Gurudev Devkar

[ DevOps Engineer ]

gurudevkar@outlook.com

- **File Permissions**
- **Types of Files**
- **Ownership**
- **Permissions**
- **Special Permission**
- **Umask**

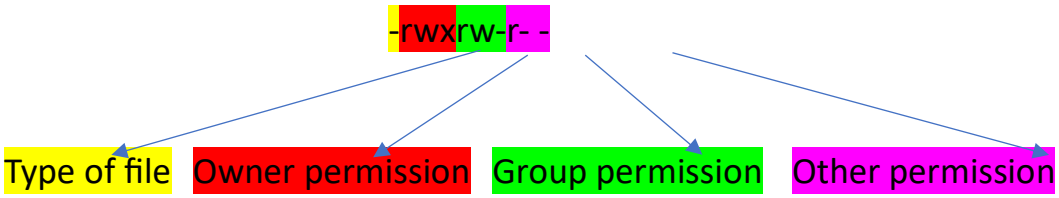
## File Permissions

File permissions define what actions can be performed on a file or directory by different users.

Permissions are set for three categories of users:

1. **Owner (User)**: The person who created the file.
2. **Group**: A set of users who share common permissions.
3. **Others**: All other users on the system.

**## Permissions are represented in three groups of three characters.**




- **First three characters**: Permissions for the owner.
- **Middle three characters**: Permissions for the group.
- **Last three characters**: Permissions for others.

**## Each Character in the groups represents a specific permissions:**

- **r** : Read permission – allows reading the file's contents.
- **w** : Write permission – allows modifying the file.
- **x** : Execute permission – allows running the file as a program.

## Type of Files.

1. **Link File ( l ) :**
   A **link file** is a shortcut or reference to another file. It allows multiple filenames to point to the same file data on disk.

   ❖ **There are two types of links:**

1. **Hard Link**: Points directly to the inode (file metadata). Deleting the original file does not remove the data.
2. **Symbolic Link (Soft Link)**: A pointer to the file name. If the original file is deleted, the symlink becomes broken.

2. **Regular File ( - ) :**

   A **regular file** is the most common type, containing data such as text, scripts, images, or binary executables. It is neither a directory nor a special type of file like links or sockets.

3. **Directory File ( d ) :**
   A **directory file** stores information about files and subdirectories inside it. It acts as a container for other files.

4. **Character File ( c ) :**
   A **character device file** represents hardware devices that send or receive data in a character-by-character (byte stream) fashion, such as a keyboard or terminal.

5. **Block File ( b ) :**

   A **block device file** represents devices that read and write data in blocks (chunks), such as hard drives or USB drives.
   These are also found in /dev.

6. **Fifo / Pipe File ( p ) :**

   A **FIFO (First In First Out)** or **pipe file** is a method for inter-process communication. It allows data flow in a sequence, where the first data written is the first to be read.
   It is used for creating pipelines between different processes.

7. **Socket File ( s ) :**
   A **socket file** is used for inter-process communication over the network or within the same machine. It allows data exchange between processes. Commonly used in networking applications, such as server-client communication.

## ## Ownership

- **ownership** refers to the relationship between a file or directory and the users or groups that have control over it.
- Ownership determines who can read, write, or execute a file based on the permissions assigned to each category.
- We can change the owner or group of the file or directory with "**chown**" and "**chgrp**" command.

  - "chown owner:group filename"
  - "chgrp group filename"

## ## Permissions

- **permissions** define what actions different users can perform on a file or directory.
- Permissions help enforce security and control access, ensuring that only authorized users can read, write, or execute files.
- We can change the permission of file or directory with "chmod" command.

**### There are two ways to do this: symbolic mode and numeric mode.**

### 1. Symbolic Mode :

- **r** : Read
- **w**: Write
- **x**: Execute
- **u**: User (owner)
- **g**: Group
- **o**: Others
- **a**: All (user, group, and others)

- chmod u+x file.txt  # Add execute permission for the owner (user).
- chmod g-w file.txt  # Remove write permission from the group.
- chmod o=r file.txt  # Set read-only permission for others.

2. **Numeric Mode:**
   Each permission is represented by a numeric value

- **r =** 4
- **w =** 2
- **x =** 1

The permissions for each category (owner, group, others) are calculated as the sum of the values:

- **rwx =** 7 (4 + 2 + 1)
- **rw- =** 6 (4 + 2)
- **r-- =** 4

- **chmod 755 file.txt**  # Set permissions to rwxr-xr-x (owner has full access, group and others have read and execute).
- **chmod 644 file.txt**  # Set permissions to rw-r--r-- (owner has read and write, group and others have read-only).

