Experiment No # 04

Experiment Name # File Operation and Permission

Aim and Objects:

On Linux and other Unix-like operating systems, there is a set of rules for each file which defines who can access that file, and how they can access it. These rules are called file permissions or file *modes*. The command name **chmod** stands for "change mode", and it is used to define the way a file can be accessed. For effective security, Linux divides authorization into 2 levels.

i. Ownership
ii. Permission

i) Ownership of Linux files :

Every file and directory on Unix/Linux system is assigned 3 types of owner, given below :

1) User :

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

2) Group

A user- group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose if I have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, I could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

3) Other

Any other user who has access to a file. This person has neither created the file, nor he belongs to a user group who could own the file. Practically, it means everybody else. Hence, when permission set for others, it is also referred as set permissions for the world.

ii) Permissions :

Every file and directory in UNIX/Linux system has following three permissions defined for all the three owners discussed above.

1) **Read:** This permission is given to the authority to open and read a file. Read permission on a directory gives the ability to lists its content.

2) **Write:** The right permission is given the authority to modify the contents of a file. The write permission on a directory is given the authority to add, remove and rename files stored in the directory. If write permission is given on file but do not have write permission on the directory

where the file is stored. One will be able to modify the file contents. But he/she will not be able to rename, move or remove the file from the directory.

3) **Execute:** In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, a program cannot run unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.

File names and permission characters:

File names can be up to 256 characters long with "-", "_", and "." characters along with letters and numbers. When a long file listing is done, there are 10 characters that are shown on the left that indicate type and permissions of the file. File permissions are shown according to the following syntax example: drwerwerwe. There are a total of 10 characters in this example, as in all Linux files. The first character indicates the type of file, and the next three indicate read, write, and execute permission for each of the three user types, user, group and other. Since there are three types of permission for three users, there are a total of nine permission bits. The table below shows the syntax:

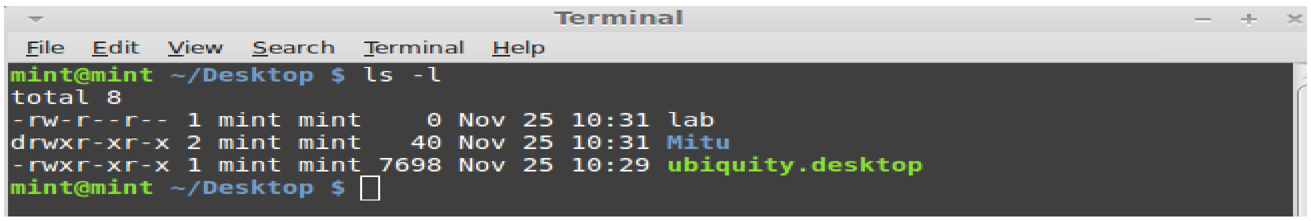
1	2	3	4	5	6	7	8	9	10
File	User Permissions			Group Permissions			Other Permissions		
Type	Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
d	r	w	e	r	w	e	r	w	e

There are 4 possible characters in the permission fields. They are:

- r = read - This is only found in the read field.
- w = write - This is only found in the write field.
- x = execute - This is only found in the execute field.
- If there is a "-" in a particular location, there is no permission. This may be found in any field whether read, write, or execute field.

Examples:

Access File Permission : "ls -l" is used to show the permission .



```
mint@mint ~/Desktop $ ls -l
total 8
-rw-r--r-- 1 mint mint    0 Nov 25 10:31 lab
drwxr-xr-x 2 mint mint   40 Nov 25 10:31 Mitu
-rwxr-xr-x 1 mint mint 7698 Nov 25 10:29 ubiquity.desktop
mint@mint ~/Desktop $ 
```

Here,

The first character in the field indicates a file type of one of the following:

- d = directory
- l = symbolic link
- s = socket
- p = named pipe
- - = regular file

Changing file/directory permissions :

We can use the '**chmod'** command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world. **Syntax:**

“chmod permissions filename”

