**# File Permissions Management in Linux**

Introduction to File Permissions

Linux file permissions determine who can read, write, or execute files and directories. Each file and directory has three levels of permission:

- \*\*Owner (User)\*\*: The creator of the file.

- \*\*Group\*\*: Users belonging to the assigned group.

- \*\*Others\*\*: All other users on the system.

Permissions are represented as:

- \*\*Read (`r` or `4`)\*\* – View file contents.

- \*\*Write (`w` or `2`)\*\* – Modify file contents.

- \*\*Execute (`x` or `1`)\*\* – Run scripts or programs.

To check file permissions, use:

```bash

ls -l filename

```

Output example:

```bash

-rwxr--r-- 1 user group 1234 Mar 28 10:00 myfile.sh

```

## Changing Permissions with `chmod`

### Using Symbolic Mode

Modify permissions using symbols:

- Add (`+`), remove (`-`), or set (`=`) permissions.

Examples:

```bash

chmod u+x filename # Add execute for user

chmod g-w filename # Remove write for group

chmod o=r filename # Set read-only for others

chmod u=rwx,g=rx,o= filename # Set full access for user, read/execute for group, and no access for others

```

### Using Numeric (Octal) Mode

Each permission has a value:

- Read (`4`), Write (`2`), Execute (`1`).

Examples:

```bash

chmod 755 filename # User (rwx), Group (r-x), Others (r-x)

chmod 644 filename # User (rw-), Group (r--), Others (r--)

chmod 700 filename # User (rwx), No access for others

```

## Changing Ownership with `chown`

Modify file owner and group:

```bash

chown newuser filename # Change owner

chown newuser:newgroup filename # Change owner and group

chown :newgroup filename # Change only group

```

Recursively change ownership:

```bash

chown -R newuser:newgroup directory/

```

## Changing Group Ownership with `chgrp`

```bash

chgrp newgroup filename # Change group

chgrp -R newgroup directory/ # Change group recursively

```

## Special Permissions

### SetUID (`s` on user execute bit)

Allows users to run a file with the file owner's permissions.

```bash

chmod u+s filename

```

Example: `/usr/bin/passwd` allows users to change their passwords.

### SetGID (`s` on group execute bit)

Files: Users run the file with the group's permissions.

Directories: Files created inside inherit the group.

```bash

chmod g+s filename # Set on file

chmod g+s directory/ # Set on directory

```

### Sticky Bit (`t` on others execute bit)

Used on directories to allow only the owner to delete their files.

```bash

chmod +t directory/

```

Example: `/tmp` directory.

## Default Permissions: `umask`

`umask` defines default permissions for new files and directories.

Check current umask:

```bash

umask

```

Set a new umask:

```bash

umask 022 # Default: 755 for directories, 644 for files

```

## Conclusion

Understanding file permissions is essential for system security and proper file management. Using `chmod`, `chown`, and `chgrp`, you can control access to files and directories efficiently.