PART A-

1.echo "Hello, World!": Prints "Hello, World!" to the terminal.

2.name="Productive": Assigns the value "Productive" to the variable name.

3.touch file.txt: Creates an empty file named file.txt if it doesn’t already exist.

4.ls -a: Lists all files and directories in the current directory, including hidden ones (those that begin with a dot).

5.rm file.txt: Deletes the file named file.txt.

6.cp file1.txt file2.txt: Copies file1.txt to file2.txt. If file2.txt already exists, it will be overwritten.

7.mv file.txt /path/to/directory/: Moves file.txt to the specified directory. If you provide a new filename, it will rename the file as it moves it.

8.chmod 755 script.sh: Changes the permissions of script.sh to be readable and executable by everyone, and writable by the owner.

9.grep "pattern" file.txt: Searches for the string "pattern" within file.txt and prints each line that contains it.

10.kill PID: Terminates the process with the specified process ID (PID).

11.mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt: Creates a directory named mydir, changes into mydir, creates an empty file named file.txt, writes "Hello, World!" to file.txt, and displays the contents of file.txt.

12.ls -l | grep ".txt": Lists all files in long format and filters the results to show only those with the ".txt" extension.

13.cat file1.txt file2.txt | sort | uniq: Concatenates file1.txt and file2.txt, sorts the combined contents, and removes duplicate lines.

14.ls -l | grep "^d": Lists all files in long format and filters the results to show only directories.

15.grep -r "pattern" /path/to/directory/: Recursively searches for the string "pattern" within all files in the specified directory and its subdirectories.

16.cat file1.txt file2.txt | sort | uniq –d: Concatenates file1.txt and file2.txt, sorts the combined contents, and displays only duplicate lines.

17.chmod 644 file.txt: Changes the permissions of file.txt to be readable and writable by the owner and readable by everyone else.

18.cp -r source_directory destination_directory: Recursively copies the contents of source_directory to destination_directory.

19.find /path/to/search -name "*.txt": Searches for all files ending with ".txt" within the specified path.

20.chmod u+x file.txt: Adds execute permission for the owner of file.txt.

21.echo $PATH: Prints the current value of the PATH environment variable.

PART B-

1.True. ls is indeed used to list files and directories in a directory.

2.True. mv is used to move files and directories.

3.False. cd is used to change directories, not to copy files and directories.

4.True. pwd stands for "print working directory" and displays the current directory.

5.True. grep is used to search for patterns in files.

6.True. chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute permissions to the group and others.

7.True. mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1 if directory1 does not exist.

8.True. rm -rf file.txt deletes a file forcefully without confirmation.

PART C-

1.chmodx is not a valid command. The correct command for changing file permissions is chmod.

2.cpy is not a valid command. The correct command for copying files and directories is cp.

3.mkfile is not a valid command. There is no standard command called mkfile for creating a new file. You can use touch to create an empty file.

4.catx is not a valid command. The correct command for concatenating files is cat.

5.rn is not a valid command. The correct command for renaming files is mv.

PART C-

1.

```
#!/bin/bash
echo "Hello, World!"
```

2.

```
#!/bin/bash
```

```
name="CDAC Mumbai"
echo $name
```

3.

```
#!/bin/bash
echo "Please enter a number:"
read number
echo "You entered: $number"
```

4.

```
#!/bin/bash
num1=5
num2=3
sum=$((num1 + num2))
echo "The sum is: $sum"
```

5.

```
#!/bin/bash
echo "Please enter a number:"
read number

if [ $((number % 2)) -eq 0 ]; then
  echo "Even"
else
  echo "Odd"
fi
```

6.

```
#!/bin/bash
for i in {1..5}
do
  echo $i
done
```

7.

```
#!/bin/bash
i=1
while [ $i -le 5 ]
do
  echo $i
  i=$((i + 1))
done
```

8.

```
#!/bin/bash
if [ -e file.txt ]; then
  echo "File exists"
else
  echo "File does not exist"
```

```
fi
```

9.

```
#!/bin/bash
echo "Please enter a number:"
read number

if [ $number -gt 10 ]; then
  echo "The number is greater than 10"
else
  echo "The number is not greater than 10"
fi
```

10.

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

11.

```
#!/bin/bash
while true
do
  echo "Please enter a number:"
  read number

  if [ $number -lt 0 ]; then
    break
  fi

  square=$((number * number))
  echo "The square of $number is $square"
done
echo "A negative number was entered. Exiting the loop."
```

PART E-

1.

Time 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
P1 P1 P1 P1 P1 P2 P2 P2 P3 P3 P3 P3 P3 P3

Waiting Time for
P1: 0

P2: 4
P3: 6

Average waiting time=0+4+6=9/3=3.33 units of time.

2.

Time 0 1 2 3 4 5 6 7 8 9 10 11 12

P1 P1 P1 P3 P4 P4 P4 P4 P2 P2 P2 P2 P2

Turnaround Time for-
P1: 3
P2: 11
P3: 2
P4: 5

Total Turnaround Time = 3 + 11 + 2 + 5 = 21
Average Turnaround Time = Total Turnaround Time / Number of Processes = 21 / 4 = 5.25 units of time

3.

Time 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
P1 P1 P1 P1 P1 P1 P2 P2 P2 P2 P4 P4 P3 P3 P3 P3 P3 P3 P3

Waiting Time for-
P1: 0
P2: 5
P3: 10
P4: 7

Total Waiting Time = 0 + 5 + 10 + 7 = 22

Average Waiting Time = Total Waiting Time / Number of Processes = 22 / 4 = 5.5 units of time

4.

Time 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
P1 P1 P2 P2 P3 P4 P1 P2 P2 P4 P4 P2 P2

Turnaround Time for-
P1: 8

P2: 13
P3: 4
P4: 8

Total Turnaround Time = 8 + 13 + 4 + 8 = 33

Average Turnaround Time = Total Turnaround Time / Number of Processes = 33 / 4 = 8.25 units of time

5.

In a program that uses the fork() system call, the child process gets a copy of the parent's address space, including the variable x. After the fork, the parent and child processes have their own separate copies of the variable x.

Initially, the parent process has a variable x with a value of 5. After the fork() call, both the parent and child processes will have their own copies of x, both initially set to 5. When both processes increment their respective copies of x by 1, they do so independently.

Therefore, the final values of x will be:

In the parent process: x = 6

In the child process: x = 6

The parent and child processes do not share the same memory space for the variable x, so the changes made by one process do not affect the other.