Module 1

Unix Shell Programming and GIT

Introduction to Linux Shell and Shell Scripting

If we are using any major operating system, we are indirectly interacting with the **shell**. While running Ubuntu, Linux Mint, or any other Linux distribution, we are interacting with the shell by using the terminal.

What is Kernel?

The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system. It manages the following resources of the Linux system –

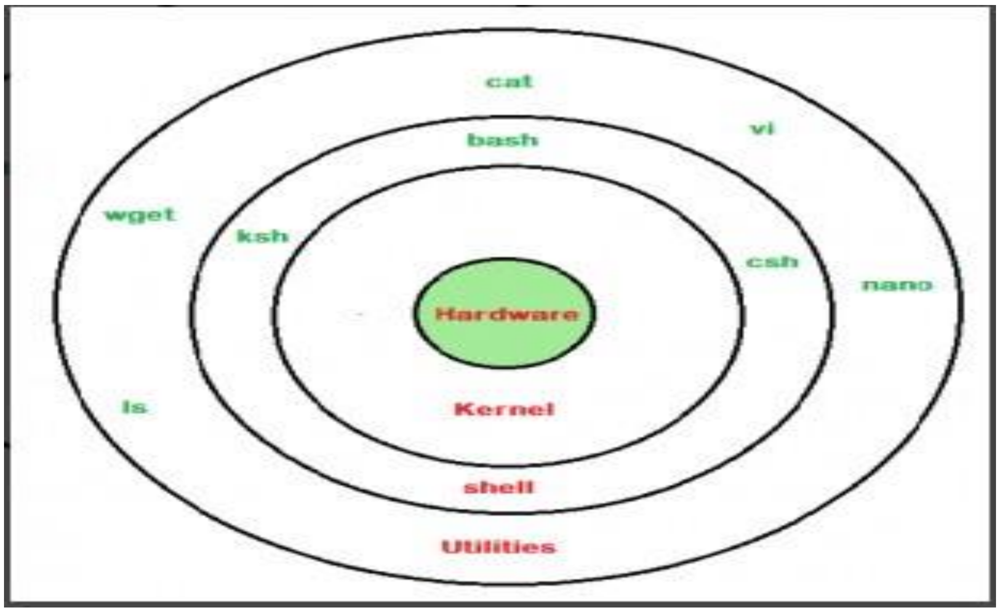
- File management
- Process management
- I/O management
- Memory management
- Device management etc.

It is often mistaken that Linus Torvalds has developed Linux OS, but actually, he is only responsible for the development of the Linux kernel.

Complete Linux system = Kernel + GNU system utilities and libraries + other management scripts + installation scripts.

What is Shell?

A shell is a special user program that provides an interface for the user to use operating system services. Shell accepts human-readable commands from users and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.

