# Linux file permissions

### Definition:

Linux file permissions determine who can read, write, or execute a file.

They're based on three categories of users:

1. **Owner (User)** – the person who owns the file.
2. **Group** – a group of users who share the same access rights.
3. **Others** – everyone else.

Example scenario for knowing why do we need file permissions:

As we know, Linux is used as multi-user system. Multiple users work simultaneously on same server but all files and folders are not meant for everyone.

If manager creates a file of instructions and wants everyone to just read that but what if someone changes it?

Exactly to avoid this, Linux file permissions are used. The file’s permissions can be restricted such that only manager can write into it and other users as developer or tester can only read them.

There are ways to do that and we will see that further in the doc.

### Features of Linux file system:

**1. Security:**

They protect sensitive files from unauthorized access. Without permissions, any user could read, modify, or delete critical system or personal files — not ideal.

Example: You don’t want other users reading your SSH keys, modifying system binaries, or deleting your documents.

**2. Multi-user Systems:**

Linux is built for multi-user environments. Permissions help define who can do what, making it possible for different users to share the same system without interfering with each other.

Example: On a shared server, permissions make sure that:

* Alice can’t edit Bob’s files,
* but both can read shared project data.

**3. System Stability:**

They prevent accidental damage. Limiting write or execute permissions **s**tops unintended changes or execution of malicious code.

Example: Even if a user tries to delete or modify a system file, they can’t — unless they have the right permissions (or root access).

**4. Automation & Script Execution:**

Scripts and binaries must have the execute (x) permission to run. This ensures that only verified or intended scripts are executable.

Let’s see a demonstration of default file permissions set by Linux:

We will create two users and a file by each user and see that both of them can just read the content of the file created by the other user but won’t be able to modify the file, delete the file or execute the file.

**Steps:**

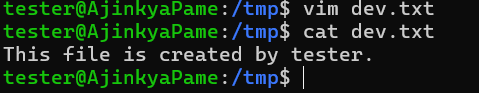
* Add two users as developer and tester (adduser username).
* Enter password and skip other details.
* Check if they are created (cat /etc/passwd).



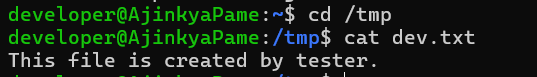
* You will be able to see something like this.
* Open two tables of terminal or powershell.
* On one tab switch to ‘developer’ user and on another tab to ‘tester’ user. (su – username)



* Create a file as a tester in /tmp folder.



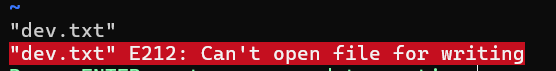
* Go to the tab of developer and try to read that file.



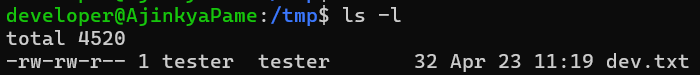
* We can do that.
* Let’s try to change the content means modify the file.
* Even if we try to insert something a warning will appear.



* If we change content and try to save it, we won’t be able to do so and an error will flash.



* This is what comes with default file permissions set by Linux for any user.
* To see the permissions of each file run command “ls -l”.



* Each of these have a meaning and we will see that now.

Let’s see by an example:  
“-rwxr-xr--”

As we can see in the screenshot, the row starts with 10 characters. Here's how to break that down:

The first character shows the type:

1. - for a regular file
2. d for a directory
3. l for a symbolic link, etc.

The next 9 characters are grouped into three sets:

1. rwx (for the owner)
2. r-x (for the group)
3. r-- (for others)

Each character means:

r = read

w = write

x = execute

- = That particular permission is missing for those user/group/others.,

If we look at our info of ‘dev.txt’ from image, it belongs to user ‘tester’ and group ‘tester’ and permissions are:

rw-: for user means tester can read and write the file.

rw-: for group means users belonging to tester group can read and write the file.

r--: for others means users other than tester and belonging to tester group can only read the file.

As per these permissions only as user ‘developer’ is part of others so it could only read the file but not write it.

Now we have seen the types of permissions and how do they work, but we don’t know how to modify them means let’s say how can we allow ‘developer’ to write that ‘dev.txt’ file.

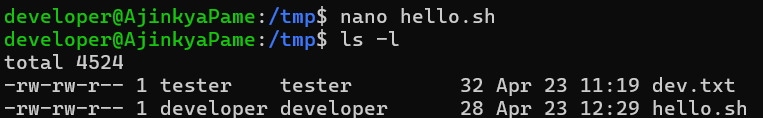
To do so, we have a command called ‘chmod’.

chmod:

chmod command is used to modify the permissions of a file for user, group or others.

