COURSEWARE

- Professional Skills
- Agile Fundamentals
- Jira
- Git
- Databases Introduction
- Java Beginner
- Maven
- Testing (Foundation)
- Java Intermediate
- HTML
- CSS
- Javascript
- Spring Boot
- Selenium
- Sonarqube
- Advanced Testing (Theory)
- Cucumber
- MongoDB
- Express
- NodeJS
- React
- Express-Testing
- Networking
- Security
- Cloud Fundamentals
- AWS Foundations
- AWS Intermediate
- Linux
  - Linux Introduction

Ownership

Contents

- Overview
- Access Attributes
- File Ownership
- File Permissions
  - The Super User
- File Protection
- What permissions are needed?
- Setting Permissions
- Setting file ownership
- Tutorial
- Exercises

Overview

In this module we will go over ownership and permissions of files and how to change them.

Access Attributes

Linux is a `multi-user system`, meaning it can be accessed by many users.
This can raise security concerns as someone could change or delete `crucial data`.
To prevent these kinds of actions we make sure our `access control` is tight.

There are two levels in which we control access for a user, ownership and permissions.

File Ownership

Every single file and directory has an owner.
The file owner can specify who can read, write and execute the file.
Every file will have a file owner and a group owner.
By default the file owner is the creator of the file but this can be changed.
When groups own a file anyone in that group will have the same permissions to the file.
Any other users who are not the file owner or part the group that owns the file, have their own permissions.

- Linux Distributions
- Bash Interpreter
- Sessions in Linux
- Lists and Directories
- Editing Text
- Aliases, Functions and Variables
- User Administration
- Ownership
- Data Streams
- Pipes and Filters
- Scripting in Linux
- Sudoers
- Managing systemd Services
- Systemd Service Configuration
- OpenSSH
- Screens in Linux

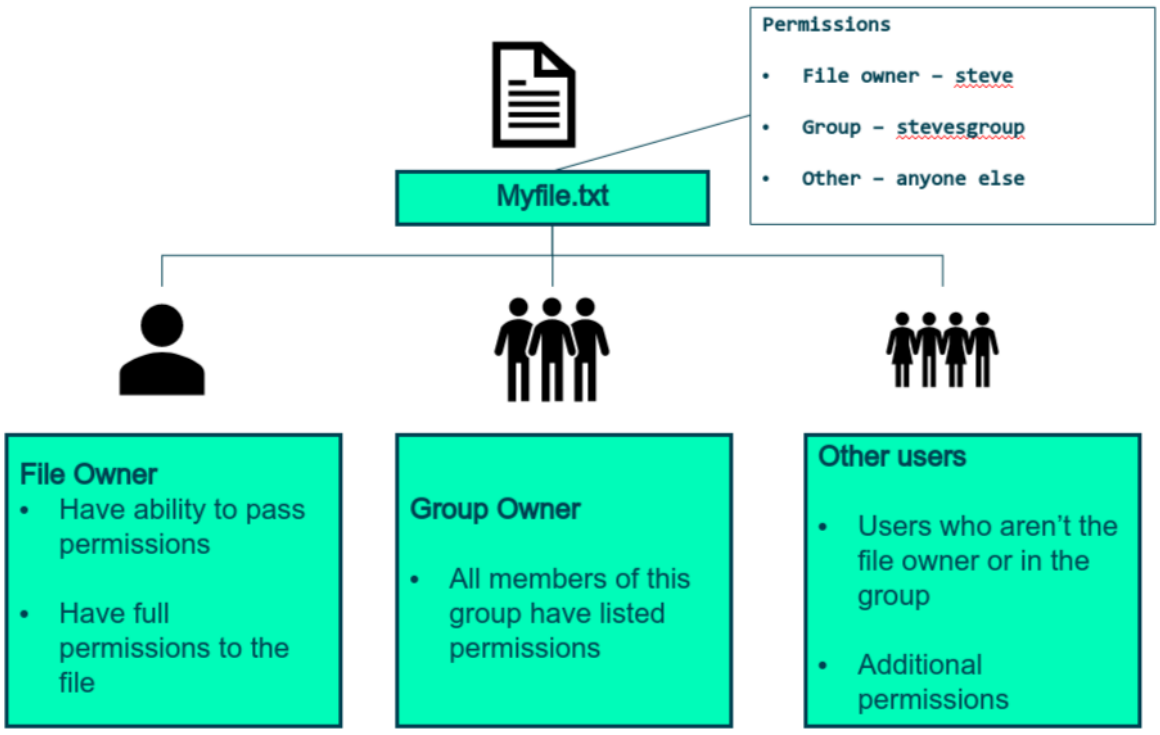
DevOps

Jenkins Introduction

Jenkins Pipeline

Markdown

IDE Cheatsheet



Execute the command `ls -l` in the bash terminal. You will get an output like this.

```
$ ls -l
-rwxrw-rw- 1 ben ben 0 Dec 22 09:25 file1
-rwxrw-r-- 1 ben ben 0 Dec 23 09:07 file2
```

This command lists all the files in your current directory while showing some extra information.

