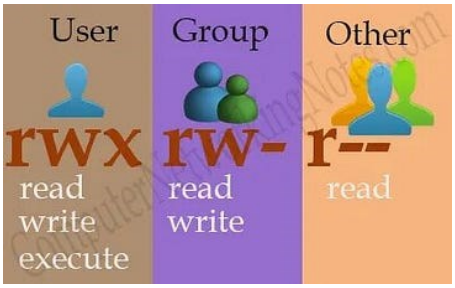
DAY 5x/10 LEARNING ABOUT LINUX

5.x: LINUX FILE and FOLDER PERMISSIONS AND OWNERSHIP



MASTERING LINUX FILE AND FOLDER PERMISSIONS AND OWNERSHIP

The Linux operating system employs a sophisticated model for controlling access to files and directories. This model hinges on three fundamental concepts: permissions, ownership, and special access rights. By assigning permissions and ownership judiciously, Linux ensures that system resources are accessible only to authorized entities, bolstering security and operational efficiency.

A. THE ESSENCE OF LINUX FILE PERMISSIONS

At the heart of Linux's access control model are permissions that are assigned to every file and directory. These permissions determine the actions that users can perform on a file or directory. Each entity is associated with an owner, a group, and delineated access rights for three distinct classes of users:

- ***Owner****: The creator of the file or directory, who has the authority to set permissions.*
- ***Group****: A collective of users classified under a common set of permissions. A file belongs to a single group.*
- ***Others****: All users other than the owner or the group members.*

There are three primary permissions:

- ***Read (r)****: Grants the capability to read a file or list a directory's contents. [r (read) = 4]*
- ***Write (w)****: Allows altering a file's content or modifying the contents of a directory. [w (write) = 2]*
- ***Execute (x)****: Permits executing a file or traversing a directory. [x (execute) = 1]*

Utilizing the **ls -l** command, one can view these permissions displayed as **a string of ten** characters: the first character indicates the file type, followed by three sets of permissions for the owner, group, and others, respectively (expl: for chmod 544 file_name.txt).

```
 -  r – x r - -  r - -    1   clim    clim 0 Apr  7 19:12 file_name.txt
| [- - -][- - -][- - -]  -   [------]   [---]
|   |     |     |     |    |       |
|   |     |     |     |    |       +-----------> 7. Group
|   |     |     |     |    +-------------------> 6. Owner
|   |     |     |     +------------------------> 5. Alternate Access Method
|   |     |     +------------------------------> 4. Others Permissions
|   |     +------------------------------------> 3. Group Permissions
|   +------------------------------------------> 2. Owner Permissions
+--------------------------------------------> 1. File Type
```

Permission	Character	Meaning on File
Read	**-**	*The file is not readable. You cannot view the file contents.*
	r	*The file is readable.*
Write	**-**	*The file cannot be changed or modified.*
	w	*The file can be changed or modified.*
Execute	**-**	*The file cannot be executed.*
	x	*The file can be executed.*
Set UID	**s**	*If found in the user triplet, it sets the setuid bit, allowing execution with the file's owner privileges. Also implies x is set.*
Set GID	**s**	*If found in the group triplet, it sets the setgid bit, allowing execution with the file's group privileges. Also implies x is set.*
	S	*Same as s, but the x flag is not set. This flag is rarely used on files.*
Sticky Bit	**t**	*If found in the others triplet, it sets the sticky bit, which is usually not applicable for files but implies x is set.*
	T	*Same as t, but the x flag is not set. This flag is useless on files.*

Table: how Linux interprets the characters associated with file permissions

B. MODIFYING FILE PERMISSIONS

Linux supports two principal methodologies for adjusting file permissions: the Numeric (Octal) Mode and the Symbolic (Text) Mode.

- **Numeric (Octal) Mode**: This method employs octal numbers to represent permission sets, offering a concise mechanism for modifying permissions. Each permission is associated with a numeric value: read (4), write (2), and execute (1). These values are summed for each class of users to derive a three-digit octal number that represents the desired permissions. For instance, chmod 754 file_name assigns full

permissions (7) to the owner, read and execute permissions (5) to the group, and read-only access (4) to others.

- **Symbolic (Text) Mode**: Leveraging alphabetic symbols, this approach allows dynamic adjustment of permissions using operators (+, -, =) alongside permission characters (r, w, x). For example, chmod g+w file_name adds write permission for the group.

C. SPECIAL PERMISSIONS: SUID, SGID, Sticky Bit

Linux introduces special permissions for advanced access control:

- **Set User ID (SUID)** and **Set Group ID (SGID)**: SUID executes a file with the file owner's permissions, while SGID uses the group's permissions. They are pivotal for files requiring elevated privileges during execution.
- **Sticky Bit**: Applied to directories to restrict file deletion to the file's owner, directory's owner, or root.

D. OWNERSHIP AND GROUP MANAGEMENT

File and directory ownership is pivotal for access control, with commands like chown and chgrp available to alter owner and group associations, facilitating flexible management of access rights.

Default Permissions and the Role of umask

The umask command influences the default permissions assigned to newly created files and directories by subtracting permissions from the system's default permissions.

Access Control Lists (ACLs)

For granular permission management, ACLs allow the definition of access rights for specific users or groups, enhancing the traditional permission model.

