

Assignment-2

Part-A

What will the following commands do?

- **echo "Hello, World!"**

Sol : Using echo command we get to display the string Hello World that has been passed to it as an argument.

- **name="Productive"**

Sol: This is basic string initialization where a name variable is declared & is assigned the string value Productive.

- **touch file.txt**

Sol: Touch command is used to create a new file.

- **ls -a**

Sol: ls command lists all the directories & files including the hidden files and directories in the listing.

- **rm file.txt**

Sol: rm command removes file.txt from the system.

- **cp file1.txt file2.txt**

Sol: it copies the contents of the first file to the second file.

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

Sol: This command moves the file "file.txt" to the target directory **"/path/to/directory/"**.

- **chmod 755 script.sh**

Sol: chmod 755 script.sh sets the permission for owner to read , write & execute while for group & others it will be read & execute for the file script.sh.

- **grep "pattern" file.txt**

Sol: grep command will search for the string "pattern" from file.txt & then will display it.

- **kill PID**

Sol: kill command is a built-in command which is used to terminate processes manually. The `kill` command requires the process ID (PID) & the signal of the process we want to terminate.

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

Sol: (&&) is used when two or more conditions are present and it returns true when all the conditions are true. The second command will only execute if the first command has executed successfully. mkdir mydir: Creates a directory named mydir. cd mydir: Changes into the newly created mydir directory. touch file.txt: Creates an empty file named file.txt. echo "Hello, World!" > file.txt: Writes the string "Hello, World!" into file.txt, overwriting any existing content. The o/p of echo command is inserted into file.txt using (>) redirection operator. cat file.txt: Displays the contents of file.txt in the terminal.

- **ls -l | grep ".txt"**

Sol: This command combines ls -l with grep using a **pipe (|)**. **ls -l** lists files in the current directory in **long format**. Pipe passes the output of ls -l as input to the next command (grep). grep searches for lines containing the string .txt, which matches **text files**

- **cat file1.txt file2.txt | sort | uniq**

Sol: This command cat file1.txt file2.txt concatenates the contents of file1.txt and file2.txt **one after the other**. Passes the combined output of

cat as input to the next command (sort). Sort- Sorts the lines **alphabetically** (by default). Uniq Removes **duplicate adjacent lines**, keeping only **unique entries**.

- **ls -l | grep "^d"**

Sol: ls -l Lists files and directories in the current directory in **long format**. Pipe passes the output of ls -l as input to the next command (grep). "**^d**" Filters the output to show **only directories**.

- **grep -r "pattern" /path/to/directory/**

Sol: grep command recursively the text "pattern" **inside all files** in the given directory and **subdirectories**.

- **cat file1.txt file2.txt | sort | uniq -d**

Sol: This command cat file1.txt file2.txt concatenates the contents of file1.txt and file2.txt **one after the other**. Passes the combined output of cat as input to the next command (sort). Sort- Sorts the lines **alphabetically** (by default). **uniq -d** Finds and prints only the duplicated lines.

- **chmod 644 file.txt**

Sol: chmod 644 file.txt sets the permission for owner to read , write while for group & others it will be read only.

- **cp -r source_directory destination_directory**

Sol: This command using -r copies the **entire directory of source_directory** and its contents, including subdirectories and their files to **destination_directory**.

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

Sol: find command is used to **search for files and directories**. Given command searches **./path/to/search** directory & its subdirectories for files ending with .txt pattern.

- **chmod u+x file.txt**

Sol: This command grants execute permission for file.txt file only to the owner of the file.

- **echo \$PATH**

Sol: **\$PATH is an environment variable** that stores a **colon-separated list of directories** where your shell looks for executable files.

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** ,

Sol: cd allows us to change the current the current working directory within our filesystem.

4. pwd stands for "print working directory" and displays the current directory -
Sol: **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**

Sol: The `-r` option is used only for directories & not the file but using `-rf` on a file still works.

Identify the Incorrect Commands:

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

Sol: `chmod` is used to change file permissions.

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

Sol: `cpy` is used to copy files and directories.

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

Sol: `touch` is used to create a new file.

4. `catx` is used to concatenate files.

