THE KNOWLEDGE HOUSE

# MODULE 6

## BASH FUNDAMENTALS

## OBJECTIVE

To learn to navigate and create project structure through the terminal.

### ANSWER THE FOLLOWING QUESTIONS IN A .TXT FILE:

WHAT_IS_A_BASH

- Explain in your own words what is Bash? (300 Words)
- Explain What is the terminal? (200 Words)
- Explain in your own words what are Bash Commands: ( 250 Words )
- Explain why is the terminal essential? (250 Words)
- Explain Bash piping (250 Words)

# MODULE 6
## BASH FUNDAMENTALS

### WHAT IS BASH?

When you interact with your computer you typically do so through a **Graphical User Interface or GUI.**

This happens when you create a new folder, save a file, open a file, search for things on your hard drive, etc.

Consider this; you are speaking with your computer through an interpreter. GUI's basically just visualize a set of commands that are sent to your computer to have it do something with the files on your hard drive.

**In addition to the things listed above, this can include things like:**

- Creating files and directories (folders)
- Opening a directory to view its content.
- Changing permissions (making a file admin only for example)
- Changing file names
- Copying or moving files
- Renaming files and directories
- Deleting things
- Basically, everything you can do on a computer outside of the browser and installed programs.

However, as a programmer, you need to learn to lose this crutch. The interpreter is for those who do not want to speak to the computer directly and that is not you. Programmers talk to computers face to face through the Terminal.

Ever seen programmers typing obscure commands into a black screen? Yea? That black screen is the terminal. Working with the terminal seems terrifying for people but it actually is much more efficient and over time we almost guarantee you will prefer it to regular GUI's for creating file structures for your projects.

# MODULE 6
## BASH FUNDAMENTALS

## WHAT IS BASH? CONT.

Say hello to the terminal:


**How to access your terminal:**

For Mac/Linux users you are in luck there are generally speaking two types of terminals the BASH terminal which comes standard on Macs, Linux Machines, and Newer Windows Versions, this handles unix commands and is pretty standard for programmers (unless you are programming specifically for windows).
For the Windows people, you are stuck with the Windows CMD prompt, which has its own set of parallel commands BUT you are in luck, Windows supports an application called Powershell which basically runs all the unix BASH commands the terminal does and works largely the same way. Newer Windows versions (10+) have their own BASH you can get from the app store, but to be safe I will go over how to install Powershell since it works on all recent windows machines.

**Powershell Vs BASH**

**Opening Powershell:**

| | |
|---|---|
| Windows 10 | Click left lower corner Windows icon, start typing PowerShell |
| Windows 8.1, 8.0 | On the start screen, start typing PowerShell. If on desktop, click left lower corner Windows icon, start typing PowerShell |

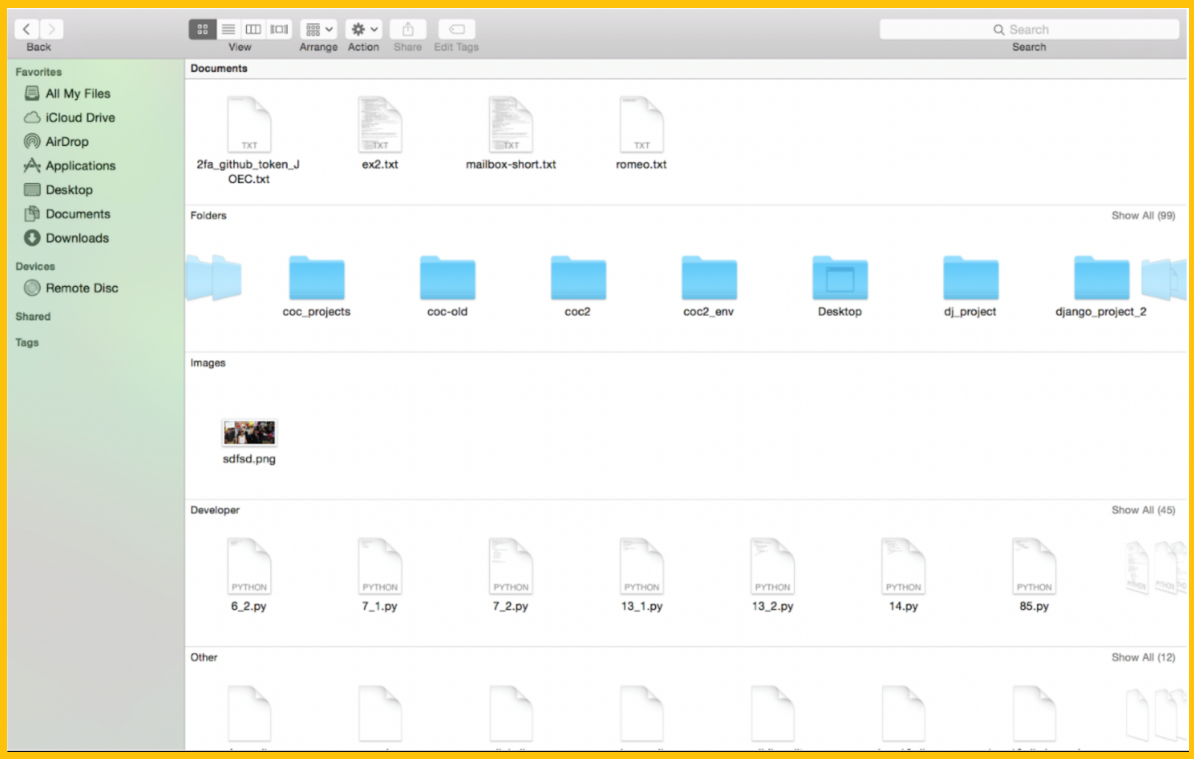
# MODULE 6
## BASH FUNDAMENTALS

## OPENING TERMINAL ON MACOS:

Go to the search icon in the top right corner of the nav bar; type in “terminal”

