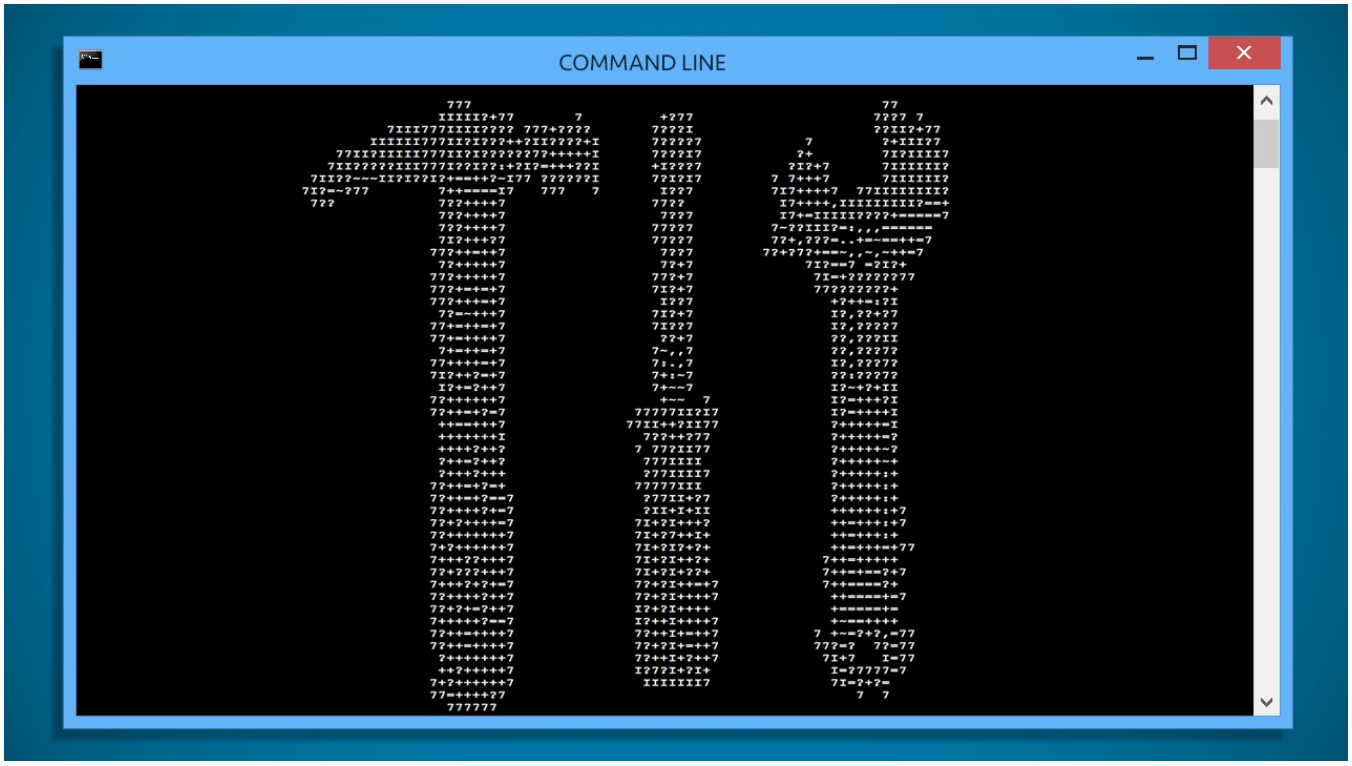
Basics of BASH for Beginners.

Learn about some of the most useful BASH commands and the utility they offer.