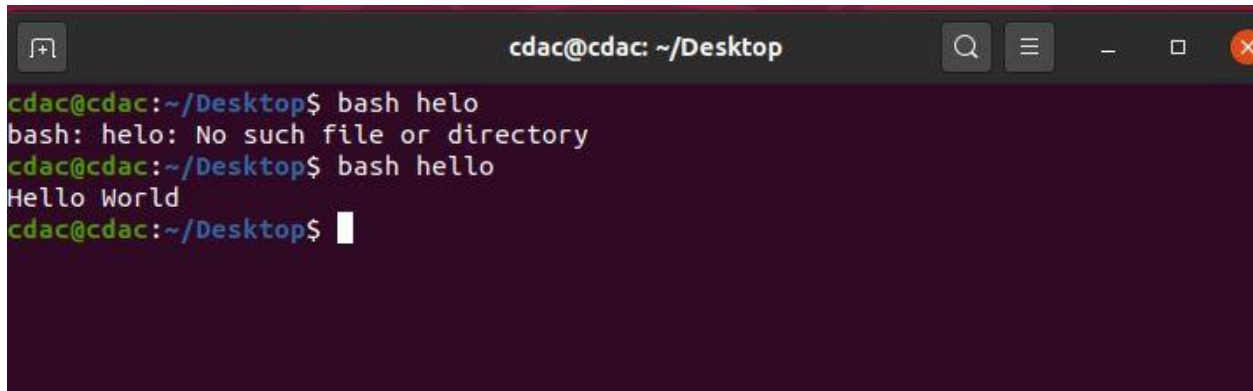
Sol: `cat` is used to concatenate files.

5. `rn` is used to rename files.

Sol: `mv` or `rename` is used to rename files.

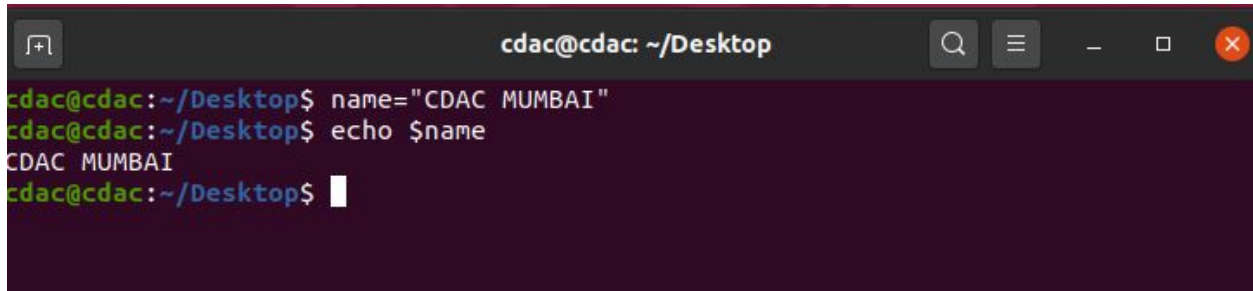
Part-C

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

A terminal window titled 'cdac@cdac: ~/Desktop' with search, menu, and window control icons. The user enters 'bash helo', which results in an error: 'bash: helo: No such file or directory'. Then, the user enters 'bash hello', which results in the output 'Hello World'.

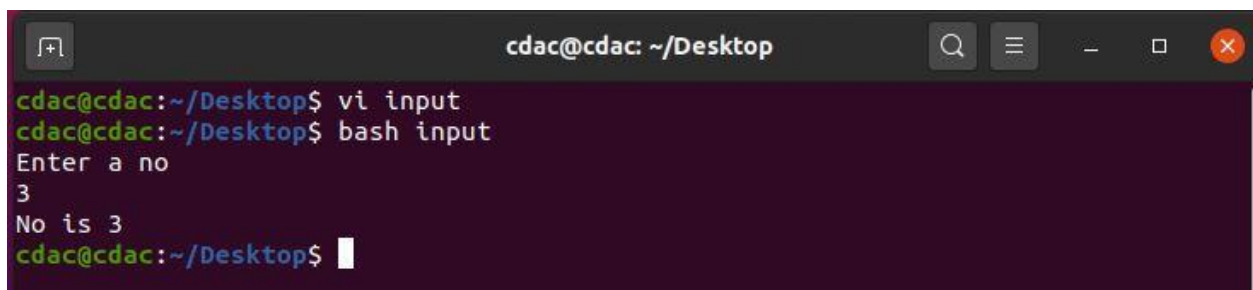
```
cdac@cdac:~/Desktop$ bash helo
bash: helo: No such file or directory
cdac@cdac:~/Desktop$ bash hello
Hello World
cdac@cdac:~/Desktop$
```

