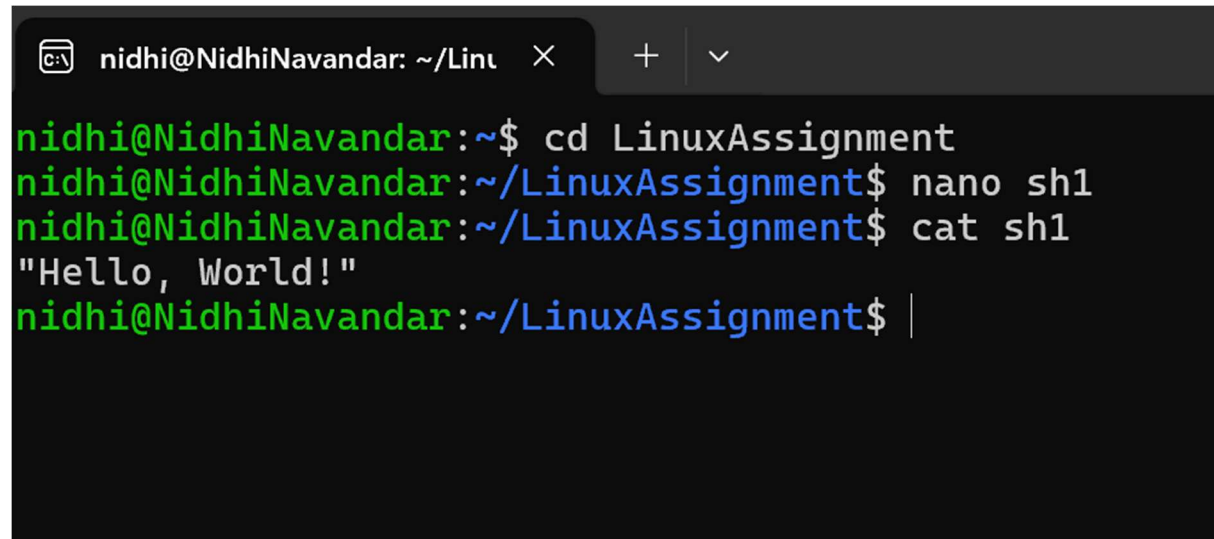


ASSIGNMENT - 02

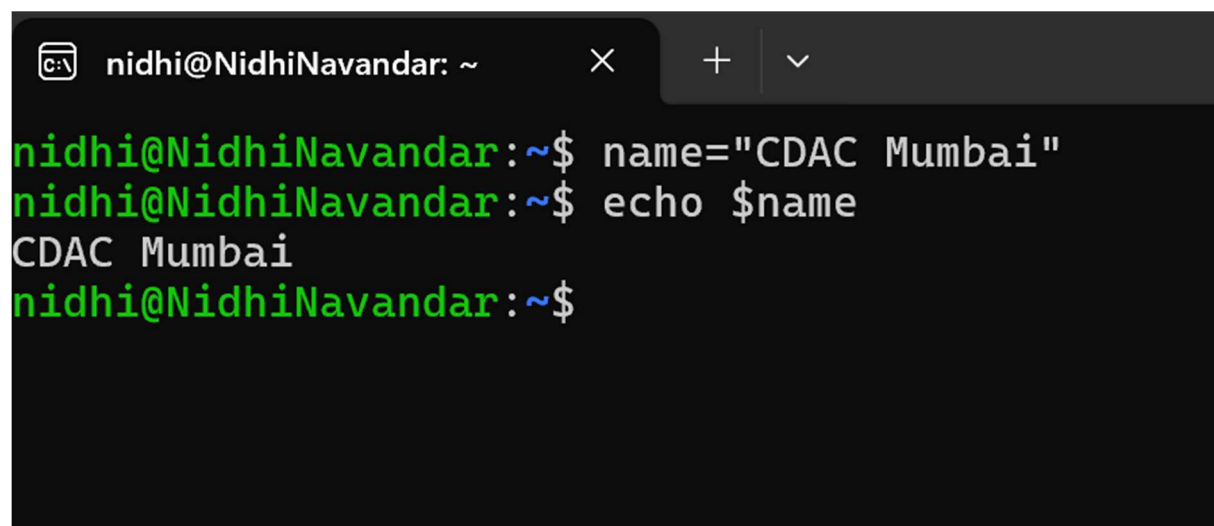
PART C

Question 1: Write a shell script that prints "Hello, World!" to the terminal.



```
nidhi@NidhiNavandar: ~/LinuxAssignment
nidhi@NidhiNavandar:~/LinuxAssignment$ nano sh1
nidhi@NidhiNavandar:~/LinuxAssignment$ cat sh1
"Hello, World!"
nidhi@NidhiNavandar:~/LinuxAssignment$
```

Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.



```
nidhi@NidhiNavandar: ~
nidhi@NidhiNavandar:~$ name="CDAC Mumbai"
nidhi@NidhiNavandar:~$ echo $name
CDAC Mumbai
nidhi@NidhiNavandar:~$
```

Question 3: Write a shell script that takes a number as input from the user and prints it.

```
nidhi@NidhiNavandar: ~/Linu × + v
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh3
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh3
echo "Enter a number: "
read number
echo "You entered: $number"
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh3
Enter a number:
13 7 2000
You entered: 13 7 2000
nidhi@NidhiNavandar:~/LinuxAssignment2$ |
```

Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.

```
nidhi@NidhiNavandar: ~/Linu × + v
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh4
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh4
num1=5
num2=3
sum=$((num1 + num2))
echo "The sum of $num1 and $num2 is: $sum"
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh4
The sum of 5 and 3 is: 8
nidhi@NidhiNavandar:~/LinuxAssignment2$ |
```

Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".

```
nidhi@NidhiNavandar: ~/Linu × + ▾  
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh5  
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh5  
echo "Enter a number: "  
read n  
if [  $$(n \% 2)$  -eq 0 ]; then  
    echo "Even"  
else  
    echo "Odd"  
fi  
nidhi@NidhiNavandar:~/LinuxAssignment2$ bashsh5  
bashsh5: command not found  
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh5  
Enter a number:  
