

Pradnya More

Form No. 241205579

CDAC MUMBAI
Concepts of Operating System
Assignment 2

Part A

What will the following commands do?

- **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 (or updates the timestamp if it exists).
- **ls -a** - Lists all files and directories, including hidden ones.
- **rm file.txt** - Deletes file.txt.
- **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 be readable and executable by everyone, but only writable by the owner.
- **grep "pattern" file.txt** - Searches for "pattern" in file.txt.
- **kill PID** - Terminates the process with the specified PID.
- **mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt** - Creates a directory mydir, navigates into it, creates file.txt, writes "Hello, World!" into it, and displays the content.
- **ls -l | grep ".txt"** - Lists details of files and directories, filtering to show only .txt files.
- **cat file1.txt file2.txt | sort | uniq** - Combines contents of both files, sorts them, and removes duplicate lines.

- **ls -l | grep "^d"** - Lists details of directories only.
- **grep -r "pattern" /path/to/directory/** - Recursively searches for "pattern" in all files under the specified directory.
- **cat file1.txt file2.txt | sort | uniq -d** - Lists only duplicate lines found in both files.
- **chmod 644 file.txt** - Sets permissions so the owner can read/write, while others can only read.
- **cp -r source_directory destination_directory** - Recursively copies a directory and its contents.
- **find /path/to/search -name "*.txt"** - Finds all .txt files under the specified path.
- **chmod u+x file.txt** - Gives the owner execution permission for file.txt.
- **echo \$PATH** - Displays the system's PATH environment variable.

Part B

Identify True or False:

ls is used to list files and directories in a directory. - **True**

mv is used to move (or rename) files and directories. - **True**

cd is used to change the current directory, not copy files. - **False** (cp is used for copying files and directories.)

pwd stands for "print working directory" and displays the current directory path. - **True**

grep is used to search for patterns in files. – **True**

mkdir -p directory1/directory2 creates nested directories, including directory1 if it does not exist, and then directory2 inside it. – **True**

rm -rf file.txt deletes the file forcefully without confirmation. - **True**

(The -r is for recursive deletion (mainly for directories), and -f forces the deletion without prompts.)

mkdir -p directory1/directory2 creates nested directories. If directory1 does not exist, it will be created along with directory2. - **True**

Identify the Incorrect Commands:

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

Incorrect - chmodx is not a valid command. The correct command for changing file permissions is **chmod**.

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

Incorrect - cpy is not a valid command. The correct command for copying files and directories is **cp**.

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

Incorrect - mkfile is not typically used in Linux to create a new file. The common way is to use **touch** or **redirection (>)** to create an empty file.

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

Incorrect - catx is not a valid command. The correct command for concatenating and displaying file contents is **cat**.

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

Incorrect - rn is not a valid command. The correct command for renaming files is **mv**.

Part C

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

```
cdac@LAPTOP-0IP9GGLJ:~$ pwd
```

```
/home/cdac
```

```
cdac@LAPTOP-0IP9GGLJ:~$ ls
```

```
LinuxAssignment
```

```
cdac@LAPTOP-0IP9GGLJ:~$ cd LinuxAssignment
```

```
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano hello
```

```
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash hello
```

```
Hello, World!
```

```
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$
```

```
cdac@LAPTOP-0IP9GGLJ:~$ pwd
/home/cdac
cdac@LAPTOP-0IP9GGLJ:~$ ls
LinuxAssignment
cdac@LAPTOP-0IP9GGLJ:~$ cd LinuxAssignment
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano hello
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash hello
Hello, World!
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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

```
cdac@LAPTOP-0IP9GGLJ:~$ pwd
/home/cdac
cdac@LAPTOP-0IP9GGLJ:~$ ls
LinuxAssignment
cdac@LAPTOP-0IP9GGLJ:~$ cd LinuxAssignment
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ name="CDAC Mumbai"
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ echo $name
CDAC Mumbai
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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

```
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano print
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat print
echo "Enter a number"
read number
echo "You Entered: $number"
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash print
Enter a number
12
You Entered: 12
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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

```
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano addition2no
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat addition2no
num1=5
num2=3
sum=$((num1 + num2))
echo "Addition of $num1 and $num2 is: $sum"
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash addition2no
Addition of 5 and 3 is: 8
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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

```
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano EvenOdd
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat EvenOdd
echo "Enter a number:"
read n
if [  $$(n \% 2)$  -eq 0 ]; then
    echo "Even"
else
    echo "Odd"
fi
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash EvenOdd
Enter a number:
2
Even
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash EvenOdd
Enter a number:
3
Odd
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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

```
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano printno
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat printno
for i in {1..5}
do
    echo $i
done
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash printno
1
2
3
4
5
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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

```
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano whileprintno
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat whileprintno
i=1
while [ $i -le 5 ]
do
    echo $i
    i=$((i + 1))
done
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash whileprintno
1
2
3
4
5
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano filecheck
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat filecheck
if [ -e file.txt ]; then
    echo "File exists"
else
    echo "File does not exists"
fi
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash filecheck
File does not exists
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano graterno
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat graterno
echo "Enter a number"
read number
if [ $number -gt 10 ]; then
    echo "The number is greater than 10"
else
    echo "The number is no greater than 10"
fi
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash graterno
Enter a number
77
The number is greater than 10
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash graterno
Enter a number
5
The number is no greater than 10
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano multiplicationtable
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat multiplicationtable
for i in {1..5}
do
    for j in {1..5}
    do
        result=`expr $i \* $j`
        echo -n "$result    "
    done
    echo
done
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash multiplicationtable
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
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```

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@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ nano posinegi
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ cat posinegi
while true
do
    echo "Enter a number:"
    read number
    if [ $number -lt 0 ]; then
        break
    fi
    echo "The square of $number is: $((number * number))"
done
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ bash posinegi
Enter a number:
34
The square of 34 is: 1156
Enter a number:
68
The square of 68 is: 4624
Enter a number:
-34
cdac@LAPTOP-0IP9GGLJ:~/LinuxAssignment$ |
```


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.

