Introduction to Linux Shell

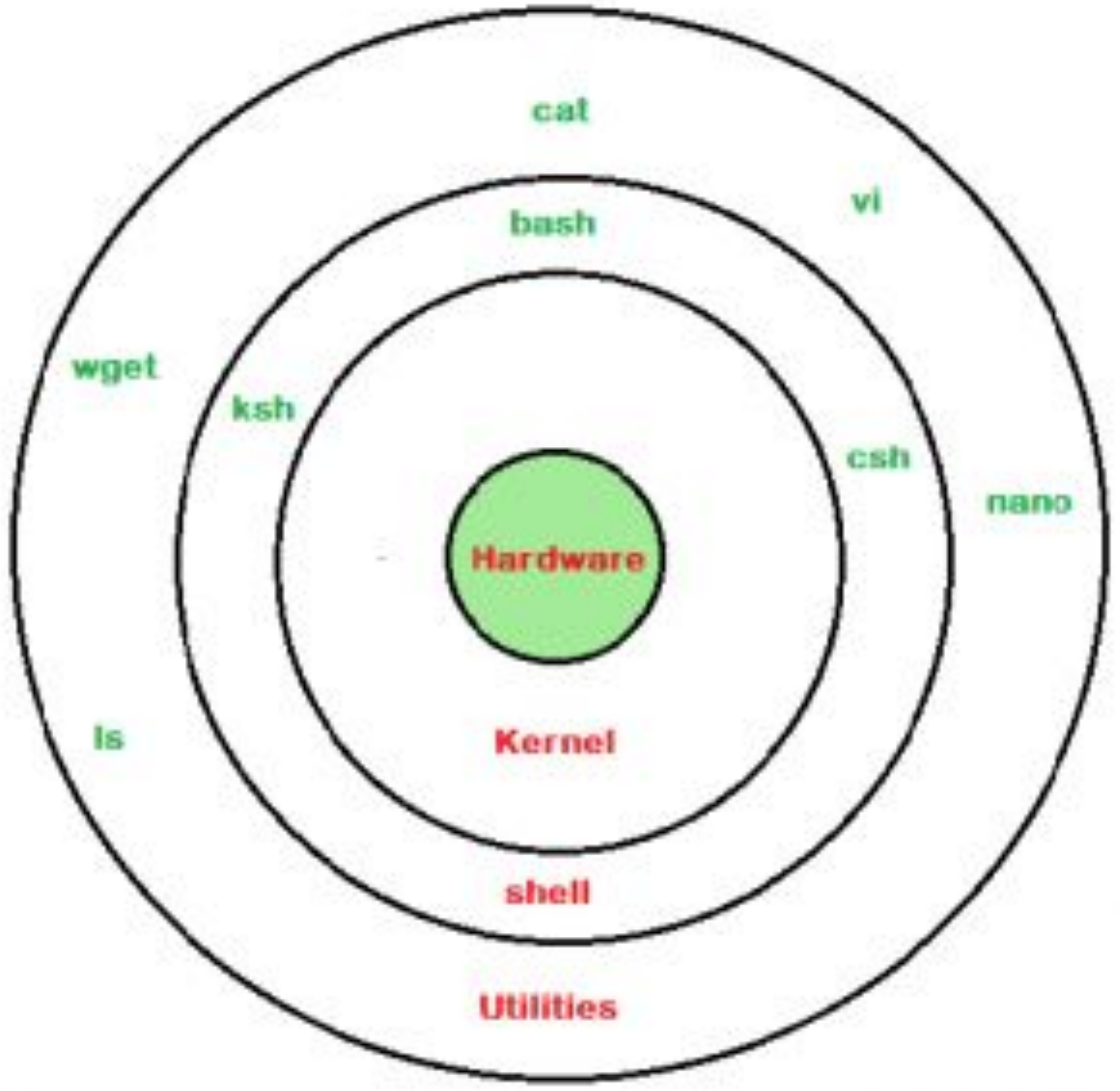
Sandy Joseph
Department of CSE,
RSET

https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/

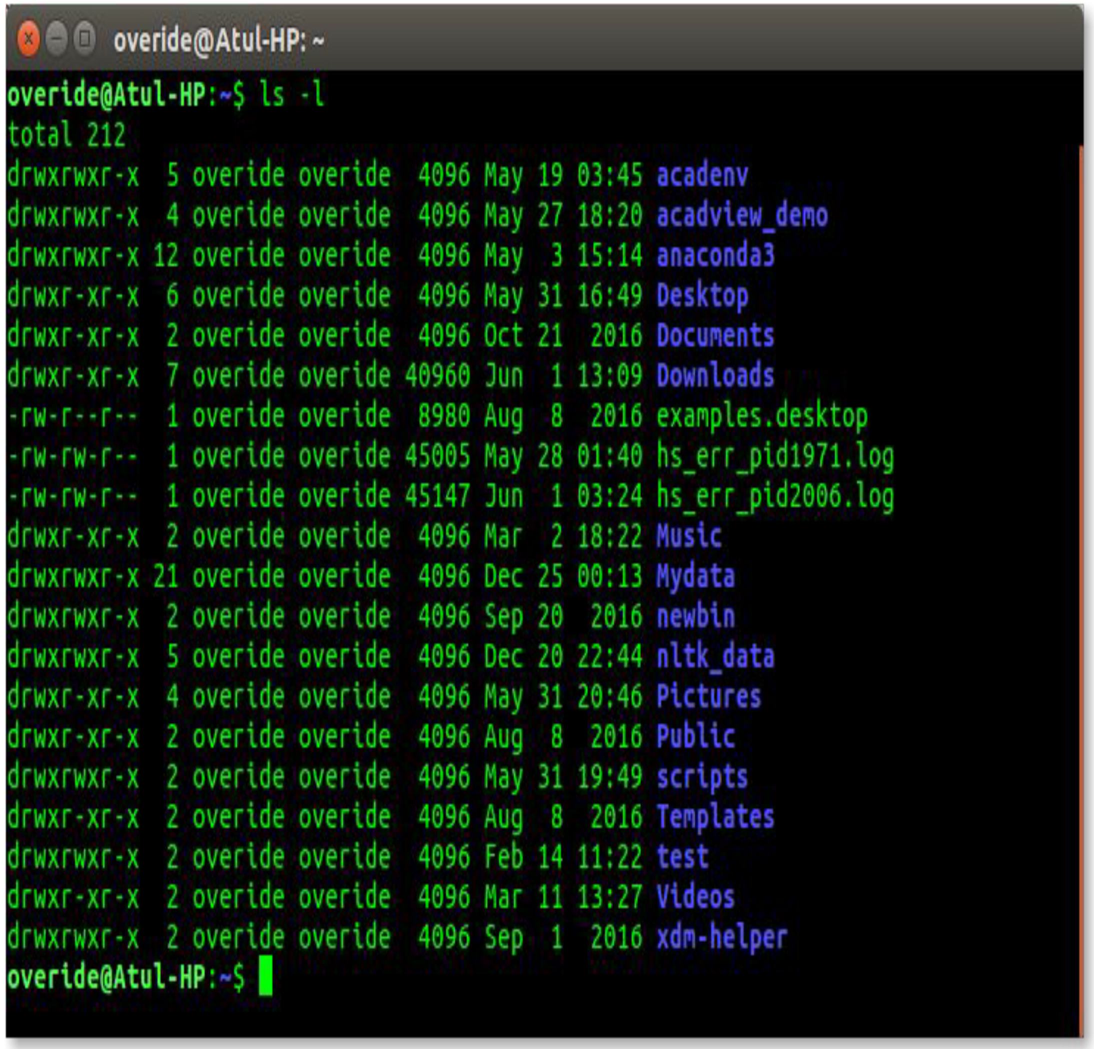
- In running Ubuntu, Linux Mint or any other Linux distribution, you are interacting to shell every time you use terminal.
- In a Linux system we have
  - Kernel
  - Shell
  - Terminal
- The **kernel** is a computer program that is the core of a computer's operating system, with complete control over everything in the system.
- It manages following resources of the Linux system –
  - File management
  - Process management
  - I/O management
  - Memory management
  - Device management etc.