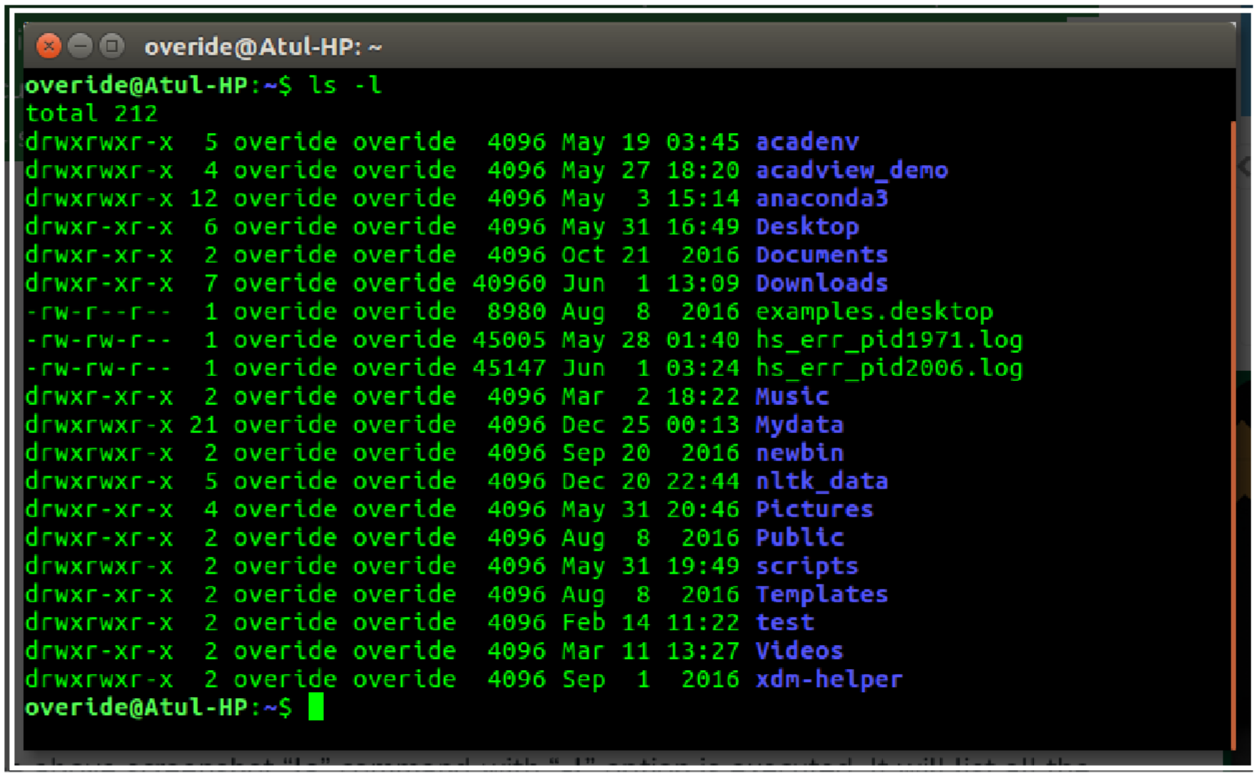


Shell is broadly classified into two categories –

- Command Line Shell
- Graphical shell

Command Line Shell

Shell can be accessed by users using a command line interface. A special program called Terminal in Linux/macOS, or Command Prompt in Windows OS is provided to type in the human-readable commands such as “cat”, “ls” etc. and then it is being executed. The result is then displayed on the terminal to the user. A terminal in Ubuntu 16.4 system looks like this –



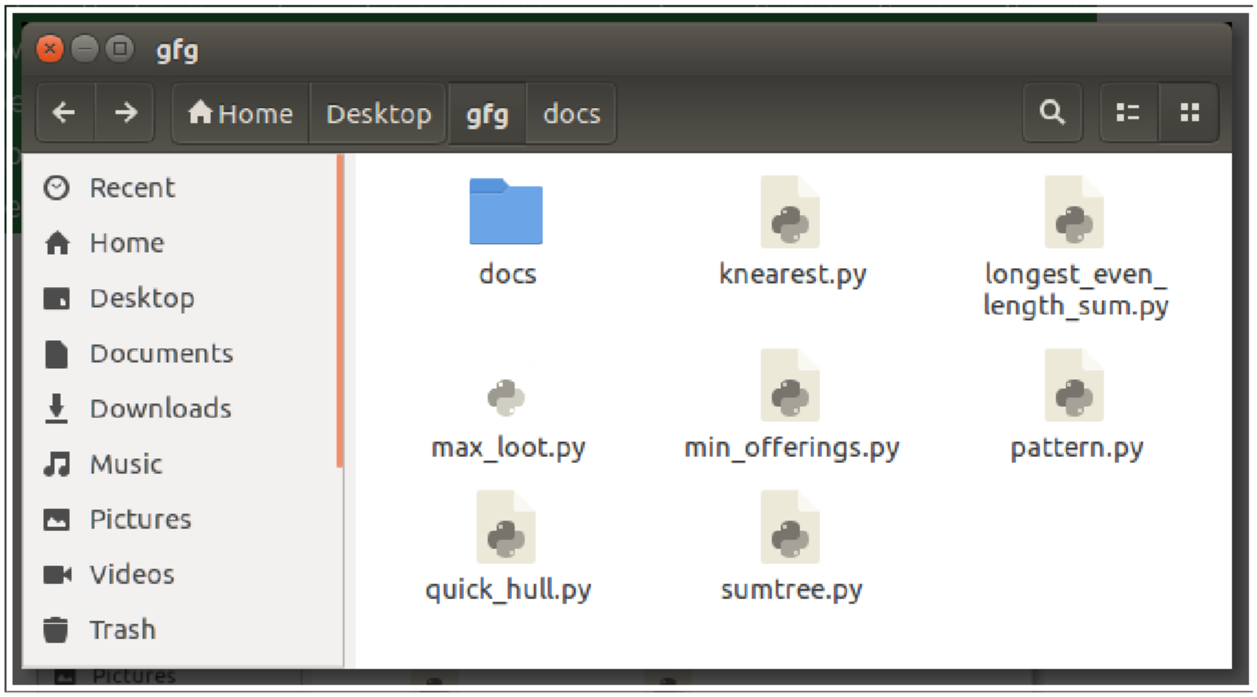
```
overide@Atul-HP:~$ ls -l
total 212
drwxrwxr-x  5 overide overide  4096 May 19 03:45 acadenv
drwxrwxr-x  4 overide overide  4096 May 27 18:20 acadview_demo
drwxrwxr-x 12 overide overide  4096 May  3 15:14 anaconda3
drwxr-xr-x  6 overide overide  4096 May 31 16:49 Desktop
drwxr-xr-x  2 overide overide  4096 Oct 21  2016 Documents
drwxr-xr-x  7 overide overide 40960 Jun  1 13:09 Downloads
-rw-r--r--  1 overide overide  8980 Aug  8  2016 examples.desktop
-rw-rw-r--  1 overide overide 45005 May 28 01:40 hs_err_pid1971.log
-rw-rw-r--  1 overide overide 45147 Jun  1 03:24 hs_err_pid2006.log
drwxr-xr-x  2 overide overide  4096 Mar  2 18:22 Music
drwxrwxr-x 21 overide overide  4096 Dec 25 00:13 Mydata
drwxrwxr-x  2 overide overide  4096 Sep 20  2016 newbin
drwxrwxr-x  5 overide overide  4096 Dec 20 22:44 nltk_data
drwxr-xr-x  4 overide overide  4096 May 31 20:46 Pictures
drwxr-xr-x  2 overide overide  4096 Aug  8  2016 Public
drwxrwxr-x  2 overide overide  4096 May 31 19:49 scripts
drwxr-xr-x  2 overide overide  4096 Aug  8  2016 Templates
drwxrwxr-x  2 overide overide  4096 Feb 14 11:22 test
drwxr-xr-x  2 overide overide  4096 Mar 11 13:27 Videos
drwxrwxr-x  2 overide overide  4096 Sep  1  2016 xdm-helper
overide@Atul-HP:~$ 
```

It will list all the files in the current working directory in a long listing format.