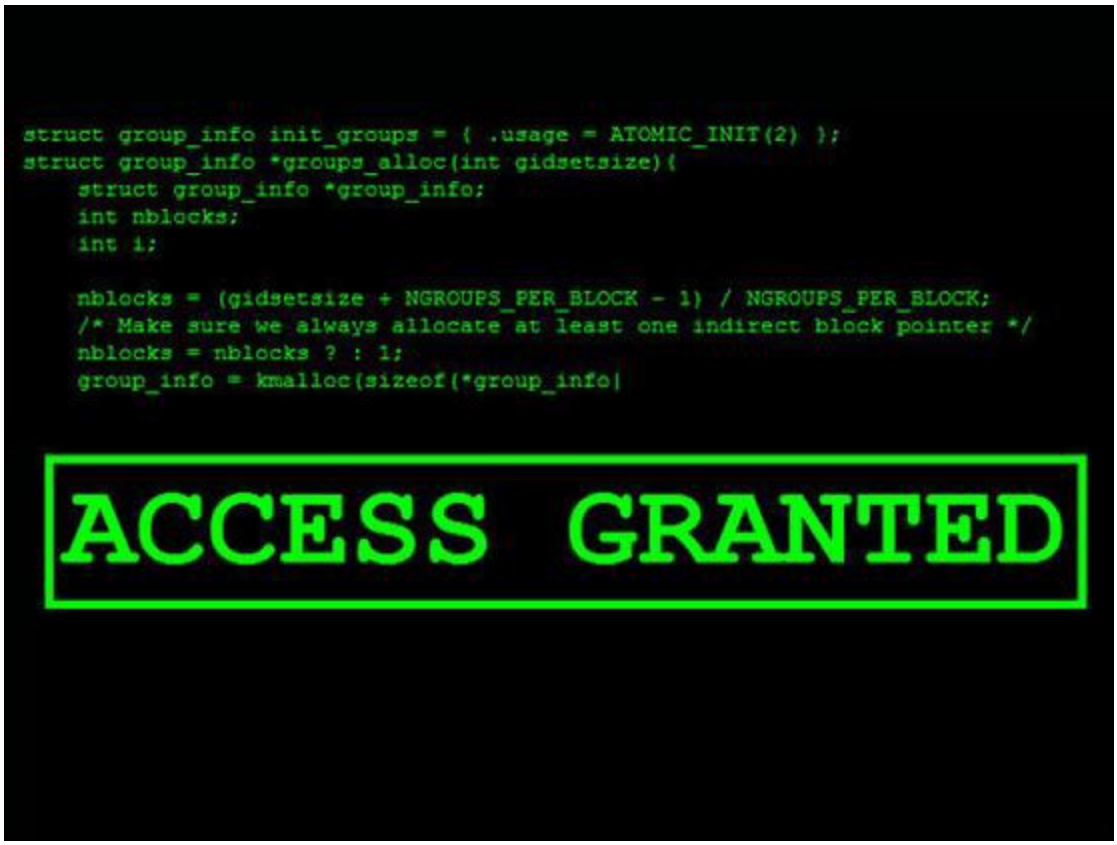
Parul Pandey Follow
Jun 29 · 11 min read



Source

> Most computers today are not powered by electricity. They instead seem to be powered by the “pumping” motion of the mouse: William Shotts

Have you ever noticed how a super nerdy hacker in movies, can easily infiltrate the most secure banks and rob them all off by merely typing some commands ferociously and staring at the green screen with a black background? How the person consistently gets access to all the passwords and takes control of the hidden cameras, anywhere, with just a few strokes on the keyboard. Well, I am not sure how the movie makers got this, but I guess it is their way of telling us that the **command line** is a powerful tool, albeit without all these hacking and ACCESS GRANTED!! absurdity.



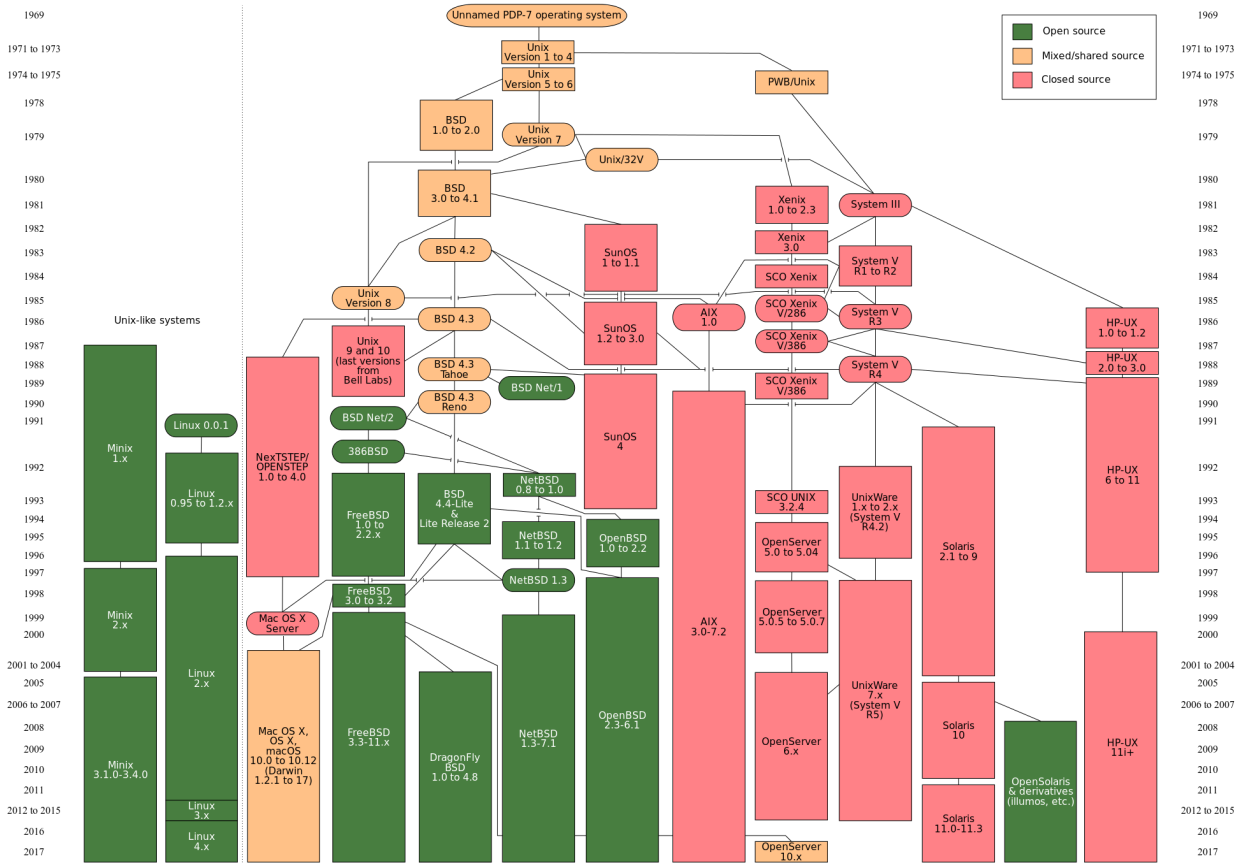
A lot of times, beginners are so used to working with the GUI based interface, that they tend to overlook the capabilities of the command line interface(CLI). A mouse comes in real handy when we need to copy about a hundred thousand files into a folder, but what if were to rename all those thousand files 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.



