Operating Systems Lab Manual

B.Tech. CSE (II YEAR – III/IV SEM)

(2024-2025)

DEPARTMENT OF COMPUTER ENGINEERING & APPLICATIONS

Institute of Engineering & Technology

GLA University

17km Stone, NH-19, Mathura-Delhi Road P.O. Chaumuhan, Mathura - 281406 (Uttar Pradesh) India

LINUX COMMANDS

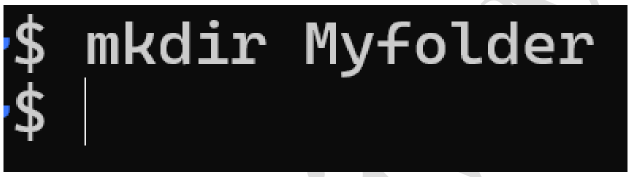
Prepared By: Dr. Premnarayan Arya

1. Implement the following basic Linux commands in Ubuntu/Terminal

mkdir	ls	cd	cat	touch	man
pwd	who	whoami	date	cal	rm
rmdir	head	tail	more	less	cp
mv	echo	history	sed	grep	pipe
cut	uniq	chmod	du	diff	last

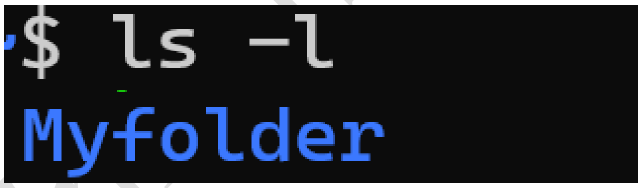
Solution 1:

- **mkdir:** create a new directories or folder.



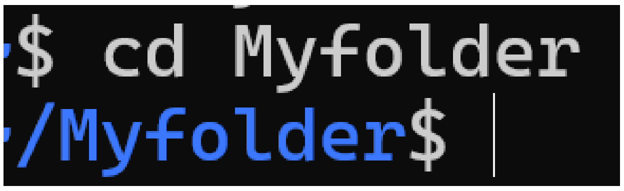
```
$ mkdir Myfolder
$
```

- **ls:** To display all files and directories.



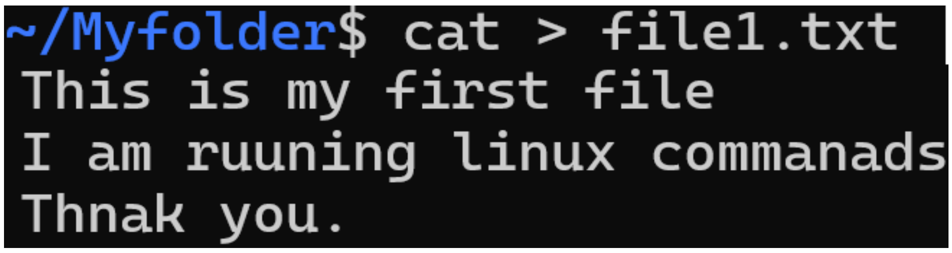
```
$ ls -l
Myfolder
```

- **cd:** To change a directory.



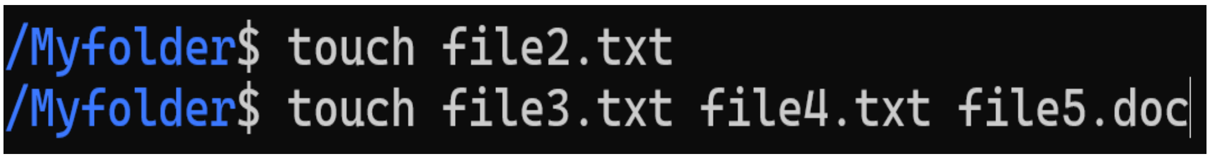
```
$ cd Myfolder
/Myfolder$
```

- **cat:** To create a new file with contents.



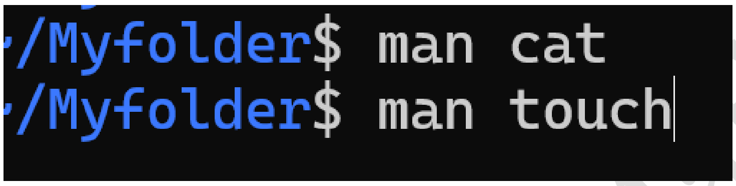
```
~/Myfolder$ cat > file1.txt
This is my first file
I am ruuning linux commanads
Thnak you.
```

- **touch:** To create an empty (or blank) file, also create multiple files.



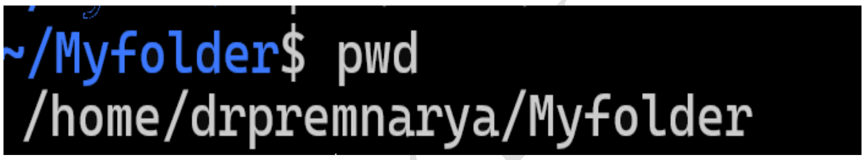
```
/Myfolder$ touch file2.txt
/Myfolder$ touch file3.txt file4.txt file5.doc
```

- **man:** To display manual of any command.



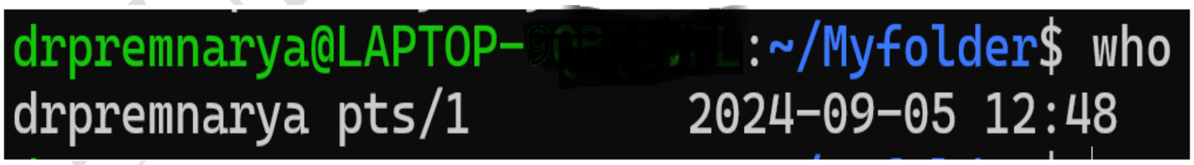
```
/Myfolder$ man cat
/Myfolder$ man touch
```

- **pwd:** To display present working directory.



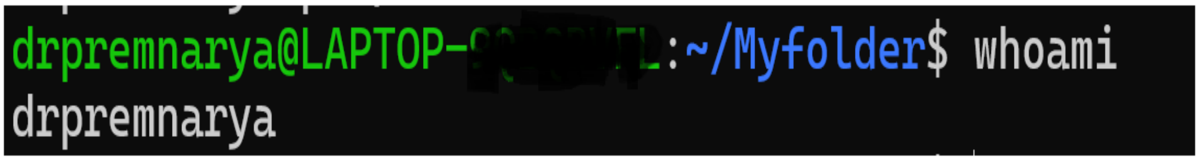
```
~/Myfolder$ pwd
/home/drpremnarya/Myfolder
```

- **who:** To display information about the users currently logged into the system. It provides details such as the username, terminal, login time, and sometimes the originating host.



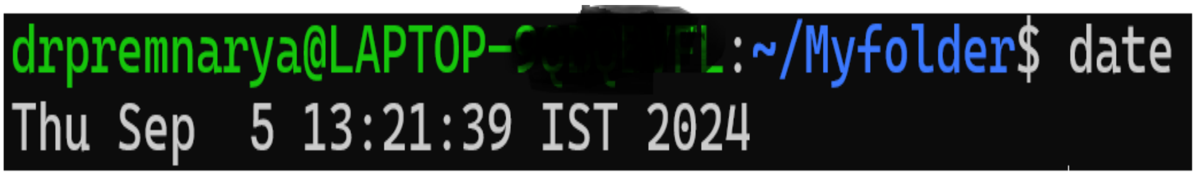
```
drpremnarya@LAPTOP-9[redacted]L:~/Myfolder$ who
drpremnarya pts/1        2024-09-05 12:48
```

- **whoami:** To display the effective username of the current user. It shows the username of the user who is currently logged in and executing the command.



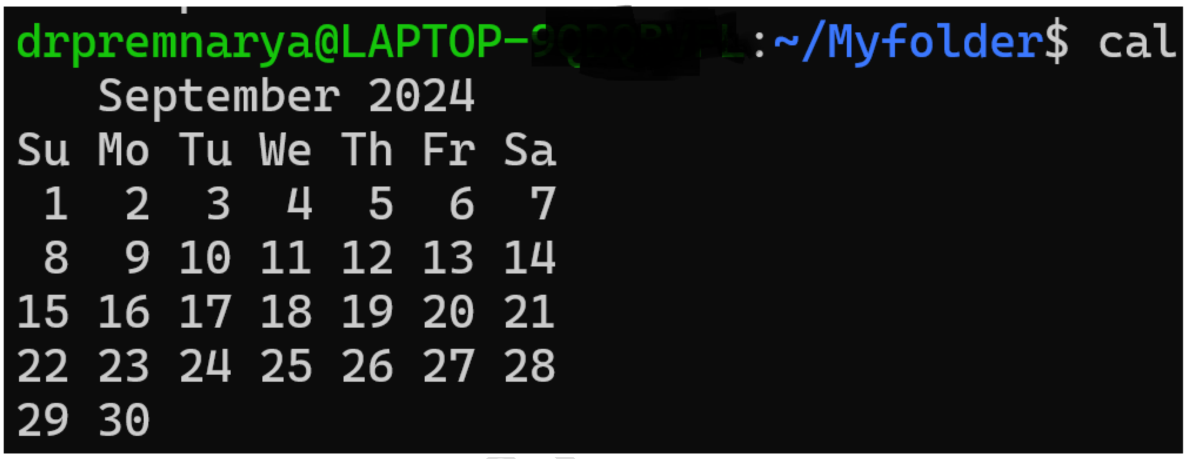
```
drpremnarya@LAPTOP-9[redacted]L:~/Myfolder$ whoami
drpremnarya
```

- **date:** print or set the system date and time.



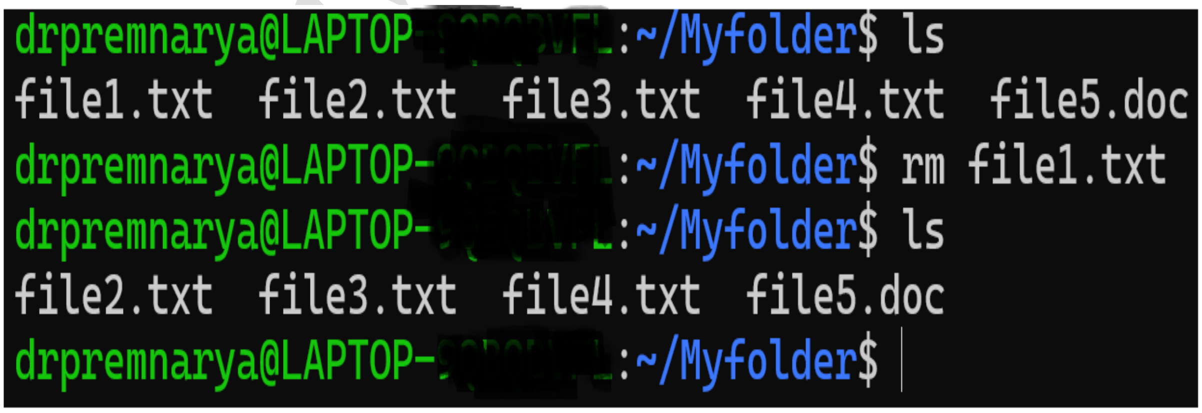
```
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ date
Thu Sep  5 13:21:39 IST 2024
```

