**Operating Systems**

**(CSL 303)**

Lab Practical Report



Faculty name: Priyanka Vashisht

Student name: Kushagra Bhatnagar

Roll No: 18csu113

Semester: V

Group: DS-B

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2019-20

**INDEX**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No** | **Experiment** | **Page No.** | **Date of Experiment** | **Date of Submission** | **Marks** | **CO Covered** | **Signature** |
| **1.** | Installation of linux operating system – Cygwin |  |  |  |  |  |  |
| **2.** | Perform basic Linux commands . |  |  |  |  |  |  |
| **3.** | Write a shell program to find factorial of a number. |  |  |  |  |  |  |
| **4.** | Write a shell program to find gross salary. |  |  |  |  |  |  |
| **5.** | Write a shell program to display menu and execute instructions. |  |  |  |  |  |  |
| **6.** | Write a shell program to find Fibonacci series. |  |  |  |  |  |  |
| **7.** | Write a shell program to find largest of 3 numbers. |  |  |  |  |  |  |
| **8.** | Write a shell program to find average of n numbers. |  |  |  |  |  |  |
| **9.** | Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time.  a) FCFS b) SJF c) Round Robin (pre-emptive) d) Priority |  |  |  |  |  |  |
| **10.** | Write a program to simulate Bankers algorithm for the purpose of deadlock avoidance |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Experiment-1**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students with Linux Operating System Commands.

**Program Outcome**

* The students will be able to perform Linux commands .