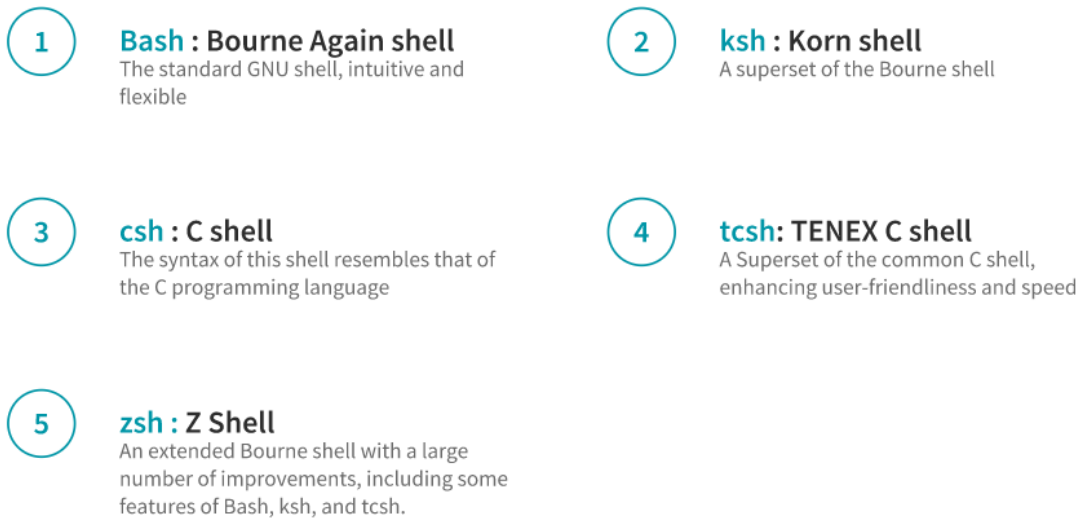
The UNIX Family Tree. Source: WIkipedia

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.

Shell types

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



Common Shell Types

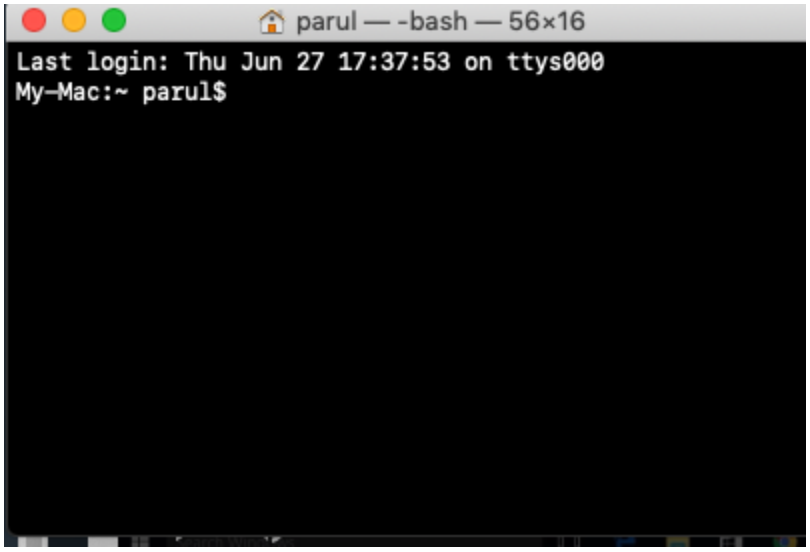
We shall, however, limit ourselves to the Bash shell in this article.

