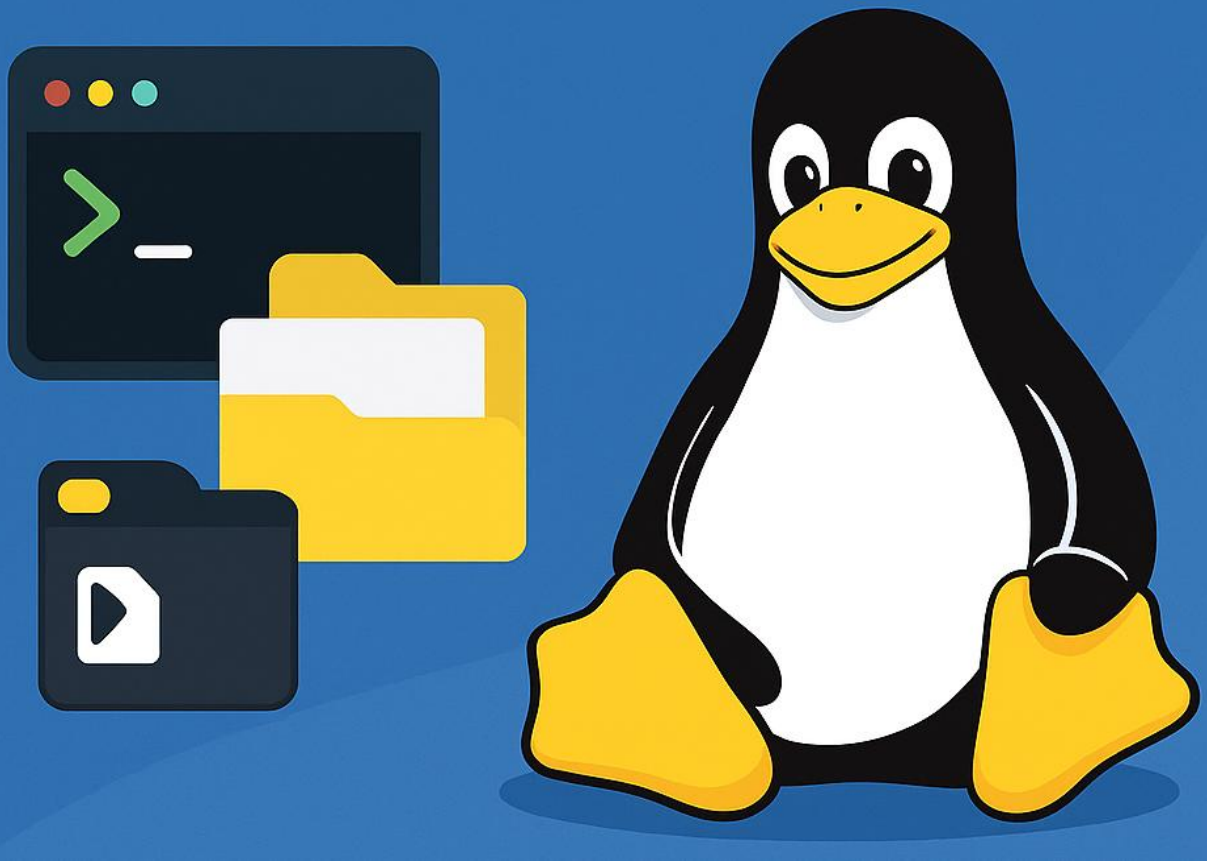
# FILE OWNERSHIP & FILE PERMISSIONS IN LINUX

Swipe →

A COMPLETE PRACTICAL GUIDE
FOR BEGINNERS & PROFESSIONALLS



# 📌 1. Introduction

Linux is a multi-user operating system. This means that **multiple users** can log into the same machine and interact with the same system resources.
To avoid chaos and ensure resource protection, Linux uses two core concepts:

- **File Ownership**
- **File Permissions**

Together, they define **who can access**, **modify**, or **execute** a file or directory.

This guide explains these concepts clearly with examples and visuals — making it perfect for professionals, students, and anyone preparing for DevOps/SRE/Linux roles.

---

# ❖ FILE OWNERSHIP IN LINUX

Every file and directory has **three types of owners**:

### 1) User (Owner)

The user who creates the file automatically becomes its owner.
Example: If user abhi creates a file, they become the owner.

They have the highest control unless ownership is changed.

### 2) Group

A group is a collection of users.

