Project Description

The goal of this project is to document the core concepts and commands used to manage file and directory permissions in a Linux operating system. By demonstrating how to check, interpret, and change permissions using the `ls` and `chmod` commands, this document serves as a guide for security professionals to properly control access to system resources. This process is fundamental to maintaining system security and adhering to the principle of least privilege.

Check File and Directory Details

The primary command for viewing detailed information about files and directories, including permissions, is `ls` with the **`-la`** options.

Command	Explanation
`ls -la`	Lists all files (`-a`, including hidden files) in a **l**ong format (`-l`) which displays permissions, owner, group, size, and last modified date.
`ls -ld example_dir`	Lists the **d**etails of the **d**irectory itself, rather than its contents. This is important when checking directory permissions.

Export to Sheets

Screenshot of your commands or typed versions of the commands:

Bash
```
# Command to check details for all files in the current directory
ls -la

# Example Output
-rw-r--r-- 1 user group 0 Sep 24 10:00 my_file.txt
drwxr-xr-x 2 user group 4096 Sep 24 10:05 my_dir

# Command to check details for a specific directory
ls -ld my_dir

# Example Output
drwxr-xr-x 2 user group 4096 Sep 24 10:05 my_dir
```

Describe the Permissions String

The file permissions are represented by a **10-character string** at the beginning of the `ls -l` output (e.g., `-rwxr-xr--`).

Position	Character	Description
1	`-` or `d`	**File Type**: `-` for a regular file, `d` for a directory.
2-4	`rwx`	**User/Owner Permissions**: What the file's owner can do.
5-7	`r-x`	**Group Permissions**: What members of the file's group can do.

Position	Character	Description
8-10	`r--`	**Other Permissions**: What all other users on the system can do.

Export to Sheets

Interpreting the 3-Character Sets (rwx)

Each set of three characters (`r`, `w`, `x`) has a corresponding numeric value used in the **Octal (Numeric) Mode** of the `chmod` command:

Permission	Value	Action
r (Read)	**4**	Allows viewing file contents or listing directory contents.
w (Write)	**2**	Allows modifying or deleting the file, or creating/deleting files within a directory.
x (Execute)	**1**	Allows running a file as a program or entering (changing into) a directory.

Export to Sheets

Example Interpretation:

The string `-rwxr-xr--` is numerically represented as **754**.

- **Owner (rwx):** 4+2+1=7 (Read, Write, Execute)
- **Group (r-x):** 4+0+1=5 (Read, Execute)
- **Other (r--):** 4+0+0=4 (Read Only)

Change File Permissions

The **`chmod`** command is used to change file permissions. The most common methods are **Octal Mode (Numeric)** and **Symbolic Mode**.

Using Octal (Numeric) Mode

This method is the most efficient. You use three or four digits to set the exact permissions for Owner, Group, and Others.

Command	Explanation
`chmod 640 my_file.txt`	Sets permissions to: **Owner** (6: rw-), **Group** (4: r--), and **Others** (0: ---).

Export to Sheets

Screenshot of your commands or typed versions of the commands:

Bash
```
# Create a test file
touch test_file.sh
```

```
# Initial permissions (e.g., 644)
ls -l test_file.sh
# -rw-r--r-- 1 user group 0 Sep 24 10:10 test_file.sh

# Change permissions to 700 (rwx for owner, none for group/others)
chmod 700 test_file.sh

# Check the new permissions
ls -l test_file.sh
# -rwx------ 1 user group 0 Sep 24 10:10 test_file.sh
```

Using Symbolic Mode

This method uses letters to add (`+`), remove (`-`), or set (`=`) permissions for specific user classes (`u` for user/owner, `g` for group, `o` for others, `a` for all).

Command	Explanation
`chmod g+w my_file.txt`	**Adds** write permission (`+w`) for the **g**roup (`g`).
`chmod o-x my_file.txt`	**Removes** execute permission (`-x`) for **o**thers (`o`).
`chmod a=r my_file.txt`	**Sets** read-only permission (`=r`) for **a**ll (Owner, Group, and Others).

Export to Sheets

Change File Permissions on a Hidden File

Hidden files (often called dotfiles) are simply files or directories whose names begin with a **dot (`.`)**. By convention, commands like `ls` and GUI file managers do not display them by default to avoid cluttering the view with configuration files. They are not a security mechanism, as anyone can view them using the `ls -a` or `ls -la` commands.

To change permissions on a hidden file, you use the exact same `chmod` command as a regular file, but you must include the leading dot in the filename.

Screenshot of your commands or typed versions of the commands:

Bash
```
# Create a hidden file
touch .hidden_config

# Check the initial permissions
ls -l .hidden_config
# -rw-r--r-- 1 user group 0 Sep 24 10:20 .hidden_config

# Change permissions to remove all access for Group and Others (600)
chmod 600 .hidden_config

# Verify the change
ls -l .hidden_config
# -rw------- 1 user group 0 Sep 24 10:20 .hidden_config
```

Change Directory Permissions

Changing directory permissions uses the same `chmod` command but the permissions have slightly different meanings for a directory:

- **Read (`r`):** Allows you to **list** the files inside the directory (using `ls`).
- **Write (`w`):** Allows you to **create, rename, or delete** files within the directory.
- **Execute (`x`):** Allows you to **enter** the directory (using `cd`) or access its contents. **The `x` bit is essential for any directory you need to access.**

Screenshot of your commands or typed versions of the commands:

Bash
```
# Create a test directory
mkdir projects

# Set standard directory permissions (rwxr-xr-x or 755)
chmod 755 projects

# Remove execute permission from others (rwxr-xr-- or 754)
chmod o-x projects

# Check the directory permissions
ls -ld projects
# drwxr-xr-- 2 user group 4096 Sep 24 10:30 projects
```

Summary

Effective management of **Linux file permissions** is a cornerstone of system security. By utilizing the **`ls -la`** command, a security professional can inspect the **10-character permissions string** to understand access levels for the Owner, Group, and Others. The **`chmod`** command, used with either **Octal (Numeric) or Symbolic Mode**, provides the necessary control to adjust these permissions. This control ensures that sensitive files, including **hidden files** (`.dotfiles`), maintain the proper level of access, thus enforcing the principle of least privilege and protecting the integrity of the system.

Note: (These are achieved with the help of Artificial Intelligence)