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

A terminal window titled 'cdac@cdac: ~/Desktop' with search, menu, and window control icons. The user enters 'name="CDAC MUMBAI"', followed by 'echo \$name', which outputs 'CDAC MUMBAI'.

```
cdac@cdac:~/Desktop$ name="CDAC MUMBAI"
cdac@cdac:~/Desktop$ echo $name
CDAC MUMBAI
cdac@cdac:~/Desktop$
```

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

A terminal window titled 'cdac@cdac: ~/Desktop' with search, menu, and window control icons. The user enters 'vi input', then 'bash input'. The script prompts 'Enter a no', the user enters '3', and the script outputs 'No is 3'.

```
cdac@cdac:~/Desktop$ vi input
cdac@cdac:~/Desktop$ bash input
Enter a no
3
No is 3
cdac@cdac:~/Desktop$
```

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

```
cdac@cdac: ~/Desktop
cdac@cdac:~/Desktop$ cat add
echo "Enter 2 nos"

read a
read b
sum=$((a + b))

echo "sum of $a and $b is $sum"
cdac@cdac:~/Desktop$ bash add
Enter 2 nos
3
7
sum of 3 and 7 is 10
cdac@cdac:~/Desktop$
```

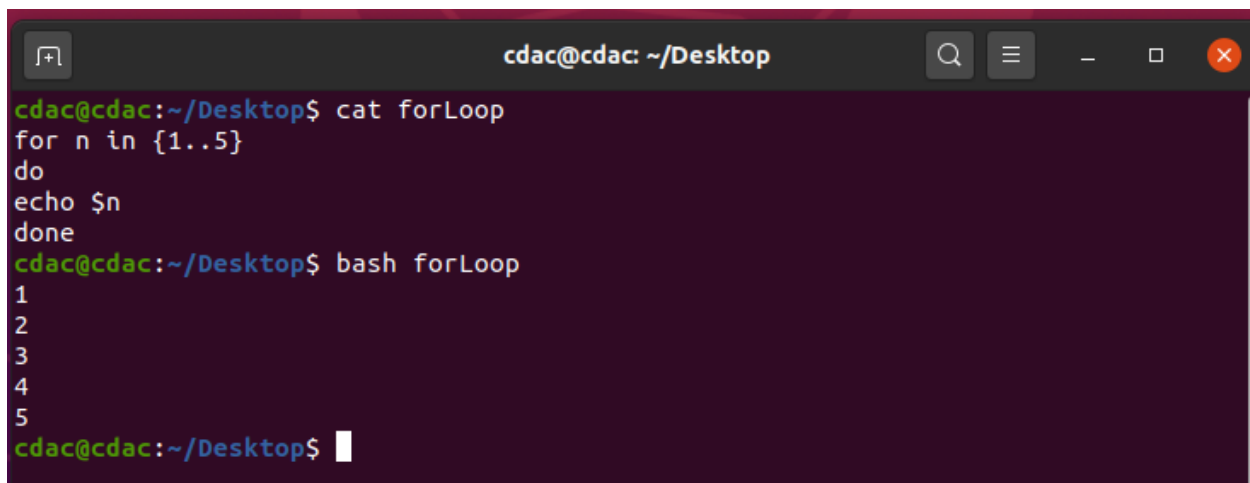
Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".

```
cdac@cdac: ~/Desktop
cdac@cdac:~/Desktop$ cat evenOdd
echo "Enter a no"
read n

if((n%2==0))
then

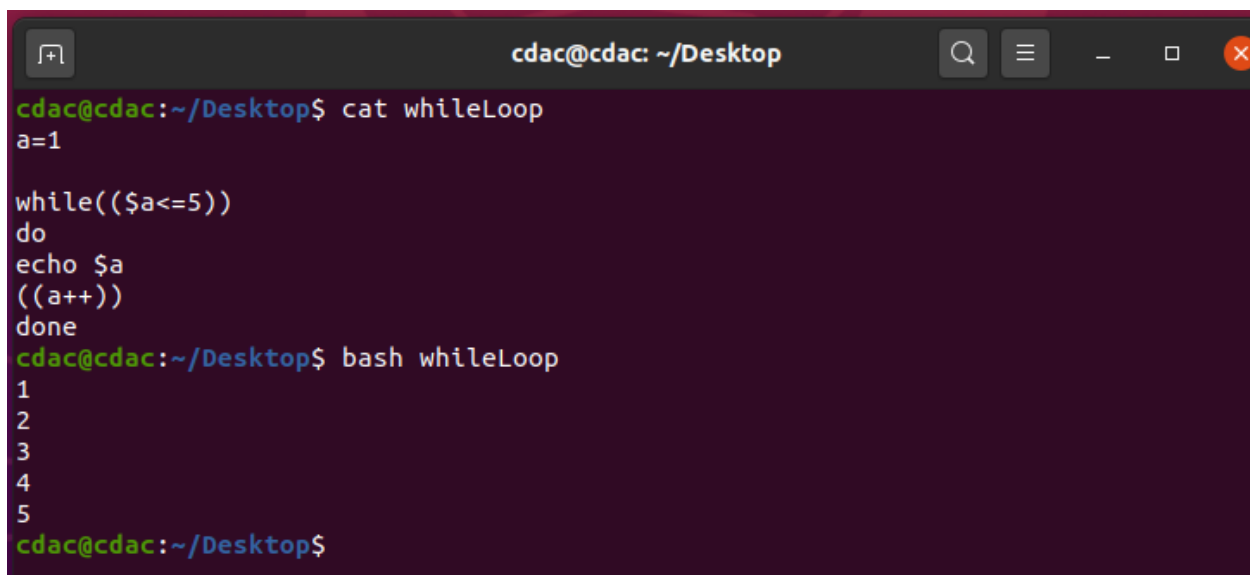
echo "$n is even"
else
echo "$n is odd"
fi
cdac@cdac:~/Desktop$ bash evenOdd
Enter a no
4
4 is even
cdac@cdac:~/Desktop$ bash evenOdd
Enter a no
1
1 is odd
cdac@cdac:~/Desktop$
```

