#90DaysOfDevOps Challenge - Day 6 - File Permissions and Access Control Lists

Welcome to Day 6 of the #90DaysOfDevOps challenge. Today, we will explore file permissions and Access Control Lists (ACLs). Understanding file permissions is crucial for managing access to files and directories in a Linux system, ensuring security and data integrity.

File Permissions Overview

In Linux, each file and directory has a set of permissions that determine who can read, write, and execute them. Permissions are assigned to three categories of users: owner, group, and others.

Owner

The owner of a file or application is the user who created it. The owner has the highest level of control over the file, including the ability to change permissions, modify contents, and delete the file.

To change the ownership of a file or directory, we use the `chown` command. For example:

```
chown new_owner file.txt
```

Group

The group that owns a file or application is a set of users who share common permissions for that file. Group permissions are useful when multiple users need access to the same files, allowing them to collaborate efficiently.

To change the group ownership of a file or directory, we use the `chgrp` command. For example:

```
chgrp new_group file.txt
```

Others

The "others" category includes all users who have access to the system but are neither the owner nor members of the group. The permissions granted to others define what actions they can perform on the file or directory.

To change the permissions for others, we use the `chmod` command. For example:

```
chmod o+rw file.txt
```

Task 1: Changing User Permissions

As a task for today, let's change the user permissions of a file and observe the changes using the `ls -ltr` command. Here's an example of how you can do it:

1. Create a simple file using the `touch` command:

```
touch file.txt
```

2. Use `ls -ltr` to view the details of the file:

```
ls -ltr file.txt
```

3. Change the user permissions of the file using `chmod`. For example, grant read and write permissions to the user:

```
chmod u+rw file.txt
```

4. Run `ls -ltr` again to see the updated permissions:

```
ls -ltr file.txt
```

By experimenting with different permission settings and observing the changes using `ls -ltr`, you can gain a better understanding of how file permissions work.

Task 2 - Article: Understanding File Permissions

File permissions in Linux play a vital role in maintaining data security and controlling access to files and directories. They follow a set of rules that determine the level of access granted to the owner, group members, and other users. Let's explore the key aspects of file permissions:

1. **Read (r) Permission**: Read permission allows users to view the contents of a file or list the files in a directory. With read permission, users can execute commands like `cat`, `more`, and `ls -l` to read file content and directory listings.

2. **Write (w) Permission**: Write permission grants users the ability to modify the contents of a file or create, rename, and delete files within a directory. Users with write permission can use commands like `vi`, `echo`, and `rm` to make changes to files and directories.

3. **Execute (x) Permission**: Execute permission is necessary for users to run executable files or access directories. It allows users to execute a program file or traverse through a directory using commands like `./program` or `cd directory`.

The combination of these permissions determines the level of access granted to different user categories. Each permission can be either enabled (+) or disabled (-) for each user category. For example, `rwx`

represents read, write, and execute permissions, while `---` indicates no permissions.

To view the permissions of a file or directory, you can use the `ls -l` command. It displays the permission settings along with other details such as ownership, group, size, and modification timestamp.

It's important to note that the order of the permission settings is owner, group, and others. For example, `-rw-r--r--` indicates that the owner has read and write permissions, while the group and others have only read permissions.

Understanding file permissions enable you to control access to sensitive data, ensuring that only authorized users can view, modify, or execute files and directories. It's essential to strike a balance between granting necessary permissions and maintaining data security.

Task 3 - Access Control Lists (ACL). Commands getfacl and setfacl

Access Control Lists (ACLs) provide a more fine-grained level of control over file permissions. While traditional file permissions only grant access to the owner, group, and others, ACL allows you to define permissions for specific users or groups individually.

Two useful commands for working with ACL are `getfacl` and `setfacl`. The `getfacl` command displays the ACL settings for a file or directory, while the `setfacl` command allows you to modify the ACL entries.

To view the ACL settings of a file, use the following command:

```
getfacl file.txt
```

```
esteban@ubuntu-server-template:/etc/opt$ getfacl file.txt
# file: file.txt
# owner: root
# group: root
user::rw-
group::r--
other::r--
```

To modify the ACL entries and grant specific permissions to users or groups, use the `setfacl` command. For example, to grant read and write permissions to a user:

```
setfacl -m u:user:rw file.txt
```

```
esteban@ubuntu-server-template:/etc/opt$ getfacl file.txt
# file: file.txt
# owner: root
# group: root
user::rw-
user:esteban:rw-
group::r--
mask::rw-
other::r--
```

ACL provides a flexible approach to managing file permissions, especially in complex scenarios where you need to grant specific access to individual users or groups.

Congratulations on completing Day 6 of the #90DaysOfDevOps challenge! Today, we explored file permissions, learned about the owner, group, and other categories, and understood how to change user permissions using commands like `chown`, `chgrp`, and `chmod`. Additionally, we dived into Access Control Lists (ACLs) and their role in providing more granular control over file permissions.

In tomorrow's challenge, we will explore package management and how to install, update, and remove software packages in Linux. Package management is a crucial aspect of DevOps, making it essential to establish a solid foundation in this topic.