

File Permissions and Access Control Lists

Day 6 : 90Days of DevOps Challenge

SS

Shivraj Salunkhe · Mar 29, 2023 · 📖 7 min read

In Linux, **file permissions** refer to the settings that determine who can **read**, **write**, or **execute** a file or directory. The permission settings are represented by a series of letters and numbers, and they can be set for the **owner of the file**, the group that the file belongs to, and other users who are not the owner or a member of the group.

The **access control list (ACL)** is an additional layer of file permission settings that allows more fine-grained control over who can access a **file or directory**. The **ACL** allows you to set permissions for specific users or groups, and it can be used in conjunction with the standard **file permission** settings to provide more granular control over file access.

Both **file permissions** and **ACLs** are important for maintaining security and controlling access to sensitive files and directories in a **Linux system**.

- **owner** - The owner of the file or application.
- **"chown"** is used to change the ownership permission of a file or directory.
- **group** - The group that owns the file or application.
- **"chgrp"** - is used to change the group permission of a file or directory.
- **others** - All users with access to the system. (outside the users are in a group)
- **"chmod"** - is used to change the other users permissions of a file or directory.

Task :

-> Create a simple file and do `ls -ltr` to see the details of the files.

here are the steps to create a simple file and check its details using the `ls -ltr` command:

1. Open a terminal window.
2. Navigate to the directory where you want to create the file. You can use the `cd` command to change directories, e.g. `cd Documents` to go to the Documents folder.
3. Create a new file using the `touch` command, followed by the file name. For example: `touch myfile.txt`
4. Use the `ls -ltr` command to list the files in the directory, sorted by modification time with the latest files appearing at the end of the list. You should see the file

you just created, along with its details such as permissions, owner, group, size, and modification time.

Here's an example of the output :

```
-rw-r--r--  1 user  staff    0 Mar 29 13:00 myfile.txt
```

COPY 

In this example, the file `myfile.txt` has read and write permissions for the owner (`user`) and read-only permissions for members of the `staff` group and other users. The file is currently empty (0 bytes), and it was last modified on March 29 at 1:00 PM.

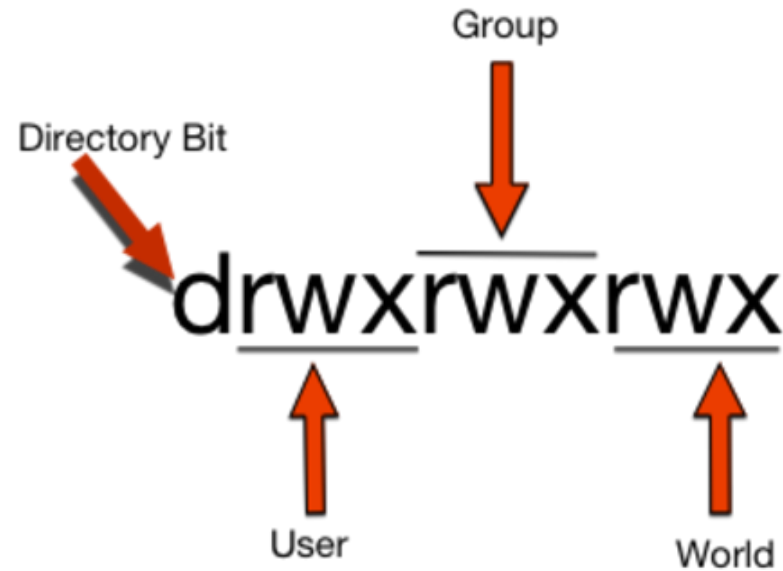
-> Write an article about File Permissions based on your understanding from the notes.

File Permissions in Linux: Understanding **User**, **Group**, and **Other** Permissions.

There are *three* types of file permissions:

1. **Readable (r)** – allows the user to read the file but not make any other edits.
2. **Writable (w)** – allows the user to read and write the files, so they can make changes to the file, but they cannot execute a file if it is a program.
3. **Executable (x)** – allows the user to execute the file. So if this file is a python code or bash script the user would have to have execute permissions to be able to run that

program.



Specifying The Class of a User

There are three classes of users that we can identify when dealing with file permissions. The owner of the file can change permissions for:

1. **The user (u)** - meaning the person using the account (yourself)
2. **The group (g)** - The second set of three permissions
3. **The others (o)** - The final set of three permissions
4. **All (a)** - This is u, g, and o. This changes the permissions for all classes of users.

File permissions are a crucial aspect of maintaining security and controlling access to files and directories in a Linux system. Understanding how file permissions work is essential for system administrators and users who need to manage and share files with others.


In Linux, every file and directory has three types of permissions: user, group, and other. Each of these permission types can be set to **read**, **write**, or **execute**, represented by the letters "**r**", "**w**", and "**x**".