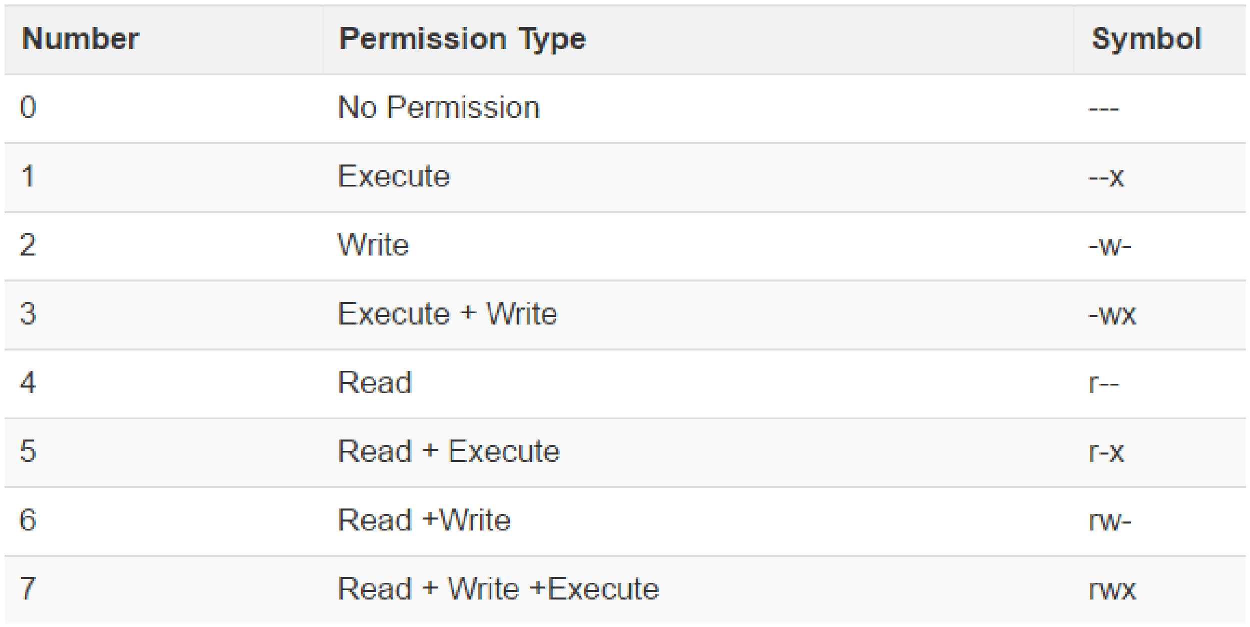
There are 2 ways to use the command -

1. Absolute mode
2. **Symbolic mode**

1) Absolute Mode :

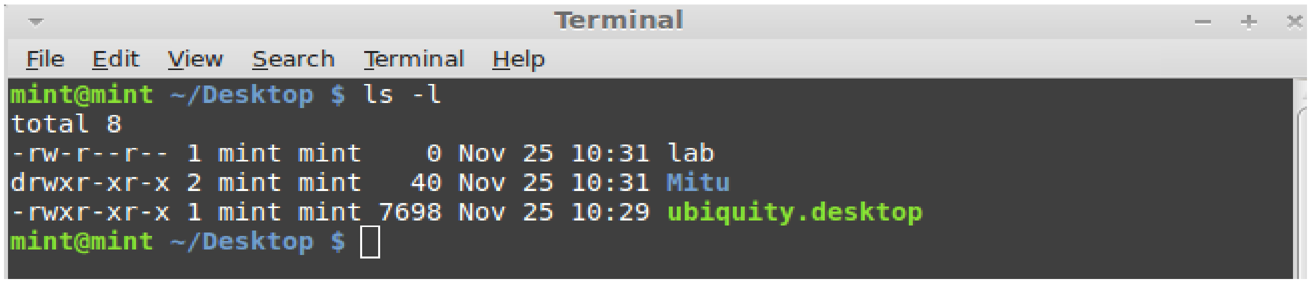
The file permission is represented by numerical value from 0 to 7. The list is given below :

Example :