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. 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.

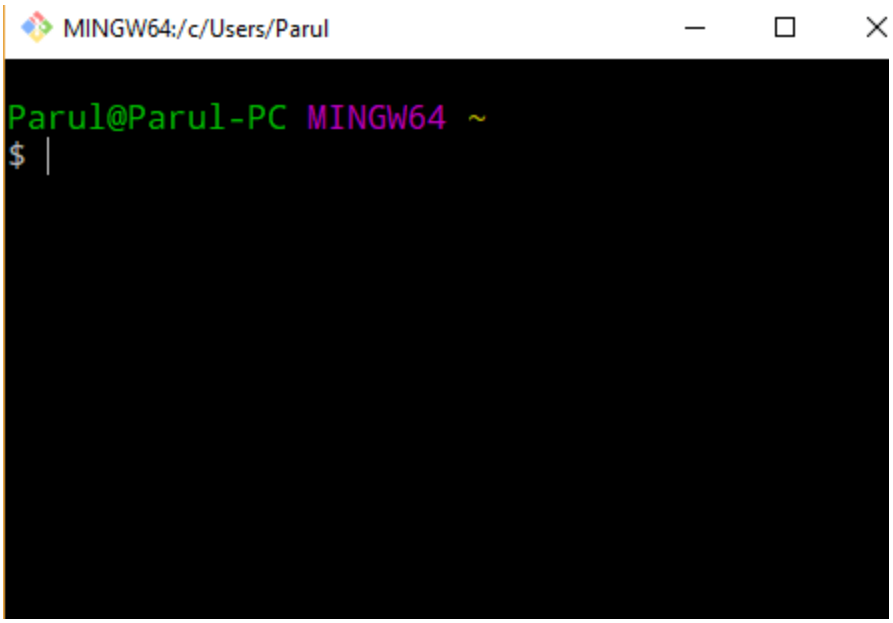
Installation on WINDOWS

- **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.

- **Git Bash**

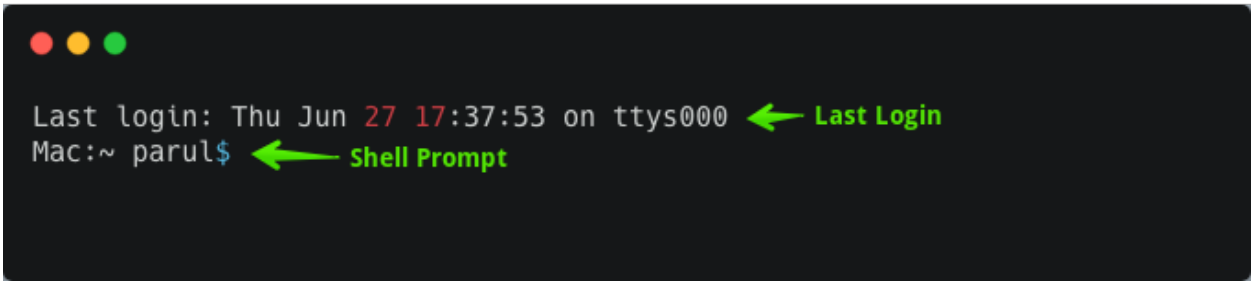
Git Bash is something that we will use for this article. Download the Git on your Windows computer from here and install it with all the default settings. What you get in the end is a terminal window, something like the one below.



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.



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.

```
$date
Sat Jun 29 19:01:43 IST 2019
$
```

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

```
$cal
     June 2019
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
$
```

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

. . .