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



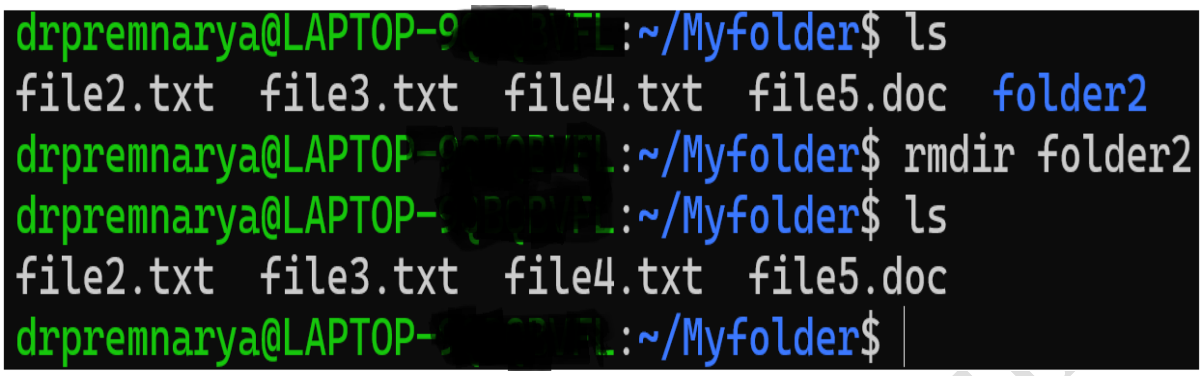
```
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ cal
   September 2024
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

- **rm:** To remove a file.



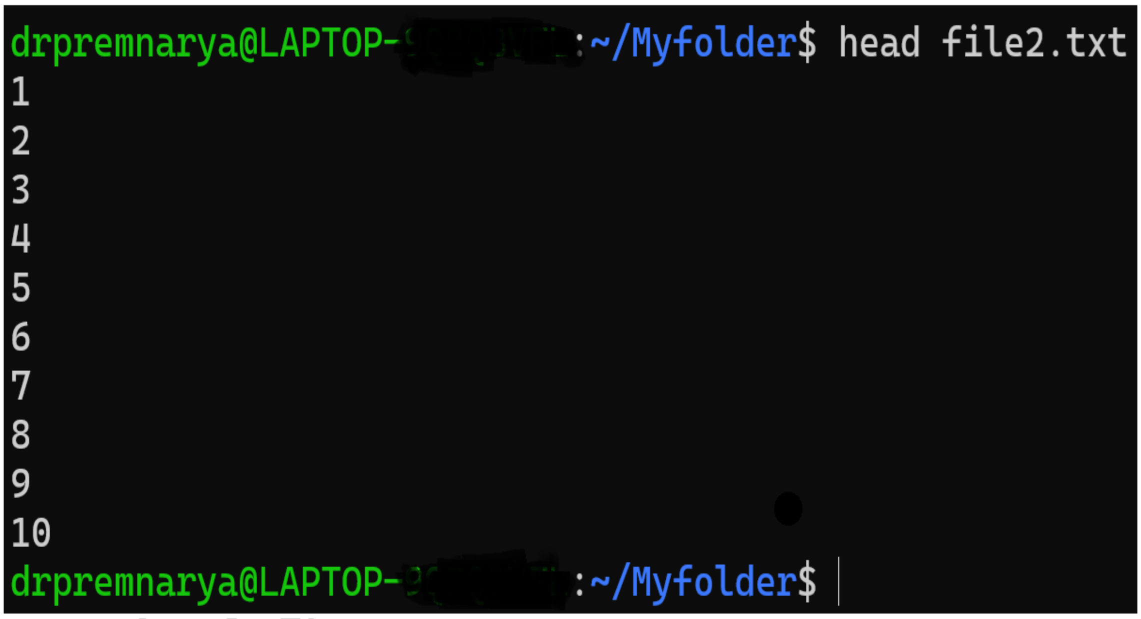
```
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ ls
file1.txt  file2.txt  file3.txt  file4.txt  file5.doc
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ rm file1.txt
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ ls
file2.txt  file3.txt  file4.txt  file5.doc
drpremnarya@LAPTOP-[redacted]:~/Myfolder$
```

- **rmdir:** To remove a directory/folder.



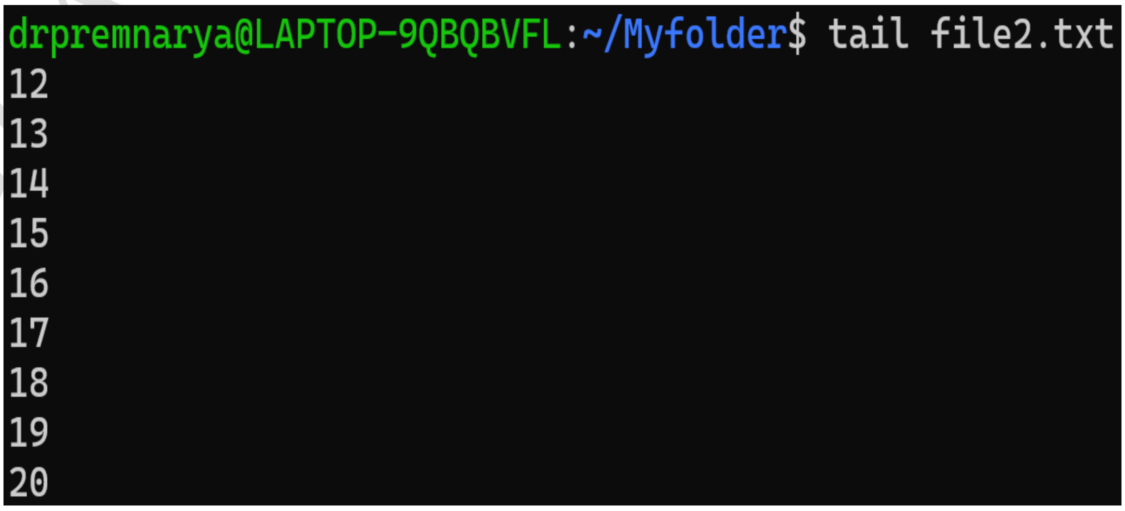
```
drpremnarya@LAPTOP-9[redacted]:~/Myfolder$ ls
file2.txt  file3.txt  file4.txt  file5.doc  folder2
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ rmdir folder2
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ ls
file2.txt  file3.txt  file4.txt  file5.doc
drpremnarya@LAPTOP-[redacted]:~/Myfolder$
```

- **head:** To print 10 lines from top of any file.



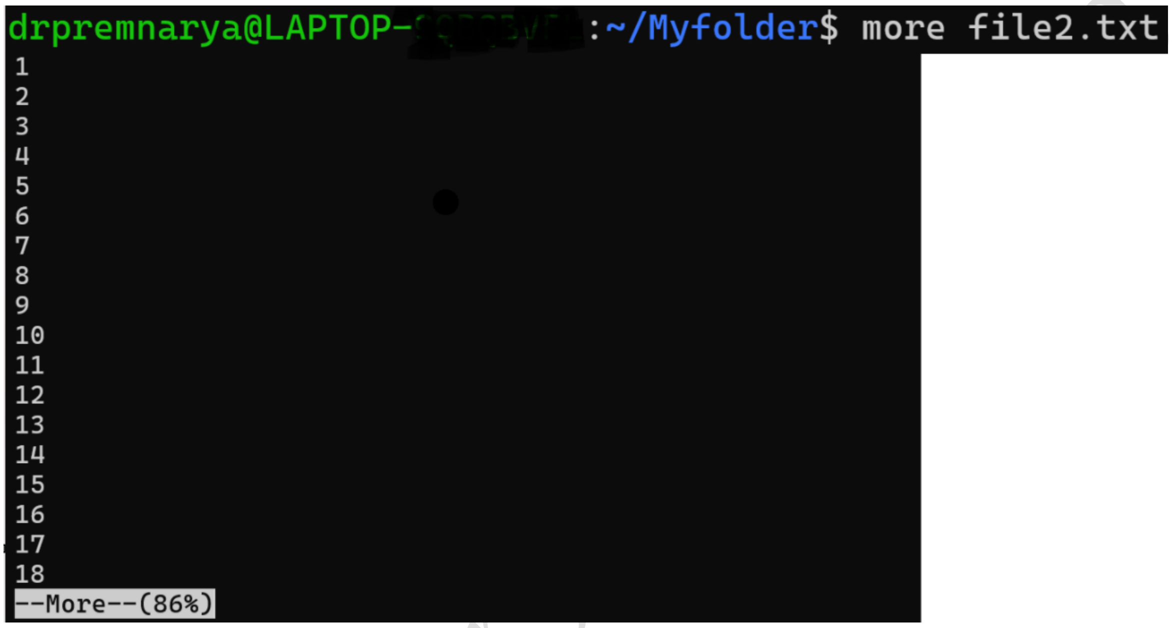
```
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ head file2.txt
1
2
3
4
5
6
7
8
9
10
drpremnarya@LAPTOP-[redacted]:~/Myfolder$
```

- **tail:** To print 10 lines from bottom of any file.



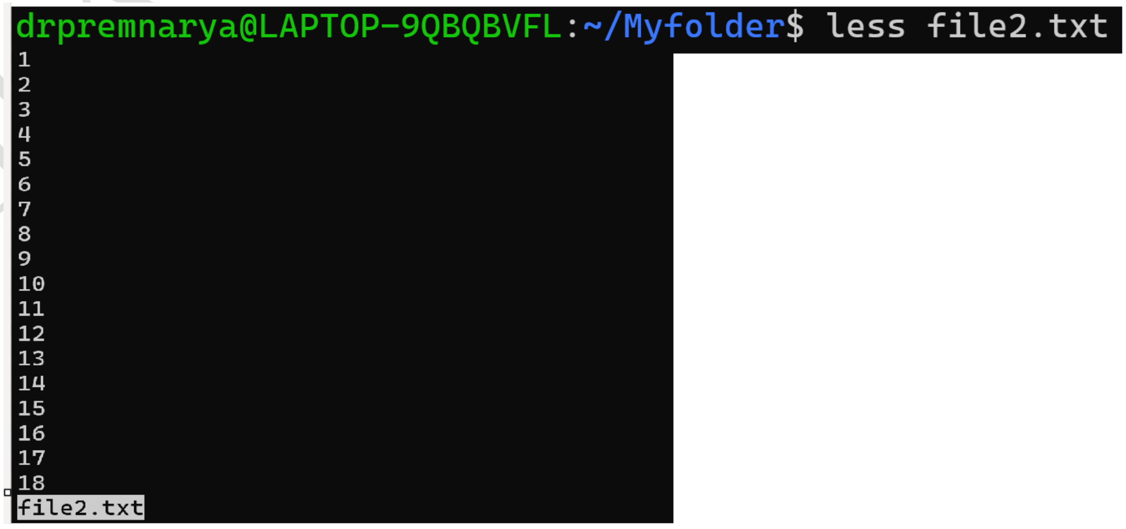
```
drpremnarya@LAPTOP-9QBQBVFL:~/Myfolder$ tail file2.txt
12
13
14
15
16
17
18
19
20
```

- **more:** The more command in Linux is a simple text pager used to view the contents of a file one screen at a time. It allows users to navigate through large files or command output in a controlled manner. Unlike cat, which displays the entire content at once, more pauses after each screenful, making it easier to read.



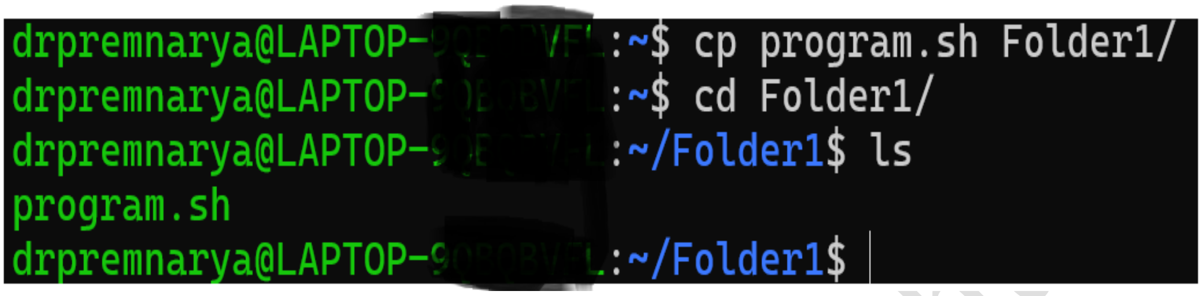
```
drpremnarya@LAPTOP-[redacted]:~/Myfolder$ more file2.txt
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
--More--(86%)
```

- **less:** The less command in Linux is a powerful text pager that allows users to view the contents of files one screen at a time, similar to the more command, but with more advanced features. Unlike more, less enables backward and forward navigation, as well as efficient searching and navigation within files.



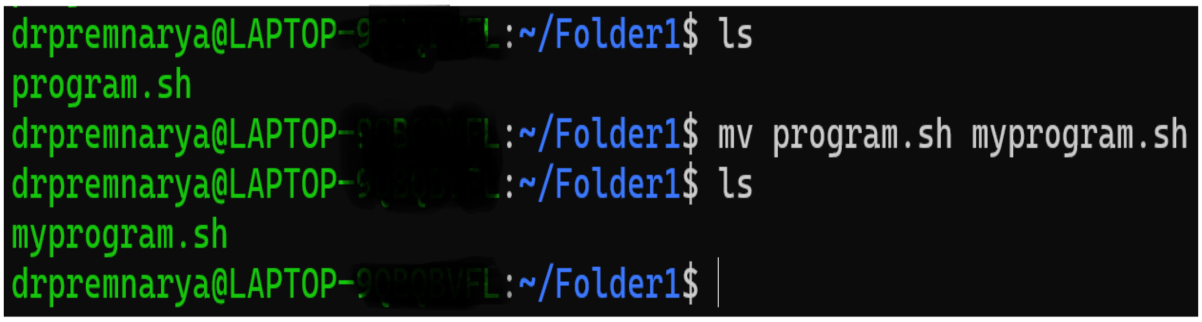
```
drpremnarya@LAPTOP-9QBQBVFL:~/Myfolder$ less file2.txt
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
file2.txt
```

- **cp:** The cp command in Linux is used to copy files and directories from one location to another. It is one of the most commonly used commands for managing files.



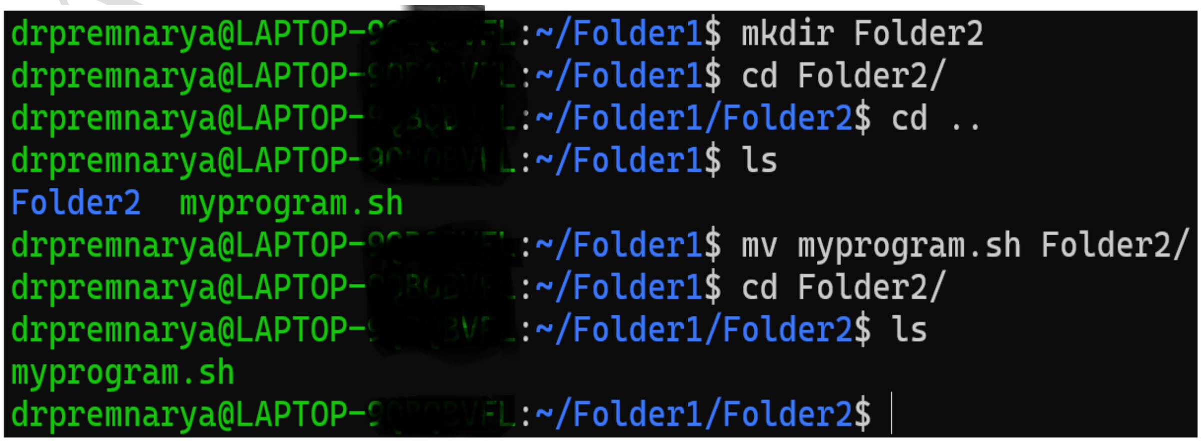
```
drpremnarya@LAPTOP-[redacted]:~$ cp program.sh Folder1/
drpremnarya@LAPTOP-[redacted]:~$ cd Folder1/
drpremnarya@LAPTOP-[redacted]:~/Folder1$ ls
program.sh
drpremnarya@LAPTOP-[redacted]:~/Folder1$
```

- **mv:** The mv command in Linux is used to move or rename files and directories. It is a versatile command that can be used both to relocate files to a different directory and to rename files and directories within the same location.
  - Rename program.sh file by new name myprogram.sh



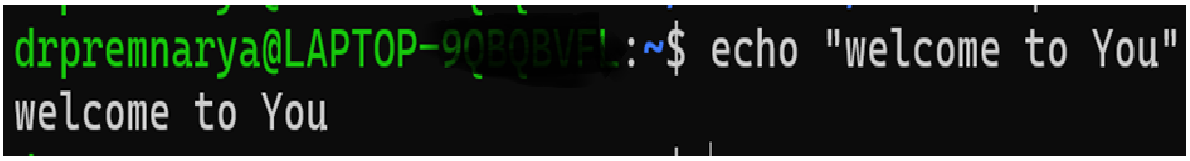
```
drpremnarya@LAPTOP-[redacted]:~/Folder1$ ls
program.sh
drpremnarya@LAPTOP-[redacted]:~/Folder1$ mv program.sh myprogram.sh
drpremnarya@LAPTOP-[redacted]:~/Folder1$ ls
myprogram.sh
drpremnarya@LAPTOP-[redacted]:~/Folder1$
```

  - Move/cut myprogram.sh file into the Folder2



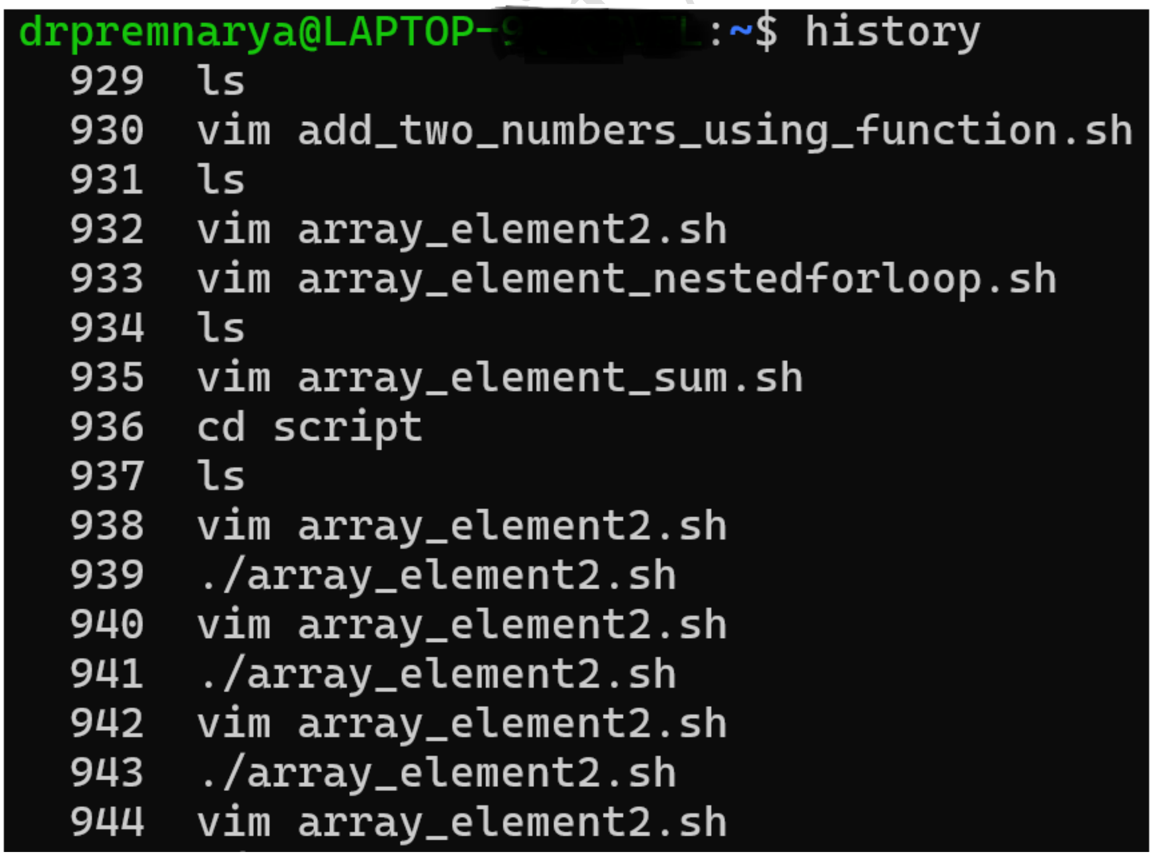
```
drpremnarya@LAPTOP-[redacted]:~/Folder1$ mkdir Folder2
drpremnarya@LAPTOP-[redacted]:~/Folder1$ cd Folder2/
drpremnarya@LAPTOP-[redacted]:~/Folder1/Folder2$ cd ..
drpremnarya@LAPTOP-[redacted]:~/Folder1$ ls
Folder2  myprogram.sh
drpremnarya@LAPTOP-[redacted]:~/Folder1$ mv myprogram.sh Folder2/
drpremnarya@LAPTOP-[redacted]:~/Folder1$ cd Folder2/
drpremnarya@LAPTOP-[redacted]:~/Folder1/Folder2$ ls
myprogram.sh
drpremnarya@LAPTOP-[redacted]:~/Folder1/Folder2$
```

- **echo:** The echo command in Linux is used to display a line of text or a string on the terminal. It is commonly used in scripts, batch files, and command-line operations to output text, display variables, and generate formatted text.



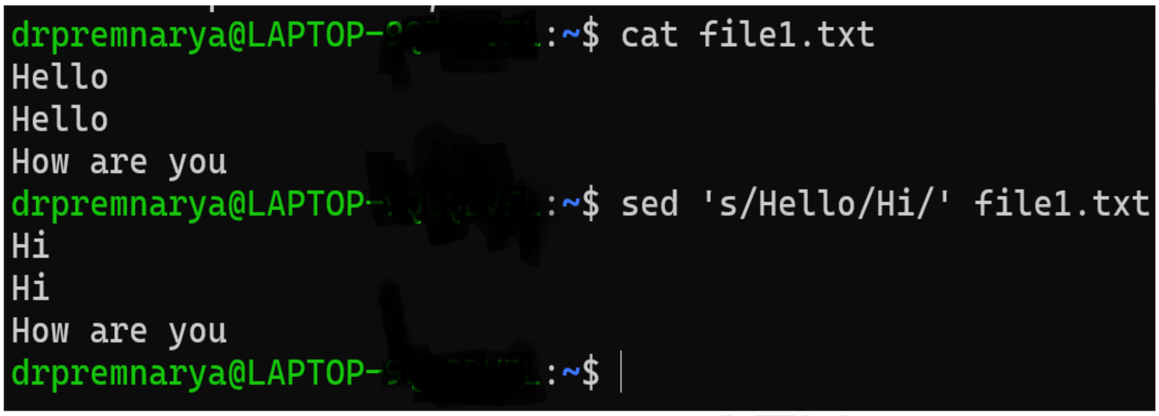
```
drpremnarya@LAPTOP-9[redacted]L:~$ echo "welcome to You"
welcome to You
```

- **history:** The history command in Linux is used to display a list of previously executed commands in the terminal. It is a helpful tool for recalling past commands, rerunning them, or even editing them. The history command is particularly useful for tracking command usage, debugging, and learning.



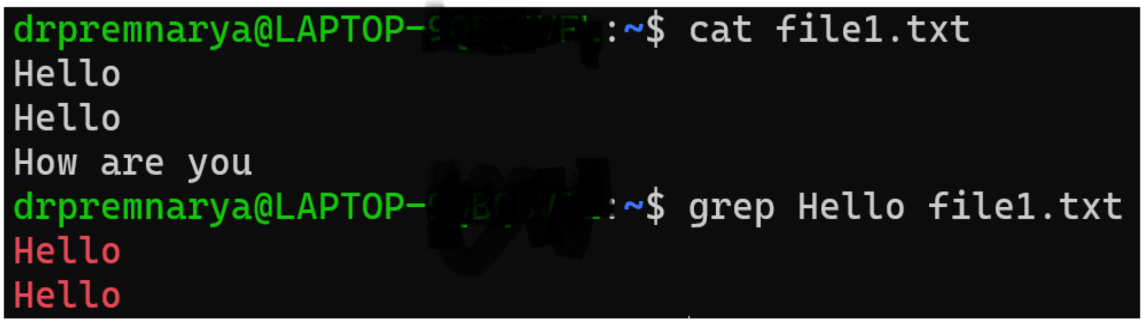
```
drpremnarya@LAPTOP-[redacted]L:~$ history
  929  ls
  930  vim add_two_numbers_using_function.sh
  931  ls
  932  vim array_element2.sh
  933  vim array_element_nestedforloop.sh
  934  ls
  935  vim array_element_sum.sh
  936  cd script
  937  ls
  938  vim array_element2.sh
  939  ./array_element2.sh
  940  vim array_element2.sh
  941  ./array_element2.sh
  942  vim array_element2.sh
  943  ./array_element2.sh
  944  vim array_element2.sh
```

- **sed:** The sed command in Linux stands for "stream editor" and is used to perform basic text transformations and manipulations on files or input streams. It processes text line by line and is commonly used for tasks such as searching, find-and-replace operations, and text formatting.



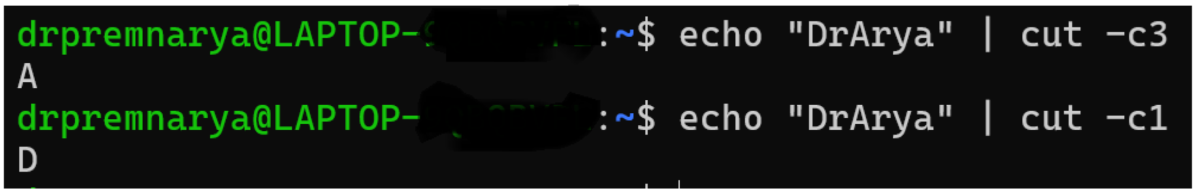
```
drpremnarya@LAPTOP-[redacted]:~$ cat file1.txt
Hello
Hello
How are you
drpremnarya@LAPTOP-[redacted]:~$ sed 's/Hello/Hi/' file1.txt
Hi
Hi
How are you
drpremnarya@LAPTOP-[redacted]:~$
```

- **grep:** The grep command in Linux is used for searching text using patterns. It stands for "Global Regular Expression Print" and is commonly used to search through files and output lines that match a given pattern.



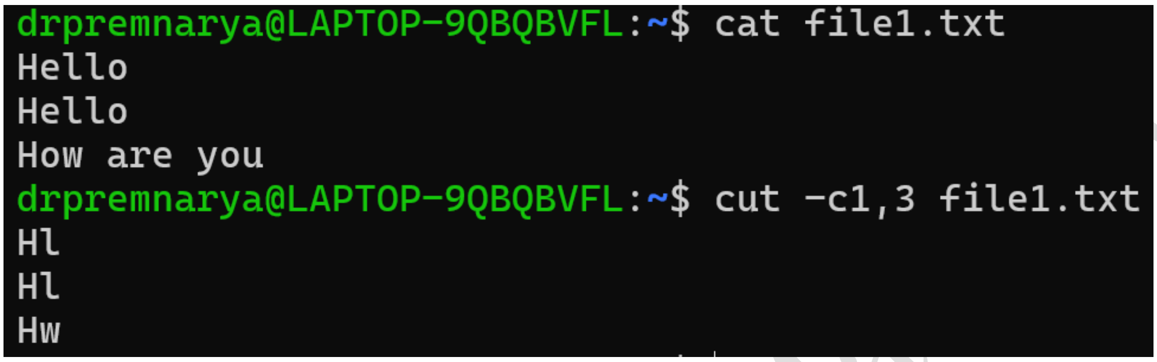
```
drpremnarya@LAPTOP-[redacted]:~$ cat file1.txt
Hello
Hello
How are you
drpremnarya@LAPTOP-[redacted]:~$ grep Hello file1.txt
Hello
Hello
```

- **pipe:** In Linux, the pipe command (|) is used to pass the output of one command as the input to another command. This allows you to chain commands together and perform complex data processing in a single line of code.



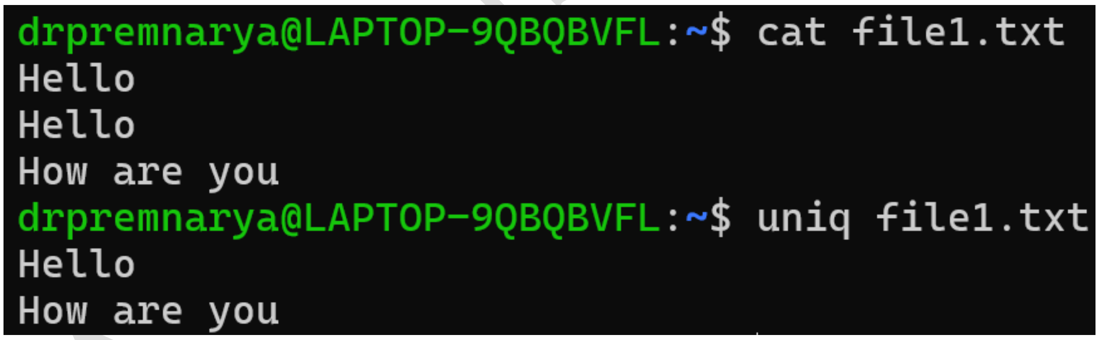
```
drpremnarya@LAPTOP-[redacted]:~$ echo "DrArya" | cut -c3
A
drpremnarya@LAPTOP-[redacted]:~$ echo "DrArya" | cut -c1
D
```

- **cut:** The cut command in Linux is used to extract sections from each line of input, such as files or standard input. It is commonly used for extracting columns or fields from structured text files or output from other commands.



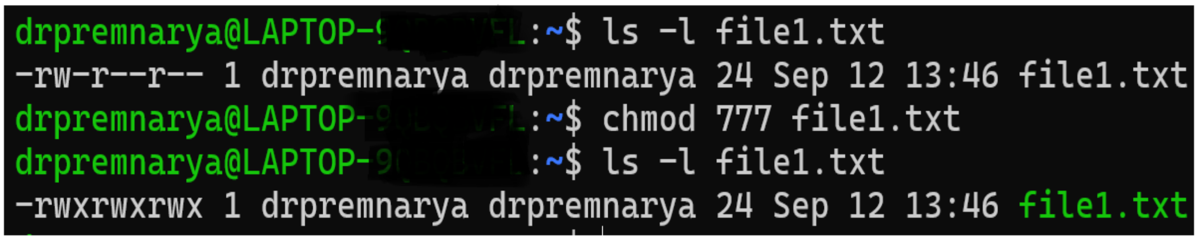
```
drpremnarya@LAPTOP-9QBQBVFL:~$ cat file1.txt
Hello
Hello
How are you
drpremnarya@LAPTOP-9QBQBVFL:~$ cut -c1,3 file1.txt
Hl
Hl
Hw
```

- **uniq:** The uniq command in Linux is used to filter out or report repeated lines in a file or input stream. It is often used to remove duplicate lines or count occurrences of unique lines. The uniq command works best when used in combination with other commands, such as sort, to ensure that duplicate lines are adjacent.



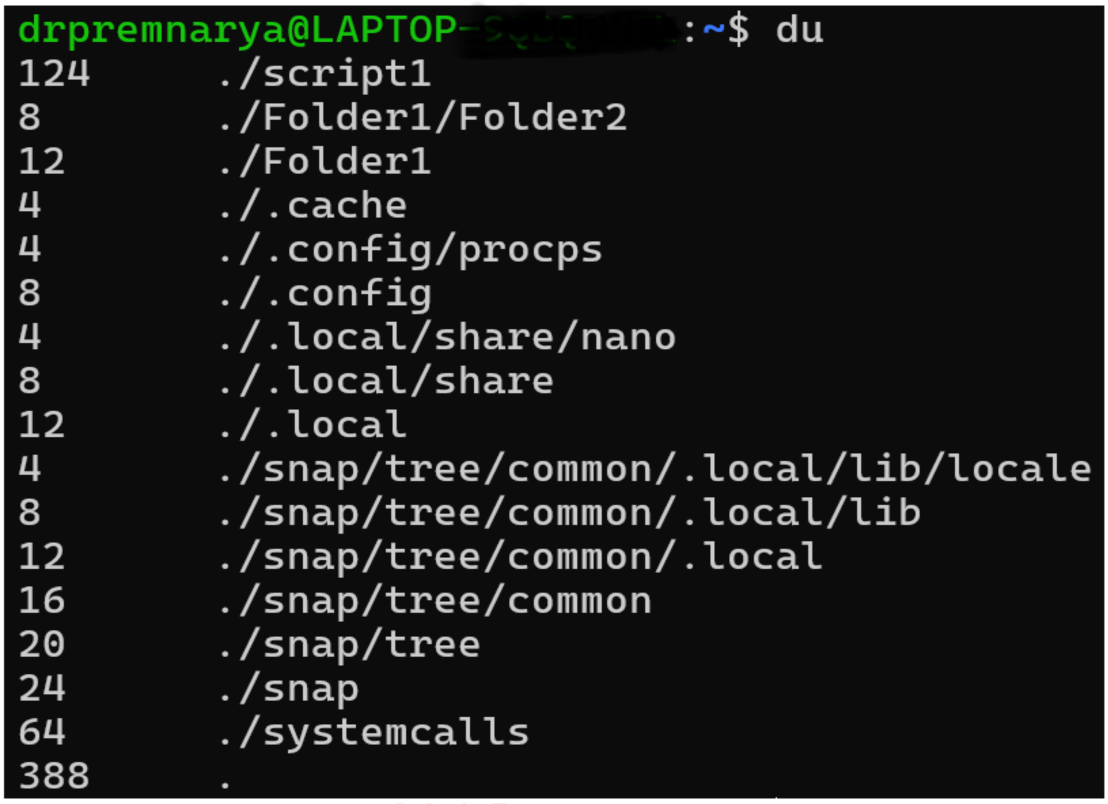
```
drpremnarya@LAPTOP-9QBQBVFL:~$ cat file1.txt
Hello
Hello
How are you
drpremnarya@LAPTOP-9QBQBVFL:~$ uniq file1.txt
Hello
How are you
```

- **chmod:** The chmod command in Linux is used to change the permissions of files and directories. The name chmod stands for "change mode," and it allows users to define who can read, write, or execute a file.



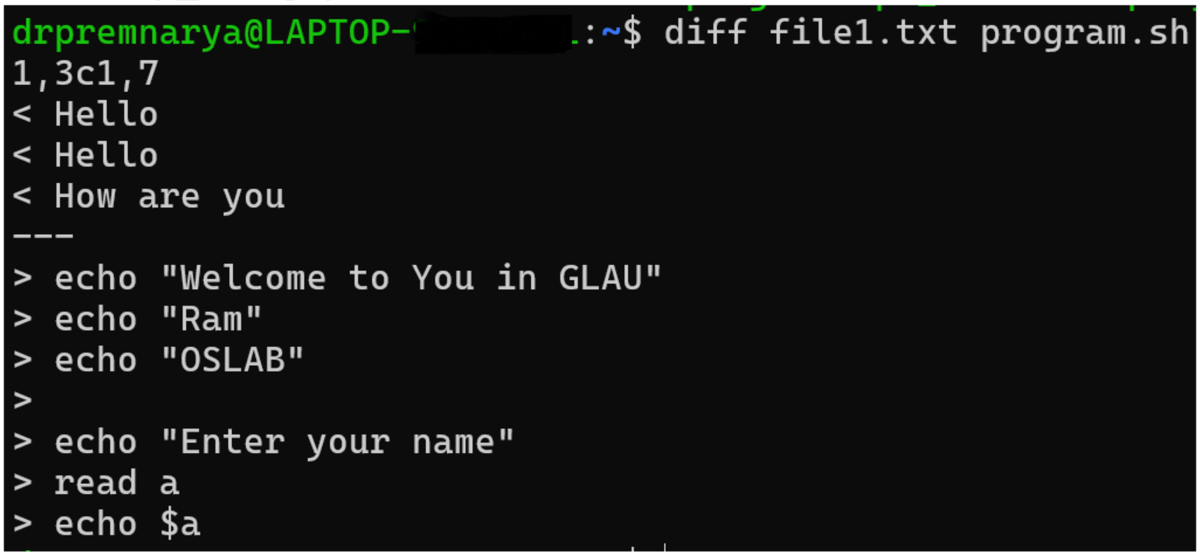
```
drpremnarya@LAPTOP-[redacted]L:~$ ls -l file1.txt
-rw-r--r-- 1 drpremnarya drpremnarya 24 Sep 12 13:46 file1.txt
drpremnarya@LAPTOP-[redacted]L:~$ chmod 777 file1.txt
drpremnarya@LAPTOP-[redacted]L:~$ ls -l file1.txt
-rwxrwxrwx 1 drpremnarya drpremnarya 24 Sep 12 13:46 file1.txt
```

- **du:** The du (disk usage) command in Linux is used to estimate and display the amount of disk space used by files and directories. It's useful for checking disk space usage and managing storage.



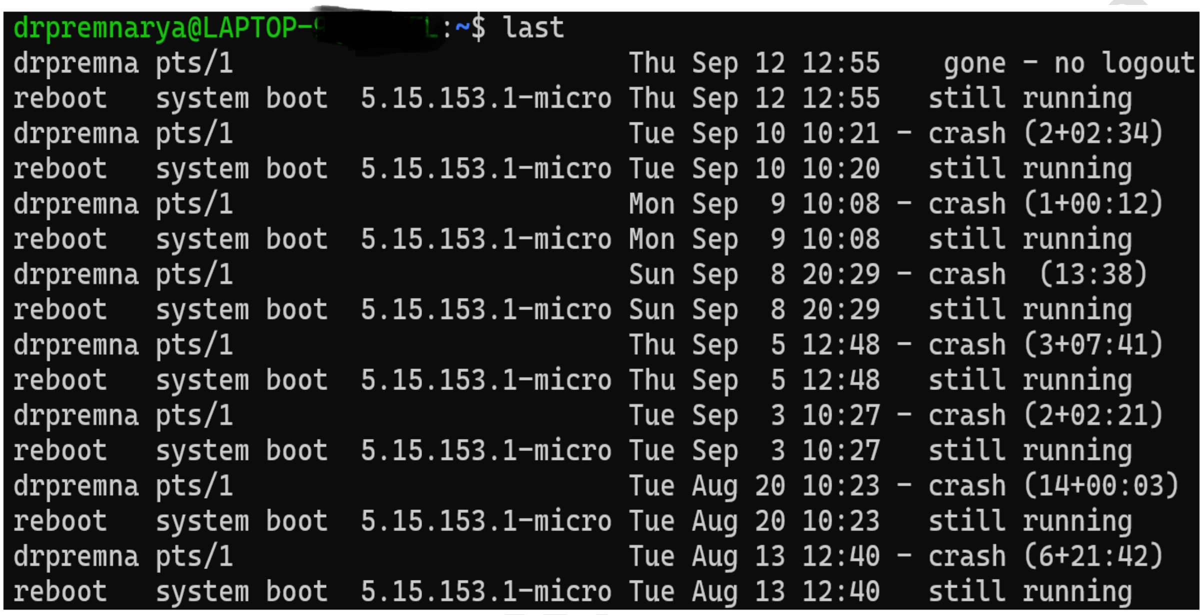
```
drpremnarya@LAPTOP-[redacted]:~$ du
124     ./script1
8       ./Folder1/Folder2
12      ./Folder1
4       ./.cache
4       ./.config/procps
8       ./.config
4       ./.local/share/nano
8       ./.local/share
12      ./.local
4       ./snap/tree/common/.local/lib/locale
8       ./snap/tree/common/.local/lib
12      ./snap/tree/common/.local
16      ./snap/tree/common
20      ./snap/tree
24      ./snap
64      ./systemcalls
388     .
```

- **diff:** The diff command in Linux is used to compare the contents of two files line by line. It displays the differences between the files, making it useful for identifying changes, updates, or discrepancies.