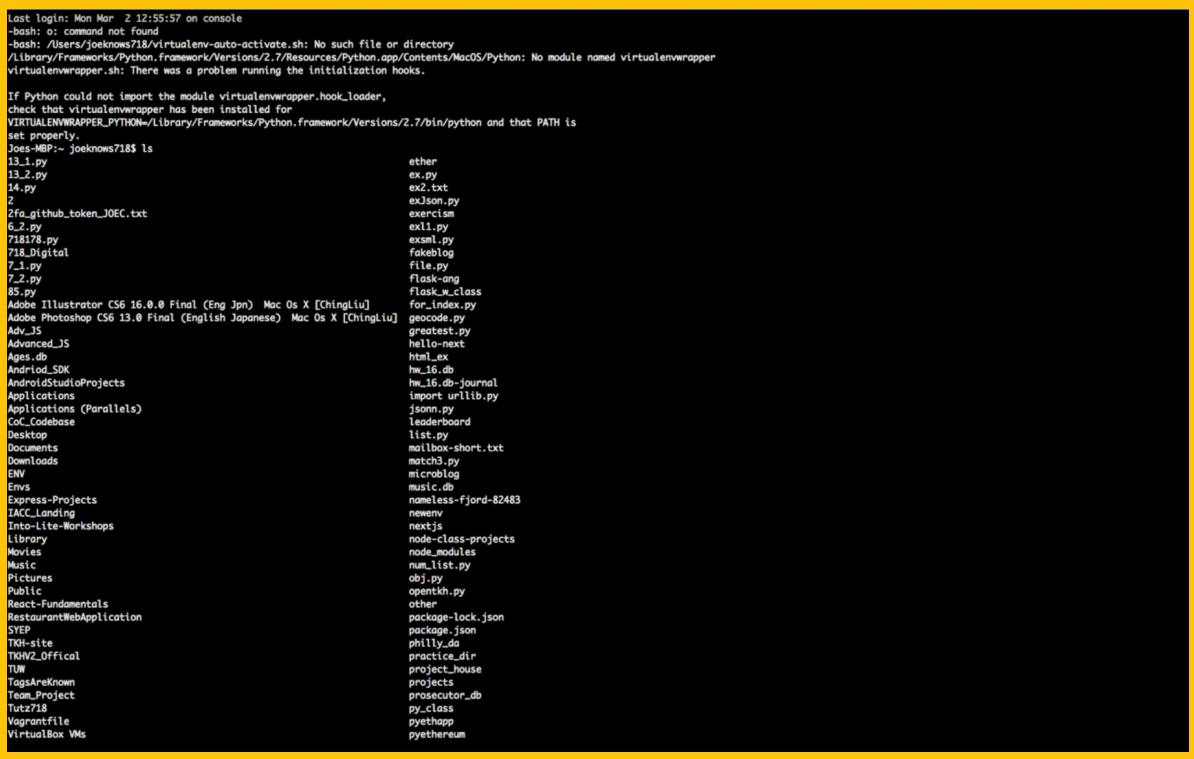
**A word on folders:**
**When you open your terminal whether bash or powershell it opens to a location on your computer.**

In bash on linux and MacOS the home folder is the folder associated with your username, this is your home folder, which contains your downloads and documents folder.



# MODULE 6
## BASH FUNDAMENTALS

## OPENING TERMINAL ON MACOS: CONT.




These screens represent the same thing. One is thru the GUI and one is thru the terminal.
If you are ever confused about where your terminal opens up to you can print your home folder :

```
#print home folder
$ echo $HOME
#print current folder
$ echo $PWD
```

**EX1: Open your terminal, print your home, print your working directory**

# MODULE 6
## BASH FUNDAMENTALS

### NAVIGATION THROUGH THE TERMINAL:

Just like your GUI you can navigate folder to folder in bash using the command cd or Change Directory.

```
$ cd Desktop
#changes directory to desktop
$ cd Desktop/Project1
#skipps two folders over
#if you did this you would be in homefolder/Desktop/Project1
#If you wanted to skip back to Desktop you could go back with the following command:
$ cd ..
#To jump back to home, you can skip two folders back:
$ cd ../..
#this would land you back at home
#also note you can autofill folder names by hitting the tab once you start
#writing the name of the directory
$ cd .
#takes you back to home
#so $ cd De + TAB would autoful Desktop
```