Basic Bash Commands

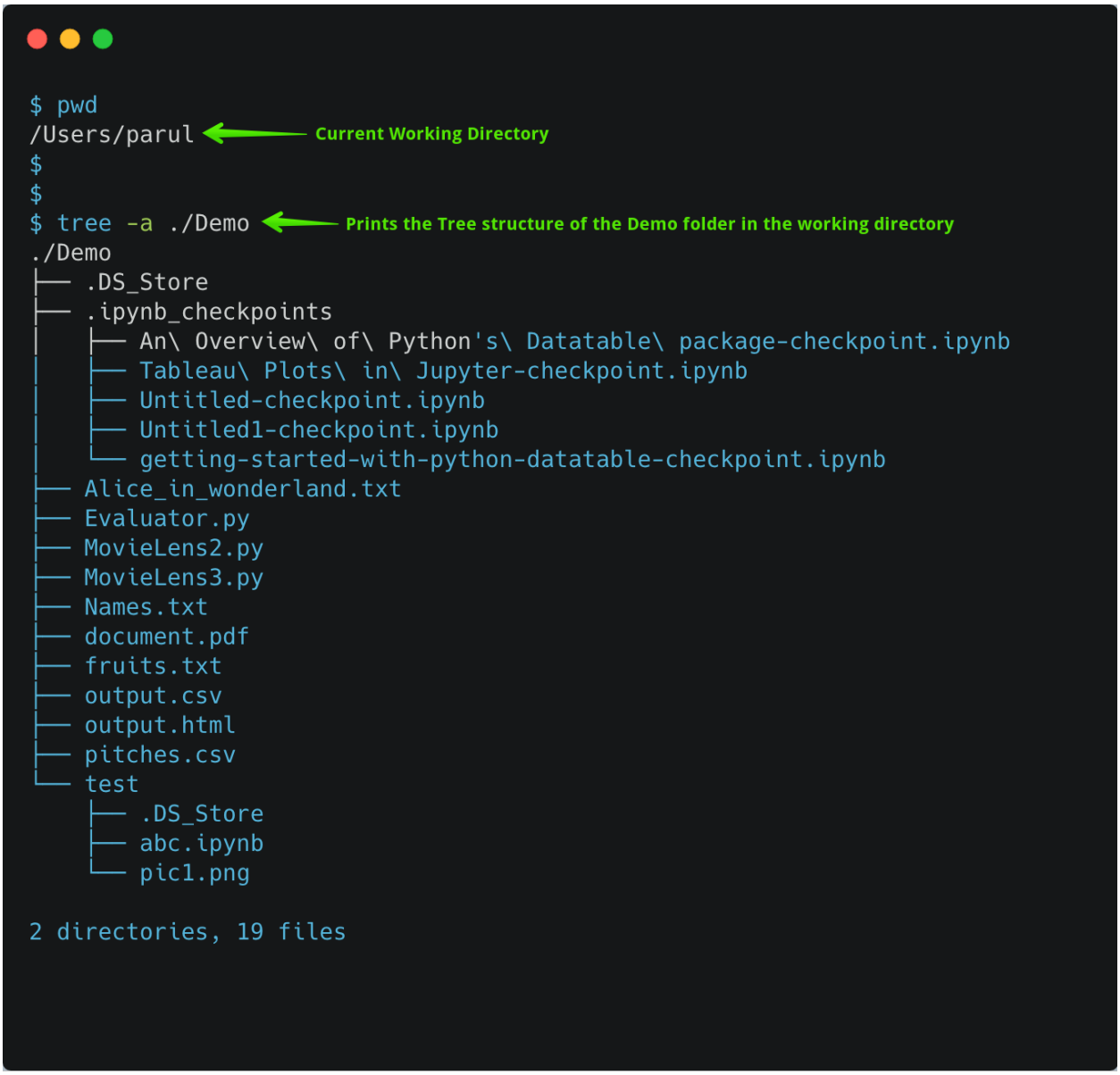
A bash command is the smallest unit of code that bash can independently execute. These commands tell bash what we need it to do. Bash generally takes in a single command from a user and returns to the user once the command has been executed.

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.



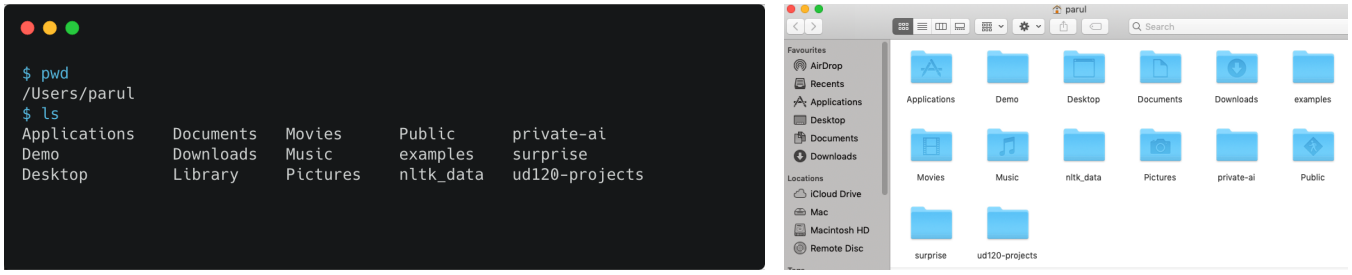
. . .

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`.



