Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

- echo "Hello, World!"

Prints the text "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 doesn't already exist. If it does exist, it updates the file's timestamp.

- ls -a

Lists all files and directories in the current directory, including hidden ones

- rm file.txt

Deletes the file named file.txt.

- cp file1.txt file2.txt

Copies the contents of file1.txt to file2.txt. If file2.txt doesn’t exist, it will be created.

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

Moves file.txt to the specified directory /path/to/directory/. If the directory doesn’t exist, the command will fail.

- chmod 755 script.sh

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

- grep "pattern" file.txt

Searches for the string "pattern" within file.txt and prints lines that match.

- kill PID

Sends a signal to terminate the process with the specified PID (Process ID).

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

Creates a directory named mydir, navigates into it, creates an empty file named file.txt, writes "Hello, World!" into file.txt, and finally displays the contents of file.txt.

- ls -l | grep ".txt"

Lists all files in long format (-l) and filters the list to only show files that contain ".txt" in their names.

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

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

- ls -l | grep "^d"

Lists all files and directories in long format and filters to show only directories

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

Recursively searches for the string "pattern" in all files within /path/to/directory/ and its subdirectories.

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

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

- chmod 644 file.txt

Changes the permissions of file.txt to 644, which means the owner can read and write, while group and others can only read.

- cp -r source_directory destination_directory

Recursively copies the source_directory and its contents to destination_directory.

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

Searches recursively from the directory /path/to/search for files with the .txt extension.

- chmod u+x file.txt

Adds execute permission to the file file.txt for the owner (u stands for user).

- echo $PATH

Displays the current value of the PATH environment variable, which is a list of directories where the shell looks for executable files.

Part B

Identify True or False:

- **ls** is used to list files and directories in a directory.
  True
- **mv** is used to move files and directories.
  True
- **cd** is used to copy files and directories.
  False - cd is used to change directories, not to copy files. cp is used to copy files and directories.

- **pwd** stands for "print working directory" and displays the current directory.
  True
- **grep** is used to search for patterns in files.
  True
- **chmod 755 file.txt** gives read, write, and execute permissions to the owner, and read and execute permissions to group and others.
  True
- **mkdir -p directory1/directory2** creates nested directories, creating directory2 inside directory1 if directory1 does not exist.
  True
- **rm -rf file.txt** deletes a file forcefully without confirmation.
  True

Identify the Incorrect Commands:

- **chmodx** is used to change file permissions.
  Incorrect - The correct command is chmod.
- **cpy** is used to copy files and directories.
  Incorrect - The correct command is cp.
- **mkfile** is used to create a new file.
  Incorrect - The correct command to create a file is touch.
- **catx** is used to concatenate files.
  Incorrect - The correct command is cat.
- **rn** is used to rename files.
  Incorrect - The correct command to rename files is mv.

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

```
#!/bin/bash

echo "Hello, World!"
```

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

```
#!/bin/bash

name="CDAC Mumbai"

echo "$name"
```

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

```
echo 'enter a number'

read number

echo $number
```

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

```
echo 'enter a number'

read num1

echo 'enter a number'

read num2

sum=`expr $num1 + $num2`

echo $sum
```

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

```
echo 'enter a number'

read num

if [ `expr $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.

```
for num in 1 2 3 4 5

do

    echo $num

done
```

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

```
i=1

while [ $i -lt 6 ]

do

    echo $i

    i=`expr $i + 1`

done
```

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".

```
if [ -e "file.txt" ]
then
    echo 'file exists'
else
    echo 'file does not exists'
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.

```
echo 'enter a number'
read num
if [ $num -lt 10 ]
then
    echo $num 'is less than 10'
else
    echo $num 'is 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.

```
for i in 1 2 3 4 5
do
    for j in 1 2 3 4 5 6 7 8 9 10
    do
        mul=`expr $i \* $j`
        echo $mul
    done
done
```

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 (negative to exit): " number
    if [ "$number" -lt 0 ]; then
        break
    fi
    square=$((number * number))
    echo "Square: $square"
done
```

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.

Average waiting time = 3.3

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.

Average turnaround time = 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.

Average waiting time = 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.

Average turnaround time = 8.5

Average turnaround time =

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 the context of the fork() system call:

- **Parent Process**:
    - Original value of x = 5
    - After increment: x = 5 + 1 = 6
- **Child Process**:
    - Inherits the original value of x from the parent, which is 5
    - After increment: x = 5 + 1 = 6

Both the parent and child processes will end up with x = 6 after the fork() call and subsequent increment.