# MODULE 6

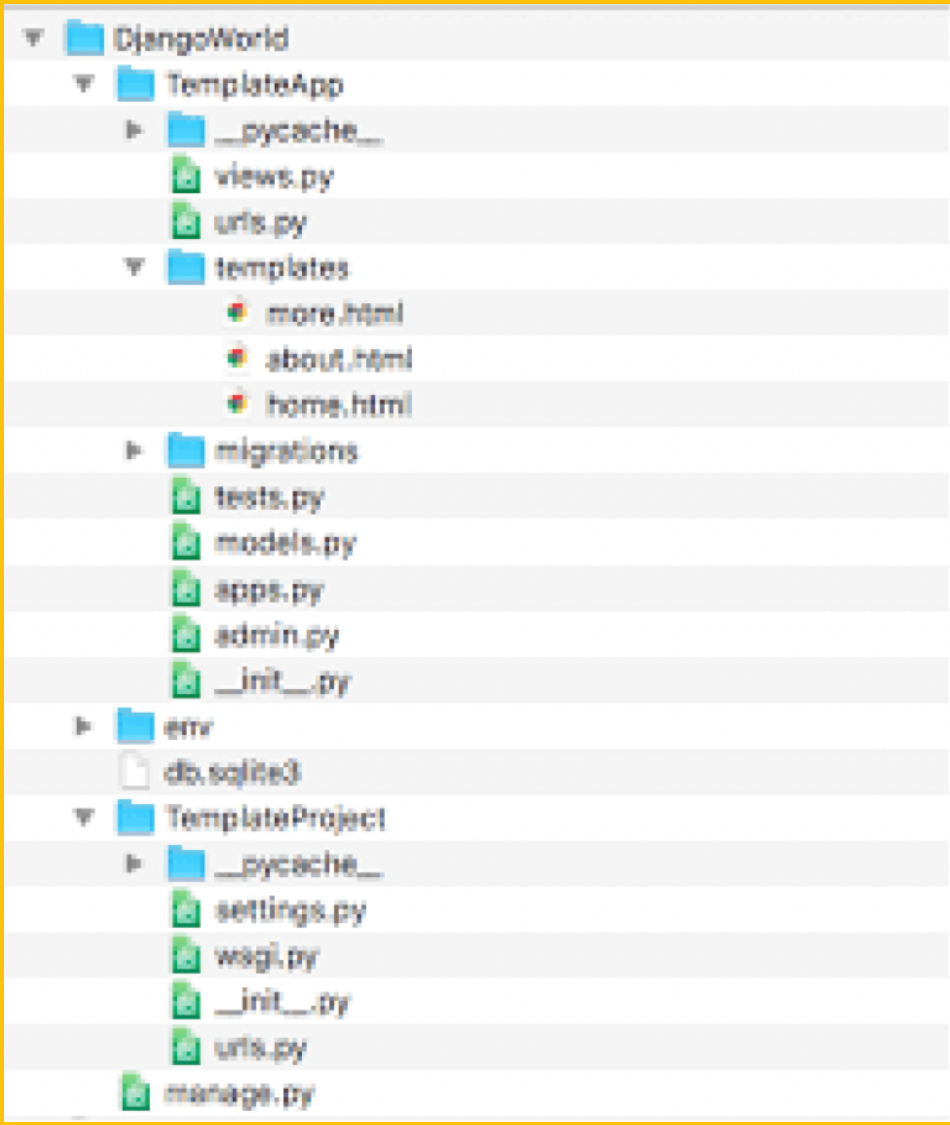
## BASH FUNDAMENTALS

### NAVIGATION THROUGH THE TERMINAL: CONT.

**So now we know where we are let's make some stuff:**

The main way you will be using the terminal is going to be creating project folders and file structures.

One of the keystones to be a developer is following best practices in setting up your projects: Below is a django project folder structure:



# MODULE 6
## BASH FUNDAMENTALS

## NAVIGATION THROUGH THE TERMINAL: CONT.

You would be setting these up through the terminal, doing so from the GUI is not a best practice and is not an acceptable move professionally. So let's talk about setting your project up.

**There are a few key commands here:**

```
#creating a new folder with the make directory command:
$ mkdir FOLDERNAME
#this creates a new directory(folder) called FOLDERNAME in the directory (folder) you are #currently working in. So doing this in your homefolder will result in homefolder/FOLDERNAME
#being created
```

Next we want to make some files:

```
#We make files by using the touch keyword
$ touch filename
#this creates a new file in whicher your working directory is
```

You can also list our the contents of whatever directory you are in:

```
$ls
#for Bash
$ dir
#for powershell
```

Moving folders and files also might be necessary:

```
$mv /homefolder/filename /homefolder/Documents
#the first argument taken is the location of the file to move, the second argument is the files #destination
#this command moves the filename file and moves it from
#the home folder to the Documents folder
$mv /homefolder/filename2 /homefolder/filename3
#this command simply renames the file because you are moving the file contents to a new file with a new name at the same directory location.
```

# MODULE 6
## BASH FUNDAMENTALS

## NAVIGATION THROUGH THE TERMINAL: CONT.

These are some of the most commonly used commands for setting up your developer environments and files structures, but there are more.

```
#Here are some common commands you might want to use:
$ clear
#this clears out your terminal window
$ cp filelocation/filename newlocation/newname
#above commands copies files and directories and places it in a new location
#Lastly is the alias command that will allow you to save common commands as whatever you #want
$alias c = "clear"
#now you can call the clear command by typing c into the terminal
```

**...Let's slow it down a bit...**

A lot of times, beginners are so used to working with the GUI-based interface, that they tend to overlook the capabilities of the command-line interface(CLI). A mouse comes in real handy when we need to copy about a hundred thousand files into a folder, but what if were to rename all those thousand files or were to separate them based on their extensions? Since GUIs are not programmable, it would take forever for us to rename or separate them With the command line, however, we could quickly achieve this in a few lines of code.

The Unix shell is a pretty powerful tool for developers of all sorts. This article intends to give a quick introduction to the very basics starting from the UNIX operating system.

