**Concepts of Operating System**

**Assignment 2**

**Part A**

What will the following commands do?

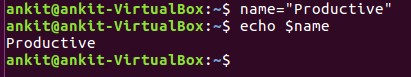
**echo "Hello, World!"**

Prints Hello, World! Statement



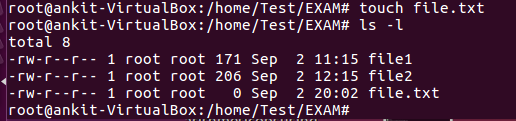
**name="Productive"**

Saves Productive in name variable and can be printed on command line using echo $name



**touch file.txt**

Create a new file.txt



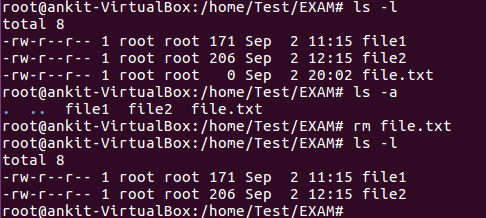
**ls -a**

List all files including hidden files



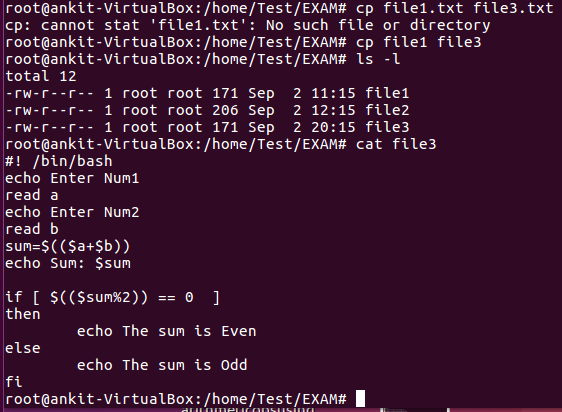
**rm file.txt**

deletes file named “file.txt”



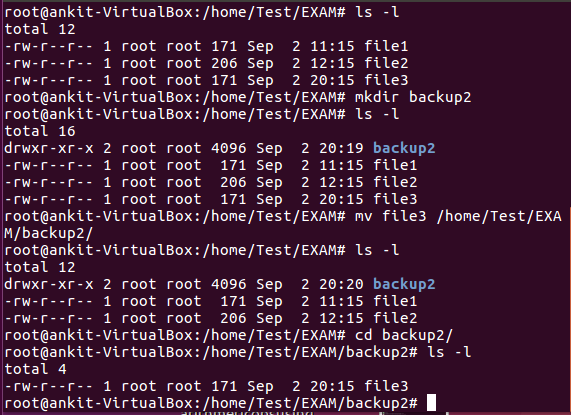
**cp file1.txt file2.txt**

Copies the content of file1 into file2



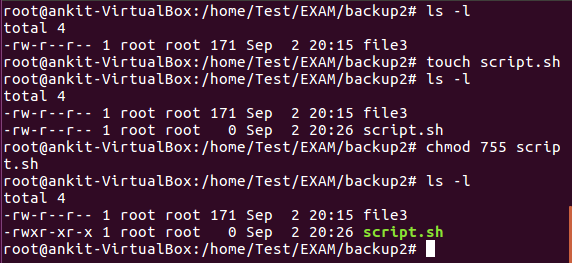
**mv file.txt /path/to/directory/**

Moves file.txt to specified directory



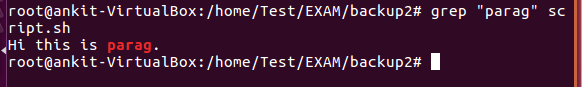
**chmod 755 script.sh**

Give read write execute permission to user and read execute permission to group and other on file script.sh



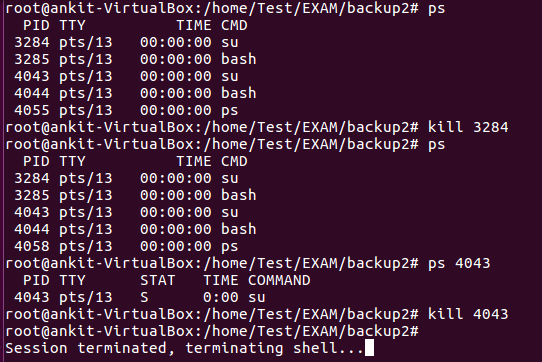
**grep "pattern" file.txt**

Finds the word specified in the file.



