File permissions in Linux

Project description

In this project, I explore various Linux commands to manage file and directory permissions. The tasks include checking file and directory details, understanding the permissions string, changing file and directory permissions, including hidden files.

Check file and directory details

To check details of files and directories, you can use the `ls` command with various options:

- `ls -l`: Lists files and directories with detailed information including permissions, number of links, owner, group, size, and modification date.

```
ls -l
```

- `ls -a`: Lists all files and directories including hidden ones (those starting with a dot).

```
ls -a
```

- `ls -la`: Combines both options to list all files and directories with detailed information.

```
ls -la
```

Example output of `ls -la`:

```
drwxr-xr-x  2 user group 4096 Jun  9 10:00 .
drwxr-xr-x  3 user group 4096 Jun  9 09:58 ..
-rw-r--r--  1 user group    0 Jun  9 10:00 file1.txt
-rw-r--r--  1 user group    0 Jun  9 10:00 .hiddenfile
```

Describe the permissions string

The permissions string is a representation of the access rights associated with files and directories. It consists of 10 characters:

- The first character indicates the type of file (`-` for a regular file, `d` for a directory, `l` for a symbolic link).
- The next nine characters are divided into three sets of three characters, representing the permissions for the owner, group, and others respectively.

Each set includes:

- `r`: Read permission
- `w`: Write permission
- `x`: Execute permission

For example, in `-rwxr-xr--`:

- `-` indicates a regular file.
- `rwx` indicates the owner has read, write, and execute permissions.
- `r-x` indicates the group has read and execute permissions.
- `r--` indicates others have read-only permission.

Describe the Permissions String in Decimal

In Linux, file and directory permissions are often represented in both symbolic and numeric (decimal) formats. The numeric format is also referred to as octal notation because it is based on the base-8 number system. Each set of permissions (read, write, and execute) for the owner, group, and others can be represented by a single digit (0-7).

Here's a breakdown of how the permissions string maps to its numeric (decimal) representation:

Symbolic Representation to Numeric Representation

Each permission (read, write, execute) is assigned a value:

- `r` (read) = 4
- `w` (write) = 2
- `x` (execute) = 1
- `-` (no permission) = 0

For each set of permissions (owner, group, others), you add up the values of the permissions.

Examples

1. **Permission String: `rwxr-xr--`**
    - **Owner (`rwx`)**:
        - Read (r) = 4
        - Write (w) = 2
        - Execute (x) = 1
        - Total = 4 + 2 + 1 = 7
    - **Group (`r-x`)**:
        - Read (r) = 4
        - No write (-) = 0
        - Execute (x) = 1
        - Total = 4 + 0 + 1 = 5
    - **Others (`r--`)**:
        - Read (r) = 4
        - No write (-) = 0
        - No execute (-) = 0
        - Total = 4 + 0 + 0 = 4
    - **Numeric Representation**: `755`
2. **Permission String: `r--r--r--`**
    - **Owner (`r--`)**:
        - Read (r) = 4
        - No write (-) = 0
        - No execute (-) = 0
        - Total = 4 + 0 + 0 = 4
    - **Group (`r--`)**:
        - Read (r) = 4
        - No write (-) = 0
        - No execute (-) = 0
        - Total = 4 + 0 + 0 = 4
    - **Others (`r--`)**:
        - Read (r) = 4
        - No write (-) = 0
        - No execute (-) = 0
        - Total = 4 + 0 + 0 = 4
    - **Numeric Representation**: `444`
3. **Permission String: `rw-r--r--`**
    - **Owner (`rw-`)**:
        - Read (r) = 4

        - Write (w) = 2
        - No execute (-) = 0
        - Total = 4 + 2 + 0 = 6
    - **Group (`r--`)**:
        - Read (r) = 4
        - No write (-) = 0
        - No execute (-) = 0
        - Total = 4 + 0 + 0 = 4
    - **Others (`r--`)**:
        - Read (r) = 4
        - No write (-) = 0
        - No execute (-) = 0
        - Total = 4 + 0 + 0 = 4
    - **Numeric Representation**: `644`

Summary of Common Numeric Values

- `0` = ---
- `1` = --x
- `2` = -w-
- `3` = -wx
- `4` = r--
- `5` = r-x
- `6` = rw-
- `7` = rwx

By understanding this mapping, you can quickly convert symbolic permission strings into their numeric equivalents, which are often used in command-line operations for setting file and directory permissions.

Change file permissions

To change file permissions, use the `chmod` command. You can specify permissions using symbolic or decimal notation.

- Symbolic notation example:

```
chmod u+x file1.txt   # Adds execute permission for the owner
chmod g-w file1.txt   # Removes write permission for the group
chmod o+r file1.txt   # Adds read permission for others
```

- Numeric (octal) notation example:

```
chmod 755 file1.txt   # Sets permissions to rwxr-xr-x
```

Change File Permissions on a Hidden File

Hidden files are those that start with a dot (`.`). Changing permissions on hidden files uses the same `chmod` command.

- Example for a hidden file:

```
chmod 600 .hiddenfile   # Sets permissions to rw-------
```

Change directory permissions

Directory permissions are also managed using the `chmod` command. Execute (`x`) permission on a directory allows a user to enter the directory and access files.

- Example:

```
chmod 755 mydirectory   # Sets permissions to rwxr-xr-x for the
directory
```

Summary

In this project, we have covered the basics of checking file and directory details, understanding and interpreting the permissions string, and modifying file and directory permissions using the `chmod` command. These commands and concepts are fundamental for managing access control and ensuring security in a Linux environment. By mastering these commands, you can effectively manage permissions for files and directories, including hidden ones, maintaining the integrity and confidentiality of your system's data.