Chapter 4: File Permissions & Ownership

1. Why File Permissions Exist in Linux

Purpose (Interview Context)

Linux is a **multi-user operating system**.
File permissions exist to:

- Protect system files
- Prevent unauthorized access
- Control who can read, write, or execute files
- Maintain system security and stability

Interview insight:

Permissions are one of the core security mechanisms in Linux.

---

2. Linux Permission Model (r, w, x)

Each file or directory in Linux has **three permission types**:

Permission	Symbol	Meaning
Read	r	View file contents / list directory
Write	w	Modify file / create or delete files
Execute	x	Run file / access directory

---

3. Permission Categories (VERY IMPORTANT)

Permissions are assigned to **three categories**:

Category	Meaning
User (Owner)	File owner
Group	Group members
Others	Everyone else

Viewing Permissions

```
ls -l file.txt
```

Example output:

```
-rwxr-xr--
```

Breakdown:

```
-   rwx   r-x   r--
|    |     |     |
|  User  Group  Others
|
File type
```

---

4. File Types (First Character Explained)

Symbol	Meaning
-	Regular file
d	Directory
l	Symbolic link
c	Character device
b	Block device

Interview note:

The first character shows the file type.

---

5. Permissions for Files vs Directories

File Permissions

- Read (r): Read file contents
- Write (w): Modify file
- Execute (x): Run the file

Directory Permissions (Common Interview Trap)

Permission	Meaning
Read (r)	List files
Write (w)	Create/delete files
Execute (x)	Enter directory

Important interview line:

Without execute permission on a directory, you cannot access files inside it.

---

6. chmod – Changing Permissions

6.1 Numeric (Octal) Mode

Value	Permission
4	Read
2	Write
1	Execute

Example:

```
chmod 755 script.sh
```

Meaning:

- User: 7 (rwx)
- Group: 5 (r-x)
- Others: 5 (r-x)

6.2 Symbolic Mode

Example:

```
chmod u+x file.sh
chmod g-w file.txt
chmod o+r file.txt
```

Interview note:

Numeric mode is faster; symbolic mode is more readable.

---

7. chown and chgrp

Changing Owner

```
chown user file.txt
```

Changing Group

```
chgrp group file.txt
```

Change Both

```
chown user:group file.txt
```

Interview line:

chown changes ownership, chmod changes permissions.

---

8. Special Permissions (VERY IMPORTANT)

8.1 SUID (Set User ID)

Definition

SUID allows a user to execute a file with the owner's privileges.

Example

```
-rwsr-xr-x
```

Real-life example:

- `/usr/bin/passwd`
- Allows normal users to change passwords

Interview insight:

SUID is required for privileged operations.

8.2 SGID (Set Group ID)

Definition

SGID causes files to be executed with group privileges.

For directories:

- New files inherit group ownership

Real-life example:

- Shared project directories

8.3 Sticky Bit

Definition

Sticky bit prevents users from deleting files they don’t own.

Example

```
drwxrwxrwt
```

Real-life example:

- `/tmp` directory

Interview line:

Sticky bit protects shared directories.

---

9. Why /etc/shadow Is Secure

Explanation

- Stores encrypted passwords
- Readable only by root
- Regular users cannot access it

Permissions:

```
-r-------- 1 root root /etc/shadow
```

Interview answer:

/etc/shadow is protected to prevent password leakage.

---

10. Default Permissions and umask

Recap

- Files default: 666
- Directories default: 777
- umask subtracts permissions

Example:

```
umask 022
```

Result:

- Files: 644
- Directories: 755

Interview line:

umask defines default permissions by subtraction.

---

11. Real-Life Production Scenarios

Scenario 1: Application Not Executing

- Check execute permission
- Check ownership
- Check SELinux (if applicable)

Scenario 2: User Cannot Access Directory

- Missing execute permission
- Incorrect group ownership

Scenario 3: Shared Directory Deletion Issue

- Sticky bit missing

Chapter 4: Interview Takeaways

- Read permission strings confidently
- Explain chmod, chown, chgrp
- Differentiate file vs directory permissions
- Explain SUID, SGID, Sticky bit with examples
- Debug permission-related issues logically