```
drpremnarya@LAPTOP-[redacted]:~$ diff file1.txt program.sh
1,3c1,7
< Hello
< Hello
< How are you
---
> echo "Welcome to You in GLAU"
> echo "Ram"
> echo "OSLAB"
>
> echo "Enter your name"
> read a
> echo $a
```

- **last:** The last command in Linux is used to display a list of the most recent logins to the system. It reads from the /var/log/wtmp file, which records login, logout, and system boot events. This command is useful for auditing and monitoring user activity.

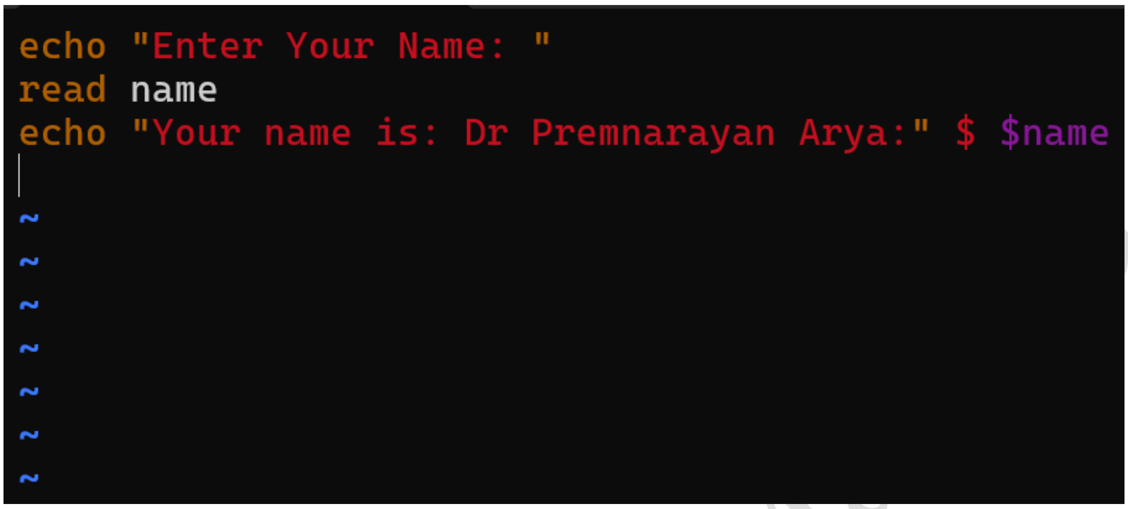


```
drpremnarya@LAPTOP-[redacted]L:~$ last
drpremna pts/1                         Thu Sep 12 12:55    gone - no logout
reboot   system boot  5.15.153.1-micro Thu Sep 12 12:55   still running
drpremna pts/1                         Tue Sep 10 10:21 - crash (2+02:34)
reboot   system boot  5.15.153.1-micro Tue Sep 10 10:20   still running
drpremna pts/1                         Mon Sep  9 10:08 - crash (1+00:12)
reboot   system boot  5.15.153.1-micro Mon Sep  9 10:08   still running
drpremna pts/1                         Sun Sep  8 20:29 - crash  (13:38)
reboot   system boot  5.15.153.1-micro Sun Sep  8 20:29   still running
drpremna pts/1                         Thu Sep  5 12:48 - crash (3+07:41)
reboot   system boot  5.15.153.1-micro Thu Sep  5 12:48   still running
drpremna pts/1                         Tue Sep  3 10:27 - crash (2+02:21)
reboot   system boot  5.15.153.1-micro Tue Sep  3 10:27   still running
drpremna pts/1                         Tue Aug 20 10:23 - crash (14+00:03)
reboot   system boot  5.15.153.1-micro Tue Aug 20 10:23   still running
drpremna pts/1                         Tue Aug 13 12:40 - crash (6+21:42)
reboot   system boot  5.15.153.1-micro Tue Aug 13 12:40   still running
```

SHELL SCRIPT PROGRAMMING

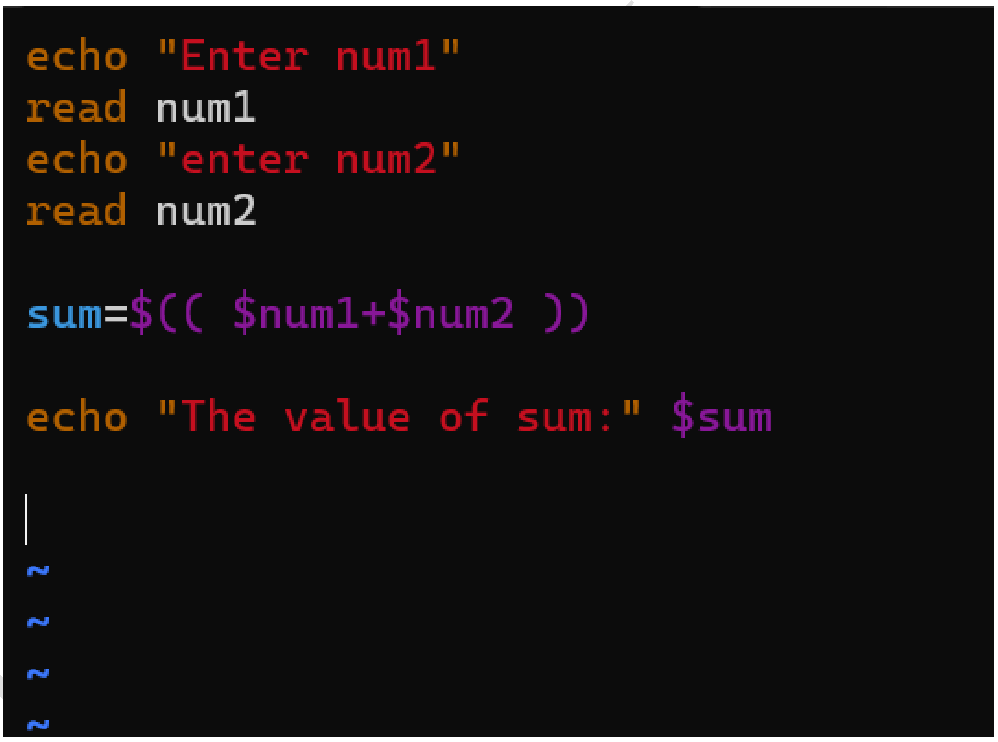
Prepared By: Dr. Premnarayan Arya

1. **write shell script programs to print name which input by the user.**



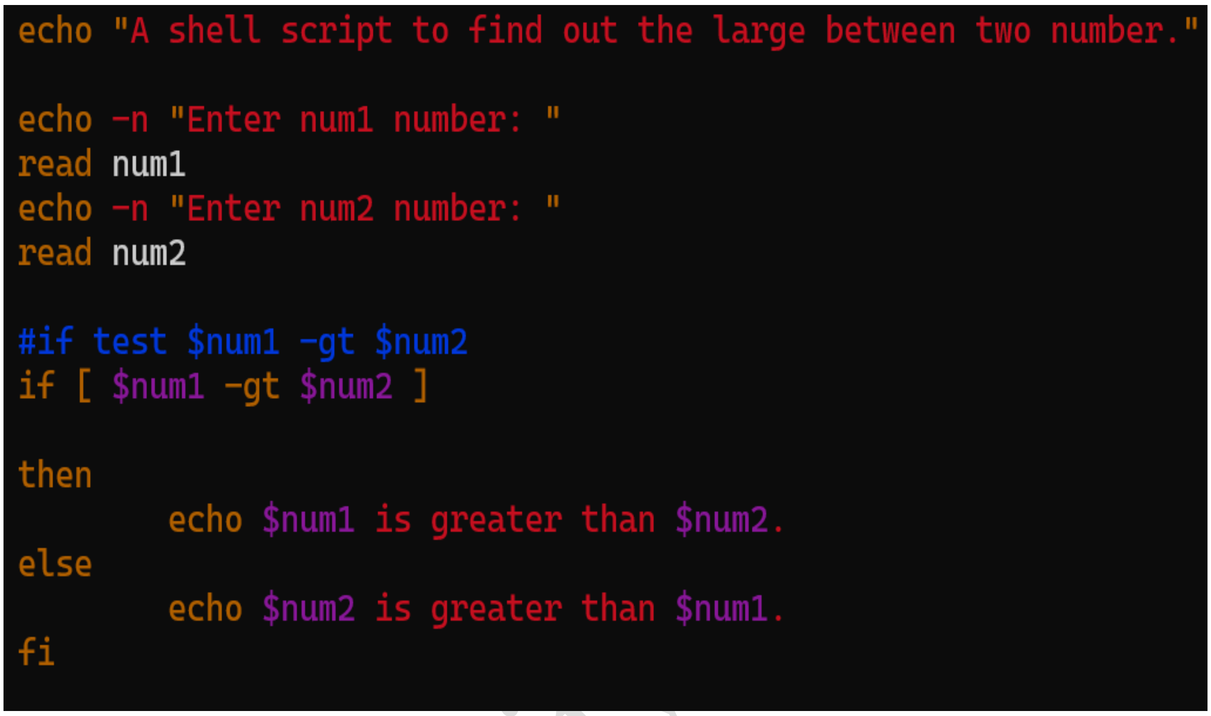
```
echo "Enter Your Name: "
read name
echo "Your name is: Dr Premnarayan Arya:" $ $name

~
~
~
~
~
~
~
```

2. **write shell script programs to sum of two numbers.**



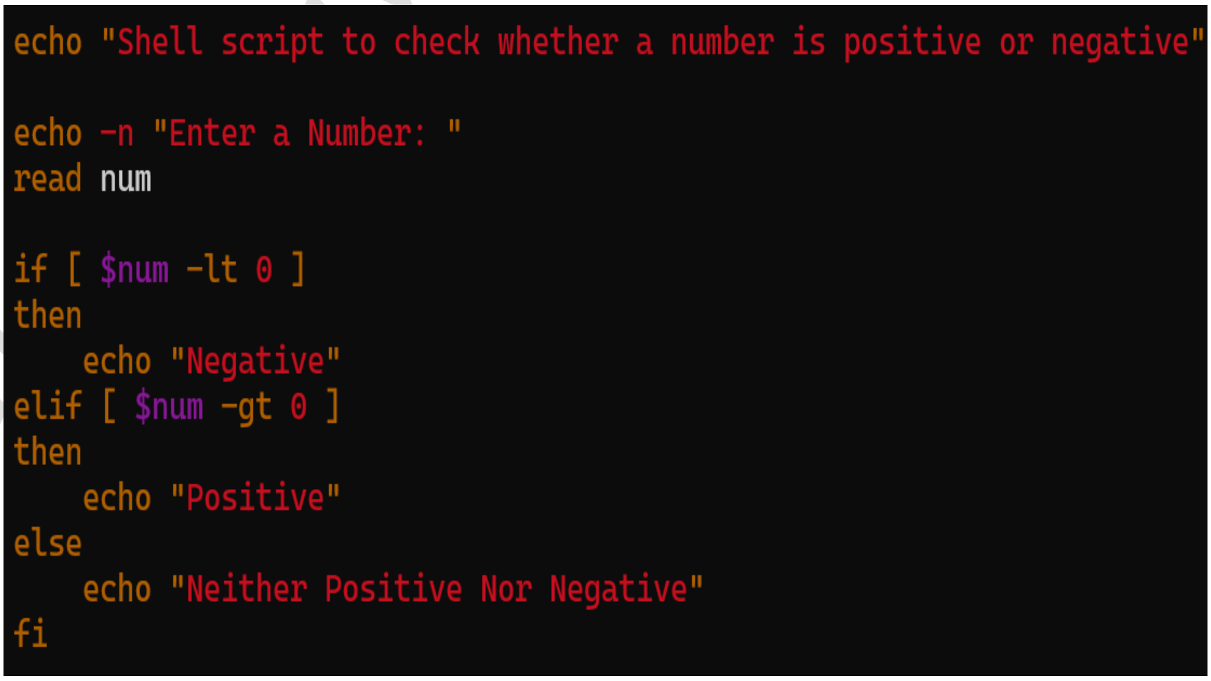
```
echo "Enter num1"
read num1
echo "enter num2"
read num2

sum=$(( $num1+$num2 ))

echo "The value of sum:" $sum


~
~
~
~
```

3. **write shell script programs to print greater number among two numbers.**



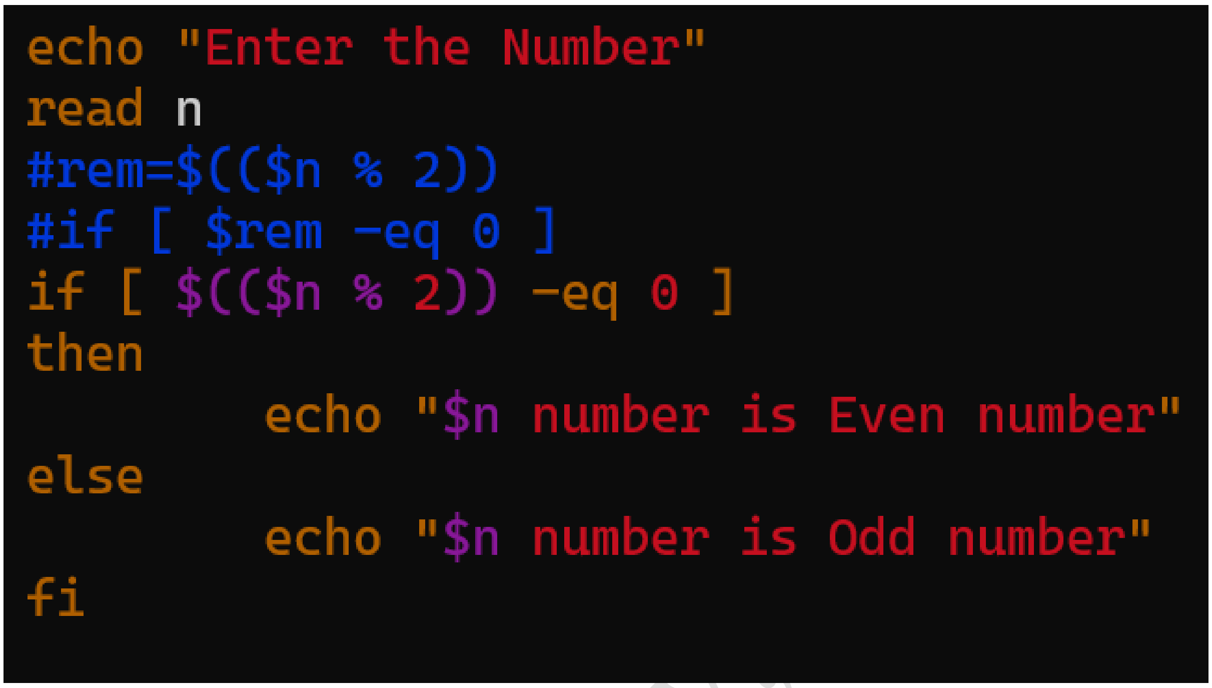
```
echo "A shell script to find out the large between two number."

