular group.
3. **All Users- (o)**: These permissions apply universally to all other users on the system, presenting the highest security risk. Assigning permissions to all users should be done cautiously to prevent potential security vulnerabilities.

How to read the permission in Linux

For example:

```
-rw-r-xr--. 1 root root 0 Jun  7 22:52 file.txt
```

Here,

- **"rw-":** the first three characters `rw-`. This means that the owner of the file can "read" it (look at its contents) and "write" it (modify its contents).
- **"r-x":** the second set of three characters "r-x". This means that the members of the group can only read and execute the files.
- **"r--":** The final three characters "r--" shows the permissions allowed to other users who have a User ID on this Linux system. This means anyone in our Linux world can read but cannot modify or execute the files' contents.

Now let's see how to add or remove permission to owner, groups and other:

The command we are going to see for permission modification is:

$ chmod - change Mode

There are two methods, we can change the permission of any file/directory in Linux:

1) Symbolic mode 2) Absolute mode/Numeric Mode

1) Symbolic Mode:

For symbolic mode we user reference for the users as below who are going to access the file/directory:

Reference	Class	Description
`u`	Owner	The user permissions apply only to the owner of the file or directory,
`g`	Group	The group permissions apply only to the group
`o`	Other users	The other permissions apply to all other users on the system
`a`	All three	All three (owner, groups, others)

Symbols we use to add or remove permissions to the file:

Operators	Definition
`+`	Add permissions
`-`	Remove permissions

`=`	Set the permissions to the specified values

1) Let's take an example: We have a file - **'file.txt'** currently with no any permissions to owner, user or group see in below screenshot:

```
---------- 1 root root 0 Jun  8 12:55 file.txt
```

Now if we want to give ***"read"*** **permission to owner (u)** of the file, so first we will start writing our command with chmod and then we have to mention to who we want to give the access **(u, g, o)**, hence we will start typing command as below.

```
$ chmod u
```

here **'u'** stands for **'owner'**, next we have to add (+) read permission to owner so use **'+'** sign

```
$ chmod u+
```

And then specify the permission we want to give either **read, write or execute** in **rwx** format, here in this example we want to give read permission only hence we will add **'r'** in the command:

```
$ chmod u+r
```

And then mention the file/directory name at last so our command will be:

```
$chmod u+r file.txt
```

Let's check the file permissions now – with `ls -l` command:

```
[root@ip-172-31-36-34 mnt]# chmod u+r file.txt
[root@ip-172-31-36-34 mnt]# ls -l
total 0
-r-------- 1 root root 0 Jun  8 12:55 file.txt
```

You can see that we have successfully given Read permission to owner (root) of the file.

2) Now let's take another example: Suppose we want to add **read and write permissions to owner and other users to our file.txt file**

So, start command by chmod and mention the users first, here we have to give access to **owner** of the file and to **other users** it means – **"uo"** so start command as:

```
$ chmod uo
```

Then add **'+'** sign to add permissions

```
$ chmod uo+
```

Now add the permission – read and write - **'rw'**

```
$ chmod uo+rw
```

And at last mention the name of file:

```
$ chmod uo+rw file.txt
```

Now let's check file permissions with `ls -l` command:

```
[root@ip-172-31-36-34 mnt]# chmod uo+rw file.txt
[root@ip-172-31-36-34 mnt]# ls -l
total 0
-rw----rw- 1 root root 0 Jun  8 12:55 file.txt
```

Like wise you can add any permission to any user or group using symbolic mode of file permission.

3) To add all permissions to all users:

```
$ chmod ugo+rwx file.txt OR $chmod a+rwx file.txt
```

```
[root@ip-172-31-36-34 mnt]# chmod a+rwx file.txt
[root@ip-172-31-36-34 mnt]# ls -l
total 0
-rwxrwxrwx 1 root root 0 Jun  8 12:55 file.txt
```

Now let's see how to remove the permissions:

Like we have used **'+'** sing to add permission we can use **'–'** sign to remove permission:

4) For example: we have a **file.txt** file with us with all permissions to all the users and we want to **remove read and write permissions of the group**.

```
-rwxrwxrwx 1 root root 0 Jun  8 12:55 file.txt
```

Start out command with chmod and then mention the owner/group/user to whom we want to change the permission here in this example **'g'**

```
$ chmod g
```

Since we want to remove permissions so type **'-'** sign

```
$ chmod g-
```

mention permissions to remove – **'rw'**

```
$ chmod g-rw
```

And then at last mention the file/directory name:

```
$ chmod g-rw file.txt
```

Check the permissions of the files now, you can see group does not have and read write access for file.txt file

```
[root@ip-172-31-36-34 mnt]# chmod g-rw file.txt
[root@ip-172-31-36-34 mnt]# ls -l
total 0
-rwx--xrwx 1 root root 0 Jun  8 12:55 file.txt
```

5) To remove all permissions of the file for all users:

```
$ chmod a-rwx file.txt
```

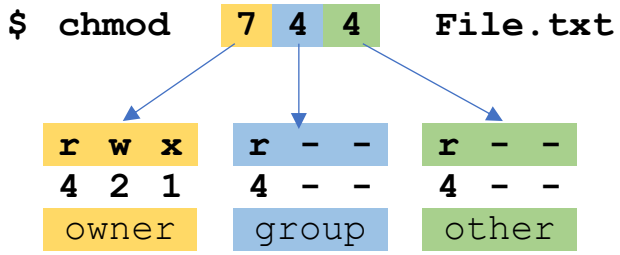
```
[root@ip-172-31-36-34 mnt]# chmod a-rwx file.txt
[root@ip-172-31-36-34 mnt]# ll
total 0
---------- 1 root root 0 Jun  8 12:55 file.txt
```

2) Absolute Mode/Numeric Mode – Octal Representation

Let's see another way of changing file permissions in Linux.

