PERMISSIONS AND OWNERSHIP: DOCUMENTATION

1- Basic ideas

a) Linux is a **multi-user system**. It means that more than one user can be operating the computer at the same time.

b) In a multi-user system, users should not be not allowed to:
- Interfere with the files belonging to another users
- Interfere with system files and directories (configuration files, boot programs, home directory, and so on)

c) In order to restrict what the users ara allowed to do, Linux comes wtih a control mechanism to determine who can access a particular file or directories and what actions they can do to it.

d) There are two parts to the file control mechanism: **Permissions** and **Ownership**.

e) Permissions determine what a user or group of users can do to a:
- File --> Read the contents, remove contents, add new contents, changes contents, execute (for binaries).
- Directory --> Add files, remove files, change files, list the contents, gain access to the directory.

f) Ownership determine the set of permissions obtained depending on who is the user or the group of users working with a file or directory.

g) Linux supports two methods of controlling who can access a file or folder and how they can acces it:
- The traditional Linux access permissions.
- ACL (Access Control Lists), which provide finer-grained control of access permissions.

This practical exercise discusses only the first method.

2- Basic Linux file and folder permission

a) In Linux, file and directory permissions and ownership control the access level that programs and users have to files. This ensures that only authorized users and programs can access specific files and directories.

b) Each file and directory has its own set of permissions. These permissions or access rights are assigned to users and groups. Permissions control the ability of users and groups to view or make changes to the contents of a file or directory.

c) In Linux, programs are bound to users and group and therefore, permissions granted to those users control the the ability of programs to view or make changes to the contents of a file or directory.

d) From the point of view of each particular file or directory, there are **three classes of users** with different kinds of ownership :

- A user called the **owner**:
  - The file or directory owner is by default the creator of that file or directory.
  - Ownership of files and directories can be changed on Linux.
  - The owner of a file or directory is the one who can assign and modify permissions for that file or directory.

- A set of users called the **group**:
  - Users who are members of this group share the same permissions and privileges on a file or directory due to their belonging to that group.
  - Group can be changed on Linux

- A set of users called the **others**:
  - Any user who is not member of **group** or is not the **owner**.
  - Members of others share the same permissions and privileges on a file or directory.

- The **root** user is a special user. Permissions and ownership do not have a practical effect on **root** user. This user can change ownership and permissions of any file or folder.

e) From the point of view of each particular file or directory, **three types of permissions** can be applied to each class of user :

- **read** permission:
  - If read permission is granted for a file to a user then, that user can view the contents of that file with the help of **nano**, **cat**, **geany**, **libreoffice**, **visual studio code**, etc...
  - If read permission is granted for a directory to a user then, that user can view the contents of that directory with the help **ls, tree**, etc...or their graphical equivalents.

- **write** permissions:
  - If write permission is granted for a file to a user then, that user can add/remove/modify the contents of that file with the help of **nano**, **cat**, **geany**, **libreoffice**, **visual studio code**, etc...
  - If write permission is granted for a directory to a user, that user can add/remove/modify the contents of that directory with the help **cp**, **rm**, **mv**, **nano**, etc... or their graphical equivalents.

- **execute** permissions:
  - If execute permission is granted for a file to a user and that file is a program then, that user can run that programan otherwise the user can not run that program.
  - If the execute permission is granted for a directory to a user then, that user is allowed to enter in the directory with the help of command **cd** or its graphical equivalent.

f) Permissions can be **denied** or **allowed**.

g) For every file and directory on your system is mandatory to specify:
- an owner and group
- permissions denied and allowed for the owner
- permissions denied and allowed for the group
- permissions denied and allowed for others

h) When you combine ownership and permissions, you will be able to control who can access files and folders and what actions they are able to do with it. Three kind of permissions and three kind of users means that for every file in your system, **9** parameters have to be set.

i) For every file or folder on the system, permissions are assigned to users by following these steps:
1st) If the user is the file/folder owner then the user gets the permissions given to the owner.
2nd) If the user is not the file/folder owner but is member of the group then, the user gets the permissions assisgned to the group. Permissions assigned for others are not taken into consideration.
3rd) If the user is not the file/folder owner and is not member of the group then the user gets the permissions given to others.

j) The following commands can display/create/change permissions and ownership for any file or folder on your system:
- **ls -ls** → displays access permissions and ownership (additionaly, it shows file size and last modification date and time)
- **tree -pug** → recursively displays access permissions and ownership
- **chmod** → sets or unsets access permissions
- **chown** → changes file owner and group
- **chgrp** → changes group ownership
- **id** → shows a list of groups (name and identifier number) a user is member of

k) Additional (not mandatory) readings:
- https://linuxize.com/post/understanding-linux-file-permissions/
- https://linuxcommand.org/lc3_lts0090.php

3- Interesting facts

a) You can see the contents of a folder using the graphical user interface and therefore you could believe that you have gained access to the directory but that's not true. You can see the contents but you are not in the directory. That happens because the GUI run 2 operations: it changes to the directory and it shows the list of its contents. The first operation does not works but the second operation works.

b) You can run a compiled program (for exemple a c program) even if you remove the execution permission. That happens because the program responsible for running a compiled program is the operating system and like the user root, the operating system is not affected by permissions over files and directories. But an interpreted program like (for example a python or bash script program) requires the help of an external program called interpreter to be run and this external program is subjected to the permissions and ownership rules.

