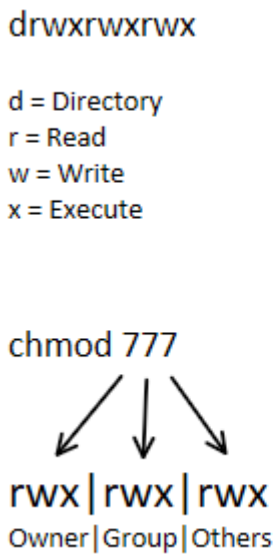
Lab 4 Notes

Permissions

- Permissions are defined as `r w x` in Linux.

File Type	r	w	x
Normal File	Being able to extract the content of file: `cat file`	Being able to modify the content of file: `vim file`	Being able to execute the file: `./file`
Directory	Same as file: `ls dir`	Being able to move/copy within the directory: `mv` or `cp` or `rm`	Being able to "access" the directory: `cd dir`

- Every user has `x` permission within their home directory **only**.
- Every file has an owner and a group owner.
- For a file, permissions are viewed by: Owners, Group Owners and Others



7	rwx	111
6	rw-	110
5	r-x	101
4	r--	100
3	-wx	011
2	-w-	010
1	--x	001
0	---	000

- `id` is used to print out the uid, gid and groups of the currently logged in user.
- `chmod <my_permissions> <file>` is used to change the permissions of files.
  - There are many different formats of `chmod`:
    1. `chmod <permission_in_octal> <file>`, for example: `chmod 777 file`
    2. `chmod u=<permission_for_user>, g=<permission_for_group>, o=<permission_for_others> <file>`, for example: `chmod u=rwx, g=r, o=w file1`
    3. `chmod u+<permission> g+<permission> o+<permission>` → in here, the + is used to **add** the permission, you can use - to **remove** the permission, for example: `chmod u+rx g-r o+x`. You can also use `a` to change permission

for users, groups and others at the same time, for example: `chmod a+x` → means *add* `x` permission to users, groups and others.

- You may use the octal system as it is faster, I'll be adding a list that contains most common permissions.

Octal Representation	Actual Permissions	Commonly used with
777	`rwxrwxrwx`	Files/Directories
664	`rw-rw-r--`	Files/Directories
755	`rwxr-xr-x`	Directories

- You can use `chmod -R` to recursively apply your permissions onto the files within the directory (Including subdirectories)
- `chown <new_owner:new_group_owner> <file>` → This changes the user owner and group owner of a file and this command can only be executed by **root**.
  - You can also use `-R` option to do this recursively, much like `chmod`.
- You can change the default permissions when creating files, we use `umask`.
  - `umark <complement_of_wanted_permission>` → We basically **negate** the permissions we want and it is put as a *mask*.

> **This only works during the current login session because if you logout, these changes will be overridden by the initialization scripts located in `/etc/profile`.**
> To make it permanent: Go to the line that has the umask command and change it.

- Maximum permissions for a directory is 777 meanwhile a file is just 666.

> **This can be noticed when you do `umask 000` (Create all files and directories with full permissions) → This is because `touch` already masks the newly created file with 666 but `mkdir` masks the newly created directory with 777.**

- Passwd command is **always** executed by root, this is because there is an advanced permission added to `/bin/passwd` that is `s` (set userid)

Permission	Details	Use cases	Effect
`s`	Set-UID	Binary files (Executables)	It executes the command with the specified uid as it's user owner. (بينفذ الأمر كإنه مالكه)
`s`	Set-GID	Binary files (Executables)	It executes the command with the specified uid as it's group owner.
`t`	Sticky Bit	Directories	It makes users unable to delete other users' file, it allows people to create other files but

Permission	Details	Use cases	Effect
			disallows removal only. (مش بتسمح لحد يشيل حاجة مش بتاعته)

These advanced permissions have an extra placeholder in the octal system, meaning that they take one extra space (s s t) → As a consequence: we can do `chmod 4777`: set Set-UID bit and `rwxrwxrwx`.

You can also use `chmod u+s g+s o+t` for some systems (Doesn't work in RHEL)

- You can find these bits capitalized if the underlying `x` is not set. (S S T) but that is rarely found.