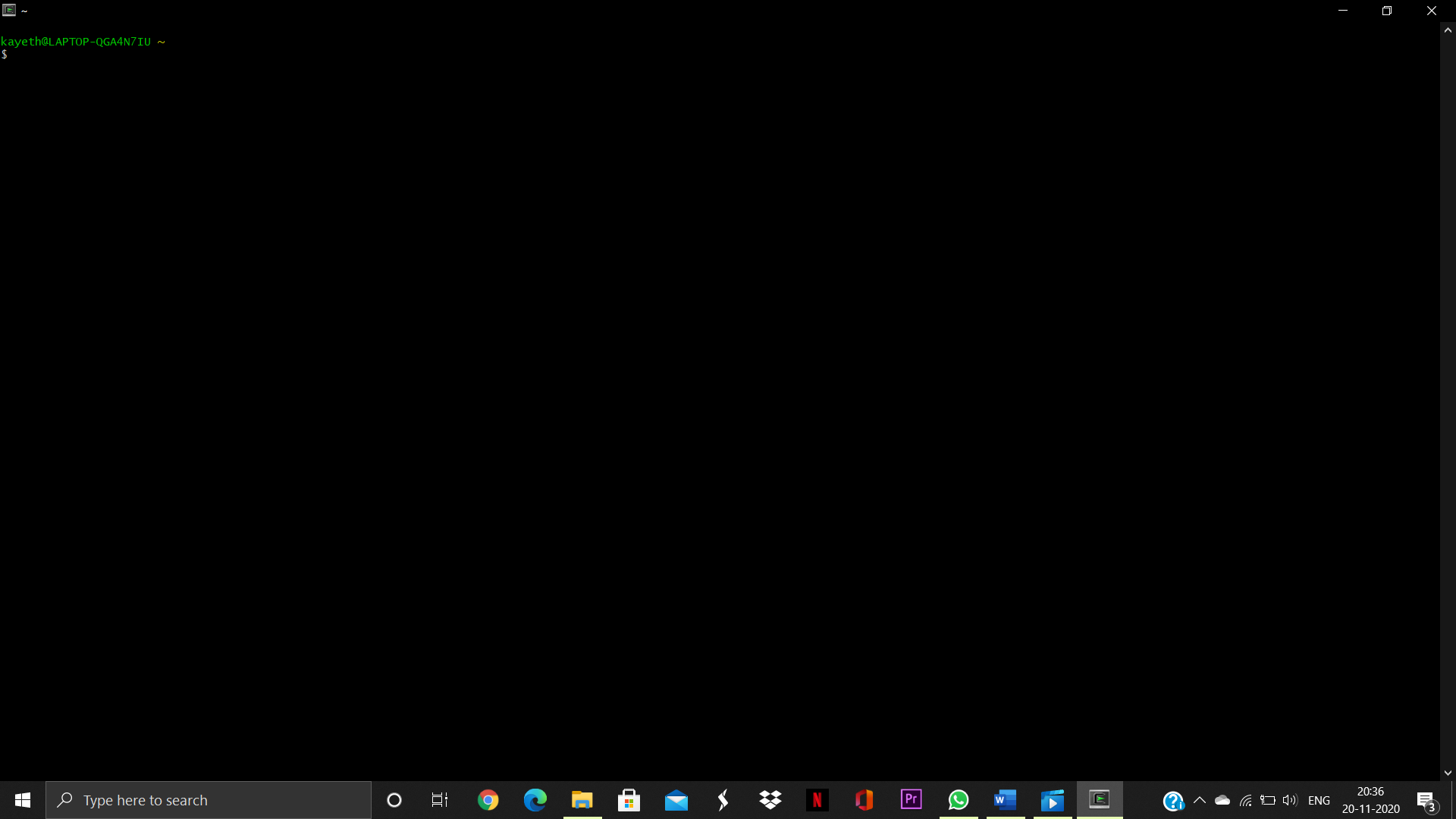
**Problem Statement**

Installation of Linux operating system - Cygwin

**Algorithm/ flowchart**

**Code :**

**Output: Successfully installed .**



**Experiment-2**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students with Linux Operating System Commands.

**Program Outcome**

* The students will be able to perform Linux commands .

**Problem Statement**

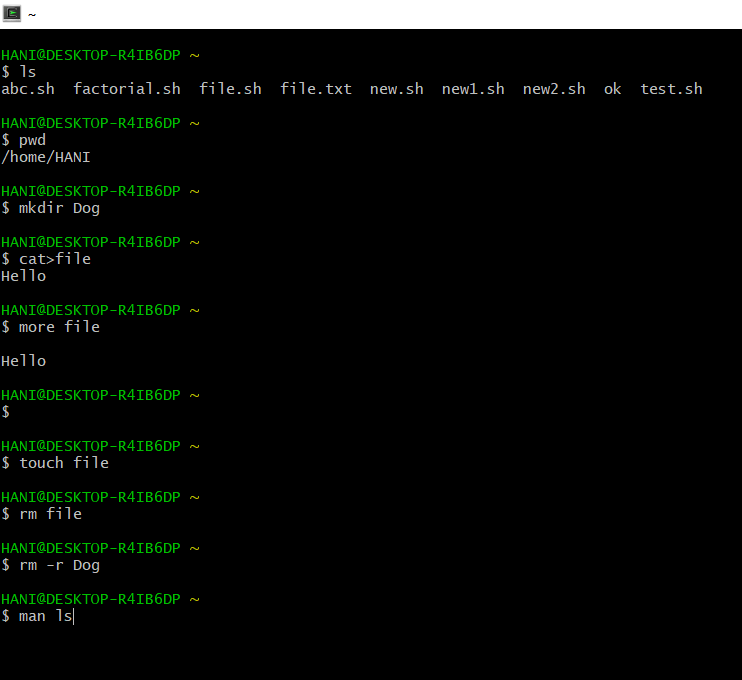
Performing basics Linux Commands .