## ## Special Permissions

special permissions provide additional control and functionality over files and directories beyond the standard read, write, and execute permissions.

These special permissions are:

1. **Setuid (Set User ID) : [ 4 ]**
   - The Setuid (Set User ID) permission allows a program to run with the privileges of the file owner (usually root), regardless of who executes it.
   - This is commonly used for programs that require elevated privileges temporarily.

- With the setuid bit set, the program runs as if the root user had executed.
- The setuid permission is represented as an s in the owner's execute position.
- Set or Remove Setuid bit "chmod u+s file" | "chmod u-s file"
  - ✓ If the owner has execute permission: **rws**
  - ✓ If the owner does not have execute permission: **rwS**

2. **Setgid (Set Group ID) : [ 2 ]**
   - The **Setgid** permission works similarly to Setuid but applies to the group. It affects both files and directories:
     - ✓ **For Files**: When set on an executable file, it allows users to execute the file with the group privileges of the file, regardless of the user's own group.
     - ✓ **For Directories**: When set on a directory, files created within the directory inherit the directory's group, rather than the user's default group. This is helpful in shared directories where users need to collaborate on files.
   - The setgid permission is represented as an **s** in the group's **execute** position.
   - "chmod g+s file_or_directory" | "chmod g-s file_or_directory"
     - ✓ If the group has execute permission: **rwxr-sr-x**
     - ✓ If the group does not have execute permission: **rw-r-Sr-x**

3. **Sticky Bit [ 1 ]**
   - The Sticky Bit is typically used on directories to restrict file deletion. When applied to a directory, only the owner of a file can delete or modify the file, even if other users have write access to the directory. This is often used in shared directories like /tmp, where multiple users have write acces.
   - The sticky bit is represented as **t** in the others' **execute** position.
   - "chmod +t directory" | "chmod -t directory"
     - ✓ If others have execute permission: **rwxrwxrwt**
     - ✓ If others do not have execute permission: **rwxrwxrwT**

## ACL ( Access Control List )

- ACLs allow you to set specific permissions for individual users and groups beyond the owner, group, and others.
- This is particularly useful when you need to give or restrict access to certain files or directories for specific users without changing the file's group ownership.
- To check whether a file or directory has an ACL, use the ls -l command. If the file has an ACL, a **+** symbol will appear at the end of the permission string.
- To view the detailed ACL for a file or directory, use the "**getfacl"** command.
- You can set or modify ACLs using the "**setfacl**" command. ACLs can be set for both users and groups.
- You can grant different access levels to multiple users without changing the file's group ownership.

**Commands of ACL:**

1. **getfacl file.txt** : Represent a file.
2. **setfacl -m u:Guru:rwx file.txt** : To give a specific user (Guru) rwx access to a file.
3. **setfacl -m g:devops:rwx file.txt** : To give a specific group (devops) rwx access to a file.
4. **setfacl -x u:Guru file.txt** : removes the ACL entry for the user Guru.
5. **setfacl -b file.txt** : To remove all ACLs

## Umask

Umask (User File Creation Mask) is system setting that determines the default permissions assigned to files and directories when they are created.

The umask value sets restrictions on the permissions, effectively subtracting permissions from the system's default settings for new files and directories.

**Understanding Default Permissions**

- **By default:**
    - New **files** are created with **666** (read and write permissions for everyone) as the base permission.
    - New **directories** are created with **777** (read, write, and execute permissions for everyone) as the base permission.
- **By default umask :**
    - **root** : 022
    - **local user** : 002
- **Calculating File and Directory Permissions with Umask**

    1. **For Files:**
        - Default file permissions: 666
        - Umask: 022
        - Subtract the umask from the default: 666 - 022 = 644
        - Final permission: rw-r--r—
        - So, a newly created file with a umask of 022 will have rw-r--r-- permissions (read and write for the owner, read-only for the group and others).

    2. **For Directories:**
        - Default directory permissions: 777
        - Umask: 022
        - Subtract the umask from the default: 777 - 022 = 755
        - Final permission: rwxr-xr-x
        - A newly created directory with a umask of 022 will have rwxr-xr-x permissions (read, write, and execute for the owner, read and execute for the group and others).

- To set a permanent umask for your user account, you can add the umask command to your shell configuration file (e.g., ~/.bashrc, ~/.bash_profile, or /etc/profile for system-wide settings).
  # Add to your ~/.bashrc or ~/.bash_profile
  **umask 027**

- **commands of umask:**
    - **umask :** To check the default umask
    - **umask 027 :** To set umask.