echo -n "Enter num1 number: "
read num1
echo -n "Enter num2 number: "
read num2

#if test $num1 -gt $num2
if [ $num1 -gt $num2 ]

then
        echo $num1 is greater than $num2.
else
        echo $num2 is greater than $num1.
fi
```

4. **write shell script programs to check whether a number is positive or negative.**



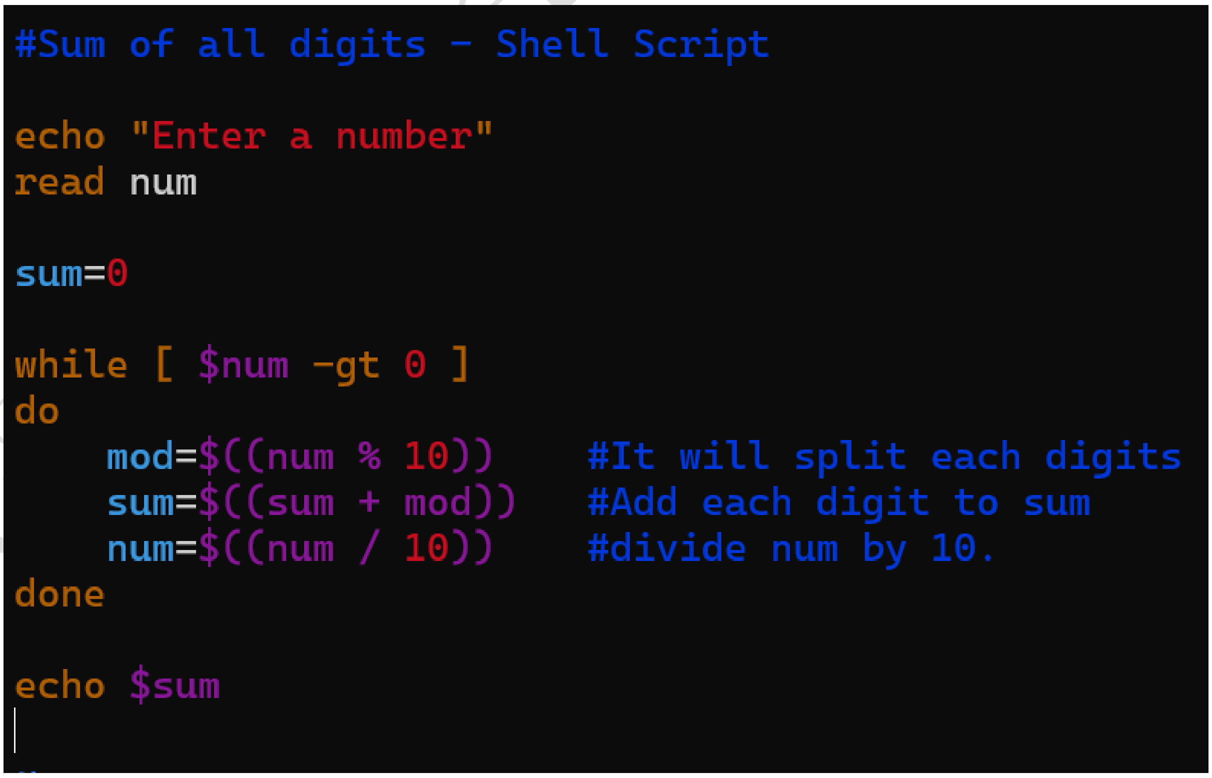
```
echo "Shell script to check whether a number is positive or negative"

echo -n "Enter a Number: "
read num

if [ $num -lt 0 ]
then
    echo "Negative"
elif [ $num -gt 0 ]
then
    echo "Positive"
else
    echo "Neither Positive Nor Negative"
fi
```

5. **write shell script programs to check entered number is odd or even.**



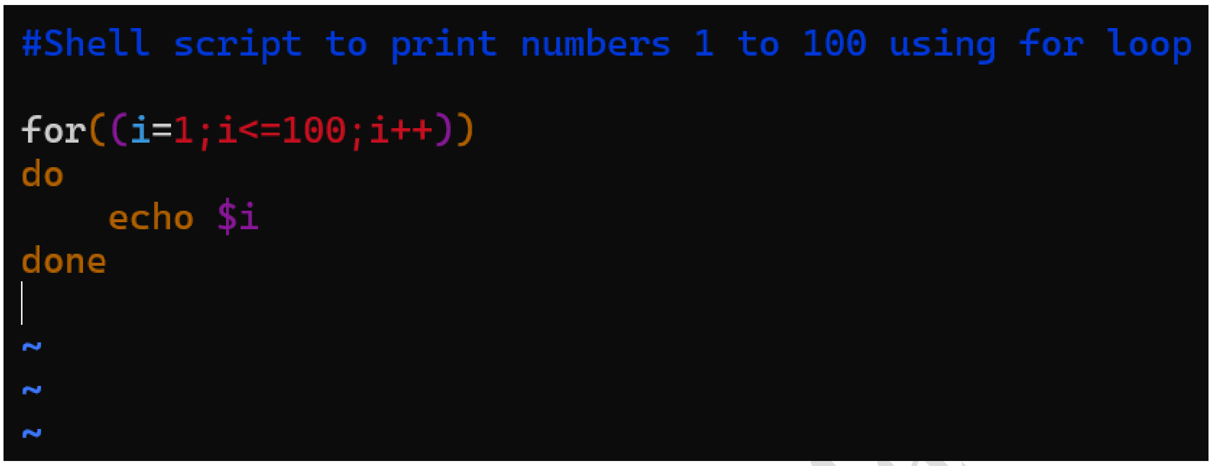
```
echo "Enter the Number"
read n
#rem=$(($n % 2))
#if [ $rem -eq 0 ]
if [ $(($n % 2)) -eq 0 ]
then
        echo "$n number is Even number"
else
        echo "$n number is Odd number"
fi
```

6. **write shell script programs to sum of all digits enter by the user.**



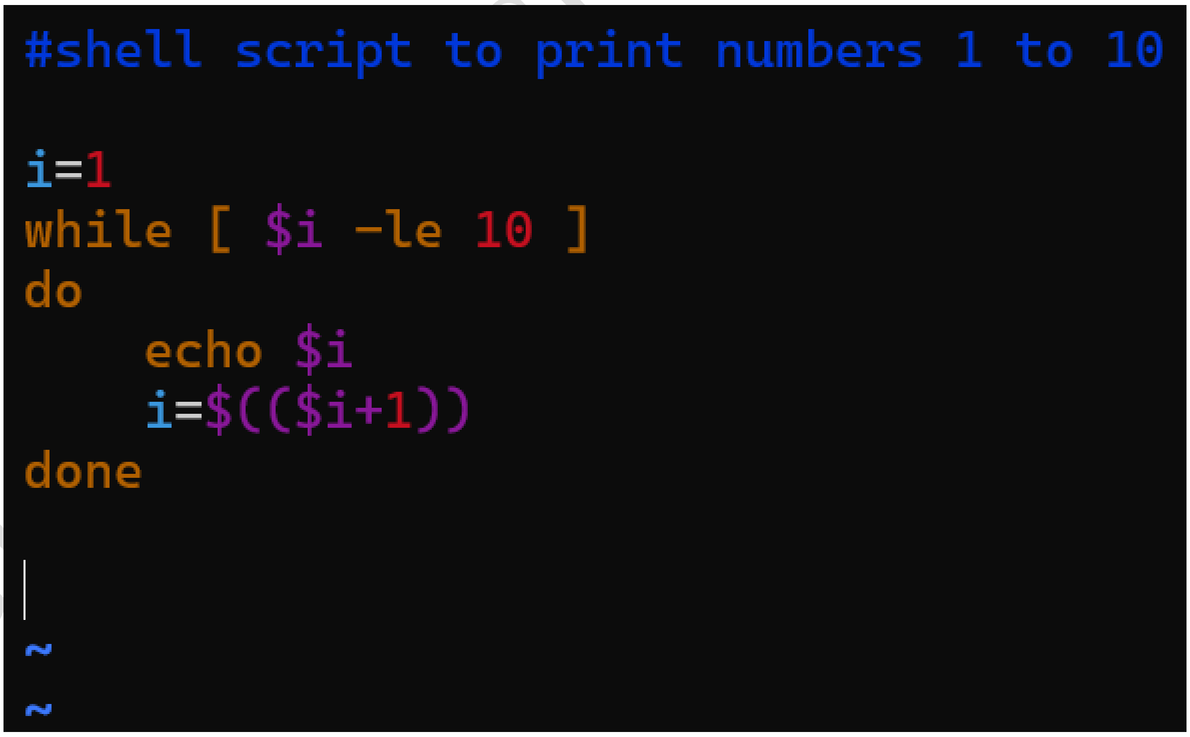
```
#Sum of all digits - Shell Script

echo "Enter a number"
read num

sum=0

while [ $num -gt 0 ]
do
    mod=$((num % 10))    #It will split each digits
    sum=$((sum + mod))   #Add each digit to sum
    num=$((num / 10))    #divide num by 10.
done

echo $sum
```

7. **write shell script programs to print numbers from 1 to 100 using for loop**



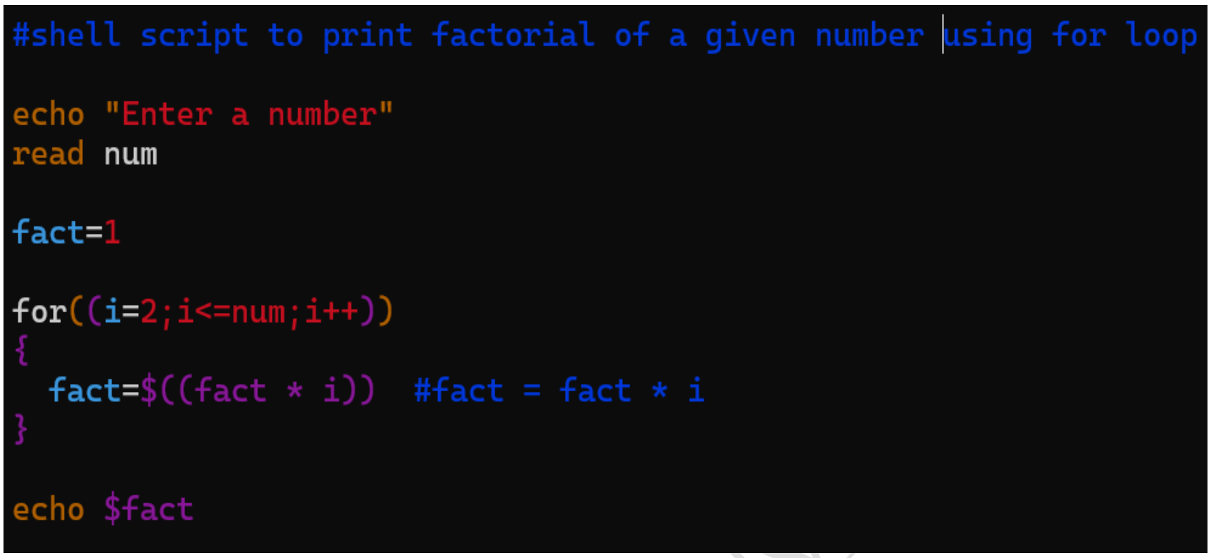
```
#Shell script to print numbers 1 to 100 using for loop

for((i=1;i<=100;i++))
do
    echo $i
done

~
~
~
```

8. **write shell script programs to print numbers from 1 to 10 using while loop.**



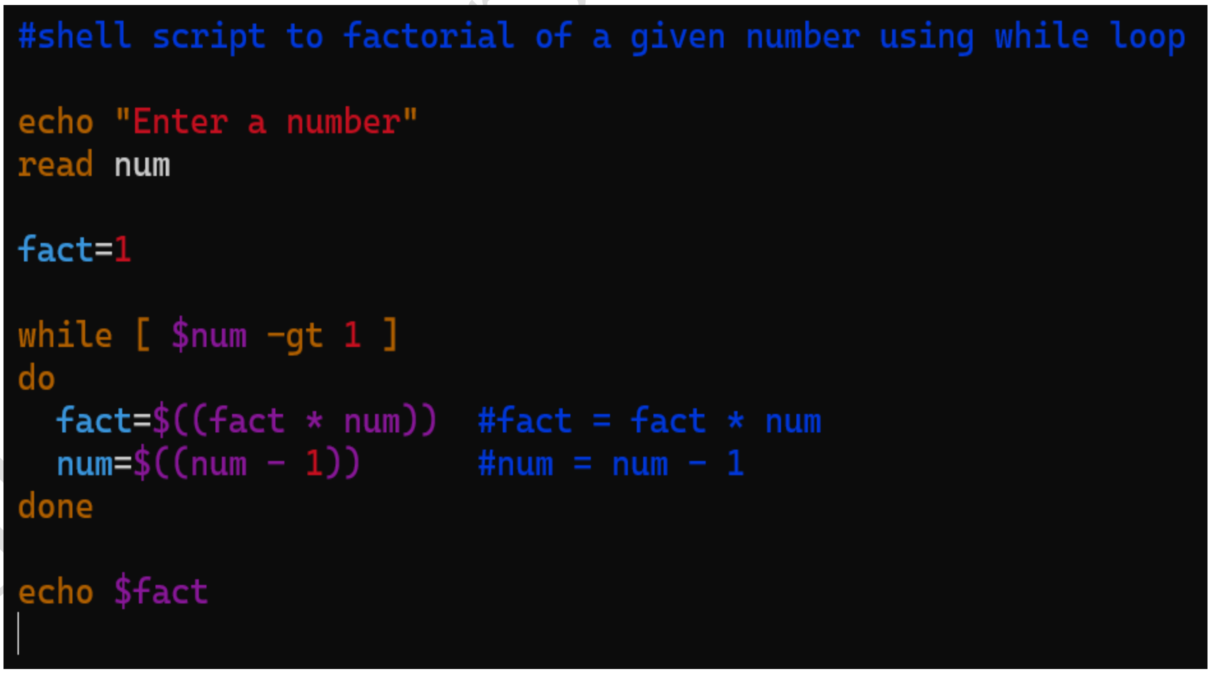
```
#shell script to print numbers 1 to 10

i=1
while [ $i -le 10 ]
do
    echo $i
    i=$(($i+1))
done


~
~
```

9. **write shell script programs to print factorial of a given number using for loop.**



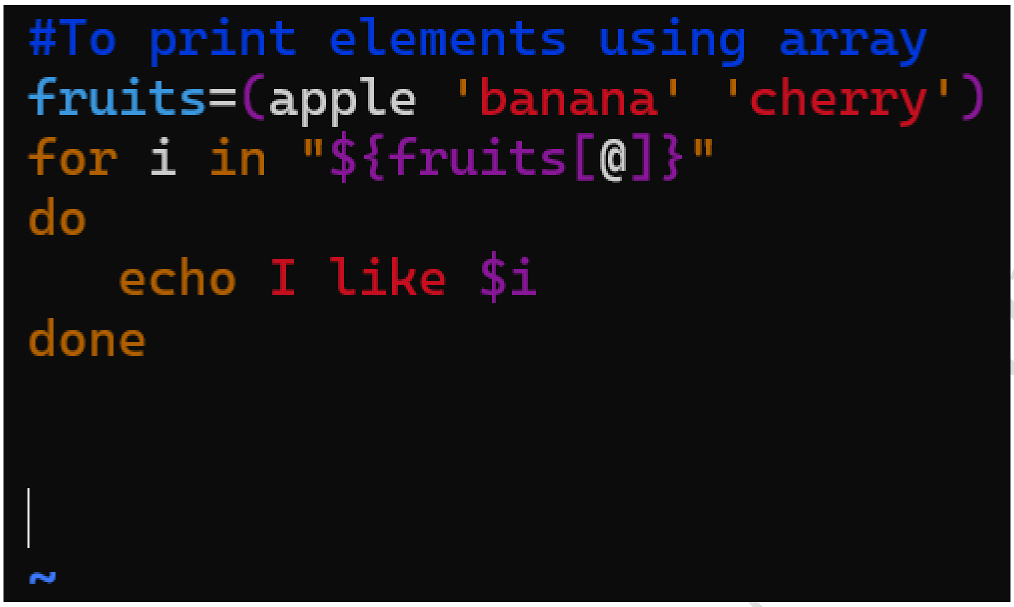
```
#shell script to print factorial of a given number using for loop

echo "Enter a number"
read num

fact=1

for((i=2;i<=num;i++))
{
  fact=$((fact * i))  #fact = fact * i
}

echo $fact
```

10. **write shell script programs to print factorial of a given number using while loop.**



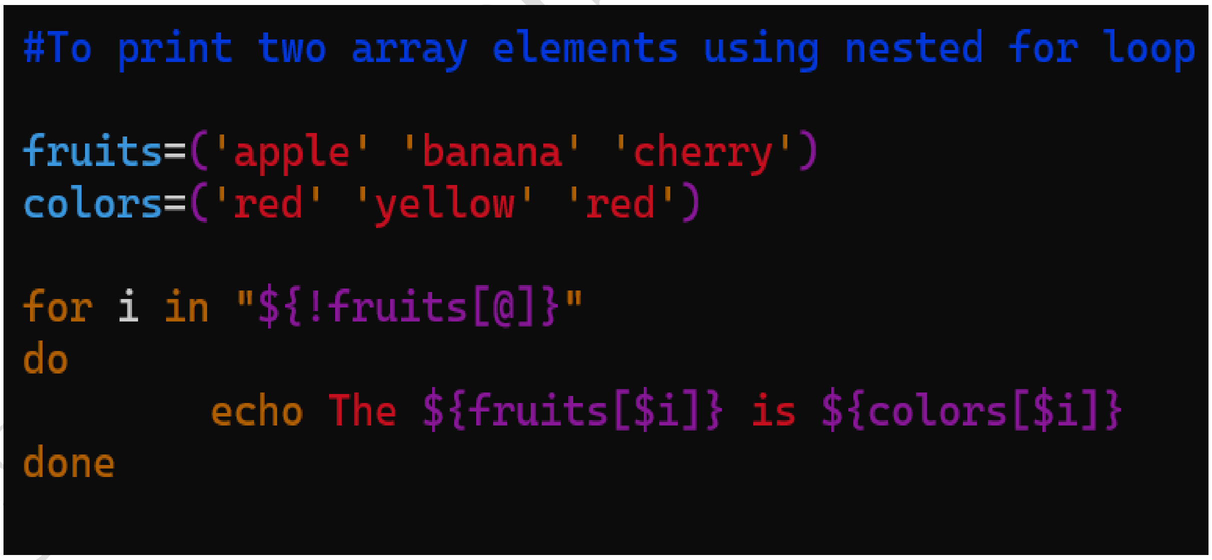
```
#shell script to factorial of a given number using while loop

echo "Enter a number"
read num

fact=1

while [ $num -gt 1 ]
do
  fact=$((fact * num))  #fact = fact * num
  num=$((num - 1))      #num = num - 1
done

echo $fact
```

11. write shell script programs to print elements using array.



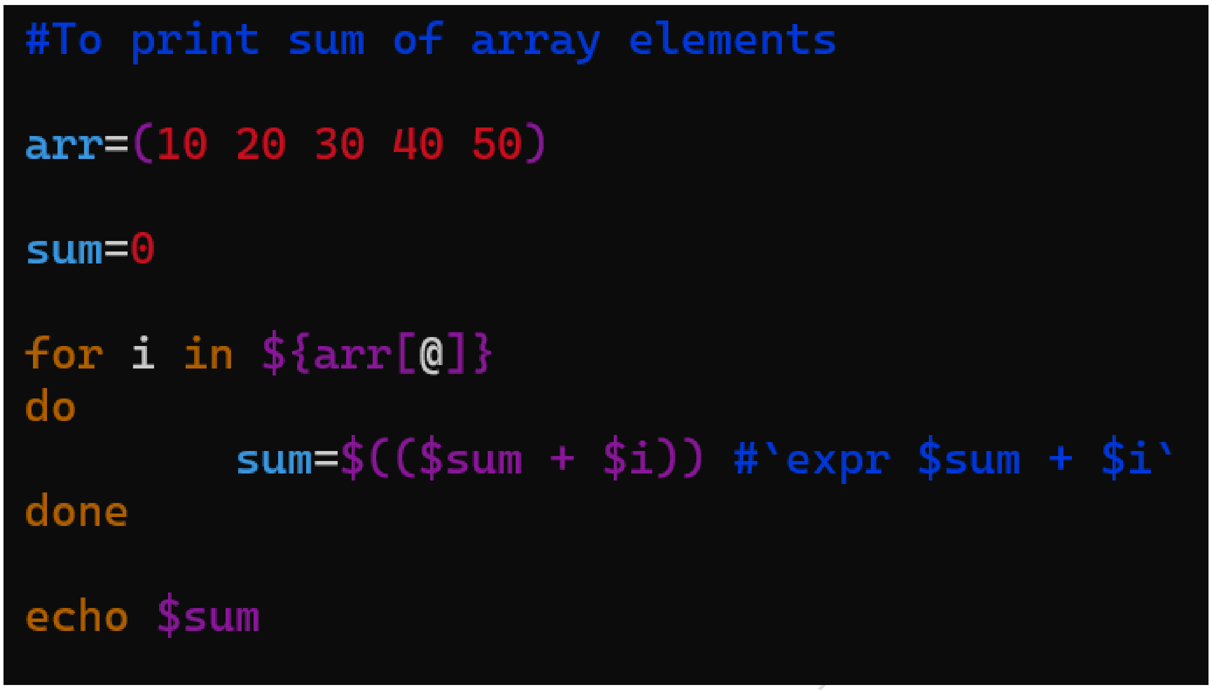
```
#To print elements using array
fruits=(apple 'banana' 'cherry')
for i in "${fruits[@]}"
do
   echo I like $i
done


~
```

12. write shell script programs to print two array elements using nested for loop.



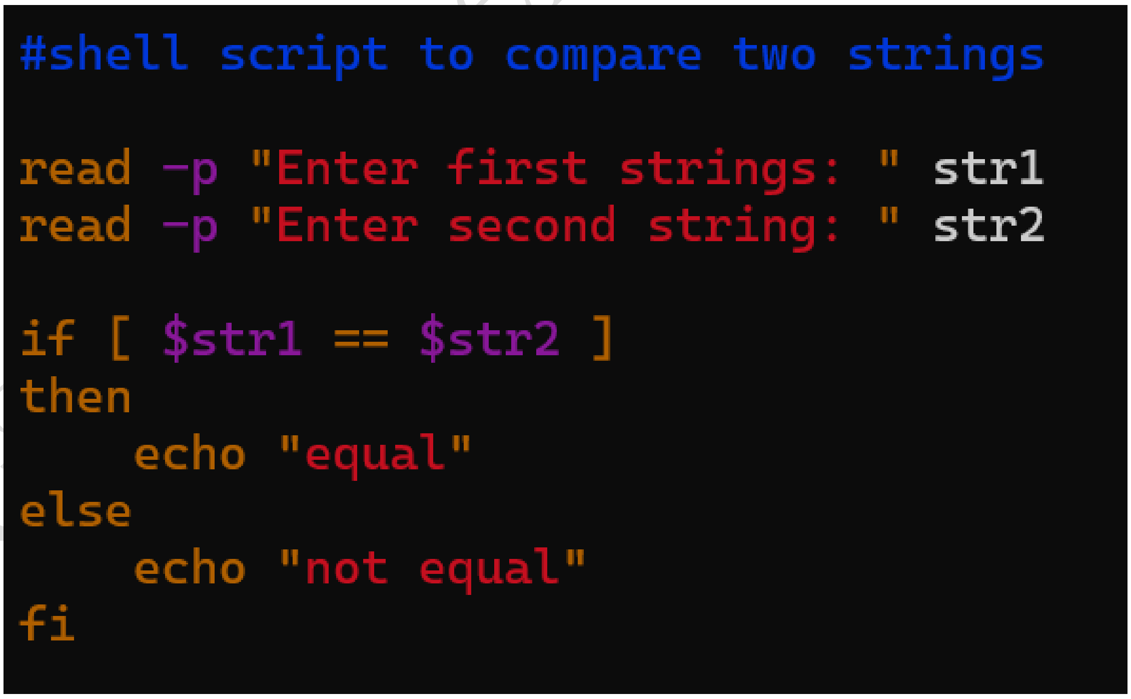
```
#To print two array elements using nested for loop

fruits=('apple' 'banana' 'cherry')
colors=('red' 'yellow' 'red')

for i in "${!fruits[@]}"
do
        echo The ${fruits[$i]} is ${colors[$i]}
done
```

13.write shell script programs to print sum of array elements.