Working with a command line shell is a bit difficult for beginners because it’s hard to memorize so many commands. It is very powerful; it allows users to store commands in a file and execute them together. This way any repetitive task can be easily automated. These files are usually called batch files in Windows and **Shell** Scripts in Linux/macOS systems.

Graphical Shells

Graphical shells provide means for manipulating programs based on the graphical user interface (GUI), by allowing for operations such as opening, closing, moving, and resizing windows, as well as switching focus between windows. Windows OS or Ubuntu OS can be considered as a good example which provides GUI to the user for interacting with the program. Users do not need to type in commands for every action. A typical GUI in the Ubuntu system –



There are several shells are available for Linux systems like -

- BASH (Bourne Again SHell) – It is the most widely used shell in Linux systems. It is used as the default login shell in Linux systems and in macOS. It can also be installed on Windows OS.
- CSH (C SHell) – The C shell's syntax and its usage are very similar to the C programming language.
- KSH (Korn SHell) - The Korn Shell was also the base for the POSIX Shell standard specifications etc.

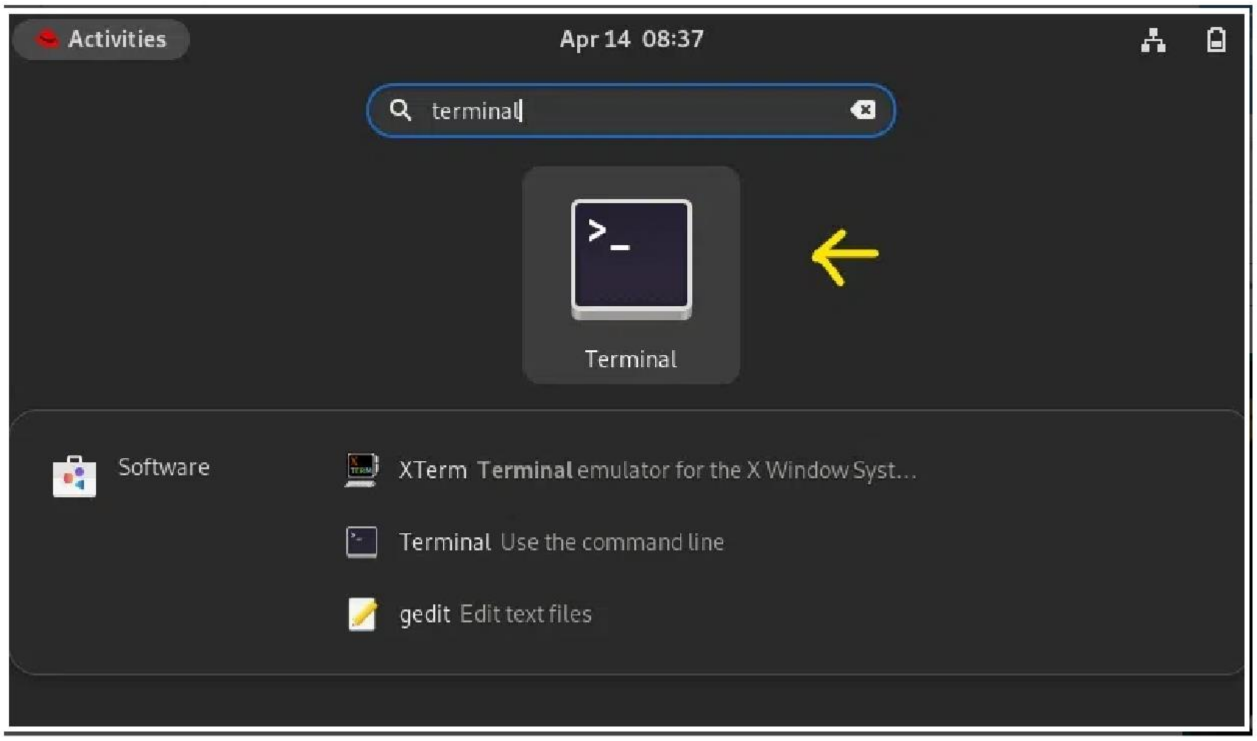
Each shell does the same job but understands different commands and provides different built-in functions.