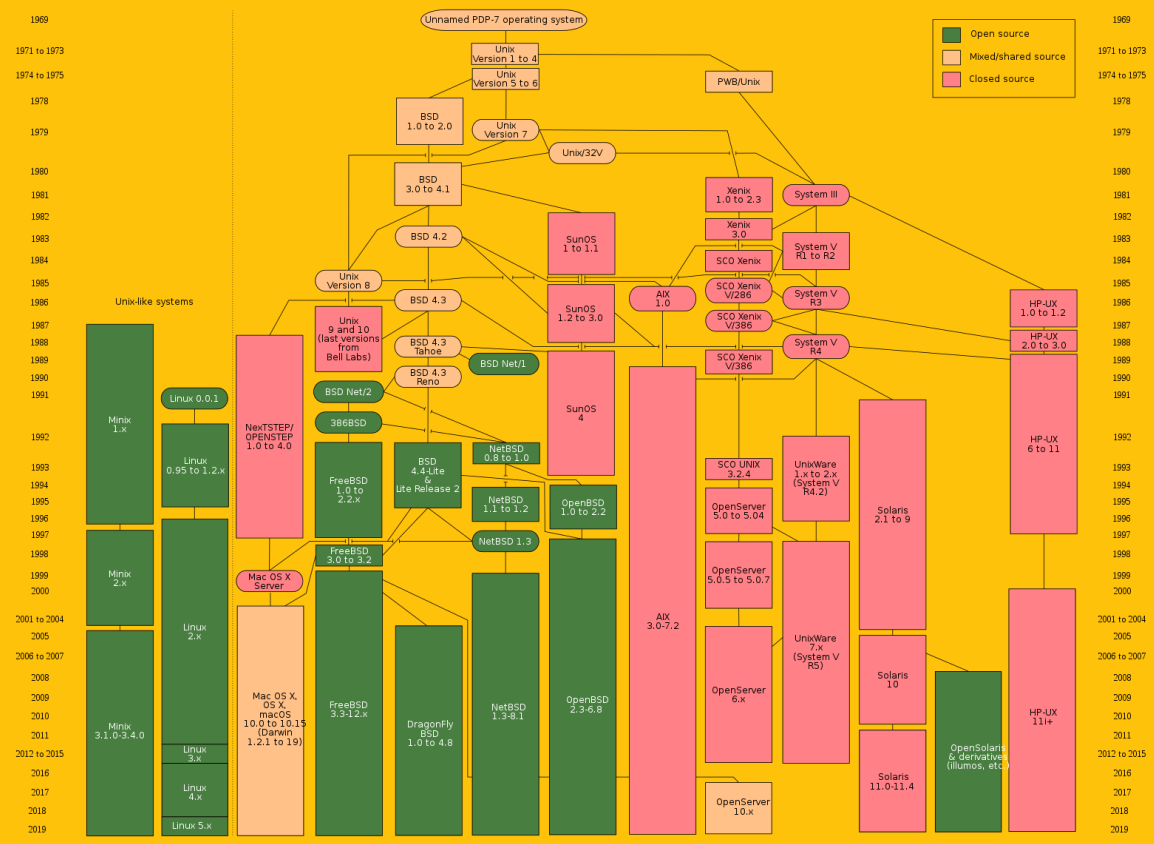
## UNIX

Most operating systems today except the WINDOWS based, are built on top of UNIX. These include a lot of Linux distributions, macOS, iOS, Android, among others. A mere glance at the family tree of UNIX-based operating systems is sufficient to highlight the importance of UNIX, and this is the reason why it has been so widely adopted in the industry. In fact, the back end of many data and computing systems, including industry giants like Facebook and Google, heavily utilize UNIX.

# MODULE 6

## BASH FUNDAMENTALS

## UNIX CONT.




The UNIX Family Tree. Source: **WIkipedia**

## SHELL

Shell is a command-line interface for running programs on a computer. The user types a bunch of commands at the prompt, the shell runs the programs for the user, and then displays the output. The commands can be either directly entered by the user or read from a file called the shell script or shell program.

# MODULE 6
## BASH FUNDAMENTALS

## SHELL TYPES

**The UNIX system usually offers a variety of shell types. Some of the common ones are:**

1. **Bash : Bourne Again shell**
   The standard GNU shell, intuitive and flexible
2. **ksh : Korn shell**
   A superset of the Bourne shell
3. **csh : C shell**
   The syntax of this shell resembles that of the C programming language
4. **tcsh: TENEX C shell**
   A Superset of the common C shell, enhancing user-friendliness and speed
5. **zsh : Z Shell**
   An extended Bourne shell with a large number of improvements, including some features of Bash, ksh, and tcsh.

Common Shell Types

We shall, however, limit ourselves to the Bash shell in this article. However, you are encouraged to read and try the other shells also especially the zsh shell since, in the latest MacOS called Catalina, zsh will replace the bash shell. So it'll be a good idea to get to know it, now.

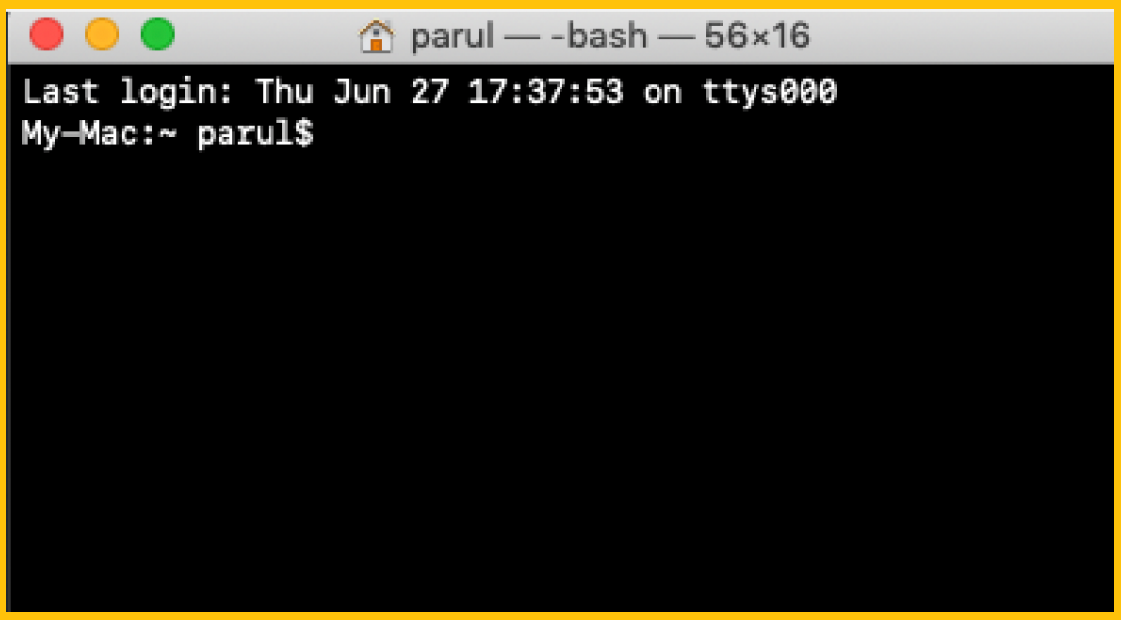
## TERMINAL

The terminal is a program that is used to interact with a shell. It is just an interface to the Shell and to the other command line programs that run inside it. This is akin to how a web browser is an interface to websites.

# MODULE 6
## BASH FUNDAMENTALS

## TERMINAL CONT.

**Here is how a typical terminal on Mac looks like:**




A Typical Mac Terminal

Mac and Linux have their respective versions of the terminal. Windows also has a built-in command shell, but that is based on the MS-DOS command line and not on UNIX. So let's see how we can install a shell and a terminal program on Windows that works the same as the ones on Mac and Linux.