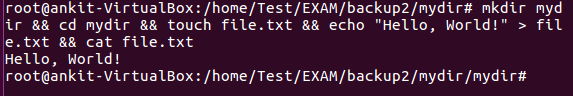
**kill PID**

Kills the process ID specified.



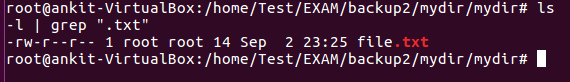
**mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**

A new directory mydir gets created then changes the current working directory to mydir. A new file file.txt will be created. Inputs Hello, World! statement in file.txt. Displays Hello, World!.



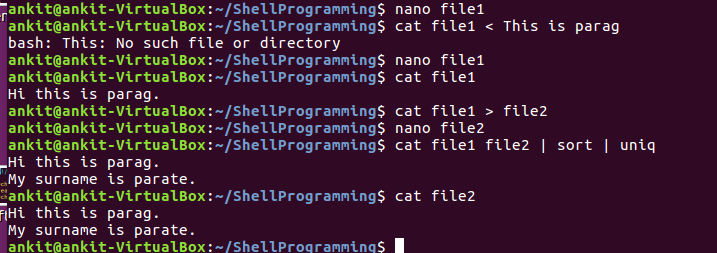
**ls -l | grep ".txt"**

List the files with .txt in their names.



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

Concatenates the content of the two files, sorts them and displays the unique values if the two strings are identical.



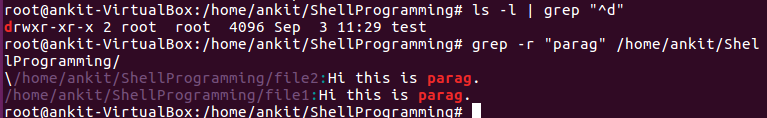
**ls -l | grep "^d”**

Lists down the directories present in the current directory.



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

Will search the pattern on the specified path files and display the names of file containing the pattern



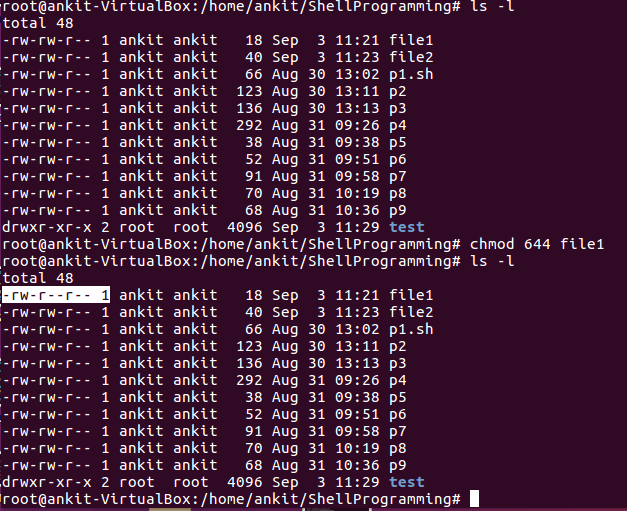
**cat file1.txt file2.txt | sort | uniq –d**

Shows the duplicate between the two files.



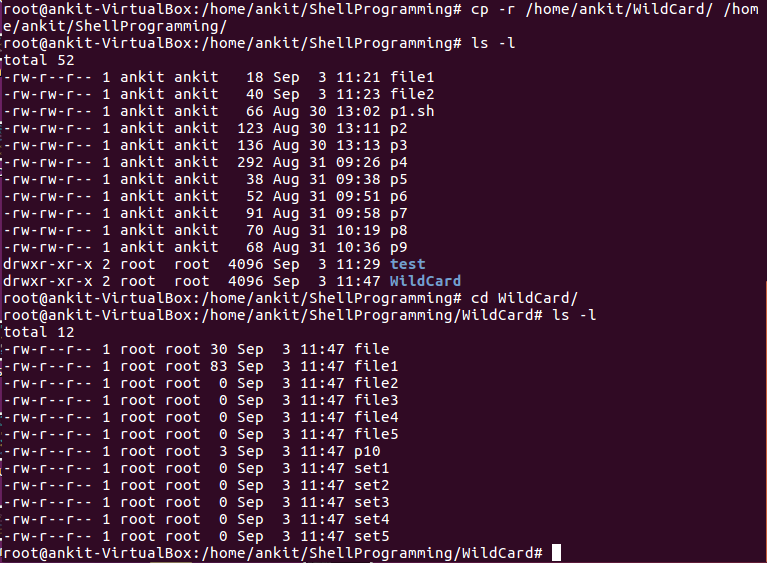
**chmod 644 file.txt**

Gives read and write permissions to users and read permissions to group & others.

