CDAC MUMBAI

Concepts of Operating System

Assignment No: 2

Part A

Name: Akash Bhadane

What will the following commands do?

1. **echo "Hello, World!"**
   - Print the text Hello World! on the terminal.

2. **name="Productive"**
   - Create a one Variable name and assign the value Production.

3. **touch file.txt**
   - Create a new empty txt file named file.txt

4. **ls -a**
   - list all files in the directory, including all hidden files.

5. **rm file.txt**
   - Delete the file name file.txt.

6. **cp file1.txt file2.txt**
   - Copy the content of file1.txt into a new file file2.txt.

7. **mv file.txt /path/to/directory/**
   - Move the file file.txt into the specified directory.

8. **chmod 755 script.sh**
   - change file permission so that:
   - The owner can read, write, and execute.
   - The group can read and execute.
   - Others can read and execute

9. **grep "pattern" file.txt**
   - Search for the word pattern inside file.txt prints the match line.

10. **kill PID**
    - Terminates the process with the given process PID.

11. **mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**
    - Create a folder mydir.
    - Change directory to mydir.
    - Create a file named file.txt.
    - Write Hello, World! Into the file.
    - Displays the content of the file on the terminal.

12. **ls -l | grep ".txt"**
    - Shows file in long format but only displays those ending with .txt.

13. **cat file1.txt file2.txt | sort | uniq**
    - Combines the content of two files, sorts the lines, and removes any duplicates.

14. **ls -l | grep "^d"**
    - lists only directories or starting with d.

15. **grep -r "pattern" /path/to/directory/**
    - Search the pattern word in all files under the given directory, including subfolder

16. **cat file1.txt file2.txt | sort | uniq –d**
    - show only the lines that appear in both files or duplicates.

17. **chmod 644 file.txt**
    - Set the file permissions using a numeric code 644.
    - The owner can read and write.
    - The group and others can only read.

18. **cp -r source_directory destination_directory**
    - copy an entire folder to another location.

19. **find /path/to/search -name "*.txt"**
    - Finds all .txt files inside the given path.

20. **chmod u+x file.txt**
    - Owner u executes permission, without changing other permissions.

21. **echo $PATH**
    - Display the systems PATH variable.

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**
   - ***Correct* -cd used to change 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 directory1if 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 command
   - *Correct command*: correct command **chmod.**

2. **cpy** is used to copy files and directories.
   - Incorrect command
   - *Correct command*: the correct command is **cp.**

3. **mkfile** is used to create a new file.
   - Incorrect command
   - *Correct command**:*** **touch** filename.

4. **catx** is used to concatenate files.
   - **Incorrect command.**
   - *Correct command:* is **cat**

5. **rn** is used to rename files.
   - **Incorrect command**
   - *Correct command:* is mv

Part C

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

Commands: 1. vi hello.sh

2. Insert text into the vi Editor press Esc + I

3. save and exit: Esc + :wq

4.chmod +x hello.sh

5. ./hello.sh

```
cdac@Akash: ~
cdac@Akash:~$ vi hello.sh
cdac@Akash:~$ chmod +x hello.sh
cdac@Akash:~$ ./hello.sh
Hello,World!
cdac@Akash:~$
```

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

- vi var.sh
- chmod +x var.sh
- ./var.sh
- #!/bin/bash
- name="CDAC Mumbai"
- echo $name

```
cdac@Akash: ~
cdac@Akash:~$ vi var.sh
cdac@Akash:~$ chmod +x var.sh
cdac@Akash:~$ ./var.sh
CDAC Mumbai
cdac@Akash:~$ cat var.sh
#!/bin/bash

name="CDAC Mumbai"
echo $name

cdac@Akash:~$
```

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

- vi input.sh
- chmod +x input.sh
- ./input.sh

```
#!/bin/bash

echo -n "Enter the number : "

read num

echo "Number is : $num"
```

```
cdac@Akash: ~
cdac@Akash:~$ vi input.sh
cdac@Akash:~$ chmod +x input.sh
cdac@Akash:~$ ./input.sh
Enter the number : 12
Number is : 12
cdac@Akash:~$ cat input.
cat: input.: No such file or directory
cdac@Akash:~$ cat input.sh

#!/bin/bash

echo -n "Enter the number : "
read num
echo "Number is : $num"
cdac@Akash:~$
```