## INSTALLATION ON WINDOWS

- Windows Subsystem for Linux (WSL)

  It is a new Linux compatibility system in Windows 10. The WSL lets developers run GNU/Linux environment — including most command-line tools, utilities, and applications — directly on Windows, unmodified, without the overhead of a virtual machine. You can read more about its installation and features **here.**

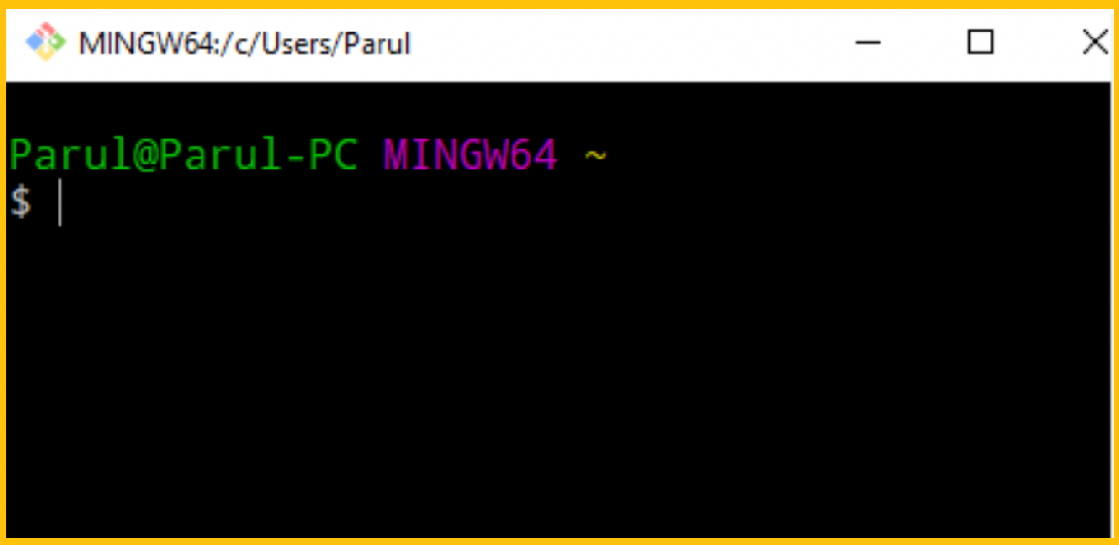
# MODULE 6
## BASH FUNDAMENTALS

## INSTALLATION ON WINDOWS CONT.

- Git Bash

  Git Bash is something that we will use for this article. Download the Git on your Windows computer from **here** and install it with all the default settings.

.**What you get in the end is a terminal window, something like the one below.**




A Windows Git Bash

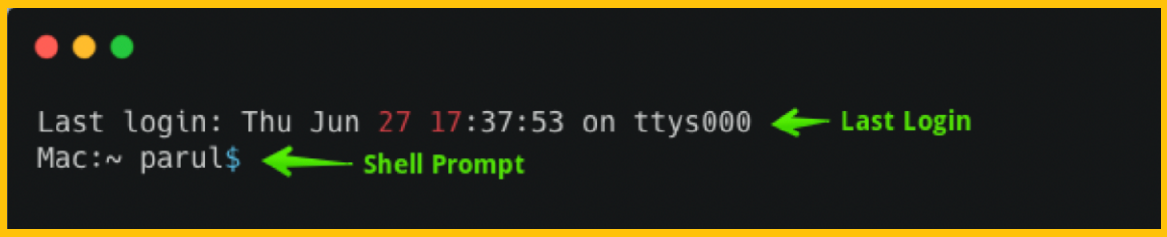
## EXPLORING THE TERMINAL

Whenever we open a terminal window, we see our last login credentials and a Shell prompt. The Shell prompt appears whenever the shell is ready to accept the input. It might vary a little in appearance depending upon the distribution, but mostly it appears as **username@machinename** followed by a $ sign.

# MODULE 6
## BASH FUNDAMENTALS

## EXPLORING THE TERMINAL CONT.

In case you do not want this whole bunch of information, you can use PS1 to customize your shell prompt.




The terminal will now only show the $ at the prompt. However, this is only temporary and will reset to its original settings, once the terminal is closed.

## GETTING STARTED

**To get a little hang of the bash, let's try a few simple commands:**

- **echo**: returns whatever you type at the shell prompt similar to Print in Python.


- **date**: displays the current time and date.

# MODULE 6
## BASH FUNDAMENTALS

## GETTING STARTED CONT.

- **cal**: displays a calendar of the current month.


- **Clearing the Terminal**: Ctrl-L or clear clears the terminal

## BASIC BASH COMMANDS

A bash command is the smallest unit of code that bash can independently execute. These commands tell bash what we need it to do. Bash generally takes in a single command from a user and returns to the user once the command has been executed.

