File permissions and ownership in Linux

- UNIX based operating systems have always been multi-user systems, meaning there is more than one user in the system.
- Users in linux have permissions that restrict access to privileges such as read and write to files and directories.
- By default, the user that creates the file or directory naturally becomes the owner of that file.

Listing permissions for files and directories

```Bash
$ ls -l
    -rwx--xr-- abdul_samad 61 Nov  9 12:42:33 hi.txt*
```

the `-l` flag stands for long, listing format

Understanding `-|rwx|rw-|r--`

- The first part, that is `-` OR `d` indicates the content is a file or a directory
- The second part is pemissions for the owner
- The third part is permissions for the group that the owner belongs to.
- The fourth and last part is the permissions of everyone else (public).

Changing ownership of a file (`chown`)

In order to change ownership of a file or a directory, you use a command called `chown` (change ownership).
Changing ownership of a file or a directory automatically gives permissions of read/write but not execute to the user.
Example:

```Bash
$ sudo chown boy hi.txt
```

Change ownership of file `hi.txt` to `boy`.

Setting permissions for groups, and other users using `chmod`.

`chmod` (change mode bits) is a command line utility that lets us set file permissions for different users as well as groups.

If we want to set permission like `rwx-rw-r--` we can do so by two methods:

- ***Absolute method*** (numerical method)
  Each read, write, and execute has a number value
  - `r` -> 4
  - `w` -> 2
  - `x` -> 1
  - no-permission -> 0

  If we want to set permission like `rwx-rw-r--` we can do so like,
  - 1st part -> r+w+x = 4 + 2 + 1 = *7*
  - 2st part -> r+w = 4 + 2 = *6*
  - 3st part -> r = 4 = *4*
    So, the final number becomes, *764*

  We use this number like,

  ```Bash
  $ sudo chmod 764 hi.txt
  $ ls -l
      -rwxrw-r-- abdul_samad 61   Nov  9 12:42:33 hi.txt*
  ```

- ***Symbolic method***

```Bash
$ sudo chmod u=rwx,g=rw,o=r hi.txt
```

This method is much easier to understand and doesn't require mathematical calculations

Process management in Linux

- `ps`: Display a snapshot of current processes
- `pstree`: Display process tree
- `top`: Process monitoring tool on linux
- `htop`: Process monitoring tool with much more functionality than top (It needs to be installed using your distributions' package manager)

Keybindings while running a process from the shell:

- `Ctrl-C`: Send interrupt/terminate signal to running process

```Bash
$ sleep 50
^C
Terminated
```

- `Ctrl-Z`: Send process to background

```Bash
$ sleep 50
^Z
[1]+  Stopped                 sleep 5
$ fg
```

`fg` brings the last process that was sent to the background to the foreground.
Do `fg <number>` to bring a particular background job to the foreground.
Example:

```Bash
$ sleep 50
^Z
[1]+  Stopped                 sleep 50

$ sleep 20
^Z
[2]+  Stopped                 sleep 20

$ fg 2
sleep 20
```

Here `fg 2` will bring the second background job to the foreground and not the first.

Process attributes (in `htop`/`top`)

- `pid`: Process id
- `user`: Owner (user) of that process
- `pri`: Priority of the process
- `S`: Status (zombied/running/sleeping)
- `CPU%`: CPU usage %
- `MEM%`: MEM usage %

- `time`: Total CPU time taken by that process
- `Command`: The command that started that process
- `Ni`: The "niceness" i.e The resource usage limitation
  (A niceness value of 20% means that the process can only use a maximum of 20% of the CPU)

Terminating processes

- `kill`: Kill or terminate a process by its PID

```Bash
$ kill 19342
```

- `pgrep`: Print the PID of a process by its name

```Bash
$ pgrep firefox
24312
```

Note: You can use this PID to kill the process

- `pkill`: A wrapper to `kill`.
- `killall`: Kill all processes matching a name.

```Bash
$ killall firefox
```

Executing commands from the shell and run it in the background

```Bash
$ firefox & disown
```

`& disown` tells the shell to separate the current shell with the shell that firefox is running on.
If we were to close our terminal, the process is still going to run since, we have separated it using `& disown`.
If we don't separate the process and simply close our terminal, this will trigger a chain reaction that leads to termination of the shell and then firefox that was running on that shell