When Linux file permissions are represented by numbers, it's called numeric mode. In numeric mode, a three-digit value represents specific file permissions. These are called octal values. The first digit is for owner permissions, the second digit is for group permissions, and the third is for other users. Each permission has a numeric value assigned to it.

```
-rwxr--r-- 1 root root 0 Jun  8 17:05 file.txt
```



n above example file- **file.txt** has `rwxr--r--` permissions, so numeric values for this are:

for **r (read)** = **4**
for **w (write)** = **2**
for **x (execute)** = **1**

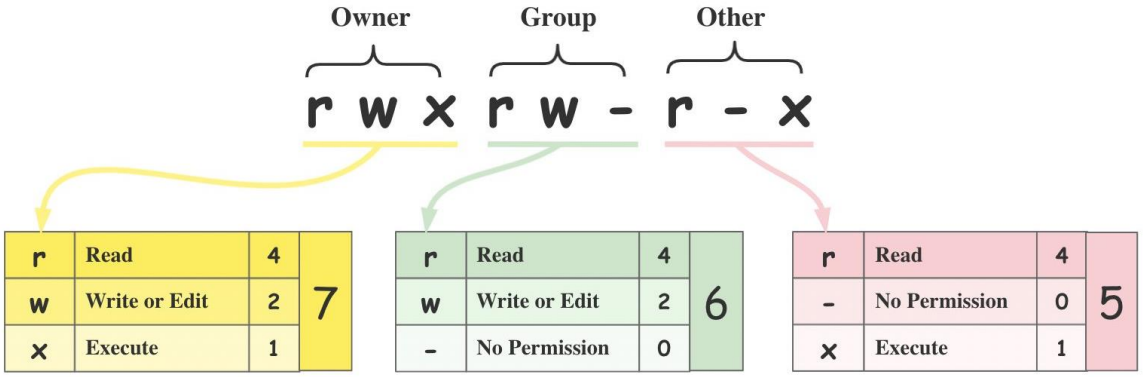
In the permission value 744, the first digit corresponds to the user, the second digit to the group, and the third digit to others. By adding up the value of each user classification, you can find the file permissions.

For example, a file might have read, write, and execute permissions for its owner, and only read permission for all other users. That looks like this:

Owner:	**rwx**	**4+2+1**	**=**	**7**
Group:	**r--**	**4+0+0**	**=**	**4**
Others:	**r--**	**4+0+0**	**=**	**4**

The results produce the three-digit value **744**

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



Let's see another example: Suppose we want to give, below set of permissions:
Owner – read+write - rw
Group – read+execute - rx
Other User – execute - x
So, our command will be:

```
$ chmod  6 5 1  File.txt
```

owner	group	other users
r = 4 w = 2 x = -	r = 4 w = - x = 1	r = - w = - x = 1
(read+write)	(read+execute)	(execute)

What do Linux file permissions actually do?

Till now we have learned how to view file permissions, who they apply to, and how to read what permissions are enabled or disabled to who. Now let's understand what do these permissions actually do in practice?

Read (r): Read permission is used to access the file's contents. You can use a tool like cat or less on the file to display the file contents. You could also use a text editor like Vi or view on the file to display the contents of the file. Read permission is required to make copies of a file, because you need to access the file's contents to make a duplicate of it.

Write (w): Write permission allows you to modify or change the contents of a file. Write permission also allows you to use the redirect or append operators in the shell (> or >>) to change the contents of a file. Without write permission, changes to the file's contents are not permitted

Execute (x): Execute permission allows you to execute the contents of a file. Typically, executables would be things like commands or compiled binary applications. However, execute permission also allows someone to run Bash shell scripts, Python programs, and a variety of interpreted languages.

How do directory permissions work?

Directory file types are indicated with d. Conceptually, permissions operate the same way, but directories interpret these operations differently.

Read (r): Like regular files, this permission allows you to read the contents of the directory. However, that means that you can view the contents (or files) stored within the directory. This permission is required to have things like the ls command work.

Write (w): As with regular files, this allows someone to modify the contents of the directory. When you are changing the contents of the directory, you are either adding files to the directory or removing files from the directory. As such, you must have write permission on a directory to move (mv) or remove (rm) files from it. You also need write permission to create new files (using touch or a file-redirect operator) or copy (cp) files into the directory.

Execute (x): This permission is very different on directories compared to files. Essentially, you can think of it as providing access to the directory. Having execute permission on a directory authorizes you to look at extended information on files in the directory (using ls -l, for instance) but also allows you to change your working directory (using cd) or pass through this directory on your way to a subdirectory underneath. Lacking execute permission on a directory can limit the other permissions in interesting ways. For example, how can you add a new file to a directory (by leveraging the write permission) if you can't access the directory's metadata to store the information for a new, additional file? You cannot. It is for this reason that directory-type files generally offer execute permission to one or more of the user owners, group owner, or others.