### The Working Directory

- **pwd**

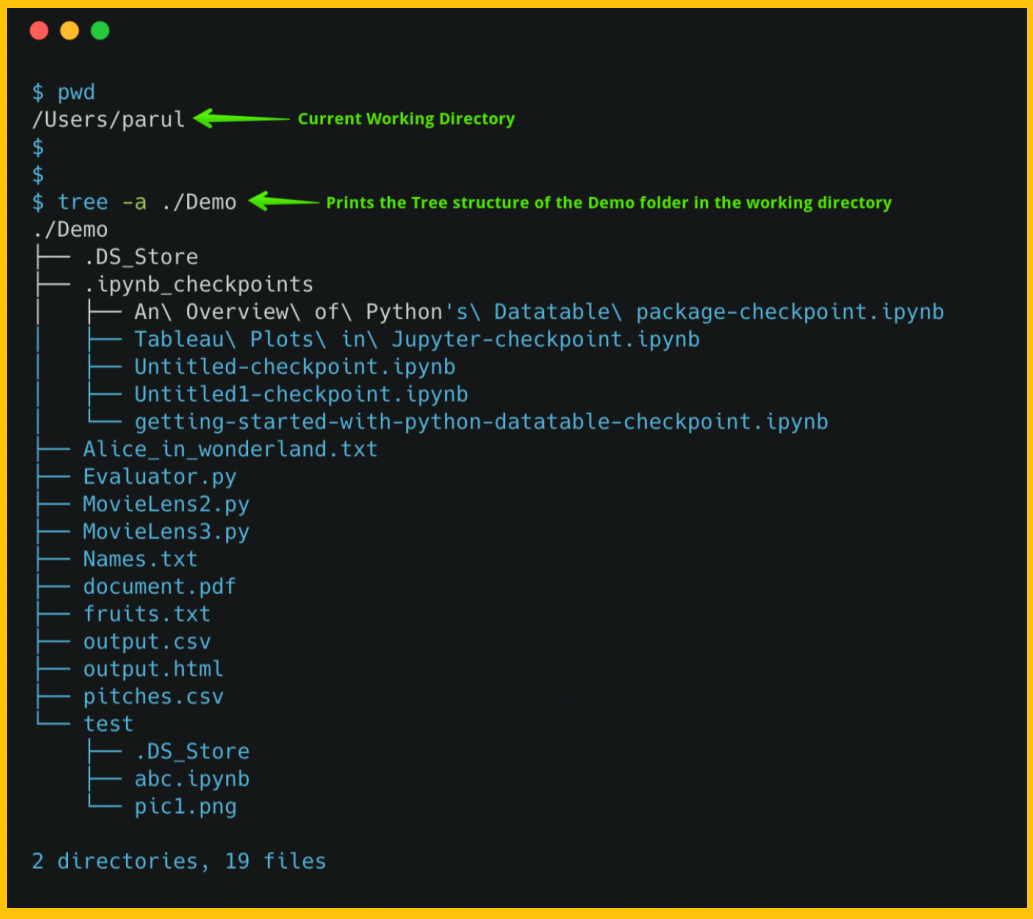
  pwd stands for print working directory and it points to the current working directory, that is, the directory that the shell is currently looking at. It's also the default place where the shell commands will look for data files.

  A directory is similar to a folder, but in the Shell, we shall stick to the name, directory. The UNIX file hierarchy has a tree structure. To reach a particular folder or file, we need to traverse certain paths within this tree structure. Paths separate every node of the above structure with the help of a slash( / ) character.

# MODULE 6
## BASH FUNDAMENTALS

## BASIC BASH COMMANDS CONT.

- **pwd**




### Navigating Directories

Commands like **ls** and **cd** are used to navigate and organize the files.

- **ls**

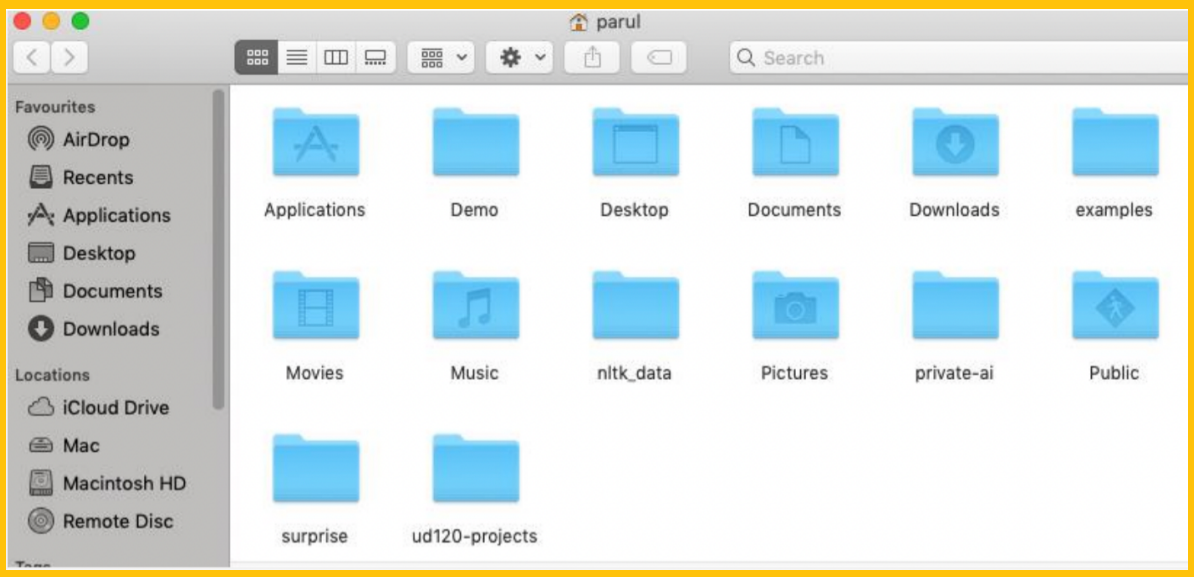
  ls stands for a list and it lists the contents of a directory. ls usually starts out looking at our home directory. This means if we print ls by itself, it will always print the contents of the current directory which in my case is /Users/parul.

# MODULE 6
## BASH FUNDAMENTALS

### BASIC BASH COMMANDS CONT.

- ls

```
$ pwd
/Users/parul
$ ls
Applications    Documents   Movies      Public      private-ai
Demo            Downloads   Music       examples    surprise
Desktop         Library     Pictures    nltk_data   ud120-projects
```




My Home directory represented in the shell and the GUI interface.