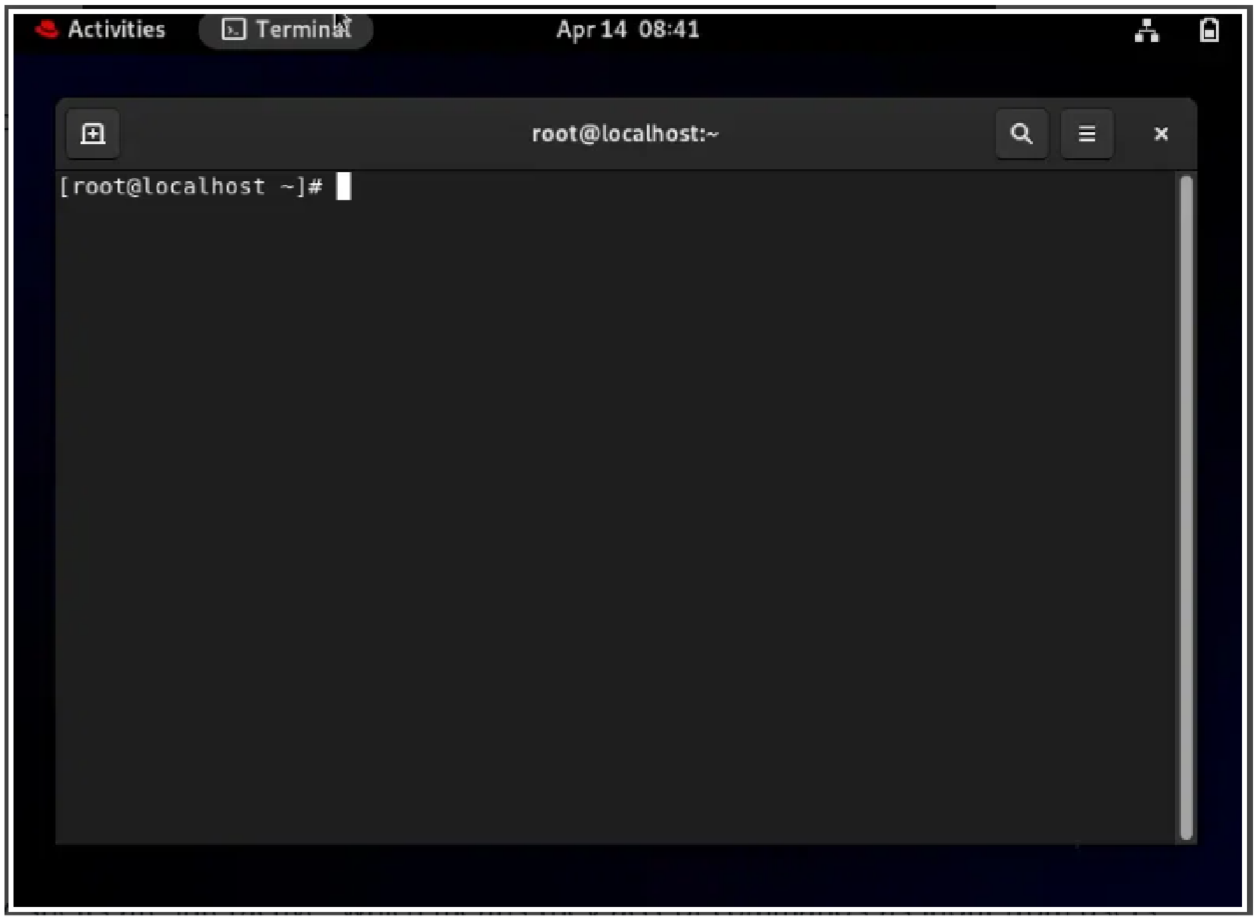
Each shell does the same job but understands different commands and provides different built-in functions.

What is a terminal?

A program which is responsible for providing an interface to a user so that he/she can access the shell. It basically allows users to enter commands and see the output of those commands in a text-based interface. Large scripts that are written to automate and perform complex tasks are executed in the terminal.

To access the terminal, simply search in search box "terminal" and double-click it


Activities
Apr 14 08:37
terminal
Terminal
Software
XTerm Terminal emulator for the X Window Syst...
Terminal Use the command line
gedit Edit text files



```
[root@localhost ~]#
```

Shell Scripting

Usually, shells are interactive, which means they accept commands as input from users and execute them. However, sometimes we want to execute a bunch of commands routinely, so we have to type in all commands each time in the terminal.

As a shell can also take commands as input from a file, we can write these commands in a file and can execute them in the shell to avoid this repetitive work. These files are called **Shell Scripts** or **Shell Programs**. Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with **`.sh`** file extension e.g., **myscript.sh.**

A shell script has syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. It would be very easy to get started with it.

A shell script comprises the following elements –

- Shell Keywords – if, else, break etc.
- Shell commands – cd, ls, echo, pwd, touch etc.
- Functions
- Control flow – if..then..else, case and shell loops etc.

Why do we need shell scripts?

There are many reasons to write shell scripts:

- To avoid repetitive work and automation
- System admins use shell scripting for routine backups.
- System monitoring
- Adding new functionality to the shell etc.

Some Advantages of shell scripts

- The command and syntax are exactly the same as those directly entered in the command line, so programmers do not need to switch to entirely different syntax
- Writing shell scripts are much quicker
- Quick start
- Interactive debugging etc.