- `-rwxrw-rw-` - File permissions
- `1` - Number of links to the file
- `ben ben` - File and group owners
- `0` - File size (in bytes)
- `Dec 22 09:25` - Date and time last modified
- `file1` - Name of file

File Permissions

So we know who has access, but what access is given to them and how?

There are `three` types of permissions assigned, read, write and execute. All these permissions are given or not given to the user owner, group and other users.

`-rwxrw-r--` is an example of a files permissions. There are always 10 slots.
The first slot describes the type of file. So `drwxrw-r--` permissions would tell us the file is a directory.
Then the 2nd to the 10th slot are, at most, `rwx` three times, one for each of the different sets of users, the file owner, the group and other users.
Therefore `-rwxrw-r--` tells us that the standard file has an owner that can read, write and execute the file, a group that can read and write into the file and other users can only read the file.

The Super User

One thing we need to consider is the super user, who has unrestricted access to the file system. As an administrator, they need to be able to access any files or directories. Due to this, administrator should be a title held by very few people.

So when someone tries to access a file, the authorisation checks are as follows.

- Are they a super user?
- Are they the file owner? Do they have the appropriate permissions?
- Are they part of the group owner? Do they have the appropriate permissions?
- Do other users have access to this file?

File Protection

So what does read, write and execute actually mean?

- `r` - To be able to open the file for reading
- `w` - To be able to open the file and edit it
- `x` - To be able to execute the file (Only really applies if the file is a program or a shell script.)

For directories they have slightly different meanings.

- `r` - To be able to read the directories list (Does not imply access to files)
- `w` - To be able to write to directory (Create, rename, delete)
- `x` - Search through directory (pass through, access files)

What permissions are needed?

To create a file you need:

- `Execute` permissions on all directories in the `pathname`
- `Write/execute` permissions on the last driectory in the `pathname`.

To read a file, you need:

- `Execute` permissions on all directories in the `pathname`.
- `Read` permissions on the file.

To write into a file, you need:

- `Execute` permissions on all directories in the `pathname`.
- `Write` permissions on the file.

Setting Permissions

We can change permissions with the `chmod` command.

```
$ chmod [options] [permissions] [file]
```

Permissions can be expressed symbolically or with numbers:

```
$ chmod ugo=rw file1
$ chmod 666 file1
```

When using symbolic permissions the user tpyes are:

- `u` - File owner
- `g` - File group owner
- `o` - Other users
- `a` - All user types

And the different access operations are:

- `=` - Set permissions
- `+` - Add permissions
- `-` - Remove permissions

```
$ chmod a+x file1
```

This command would give all user types permission to execute the file.

```
$ chmod o-wx file1
```

This command removes the permission to write and execute the file for users that are not the owner or part of the group owner.

When setting permissions with numbers we use:

- `r` = 4

- `w` = 2
- `x` = 1

And the octal value of a user type's permission is the sum of the actions they can take. For example, if a user type can read, write and execute, their octal vaule is `4 + 2 + 1 = 7`.

When using the `chmod` command, you give 3 numbers, one for each user type and each number is that user type's octal number.

```
$ chmod 666 file1
```

This command would give read and write permissions to all users.

Setting file ownership

You can change the owner and group owner of a file with the `chown` and `chgrp` respectivley.

```
$ chown [-R] [user] [file]
$ chgrp [-R] [group] [file]
```

You can change the owner and the group using the `chown` command.

```
$ chown [-R] [user:group] [file]
```

```
$ chown ben:docker file1
```

This command would change the file owner to ben and the file group owner to the docker group.

Tutorial

There is no tutorial for this module.

Exercises

Test adding and changing permissions on files and directories.

- Create a file, file1 and write commands to do the following.
- Add read, write and execute to file1 for the file owner
- Add read permissions to file1 for all users.
- Add read and write to file1 for the group owner
- Take write permissions from the file owner and group owner of file1
- Add read, write and execute to file1 for all users.

▶ Answers