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

- vi add.sh
- chmod +x add.sh
- ./add.sh

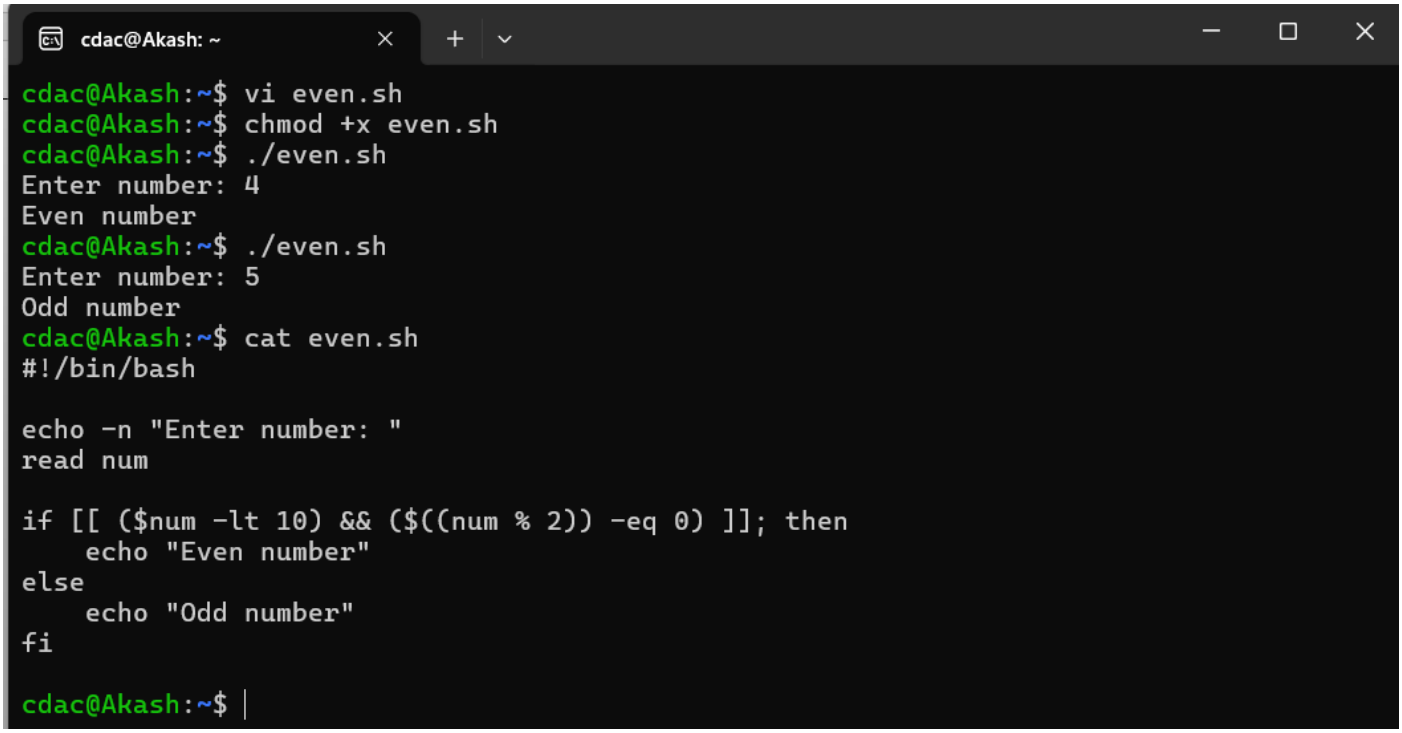
```
#!/bin/bash

a=5

b=3

sum=$((a+b))

echo "add two numbers  = $sum"
```

```
cdac@Akash: ~
cdac@Akash:~$ vi add.sh
cdac@Akash:~$ chmod +x add.sh
cdac@Akash:~$ ./add.sh
add two numbers  = 8
cdac@Akash:~$ cat add.sh
#!/bin/bash

a=5
b=3
sum=$((a+b))
echo "add two numbers  = $sum"
```

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

- vi even.sh
- chmod +x even.sh
- ./even.sh

```
#!/bin/bash

echo -n "Enter number: "

read num

  if [[ ($num -lt 10) && ($((num % 2)) -eq 0) ]]; then

    echo "Even number"

 else

   echo "Odd number"

 fi
```



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

- vi loop.sh
- chmod +x loop.sh
- ./loop.sh

```
#!/bin/bash

for i in {1..5}

do

  echo $i

done
```

```
cdac@Akash: ~
cdac@Akash:~$ vi loop.sh
cdac@Akash:~$ chmod +x loop.sh
cdac@Akash:~$ ./loop.sh
1
2
3
4
5
cdac@Akash:~$ cat loop.sh
#!/bin/bash

for i in {1..5}
do
  echo $i
done

cdac@Akash:~$ |
```