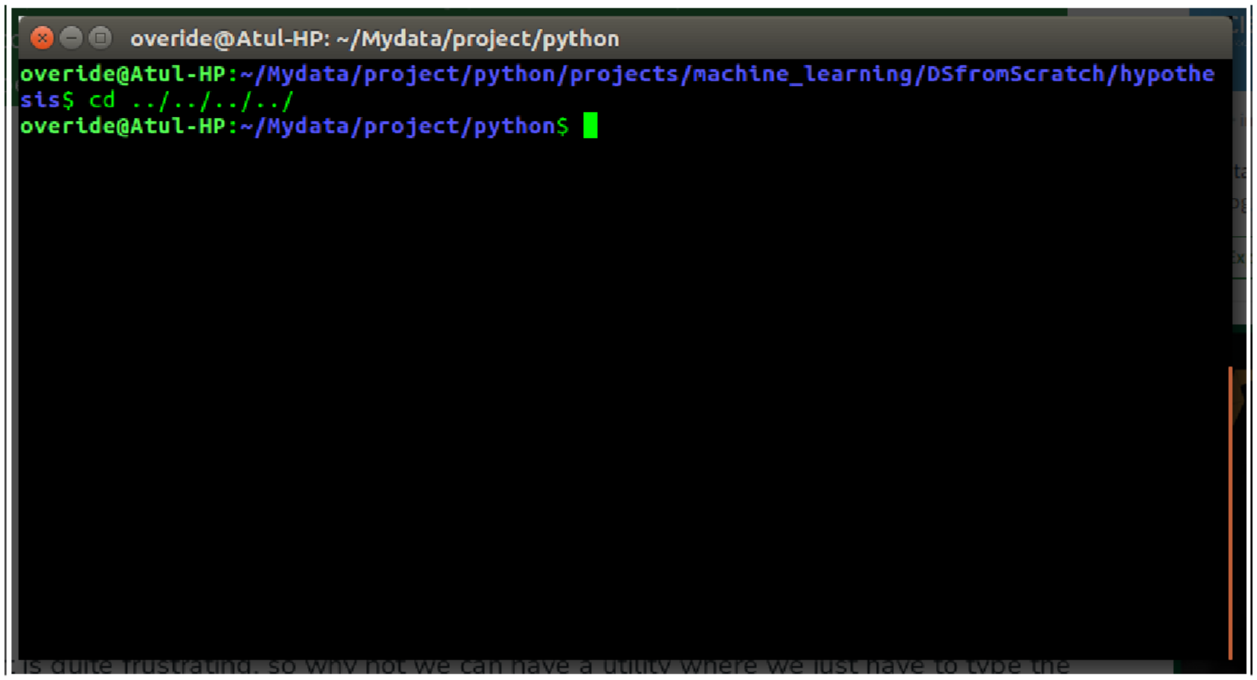
Some Disadvantages of shell scripts

- Prone to costly errors, a single mistake can change the command which might be harmful.
- Slow execution speed
- Design flaws within the language syntax or implementation
- Not well suited for large and complex task

- Provide minimal data structure unlike other scripting languages. etc.

Simple demo of shell scripting using Bash Shell

If you work on a terminal, something you traverse deep down in directories. Then for coming few directories up in path we have to execute a command like this as shown below to get to the "python" directory:



```
overide@Atul-HP: ~/Mydata/project/python
overide@Atul-HP:~/Mydata/project/python/projects/machine_learning/DSfromScratch/hypothe
sis$ cd ../../../../
overide@Atul-HP:~/Mydata/project/python$ 
```

Importance of shell scripting

Shell scripting is primarily used to automate repetitive system tasks, such as backing up files, monitoring system resources, and managing user accounts. By turning a series of commands into a script, system administrators can save time, increase accuracy, and simplify complex tasks.

Different types of shells

Types of Shell:

- **The C Shell –**

`Denoted as **csh**`

- Bill Joy created it at the University of California at Berkeley. It incorporated features such as aliases and command history. It includes helpful programming features like built-in arithmetic and C-like expression syntax. In C shell:

```
Command full-path name is /bin/csh,

Non-root user default prompt is hostname %,

Root user default prompt is hostname #.
```

- **The Bourne Shell –**

`Denoted as **sh**`

- It was written by Steve Bourne at AT&T Bell Labs. It is the original UNIX shell. It is faster and more preferred. It lacks features for interactive use like the ability to recall previous commands. It also lacks built-in arithmetic and logical expression handling. It is the default shell for Solaris OS. For the Bourne shell the:

```
Command full-path name is /bin/sh and /sbin/sh,

Non-root user default prompt is $,
Root user default prompt is #.
```

- **The Korn Shell**

```
It is denoted as ksh
```

- It was written by David Korn at AT&T Bell Labs. It is a superset of the Bourne shell. So it supports everything in the Bourne shell.It has interactive features. It includes features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities. It is faster than C shell. It is compatible with scripts written for C shell. For the Korn shell the:

```
Command full-path name is /bin/ksh,

Non-root user default prompt is $,

Root user default prompt is #.
```

What are the different types of shells available in Linux?

Linux offers a variety of shells, each with unique features and capabilities. Some of the most popular and widely used shells include:

1. ***Bash (Bourne-Again SHell)**: The most common default shell in Linux distributions. It’s an enhancement of the original Bourne shell (sh), incorporating features from other shells like ksh and csh.*

2. ***sh (Bourne Shell)**: The original shell that was used on UNIX. It is simple and fast but lacks many features of more modern shells.*
3. ***Dash (Debian Almquist Shell)**: A variant of the Almquist Shell (ash). Dash is used as the default `/bin/sh` on Debian-based systems because of its speed and compliance with POSIX standards.*
4. ***Ksh (Korn Shell)**: Offers many features, combining elements of both the Bourne shell and C shell. It provides powerful programming features as well as interactive use.*
5. ***Csh (C Shell)** and **Tcsh (TENEX C Shell)**: Csh offers a syntax that is quite similar to the C programming language, from which it derives its name. Tcsh is an improved version of csh that includes command line editing and completion.*
6. ***Zsh (Z Shell)**: Combines many of the useful features of Bash, ksh, and tcsh. It is known for its interactive use enhancements and extensive customization capabilities.*
7. ***Fish (Friendly Interactive SHell)**: Known for its user-friendly and interactive features, like syntax highlighting, autosuggestions, and tab completions.*

How to Create a Shell Script in linux

Shell is an interface of the operating system. It accepts commands from users and interprets them to the operating system. If you want to run a bunch of commands together, you can do so by creating a shell script. Shell scripts are very useful if you need to do a task routinely, like taking a backup. You can list those commands and execute them all with just a single script. Let’s see how you can create a shell script and run it on Linux.

