- 3. 3. Users, Groups and File Permissions -

- INDEX -

- 3. 3. 1. Users -

Linux is a multiuser operating system.

In a multiuser environment, it is a common administration task to create new users, modify existing users, or remove users.

For ease of access management, users are assigned to groups.

Creating, deleting, and modifying groups is also another common administration task.

Users of the system may be human users (people who log into the system) or they can be system users (used to start non-interactive background services such as databases).

From the perspective of the operating system, there is no distinction between human users and system users and all the information is stored in the same file.

The information about users is stored in **/etc/passwd** file.

To view the first line of the file, execute the following command:

```
me@home:~$ head -1 /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
```

The first **root** is the username.

The character x is used as a placeholder for password.

0 is the user ID for this user.

0 is the group ID for this user.

The next **root** is a comment about this user.

/root is the home directory for this user.

/bin/bash is the shell for this user.

Adding Users

You can add new users with the **useradd** command.

The very minimum needed by the **useradd** command is a username.

However, you'll generally need more than just a username.

For example, you can execute the following commands to create a user named john:

```
me@home:~$ useradd -c "John from Accounts" -m -s /bin/bash john
```

The -c flag adds a comment in the /etc/passwd file for this account.

-m automatically creates the home directory for this user under /home with the same name as the username i.e. /home/john

-s assigns the shell for the user.

To see the entry created for this new user, execute the following command:

```
me@home:~$ grep john /etc/passwd
```

```
john:x:1001:1001:John from Accounts:/home/john:/bin/bash
```

The user john has been assigned the user ID 1001 and the group ID 1001.

You can manually specify the user ID using the -u flag followed by the user ID.

When this is not specified, the first available user ID is chosen.

If the group name is not assigned using the -g flag, a group is created with the same name and ID as the user and is made the primary group of the user.

Note that the user has not been assigned a password yet.

The following summarizes some of the flags that can be used with **useradd**:

-c Adds a comment. For example: -c "John from Accounts"

-d Specifies home directory for the user. Use this if the name of the home directory is different from the username. For example: -d /home/accounts/john

-e Specifies the expiration date for the account in YYYY-MM-DD format. For example: -e 2030-01-01

-g Specifies the primary group of the user. The group must already exist in the /etc/group file. For example: -g accounts

-G Specifies the additional groups to which the user belongs. For example: -G employees

-m Automatically creates the home directory for this user under /home with the same name as the username.

-p Specifies the password to be associated with this account. This must be an encrypted password. You can assign the password later using passwd command. For example: -p hashed_password

-s Specifies the shell to be associated with this account. For example: -s /bin/bash

-u Specifies the user ID to be used with this account. Without -u flag, the first available user ID will be assigned. For example: -u 1005

Setting a Password

The newly made account has no password.

The **passwd** command is used to add a password to the account.

For example, you can execute the following command:

```
me@home:~$ sudo passwd john

[sudo] password for me:

Enter new UNIX password:

Retype new Unix Password:

passwd: password updated successfully
```

The **/etc/shadow** file contains password and account expiration information for users.

Modifying Users

The **usermod** command is used to modify any existing user account.

Suppose we want to change the username of john to johnny, here’s how we’d do it:

```
me@home:~$ sudo usermod -l johnny john

[sudo] password for me:

me@home:~$ grep john /etc/passwd

johnny:x:1001:1001:John from Accounts:/home/john:/bin/bash
```

The **-l** flag changes the username of the user.

Notice that the username has changed but the home directory is still the same.

Let's change the home directory.

Execute the following command:

```
me@home:~$ sudo usermod -m -d /home/johnny johnny

[sudo] password for me:

me@home:~$
```

If you were to now look at the entry in /etc/passwd, you'd notice that the home directory has been changed.

The **-d** flag specifies the new home directory and the **-m** flag copies the contents over from the old home directory to the new one.

You can even lock and unlock the account by using the **-L** and **-U** flags respectively.

Execute the following command to lock the account:

```
me@home:~$ sudo usermod -L johnny

[sudo] password for me:

me@home:~$ sudo grep johnny /etc/shadow

johnny:!$6$mC3IOEDs$TMWBP2IJfxgDHKjW6cxFk80BY9aqFThvN8MfED/P

JnVqI.mB7Ddtqn35VM5Q4Rm4l8bNIsOd3PXhRktJPwMlc0:16479:17104:0:99999:7:::
```

Because the user account has been locked, there is an exclamation mark before the hash of the password.