Instead of giving permissions to each user individually, Linux allows assigning permissions to a **group**, making administration efficient — especially in enterprise setups with separate teams like:

- dev
- qa
- security
- sysadmin

📌 A user can belong to **multiple groups**.

✔ Check your groups:

groups

### 3) Others

This refers to *everyone else* on the system who is not the owner and not in the owner's group.

## 📌 Section: File Permissions in Linux

Every file and directory in Linux has the following three permissions for the three types of owners:

**Permissions for Files**

- **Read (r)** – View or copy file contents
- **Write (w)** – Modify file content
- **Execute (x)** – Run the file

**Permissions for Directories**

- **Read (r)** – List files
- **Write (w)** – Add/delete files
- **Execute (x)** – Enter directory

- ## Understanding file permissions and ownership in Linux

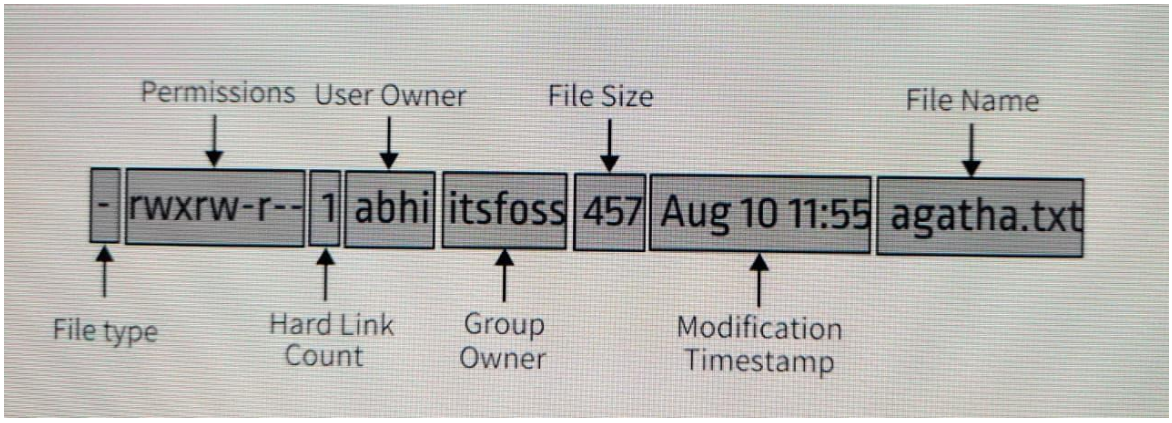
You can check permissions using:

ls -l filename

Example output:

**-rwxrw-r-- 1 abhi itsfoss 457 Aug 10 11:55 agatha.txt**

**Let me explain this output with a picture:**



**Let me further explain the entire output in detail:**

- **File type**: Denotes the type of file. d means directory, – means regular file, l means a symbolic link.
- **Permissions**: This field shows the permission set on a file. I'll explain it in detail in the next section.
- **Hard link count**: Shows if the file has hard links. Default count is one.
- **User**: The user who owns the files.
- **Group**: The group that has access to this file. Only one group can be the owner of a file at a time.
- **File size**: Size of the file in bytes.
- **Modification time**: The date and time the file was last modified.
- **Filename**: Obviously, the name of the file or directory.

---

Now that you have understood the ls -l command output, let's focus on the file permission part.
In the above command, you see the file permission like this in the nine digit **format**:

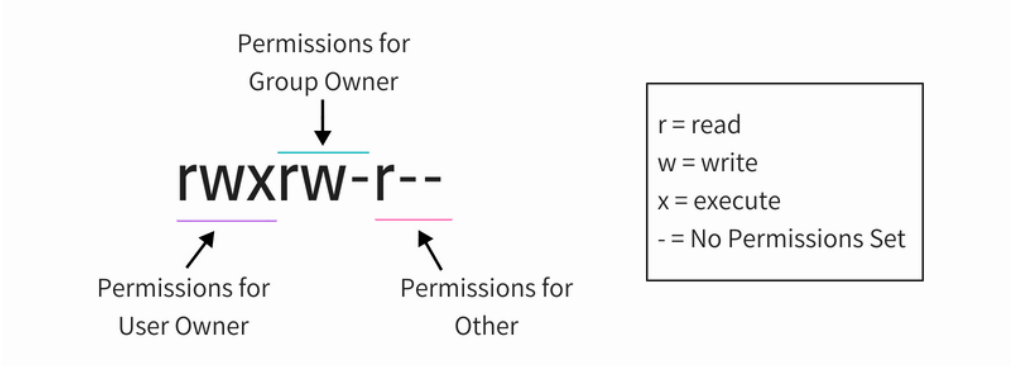
**rwxrw-r--**

Each letter denotes a particular permission:

- r : Read permission
- w : Write permission
- x : Execute permission
- – : No permission set

Permissions are always in the order of read, write and execute, i.e., rwx. And then these permissions are set for all three kind of owners (see the ownership section) in the order of User, Group and Other.