Creating a Shell Script

Login to your Linux machine and open the terminal, navigate to the folder where you want to store the shell script. Shell scripts end with the extension “.sh”. Let’s create our first shell script. Type in

```
touch script.sh
```

Now, this script file is not executable by default, we have to give the executable permission to this file. Type in

```
chmod +x script.sh
```

Now, we will add some commands to this shell script. Open this shell script with any text editor of your choice (command-line based or GUI based) and add some commands. We will use nano. Type in

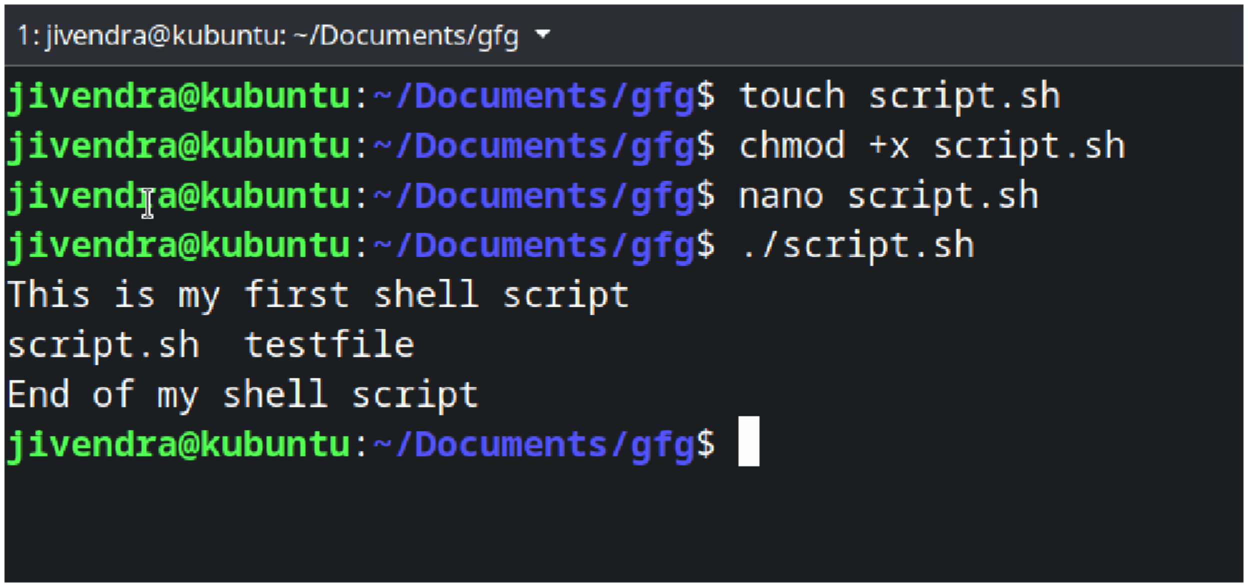
```
nano script.sh
```

Add the following commands to test this shell script

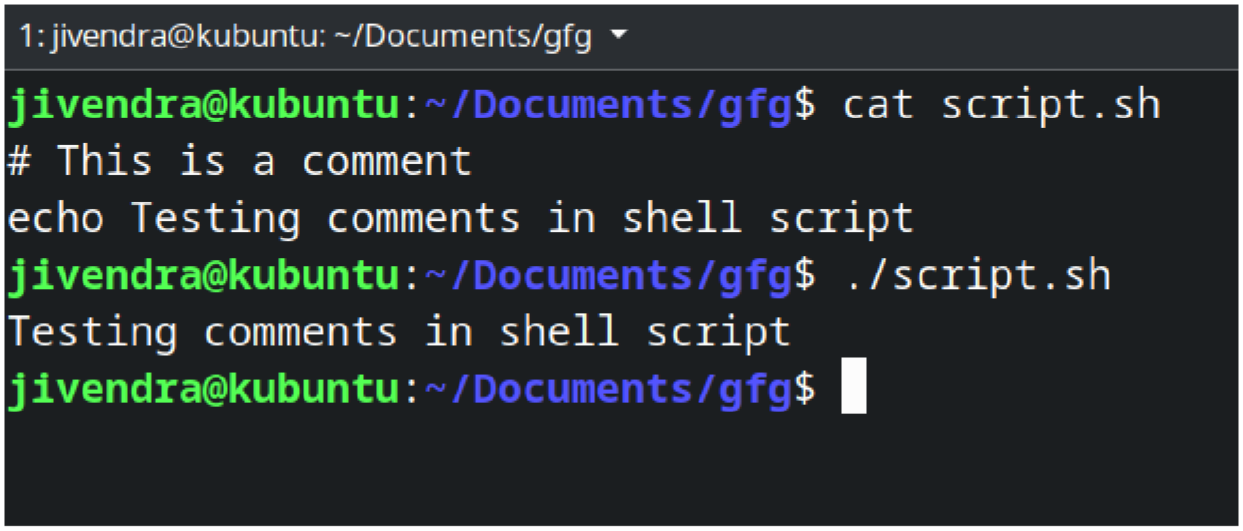
```
echo This is my first shell script
touch testfile
ls
echo End of my shell script
```