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

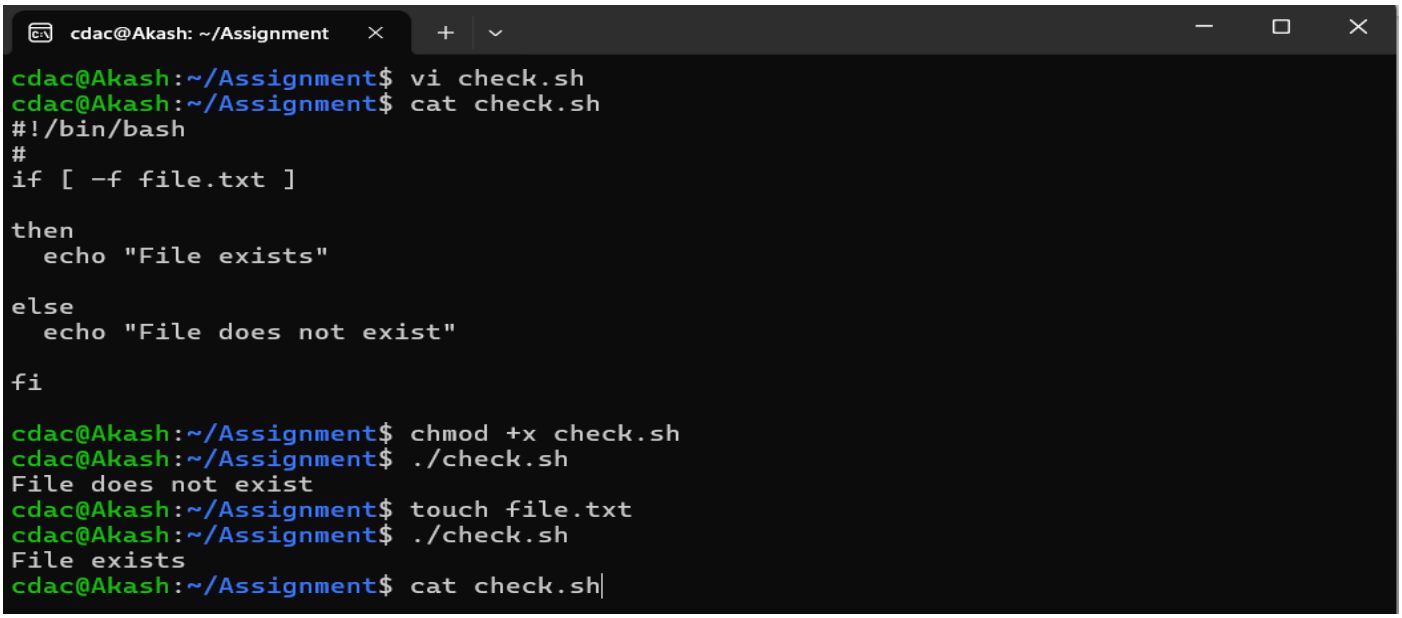
- vi while.sh
- chmod +x while.sh
- cat while.sh
- ./while.sh

```
cdac@Akash: ~/Assignment
cdac@Akash:~/Assignment$ vi while.sh
cdac@Akash:~/Assignment$ chmod +x while.sh
cdac@Akash:~/Assignment$ cat while.sh
#!/bin/bash

i=1

while [ $i -le 5 ]

do
  echo $i
  ((i++))

done

cdac@Akash:~/Assignment$ ./while.sh
1
2
3
4
5
cdac@Akash:~/Assignment$ |
```

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".

- vi check.sh
- cat check.sh
- chmod +x check.sh
- ./check.sh

```
#!/bin/bash

if [ -f file.txt ]

then

  echo "File exists"

else

  echo "File does not exist"

fi
```



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.

```
#!/bin/bash


echo -n "Enter a number: "

read num

 if [ $num -gt 10 ]

   then
```

```
    echo "The number is greater than 10"
  else
    echo "The number is not greater than 10"
fi
```

```
cdac@Akash: ~
cdac@Akash:~$ vi greater.sh
cdac@Akash:~$ chmod +x greater.sh
cdac@Akash:~$ ./greater.sh
./greater.sh: line 4: read: Enter a number: : invalid number
./greater.sh: line 6: [: -gt: unary operator expected
The number is not greater than 10
cdac@Akash:~$ vi greater.sh
cdac@Akash:~$ ./greater.sh
Enter a number: 12
The number is greater than 10
cdac@Akash:~$ cat greater.sh
#!/bin/bash

echo -n "Enter a number: "
read num

  if [ $num -gt 10 ]
    then
       echo "The number is greater than 10"

  else
    echo "The number is not greater than 10"

fi
```