```
#To print sum of array elements

arr=(10 20 30 40 50)

sum=0

for i in ${arr[@]}
do
        sum=$(($sum + $i)) #`expr $sum + $i`
done

echo $sum
```

14.write shell script programs to compare two strings are equal or not equal.



```
#shell script to compare two strings

read -p "Enter first strings: " str1
read -p "Enter second string: " str2

if [ $str1 == $str2 ]
then
    echo "equal"
else
    echo "not equal"
fi
```

CPU SHEDULING ALGORITHMS PROGRAMMING In SHELL SCRIPT LANGUAGE

Prepared By: Dr. Premnarayan Arya

1. Implement FCFS CPU Scheduling Algorithm in Shell Script language to determine the average waiting time and average turnaround time given n processes and their burst times.

Code:

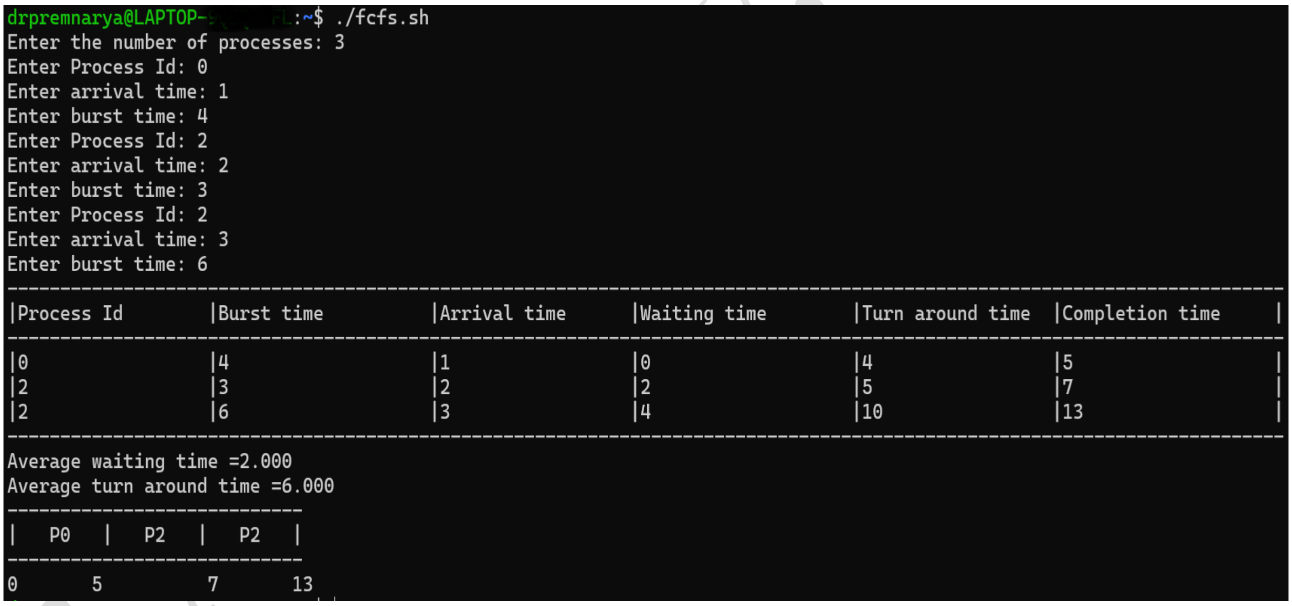
```
# Bash script to implement first come first served CPU scheduling algorithm.

sort(){
        for ((i = 0; i<$n; i++))
        do

           for((j = 0; j<`expr $n - $i - 1`; j++))
           do

                 if [ ${arrival_time[j]} -gt ${arrival_time[$((j+1))]} ]
                 then
                    # swap
                    temp=${arrival_time[j]}
                    arrival_time[$j]=${arrival_time[$((j+1))]}
                    arrival_time[$((j+1))]=$temp
                    temp=${burst_time[j]}
                    burst_time[$j]=${burst_time[$((j+1))]}
                    burst_time[$((j+1))]=$temp
                    temp=${pid[j]}
                    pid[$j]=${pid[$((j+1))]}
                    pid[$((j+1))]=$temp
                 elif [ ${arrival_time[j]} -eq ${arrival_time[$((j+1))]} ]
                 then
                         if [ ${pid[j]} -eq ${pid[$((j+1))]} ]
                         then
                            temp=${arrival_time[j]}
```

```
                        arrival_time[$j]=${arrival_time[$((j+1))]}
                        arrival_time[$((j+1))]=$temp
                        temp=${burst_time[j]}
                        burst_time[$j]=${burst_time[$((j+1))]}
                        burst_time[$((j+1))]=$temp
                        temp=${pid[j]}
                        pid[$j]=${pid[$((j+1))]}
                        pid[$((j+1))]=$temp
                      fi
                fi
            done
        done
}

border(){
        z=121
        for ((i=0; i<$z; i++))
        do
        echo -n "-"
        done
        echo ""
}

findWaitingTime(){
        service_time[0]=0
        waiting_time[0]=0
        for ((i=1; i<$n; i++))
        do
                z=1
                y=`expr $i - $z`
                service_time[$i]=`expr ${service_time[$y]} + ${burst_time[$y]} `
                waiting_time[$i]=`expr ${service_time[$i]} - ${arrival_time[$i]}`
                if [ ${waiting_time[$i]} -lt 0 ]
                then
```

```
                        waiting_time[$i]=0
                fi
        done
}

findTurnAroundTime(){
        for ((i=0; i<$n; i++))
        do
                tat[$i]=`expr ${waiting_time[$i]} + ${burst_time[$i]}`
        done
}

findAverageTime(){
        sort
        findWaitingTime
        findTurnAroundTime
        total_wt=0
        total_tat=0
        border
        printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time"
"Arrival time" "Waiting time" "Turn around time" "Completion time"
        border
        for ((i=0; i<$n; i++))
        do
                total_wt=`expr $total_wt + ${waiting_time[$i]}`
                total_tat=`expr ${tat[$i]} + $total_tat`
                completion_time=`expr ${arrival_time[$i]} + ${tat[$i]}`
                printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" ${pid[$i]}
${burst_time[$i]} ${arrival_time[$i]} ${waiting_time[$i]} ${tat[$i]} $completion_time
                #echo "${burst_time[$i]}     ${arrival_time[$i]}     ${waiting_time[$i]}
${tat[$i]}        $completion_time"
        done
        border
        #avgwt=`echo "scale=3; $total_wt / $n" | bc`
        echo -n "Average waiting time ="
    printf %.3f\\n "$(($total_wt / $n))"
        #avgtat=`echo "scale=3; $total_tat / $n" | bc`
        echo -n "Average turn around time ="
    printf %.3f\\n "$(($total_tat / $n))"

        for ((i=0; i<8*n+n+1; i++))
        do
                echo -n "-"
                done
                echo ""

        for ((i=0; i<$n; i++))
        do
                echo -n "|   "
                echo -n "P${pid[$i]}"
                echo -n "   "
        done
        echo "|"
        for ((i=0; i<8*n+n+1; i++))
        do
                echo -n "-"
                done
                echo ""
        echo -n "0      "
        for ((i=0; i<$n; i++))
        do
                echo -n "`expr ${arrival_time[$i]} + ${tat[$i]}`"
                echo -n "           "
        done
        echo ""
}
```

```
echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++))
do
echo -n "Enter Process Id: "
read pid[$i]
echo -n "Enter arrival time: "
read arrival_time[$i]
echo -n "Enter burst time: "
read burst_time[$i]
done
findAverageTime
```

Output:



```
drpremnarya@LAPTOP-[redacted]:~$ ./fcfs.sh
Enter the number of processes: 3
Enter Process Id: 0
Enter arrival time: 1
Enter burst time: 4
Enter Process Id: 2
Enter arrival time: 2
Enter burst time: 3
Enter Process Id: 2
Enter arrival time: 3
Enter burst time: 6
------------------------------------------------------------------------------------------------------------------------
|Process Id         |Burst time          |Arrival time      |Waiting time        |Turn around time  |Completion time     |
------------------------------------------------------------------------------------------------------------------------
|0                  |4                   |1                 |0                   |4                 |5                   |
|2                  |3                   |2                 |2                   |5                 |7                   |
|2                  |6                   |3                 |4                   |10                |13                  |
------------------------------------------------------------------------------------------------------------------------
Average waiting time =2.000
Average turn around time =6.000
----------------------------
|   P0    |   P2    |   P2    |
----------------------------
0       5          7        13
```

2. Implement Priority CPU Scheduling Algorithm in Shell Script programming language to determine the average waiting time and average turnaround time given n processes and their burst times.

Code:

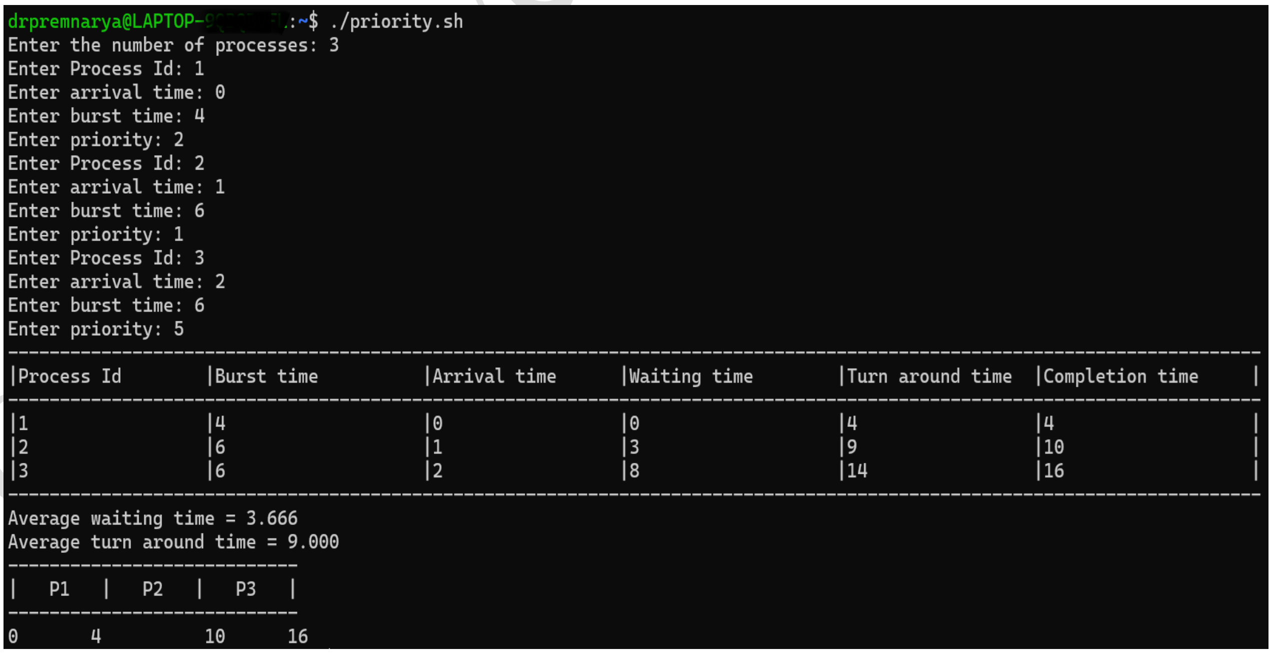
```
# Bash Script to implement Priority Scheduling Algorithm (non pre-emptive)

border(){
        z=121
        for ((i=0; i<$z; i++))
        do
           echo -n "-"
        done
        echo ""
}
arrangeArrival(){
        z=1
        for ((i=0; i<$n; i++))
        do
                for ((j=i+1; j<$n; j++))
                do
                        if [ ${arrival_time[$i]} -gt ${arrival_time[$j]} ]
                        then
                           temp=${arrival_time[$j]}
                           arrival_time[$j]=${arrival_time[$i]}
                           arrival_time[$i]=$temp

                           temp=${burst_time[$j]}
                           burst_time[$j]=${burst_time[$i]}
                        burst_time[$i]=$temp

                        temp=${priority[$j]}
                        priority[$j]=${priority[$i]}
                        priority[$i]=$temp

                        temp=${pid[$j]}
                        pid[$j]=${pid[$i]}
                        pid[$i]=$temp
                    fi
            done
    done
}
arrangePriority(){
    z=1
    for ((i=0; i<$n; i++))
    do
            for ((j=i+1; j<$n; j++))
            do
                    if [ ${arrival_time[$i]} -eq ${arrival_time[$j]} ]
                    then
                            if [ ${priority[$i]} -gt ${priority[$j]} ]
                            then
                                temp=${arrival_time[$j]}
                                arrival_time[$j]=${arrival_time[$i]}
                                arrival_time[$i]=$temp
                                temp=${burst_time[$j]}
                                burst_time[$j]=${burst_time[$i]}
                                burst_time[$i]=$temp
                                temp=${priority[$j]}
                                priority[$j]=${priority[$i]}
                                priority[$i]=$temp

                                temp=${pid[$j]}
                                pid[$j]=${pid[$i]}
                                pid[$i]=$temp
                        fi
                fi
        done
    done
}
findWaitingTime(){
    service_time[0]=0
    waiting_time[0]=0
    for ((i=1; i<$n; i++))
    do
        z=1
        y=`expr $i - $z`
        service_time[$i]=`expr ${service_time[$y]} + ${burst_time[$y]} `
        waiting_time[$i]=`expr ${service_time[$i]} - ${arrival_time[$i]}`
        if [ ${waiting_time[$i]} -lt 0 ]
        then
            waiting_time[$i]=0
        fi
    done
}
```

```
findTurnAroundTime(){
        for ((i=0; i<$n; i++))
        do
                tat[$i]=`expr ${waiting_time[$i]} + ${burst_time[$i]}`
        done
}
echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++))
do
echo -n "Enter Process Id: "
read pid[$i]
echo -n "Enter arrival time: "
read arrival_time[$i]
echo -n "Enter burst time: "
read burst_time[$i]
echo -n "Enter priority: "
read priority[$i]
done
arrangeArrival
arrangePriority
findWaitingTime
findTurnAroundTime
total_wt=0
total_tat=0
border
printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time"
"Arrival time" "Waiting time" "Turn around time" "Completion time"
border
```

```
for ((i=0; i<$n; i++))
do
        total_wt=`expr $total_wt + ${waiting_time[$i]}`
        total_tat=`expr ${tat[$i]} + $total_tat`
        completion_time=`expr ${arrival_time[$i]} + ${tat[$i]}`
        printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" ${pid[$i]}
${burst_time[$i]} ${arrival_time[$i]} ${waiting_time[$i]} ${tat[$i]}
$completion_time
        #echo "${burst_time[$i]}     ${arrival_time[$i]}     ${waiting_time[$i]}
${tat[$i]}        $completion_time"
done
border
avgwt=`echo "scale=3; $total_wt / $n" | bc`
echo "Average waiting time = $avgwt"
avgtat=`echo "scale=3; $total_tat / $n" | bc`
echo "Average turn around time = $avgtat"


for ((i=0; i<8*n+n+1; i++))
do
        echo -n "-"
        done
        echo ""

for ((i=0; i<$n; i++))
do
        echo -n "|   "
        echo -n "P${pid[$i]}"
        echo -n "   "
```

```
done
echo "|"
for ((i=0; i<8*n+n+1; i++))
do
        echo -n "-"
        done
        echo ""
echo -n "0    "
for ((i=0; i<$n; i++))
do
        echo -n "`expr ${arrival_time[$i]} + ${tat[$i]}`        "
done
echo ""
```

Output:



```
drpremnarya@LAPTOP-9[redacted]:~$ ./priority.sh
Enter the number of processes: 3
Enter Process Id: 1
Enter arrival time: 0
Enter burst time: 4
Enter priority: 2
Enter Process Id: 2
Enter arrival time: 1
Enter burst time: 6
Enter priority: 1
Enter Process Id: 3
Enter arrival time: 2
Enter burst time: 6
Enter priority: 5
------------------------------------------------------------------------------------------------------------------------
|Process Id         |Burst time          |Arrival time      |Waiting time        |Turn around time  |Completion time    |
------------------------------------------------------------------------------------------------------------------------
|1                  |4                   |0                 |0                   |4                 |4                  |
|2                  |6                   |1                 |3                   |9                 |10                 |
|3                  |6                   |2                 |8                   |14                |16                 |
------------------------------------------------------------------------------------------------------------------------
Average waiting time = 3.666
Average turn around time = 9.000
----------------------------
|   P1    |   P2    |   P3    |
----------------------------
0       4          10       16
```

3. Implement Round Robin CPU Scheduling Algorithm in Shell Script programming language to determine the average waiting time and average turnaround time given n processes and their burst times.

Code:

```
# Bash script to implement Round Robin CPU scheduling algorithm.