12  
Even  
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh5  
Enter a number:  
1  
Odd  
nidhi@NidhiNavandar:~/LinuxAssignment2$ \
```

Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.

```
nidhi@NidhiNavandar: ~/Linu × + v
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh6
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh6
for i in {1..5}
do
    echo $i
done
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh6
1
2
3
4
5
nidhi@NidhiNavandar:~/LinuxAssignment2$ |
```

Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.

```
nidhi@NidhiNavandar: ~/Linu × + v
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh7
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh7
i=1
while [ $i -le 5 ]
do
    echo $i
    i=$((i + 1))
done
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh7
1
2
3
4
5
nidhi@NidhiNavandar:~/LinuxAssignment2$ |
```

Question 8: Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".

```
nidhi@NidhiNavandar: ~/Linu X + v
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh8
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh8
if [ -e file.txt ]; then
    echo "File exists"
else
    echo "File does not exist"
fi
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh8
File does not exist
nidhi@NidhiNavandar:~/LinuxAssignment2$ |
```

Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.

```
nidhi@NidhiNavandar: ~/Linu X + v
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh9
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh9
echo "Enter a number: "
read number
if [ $number -gt 10 ]; then
    echo "The number is greater than 10"
else
    echo "The number is not greater than 10"
fi
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh9
Enter a number:
13
The number is greater than 10
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh9
Enter a number:
2
The number is not greater than 10
nidhi@NidhiNavandar:~/LinuxAssignment2$ |
```