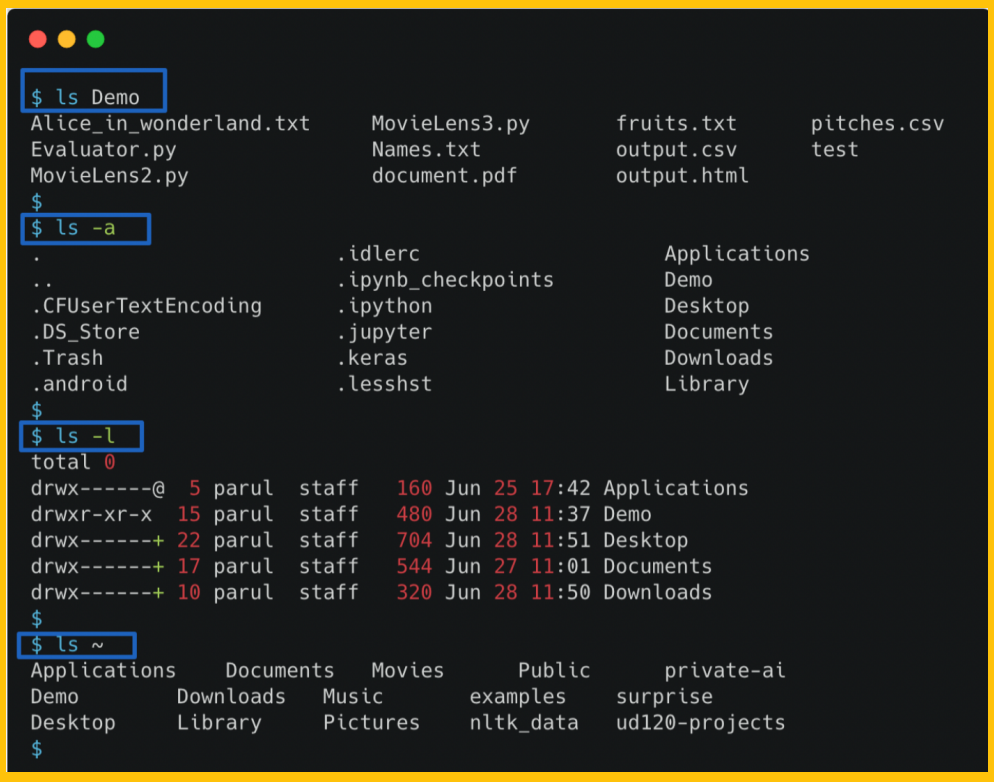
# MODULE 6

## BASH FUNDAMENTALS

### PARAMETERS

The parameters and options turn on some special features when used with the **ls** command.

- ls <folder> : to see the contents of a particular folder.
- ls -a: For listing all the hidden files in a folder
- ls -l: Prints out a longer and more detailed listing of the files. ls -l can also be used with the name of the Directory to list the files of that particular directory.
- ls ~: tilde(~) is a shortcut which denotes the home directory. So, regardless of what directory we are into, ls ~ will always list the home directory.




### WILDCARD

The shell also lets us match filenames with patterns, denoted by an asterisk(*). It serves as a wildcard to replace any other character within a given pattern. For example, if we list *.txt, it will list all the files with a .txt extension. Let's try and list out all the .py files in our Demo folder:

# MODULE 6
## BASH FUNDAMENTALS

## CD

cd stands for Change Directory and changes the active directory to the path specified. After we cd into a directory, **ls** command can be used to see the contents of that directory.

**Let's see some of the ways in which this command can be used:**

- **cd <Directory>**: changes the current directory to the desired Directory. Let's navigate to the test directory which lies within the Demo directory and see the contents of it with the ls command. Note that we can also use a semicolon(;) to write two commands on the same line.

```
ls -l Demo/.py
-rwxrwxrwx@ 1 parul  staff  3834 May  9  2018 Demo/Evaluator.py
-rwxrwxrwx@ 1 parul  staff  5787 Jul 18  2018 Demo/MovieLens2.py
-rwxrwxrwx@ 1 parul  staff  6838 Jul 18  2018 Demo/MovieLens3.py
```

- **cd ..** : To go back to the parent directory.
- **cd :** To go back to the home directory

## ORGANIZING FILES

There are certain commands which let us move, remove, create, and copy files from within the shell itself.

- **mkdir**

  mkdir stands for Make directory and is used to make a new directory or a folder.

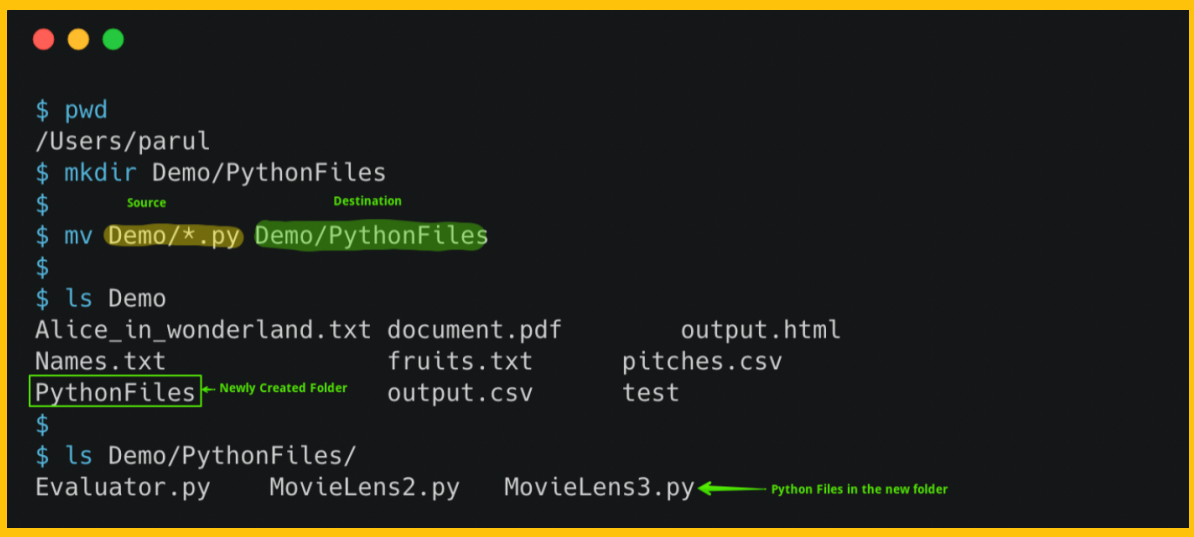
- **mv**

  mv stands for Move and it moves one or more files or directories from one place to another. We need to specify what we want to move, i.e., the source and where we want to move them, i.e., the destination.

# MODULE 6
## BASH FUNDAMENTALS

## ORGANIZING FILES CONT.

Let's create a new directory in the Demo Folder called PythonFiles and move all the .py files from the Demo folder into it using the above two commands.