sort(){
        for ((i = 0; i<$n; i++))
        do

           for((j = 0; j<`expr $n - $i - 1`; j++))
           do

                if [ ${arrival_time[j]} -gt ${arrival_time[$((j+1))]} ]
                then
                   # swap
                   temp=${arrival_time[j]}
                   arrival_time[$j]=${arrival_time[$((j+1))]}
                   arrival_time[$((j+1))]=$temp

                   temp=${burst_time[j]}
                   burst_time[$j]=${burst_time[$((j+1))]}
                   burst_time[$((j+1))]=$temp

                   temp=${pid[j]}
                   pid[$j]=${pid[$((j+1))]}
                   pid[$((j+1))]=$temp

                   temp=${burst_time_copy[$j]}
```

```
            burst_time_copy[$j]=${burst_time_copy[$i]}
            burst_time_copy[$i]=$temp

            temp=${arrival_time_copy[$j]}
            arrival_time_copy[$j]=${arrival_time_copy[$i]}
            arrival_time_copy[$i]=$temp

        elif [ ${arrival_time[j]} -eq ${arrival_time[$((j+1))]} ]
        then
                if [ ${pid[j]} -eq ${pid[$((j+1))]} ]
                then
                    temp=${arrival_time[j]}
                    arrival_time[$j]=${arrival_time[$((j+1))]}
                    arrival_time[$((j+1))]=$temp

                    temp=${burst_time[j]}
                    burst_time[$j]=${burst_time[$((j+1))]}
                    burst_time[$((j+1))]=$temp

                    temp=${pid[j]}
                    pid[$j]=${pid[$((j+1))]}
                    pid[$((j+1))]=$temp

                    temp=${burst_time_copy[$j]}
                    burst_time_copy[$j]=${burst_time_copy[$i]}
                    burst_time_copy[$i]=$temp

                    temp=${arrival_time_copy[$j]}
                    arrival_time_copy[$j]=${arrival_time_copy[$i]}
                    arrival_time_copy[$i]=$temp
                fi
                fi
            done
        done
}

border(){
        z=121
        for ((i=0; i<$z; i++))
        do
        echo -n "-"
        done
        echo ""
}

calcWaitingtime(){
        t=0
        arrival=0
        is_completed=0

        while [ $is_completed -eq 0 ]
        do
                is_completed=1
                for ((i=0; i<$n; i++))
                do
                        chart[$t]=`expr $i + 1`
                        if [ ${burst_time[$i]} -gt 0 ]
                        then
                                is_completed=0
                                if [ ${burst_time[$i]} -gt $quantum -a ${arrival_time[$i]}
-le $arrival ]
                                then
```

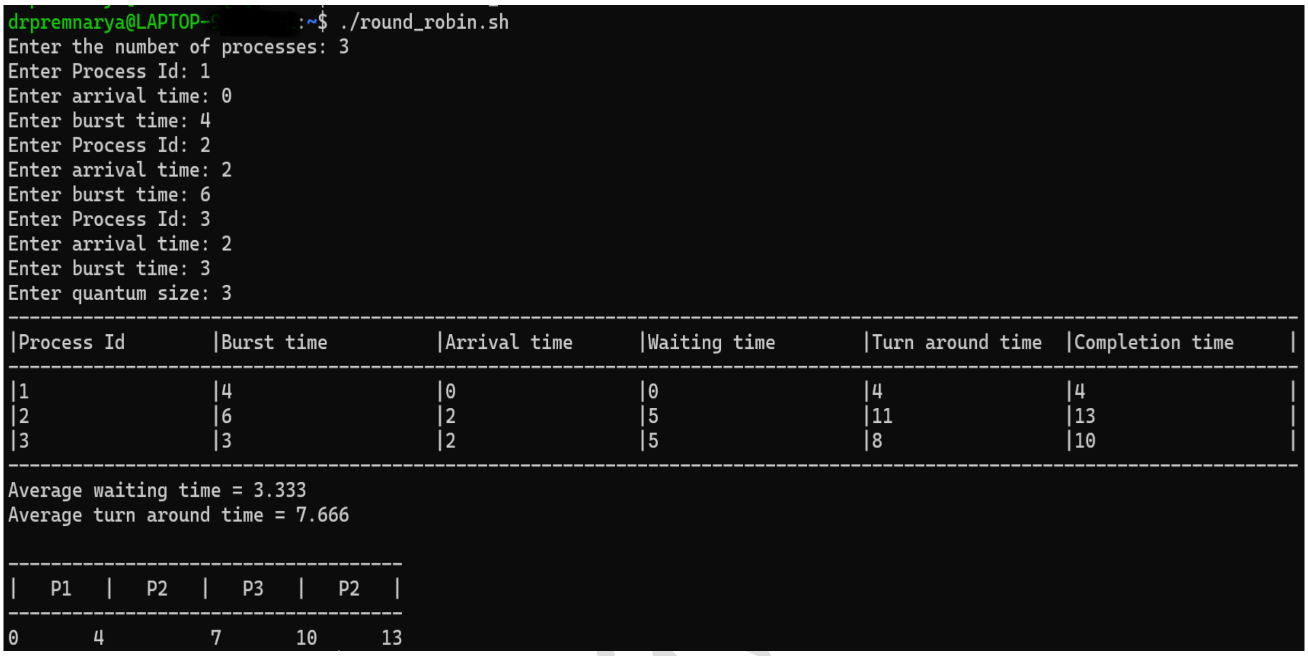
```
                                        for ((j=0; j<$quantum; j++))
                                        do
                                                chart[$t]=`expr $i + 1`
                                                ((t++))
                                        done
                                        #t=`expr $t + $quantum`
                                        burst_time[$i]=`expr ${burst_time[$i]} -
$quantum`
                                        ((arrival++))
                                else
                                        if [ ${arrival_time[$i]} -le $arrival ]
                                        then
                                                ((arrival++))
                                                for ((j=0; j<${burst_time[$i]}; j++))
                                                do
                                                        chart[$t]=`expr $i + 1`
                                                        ((t++))
                                                done
                                                #t=`expr $t + ${burst_time[$i]}`
                                                burst_time[$i]=0
                                                completion_time[$i]=$t
                                        fi
                                fi
                        fi
                done
                if [ $is_completed -eq 1 ]
                then
                        h=$t
                fi
        done
```

```
	for ((i=0; i<$n; i++))
	do
		tat[$i]=`expr ${completion_time[$i]} - ${arrival_time_copy[$i]}`
		waiting_time[$i]=`expr ${tat[$i]} - ${burst_time_copy[$i]}`
	done

	total_wt=0
	total_tat=0
	border
	printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time" 
"Arrival time" "Waiting time" "Turn around time" "Completion time"
	border
	for ((i=0; i<$n; i++))
	do
		total_wt=`expr $total_wt + ${waiting_time[$i]}`
		total_tat=`expr ${tat[$i]} + $total_tat`
		completion_time=`expr ${arrival_time[$i]} + ${tat[$i]}`
		printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" ${pid[$i]} 
${burst_time_copy[$i]} ${arrival_time_copy[$i]} ${waiting_time[$i]} ${tat[$i]} 
$completion_time
		#echo "${burst_time[$i]}     ${arrival_time[$i]}     ${waiting_time[$i]}     
${tat[$i]}        ${completion_time[$i]}"
	done
	border
	avgwt=`echo "scale=3; $total_wt / $n" | bc`
	echo "Average waiting time = $avgwt"
	avgtat=`echo "scale=3; $total_tat / $n" | bc`
	echo "Average turn around time = $avgtat"


	count_cols=1
```

```
cols_id[0]=${chart[0]}
cols[0]=0
j=1
for ((i=1; i<$h; i++))
do
        if [ ${chart[$i]} -ne ${chart[`expr $i - 1`]} ]
        then
                ((count_cols++))
                cols[$j]=$i
                cols_id[$j]=${chart[$i]}
                ((j++))
        fi
done

echo ""

for ((i=0; i<8*count_cols+count_cols+1; i++))
do
        echo -n "-"
        done
        echo ""

for ((i=0; i<$count_cols; i++))
do
        echo -n "|   "
        echo -n "P${cols_id[$i]}"
        echo -n "   "
done
echo "|"
for ((i=0; i<8*count_cols+count_cols+1; i++))
do
```

```
                echo -n "-"
                done
                echo ""
        echo -n "0      "
        for ((i=1; i<$count_cols; i++))
        do
                echo -n "${cols[$i]}"
                echo -n "           "
        done
        echo -n "$h"
        echo ""
}

echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++))
do
echo -n "Enter Process Id: "
read pid[$i]
echo -n "Enter arrival time: "
read arrival_time[$i]
arrival_time_copy[$i]=${arrival_time[$i]}
echo -n "Enter burst time: "
read burst_time[$i]
burst_time_copy[$i]=${burst_time[$i]}
done
echo -n "Enter quantum size: "
read quantum
sort
calcWaitingtime
```

Output:



```
drpremnarya@LAPTOP-[redacted]:~$ ./round_robin.sh
Enter the number of processes: 3
Enter Process Id: 1
Enter arrival time: 0
Enter burst time: 4
Enter Process Id: 2
Enter arrival time: 2
Enter burst time: 6
Enter Process Id: 3
Enter arrival time: 2
Enter burst time: 3
Enter quantum size: 3
------------------------------------------------------------------------------------------------------------------------
|Process Id         |Burst time          |Arrival time       |Waiting time        |Turn around time  |Completion time    |
------------------------------------------------------------------------------------------------------------------------
|1                  |4                   |0                  |0                   |4                 |4                  |
|2                  |6                   |2                  |5                   |11                |13                 |
|3                  |3                   |2                  |5                   |8                 |10                 |
------------------------------------------------------------------------------------------------------------------------
Average waiting time = 3.333
Average turn around time = 7.666

-------------------------------------
|   P1    |   P2    |   P3    |   P2    |
-------------------------------------
0       4          7       10       13
```

4. Implement Shortest Job First (Non-Pre-emptive) CPU Scheduling Algorithm in Shell Script programming language to determine the average waiting time and average turnaround time given n processes and their burst times.

Code:

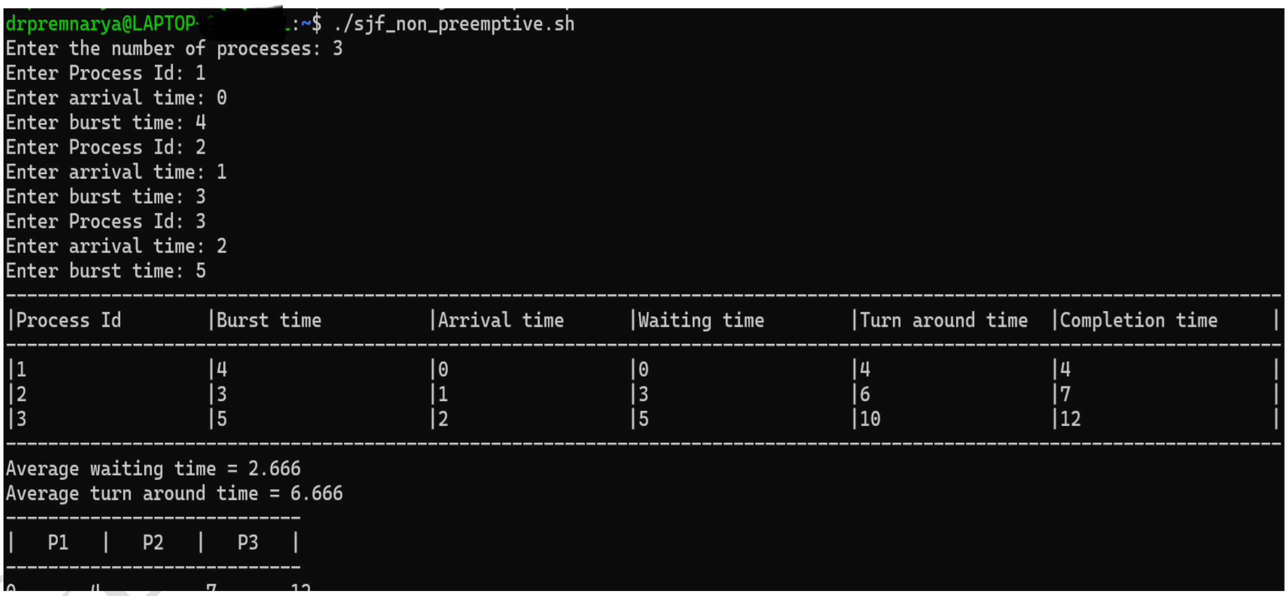
```
# Bash Script to implement Shortest Job First Scheduling Algorithm (non pre-emptive)

border(){
        z=121
        for ((i=0; i<$z; i++))
        do
        echo -n "-"
        done
        echo ""
}

arrangeArrival(){
        z=1
        for ((i=0; i<$n; i++))
        do
                for ((j=i+1; j<$n; j++))
                do
                        if [ ${arrival_time[$i]} -gt ${arrival_time[$j]} ]
                        then
                          temp=${arrival_time[$j]}
                          arrival_time[$j]=${arrival_time[$i]}
                          arrival_time[$i]=$temp

                          temp=${burst_time[$j]}
                          burst_time[$j]=${burst_time[$i]}
                          burst_time[$i]=$temp
                          temp=${pid[$j]}
                          pid[$j]=${pid[$i]}
                          pid[$i]=$temp
                        fi
                done
        done
}

arrangeBurst(){
        z=1
        for ((i=0; i<$n; i++))
        do
                for ((j=i+1; j<$n; j++))
                do
                        if [ ${arrival_time[$i]} -eq ${arrival_time[$j]} ]
                        then
                                if [ ${burst_time[$i]} -gt ${burst_time[$j]} ]
                                then
                                  temp=${arrival_time[$j]}
                                  arrival_time[$j]=${arrival_time[$i]}
                                  arrival_time[$i]=$temp

                                  temp=${burst_time[$j]}
                                  burst_time[$j]=${burst_time[$i]}
                                  burst_time[$i]=$temp

                                  temp=${pid[$j]}
                                  pid[$j]=${pid[$i]}
                                  pid[$i]=$temp
                                fi
                        fi
                done
        done
}
```

```
completionTime(){
        completion_time[0]=`expr ${arrival_time[0]} + ${burst_time[0]}`
        tat[0]=`expr ${completion_time[0]} - ${arrival_time[0]}`
        waiting_time[0]=`expr ${tat[0]} - ${burst_time[0]}`
        for ((i=1; i<$n; i++))
        do
                temp=${completion_time[`expr $i - 1`]}
                low=${burst_time[$i]}
                for ((j=i; j<$n; j++))
                do
                        if [ $temp -ge ${arrival_time[$j]} ]
                        then
                                if [ $low -ge ${burst_time[$j]} ]
                                then
                                        low=${burst_time[$j]}
                                        val=$j
                                fi
                        fi
                done
                completion_time[$val]=`expr $temp + ${burst_time[$val]}`
                tat[$val]=`expr ${completion_time[$val]} - ${arrival_time[$val]}`
                waiting_time[$val]=`expr ${tat[$val]} - ${burst_time[$val]}`

                if [ $val -ne $i ]
                then
                        temp=${arrival_time[$val]}
                        arrival_time[$val]=${arrival_time[$i]}
                        arrival_time[$i]=$temp

                        temp=${burst_time[$val]}
                        burst_time[$val]=${burst_time[$i]}
                        burst_time[$i]=$temp
                        temp=${pid[$val]}
                        pid[$val]=${pid[$i]}
                        pid[$i]=$temp

                        temp=${completion_time[$val]}
                        completion_time[$val]=${completion_time[$i]}
                        completion_time[$i]=$temp

                        temp=${waiting_time[$val]}
                        waiting_time[$val]=${waiting_time[$i]}
                        waiting_time[$i]=$temp

                        temp=${tat[$val]}
                        tat[$val]=${pid[$i]}
                        tat[$i]=$temp
                fi
        done
}

echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++))
do
echo -n "Enter Process Id: "
read pid[$i]
echo -n "Enter arrival time: "
read arrival_time[$i]
echo -n "Enter burst time: "
read burst_time[$i]
done
arrangeArrival
arrangeBurst
completionTime
total_wt=0
```

```
total_tat=0
border
printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time" "Arrival time"
"Waiting time" "Turn around time" "Completion time"
border
for ((i=0; i<$n; i++))
do
        total_wt=`expr $total_wt + ${waiting_time[$i]}`
        total_tat=`expr ${tat[$i]} + $total_tat`
        completion_time=`expr ${arrival_time[$i]} + ${tat[$i]}`
        printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" ${pid[$i]} ${burst_time[$i]}
${arrival_time[$i]} ${waiting_time[$i]} ${tat[$i]} $completion_time
        #echo "${burst_time[$i]}     ${arrival_time[$i]}     ${waiting_time[$i]}       ${tat[$i]}
$completion_time"
done
border
avgwt=`echo "scale=3; $total_wt / $n" | bc`
echo "Average waiting time = $avgwt"
avgtat=`echo "scale=3; $total_tat / $n" | bc`
echo "Average turn around time = $avgtat"


for ((i=0; i<8*n+n+1; i++))
do
        echo -n "-"
        done
        echo ""

for ((i=0; i<$n; i++))
do
        echo -n "|   "
        echo -n "P${pid[$i]}"
        echo -n "   "
done
```

```
echo "|"
for ((i=0; i<8*n+n+1; i++))
do
        echo -n "-"
        done
        echo ""
echo -n "0       "
for ((i=0; i<$n; i++))
do
        echo -n "`expr ${arrival_time[$i]} + ${tat[$i]}`        "
done
echo ""
```

Output:



```
drpremnarya@LAPTOP-[redacted]:~$ ./sjf_non_preemptive.sh
Enter the number of processes: 3
Enter Process Id: 1
Enter arrival time: 0
Enter burst time: 4
Enter Process Id: 2
Enter arrival time: 1
Enter burst time: 3
Enter Process Id: 3
Enter arrival time: 2
Enter burst time: 5
------------------------------------------------------------------------------------------------------------------------
|Process Id        |Burst time          |Arrival time      |Waiting time        |Turn around time  |Completion time     |
------------------------------------------------------------------------------------------------------------------------
|1                 |4                   |0                 |0                   |4                 |4                   |
|2                 |3                   |1                 |3                   |6                 |7                   |
|3                 |5                   |2                 |5                   |10                |12                  |
------------------------------------------------------------------------------------------------------------------------
Average waiting time = 2.666
Average turn around time = 6.666
----------------------------
|   P1   |   P2   |   P3   |
----------------------------
0       4        7        12
```

5. Implement SJF Pre-emptive / Shortest Remaining Time First (SRTF) CPU Scheduling Algorithm in Shell Script programming language to determine the average waiting time and average turnaround time given n processes and their burst times.

Code:

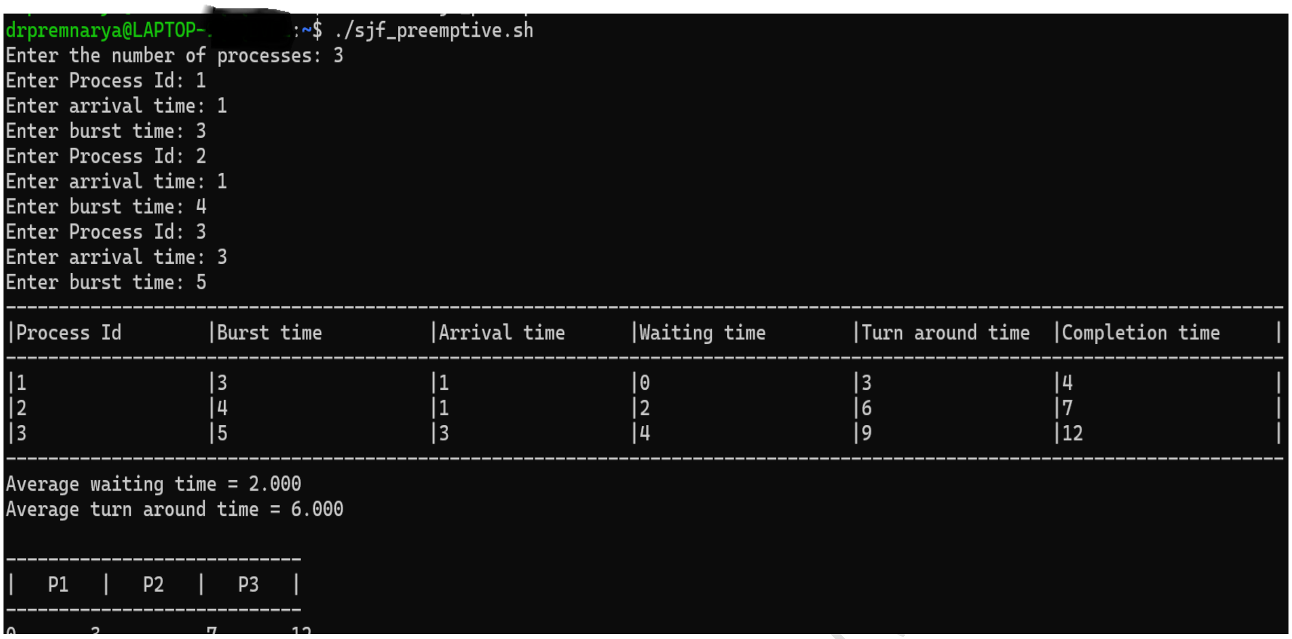
```
# Bash script to implement Shortest Job First Scheduling Algorithm (Pre-emptive)

border(){
        z=121
        for ((i=0; i<$z; i++))
        do
        echo -n "-"
        done
        echo ""
}

arrangeArrival(){
        z=1
        for ((i=0; i<$n; i++))
        do
                for ((j=i+1; j<$n; j++))
                do
                        if [ ${arrival_time[$i]} -gt ${arrival_time[$j]} ]
                        then
                          temp=${arrival_time[$j]}
                          arrival_time[$j]=${arrival_time[$i]}
                          arrival_time[$i]=$temp

                          temp=${burst_time[$j]}
                          burst_time[$j]=${burst_time[$i]}
                          burst_time[$i]=$temp
```

```
                    temp=${burst_time_copy[$j]}
                    burst_time_copy[$j]=${burst_time_copy[$i]}
                    burst_time_copy[$i]=$temp

                    temp=${pid[$j]}
                    pid[$j]=${pid[$i]}
                    pid[$i]=$temp
                fi
            done
        done
}

arrangeBurst(){
        z=1
        for ((i=0; i<$n; i++))
        do
            for ((j=i+1; j<$n; j++))
            do
                if [ ${arrival_time[$i]} -eq ${arrival_time[$j]} ]
                then
                    if [ ${burst_time[$i]} -gt ${burst_time[$j]} ]
                    then
                        temp=${arrival_time[$j]}
                        arrival_time[$j]=${arrival_time[$i]}
                        arrival_time[$i]=$temp

                        temp=${burst_time[$j]}
                        burst_time[$j]=${burst_time[$i]}
                        burst_time[$i]=$temp

                        temp=${burst_time_copy[$j]}
                        burst_time_copy[$j]=${burst_time_copy[$i]}
                        burst_time_copy[$i]=$temp
                        temp=${pid[$j]}
                        pid[$j]=${pid[$i]}
                        pid[$i]=$temp
                    fi
                fi
            done
        done
}

timecalc(){
        is_completed=0
        current_time=0
        cp=0
        count=0
        max=1000
        for ((i=0; i<$n; i++))
        do
            if [ ${arrival_time[$i]} -le $current_time ]
            then
                if [ ${burst_time[$i]} -lt $max ]
                then
                    if [ ${burst_time[$i]} -ne 0 ]
                    then
                        cp=$i
                        max=${burst_time[$i]}
                        if [ ${burst_time[$i]} -eq ${burst_time_copy[$i]} ]
                        then
                            waiting_time[$i]=$current_time
                        fi
                    fi
                fi
            fi
        done
```