This picture will explain things better:



- **Putting it together**

Now we can read the example correctly:

-rwxrw-r-- 1 abhi itsfoss 457 Aug 10 11:55 agatha.txt

✔ **User (abhi):** rwx
✔ **Group (itsfoss):** rw-
✔ **Others:** r--

Meaning:

- The owner *abhi* can read, write, and execute
- Group members can read and write
- Everyone else can only read
-

---

## ❖ Change file permissions in Linux

You can use **chmod** command for changing the permissions on a file in Linux.

There are two ways to use the chmod command:

- Absolute mode
- Symbolic mode

### ➢ Using chmod in absolute mode

In the absolute mode, permissions are represented in numeric form (octal system to be precise). In this system, each file permission is represented by a number.

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

– (no permission) = 0

With these numeric values, you can combine them and thus one number can be used to represent the entire permission set.

| Number | Permission |
|---|---|
| 0 | — |
| 1 | –x |
| 2 | -w- |
| 3 (i.e. 2+1) | -wx |
| 4 | r– |
| 5 (i.e. 4+1) | r-x |
| 6 (i.e. 4+2) | rw- |
| 7 (i.e. 4+2+1) | rwx |

Examples:

```
chmod 755 file      # rwxr-xr-x
chmod 700 file      # rwx------
chmod 644 file      # rw-r--r--
chmod 600 file      # rw-------
chmod 777 file      # dangerous: rwxrwxrwx
```

**Meaning:**

- **755** → common for scripts & binaries
- **644** → common for configuration files
- **700** → private files
- **777** → avoid! Gives full access to everyone

## ➢ Using chmod in symbolic mode

The problem with the absolute mode is that you should always provide three numbers for all the three owners even if you want to change the permission set for just one owner.

This is where you can use the symbolic mode with chmod command.

In symbolic mode, owners are denoted with the following symbols:

- u = user owner
- g = group owner
- o = other
- a = all (user + group + other)

The symbolic mode uses mathematical operators to perform the permission changes:

- + for adding permissions
- – for removing permissions
- = for overriding existing permissions with new value

**Examples**

```
chmod u+x file       # add execute for user
chmod g-w file       # remove write from group
chmod o=r file       # others get only read
chmod a-x file       # remove execute for everyone
chmod u=rwx,g=rw,o=r file  # full control setup
```

---

## ❖ Change file ownership in Linux

To change the ownership of a file, Linux provides two commands:

- **chown** → change user and group ownership
- **chgrp** → change only group ownership

---

### 1)Using chown (Change Owner)

The basic syntax:

**chown <new_user_name> <filename>**

Example:

sudo chown abhi report.txt

This changes the owner of *report.txt* to user **abhi**.

---

### 2)Change Both Owner and Group

You can modify owner **and** group at the same time:

**chown <new_user_name>:<new_user_group> <filename>**

Example:

sudo chown abhi:developers project.sh

Now:

- User owner → **abhi**
- Group owner → **developers**

---

## 3)Change Only Group Using chown

**chown :<new_user_group> <filename>**

Example:

sudo chown :qa test.log

Only the group changes → user stays the same.

---

## 4)Using chgrp (Change Group Only)

**chgrp <new_group> <filename>**

Example:

sudo chgrp devops /var/log/app.log

Same result as using chown :devops.

---

## 5)Change Ownership Recursively (Very Important)

If you want to change ownership of a **directory and all files inside it**, use:

**sudo chown -R user:group /path/to/directory**

Example:

sudo chown -R abhi:developers /var/www/html

This updates **ALL files & folders** inside /var/www/html.

---

## 6)Real-World Use Cases

### ✔ Allow a developer access to a folder:

**sudo chown abhi:devteam /opt/projects**

### ✔ Repair broken ownership after copying files manually:

**sudo chown -R root:root /etc**

### ✔ Give webserver permission to serve files:

**sudo chown -R www-data:www-data /var/www**

---

## ⚠ Important Notes

- Changing ownership of system files may break the OS
- Root user can change ownership anywhere
- Regular users can only change ownership **if they own the file**
- Use chmod + chown together to configure access properly

---