Shell

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



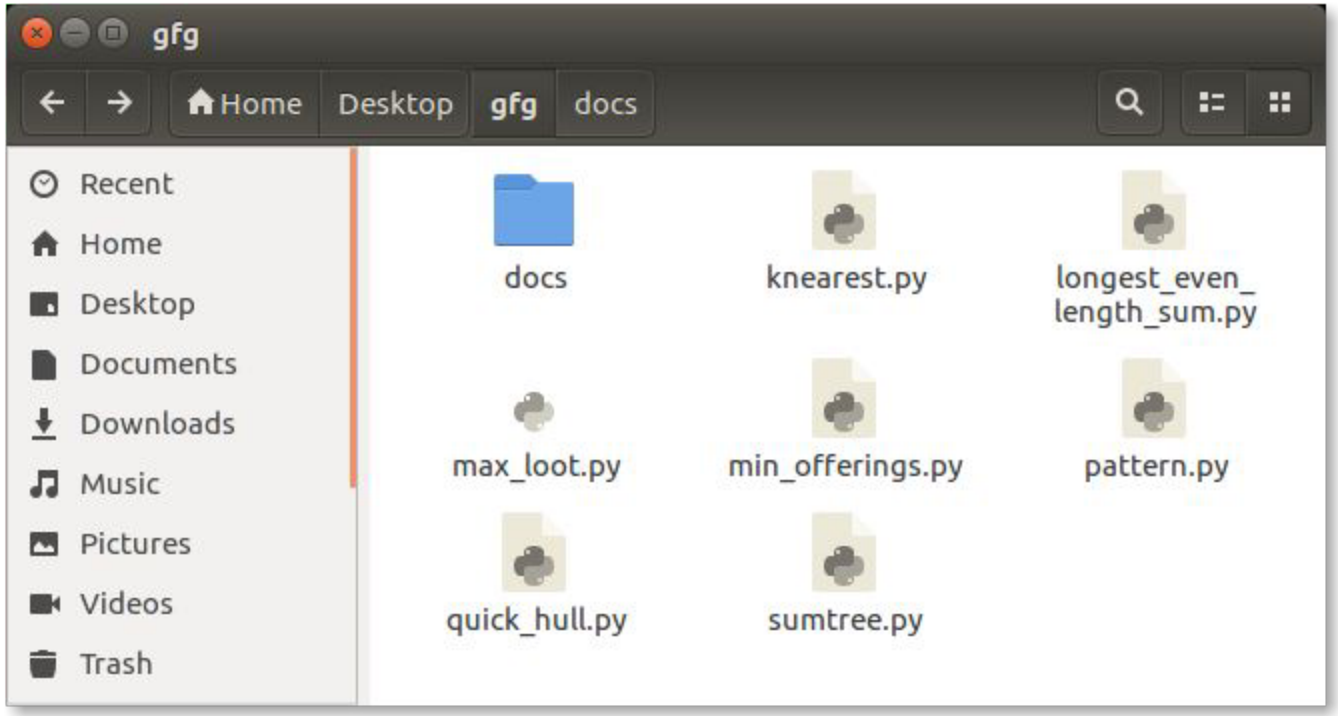
- Shell is broadly classified into two categories
  - Command Line Shell
  - Graphical shell
- **Command Line Shell**
- Shell can be accessed by user using a command line interface.
- A special program called Terminal in linux/macOS or Command Prompt in Windows OS is provided to type in the human readable commands such as “cat”, “ls” etc. and then it is being execute.
- 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:~$
```

Graphical Shells

- Window OS or Ubuntu OS can be considered as good example which provide GUI to user for interacting with program. User do not need to type in command for every actions. A typical GUI in Ubuntu system –



- There are several shells are available for Linux systems like –
- BASH (Bourne Again SHell) – It is most widely used shell in Linux systems. It is used as 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 usage are very similar to the C programming language.
- KSH (Korn SHell) – The Korn Shell also was the base for the POSIX Shell standard specifications etc.
- Each shell does the same job but u

Shell Scripting

- Usually shells are interactive that mean, they accept command as input from users and execute them.
- However some time we want to execute a bunch of commands routinely, so we have type in all commands each time in terminal.
- As shell can also take commands as input from file we can write these commands in a file and can execute them in 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 eg. **myscript.sh**

- A shell script comprises 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.

Need of 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.

Advantages of shell scripts

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

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.

Syntax

- A **Shell** provides you with an interface to the Unix system.
- We are using Bourne Again shell (bash)
- **$** character is the default prompt.
- Files are created as
  - gedit filename.sh
  - Eg:- gedit test.sh
  - Before you add anything else to your script, you need to alert the system that a shell script is being started.
    - This is done using the **shebang** construct.
    - For example #!/bin/sh
    - This is the first line of a shell program
  - Running the program
    - bash filename.sh

Variables

- Variable Names
- The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).
- By convention, Unix shell variables will have their names in UPPERCASE.
- The following examples are valid variable names :-
  - _ARUN
  - TOKEN_A
  - VAR_1
  - VAR_2
  - ITEM
  - N
- Read only variables- value can't be changed
  - readonly NAME
- Arrays
  - array_name[index]=value
  - E:- NAMES[0]= "HARI"
  - NAMES[1]="ARUN"
  - echo ${NAMES[0]}

Input and Output statement

Input statement

```
read variable_name
Eg:- read PERSON
     read N
     read ITEM
```

Assignment statement

```
variable_name = variable _value
Eg:- N=2
     PERSON = rahul
```

Output statement

```
echo “message”
Eg:- echo “hello world”
echo $variable_name
```

Printing

Develop a shell script to display a message "Hello World".