4- Commands

4.1- *ls -ls* command: Displaying file/folder access permissions and ownership

a) When you run **ls** with **-ls** option and the name of a file, the command **ls** displays a line of localrmation about the file. For instance:
dacomo@inf1-dacomo:~$ ls -ls zpack.atr.gz
376 -rwxr-xr-- 1 dacomo teachers 382911 Nov 23 zpack.tar.gz

From left to right, the line contains the following localrmation:

Size (blocks)	Type	Permissions	Number of Links or directories inside	Owner	Group	Additional localrmation
376	-	rwxr-xr--	1	dacomo	teachers	382911 Nov 23 zpack.tar.gz
Blocks of 1024 bytes	**-** for a file **d** for a folder **l** for a link	**r** indicates read permission **w** indicates write permission **x** indicates execute permission **-** The user does not have the permission in that position	**1** for a file **1 or more** for a folder	Name of the owner	Name of the group	- Size in bytes - The date when the file/folder was created or modified - The name of the file or folder

b) The nine characters of Permissions are divided in three groups:
- First group (characters from 1st to 3rd): Permission for the owner of the file/folder
- Second group (characters from 4th to 6th): Permissions for the special group.
- Third group (characters from 7h to 9th): Permissions for others.

c) When you run **ls** with **-lsd** option and the name of a directory, the command **ls** displays a line of localrmation about the directory. For instance:

dacomo@inf1-dacomo:~$ ls -lsd Desktop
-rwxr-xr-- 1 student00 students 465 22 may 2011 README

d) Recursive option **-R** for folders --> **ls -ls -R folder_name**. Example: **ls -ls -R /boot**

4.2- *tree -pug* command: Displaying folder access permissions and ownership in a tree-like format

a) Description: The **tree -pug** command displays the access permissions, owner and grup of a folder, recursively in a tree-like format.

b) Synopsis: **tree -pug /home/dacomo**

4.3- *chmod* command: Changing access permissions

a) Description: The **chmod** command-line utility changes the access permissions of a file or folder

b) Synopsis: **chmod <permissions> file_or_folder_name**

c) Permissions in numeric mode: A three digit number in octal format (0 to 7):
0 octal => 000 binary => **---** 4 octal => 100 binary => **r--**
1 octal => 001 binary => **--x** 5 octal => 101 binary => **r-x**
2 octal => 010 binary => **-w-** 6 octal => 110 binary => **rw--**
3 octal => 011 binary => **-wx** 7 octal => 111 binary => **rwx**

d) Permissions in symbolic mode: **ugoa** (user/group/other/all), **+/-** (add/remove), **rwx** (read,write,execute)

e) Examples:
chmod 754 prova.sh
- owner permissions: read, write and excute
- group permissions: read and execute
- other permissions: read.

chmod 310 prova.sh
- owner permissions: write and execute
- group permissions: execute
- other permissions: none

chmod u+r prova.sh => Adding read permissions to owner user.

chmod g-x prova.sh => Removing execute permissions to group.

chmod a+x prova.sh => Adding execute permissions to all (everyone).

chmod ug+rw prova.sh => Adding read and write permissions to owner user and group.

chmod ugo-wx prova.sh => Removing write and execute permissions to owner user, group and others

chmod a-wr prova.sh => Removing write and read permissions to all (owner, group and others)

f) Recursive option **-R** for folders --> **chmod -R <permissions> folder_name**.

Example: **chmod -R 755 /home/student00**

Permissions of all files and folders in **/home/student00** will be changed to **rwxr-xr-x** using this single command.

4.4- *chown* command: Changing user and group ownership

a) Description: The **chown** command-line utility changes the owner and group of a file/folder.

b) Synopsis 1: **chown <new_owner:new_group> file_or_folder_name**

c) Synopsis 2: **chown <new_owner> file_or_folder_name**

d) Examples:

chown etpclot:users prova.sh => Changes to user **etpclot** and **group** users the ownership of file **prova.sh**.
chown etpclot prova.sh => Changes to user **etpclot** the ownership of file **prova.sh**.

e) Recursive option **-R** for folders **--> chown -R <new_owner:new_group> folder_name.**

Example: **chown -R etpclot:users /home/student00**

Ownership of all files and folders in **/home/student00** will be changed to **etpclot:users**

4.5- *chgrp* command: Changing group ownership

a) Description: The **chgrp** command-line utility changes the group of a file/folder.

b) Synopsis: **chgrp <new_group> file_or_folder_name**

d) Examples:

chgrp users prova.sh => Changes to **users** the group of file **prova.sh**.
chown users /home => Changes to **users** the group of folders **/home**.

e) Recursive option **-R** for folders --> **chgrp -R <new_group> older_name**.

Example: **chgrp -R users /home**.

Group of all files and folders in /home will be changed to users using this single command.

4.6- *id* command: Displaying list of groups of which a user is member

The **id** command-line utility print a list of groups of which a user is a member. For instance:

dacomo@inf1-dacomo:~$ **id**
uid=1000(dacomo) gid=1000(dacomo) grups=1000(dacomo),24(cdrom),25(floppy),27(sudo),
29(audio),30(dip),44(video),46(plugdev),109(netdev),113(bluetooth),120(scanner),998(vboxsf)

In this example, **dacomo** is member of: dacomo, cdrom, floppy, sudo, audio, dip, video, plugdev, netdev, bluetooth, scanner and vboxsf

5- Writing proper sentences about permissions

Read (mandatory) the following document: About Permissions