Save the changes, and run the shell script by typing in

```
./script.sh
```



```
1: jivendra@kubuntu: ~/Documents/gfg
jivendra@kubuntu:~/Documents/gfg$ touch script.sh
jivendra@kubuntu:~/Documents/gfg$ chmod +x script.sh
jivendra@kubuntu:~/Documents/gfg$ nano script.sh
jivendra@kubuntu:~/Documents/gfg$ ./script.sh
This is my first shell script
script.sh  testfile
End of my shell script
jivendra@kubuntu:~/Documents/gfg$
```



```
1: jivendra@kubuntu: ~/Documents/gfg
jivendra@kubuntu:~/Documents/gfg$ cat script.sh
# This is a comment
echo Testing comments in shell script
jivendra@kubuntu:~/Documents/gfg$ ./script.sh
Testing comments in shell script
jivendra@kubuntu:~/Documents/gfg$
```

Making shell scripting executable

Steps to write and execute a script

- Open the terminal. Go to the directory where you want to create your script.

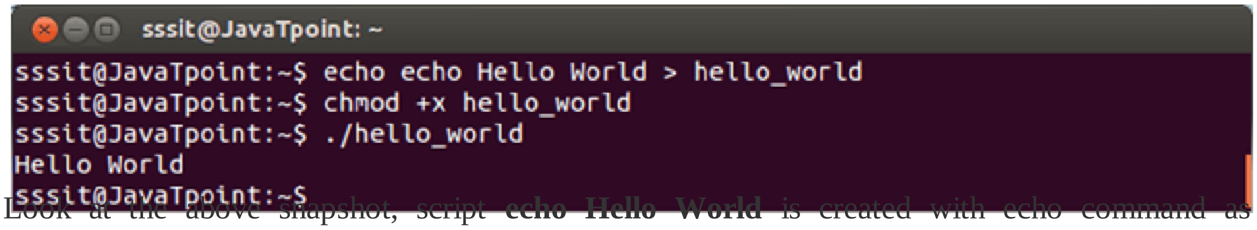
- Create a file with .sh extension.
- Write the script in the file using an editor.
- Make the script executable with command chmod +x <fileName>.
- Run the script using ./<fileName>.

Note: In the last step you have to mention the path of the script if your script is in other directory.

Hello World script

Here we'll write a simple programme for Hello World.

First of all, create a simple script in any editor or with echo. Then we'll make it executable with **chmod +x** command. To find the script you have to type the script path for the shell.



```
sssit@JavaTpoint:~$ echo echo Hello World > hello_world
sssit@JavaTpoint:~$ chmod +x hello_world
sssit@JavaTpoint:~$ ./hello_world
Hello World
sssit@JavaTpoint:~$
```

Look at the above snapshot, script **echo Hello World** is created with echo command as **hello_world.** Now command **chmod +x hello_world** is passed to make it executable. We have given the command **./hello_world** to mention the hello_world path. And output is displayed.

Shell Input & Output

Output Redirection

The output from a command normally intended for standard output can be easily diverted to a file instead. This capability is known as output redirection.

If the notation > file is appended to any command that normally writes its output to standard output, the output of that command will be written to file instead of your terminal.

Check the following **who** command which redirects the complete output of the command in the users file.

```
$ who > users
```

Notice that no output appears at the terminal. This is because the output has been redirected from the default standard output device (the terminal) into the specified file. You can check the users file for the complete content −

```
$ cat users

oko         tty01  Sep 12 07:30

ai          tty15  Sep 12 13:32

ruth        tty21   Sep 12 10:10

pat         tty24   Sep 12 13:07

steve       tty25  Sep 12 13:03

$
```

If a command has its output redirected to a file and the file already contains some data, that data will be lost. Consider the following example −

```
$ echo line 1 > users

$ cat users

line 1

$
```

You can use >> operator to append the output in an existing file as follows −

```
$ echo line 2 >> users

$ cat users

line 1

line 2

$
```

Input Redirection

Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. As the **greater-than character >** is used for output redirection, the **less-than character <** is used to redirect the input of a command.

The commands that normally take their input from the standard input can have their input redirected from a file in this manner. For example, to count the number of lines in the file users generated above, you can execute the command as follows −

```
$ wc -l users
```

```
2 users

$
```

Upon execution, you will receive the following output. You can count the number of lines in the file by redirecting the standard input of the **wc** command from the file users −

```
$ wc -l < users2

$
```

Pipes and Filters in Linux/Unix

Pipes in UNIX

The novel idea of Pipes was introduced by **M.D McIlroy** in **June 1972**- version 2, 10 UNIX installations. Piping is used to give the output of one command (written on LHS) as input to another command (written on RHS). Commands are piped together using vertical bar “ **|** ” symbol. **Syntax:**

```
command 1|command 2
```

Example:

- **Input:** ls|more
- **Output:** more command takes input from *ls* command and appends it to the standard output. It displays as many files that fit on the screen and highlighted *more* at the bottom of the screen. To see the

next screen hit enter or space bar to move one line at a time or one screen at a time respectively.

Filters in UNIX

In UNIX/Linux, filters are the set of commands that take input from standard input stream i.e. **stdin**, perform some operations and write output to standard output stream i.e. **stdout**. The stdin and stdout can be managed as per preferences using redirection and pipes. Common filter commands are: grep, more, sort.