Step 1: Compute Completion Time (CT)

FCFS executes processes in the order they arrive.

1. **P1** starts at time **0** and finishes at $0+5=5$ + 5 = 5.
2. **P2** starts at time **5** and finishes at $5+3=8$ + 3 = 8.
3. **P3** starts at time **8** and finishes at $8+6=14$ + 6 = 14.

Process Arrival Time Burst Time Completion Time (CT)

P1	0	5	5
P2	1	3	8
P3	2	6	14

Step 2: Compute Turnaround Time (TAT)

$TAT = CT - AT$

Process Arrival Time Completion Time Turnaround Time (TAT)

P1	0	5	$5 - 0 = 5$
P2	1	8	$8 - 1 = 7$
P3	2	14	$14 - 2 = 12$

Step 3: Compute Waiting Time (WT)

$WT = TAT - BT$

Process Turnaround Time (TAT) Burst Time Waiting Time (WT)

P1	5	5	$5-5=0$ $5-5=0$
P2	7	3	$7-3=4$ $7-3=4$
P3	12	6	$12-6=6$ $12-6=6$

Step 4: Compute Average Waiting Time (AWT)

$$0+4+6/3 = 3.33$$

Average Waiting Time (AWT) = 3.33 ms

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.

Step 1: Arrange Processes Based on Arrival Time

SJF selects the process with the shortest burst time that has arrived at any given time.

Step 2: Compute Completion Time (CT)

- P1 starts at 0 and finishes at $0+3=3$ $0+3=3$.
- P3 has the shortest burst time (1) among available processes (P2, P3, P4) at time 3, so it starts at 3 and finishes at $3+1=4$ $3+1=4$.
- P4 has the next shortest burst time (4), so it starts at 4 and finishes at $4+4=8$ $4+4=8$.
- P2 is the only process left, so it starts at 8 and finishes at $8+5=13$ $8+5=13$.

Process Arrival Time Burst Time Completion Time (CT)

P1	0	3	3
----	---	---	---

Process Arrival Time Burst Time Completion Time (CT)

P3	2	1	4
P4	3	4	8
P2	1	5	13

Step 3: Compute Turnaround Time (TAT)

$$TAT = CT - AT \quad TAT = CT - AT$$

Process Arrival Time Completion Time Turnaround Time (TAT)

P1	0	3	$3 - 0 = 3$
P3	2	4	$4 - 2 = 2$
P4	3	8	$8 - 3 = 5$
P2	1	13	$13 - 1 = 12$

Step 4: Compute Average Turnaround Time (ATAT)

$$\frac{3 + 2 + 5 + 12}{4} = 5.5$$

Average Turnaround Time (ATAT) = 5.5 ms

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.

Step 1: Execution Order Based on Priority

- At time 0, P1 is the only available process, so it starts first.
- At time 6, P2 (priority 1) arrives and has the highest priority, so it runs next.
- At time 10, P4 (priority 2) is the next highest priority, so it runs next.
- At time 12, P3 (priority 4) runs last.

Execution Order Process Arrival Time Burst Time Priority

1st	P1	0	6	3
2nd	P2	1	4	1
3rd	P4	3	2	2
4th	P3	2	7	4

Step 2: Compute Completion Time (CT)

- P1 starts at 0 and finishes at $0+6=6$ $0 + 6 = 6$.
- P2 starts at 6 and finishes at $6+4=10$ $6 + 4 = 10$.
- P4 starts at 10 and finishes at $10+2=12$ $10 + 2 = 12$.
- P3 starts at 12 and finishes at $12+7=19$ $12 + 7 = 19$.

Process Arrival Time Burst Time Completion Time (CT)

P1	0	6	6
P2	1	4	10
P4	3	2	12
P3	2	7	19

Step 3: Compute Turnaround Time (TAT)

$$TAT = CT - AT$$

Process Arrival Time Completion Time Turnaround Time (TAT)

P1	0	6	$6-0=6$ $6 - 0 = 6$
P2	1	10	$10-1=9$ $10 - 1 = 9$
P4	3	12	$12-3=9$ $12 - 3 = 9$

Process Arrival Time Completion Time Turnaround Time (TAT)

P3	2	19	$19-2=17$
----	---	----	-----------

Step 4: Compute Waiting Time (WT)

WT=TAT-BT

Process Turnaround Time (TAT) Burst Time Waiting Time (WT)

P1	6	6	$6-6=0$
----	---	---	---------

P2	9	4	$9-4=5$
----	---	---	---------

P4	9	2	$9-2=7$
----	---	---	---------

P3	17	7	$17-7=10$
----	----	---	-----------

Step 5: Compute Average Waiting Time (AWT)

$$\frac{0+5+7+10}{4} = 5.5$$

Average Waiting Time (AWT) = 5.5 ms