```
while [ $is_completed -eq 0 ]
do
    if [ $count -eq $n ]
    then
        is_completed=1
        h=$current_time
    fi
    chart[$current_time]=`expr $cp + 1`
    ((current_time++))
    if [ ${burst_time[$cp]} -gt 0 ]
    then
        burst_time[$cp]=`expr ${burst_time[$cp]} - 1`
        max=${burst_time[$cp]}
        if [ ${burst_time[$cp]} -eq 0 ]
        then
            ((count++))
            completion_time[$cp]=$current_time
            max=1000
        fi
    fi
    prevcp=$cp
    for ((i=0; i<$n; i++))
    do
        if [ ${arrival_time[$i]} -le $current_time ]
        then
            if [ ${burst_time[$i]} -lt $max ]
            then
                if [ ${burst_time[$i]} -ne 0 ]
                then
                    cp=$i
                    max=${burst_time[$i]}
                fi
            fi
        fi
        done
        if [ $prevcp -ne $cp ]
        then
            waiting_time[$i]=$current_time
        fi

    done
    for ((i=0; i<$n; i++))
    do
        waiting_time[$i]=`expr ${completion_time[$i]} - ${arrival_time[$i]} -
${burst_time_copy[$i]}`
        if [ ${waiting_time[$i]} -lt 0 ]
        then
            waiting_time[$i]=0
        fi
        tat[$i]=`expr ${waiting_time[$i]} + ${burst_time_copy[$i]}`
    done
    total_wt=0
    total_tat=0
    border
    printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time"
"Arrival time" "Waiting time" "Turn around time" "Completion time"
    border
    for ((i=0; i<$n; i++))
    do
        total_wt=`expr $total_wt + ${waiting_time[$i]}`
        total_tat=`expr ${tat[$i]} + $total_tat`
        completion_time=`expr ${arrival_time[$i]} + ${tat[$i]}`
        printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" ${pid[$i]}
${burst_time_copy[$i]} ${arrival_time[$i]} ${waiting_time[$i]} ${tat[$i]}
$completion_time
        #echo "${burst_time[$i]}     ${arrival_time[$i]}     ${waiting_time[$i]}
${tat[$i]}      ${completion_time[$i]}"
    done
```

```
border
avgwt=`echo "scale=3; $total_wt / $n" | bc`
echo "Average waiting time = $avgwt"
avgtat=`echo "scale=3; $total_tat / $n" | bc`
echo "Average turn around time = $avgtat"

count_cols=1
cols_id[0]=${chart[0]}
cols[0]=0
j=1
for ((i=1; i<$h; i++))
do
        if [ ${chart[$i]} -ne ${chart[`expr $i - 1`]} ]
        then
                ((count_cols++))
                cols[$j]=$i
                cols_id[$j]=${chart[$i]}
                ((j++))
        fi
done

echo ""

for ((i=0; i<8*count_cols+count_cols+1; i++))
do
        echo -n "-"
        done
        echo ""

for ((i=0; i<$count_cols; i++))
do
        echo -n "|   "
        echo -n "P${cols_id[$i]}"
        echo -n "   "
        done
        echo "|"
        for ((i=0; i<8*count_cols+count_cols+1; i++))
        do
                echo -n "-"
                done
                echo ""
        echo -n "0       "
        for ((i=1; i<$count_cols; i++))
        do
                echo -n "${cols[$i]}"
                echo -n "            "
        done
        echo -n "$h"
        echo ""
}

echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++))
do
echo -n "Enter Process Id: "
read pid[$i]
echo -n "Enter arrival time: "
read arrival_time[$i]
echo -n "Enter burst time: "
read burst_time[$i]
burst_time_copy[$i]=${burst_time[$i]}
done
arrangeArrival
arrangeBurst
timecalc
```

Output:



```
drpremnarya@LAPTOP-[redacted]:~$ ./sjf_preemptive.sh
Enter the number of processes: 3
Enter Process Id: 1
Enter arrival time: 1
Enter burst time: 3
Enter Process Id: 2
Enter arrival time: 1
Enter burst time: 4
Enter Process Id: 3
Enter arrival time: 3
Enter burst time: 5
------------------------------------------------------------------------------------------------------------------------
|Process Id         |Burst time          |Arrival time       |Waiting time        |Turn around time  |Completion time    |
------------------------------------------------------------------------------------------------------------------------
|1                  |3                   |1                  |0                   |3                 |4                  |
|2                  |4                   |1                  |2                   |6                 |7                  |
|3                  |5                   |3                  |4                   |9                 |12                 |
------------------------------------------------------------------------------------------------------------------------
Average waiting time = 2.000
Average turn around time = 6.000