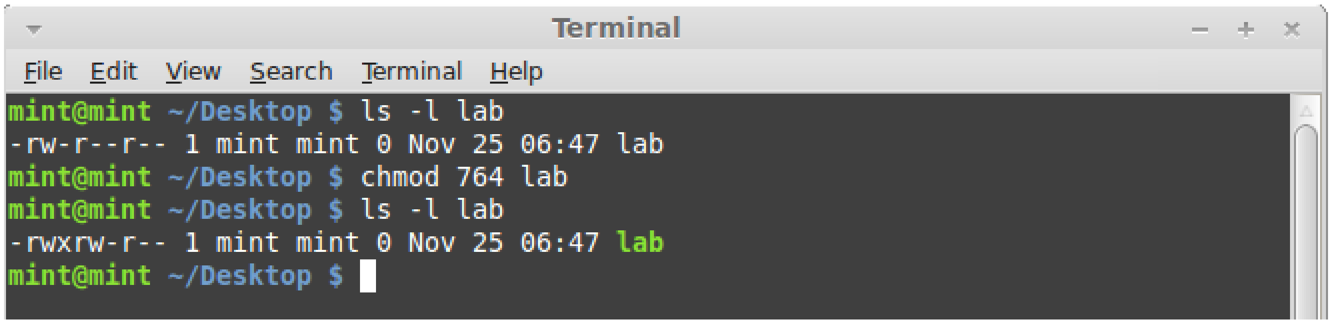
Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rwx

Checking Current File Permission :



```
mint@mint ~/Desktop $ ls -l
total 8
-rw-r--r-- 1 mint mint    0 Nov 25 10:31 lab
drwxr-xr-x 2 mint mint   40 Nov 25 10:31 Mitu
-rwxr-xr-x 1 mint mint 7698 Nov 25 10:29 ubiquity.desktop
mint@mint ~/Desktop $ 
```

Change permission :



```
mint@mint ~/Desktop $ ls -l lab
-rw-r--r-- 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $ chmod 764 lab
mint@mint ~/Desktop $ ls -l lab
-rwxrw-r-- 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $ 
```

2) Symbolic Mode :

The file permission is represented by mathematical symbols.

Operator	Description
+	Adds a permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

The various owners are represented as -

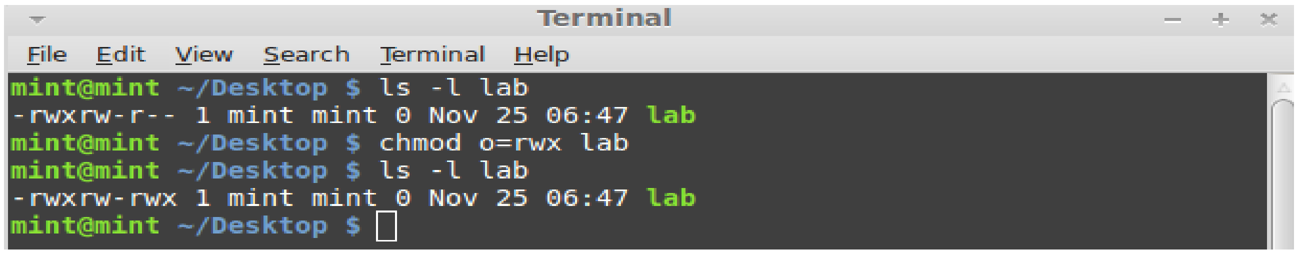
u = user /owner

g = group

o = other

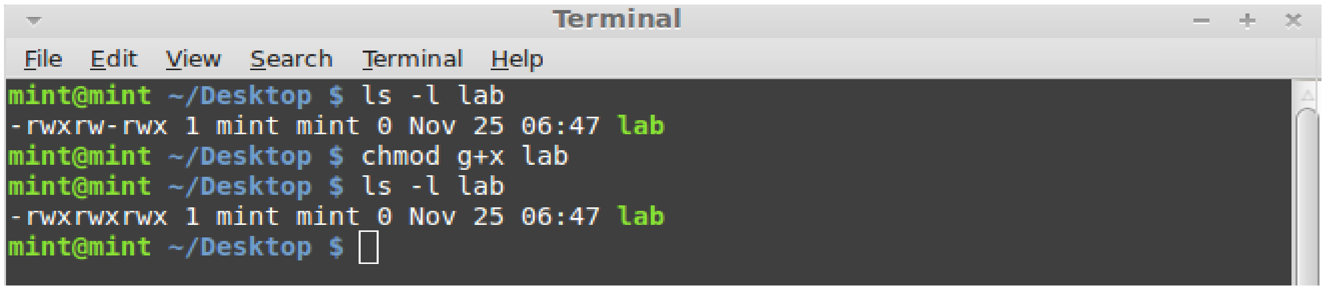
a = all

Giving Permission to the other user :