To unlock the account, execute the following:

```
me@home:~$ sudo usermod -U johnny

[sudo] password for me:

me@home:~$
```

Deleting Users

The **userdel** command is used to delete users.

Here's how you'd delete the user johnny.

```
me@home:~$ sudo userdel --remove johnny

[sudo] password for me:

me@home:~$
```

This not only deletes the user but also removes all the files that belong to the user.

- 3. 3. 2. Groups -

Groups are a collection of users.

Assigning users to groups makes it easier to manage permissions.

For example, you can set permissions to ensure that files are accessible to people in a particular group like accounts, hr, etc.

Whenever a user is created, by default, they are added to a new group with the same name as the username.

This is called the primary group of the user.

A user john would be added to a group named john.

There are two types of groups in Linux:

- Primary group: is the main group that is associated with user account. Each user is a member of exactly one primary group.
- Secondary group: used to provide additional rights to user. For example, access to the DVD/CD drive can be granted with the help of the "cdrom" group.

/etc/group File

The information about groups is stored in the **/etc/group** file.

```
me@home:~$ grep $(whoami) /etc/group

adm:x:4:syslog,me

cdrom:x:24:me

...
```

The first part is the name of the group.

x is a placeholder for password.

The next part is the group ID.

The last part is a comma-separated list of usernames that belong to that group, but note that a user does not appear on the list of his/her primary group.

You can change the primary group of the user using the usermod command with the **-g** flag.

Suppose John has moved from accounts to HR.

To update the primary group, execute the following:

```
me@home:~$ sudo usermod -g hr johnny

[sudo] password for me:

me@home:~$
```

Similarly, you can add the user to more secondary groups using the **-a -G** flag.

To add the user johnny to manager group, execute the following:

```
me@home:~$ sudo usermod -a -G manager johnny

[sudo] password for me:

me@home:~$
```

If you want to replace the secondary groups the user was a part of, instead of adding new groups, use the **-G** flag: be careful using this option with a member of the "sudo" group, because you will delete that user from the "sudo" group and he/she will not be able to execute "sudo" anymore.

You can change the user ID of the user using the -u flag.

Execute the following command to change the user ID of the user johnny.

```
me@home:~$ sudo usermod -u 3000 johnny

[sudo] password for me:

me@home:~$
```

Some of the flags that can be used with the **usermod** command are the following:

-c Changes the username associated with the account. For example: -c johnny

-d Changes the home directory associated with the account. For example: -c /home/johnny

-e Changes the expiration date associated with the account. Must be written in YYYY-MM-DD format. For example: -e 2035-01-01

-g Changes the primary group for the account. For example: -g employee

-G Changes the additional groups the user is part of. If you want to add the user to more groups, use the **-a -G** flag. Using -G will replace the existing list of groups.

-m Specifies that the contents of the old home directory should be copied over to the new one. Can only used with -d flag.

Adding Groups

The **groupadd** command is used to create a new group.

To create a group, execute the following command:

```
me@home:~$ sudo groupadd manager

[sudo] password for me:

me@home:~$
```

This creates a new group named manager and assigns a group ID to it.

With the -g flag, you can manually assign a group ID to it.

Once a group has been created, you can assign it to a user using the **usermod** command.

If you want to add the user "johnny" to the groups "group1", "group2" and "group3", you can execute the following command:

```
me@home:~$ sudo usermod -a -G group1,group2,group3 johnny

[sudo] password for me:

me@home:~$
```

Modifying Groups

The **groupmod** command is used to modify an existing group.

Here's how you'd modify the ID of the group:

```
me@home:~$ sudo groupmod -g 300 manager

[sudo] password for me:

me@home:~$
```

You can change the name of the group as follows:

```
me@home:~$ sudo groupmod -n managers manager

[sudo] password for me:

me@home:~$
```

Deleting Groups

The **groupdel** command is used to delete a group.

Here’s how you’d delete the managers group:

```
me@home:~$ sudo groupdel managers

[sudo] password for me:

me@home:~$
```

Managing Groups with gpasswd

The **gpasswd** command can be used to add users to a group, remove them, and set admins for the group.

To add a user to a group, execute the following command:

```
me@home:~$ sudo gpasswd -a john manager

[sudo] password for me:

me@home:~$
```

To add multiple users to a group, execute the following command:

```
me@home:~$ sudo gpasswd -M john,jane manager

[sudo] password for me:

me@home:~$
```

Keep in mind that this solution overwrites the existing group members list.