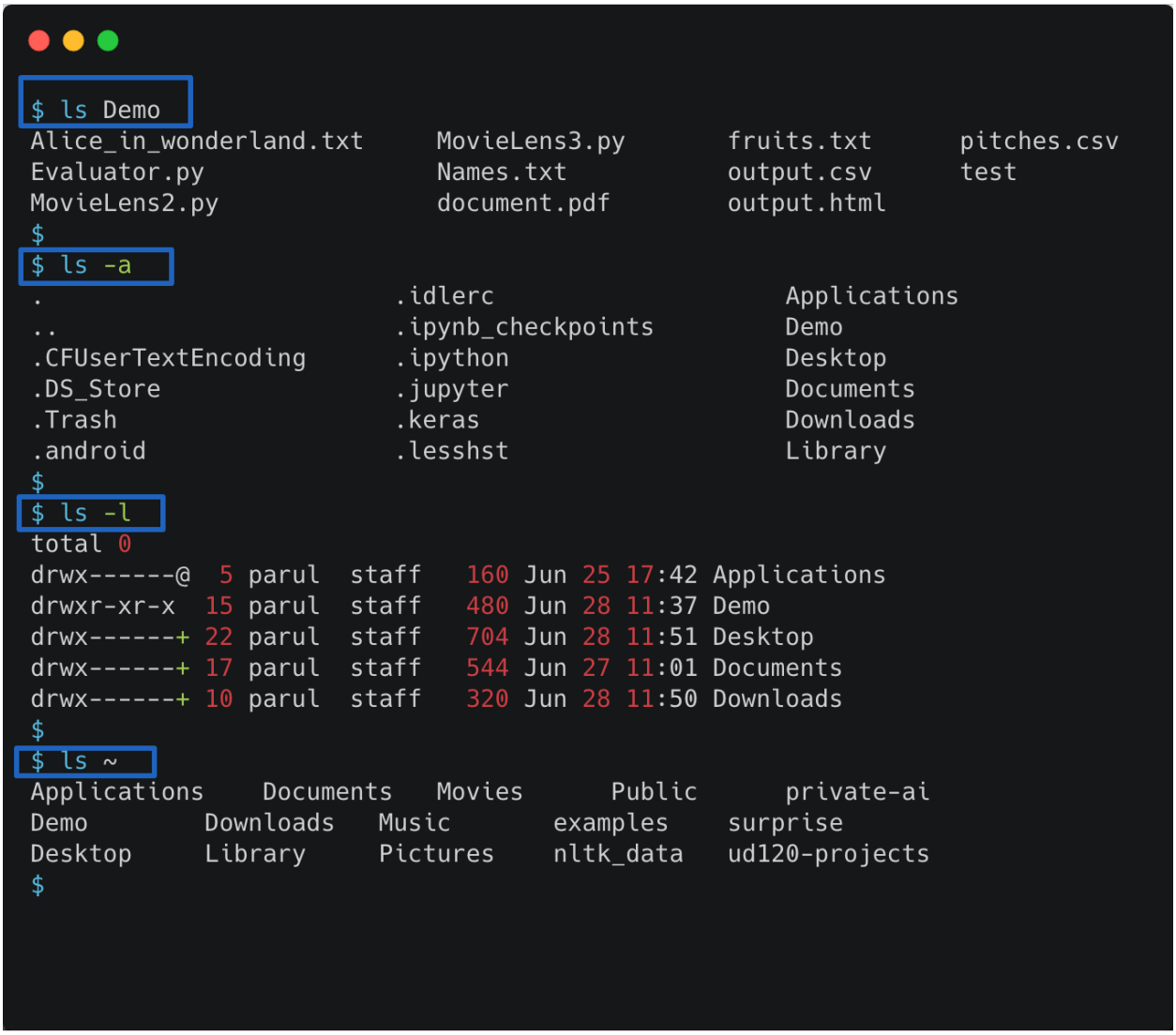
My Home directory represented in the shell and the GUI interface.

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 short cut which denotes the home directory. So, regardless of what directory we are into, `ls ~` will always list

the home directory.



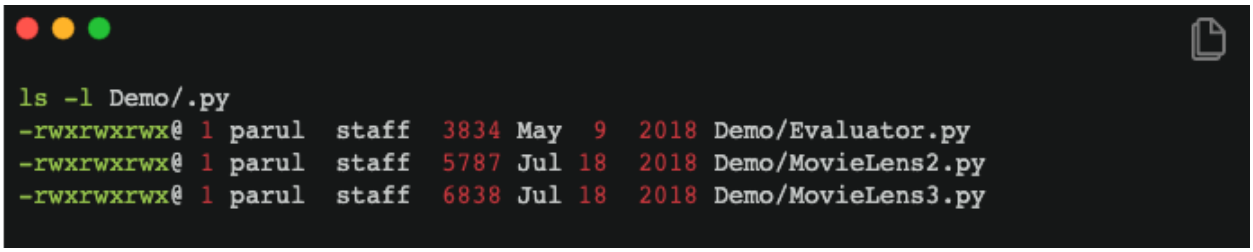
Wildcard

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:

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.