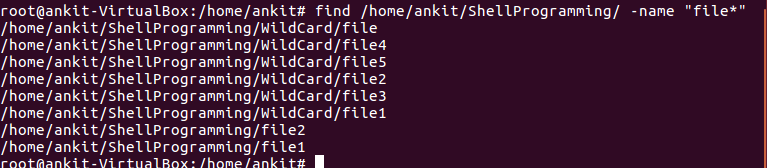


**cp -r source\_directory destination\_directory**

It is used to copy one directory into another.

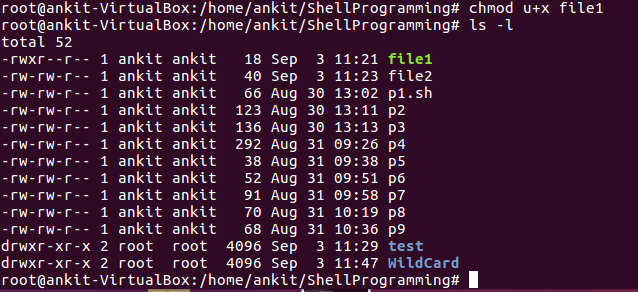


**find /path/to/search -name "\*.txt"**

Used to retrieve all the text files from the given path.

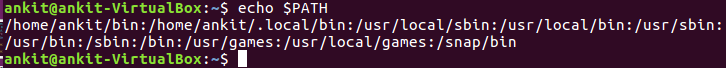
**chmod u+x file.txt**

Gives execute permission to user.



**echo $PATH**

It will show you different paths separated by ':' for the user. /usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin. If you type a command system will search these locations for a binary. If there are no commands with the name in those locations system will tell you ' Command not found '



**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. -chmod

2. cpy is used to copy files and directories. -cp

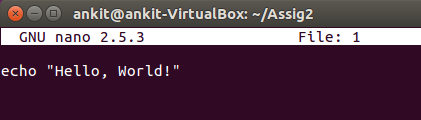
3. mkfile is used to create a new file. -nano, touch, cat

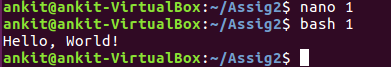
4. catx is used to concatenate files. -cat

5. rn is used to rename files. -mv

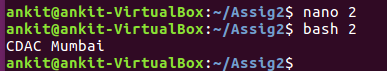
**Part C**

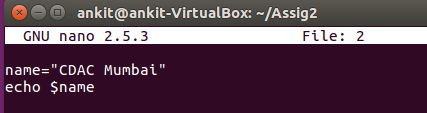
Question 1: Write a shell script that prints "Hello, World!" to the terminal.



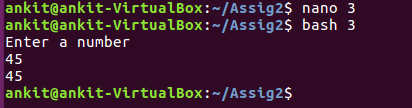


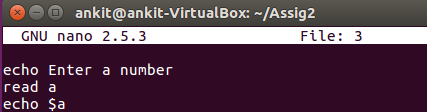
Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.



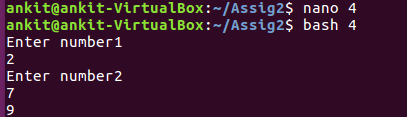


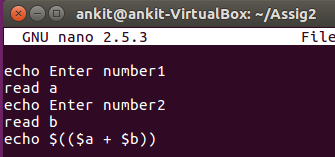
Question 3: Write a shell script that takes a number as input from the user and prints it.



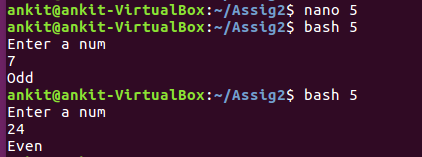


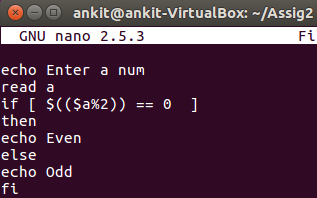
Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.



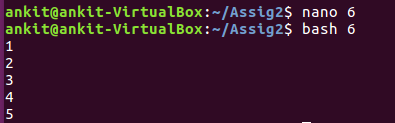


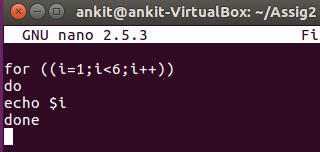
Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".