----------------------------
|   P1    |   P2    |   P3    |
----------------------------
0        3          7       12
```

What content should be covered from a placement perspective for Linux companies/industries in 2024?

Prepared By: Dr. Premnarayan Arya

For 2024 placements, companies expect students to have a comprehensive understanding of Linux, covering foundational to advanced topics. Here's a detailed breakdown of the content that students should study to meet industry expectations:

1. Introduction to Linux and Basic Commands

- **Linux Overview**: History, distributions (Debian, Red Hat, Ubuntu, CentOS), open-source principles.
- **Basic Commands**:
  - File and directory operations: ls, cd, cp, mv, rm, mkdir, touch.
  - Viewing file contents: cat, more, less, head, tail.
  - Text processing: grep, awk, sed, cut, sort, uniq, wc.
- **File System Hierarchy**: Understanding /home, /etc, /var, /usr, /bin, /tmp, etc.

2. File Management and Permissions

- **File Permissions**: Read, write, and execute permissions for files and directories; managing permissions using chmod, chown, chgrp.
- **File Types**: Regular files, directories, symbolic links, special files.
- **File Compression**: Using tar, gzip, bzip2, zip for archiving and compressing files.
- **Disk Usage**: Commands like df, du, and disk partitioning tools (fdisk, parted).

3. Shell Scripting and Automation

- **Shell Scripting Basics**: Writing and executing shell scripts using bash. Understanding variables, loops (for, while), conditionals (if, else), and functions.
- **Automation**: Using scripts to automate tasks like backups, file processing, log management.
- **Scheduled Tasks**: Using cron and at for automating repetitive tasks.

4. User and Group Management

- **User Management**: Commands like useradd, usermod, userdel, passwd for managing users.
- **Groups and Permissions**: Managing groups with groupadd, groupmod, groupdel. Assigning users to groups, managing group permissions.
- **User Environment**: Understanding user profiles (.bashrc, .bash_profile, /etc/profile).

5. System Administration and Process Management

- **System Monitoring**: Using top, htop, ps, vmstat, free to monitor system performance.
- **Process Management**: Starting, stopping, and managing processes using kill, killall, nice, renice.
- **Log Files**: Analyzing system logs in /var/log using journalctl, dmesg.

6. Package Management

- **Package Managers**: Using package managers like apt (Debian/Ubuntu), yum/dnf (Red Hat/CentOS) to install, update, and remove software packages.
- **Repositories**: Configuring repositories, adding PPAs (for Ubuntu), installing software from source.

7. Networking Fundamentals

- **Basic Networking**: Commands like ping, traceroute, netstat, ip, ifconfig, hostname, route.
- **Remote Access**: Configuring and using SSH for secure remote login, scp for file transfer, and sftp.
- **Network Services**: Configuring basic network services like DNS (bind), DHCP, web servers (Apache, Nginx), FTP servers.

8. Storage and File Systems

- **Disk Partitioning**: Creating and managing partitions with fdisk, parted.
- **File Systems**: Understanding different file systems (ext4, XFS, Btrfs). Mounting and unmounting file systems using mount, umount.
- **LVM (Logical Volume Manager)**: Basic understanding of LVM concepts and usage for managing disk space.
- **Disk Quotas**: Implementing disk quotas to manage user disk space usage.

9. Security and Access Control

- **File Permissions**: Managing file and directory permissions using chmod, chown, and ACLs.
- **Firewall Configuration**: Setting up and managing firewalls using iptables, firewalld.

- **System Hardening**: Disabling unnecessary services, securing SSH (key-based authentication, disabling root login).
- **SELinux and AppArmor**: Basics of implementing mandatory access controls for enhanced system security.

10. Virtualization and Containerization

- **Virtualization**: Introduction to virtualization using tools like KVM, VirtualBox.
- **Docker**: Basics of containerization, creating and managing containers, using Dockerfiles.
- **Kubernetes (Optional)**: Basic understanding of container orchestration, managing applications in Kubernetes clusters.

11. Cloud Computing and Linux in the Cloud

- **Cloud Platforms**: Overview of Linux usage in cloud environments like AWS, Google Cloud Platform, Azure.
- **CLI Tools**: Using cloud CLI tools (aws-cli, gcloud, az) to manage cloud instances.
- **Managing Linux Servers**: Deploying and managing Linux servers on cloud platforms.

12. Automation Tools and DevOps Practices

- **Ansible, Puppet, Chef**: Introduction to configuration management tools for automating infrastructure setup and management.
- **CI/CD Pipelines**: Basic understanding of setting up CI/CD using tools like Jenkins, GitLab CI/CD.
- **Version Control**: Using git for source code version control and collaboration.

13. System Maintenance and Troubleshooting

- **Log Management**: Using tools like logrotate to manage system logs, analyzing logs for troubleshooting.
- **Backup and Recovery**: Using rsync, tar, dd for data backup and recovery.
- **System Performance**: Using monitoring tools (top, htop, iostat, vmstat) to identify and troubleshoot performance issues.

14. Advanced Networking and Security (Optional)

- **Advanced Networking**: Use of tcpdump, Wireshark for network diagnostics, setting up VPNs (OpenVPN, WireGuard).
- **Security Best Practices**: Implementing security measures like SELinux, AppArmor, intrusion detection systems (fail2ban), SSL/TLS encryption.

15. Linux in Production Environments

- **Deployment**: Best practices for deploying and maintaining applications on Linux servers.
- **Monitoring Tools**: Use of Nagios, Zabbix, Prometheus for server and network monitoring.
- **Scaling**: Basics of load balancing and scaling applications using tools like HAProxy, Nginx.

Additional Skills and Certifications (Optional)

- **Certifications**: Preparing for Linux certifications like CompTIA Linux+, Red Hat Certified System Administrator (RHCSA), Linux Professional Institute Certification (LPIC) to gain a competitive edge.
- **Soft Skills**: Emphasizing problem-solving, critical thinking, and a willingness to learn new technologies, as companies value candidates who can adapt to the evolving IT landscape.

Note: Covering these topics will provide students with the well-rounded skill set that companies are seeking in Linux professionals, especially those working in system administration, cloud computing, DevOps, cybersecurity, and software development.

What proficiency Linux company/Industry want to hire students?

Prepared By: Dr. Premnarayan Arya

Companies hiring students for Linux-related roles seek candidates who exhibit a blend of fundamental and advanced skills. Here's a breakdown of the proficiency level companies generally expect:

1. Foundational Linux Knowledge

- **Basic Command Line Skills**: Students should have strong command-line navigation skills, including file and directory management, process control, and basic text processing using commands like grep, awk, and sed.
- **Shell Scripting**: Ability to write simple shell scripts for automating routine tasks, including backups, monitoring, and batch processing.

2. System Administration Proficiency

- **User and File Management**: Understanding of user/group management, permissions, file systems, and disk management (LVM).
- **Software Installation and Package Management**: Competence with package managers (apt, yum, dnf) for installing and updating software.
- **Network Configuration**: Basic network setup, SSH usage, and troubleshooting common network issues.
- **Security Basics**: Fundamental knowledge of securing a Linux system, including setting up a firewall (iptables, firewalld), managing user access, and system hardening.

3. Intermediate to Advanced Skills

- **Virtualization and Containerization**: Basic understanding of using containers (Docker) and, ideally, some exposure to Kubernetes for deploying and managing containerized applications.
- **Cloud Computing**: Experience or familiarity with cloud platforms like AWS, Google Cloud, or Microsoft Azure, using CLI tools, and managing Linux-based cloud instances.
- **Automation and DevOps Tools**: Basic to intermediate knowledge of using tools like Ansible, Puppet, or Chef for configuration management. Familiarity with CI/CD pipelines using tools like Jenkins or GitLab CI is a plus.
- **Troubleshooting and Problem Solving**: Ability to diagnose and troubleshoot common Linux issues, including performance bottlenecks, network issues, and system failures.

4. Soft Skills

- **Willingness to Learn**: Companies value candidates who show an eagerness to learn and adapt to new tools and technologies.
- **Collaboration**: Ability to work effectively in a team environment, especially in roles that involve cross-functional collaboration with development, security, and IT teams.

5. Certifications (Optional but Beneficial)

- Certifications such as **CompTIA Linux+**, **Red Hat Certified System Administrator (RHCSA)**, or **Linux Professional Institute Certification (LPIC)** can give students an advantage, demonstrating their verified skills and commitment to mastering Linux.

Expected Proficiency Level

- **Intermediate**: Most companies hiring entry-level students expect intermediate-level proficiency in Linux, including basic system administration, shell scripting, and understanding core networking and security concepts.
- **Advanced (Preferred)**: Some roles may require more advanced knowledge, particularly in cloud computing, DevOps, and automation tools, especially for positions that involve managing production environments or working on large-scale systems.

Company/Industries name with details in which Linux demand in 2024?

Prepared By: Dr. Premnarayan Arya

In 2024, the demand for Linux skills continues to rise as companies increasingly adopt open-source technologies for cloud computing, cybersecurity, software development, data science, and other IT operations. Here's a list of companies across various industries that are actively seeking Linux proficiency, along with some details:

1. Google

- **Industry**: Technology
- **Overview**: Google's operations and services, including search, Google Cloud, Gmail, and Android, are heavily reliant on Linux. In 2024, Google continues to look for professionals skilled in Linux for roles in cloud infrastructure (Google Cloud Platform), software development, network engineering, and system administration.
- **Key Roles**: Site Reliability Engineers (SREs), Cloud Engineers, System Administrators, DevOps Engineers.

2. Amazon (Amazon Web Services - AWS)

- **Industry**: Cloud Computing
- **Overview**: AWS provides cloud services that rely on Linux-based systems. In 2024, AWS continues to expand its cloud offerings, requiring experts in Linux to manage cloud infrastructure, build serverless applications, and optimize cloud deployments.
- **Key Roles**: Cloud Engineers, Solutions Architects, DevOps Engineers, Linux Systems Administrators.

3. Microsoft

- **Industry**: Technology
- **Overview**: With its increasing support for Linux through Azure (which supports various Linux distributions) and tools like Windows Subsystem for Linux (WSL), Microsoft actively seeks Linux professionals for its cloud services, open-source software integration, and hybrid cloud solutions.
- **Key Roles**: Cloud Engineers, Software Developers, Linux Systems Engineers, Azure Specialists.

4. IBM

- **Industry**: Technology and Consulting

- **Overview**: IBM's acquisition of Red Hat has bolstered its Linux-based services, especially around Red Hat Enterprise Linux (RHEL) and OpenShift. In 2024, IBM seeks professionals skilled in Linux for cloud computing, enterprise servers, and hybrid cloud solutions.
- **Key Roles**: Linux Administrators, OpenShift Engineers, Cloud Engineers, Mainframe Specialists.

5. Red Hat (IBM Subsidiary)

- **Industry**: Software
- **Overview**: Red Hat continues to be a pioneer in the Linux space, providing enterprise-level solutions like Red Hat Enterprise Linux (RHEL), OpenShift, and Ansible. In 2024, Red Hat is looking for Linux experts to support its software products, provide consulting services, and contribute to open-source projects.
- **Key Roles**: Linux System Administrators, DevOps Engineers, OpenStack Engineers, Technical Consultants.

6. Canonical (Ubuntu)

- **Industry**: Software
- **Overview**: Canonical, the company behind Ubuntu, continues to promote Ubuntu Linux in 2024 as a key OS for cloud, IoT, and server environments. They seek professionals who can work on developing, maintaining, and supporting Ubuntu-based solutions.
- **Key Roles**: Cloud Engineers, Linux System Administrators, DevOps Engineers, IoT Engineers.

7. Oracle

- **Industry**: Software/Database Management
- **Overview**: Oracle provides enterprise-level Linux distributions, Oracle Linux, optimized for its database solutions and cloud services. In 2024, they continue to hire Linux professionals for database management, cloud computing, and software development.
- **Key Roles**: Database Administrators (DBAs), Cloud Engineers, Linux Systems Engineers, Support Engineers.

8. Meta (Facebook)

- **Industry**: Social Media/Technology
- **Overview**: Meta's infrastructure is based on Linux. They use custom Linux distributions to power their data centers and services. In 2024, Meta continues to demand Linux professionals to manage their massive-scale infrastructure, optimize network operations, and support internal development.
- **Key Roles**: Site Reliability Engineers (SREs), Network Engineers, Systems Administrators, Infrastructure Engineers.

9. Netflix

- **Industry**: Entertainment/Streaming
- **Overview**: Netflix runs its content delivery network on Linux-based systems. In 2024, they seek professionals with Linux expertise to help manage their high-availability infrastructure, ensure secure streaming, and enhance the end-user experience.
- **Key Roles**: Linux Systems Engineers, Network Operations Engineers, DevOps Engineers, Site Reliability Engineers (SREs).

10. Tesla

- **Industry**: Automotive/Technology
- **Overview**: Tesla uses Linux-based systems for its vehicles' operating software, autonomous driving technology, and internal IT infrastructure. They require Linux experts to help develop, maintain, and secure these systems.
- **Key Roles**: Embedded Systems Engineers, Software Developers, IT Administrators, Security Engineers.

11. Dell Technologies

- **Industry**: Computer Hardware/IT Services
- **Overview**: Dell offers servers and workstations that come pre-installed with various Linux distributions. In 2024, they look for Linux professionals to support their enterprise hardware, provide consulting services, and engage in systems integration.
- **Key Roles**: Systems Engineers, Support Engineers, Solutions Architects, Linux Administrators.

12. Hewlett Packard Enterprise (HPE)

- **Industry**: Computer Hardware/IT Services
- **Overview**: HPE supports a range of Linux distributions on its hardware and offers solutions in cloud computing and data analytics. They seek Linux professionals to help with infrastructure management, cloud solutions, and system optimization.
- **Key Roles**: Cloud Engineers, Systems Administrators, Linux Support Engineers, IT Consultants.

13. SAP

- **Industry**: Enterprise Software
- **Overview**: SAP software solutions run on Linux-based systems, especially in cloud and on-premises data centers. In 2024, SAP demands Linux expertise for cloud solutions, database management (SAP HANA), and software development.
- **Key Roles**: Linux Systems Engineers, Database Administrators, Cloud Platform Engineers.

14. Citrix

- **Industry**: Cloud Computing/Virtualization
- **Overview**: Citrix provides virtualization, cloud computing, and networking solutions, many of which are deployed on Linux. They continue to hire Linux professionals to enhance and maintain their products.
- **Key Roles**: Cloud Engineers, Virtualization Engineers, Linux Systems Administrators.

15. SAP

- **Industry**: Enterprise Software
- **Overview**: SAP software solutions run on Linux-based systems, especially in cloud and on-premises data centers. In 2024, SAP demands Linux expertise for cloud solutions, database management (SAP HANA), and software development.
- **Key Roles**: Linux Systems Engineers, Database Administrators, Cloud Platform Engineers.

16. Nokia

- **Industry**: Telecommunications

- **Overview**: Nokia relies on Linux for its network solutions and infrastructure products. In 2024, Nokia continues to need Linux-skilled professionals to develop and maintain their network solutions.
- **Key Roles**: Network Engineers, Systems Administrators, DevOps Engineers.

These companies demonstrate the ongoing and growing demand for Linux expertise across various sectors. In 2024, candidates with Linux skills can find opportunities in cloud computing, IT infrastructure, cybersecurity, software development, data analytics, and more.

EMPLOYEE KEY ROLES IN LINUX INDUSTRIES

Prepared By: Dr. Premnarayan Arya

Key Roles of Systems Engineers in Linux industries

In industries where Linux is a central part of the infrastructure, Systems Engineers play a crucial role in designing, deploying, managing, and optimizing Linux-based systems. Their responsibilities often overlap with network administration, cloud computing, cybersecurity, and DevOps. Here's a detailed look at the role of Systems Engineers in Linux-based industries:

1. Linux System Architecture and Design

- Systems Engineers are responsible for designing the architecture of Linux-based systems, ensuring scalability, reliability, and performance.
- They make decisions regarding the choice of Linux distributions (e.g., Ubuntu, CentOS, Red Hat) based on the specific requirements of the project or company.
- They ensure that the system architecture is aligned with the company's overall IT infrastructure and goals.

2. System Configuration and Deployment

- They handle the installation and configuration of Linux operating systems on servers, virtual machines, or cloud instances.
- Systems Engineers ensure that Linux systems are optimized for the workloads they need to support, whether it's web hosting, database management, or application development.

3. Automation and Scripting

- Automation is a key part of the Linux Systems Engineer's role. They write and manage scripts (using tools like Bash, Python, or Perl) to automate tasks such as software deployment, system monitoring, backups, and user management.
- They often use automation tools like **Ansible**, **Puppet**, **Chef**, or **SaltStack** to manage large fleets of Linux servers efficiently.

4. System Monitoring and Maintenance

- Systems Engineers are responsible for continuously monitoring Linux systems to ensure optimal performance and availability. This involves using monitoring tools like **Nagios**, **Zabbix**, or **Prometheus**.
- They conduct regular system audits, apply updates and patches, and troubleshoot any issues that arise, ensuring minimal downtime.

5. Security and Hardening

- Security is a top priority in Linux environments, and Systems Engineers are responsible for securing systems through hardening techniques. This includes configuring firewalls (like **iptables** or **firewalld**), applying security patches, managing user permissions, and ensuring compliance with security standards (e.g., **SELinux**, **AppArmor**).
- They manage and configure encryption, secure file systems, and protect systems against attacks like DDoS, malware, and unauthorized access.

6. Networking and Infrastructure

- Systems Engineers working in Linux environments often manage networking components, ensuring seamless communication between Linux servers, databases, and other systems.
- They configure network services like **DNS**, **DHCP**, **VPNs**, and ensure secure access through **SSH** or similar protocols.
- They work with cloud platforms (AWS, Google Cloud, Azure) to design, deploy, and manage Linux instances and containers.

7. Performance Tuning and Optimization

- Linux Systems Engineers are tasked with optimizing systems for performance, especially in high-traffic or resource-intensive environments.
- They fine-tune kernel parameters, manage system resources, and optimize disk I/O and memory usage.
- Load balancing and redundancy strategies (using tools like **HAProxy**, **Nginx**, or **GlusterFS**) are also key responsibilities.

8. Storage Management

- They manage and configure various storage solutions such as **LVM**, **RAID**, **NFS**, and **SAN** to ensure efficient data management and redundancy.
- They work on managing databases like **MySQL**, **PostgreSQL**, or NoSQL databases like **MongoDB**, ensuring that they run smoothly on Linux systems.

9. Cloud and Containerization

- In modern Linux environments, Systems Engineers are deeply involved in the adoption of cloud and container technologies like **Docker**, **Kubernetes**, and **OpenStack**.
- They are responsible for creating, deploying, and managing containers and orchestrating them across clusters, ensuring scalability and fault tolerance in microservices architectures.

10. DevOps and CI/CD

- Many Linux Systems Engineers work closely with development teams in a **DevOps** environment, helping to set up continuous integration and continuous deployment (CI/CD) pipelines.
- They manage tools like **Jenkins**, **GitLab CI**, or **Travis CI**, ensuring smooth code deployment processes and managing the underlying Linux servers that support these tools.

11. Disaster Recovery and Backup

- They plan and implement disaster recovery strategies, ensuring that data is backed up regularly and can be restored quickly in case of failure.
- Systems Engineers are responsible for configuring backup solutions and verifying that these backups are reliable and accessible.

12. Documentation and Support

- Systems Engineers maintain detailed documentation of Linux system configurations, processes, and troubleshooting procedures.
- They provide ongoing support to other teams within the company, helping with problem resolution, system updates, and user training.

Key Roles of Support Engineers in Linux industries

Support Engineers in Linux industries play a vital role in ensuring the reliability, stability, and smooth operation of Linux-based systems and infrastructure. They are the frontline problem-solvers, providing technical assistance to both internal teams and external customers who encounter issues with Linux systems. Here's a detailed look at the key responsibilities and functions of Support Engineers in Linux-based environments:

1. Technical Support and Troubleshooting

- Support Engineers diagnose and resolve issues related to the Linux operating system, applications, services, and network configurations.
- They provide assistance with issues like system crashes, performance degradation, software conflicts, and network problems, often handling tasks like checking system logs, managing kernel errors, and debugging scripts.
- They use troubleshooting tools and techniques to analyze system behavior, and if necessary, escalate issues to higher-level engineers or developers.

2. Incident Management

- In the event of system failures or service disruptions, Support Engineers take immediate action to restore services.
- They often follow structured incident management processes, including logging incidents, performing root cause analysis, and documenting solutions for future reference.
- They are responsible for minimizing downtime, especially in production environments, ensuring business continuity.

3. User Support and Customer Service

- Support Engineers work closely with end-users, clients, or internal teams to resolve Linux-related issues. They may answer helpdesk tickets, troubleshoot remote access problems, or provide real-time support.
- They assist with tasks like configuring Linux environments for users, managing permissions, and resolving software compatibility issues.
- In customer-facing roles, they ensure customer satisfaction by providing clear, timely communication and support, sometimes interacting with businesses that run Linux servers or applications on cloud platforms.

4. System Monitoring and Maintenance

- Support Engineers are often responsible for monitoring Linux systems to ensure performance and availability. They use tools like Nagios, Zabbix, or Prometheus to identify and address potential issues before they escalate.

- They conduct routine system health checks, apply updates, and handle preventive maintenance tasks such as patching vulnerabilities, updating software packages, and ensuring system security.

5. Configuration and System Management

- They assist in configuring Linux systems, whether it's setting up new servers, configuring network interfaces, or managing file systems.
- Support Engineers may help with setting up services like Apache, Nginx, MySQL, or PostgreSQL, and configuring users, groups, and permissions to ensure secure and optimized operations.

6. Automation and Scripting

- Many Linux Support Engineers develop and use scripts (in Bash, Python, etc.) to automate repetitive tasks, such as system backups, log rotation, and system performance checks.
- They may also create automated tools to streamline support tasks, improving efficiency and reducing the likelihood of human error.

7. Security and System Hardening

- Support Engineers play a key role in securing Linux systems by ensuring proper firewall configurations, enforcing SELinux or AppArmor policies, and managing user access through secure methods like SSH.
- They often handle day-to-day security tasks like auditing access logs, applying security patches, and monitoring for signs of intrusion or abnormal activity.

8. Documentation and Knowledge Management

- Support Engineers are responsible for maintaining detailed documentation of system configurations, troubleshooting procedures, and solutions to known issues.
- They contribute to knowledge bases and internal wikis, enabling faster resolution of recurring problems and helping other team members or end-users understand system procedures.

9. Collaboration with Development and Operations Teams

- In DevOps or agile environments, Linux Support Engineers work closely with developers and operations teams to ensure smooth deployment of applications and services.
- They may assist with tasks such as deploying code to production environments, managing containerized applications (e.g., Docker, Kubernetes), or handling infrastructure as code (e.g., Terraform, Ansible).
- They provide feedback to development teams about bugs or performance issues encountered in production.

10. System Upgrades and Migrations

- Support Engineers are often involved in system upgrades, whether that's upgrading Linux distributions, kernel versions, or migrating services to newer or more scalable infrastructure (such as moving from on-premise systems to cloud platforms like AWS or Azure).
- They ensure that upgrades are performed with minimal disruption, testing changes in staging environments before rolling them out to production.

11. Backup and Disaster Recovery

- They manage and monitor backup solutions, ensuring that data is consistently backed up and can be restored in case of failure.
- Support Engineers are often tasked with implementing disaster recovery plans and verifying that recovery systems are functional, ensuring that Linux environments are prepared for unforeseen incidents.

12. Training and User Education

- Support Engineers may also provide training to less technical staff or clients on how to use Linux-based systems or tools.
- They help users understand basic Linux commands, system navigation, and application use to reduce reliance on technical support for minor issues.

Key Roles of Solution Architect in Linux industries

In Linux industries, Solutions Architects play a pivotal role in designing and overseeing the implementation of complex systems and architectures based on Linux platforms. They bridge the gap between business requirements and technical solutions, ensuring that the systems and services designed are scalable, efficient, and aligned with organizational goals. Below is a breakdown of their roles and responsibilities:

1. Architectural Design and Planning

- Solutions Architects design high-level solutions using Linux-based technologies to meet the specific needs of a business or client.
- They create blueprints for the infrastructure, ensuring that the Linux systems are integrated seamlessly with other technologies such as cloud platforms, databases, and applications.
- They consider factors like scalability, reliability, performance, and cost when designing these architectures, ensuring the proposed solutions align with long-term business objectives.

2. Requirements Gathering and Analysis

- Solutions Architects work closely with stakeholders (e.g., business teams, IT management, and end-users) to gather and analyze both functional and technical requirements.
- They determine how Linux systems can best support these requirements and translate them into actionable technical designs, ensuring that the solution is both feasible and cost-effective.

3. Technology Stack Selection

- Solutions Architects are responsible for selecting the appropriate Linux distributions (e.g., **Red Hat**, **Ubuntu**, **Debian**) and related open-source technologies that fit the project's needs.
- They evaluate and recommend tools and platforms (such as **Docker**, **Kubernetes**, **Ansible**, **Apache**, **Nginx**) that work well in Linux environments for tasks like containerization, automation, load balancing, and configuration management.

4. Cloud Integration and Hybrid Solutions

- In industries where cloud computing is prevalent, Solutions Architects design cloud-native or hybrid architectures using Linux-based instances in platforms like **AWS**, **Google Cloud**, or **Azure**.
- They ensure that Linux systems can scale efficiently in cloud environments, manage security, and optimize resource allocation for both on-premise and cloud-based systems.

5. System Integration

- They are responsible for integrating Linux systems with other enterprise software, hardware, and cloud infrastructure, ensuring smooth interaction between different components (e.g., databases, APIs, network services).
- This includes integrating with DevOps pipelines, ensuring continuous integration/continuous deployment (CI/CD) processes are efficient, and that the Linux-based infrastructure supports agile development workflows.

6. Security Architecture and Compliance

- Solutions Architects design secure architectures, ensuring that Linux systems are hardened and comply with security standards (such as **ISO**, **PCI-DSS**, or **GDPR**).
- They create security models that include firewalls, encryption, access control (e.g., **SELinux**, **AppArmor**), and secure communication channels like **VPNs** and **SSH**.
- They also implement policies for patch management and vulnerability assessments to protect against potential threats.

7. Performance Optimization and Scalability

- Solutions Architects optimize Linux-based systems for high performance, ensuring that applications can handle large volumes of traffic or data efficiently.
- They design architectures that support horizontal and vertical scaling, using techniques like load balancing (with tools like **HAProxy**, **Nginx**) and distributed computing to ensure performance is maintained even under heavy loads.
- They also fine-tune Linux kernel parameters and system configurations for optimal performance in specific use cases, such as databases, web applications, or real-time analytics.

8. Collaboration with Development and Operations Teams

- Solutions Architects work closely with developers and operations teams (especially in DevOps environments) to ensure that the infrastructure aligns with software development goals.
- They assist in defining infrastructure-as-code practices, integrating automation tools (like **Terraform**, **Ansible**), and ensuring that deployment pipelines work smoothly across Linux-based systems.
- By collaborating with these teams, they help to reduce time-to-market for new products and services while ensuring reliability.

9. Cost and Resource Optimization

- Solutions Architects design cost-effective Linux infrastructures, ensuring that resources such as CPU, memory, and storage are used efficiently.

- They help businesses save costs by recommending open-source tools, minimizing licensing fees, and optimizing cloud infrastructure usage (e.g., by choosing appropriate instance types in cloud environments).
- They also work on designing energy-efficient systems where applicable, contributing to sustainability goals.

10. Automation and Orchestration

- Automation is key in modern Linux-based environments, and Solutions Architects ensure that repetitive tasks are automated using tools like **Ansible**, **Puppet**, **Chef**, and **SaltStack**.
- They design orchestration frameworks for managing large-scale environments with tools like **Kubernetes** (for containers) or **OpenStack** (for private cloud infrastructure), ensuring that deployments are efficient and infrastructure is dynamically scalable.

11. Documentation and Technical Leadership

- Solutions Architects create comprehensive documentation for the proposed Linux-based architectures, including system diagrams, network topologies, and technical specifications.
- They provide technical leadership and mentorship to engineering teams, ensuring that the implementation of Linux systems follows best practices and aligns with the architectural vision.
- They also provide guidance on troubleshooting and optimization strategies for maintaining a healthy, scalable, and reliable Linux environment.

12. Proof of Concept (PoC) and Prototyping

- Solutions Architects often create Proof of Concepts (PoC) to validate that the proposed Linux-based architecture can meet business and technical requirements before full-scale implementation.
- They prototype new solutions, test different configurations, and assess the performance, security, and feasibility of various approaches.

13. Disaster Recovery and High Availability

- Solutions Architects design systems that are fault-tolerant and highly available. This often involves configuring clustering solutions (e.g., **Pacemaker**, **Corosync**) or implementing failover strategies to ensure that Linux systems can recover from failures with minimal downtime.
- They also plan and design backup and disaster recovery solutions to ensure data integrity and business continuity in the event of system failures or other disasters.

Key Roles of Linux Administrator in Linux industries

Linux Administrators in Linux industries are responsible for managing and maintaining the operation, performance, and security of Linux-based systems. They are crucial in ensuring the smooth functioning of Linux environments, whether in small businesses or large enterprises. Their role includes a wide range of activities such as system setup, monitoring, security management, and user support. Below is a detailed explanation of their responsibilities and key functions:

1. System Installation and Configuration

- **Linux Administrators** are responsible for installing Linux operating systems on servers, workstations, and other hardware, ensuring that all necessary software and services are configured properly.
- They select the appropriate Linux distributions (e.g., **Ubuntu**, **CentOS**, **Red Hat**), depending on the needs of the organization, and ensure that the system is customized to meet specific performance or application requirements.
- Admins configure core system settings such as network interfaces, storage devices, and user accounts.

2. User Management

- Linux Administrators handle user accounts, permissions, and access controls to ensure secure and organized usage of Linux systems.
- They manage user groups and roles, assign file and directory permissions, and implement policies such as password expiration or multi-factor authentication (MFA) for added security.
- They also handle user support requests, helping with access issues and application setups.

3. System Monitoring and Performance Tuning

- They monitor the health and performance of Linux systems using tools like **Nagios**, **Zabbix**, or **Prometheus** to detect any issues related to CPU usage, memory, storage, or network traffic.
- Linux Admins optimize system performance by tuning system settings, adjusting kernel parameters, and troubleshooting resource bottlenecks.
- Regular system audits and health checks are performed to ensure optimal operation and to identify potential areas for improvement.

4. Security Management and System Hardening

- A key responsibility of Linux Administrators is maintaining system security by applying best practices for hardening Linux systems. This includes configuring firewalls (e.g., **iptables**, **firewalld**), enforcing security policies (e.g., **SELinux**, **AppArmor**), and keeping the system up to date with security patches.

- They manage encryption, secure file systems, and secure access to services using tools like **SSH**, ensuring that only authorized users have access to sensitive parts of the system.
- Admins also configure intrusion detection systems (IDS) and monitor logs for suspicious activity.

5. Backup and Disaster Recovery

- Linux Administrators are responsible for setting up regular backups to ensure data is safe and can be restored in case of hardware failure, cyberattacks, or other disasters.
- They configure and manage backup systems, ensuring both system-level (e.g., **rsync**, **tar**) and application-level backups (e.g., databases, filesystems) are performed and tested regularly.
- They plan and implement disaster recovery strategies, ensuring the system can recover quickly from failures.

6. Software and Patch Management

- Linux Administrators manage and update software packages on Linux servers, ensuring that applications are up to date and that security patches are applied promptly.
- They use package management tools (e.g., **YUM**, **APT**, **dnf**) to install, update, and remove software packages.
- Admins are responsible for testing updates in a staging environment before deploying them to production to minimize the risk of system issues.

7. Automation and Scripting

- Automation is an important part of a Linux Administrator's role. They write scripts (e.g., **Bash**, **Python**) to automate routine tasks such as system monitoring, backups, and user account management.
- They implement automation tools like **Ansible**, **Puppet**, or **Chef** to manage large numbers of Linux servers efficiently, automating configuration management and deployment processes.

8. Network Configuration and Management

- Linux Administrators configure and manage network settings on Linux systems, including IP addresses, DNS, DHCP, and routing.
- They handle network services like **Apache**, **Nginx**, **FTP**, **SSH**, and **VPNs**, ensuring that the services are running securely and optimally.
- Admins troubleshoot network issues, including firewall rules, port forwarding, and packet filtering to ensure smooth communication between systems.

9. Storage and Filesystem Management

- Linux Administrators are responsible for managing filesystems, including mounting and unmounting disks, managing RAID configurations, and using logical volume management (**LVM**) for flexible disk storage.
- They ensure that the file systems are optimized for performance and scalability, using file systems such as **ext4**, **XFS**, **ZFS**, and **Btrfs**.
- They monitor storage utilization and ensure proper allocation of disk space, as well as manage data redundancy and fault tolerance.

10. System Backups and Data Management

- Linux Administrators implement and maintain data backup strategies to ensure that critical data is backed up and recoverable in the event of system failure or data loss.
- They regularly verify the integrity of backup files and test the system's ability to restore data from backups.
- Admins also manage data retention policies and handle file archiving and cleanup processes to ensure that the system runs efficiently.

11. High Availability and Load Balancing

- They configure high-availability systems using technologies like **Pacemaker** and **Corosync** to ensure that services remain operational even in the event of hardware or software failures.
- Admins manage load balancing solutions, such as **HAProxy** or **Nginx**, to distribute traffic across multiple servers and ensure optimal performance, especially in web hosting or database-heavy environments.

12. Collaboration with Development and Operations Teams

- Linux Administrators often work closely with development, DevOps, and security teams to ensure that systems meet business needs and are properly integrated into the overall IT infrastructure.
- They help developers with the setup of environments, containers, and configurations needed to run applications smoothly on Linux systems.
- They collaborate with operations teams to ensure seamless CI/CD pipelines and infrastructure-as-code (IaC) practices.

13. Documentation and Knowledge Sharing

- Administrators document system configurations, procedures, and troubleshooting steps to ensure consistency and provide a knowledge base for other team members.
- They may create internal wikis, guides, or SOPs (Standard Operating Procedures) to facilitate knowledge transfer and ensure that systems can be managed by other administrators or support staff.
- Continuous learning is often part of their role, staying up to date with the latest Linux technologies and best practices.

Key Roles of Cloud Engineer in Linux industries

Cloud Engineers in Linux industries are responsible for designing, implementing, managing, and optimizing cloud-based infrastructures that often run on Linux operating systems. Their role revolves around leveraging cloud technologies to build scalable, secure, and efficient solutions while managing and integrating Linux systems within these environments. Below is a detailed breakdown of their roles and responsibilities:

1. Cloud Infrastructure Design and Implementation

- Cloud Engineers design cloud-based infrastructure that is primarily based on Linux operating systems. This includes creating scalable architectures that support web applications, databases, and other services in cloud environments.
- They work with cloud providers like **AWS**, **Google Cloud**, **Azure**, or **OpenStack** to provision virtual machines (Linux instances), manage containers, and deploy cloud services.
- They determine the appropriate cloud services to use (compute, storage, networking) and architect solutions that balance performance, cost, and scalability.

2. Linux System Management in the Cloud

- Since many cloud environments use Linux as the default operating system, Cloud Engineers must configure and manage Linux instances in the cloud, handling setup, security, and performance tuning.
- This involves managing services like **SSH** access, software updates, and configuring critical system parameters to ensure optimal performance.
- They are also responsible for ensuring that Linux instances comply with organizational security policies and best practices for cloud security.

3. Cloud Automation and Orchestration

- Automation is a critical part of a Cloud Engineer's role. They use tools like **Terraform**, **Ansible**, **Puppet**, or **Chef** to automate the provisioning and management of cloud resources, ensuring that infrastructure can scale dynamically based on demand.
- They often develop infrastructure-as-code (IaC) scripts to automate cloud deployments, managing configurations and services across large Linux fleets with minimal manual intervention.
- Orchestration tools like **Kubernetes** are used for managing containers (e.g., **Docker**) in the cloud, allowing Cloud Engineers to deploy, manage, and scale containerized applications.

4. Cloud Networking and Security Management

- Cloud Engineers configure and manage the networking infrastructure in cloud environments, including **VPCs** (Virtual Private Clouds), **subnets**, **firewalls**, **VPNs**, and load balancers.
- They ensure that Linux systems in the cloud are secure, implementing firewalls, encryption, and other network security measures to safeguard data and applications.
- They work with security tools and services such as **IAM** (Identity and Access Management), **VPNs**, **SSL certificates**, and network policies to restrict access and protect cloud resources.

5. Storage and Database Management

- Cloud Engineers manage storage solutions in cloud environments, such as **Amazon S3**, **Google Cloud Storage**, or **Azure Blob Storage**, often optimizing storage for Linux-based applications and systems.
- They work with Linux-based database servers such as **MySQL**, **PostgreSQL**, **MongoDB**, ensuring that cloud resources are optimized for data storage, retrieval, and backup.
- They implement automated backups, data replication, and disaster recovery solutions to ensure data integrity and availability.

6. Cloud Security and Compliance

- Security is a top priority in cloud environments, and Cloud Engineers are responsible for securing Linux systems and cloud resources against threats.
- They implement cloud security best practices, such as encrypting data at rest and in transit, managing firewalls, and securing API endpoints.
- They ensure that Linux systems and cloud services comply with industry regulations (e.g., **PCI-DSS**, **HIPAA**, **GDPR**) by configuring security settings and conducting regular audits of security policies.

7. Performance Monitoring and Optimization

- Cloud Engineers monitor the performance of Linux instances and cloud services using tools like **Prometheus**, **CloudWatch**, or **Datadog**.
- They are responsible for ensuring that cloud resources are optimized for cost and performance, scaling resources up or down based on traffic, usage patterns, and application needs.
- They also troubleshoot performance issues, such as bottlenecks in CPU, memory, disk I/O, or network performance, ensuring that applications running in the cloud maintain high availability and efficiency.

8. Cost Management and Optimization

- Cloud Engineers work to optimize the cost of running cloud-based Linux environments by selecting the appropriate resource types (e.g., instance types, storage tiers) and ensuring that cloud resources are not over-provisioned.

- They monitor usage and recommend ways to reduce costs, such as by leveraging spot instances, reserved instances, or auto-scaling capabilities.
- Engineers implement cost-monitoring tools to provide insights into resource usage and billing, ensuring that organizations are maximizing their cloud investment efficiently.

9. Disaster Recovery and High Availability

- Cloud Engineers design disaster recovery strategies to ensure that data and applications running on Linux systems in the cloud can recover quickly in case of failure.
- They configure systems for high availability using load balancing, failover strategies, and geographic redundancy to minimize downtime.
- Backup strategies are implemented using cloud-native tools to ensure that Linux-based applications and services can be restored in the event of a disaster.

10. Collaboration with DevOps and Development Teams

- Cloud Engineers work closely with DevOps, development, and operations teams to integrate cloud-based Linux environments with CI/CD pipelines.
- They automate deployment processes, enabling faster and more reliable software releases. This includes configuring cloud environments for containerized applications and managing microservices architecture.
- They also help ensure that Linux environments in the cloud are aligned with the overall goals of the development teams, providing the necessary infrastructure to support new features or services.

11. Backup and Data Management

- Cloud Engineers implement and manage backup strategies for data stored in the cloud, ensuring data is replicated and recoverable.
- They use cloud storage solutions and backup services (like **AWS Backup** or **Google Cloud Storage** snapshots) to protect critical data hosted on Linux systems.
- They configure data retention policies, ensuring that backups are automated and optimized for both recovery speed and cost-efficiency.

12. Migration to the Cloud

- Cloud Engineers are often responsible for migrating existing on-premise Linux infrastructure to the cloud. This involves assessing current systems, planning the migration process, and executing the migration with minimal downtime.
- They ensure that services, applications, and data are transferred seamlessly to the cloud and that the Linux systems operate as expected in the new environment.
- Cloud Engineers may also handle hybrid cloud setups, where some infrastructure remains on-premise while other parts are moved to the cloud.