Question 10: Write a shell script that uses nested for loops to print a multiplication table for numbers from 1 to 5. The output should be formatted nicely, with each row representing a number and each column representing the multiplication result for that number.

```
nidhi@NidhiNavandar: ~/Linu  ×  +  ∨  
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh10  
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh10  
for i in {1..5}  
do  
    for j in {1..5}  
    do  
        printf "%4d" $((i * j))  
    done  
    echo  
done  
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh10  
1    2    3    4    5  
2    4    6    8    10  
3    6    9    12   15  
4    8    12   16   20  
5    10   15   20   25  
nidhi@NidhiNavandar:~/LinuxAssignment2$ |
```


Question 11: Write a shell script that uses a while loop to read numbers from the user until the user enters a negative number. For each positive number entered, print its square. Use the break statement to exit the loop when a negative number is entered.

```
nidhi@NidhiNavandar: ~/Linu × + v
nidhi@NidhiNavandar:~/LinuxAssignment2$ nano sh11
nidhi@NidhiNavandar:~/LinuxAssignment2$ cat sh11
while true
do
    echo "Enter a number: "
    read number
    if [ $number -lt 0 ]; then
        break
    fi
    echo "The square of $number is: $((number * number))"
done
nidhi@NidhiNavandar:~/LinuxAssignment2$ bash sh11
Enter a number:
23
The square of 23 is: 529
Enter a number:
41
The square of 41 is: 1681
Enter a number:
13
The square of 13 is: 169
Enter a number:
-12
nidhi@NidhiNavandar:~/LinuxAssignment2$ |
```

PART A - What will the following commands do?

1. **echo "Hello, World!"**

Prints "Hello, World!" to the terminal.

2. **name="Productive"**

Assigns the string "Productive" to the variable name.

3. **touch file.txt**

Creates an empty file named file.txt if it does not exist. If it already exists, updates its timestamp.

4. **ls -a**

Lists all files and directories, including hidden files (those starting with .).

5. **rm file.txt**

Deletes the file file.txt.

6. **cp file1.txt file2.txt**

Copies file1.txt to file2.txt. If file2.txt exists, it will be overwritten.

7. **mv file.txt /path/to/directory/**

Moves file.txt to the specified directory (/path/to/directory/).

8. **chmod 755 script.sh**

Changes the permissions of script.sh to 755 (owner can read, write, execute; group and others can read and execute).

9. **grep "pattern" file.txt**

Searches for the string "pattern" in file.txt and displays matching lines.

10. **kill PID**

Terminates the process with the given PID (Process ID).

11. **mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**

Creates a directory mydir.

Changes into mydir.

Creates an empty file file.txt.

Writes "Hello, World!" into file.txt.

Displays the content of file.txt.

12. `ls -l | grep ".txt"`

Lists files in long format and filters out only those containing .txt in their names.

13. `cat file1.txt file2.txt | sort | uniq`

Concatenates file1.txt and file2.txt, sorts the lines, and removes duplicate lines.

14. `ls -l | grep "^d"`

Lists only directories (^d indicates lines starting with "d", which denotes directories in ls -l).

15. `grep -r "pattern" /path/to/directory/`

Recursively searches for "pattern" inside all files within /path/to/directory/.

16. `cat file1.txt file2.txt | sort | uniq -d`

Concatenates file1.txt and file2.txt, sorts the lines, and displays only duplicate lines.

17. `chmod 644 file.txt`

Sets file permissions to 644 (owner can read and write; group and others can only read).

18. `cp -r source_directory destination_directory`

Copies source_directory and all its contents (recursively) to destination_directory.

19. `find /path/to/search -name "*.txt"`

Searches for files with a .txt extension in /path/to/search and its subdirectories.

20. `chmod u+x file.txt`

Grants execute (+x) permission to the user (u) for file.txt.

21. `echo $PATH`

Displays the system's PATH environment variable, which contains directories where executable files are searched for.