**Algorithm/ flowchart**

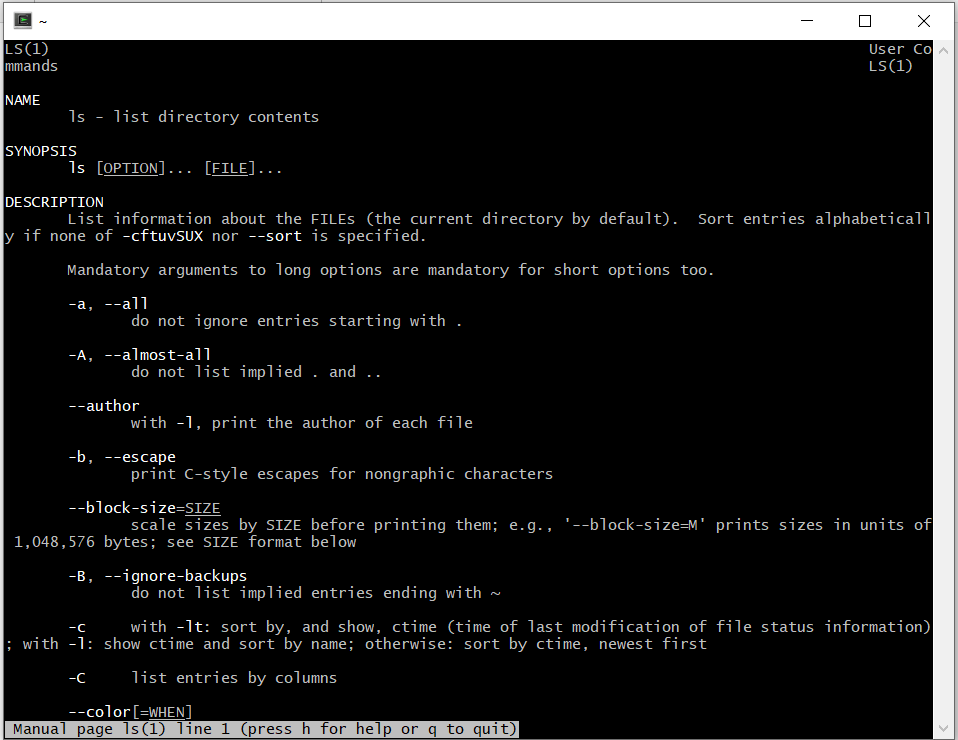
**Code :**

**Output: Successfully installed .**

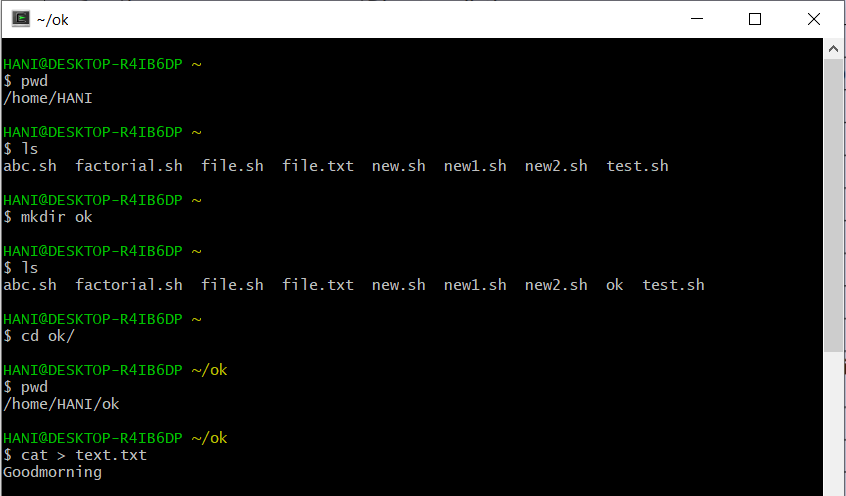
1. Creating and Deleting Directory & Files