- **touch**

  The touch command is used to create new, empty files. It is also used to change the timestamps on existing files and directories. Here is how we can create a file called foo.txt in the Demo folder.


- **rm**

  rm stands for Remove and it removes files or directories. By default, it does not remove directories, but if used as rm -r * within a directory, then every directory and file inside that directory is deleted.

# MODULE 6
## BASH FUNDAMENTALS

## ORGANIZING FILES CONT.

- **rm**

Let's now remove the previously created foo.txt file.

```
$ rm foo.txt
$ ls
Alice_in_wonderland.txt        document.pdf      output.html
Names.txt                      fruits.txt        pitches.csv
PythonFiles                    output.csv        test
```

- **rmdir**

  rmdir stands for remove directory and is used to remove empty directories from the filesystem. Let's delete the PythonFiles folder that we created a while ago.

```
$ pwd
/Users/parul
$ cd Demo/PythonFiles/
$ ls
Evaluator.py    MovieLens2.py    MovieLens3.py
$ rm *          <-- Removes all the Files
$ rmdir ../PythonFiles/    <-- Removes 'PythonFiles' Directory
```

Note that ../ denotes the parent directory.

## VIEWING FILES

This is another aspect of the shell, which is super useful. There are commands which help us to view the contents of a file so that we can then manipulate them.

- **cat**

  cat stands for concatenate and it reads a file and outputs its content. It can read any number of files, and hence the name concatenate. There are some text files in our Demo folder and let's use cat to view their content.

# MODULE 6
## BASH FUNDAMENTALS

## VIEWING FILES CONT.

- **cat**

```
$ pwd
/Users/parul
$ cd Demo
$ ls
Names.txt   babynames.txt   fruits.txt  output.html
PythonFIles document.pdf    output.csv  pitches.csv
$
$ cat fruits.txt
Peaches
Apples
Strawberries
Grapes
Oranges
Apples
Peaches
Melon
```

To view more than one file, mention both the filenames after the cat command:
$ cat Names.txt fruits.txt

- **less**

  The cat command displays the contents of a file on the screen. This is fine when the contents are less but becomes a problem when the file is big. As can be seen in the example below, the command pops out everything at the terminal at a very high speed, and we cannot make sense of all the contents of the file. Fortunately, there is a command called less which lets us view the contents, one screen at a time. $ less babynames.txt


**There are certain options with less that can be used:**

- Spacebar: To go to the next screen
- b: to go to the previous screen
- /: to search for a specific word
- q: quit

# MODULE 6

## BASH FUNDAMENTALS

## VIEWING FILES CONT.

- **man**

  The man command displays the man pages which are a user manual built default into many Linux and most Unix operating systems.

  man bash: To display the entire manual

  man <keyword> eg man ls gives information about the ls command.

```
LS(1)                    BSD General Commands Manual                    LS(1)

NAME
     ls -- list directory contents

SYNOPSIS
     ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls displays
     its name as well as any requested, associated information.  For each operand
     that names a file of type directory, ls displays the names of files contained
     within that directory, as well as any requested, associated information.

     If no operands are given, the contents of the current directory are displayed.
     If more than one operand is given, non-directory operands are displayed first;
     directory and non-directory operands are sorted separately and in lexicographi-
     cal order.

     The following options are available:

     -@      Display extended attribute keys and sizes in long (-l) output.

     -1      (The numeric digit ``one''.)  Force output to be one entry per line.
             This is the default when output is not to a terminal.

     -A      List all entries except for . and ...  Always set for the super-user.
```

# MODULE 6
## BASH FUNDAMENTALS

## PIPELINES AND FILTERS

The pipe operator '|' (vertical bar), is a way to send the output of one command as an input to another command.

**command1 | command2**

When a command sends its output to a pipe, the receiving end for that output is another command, not a file. The figure below shows how the wc command counts the contents of a file which have been displayed by the cat command.


In a way, wc is a command that takes in inputs and transforms those inputs in some way. Such commands are called filters and are placed after the Unix pipe.

### Filters

Let's now look at some of the commonly used filter commands. We shall be working with a file called **babynames.txt** that contains around 1000 baby names and a **fruits.txt** file that contains names of few fruits.

- **grep** or global regular expression print searches for lines with a given string or looks for a pattern in a given input stream. The following command will read all the files and output all the lines that contain either the word 'Tom.'


But this is a vast list, and we cannot possibly make sense of all these data just blasted at the terminal. Let's see how we can use the pipe operator to make sense out of it.

# MODULE 6
## BASH FUNDAMENTALS

## PIPELINES AND FILTERS CONT.

**Filters**

- **wc** is short for word count. It reads a list of files and generates one or more of the following statistics: newline count, word count, and byte count. Let's input the output of the above grep command to wc to count the number of lines that contain the word 'Tom.'

```
$grep Tom babynames.txt | wc
      50     250    1096
$
$grep Tom babynames.txt | wc -l   <-- No of lines containing the pattern 'Tom'
      50
$
$grep Tom babynames.txt | wc -w   <-- No of total words
     250
$
$grep Tom babynames.txt | wc -m   <-- Total Characters
    1096
$
```

- **Sort filter** sorts lines alphabetically or numerically

```
$cat fruits.txt | sort
Apples
Apples
Grapes
Melon
Oranges
Peaches
Peaches
Strawberries
```

The cat command first reads the contents of the file fruits.txt and then sorts it.

# MODULE 6
## BASH FUNDAMENTALS

## PIPELINES AND FILTERS CONT.

**Filters**

- **uniq** stands for unique and gives us the number of unique lines in the input tream.

```
$cat fruits.txt | sort | uniq
Apples
Grapes
Melon
Oranges
Peaches
Strawberries
```

It is important to note that uniq cannot detect duplicate entries unless they are adjacent. Hence we have used sorted the file before using the sort command. Alternatively, you can also use sort -u instead of uniq.

```
$cat fruits.txt | sort -u
Apples
Grapes
Melon
Oranges
Peaches
Strawberries
```

Pipelines come in very handy for performing some complex tasks as several of the commands can be put together into a pipeline.