You already know what files are, how to create them, and what files Unix has. Now let's find out how we can access them. Below we will discuss what authorization levels there are in Unix, who may have access to the files, and what limits users' actions in regard to the files.

Authorization levels

Unix systems like Linux can be accessed by many users simultaneously. They can also be used in mainframes and servers without any modifications. But this raises security concerns as arbitrary or even malicious users can corrupt, change or remove crucial data. For effective security, Unix divides authorization into 2 levels:

- Ownership
- Permission

Let's study each of these two levels starting with the Ownership.

File ownership

File ownership, in other words, signifies users that can work with a file. Every file and directory in Unix is assigned 3 types of ownership described below:

- A **user** is the owner of the file. By default, the person who created a file becomes its owner. So, the user can sometimes be called an owner.
- A **group** contains several users. All of them have the same access permissions to the file. The group is useful when you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all the users to a group, and assign group permission to a file so that only the members of this group and no one else can read or modify the file.

- Any **other** user who has access to a file is the one who did not create this file and also does not belong to a group. Practically, it means everybody else. Hence, when you set the permissions for others, it is also referred to as set permissions for the world.

Now, the big question is how does Unix distinguish between these three user types, so that user *A* cannot affect a file which contains user *B's* vital data. This is where permissions come in, defining user behavior.

File permissions

File permissions reflect what users can do with a file. Every file and directory in Unix has the following 3 permissions defined for all the 3 types of users discussed above:

- **Read permission** gives users the authority to open and read a file. Read permission on a directory provides users with the list of its content.
- **Write permission** gives users the authority to modify the contents of a file. The write permission on a directory gives users the authority to add, remove, and rename files stored in the directory. Imagine you have a write permission on a file but do not have a write permission on the directory where the file is stored. You will be able to modify only the contents of the said file. However, you will not be able to rename, move, or remove the file from the directory.
- **Execute permission** lets users run a program. If the execute permission is not set, they might still be able to see/modify the program code (provided read and write permissions are set), but not run it.
- **No permission** means users can't do anything with a file.

It's possible to combine all these permission types, and there are eight possible variants of such combinations. They are shown in the table below:

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x

6	Read + Write	rw-
7	Read + Write + Execute	rwx

In the *Symbol* column, permission combinations are abbreviated. So,

`-`

means *no permission*,

`w`

means *write*,

`r`

means *read*, and

`x`

stands for *execute*. Also you can define a permission set using its corresponding three-digit octal number. Thus, 7 is the same as

`rwx`

and 2 is the same as

`-w-`

and so on. Let's take a look at some examples and common cases below.

Common cases

The whole permission set presented as symbols can look like this:

`-rwxrw-r--`

, where the first

`-`

implies that we have selected a certain file. If it were a directory,

`d`

would have been used instead of the first

`-`

. After it, there are 3 triplets of symbols. Each of them defines permissions for owner, group, and other users correspondingly. So, in the example, the owner can read, write and execute the file. The group can read and write, while the other can only read the file.

The most common sets presented as octal numbers are as follows:

- **755** is commonly used by web servers. The owner has all the permissions to read, write and execute it. Everyone else can read and execute but cannot make changes to the file.
- **644** means only the owner can read and write. Everyone else can only read. No one can execute this file.
- **655** means only the owner can read and write and cannot execute the file. Everyone else can read and execute and cannot modify the file.
- **777** means every user can read, write, and execute. Because it grants full permission, it should be used with caution. However, in some cases, you will need to set the 777 permissions before you can upload any file to the server.

The directory permissions are similar but differ from the file ones in some ways. Let's figure this difference out.

File permissions vs directory permissions

It is important to remember that the right to read, write and execute in a directory does not imply the same rights to all files from that directory:

	File	Directory
Read	One can open a file and see its content	One can view information about files with `ls -l` command if execute permission is also set, and only name of files if execute permission is not set
Write	One can change the contents of the file	One can rename, delete and create files if

		execute permission is also set
Execute	One can run a file	One can enter directory with `cd` command and have access to files

Conclusion

To sum up,

- there are two authorization levels in Unix: ownership and permission;
- there are three ownership types in Unix: user, group, and other;
- permission set may include such actions as read, write and execute;
- there are two ways to represent a permission set: as symbols or as octal numbers.

Ajabu directory

Report a typo

Let's practice with a directory that has some strange permissions. Let's look at the directory *./ajabu* with permissions

`d--x--x--x`

. What can you do in this directory? Choose the correct statements.

You can practice with this directory on your machine, create it and look at permissions with these commands.

```
mkdir ajabu
touch ajabu/idanwo1 ajabu/idanwo2
chmod 111 ajabu

ls -ld ajabu
```

You can enter the directory using cd ajabu command