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.

```
#!/bin/bash
for i in {1..5}
do
  for j in {1..5}
  do
    echo -n "$((i*j)) "
  done
  echo
done
```

```
cdac@Akash: ~
cdac@Akash:~$ vi table.sh
cdac@Akash:~$ cat table.sh
#!/bin/bash

for i in {1..5}

do

  for j in {1..5}

  do

    echo -n "$((i*j)) "

  done
  echo
done

cdac@Akash:~$ chmod +x table.sh
cdac@Akash:~$ ./table.sh
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@Akash:~$
```

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.

```
#!/bin/bash

while true

do

   read -p "Enter a number: " num

   if [ "$num" -lt 0 ]; then

      break

   fi

   echo "Square = $((num * num))"

done
```

```
cdac@Akash: ~
cdac@Akash:~$ vi square.sh
cdac@Akash:~$ cat square.sh
#!/bin/bash

while true
do
    read -p "Enter a number: " num

    if [ "$num" -lt 0 ]; then
        break
    fi

    echo "Square = $((num * num))"
done

cdac@Akash:~$ chmod +x square.sh
cdac@Akash:~$ ./square
-bash: ./square: No such file or directory
cdac@Akash:~$ ./square.sh
Enter a number: 4
Square = 16
Enter a number: -2
```

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.

(1)

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

∴ Gantt Chart: / sequence P1 → P2 → P3

P1	P2	P3

0 5 8 14

Process	CT	TAT = CT−AT	WT = TAT − BT
P1	5	5	0
P2	8	7	4
P3	14	12	6

∴ Average waiting Time (AWT)

$$= \frac{(0+4+6)}{3} = \frac{10}{3} = 3.33$$

× Average Turnaround (ATAT:

$$\frac{(5+7+12)}{3} = \frac{24}{3} = 8$$

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.

Q2)

Process	AT	BT
P1	0	3
P2	1	5
P3	2	1
P4	3	4

- SJF - Short Job First (non-preemptive)

Sequence: P1 → P3 → P4 → P2

- Gantt chart

P1	P3	P4	P2

0 3 4 8 13

(3) Completion | Turnaround | Waiting.

Process	CT	TAT = CT − AT	WT = TAT − BT
P1	3	3 − 0 = 3	3 − 3 = 0
P3	4	4 − 2 = 2	2 − 1 = 1
P4	8	8 − 3 = 5	5 − 4 = 1
P2	13	13 − 1	12 − 5 = 7

∴ Average Turnaround Time =

$$\frac{(3+2+5+12)}{4} = \frac{22}{4} = 5.5$$

∴ $AWT = \frac{(0+1+1+7)}{4} = \frac{9}{4} = 2.25$

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.

Q3) Priority Scheduling is a non-preemptive.

process	AT	BT	priority (lower → higher)
P1	0	6	3
P2	1	4	1
P3	2	7	4
P4	3	2	2

∴ Sequence: P1 → P2 → P4 → P3

Gantt chart:

P1	P2	P4	P3

0 6 10 12 19

CT	TAT = CT − AT	WT = TAT − BT
6	6 − 0 = 6	6 − 6 = 0
10	10 − 1 = 9	9 − 4 = 5
12	12 − 3 = 9	9 − 2 = 7
19	19 − 2 = 17	17 − 7 = 10

- Average Waiting Time

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

$= \frac{22}{4}$

WT $= 5.5$

- Average Turnaround Time

$= (6+9+9+17)$

TT $= 10.25$

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.

Q4)

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

∴ calculate average turnaround time.

Gantt chart! Time Quantum = 2

P1	P2	P3	P1	P4	P2	P4	P2

0 2 4 6 8 10 12 13 14

Sequence → P1 → P2 → P3 → P1 → P4 → P2 → P4

CT	TAT = CT − AT	WT = TAT − BT
8	8 − 0 = 8	8 − 4 = 4
14	14 − 1 = 13	13 − 5 = 8
6	6 − 2 = 4	4 − 2 = 2
13	13 − 3 = 10	10 − 3 = 7

Average Turnaround Time.

$$= \frac{8+13+4+10}{4} = \frac{35}{4}$$

$$= 8.75$$

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?

fork()

1. Before fork()
parents has $x=5$

2. After fork().
parent: $x=5$
child: $x=5$

Both Increment by 1.
parent: $x=6$
child: $x=6$

= parent process $x=6$
= child process $x=6$