Best Practices in Permission Management

- Employ minimum necessary permissions to secure access.
- Regularly audit permissions for sensitive files and directories.
- Utilize ACLs for detailed access control in complex environments.

Practical Commands and Usage

- To view permissions: ls -l file_name shows detailed permissions.
- To change permissions using Octal Mode: chmod 644 file.txt configures read and write permissions for the owner, read-only for the group and others.
- To alter ownership: sudo chown user:group file.txt modifies both the owner and group of file.txt.

Grasping the nuances of file and folder permissions, ownership, and special rights is crucial for Linux users and administrators. These mechanisms are foundational to Linux's security architecture, ensuring that system resources are shielded from unauthorized access while enabling legitimate use cases.

Homework TC 5x

a) Homework 1: Understanding Basic File Permissions

Objective: Familiarize yourself with basic file permissions (read, write, execute) in Linux.

1. **Create a Text File and Modify Permissions**:
    - Create a text file named practice.txt.
    - Use chmod to set its permissions to read-only for the owner, and no permissions for the group and others.
    - Verify the permissions using ls -l.
2. **Toggle Execute Permission**:
    - Add execute permission for the owner.
    - Try executing the file as a script (you may need to add a shebang line #!/bin/bash and a simple command like echo "Hello World").
    - Remove execute permission and try executing it again.

```
#Create a Text File and Modify Permissions:

touch practice.txt

chmod 400 practice.txt

ls -l practice.txt

#Toggle Execute Permission:

## Add execute permission and attempt execution

chmod +x practice.txt

./practice.txt # Assuming the file contains a valid script

## Remove execute permission and attempt execution again

chmod -x practice.txt

./practice.txt # This should fail
```

b) Homework 2: Exploring Special Permissions

Objective: Explore the effects of special permissions (SUID, SGID, Sticky Bit).

1. **Experimenting with SUID and SGID**:
    - Create two executable files, file_suid and file_sgid, containing simple echo commands.
    - Set the SUID bit on file_suid and the SGID bit on file_sgid.
    - Execute both files and observe the effective user and group IDs.

2. **Applying Sticky Bit to a Directory**:
    - Create a directory sticky_dir and set the sticky bit on it.
    - Create multiple files within the directory as different users.
    - Attempt to delete or rename files created by other users.

```
#Experimenting with SUID and SGID:
echo 'echo "Current user: $(whoami)"' > file_suid
echo 'echo "Current group: $(groups)"' > file_sgid
chmod +x file_suid file_sgid
chmod u+s file_suid
chmod g+s file_sgid
./file_suid
./file_sgid
#Applying Sticky Bit to a Directory:
mkdir sticky_dir
chmod +t sticky_dir
# Create files as different users and attempt to delete each other's files
```

c) Homework 3: Numeric Permission Modes

Objective: Practice setting permissions using numeric (octal) modes.

1. **Setting Permissions Using Numeric Mode**:
    - Create a file numeric_mode.txt and a directory numeric_dir.
    - Set the file's permissions to 644 and the directory's permissions to 755 using numeric mode.
    - Verify the permissions and attempt to write to the file and list the directory's contents as a different user.
2. **Special Permissions with Numeric Mode**:
    - Create an executable file special_numeric and a directory special_dir.
    - Apply 4755 permission to special_numeric and 2777 to special_dir.
    - Observe the effect of SUID on the file and SGID plus the sticky bit on the directory.

```
#Setting Permissions Using Numeric Mode:
touch numeric_mode.txt
mkdir numeric_dir
chmod 644 numeric_mode.txt
chmod 755 numeric_dir
```

```
ls -l numeric_mode.txt numeric_dir
#Special Permissions with Numeric Mode:
echo 'echo "Hello World"' > special_numeric
chmod +x special_numeric
mkdir special_dir
chmod 4755 special_numeric
chmod 2777 special_dir
ls -l special_numeric special_dir
```

d) Homework 4: Symbolic Permission Modes

Objective: Utilize symbolic modes to set and modify file and directory permissions.

1. **Basic Symbolic Mode Operations**:
    - Create a file symbolic_file.txt and a directory symbolic_dir.
    - Use symbolic mode to add execute permission for the group on the file and write permission for others on the directory.
    - Verify changes and attempt access accordingly.
2. **Using Symbolic Mode for Special Permissions**:
    - On an existing file and directory, use symbolic mode to set the SUID bit on the file and both SGID and sticky bits on the directory.
    - Validate the permissions and observe their implications on file execution and directory behavior.

```
#Basic Symbolic Mode Operations:
touch symbolic_file.txt
mkdir symbolic_dir
chmod g+x symbolic_file.txt
chmod o+w symbolic_dir
ls -l symbolic_file.txt symbolic_dir
#Using Symbolic Mode for Special Permissions:
## (Assuming existing_file and existing_dir are already created:)
chmod u+s existing_file
chmod g+s,o+t existing_dir
ls -l existing_file existing_dir
```

Submission Guidelines

- *Ensure your scripts are well-commented to explain your logic and approach.*
- *Test your scripts with sample data to verify correctness.*
- *Submit (in the forum) your commands along with a brief report describing your solution strategy for each task. Also export history and upload it as usual.*