PART B

Identify True or False:

1. **ls** is used to list files and directories in a directory. - TRUE
2. **mv** is used to move files and directories. - TRUE
3. **cd** is used to copy files and directories. - FALSE used for changing the directory
4. **pwd** stands for "print working directory" and displays the current directory. - TRUE
5. **grep** is used to search for patterns in files. - TRUE
6. **chmod 755 file.txt** gives read, write, and execute permissions to the owner, and read and execute permissions to group and others. - TRUE
7. **mkdir -p directory1/directory2** creates nested directories, creating directory2 inside directory1 if directory1 does not exist. - TRUE
8. **rm -rf file.txt** deletes a file forcefully without confirmation. - TRUE

Identify the Incorrect Commands:

1. **chmodx** is used to change file permissions.

chmod is used to change the file permissions.

2. **cpy** is used to copy files and directories.

cp command is used to copy the files.

3. **mkfile** is used to create a new file.

touch command is used to create files.

4. **catx** is used to concatenate files.

cat command is used to concatenate the files.

5. **rn** is used to rename files.

mv command is used to rename the files.

PART – E

1. Consider the following processes with arrival times and burst times:

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	6

Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.

As it is a First Come, First Served :-

Process	Arrival Time	Burst Time	Waiting Time (Start Time – Arrival Time)
P1	0	5	0
P2	1	3	4
P3	2	6	6

Gantt chart

0 5 8 14

P1	P2	P3
----	----	----

AVG Waiting Time = $(0+4+6)/3 = 3.33$

2. Consider the following processes with arrival times and burst times:

Process	Arrival Time	Burst Time
P1	0	3
P2	1	5
P3	2	1
P4	3	4

Calculate the average turnaround time using Shortest Job First (SJF) scheduling.

Process	Arrival Time	Burst Time	Waiting Time	TurnAround Time (Burst Time + Waiting Time)
P1	0	3	0	3
P2	1	5	7	12
P3	2	1	1	2
P4	3	4	1	5

Gantt chart

0 3 4 8 13

P1	P3	P4	P2
----	----	----	----

AVG Turn Around Time :- $(3+12+2+5)/4 = 22/4 = 5.5$

3. Consider the following processes with arrival times, burst times, and priorities (lower number

indicates higher priority):

Process	Arrival Time	Burst Time	Priority
P1	0	6	3
P2	1	4	1
P3	2	7	4
P4	3	2	2

Calculate the average waiting time using Priority Scheduling.

Non Preemptive

Process	Arrival Time	Burst Time	Priority	Waiting Time (Start Time – Arrival Time)
P1	0	6	3	0
P2	1	4	1	5
P3	2	7	4	10
P4	3	2	2	7

Gantt chart

0 6 10 12 19

P1	P2	P4	P3
----	----	----	----

AVG Waiting Time = $(0+5+10+7)/4 = 5.5$

4. Consider the following processes with arrival times and burst times, and the time quantum for

Round Robin scheduling is 2 units:

Process	Arrival Time	Burst Time
P1	0	4
P2	1	5
P3	2	2
P4	3	3

Calculate the average turnaround time using Round Robin scheduling.

Process	Arrival Time	Burst Time	Waiting Time	Turn Around Time
P1	0	4	6	10
P2	1	5	8	13
P3	2	2	2	4
P4	3	3	7	10

Gantt Chart

0	2	4	6	8	10	12	13	14
P1	P2	P3	P4	P1	P2	P4	P2	

AVG Turn Around Time :- $(10+13+4+10)/4 = 37/4 = 9.25$

5. Consider a program that uses the fork() system call to create a child process. Initially, the parent

process has a variable x with a value of 5. After forking, both the parent and child processes

increment the value of x by 1.

What will be the final values of x in the parent and child processes after the fork() call?

In a program using the fork() system call, the parent process has a variable x with a value of 5. After forking, both the parent and child processes increment the value of x by 1.

After the fork() call, both the parent and child processes have their own copies of the variable x. Incrementing x by 1 each Process result in:

Parent Process :- x becomes 6

Child Process :- x becomes 6

Therefore, the final value of x is 6 in both the parent and child processes.