1. grep Command: It is a pattern or expression matching command. It searches for a pattern or regular expression that matches in files or directories and then prints found matches.

Syntax:

```
$grep[options] "pattern to be matched" filename
```

Example:

```
Input : $grep 'hello' ist_file.txt
Output : searches hello in the ist_file.txt and outputs/returns
the lines containing 'hello'.
```

echo command in Linux with Examples

the echo command in Linux is a built-in command that allows users to display lines of text or strings that are passed as arguments. It is commonly used in shell scripts and batch files to output status text to the screen or a file.

Syntax of `echo` command in Linux

```
echo [option] [string]
```

Here,

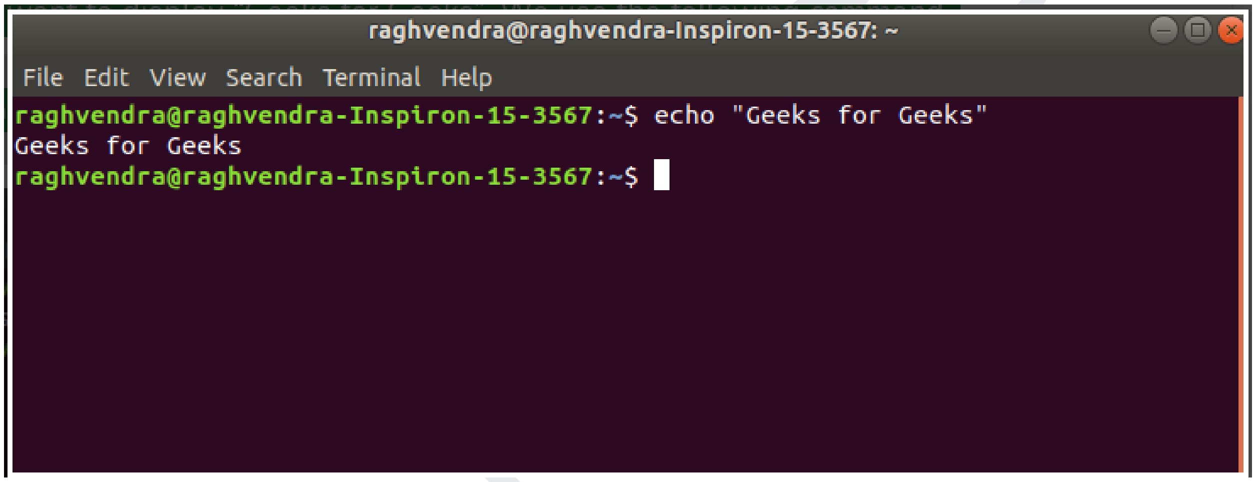
[options] = The various options available for modifying the behavior of the `echo` command

[string] = It is the string that we want to display.

Example:

If we want to display "Geeks for Geeks". We use the following command.

```
echo "Geeks for Geeks"
```


raghvendra@raghvendra-Inspiron-15-3567: ~
File Edit View Search Terminal Help

```
raghvendra@raghvendra-Inspiron-15-3567:~$ echo "Geeks for Geeks"
Geeks for Geeks
raghvendra@raghvendra-Inspiron-15-3567:~$
```

Linux Print

In Linux, different commands are used to print a file or output. Printing from a Linux terminal is a straightforward process. The **lp** and **lpr** commands are used to **print from the terminal**. And, the **lpg** command is used to **display queued print jobs**.

Printing the double-sided document or in portrait mode is a bit complicated process. And there may be many other operations that we want to perform, such as printing multiple copies or canceling a print job, which can be difficult to perform.

Linux lp and lpr command

CUPS (Common Unix Printing System) provides the system commands for printing files. Additionally, it supports several standard options to control the print operation. Let's see how to print files.

Read command in Linux with Examples

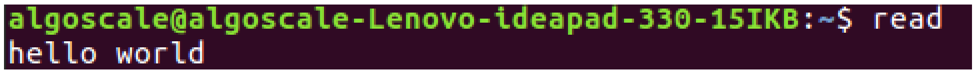
read command in Linux system is used to read from a file descriptor. Basically, this command read up the total number of bytes from the specified file descriptor into the buffer. If the number or count is zero then this command may detect the errors. But on success, it returns the number of bytes read. Zero indicates the end of the file. If some errors found then it returns -1.

Syntax:

```
read
```

Examples:

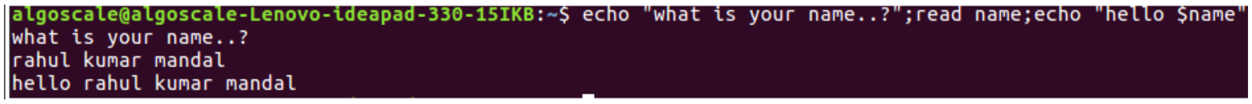
- **read command without any option:** The read command asks for the user's input and exit once the user provides some input.



```
algoscale@algoscale-Lenovo-ideapad-330-15IKB:~$ read
hello world
```

- In the following example we are acquiring the user's name and then showing the user's name with a greeting.

```
echo "what is your name..?";read name;echo "hello $name"
```



```
algoscale@algoscale-Lenovo-ideapad-330-15IKB:~$ echo "what is your name..?";read name;echo "hello $name"
what is your name..?
rahul kumar mandal
hello rahul kumar mandal
```