- **`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

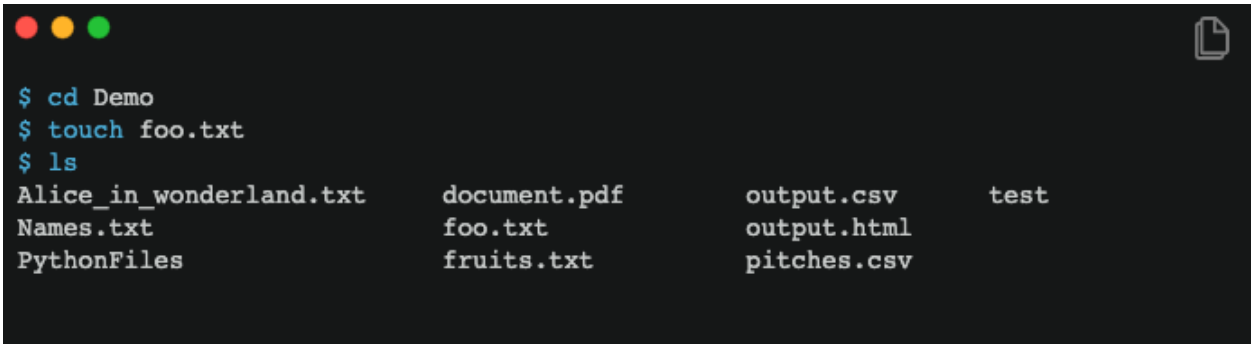
`mv` stands for `Move` and it moves one or more files or directories from one place to another. We need to specify what we want to move, i.e., the source and where we want to move them, i.e., the destination.

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

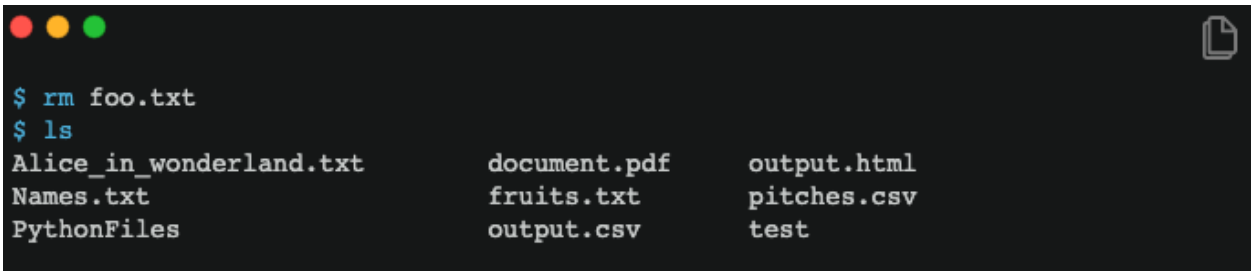
to change the timestamps on existing files and directories. Here is how we can create a file called `foo.txt` in the Demo folder.



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.

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



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.

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.

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

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.
```

. . .

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
```