To see the working of ‘chmod’ we will create a .sh file as a developer and try to execute that file before and after modifying the permissions using ‘chmod’.

* Create a shell script file as developer and check for the permissions.



* Even though file is created by developer, it is not allowed to execute that and can only read or write it.



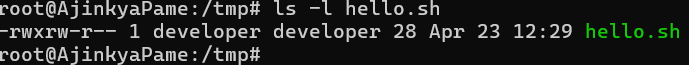
* Let’s modify the permission of that file for developer to execute user by switching to root user, we can do that using ‘sudo’ command too but I am doing by switching to root user.



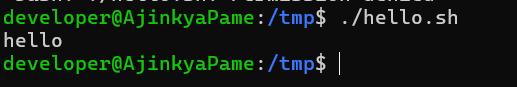
* Before executing the file let’s see what this command is doing and what each of those character mean?
* chmod => Change modifications

**Letter format:**

* u=rwx => For user (in our case developer as it created that file) give the permissions to read, write and execute too.
* hello.sh => Above changes for file hello.sh.
* Check for permissions of that file now.



* Let’s try to execute that file.



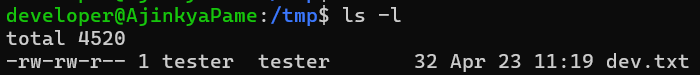
* This time I could do it.
* In this way chmod works to modify the permissions of a file or directory.
* chmod u=rw filename => user can just read and write.
* chmod g=x filename => Group members can only execute the file.
* chmod o=rx filename => Others can read and execute the file but not write anything to it.

**Number format:**

* **r = 4**
* **w = 2**
* **x = 1**
* If we want to give “rwx” permissions to everyone (user, group and others), we just need to use command “chmod 777 filename”.
* If you notice how the number 7 came and why thrice so let’s divide it:
  + 7 = 4+2+1 means permission to read + write + execute
  + The reason to use thrice is first 7 is for user, 2nd one for group and last one for others.
* Let’s see some other examples:
  + chmod 400 filename:
    - Allow only user to only read the file and no other permission to user as well as others and group.
  + chmod 765 filename:
    - Allow user all permissions, group members can read and write only and others can read and execute but not write/modify the file.

So, number system and letter system are two ways to change the permissions of files or folders like read, write and execute.

This was about the permissions of file to someone but we are giving permissions based on ownership means we are giving read permission to the user means the person who created the file means owner of the file.



If we look at this file details carefully its owner is user “tester” and group too “tester”.

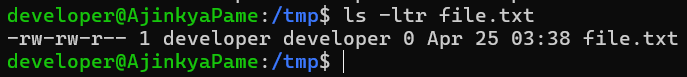
What if we want to change the ownership of the file to someone else, in this case the 2nd command of file permissions comes into play called “chown”.

### chown:

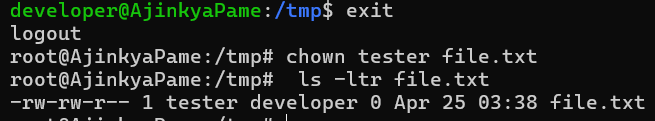
* Using ‘chown’ command, we can change the ownership of a file or folder.
* As we discussed owner can be user and group so there are two ways to assign it:
  + chown username filename/folder
  + chown username: groupname filename/folder
* Ownership can be changed by root user only.
* To change ownership of files in the folder recursively:
  + chown -R developer: developer /var/www

Let’s create a file in /tmp folder being developer and change its ownership to tester.

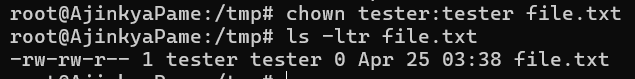
* Switch to developer, cd to /tmp and create a file.



* Switch to root user and change the ownership to tester.



* We have changed owner of the file to tester but group is still developer, so let’s change both to ‘tester’.



or we can do “chown :tester file.txt”.

### A question regularly asked about file ownership/permissions:

#### If a file is present in “Frontend” folder and user “developer” is owner of the file but it does not have permission to Frontend folder, will developer be able to access that file?

Clear answer of this question is **“NO”**.

See by example, if you own a locker in bank but not allowed to enter the bank, is there any meaning to have ownership of the locker if you cannot enter the bank premises.