CDAC MUMBAI Concepts of Operating System Assignment 2 Part A

What will the following commands do?

- echo "Hello, World!"
- name="Productive"
- touch file.txt
- ls -a
- rm file.txt
- cp file1.txt file2.txt
- mv file.txt /path/to/directory/
- chmod 755 script.sh
- grep "pattern" file.txt
- kill PID
- mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt
- ls -l | grep ".txt"
- cat file1.txt file2.txt | sort | uniq
- ls -l | grep "^d"
- grep -r "pattern" /path/to/directory/
- cat file1.txt file2.txt | sort | uniq –d
- chmod 644 file.txt
- cp -r source_directory destination_directory
- find /path/to/search -name "*.txt"

- echo "Hello, World!"
    - Prints "Hello, World!" to the terminal.
- name="Productive"
    - Assigns the value "Productive" to the variable name.
- touch file.txt
    - Creates an empty file named file.txt if it does not exist, or updates its last modified timestamp if it does.
- ls -a
    - Lists all files and directories, including hidden files (those starting with a dot .).
- rm file.txt
    - Deletes file.txt permanently.
- cp file1.txt file2.txt
    - Copies file1.txt to file2.txt.

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

    - Moves file.txt to the specified directory.

- chmod 755 script.sh

    - Changes permissions of script.sh to rwxr-xr-x (owner: read, write, execute; group & others: read and execute).

- grep "pattern" file.txt

    - Searches for the specified "pattern" in file.txt and prints matching lines.

- kill PID

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

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

    - Creates a directory mydir, enters it, creates file.txt, writes "Hello, World!" into file.txt, and then displays its content.

- ls -l | grep ".txt"

    - Lists files in long format and filters only .txt files using grep.

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

    - Concatenates file1.txt and file2.txt, sorts the lines, and removes duplicates.

- ls -l | grep "^d"

    - Lists directories only (^d filters lines that start with "d", which denotes directories in ls -l).

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

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

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

  Displays only duplicate lines that appear in both file1.txt and file2.txt.

- chmod 644 file.txt

    - Sets permissions to rw-r--r-- (owner: read/write, others: read-only).

- cp -r source_directory destination_directory

    - Copies source_directory and its contents recursively to destination_directory.

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

    - Searches for all .txt files in /path/to/search.

- chmod u+x file.txt

    - Gives the owner (u) execute (x) permission on file.txt.

- echo $PATH

    - Prints the system's PATH variable, which contains directories where executable files are stored.

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

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. → Incorrect
2. cpy is used to copy files and directories. → Incorrect
3. mkfile is used to create a new file. → Incorrect
4. catx is used to concatenate files. → Incorrect
5. rn is used to rename files. → Incorrect

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

Input-> nano hello.sh
chmod u+x hello.sh
cat hello.sh
Output->Hello, World!

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

->name="CDAC MUMBAI"
echo $name
output->CDAC MUMBAI

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

-> echo -n "Enter n1"
read n1
5
echo $n1
Output->5

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

Input-> a=5
b=3
sum=$((a + b))
echo The sum of $5 and $3 is $sum
Output-root@DESKTOP-2UVI7R7:~/ShellScript# bash addition.sh

The sum of 5 and 3 is 8
root@DESKTOP-2UVI7R7:~/ShellScript#

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

```
Input-> echo "Enter num1"
read num1
if (($num1%2==0))
then
echo "Number is even."
else
echo "Number is odd."
Fi
```

```
Output->root@DESKTOP-2UVI7R7:~/ShellScript# bash evenodd.sh
Enter num1
8
Number is even.
root@DESKTOP-2UVI7R7:~/ShellScript# bash evenodd.sh
Enter num1
9
Number is odd.
```

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

```
Input->a=0
 for a in 1 2 3 4 5
do
```

```
echo $a
done
Output->root@DESKTOP-2UVI7R7:~/ShellScript# bash
forloop.sh
1
2
3
4
5
```

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

```
Input->a= 1
while [ $a -lt 6 ]
do
echo $a
a=` expr $a + 1 `
done
Output->root@DESKTOP-2UVI7R7:~/ShellScript# bash
while.sh
1
2
3
4
5
```

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".

```
Input-> echo "Enter file name"
read name
if test $name == file.txt
```

then
echo "File.txt is exist."
else
echo "file.txt is not exist."
Fi
Output->root@DESKTOP-2UVI7R7:~/ShellScript# bash ifelse.sh
Enter file name
exit.txt
file.txt is not exist.
root@DESKTOP-2UVI7R7:~/ShellScript# bash ifelse.sh
Enter file name
file.txt
File.txt is exist.

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.

