Introduction to Command Line

Here, you will briefly cover command line and will discuss a few special characters/operators which come in really handy while using the terminal.

What is the Command Line?

> A *command-line interface* (CLI) or a command line interpreter or shell is simply a mean for the user to interact with the system in the form of progressive and sequential commands.

A useful way for the users to conduct various actions like viewing, handling, and manipulating files on the computer is *Graphical-User-Interface* or GUI. However, CLI is the primary approach to interact with the computer.

CLI has essential *advantages*, especially for meticulous computer users. It lets you more control over your system by allowing you to run system commands and automate your various tasks. Similarly, you can add modifiers to define exactly how you want your program to execute.

Types

“Shells” have evolved through quite a many adaptations. In *OS X* and *Linux*, most commonly used shell is `bash`, whereas in *Windows* has `MS-DOS` based CLI.

Open Your Command Line Interface

- In Linux:

Press `Ctrl+Alt+T`

- In OS X:

`Applications → Utilities → Terminal`

- In Windows:

Press `Win-R`, type `cmd`, press `Enter`

`Start → Program Files → Accessories → Command Prompt`

Observe the Prompt

The command line prompts in different ways for different OS environments.

- In Windows: `>`
- In Linux or OS X: `$`

Embark on with your First Command

So, you've made it this far. Hopefully, you might have grasped a basic understanding of shell. Now start off with your first command:

```
whoami
```

This command returns the username of the owner of current login session. So, your computer just printed your name, right? Wow!

Basic Command Line Editing

You can use the following key combinations to edit and recall commands:

- **Esc** + **T**: Swap the last two words before the cursor
- **Ctrl** + **H**: Delete the letter starting at the cursor
- **Ctrl** + **W**: Delete the word starting at the cursor
- **TAB**: Auto-complete files, directory, command names and much more
- **Ctrl** + **R**: To see the command history.
- **Ctrl** + **U**: Clear the line
- **Ctrl** + **C**: Cancel currently running commands.
- **Ctrl** + **L**: Clear the screen
- **Ctrl** + **T**: Swap the last two characters before the cursor

Special Characters in Bash

Each special character, in Bash, holds a unique meaning. Let's look at this table to find out the meaning of each character:

Characters	Description
/	Directory separator, used to separate a string of directory names. **Example:** `/home/projects/file`
\	Escape character. If you want to reference a special character, you must "escape" it with a backslash first. **Example:** `\n` means newline; `\v` means vertical tab; `\r` means return
#	Lines starting with `#` will not be executed. These lines are comments
.	Current directory. When its the first character in a filename, it can also "hide" files
..	Returns the parent directory
~	Returns user's home directory
~+	Returns the current working directory. It corresponds to the `$PWD` internal variable
~-	Returns the previous working directory. It corresponds to the

~-	directory. It corresponds to the `$OLDPWD` internal variable
*	Represents 0 or more characters in a filename, or by itself, it matches all files in a directory. **Example:** `file*2019` can return: `file2019`, `file_comp2019`, `fileMay2019`
[]	Can be used to represent a range of values, e.g. [0-9], [A-Z], etc. **Example:** `file[3-5].txt` represents `file3.txt`, `file4.txt`, `file5.txt`
|	Known as “pipe". It redirects the output of the previous command into the input of the next command. **Example:** `ls | less`
<	It redirects a file as an input to a program. **Example:** `more < file.txt`
>	In **script name** >**filename** it will redirect the output of “script name” to “file filename”. Overwrite filename if it already exists. **Example:** `ls > file.txt`
>>	Redirect and append the output of the command to the end of the file. **Example:** `echo "To the end of file" << file.txt`
&	Execute a job in the background and immediately get your shell back. **Example:** `sleep 10 &`

&&	"AND logical operator". It returns (success) only if both the linked test conditions are true. It would run the second command only if the first one ran without errors. **Example:** `let "num = (( 0 && 1 ))"`; `cd/comp/projs && less messages`
;	"Command separator". Allows you to execute multiple commands in a single line. **Example:** `cd/comp/projs ; less messages`
?	This character serves as a single character in a filename. **Example:** `file?.txt` can represent `file1.txt`, `file2.txt`, `file3.txt`

Exiting the Command Line

You can now safely exit from the Shell by writing:

```
exit
```