Lab 3

Practicing File Operations​

**Introduction**

The `chmod` command in a Unix-like terminal is used to change the permissions (read, write, execute) of files and directories. It allows you to control who can access and manipulate these files and directories. This report will provide a comprehensive guide on how to use the `chmod` command effectively in the terminal.

**Understanding File Permissions**

Before diving into how to use `chmod`, it's essential to understand the basics of file permissions in Unix-like systems:

- Read (r): If a user has read permission on a file or directory, they can view its contents and list the directory's files.

- Write (w): Write permission allows users to modify the content of a file or create, delete, or rename files within a directory.

- Execute (x): Execute permission is required to run scripts and execute binaries (programs).

File permissions are usually represented as a series of letters or numbers. For example, "rwxr-xr--" signifies read, write, and execute permissions for the owner, read and execute permissions for the group, and read-only permissions for others.

**Using `chmod` Syntax**

The `chmod` command follows this syntax:

chmod [options] mode file

- `options`: Optional flags to modify the behavior of `chmod`.

- `mode`: Specifies the permissions you want to set, often expressed as three or four octal digits or symbolic notation.

- `file`: The file or directory you want to modify the permissions of.

**Changing Permissions with Octal Notation**

Octal notation is a numeric representation of file permissions, where each permission (read, write, execute) is represented by a digit:

- `4` represents read permission.

- `2` represents write permission.

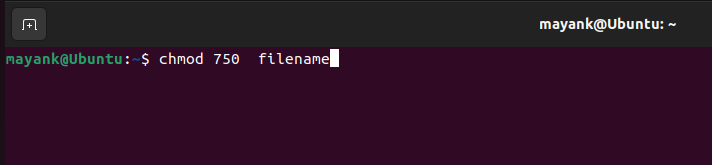
- `1` represents execute permission.

You calculate the desired permission by adding these values:

- 4 (read) + 2 (write) + 1 (execute) = 7 (full permissions).

- 4 (read) + 1 (execute) = 5 (read and execute only).

For example, to give the owner full permissions, the group read-only permissions, and others no permissions, you'd use:



**Changing Permissions with Symbolic Notation**

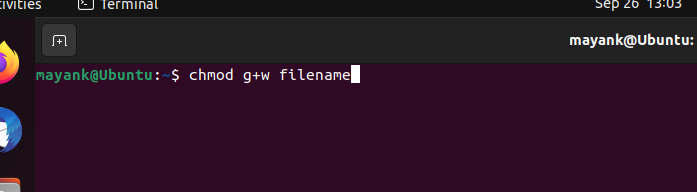
# Symbolic notation is a more human-readable way to set permissions, using letters to represent the user (u), group (g), and others (o), along with the operators (+, -, =) to add, subtract, or set permissions:

- `+` adds permissions.

- `-` removes permissions.

- `=` sets permissions.

For instance, to add write permission for the group, you can use:



Common `chmod` Options

- `-R` or `--recursive`: Apply permissions recursively to all files and directories within the specified path.

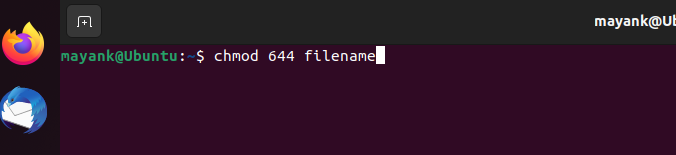
- `-v` or `--verbose`: Display detailed information about changes made.

- `-c` or `--changes`: Report only when changes are made.

Examples

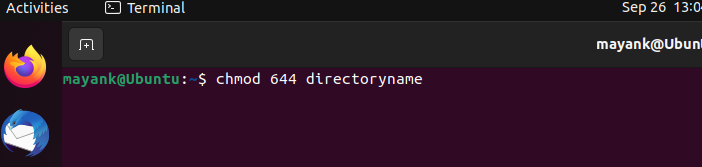
1. Change Permissions of a File:

To give read and write permissions to the owner and read-only permissions to the group and others, use:



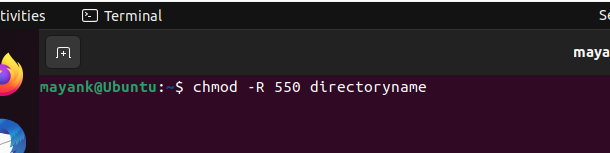
2. Change Permissions of a Directory:

To give full permissions to the owner and read-only permissions to the group and others for a directory, use:



3. Recursively Change Permissions:

To recursively give read and execute permissions to the owner and group, and no permissions to others for a directory and its contents, use:



Conclusion

The `chmod` command is a powerful tool for managing file and directory permissions in Unix-like systems. Understanding how to use it is crucial for maintaining security and access control on your system. By following the syntax and examples provided in this report, you can confidently use `chmod` to control file permissions in the terminal.