Input->echo "Enter a num"
read num
if [$num –gt 10]
then
echo "Number $num is Greater than 10."
else
echo "Number $num is Not Greater than 10."
Fi
Outpu->root@DESKTOP-2UVI7R7:~/ShellScript# bash gt.sh
Enter a number
52
Number 52 is greater than 10.
root@DESKTOP-2UVI7R7:~/ShellScript# bash gt.sh
Enter a number
03

Number 03 is not greater than 10.

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.

```
Input-> i=0
j=0
for i in $(seq 1 5);
do
for j in $(seq 1 10);
do
product=` expr $i \* $j `
echo -ne " $i x $j = $product\t"
done
echo
done ->Ouput
root@DESKTOP-2UVI7R7:~/ShellScript# bash multi.sh
```

1 x 1 = 1 1 x 2 = 2 1 x 3 = 3 1 x 4 = 4 1 x 5 = 5
1 x 6 = 6 1 x 7 = 7 1 x 8 = 8 1 x 9 =
9 1 x 10 = 10
2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10
2 x 6 = 12 2 x 7 = 14 2 x 8 = 16 2 x
9 = 18 2 x 10 = 20
3 x 1 = 3 3 x 2 = 6 3 x 3 = 9 3 x 4 = 12 3 x 5 = 15
3 x 6 = 18 3 x 7 = 21 3 x 8 = 24 3 x
9 = 27 3 x 10 = 30
4 x 1 = 4 4 x 2 = 8 4 x 3 = 12 4 x 4 = 16 4 x 5 = 20
4 x 6 = 24 4 x 7 = 28 4 x 8 = 32 4
x 9 = 36 4 x 10 = 40
5 x 1 = 5 5 x 2 = 10 5 x 3 = 15 5 x 4 = 20 5 x 5 = 25
5 x 6 = 30 5 x 7 = 35 5 x 8 = 40 5
x 9 = 45 5 x 10 = 50

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.

Input-> while true;
do
echo -ne "Enter a number"
read num
if [[$num -lt 0]]
then
echo "Number is negative. exiting..."
break
fi
square= ` expr $num \* $num `
echo 'square is :'$square

done
Output->root@DESKTOP-2UVI7R7:~/ShellScript# bash sq.sh
Enter a number85
sq.sh: line 10: 7225: command not found
square is :
Enter a number

Part E

1. Consider the following processes with arrival times and burst times: | Process | Arrival Time | Burst Time.

	A	B	C	D	E	F	G	H
1	Process	Arrival Time	Burst Time	Response Time	Waiting Time	TAT		
2	P1	0	5	0	0	5		
3	P2	1	3	5	4	7		
4	P3	2	6	8	6	12		
5					Avg WT=3.33	Avg TAT=8		
6								
7								
8				P1	P2		P3	
9			Giant chart	0	5	8		14
10								

2. Consider the following processes with arrival times and burst times.Calculate the average turnaround time using Shortest Job First (SJF) scheduling.

	A	B	C	D	E	F	G
1	Process	Arrival Time	Burst Time	Response Time	Waiting Time	TAT	
2	P1	0	3	0	0	3	
3	P2	1	5	8	7	12	
4	P3	2	1	3	1	2	
5	P4	3	4	4	1	5	
6				Avg RT = 3.75	Avg WT = 2.25	Avg TAT=5.5	
7							
8							
9							
10		P1	P3		P4		P2
11	Giant chart	0	3	4		8	13
12							

3. Consider the following processes with arrival times, burst times, and priorities (lower number indicates higher priority): Calculate the average waiting time using Priority Scheduling. Non-Preemptive Algo.

	A	B	C	D	E	F	G
1	Process	Arrival Time	Burst Time	Priority	Response Time	Waiting Time	TAT
2	P1	0	6	3	0	0	6
3	P2	1	4	1	6	5	9
4	P3	2	7	4	12	10	17
5	P4	3	2	2	10	7	9
6					Avg RT = 7	Avg WT = 5.5	Avg TAT = 10.25
7							
8							
9							
10		P1	P2	P4		P3	
11	Giant chart	0	6	10	12		19
12							

3. Consider the following processes with arrival times and burst times, and the time quantum for Round Robin scheduling is 2 units: Calculate the average turnaround time using Round Robin scheduling.

F9 fx 8

	A	B	C	D	E	F	G	H	I	
1	Process Id	Arrival Time	Burst Time	Response Time	Waiting Time	Turn Around Time				
2	P1	0	4	0	6	10				
3	P2	1	5	2	8	13				
4	P3	2	2	4	2	4		RR=2Units		
5	P4	3	3	6	10	13				
6					Avg WT = 6.5	Avg TAT = 10				
7										
8		P1	P2	P3	P4	P1	P2	P4	P2	
9	Giant Chart	0	2	4	6	8	10	12	13	14
10										