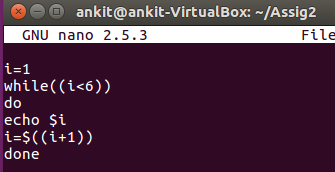


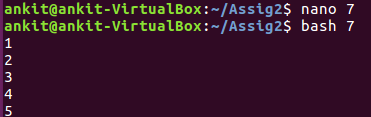
Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.



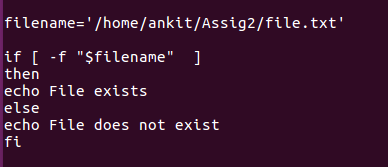


Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.



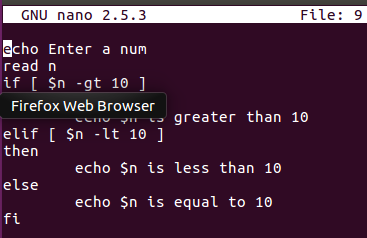


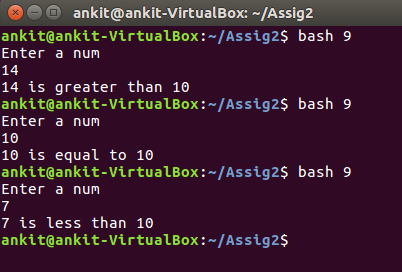
**Question 8:** Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".



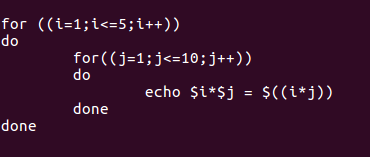


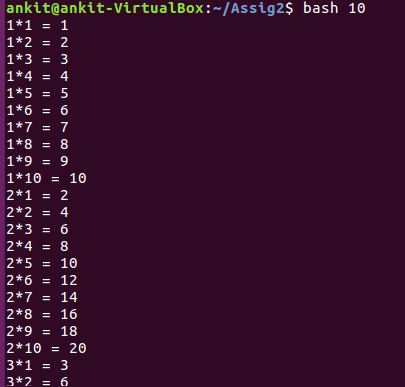
Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.



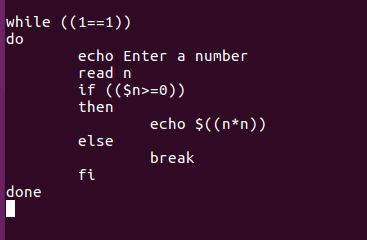


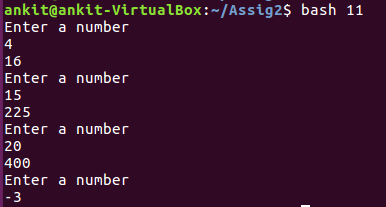
Question 10: Write a shell script that uses nested for loops to print a multiplication table for numbers from 1 to 5. The output should be formatted nicely, with each row representing a number and each column representing the multiplication result for that number.





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.





**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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | Arrival Time | Burst time | Completion Time | Turn Around Time | Waiting Time |
| P1 | 0 | 5 | 5 | 5 | 0 |
| P2 | 1 | 3 | 8 | 7 | 4 |
| P3 | 2 | 6 | 14 | 12 | 6 |

Average Waiting Time = 10/3 = 3.33 msec

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Completion Time | Turn Around Time |
| P1 | 0 | 3 | 3 | 3 |
| P2 | 1 | 5 | 13 | 12 |
| P3 | 2 | 1 | 4 | 2 |
| P4 | 3 | 4 | 8 | 5 |

Average Turn Around Time = 22/4 = 5.5 msec

**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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Priority | Completion Time | Turn Around Time | Waiting Time |
| P1 | 0 | 5 | 3 | 11 | 11 | 6 |
| P2 | 1 | 4 | 1 | 5 | 4 | 0 |
| P3 | 2 | 7 | 4 | 18 | 16 | 9 |
| P4 | 3 | 2 | 2 | 7 | 4 | 2 |

Average Waiting Time = 17/4 = **4.25 msec**

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Completion Time | Turn Around Time |
| P1 | 0 | 4 | 10 | 10 |
| P2 | 1 | 5 | 14 | 13 |
| P3 | 2 | 2 | 6 | 4 |
| P4 | 3 | 3 | 13 | 10 |

Average Turn Around Time = 37/4 = 9.25 msec

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?

The final value of x in both the parent and child processes will be 6.