To remove a user from a group, execute the following command:

```
me@home:~$ sudo gpasswd -d john manager

[sudo] password for me:

me@home:~$
```

To make the user an admin to the group, execute the following command:

```
me@home:~$ sudo gpasswd -A jane manager

[sudo] password for me:

me@home:~$
```

Listing all the Groups of a User

The **groups** command lists all the groups of a particular user.

For example, to list all the groups of the user john, execute the following command:

```
me@home:~$ groups john
```

Listing all the Users of a Group

You can install it using the "members" command:

```
me@home:~$ sudo apt install members
```

Once you have the "members" command installed, you can list all the members of the *group_name* group doing the following:

```
me@home:~$ members group_name
```

"Root" User Account in Ubuntu

In Ubuntu Linux, the **root** user account is disabled by default for security reasons.

Ubuntu users are encouraged to perform system administrative tasks by granting administrative privileges to a regular user using a tool named **sudo**.

Sudo allows authorized users to run programs as another user, usually the root user.

By default on Ubuntu systems, members of the group "sudo" are granted with sudo access.

The initial user created by the Ubuntu installer is already a member of the sudo group.

Chances are that the user you are logged in as is already granted with administrative privileges.

If you want to grant sudo access to another user (for example, "kate"), simply add the user to the sudo group:

```
me@home:~$ sudo usermod -a -G sudo kate
```

To temporarily elevate root user privileges using a particular command, run the command prefixed with sudo:

```
me@home:~$ sudo command
```

The first time you use sudo in a session, you will be prompted to enter the user password.

If for some reason, you need to enable the root account, you just need to set a password for the root user.

To enable root account in Ubuntu, run the following command:

```
me@home:~$ sudo passwd root
```

You will be prompted to enter and confirm the new root password:

```
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Note that the password is not shown on the screen when you type it.

You have successfully enabled the root account.

You can now log in to your Ubuntu machine as user root using the new password, with the following command:

```
me@home:~$ su
```

su is the command that can be executed by a user who is already logged into the system, in order to change to the root user.

To finish using the root user and coming back to your regular user, use the "exit" command:

```
root@home:~$ exit
```

If you previously enabled the root user in Ubuntu and now you want to disable it, set the root password to expire.

To disable the root account password, use the following command:

```
me@home:~$ sudo passwd -l root
```

To enable the root user account in Ubuntu, all you need to do is to set the root password.

When setting the password, make sure you're using a strong and unique password.

Having a strong password is the most important aspect of the security of your account.

Often a strong password has at least 16 characters, at least one uppercase letter, one lowercase letter, one number, and one special character.

- 3. 3. 3. File Permissions -

In Linux, each and every file is owned by a single user and a single group, and has its own access permissions.

Let's look at how to view the ownership and permissions of a file.

The most common way to view the permissions of a file is to use **ls** with the long listing option:

```
ls -l my file
```

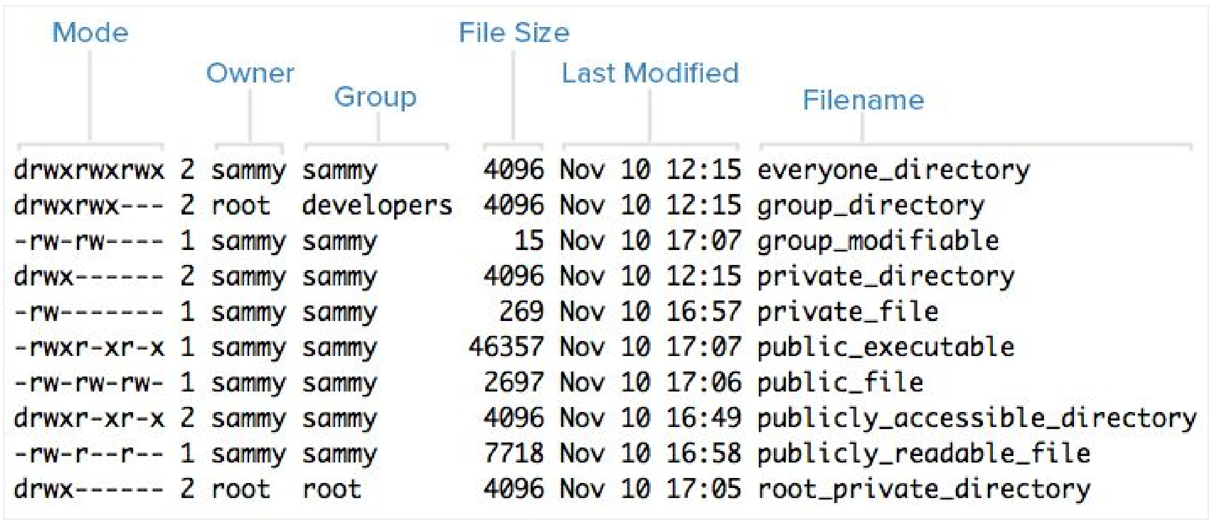
If you want to view the permissions of all of the files in your current directory, run the command without an argument, like this:

```
ls -l
```

If you are in an empty home directory, and you haven't created any files to view yet, you can follow along by listing the contents of the **/etc** directory by running this command:

```
ls -l /etc
```

Here is an example screenshot of what the output might look like, with labels of each column of output:



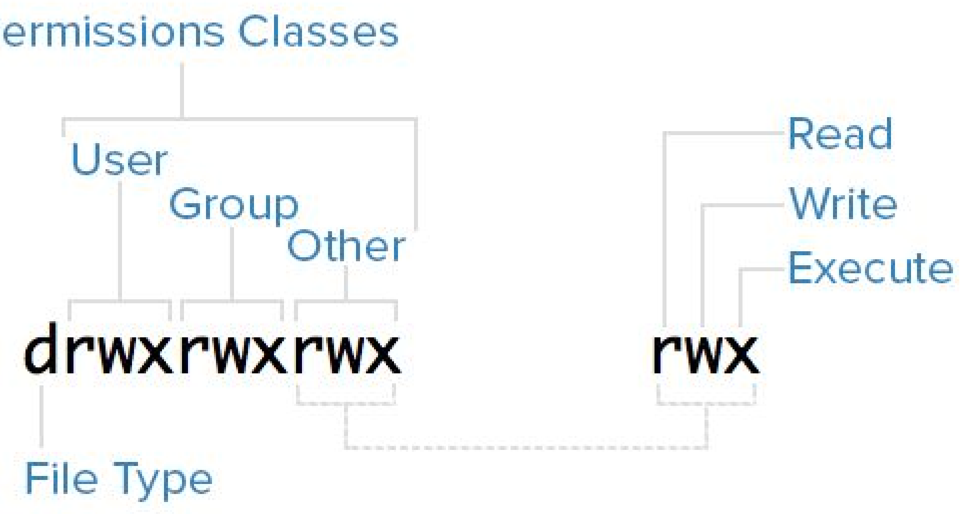
```
drwxrwxrwx 2 sammy sammy       4096 Nov 10 12:15 everyone_directory
drwxrwx--- 2 root  developers  4096 Nov 10 12:15 group_directory
-rw-rw---- 1 sammy sammy         15 Nov 10 17:07 group_modifiable
drwx------ 2 sammy sammy       4096 Nov 10 12:15 private_directory
-rw------- 1 sammy sammy        269 Nov 10 16:57 private_file
-rwxr-xr-x 1 sammy sammy      46357 Nov 10 17:07 public_executable
-rw-rw-rw- 1 sammy sammy       2697 Nov 10 17:06 public_file
drwxr-xr-x 2 sammy sammy       4096 Nov 10 16:49 publicly_accessible_directory
-rw-r--r-- 1 sammy sammy       7718 Nov 10 16:58 publicly_readable_file
drwx------ 2 root  root        4096 Nov 10 17:05 root_private_directory
```

Note that each file's mode (which contains permissions), owner, group, and name are listed.

Aside from the *Mode* column, this listing is fairly easy to understand.

To help explain what all of those letters and hyphens mean, let's break down the *Mode* column into its components.

To help explain what all the groupings and letters mean, take a look at this closeup of the *mode* of the first file in the example above:



File Type

In Linux, there are two basic types of files: *normal* and *special*.

The file type is indicated by the first character of the *mode* of a file–in this guide, we refer to this as the *file type field*.

Normal files can be identified by files with a hyphen (`-`) in their file type fields.

Normal files are just plain files that can contain data.

They are called normal, or regular, files to distinguish them from special files.

Special files can be identified by files that have a non-hyphen character, such as a letter, in their file type fields, and are handled by the OS differently than normal files.

The character that appears in the file type field indicates the kind of special file a particular file is.

For example, a directory, which is the most common kind of special file, is identified by the `d` character that appears in its file type field (like in the previous screenshot).

Permissions Classes

From the diagram, we know that *Mode* column indicates the file type, followed by three triads, or classes, of permissions: user (owner), group, and other.

The order of the classes is consistent across all Linux distributions.

Let’s look at which users belong to each permissions class:

- **User**: The *owner* of a file belongs to this class.
- **Group**: The members of the file’s group belong to this class.

- **Other**: Any users that are not part of the *user* or *group* classes belong to this class.

Reading Symbolic Permissions

The next thing to pay attention to are the sets of three characters, or triads, as they denote the permissions, in symbolic form, that each class has for a given file.

In each triad, read, write, and execute permissions are represented in the following way:

- **Read**: Indicated by an `r` in the first position.
- **Write**: Indicated by a `w` in the second position.
- **Execute**: Indicated by an `x` in the third position. In some special cases, there may be a different character here.

A hyphen (`-`) in the place of one of these characters indicates that the respective permission is not available for the respective class.

For example, if the *group* triad for a file is `r--`, the file is “read-only” to the group that is associated with the file.

Understanding Read, Write, Execute

Now that you know how to read which permissions of a file, you probably want to know what each of the permissions actually allow users to do.

We will explain each permission individually, but keep in mind that they are often used in combination with each other to allow for meaningful access to files and directories.

Here is a quick breakdown of the access that the three basic permission types grant a user.

Read

For a normal file, read permission allows a user to view the contents of the file.

For a directory, read permission allows a user to view the names of the file in the directory.

Write

For a normal file, write permission allows a user to modify and delete the file.

For a directory, write permission allows a user to delete the directory, modify its contents (create, delete, and rename files in it), and modify the contents of files that the user can read.

Execute

For a normal file, execute permission allows a user to execute a file (the user must also have read permission).

As such, execute permissions must be set for executable programs and shell scripts before a user can run them.

For a directory, execute permission allows a user to access, or traverse, into (i.e. `cd`) and access metadata about files in the directory (the information that is listed in an `ls -l`).

Examples of Modes (and Permissions)

Now that know how to read the mode of a file, and understand the meaning of each permission, we will present a few examples of common modes, with brief explanations, to bring the concepts together.

- `-rw-------`: A file that is only accessible by its owner
- `-rwxr-xr-x`: A file that is executable by every user on the system. A "world-executable" file
- `-rw-rw-rw-`: A file that is open to modification by every user on the system. A "world-writable" file
- `drwxr-xr-x`: A directory that every user on the system can read and access
- `drwxrwx---`: A directory that is modifiable (including its contents) by its owner and group
- `drwxr-x---`: A directory that is accessible by its group

As you may have noticed, the owner of a file usually enjoys the most permissions, when compared to the other two classes.

Typically, you will see that the *group* and *other* classes only have a subset of the owner's permissions (equivalent or less).

This makes sense because files should only be accessible to users who need access to them for a particular reason.

Another thing to note is that even though many permissions combinations are possible, only certain ones make sense in most situations.

For example, *write* or *execute* access is almost always accompanied by *read* access, since it's hard to modify, and impossible to execute, something you can't read.

Changing Directory and File Permissions

To view file permissions and ownership on files and directories, use the `ls -al` command.

The `a` option is to show hidden files or all files, and the `l` option is for the long listing.

The output will be similar to the following:

```
drwxr-xr-x 2 user user 4096 Jan  9 10:11 documents
-rw-r--r-- 1 user user  675 Jan  7 12:05 .profile
drwxr-xr-x 4 user user 4096 Jan  7 14:55 public
```

The first column with the ten letters and dashes shows the permissions of the file or directory.

The second column (with the single number) indicates the number of files or directories contained in the directory.

The next column indicates the owner, followed by the group name, the size, date, and time of last access, and finally the name of the file.

For example, using the first line from the output above, the details are as follows:

```
`drwxr-xr-x` are the permissions
`2` is the number of files or directories
`user` is the owner
`user` is the group
`4096` is the size
`Jan  9 10:11` is the date/time of last access
`documents` is the directory
```

Note: since a directory itself is a file, any directory will always show 4096 as it’s size.

This does not reflect the size of the contents of the directory.

Chmod Command

The command `chmod` is short for change mode.

Chmod is used to change permissions on files and directories.

The command `chmod` may be used with either letters or numbers (also known as octal) to set the permissions.

The letters used with **chmod** are in the table below:

Letter	Permission
r	Read
w	Write
x	Execute
X	Execute (only if file is a directory)

s	Set user or group ID on execution
t	Save program text on swap device
u	Current permissions the file has for owner
g	Current permissions the file has for users in the same group
o	Current permissions the file has for others not in the group

Note that the dash (-) denotes permissions are removed.

Therefore, with the “all others” group, r– translates to read permission only, the write and execute permissions were removed.

Conversely, the plus sign (+) is equivalent to granting permissions: `chmod u+r,g+x <filename>`

The example above translates as follows:

```
u is for user
r is for read
g is for group
x is for execute
```

In other words, the user was given read permission and the group was given execute permission for the file.

Note, when setting multiple permissions for a set, a comma is required between sets.

Chmod Octal Format

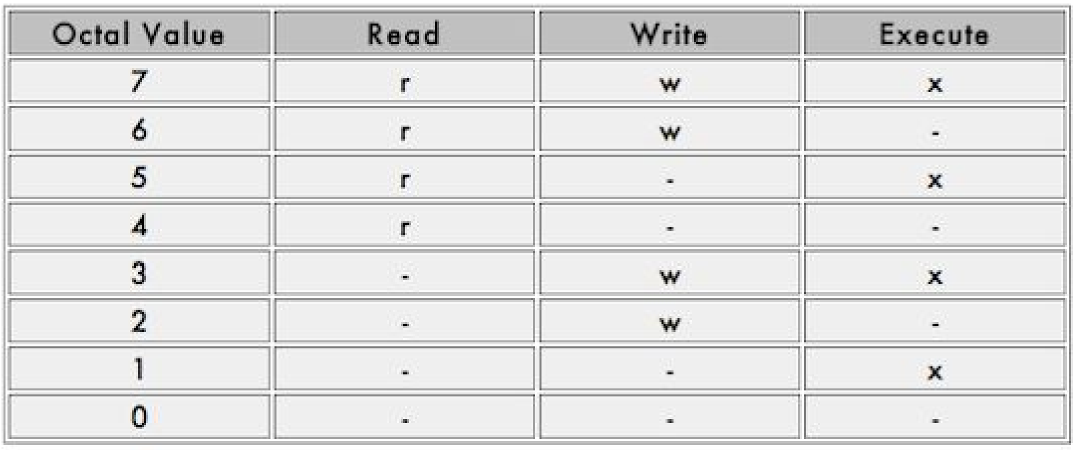
To use the octal format, you have to calculate the permissions for each portion of the file or directory.

The first ten characters mentioned above will correspond to a four digit numbers in octal.

The execute permission is equal to the number one (1), the write permission is equal to the number two (2), and the read permission is equal to the number four (4).

Therefore, when you use the octal format, you will need to calculate a number between 0 and 7 for each portion of the permission.

A table has been provided below for clarification:



Octal Value	Read	Write	Execute
7	r	w	x
6	r	w	-
5	r	-	x
4	r	-	-
3	-	w	x
2	-	w	-
1	-	-	x
0	-	-	-

Although octal format may seem difficult to understand, it is easy to use once you get the gist of it.

However, setting permissions with r, w, and x may be easier.

Below are examples of how to use both letters and octal format to set permissions on a file or directory.

Sample syntax: `chmod <octal or letters> <file/directory name>`

Letter format: `chmod go-rwx Work` (Deny rwx permission for the group and others)

The output of ls -al after the chmod command above would looks as follows:

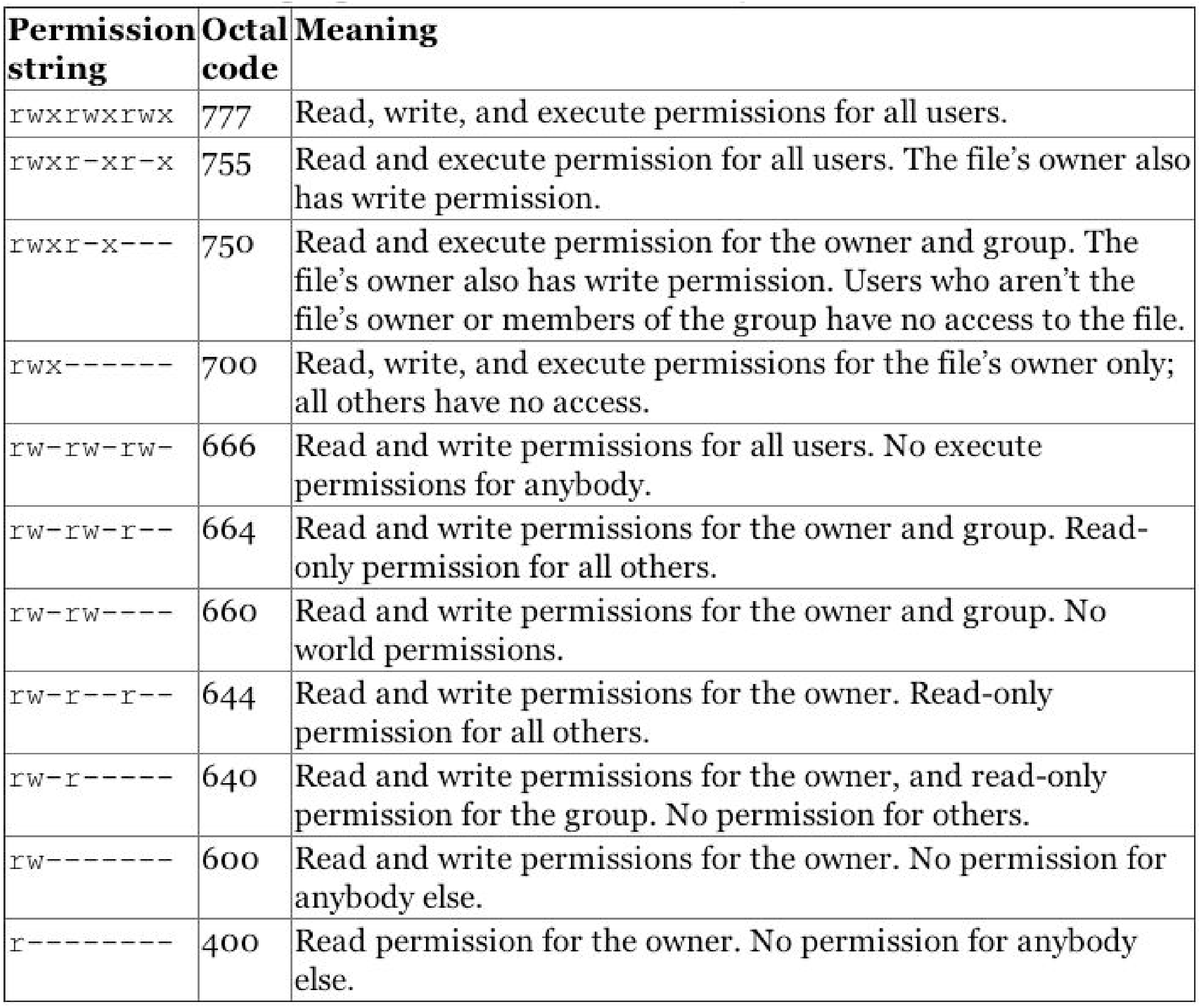
```
dr-------- 2 user user 4096 Dec 17 14:38 Work
```

Octal format: `chmod 444 Work`

The output of ls -al after the chmod command above would look as follows:

```
dr--r--r-- 2 user user 4096 Dec 17 14:38 Work
```

An octal table showing the numeric equivalent for permissions is provided below:



Permission string	Octal code	Meaning
rwxrwxrwx	777	Read, write, and execute permissions for all users.
rwxr-xr-x	755	Read and execute permission for all users. The file's owner also has write permission.
rwxr-x---	750	Read and execute permission for the owner and group. The file's owner also has write permission. Users who aren't the file's owner or members of the group have no access to the file.
rwx------	700	Read, write, and execute permissions for the file's owner only; all others have no access.
rw-rw-rw-	666	Read and write permissions for all users. No execute permissions for anybody.
rw-rw-r--	664	Read and write permissions for the owner and group. Read-only permission for all others.
rw-rw----	660	Read and write permissions for the owner and group. No world permissions.
rw-r--r--	644	Read and write permissions for the owner. Read-only permission for all others.
rw-r-----	640	Read and write permissions for the owner, and read-only permission for the group. No permission for others.
rw-------	600	Read and write permissions for the owner. No permission for anybody else.
r--------	400	Read permission for the owner. No permission for anybody else.

Changing File Ownership

By default, all files are "owned" by the user who creates them and by that user's default group.

To change the ownership of a file, use the `chown` command in the `chown user:group /path/to/file` format.

In the following example, the ownership of the "list.html" file will be changed to the "john" user in the "marketing" group:

```
chown john:marketing list.html
```

To change the ownership of a directory and all the files contained inside, use the recursive option with the `-R` flag.

In the following example, change the ownership of `/srv/smb/leadership/` to the "john" user in the "marketing" group:

```
chown -R john:marketing /srv/smb/leadership/
```

Changing Group Ownership

The **chgrp** command in Linux is used to change the group ownership of a file or directory.

All files in Linux belong to an owner and a group.

You can set the owner by using **chown** command, and the group by the **chgrp** command.

In the following example, the ownership of the “list2.html” file will be changed to the “sales” group:

```
chgrp sales list2.html
```

To change the group ownership of a directory and all the files contained inside, use the recursive option with the `-R` flag.

In the following example, change the ownership of `/srv/smb/team/` to the “sales” group:

```
chgrp -R sales /srv/smb/team/
```

- Exercises -

You are going to create a new Google Document inside the "3. Linux" folder of your Google Drive, named:

"3. 3. Users, Groups and File Permissions - Apellidos, Nombre"

being "Apellidos, Nombre" your Last Name and Name.

Share this Google Document with the teacher (jorge@iesdoctorbalmis.com) with "Edit" permissions.

Inside this Google Document you are going to copy and answer all the "Exercises" of this sub-unit.

1. Create a user with the username "michael" and a comment "Michael Smith from Sales". His home folder should be created automatically.
2. See the entry created for this new user.
3. Set a password ("Ciegsa1") for the user "michael".
4. Change the username of "michael" to "mike".
5. Change the home directory of the user "mike" to /home/mike .
6. Change the user ID of the user "mike" to "6666".
7. Lock the user "mike" and check that this user has been locked.
8. Unlock the user "mike" and check that this user has been unlocked.
9. Create a user with the username "john".
10. Delete the user "john" removing all his files.
11. View the current groups.
12. Create the group "Sales".
13. Modify the group ID of the group "Sales" to "9999".
14. Modify the name of the group "Sales" to "Marketing".
15. Add the user "mike" to the "Marketing" group and check that the user has been added to the group.
16. Update the primary group of the user "mike" to the "Marketing" group.
17. Create the group "HR".

18. Add the user "mike" to the "HR".

19. List all the groups of the user "mike".

20. Delete the group "HR".

21. Create the users "frank" and "bob", both with the password "Ciegsa1", and with the home directory automatically created.

22. Create the group "office1".

23. "frank" and "bob" should only be members of the group "office1".

24. Create the users "lisa" and "marie", both with the password "Ciegsa1", and with the home directory automatically created.

25. Create the group "office2".

26. "lisa" and "marie" should only be members of the group "office2".

27. As the user "frank", create in the home directory a file named "only-frank.txt", and modify the permissions of this file so only the user "frank" has all the permissions.

28. As the user "frank", create in the home directory a file named "only-office1.txt", and modify the permissions of this file so the user "frank" and the group "office1" has all the permissions.

29. As the user "frank", create in the home directory a file named "all-access.txt", and modify the permissions of this file so everyone has all the permissions.

30. As the user "bob", go to the directory /home/frank and try to edit the files "only-frank.txt" and "only-office1.txt": you should only edit the file "only-office1.txt".

31. As the user "lisa", go to the directory /home/frank and try to edit the files "only-frank.txt", "only-office1.txt" and “all-access.txt”. You should only edit the file “all-access.txt”.

32. As a sudoer user (such as "alumno"), change the file ownership of the file "only-frank.txt" to the user "lisa".

33. As a sudoer user (such as "alumno"), change the group ownership of the file "only-office1.txt" to the group "office2".

34. As the user "lisa", go to the directory /home/frank and try to edit the files "only-frank.txt", "only-office1.txt" and “all-access.txt”. You should edit the 3 files.

35. As the user "marie", go to the directory /home/frank and try to edit the files "only-frank.txt", "only-office1.txt" and “all-access.txt”. You should only edit the files "only-office1.txt" and “all-access.txt”.