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

A terminal window titled 'cdac@cdac: ~/Desktop' with search, menu, and window control icons. The user enters 'cat forLoop' and the script content is displayed: 'for n in {1..5}', 'do', 'echo \$n', 'done'. Then the user enters 'bash forLoop' and the output shows numbers 1 through 5 on separate lines. The prompt returns to 'cdac@cdac:~/Desktop\$'.

```
cdac@cdac:~/Desktop$ cat forLoop
for n in {1..5}
do
echo $n
done
cdac@cdac:~/Desktop$ bash forLoop
1
2
3
4
5
cdac@cdac:~/Desktop$
```

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

A terminal window titled 'cdac@cdac: ~/Desktop' with search, menu, and window control icons. The user enters 'cat whileLoop' and the script content is displayed: 'a=1', 'while((\$a<=5))', 'do', 'echo \$a', '((a++))', 'done'. Then the user enters 'bash whileLoop' and the output shows numbers 1 through 5 on separate lines. The prompt returns to 'cdac@cdac:~/Desktop\$'.

```
cdac@cdac:~/Desktop$ cat whileLoop
a=1
while(($a<=5))
do
echo $a
((a++))
done
cdac@cdac:~/Desktop$ bash whileLoop
1
2
3
4
5
cdac@cdac:~/Desktop$
```

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".


```
cdac@cdac: ~/Desktop
cdac@cdac:~/Desktop$ cat checkFile
if [ -e "file.txt" ]; then
    echo "File exists"
else
    echo "File does not exist"
fi
cdac@cdac:~/Desktop$ bash checkFile
File exists
cdac@cdac:~/Desktop$ rm file.txt
cdac@cdac:~/Desktop$ bash checkFile
File does not exist
cdac@cdac:~/Desktop$
```

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.

```
cdac@cdac: ~/Desktop
cdac@cdac:~/Desktop$ cat greaterNum
echo "Enter a number"
read n

if [ $n -gt 10 ]
then
    echo " $n is greater than 10 "
else
    echo " $n is not greater than 10 "
fi
cdac@cdac:~/Desktop$ bash greaterNum
Enter a number
40
40 is greater than 10
cdac@cdac:~/Desktop$ bash greaterNum
Enter a number
7
7 is not greater than 10
cdac@cdac:~/Desktop$
```

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.

```
cdac@cdac: ~/Desktop
cdac@cdac:~/Desktop$ cat mul
for i in {1..5}
do
  for j in {1..10}
  do
    echo -ne " $((i*j))\t"
  done

  echo
done
cdac@cdac:~/Desktop$ bash mul
1      2      3      4      5      6      7      8      9      10
2      4      6      8      10     12     14     16     18     20
3      6      9      12     15     18     21     24     27     30
4      8      12     16     20     24     28     32     36     40
5      10     15     20     25     30     35     40     45     50
cdac@cdac:~/Desktop$
```