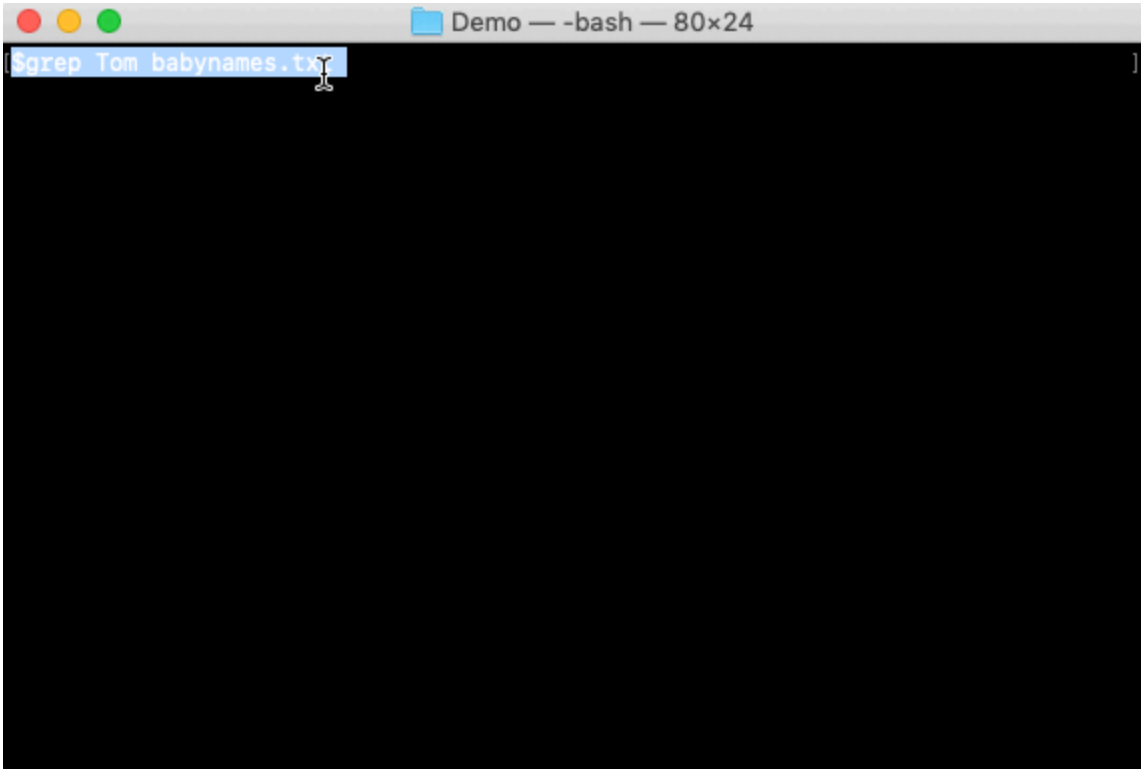
When a command sends its output to a pipe, the receiving end for that output is another command, not a file. The figure below shows how the `wc` command count the contents of a file which have been displayed by the `cat` command.

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

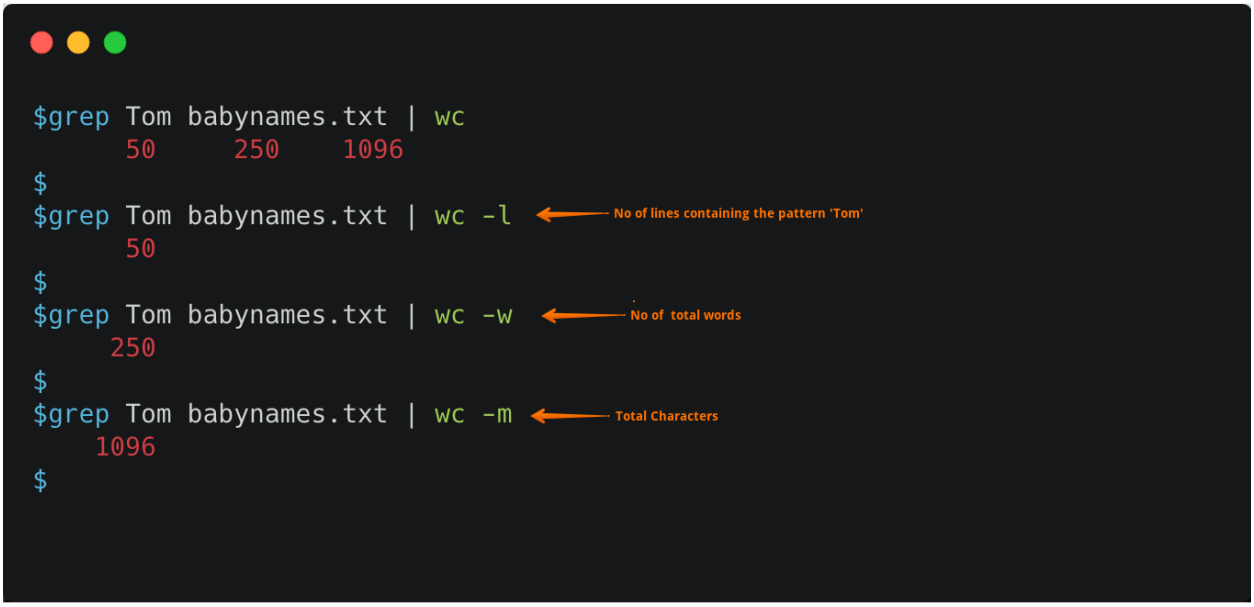
Let's now look at some of the commonly used filter commands. We shall be working with a file called **babynames.txt** that contains around 1000 baby names and a **fruits.txt** file that contains names of few fruits.

- **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.

- **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.'



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

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

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

```
$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.

. . .

Further Reading

Command Line tools can be a great addition to the toolkit. Initially, it

can be intimidating for beginners, but once they get the hang of it, its real advantages and benefits can be realized. This article is to make one started and only scratches the surface. There are a plethora of useful resources available online to explore and learn about the shell in details. Here are some of the recommended resources:

- The Linux Command Line Fifth Internet Edition William Shotts
- Bash Beginners Guide
- The Bash Academy