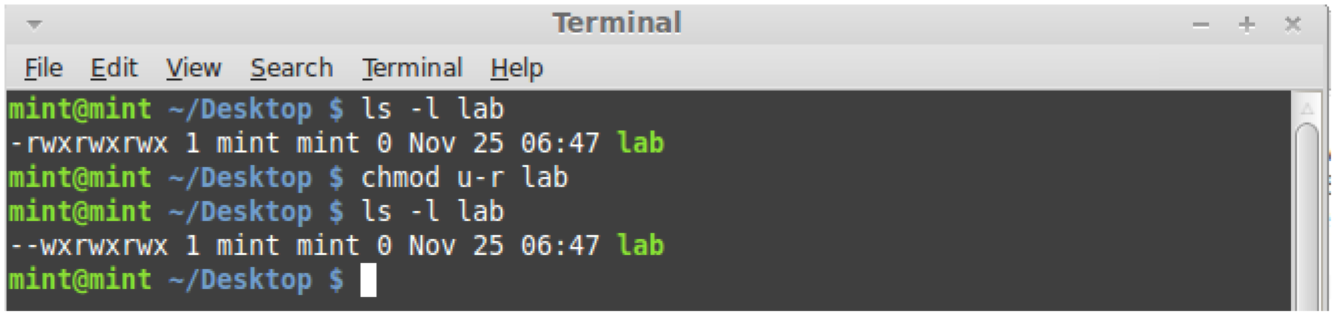
```
mint@mint ~/Desktop $ ls -l lab
-rwxrw-r-- 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $ chmod o=rwx lab
mint@mint ~/Desktop $ ls -l lab
-rwxrw-rwx 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $ 
```

Adding Execution Permission to the group :



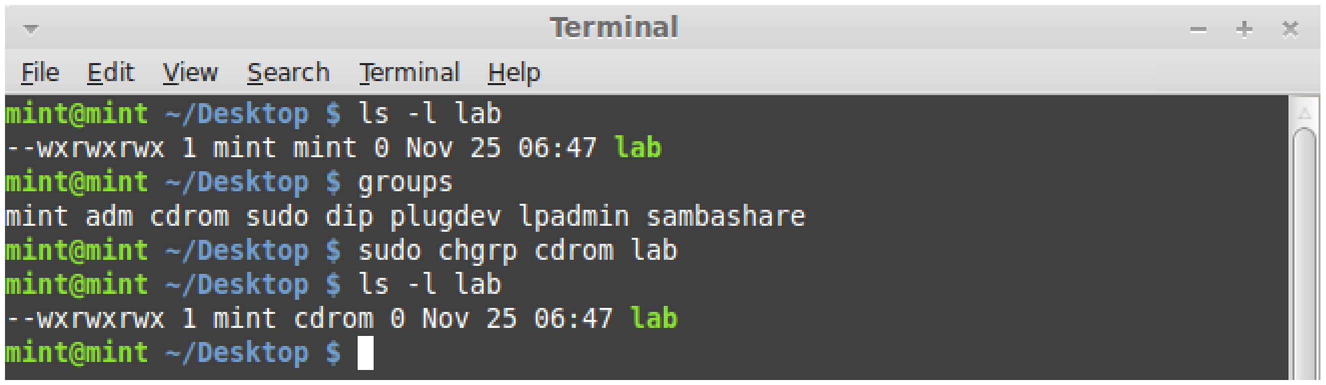
```
mint@mint ~/Desktop $ ls -l lab
-rwxrw-rwx 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $ chmod g+x lab
mint@mint ~/Desktop $ ls -l lab
-rwxrwxrwx 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $
```

Removing Read permission from the user :



```
mint@mint ~/Desktop $ ls -l lab
-rwxrwxrwx 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $ chmod u-r lab
mint@mint ~/Desktop $ ls -l lab
--wxrwxrwx 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $
```

Changing Group :



```
mint@mint ~/Desktop $ ls -l lab
--wxrwxrwx 1 mint mint 0 Nov 25 06:47 lab
mint@mint ~/Desktop $ groups
mint adm cdrom sudo dip plugdev lpadmin sambashare
mint@mint ~/Desktop $ sudo chgrp cdrom lab
mint@mint ~/Desktop $ ls -l lab
--wxrwxrwx 1 mint cdrom 0 Nov 25 06:47 lab
mint@mint ~/Desktop $
```

<u>Conclusion :</u>

Linux being a multi-user system uses permissions and ownership for security. There are three user types on a Linux system User, Group and Other. They can change their owner and group and also can change their permission types.