## **File Permissions in Linux/Unix**

The concept of **ownership** and **permissions** is crucial in Linux.

### **Ownership of Linux files**

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.

* ***Group***

A user- group can contain multiple users. All users belonging to a group will have the same access permissions 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***

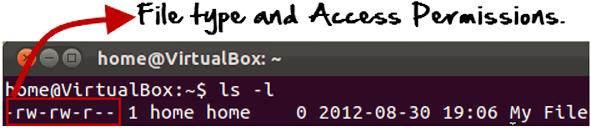
Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

### **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.

**ls - l** on terminal gives



Here, the first '**-**' implies that it is a file Else, if it were a directory, ‘d’ would have been shown.

[File Permissions in Linux/Unix](https://www.guru99.com/images/Directory.png)

The characters are pretty easy to remember.

* **r = read permission**
* **w = write permission**
* **x = execute permission**
* **- = no permission**

The first part of the code is **'rw-**'. This suggests that the owner 'Home' can:

* Read the file
* Write or edit the file
* He cannot execute the file since the execute bit is set to '-'.

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**

'chmod' command is used to change the file/directory permissions

Syntax: **chmod permissions filename**

There are 2 ways to use the command -

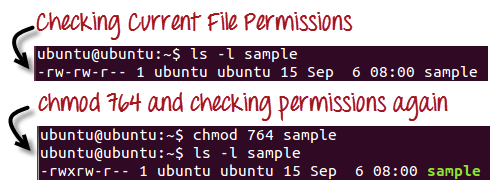
* **Absolute mode**
* **Symbolic mode**

1. ***Absolute(Numeric) Mode***

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 |

[](https://www.guru99.com/images/chmod_new(1).png)

'***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-**

1. ***Symbolic Mode***

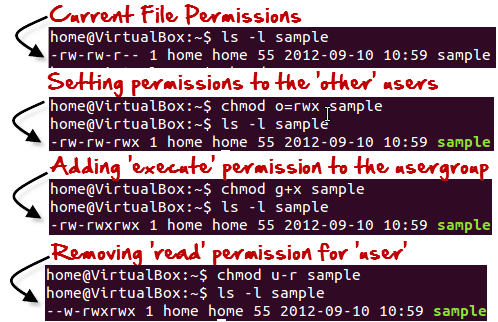
In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

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

[](https://www.guru99.com/images/Symbolic_Mode(1).png)

**Changing Ownership and Group**

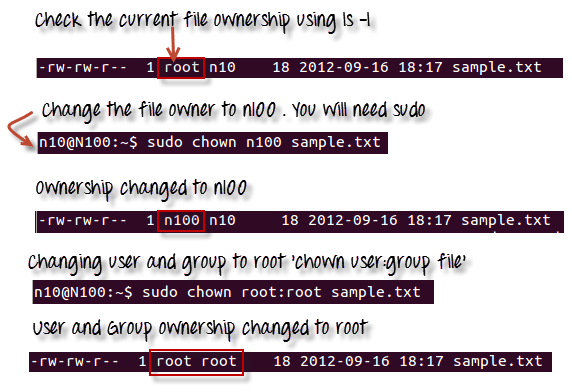
For changing the ownership of a file/directory, you can use the following command:

**chown user**

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

**chown user:group filename**

The 'chgrp' command can change the group ownership **chrgrp group filename**

[](https://www.guru99.com/images/chown_comm(1).png)