The permission settings are indicated by a series of 10 characters, which can be broken down into four sections:

- The first character indicates the file type. A "-" indicates a regular file, "d" indicates a directory, "l" indicates a symbolic link, and other characters may indicate other file types.
- The next three characters indicate the permissions for the file owner. For example, "**rw**x" means the owner has read, write, and execute permissions, while "**r--**" means the owner can only read the file.
- The next three characters indicate the permissions for the group that the file belongs to. For example, "**r-x**" means members of the group can read and execute the file, but not write to it.
- The final three characters indicate the permissions for all other users who are not the owner or a member of the group. For example, "**r--**" means other users can only read the file.

To change the permissions of a file or directory, you can use the `chmod` command, followed by the permission settings you want to apply. For example, to give the owner of a file read, write, and execute permissions, while removing all permissions for group members and other users, you could use the following command:

```
chmod 700 myfile.txt
```

COPY 

The "7" in the permission settings gives the owner "rwx" permissions, while the "0" removes all permissions for the group and other users.

In conclusion, file permissions are an essential aspect of maintaining security and controlling access to files and directories in a **Linux system**. By understanding the basics of file permissions and how to use the `chmod` command, you can manage your files and directories more effectively and protect sensitive information from **unauthorized access**.

-> **Read about ACL and try out the commands** `getfacl` **and** `setfacl`.


Access Control Lists (ACLs) provide a more fine-grained control over file access permissions in Linux systems. With ACLs, you can set permissions for individual users or groups, allowing you to grant or deny access to files based on specific user accounts.

The `getfacl` and `setfacl` commands are used to view and modify the ACLs for files and directories.

brief overview of these commands:

-> `getfacl`: The `getfacl` command is used to display the ACLs for a file or directory. The syntax is as follows:

```
getfacl filename
```

COPY 

For example, to display the ACLs for a file called "example.txt", you would use the following command:

```
getfacl example.txt
```

COPY 

The output of the command will show you the access permissions for the file or directory, including the owner, group, and any additional users or groups with specific permissions.


-> `setfacl`: The `setfacl` command is used to set or modify the ACLs for a file or directory. The syntax is as follows:

```
setfacl options filename
```

COPY 

For example, to grant read and write access to a specific user, you could use the following command:

```
setfacl -m u:shiv:rw example.txt
```

COPY 

This command adds the "rw" (read and write) permissions for the user "shiv" to the file "example.txt". The "-m" option specifies that we are modifying the ACLs, and "u:shiv" specifies that we are modifying the ACLs for the user "shiv".

You can also use the `-x` option to remove an ACL entry, and the `-b` option to remove all ACLs from a file or directory.

Note that in order to use ACLs, your file system must be mounted with the `acl` option. You can check if your file system supports ACLs by running the command `mount` and checking if the `acl` option is listed for your file system.

In conclusion, ACLs provide a more granular control over file permissions, allowing you to set permissions for individual users and groups. The `getfacl` and `setfacl` commands can be used to view and modify ACLs for files and directories, and are essential tools for managing file access in Linux systems.

Thank You for Reading the Blog.. 

Stay Connect with me for future updates.

connect with me : [linkedin.com/in/shivraj-salunkhe-5881141a4](https://www.linkedin.com/in/shivraj-salunkhe-5881141a4)

follow my blog channel : shivrajofficial.hashnode.dev



Subscribe to my newsletter

Read articles from directly inside your inbox.
Subscribe to the newsletter, and don't miss out.

SUBSCRIBE

Devops

Linux

Learning Journey

file-permission

shell script



WRITTEN BY

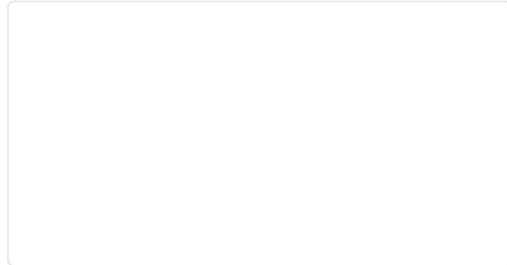
Shivraj Salunkhe

 Follow

Dedicated and hardworking undergraduate student pursuing a degree in Information Technology with a passion for learning, leadership experience, and real-world skills through internships and part-time jobs in IT sector. Actively looking for new Opportunities and committed to do personal and professional growth.

MORE ARTICLES

Shivraj Salunkhe



Understanding package manager and systemctl

What is a package manager in Linux?
In Linux, a package manager is a software tool that helps users...

Shivraj Salunkhe

Advanced Linux Shell Scripting for DevOps Engineers with User management

Advanced Linux Shell Scripting for DevOps Engineers covers a wide range of topics, including :
Auto...

Shivraj Salunkhe

Basic Linux Shell Scripting for DevOps Engineers

What is Kernel ? In computing, a kernel is the core component of an operating system that manages s...

[Archive](#) · [Privacy_policy](#) · [Terms](#)



Publish with Hashnode

Powered by [Hashnode](#) - Home for tech writers and readers