```
#!/bin/bash
echo "Hello World"
```

Arithmetic operators

+	Addition
-	Subtraction
•	Multiplication
/	Division
%	Modulus
=	Assignment
++	increment
--	decrement

Relational operators

-eq	equal to
-ne	not equal to
-gt	greater than
-lt	less than
-ge	greater than or equal to
-le	less than or equal to

OR

==	equal to
!=	not equal to
>	Greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Logical operators

!	logical negation
-o	logical OR
-a	logical AND

OR

!	Logical negation
||	logical OR
&&	logical AND

Computations

Develop a shell script to find the sum of two numbers.

```
#!/bin/bash
echo "Enter first number"
read n1
echo "Enter second number"
read n2
s=$((n1 + n2))
echo "Sum of $n1 and $n2 is $s"
```

Another method

```
#!/bin/bash
echo "Enter first number"
read n1
echo "Enter second number"
read n2
s=`expr $n1 + $n2`
echo "Sum of $n1 and $n2 is $s"
```

if else

- **if...fi statement**
```
if [ expression]
then
 statements
fi
```
- **if...else...fi statement**
```
if [ expression]
then
    statements
else
        statements
fi
```

Develop a shell script to check if a number is odd or even.

if...elif...else...fi statement
```
if [expression 1]
then
     Statements
elif[expression 2]
then
     statements
elif[expression 3]
then
      statements
else
     statements
fi
```

```
#!/bin/bash
echo "Enter a number"
read n
r=$((n%2))
if [ $r -eq 0 ]
then
echo "$n is even"
else
echo "$n is odd"
fi
```

```
#!/bin/bash
echo "Enter a number"
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then
echo "$n is even"
else
echo "$n is odd"
fi
```

else if ladder

Develop a shell script to find the largest among two numbers

```
#!/bin/bash

echo "Enter first number"

read n1

echo "Enter second number"

read n2
```

```
if [ $n1 -gt $n2 ]
then
echo "$n1 is largest"
elif [ $n1 -lt $n2 ]
then
echo "$n2 is largest"
else
echo "Both numbers are equal"
fi
```

if elif elif else ladder

```
#!/bin/sh
a=10
b=20
if [ $a == $b ]
then
    echo "a is equal to b"
elif [ $a -gt $b ]
then
     echo "a is greater than b“
elif [ $a -lt $b ]
then
      echo "a is less than b"
else echo "None of the condition met“
fi
```

Decision making

- while loop
- for loop
- until loop
- select loop

```
while [condition]
do
    statements
done
```

for loop

Develop a shell script to print all the odd numbers less than n.

```
#!/bin/bash
echo "Enter limit"
read n
for(( i=1; i<n; i=i+2 ))
do
echo "$i"
done
```

while loop

Develop a shell script to print all the even number less than or equal to n.

```
#!/bin/bash

echo "Enter limit"

read n

i=2

while [ $i -le $n ]

do

echo "$i"

i=`expr $i + 2`

done
```

break

Develop a shell script to check if a number is prime or not.

```
#!/bin/bash

echo -n "Enter the number : "

read num

flag=0
```

```
for(( i=2; i<num; i++ ))
do
    if [ $((num%i)) -eq 0 ]
    then
        flag=1
        break
    fi
done
```

```
if [ $flag -eq 0 ]
then
    echo "$num is a prime number"
else
    echo "$num is not a prime number"
fi
```

Develop a shell script to print all numbers from 1 to 100

```
#!/bin/bash
for(( i=1; i<=100; i++ ))
do
echo "$i"
done
```

```
Step 1: Start
Step 2: Write the shebang directive
Step 3: Initialise $i = 1$
Step 4: Repeat as long as $i \leq 100$
        Step 4.1: Print $i$
        Step 4.2: Set $i = i + 1$
Step 5: Stop
```

nested for loops

Develop a shell script to print all the prime numbers between the given upper and lower limits.

arrays

Develop a shell script to perform linear search on an array.

```
#!/bin/bash
echo "Enter limit of array"
read n
echo "Enter elements in the
array"
for(( i=0; i<n; i++ ))
do
    read a[$i]
done
```

```
echo "Elements in the array are:"
for(( i=0; i<n; i++ ))
do
    echo "${a[$i]}"
done
echo "Enter the element to be searched"
read element
flag=0
```

```
for(( i=0; i<n; i++ ))
do
if [ ${a[$i]} -eq $element ]
then
    flag=1
    break
fi
done
```

```
if [ $flag -eq 1 ]
then
    echo "$ele is found"
else
    echo "$ele is not found"
fi
```

Running shell script

Open file

```
gedit filename.sh
```

Running the script

```
bash filename.sh
```

Thank You!