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.

```
cdac@cdac: ~/Desktop
cdac@cdac:~/Desktop$ cat Q11
while [ true ]
do
    echo "Enter a no"
    read n

    if [ $n -lt 0 ]
    then
        echo "Negative no entered. Terminate..."
        break
    fi

    square=$((n*n));
    echo "Square of $n is : $square"
done
cdac@cdac:~/Desktop$ bash Q11
Enter a no
4
Square of 4 is : 16
Enter a no
-9
Negative no entered. Terminate...
cdac@cdac:~/Desktop$
```

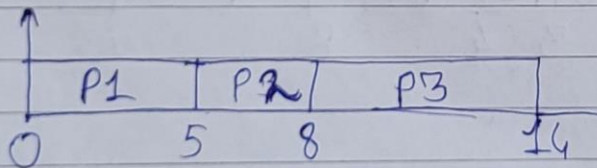
Part-E

1) Consider the following processes with arrival times & burst times: Calculate the avg waiting time using FCFS scheduling.

Process	Arrival time	Burst time	Response time	Wait time
P1	0	5	0	0
P2	1	3	5	4
P3	2	6	8	6
				<u>10</u>

→

Gantt chart



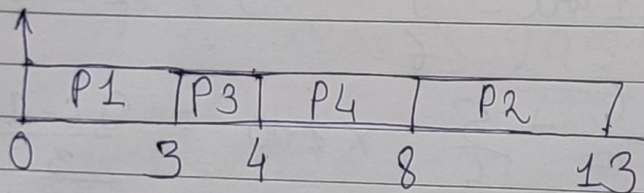
$$\text{Avg waiting time} = \frac{10}{3} = 3.33 \text{ units}$$

2) Consider the following processes with arrival times & burst times & calculate the avg turnaround time using SJF scheduling.

Process	Arrival Time	Burst time	Response Time	Wait Time	TAT
P1	0	3	0	0	3
P2	1	5	8	7	12
P3	2	1	3	1	2
P4	3	4	4	1	5

→ Solved this problem using non-preemptive scheduling (Shortest Next Time First).

Gantt chart:

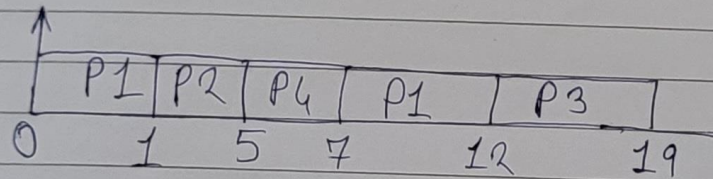


$$\text{Avg TAT} = \frac{3 + 12 + 2 + 5}{4} = \frac{22}{4} = 5.5$$

3) Consider the following processes with arrival times, burst times & priorities (lower the avg waiting time using priority scheduling).

Process	Arrival Time	Burst Time	Priority	Response time	Wait time	TAT
P1	0	6	3	0+6	6	12
P2	1	4	1	1	0	4
P3	2	7	4	12	10	17
P4	3	2	2	5	2	4

→ Gantt chart:

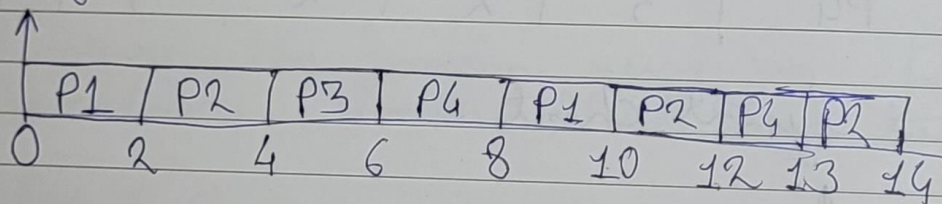


$$\text{Avg waiting time} = \frac{6 + 0 + 10 + 2}{4} = 4.5$$

4) Consider the following processes with arrival times & burst times & the time quantum for Round Robin scheduling is 2 units:

Process	Arrival time	Burst time	Response time	Wait time	TAT
P1	0	4	0	0+6	10
P2	1	5	2	1+6+1	13
P3	2	2	4	2	4
P4	3	3	6	3+4	10

→ Gantt chart:



$$\text{Avg TAT} = \frac{10 + 13 + 4 + 10}{4} = \frac{37}{4} = 9.25$$

5) → Initially the parent process has a variable
 $x = 5$

→ After forking a new process is created which is called child process that is a copy of the parent process. Both the process have their own memory.

→ So forking increments the value of x in parent & child process by 1.
 $\boxed{x = 6}$ $\boxed{x = 6}$