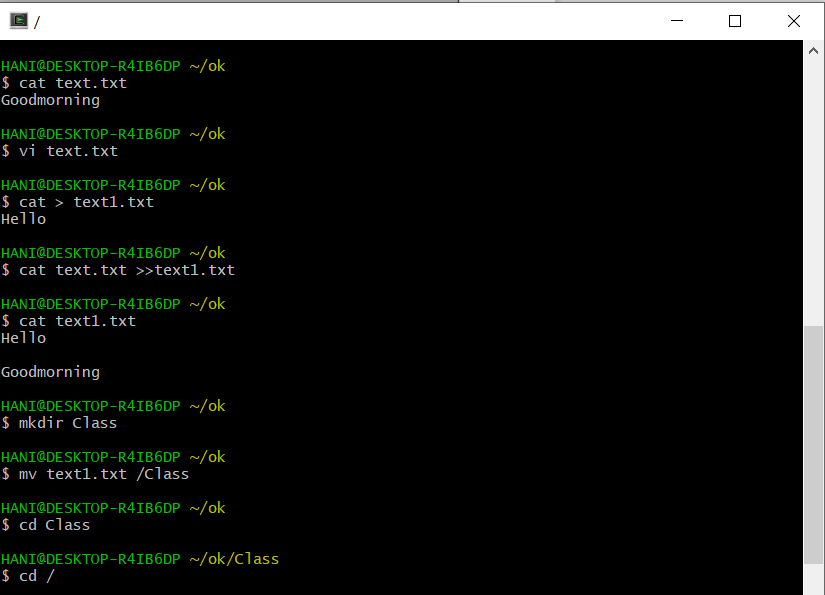


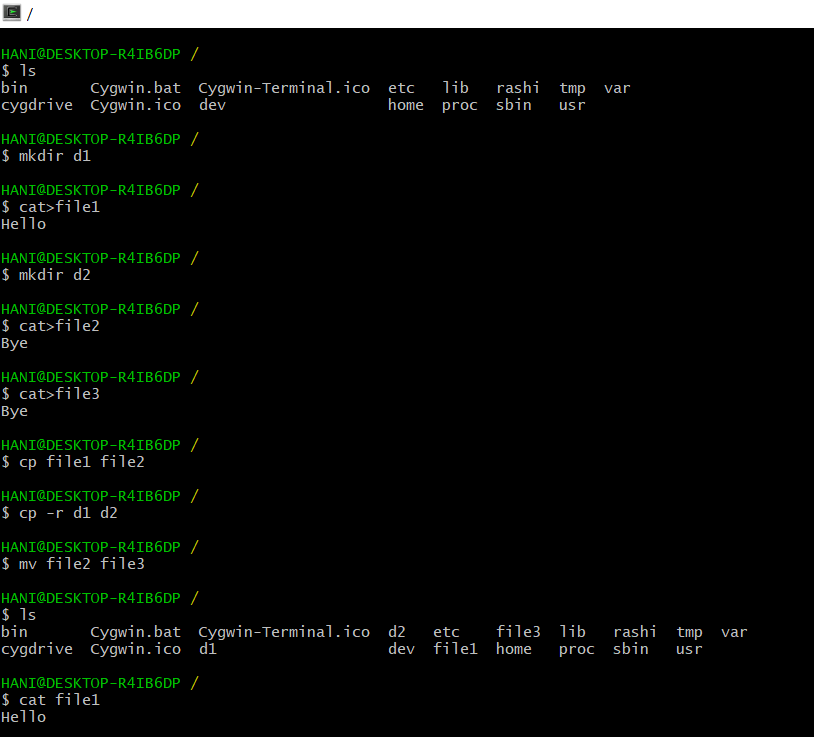
1. For Help: man ls



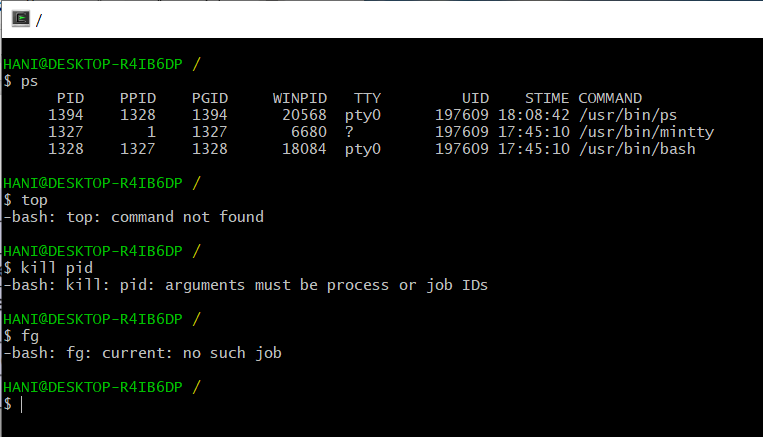
1. Copying the Contents

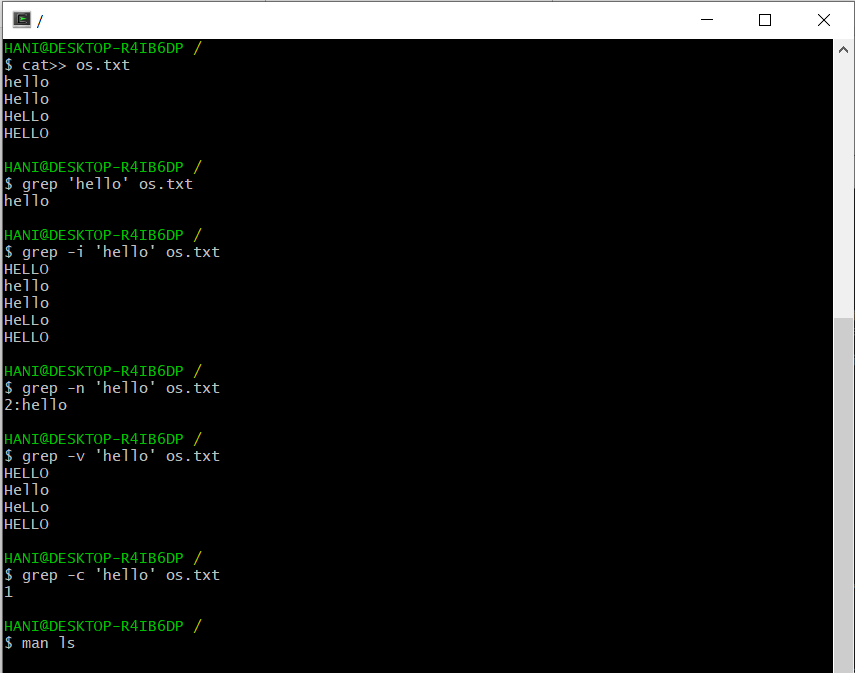


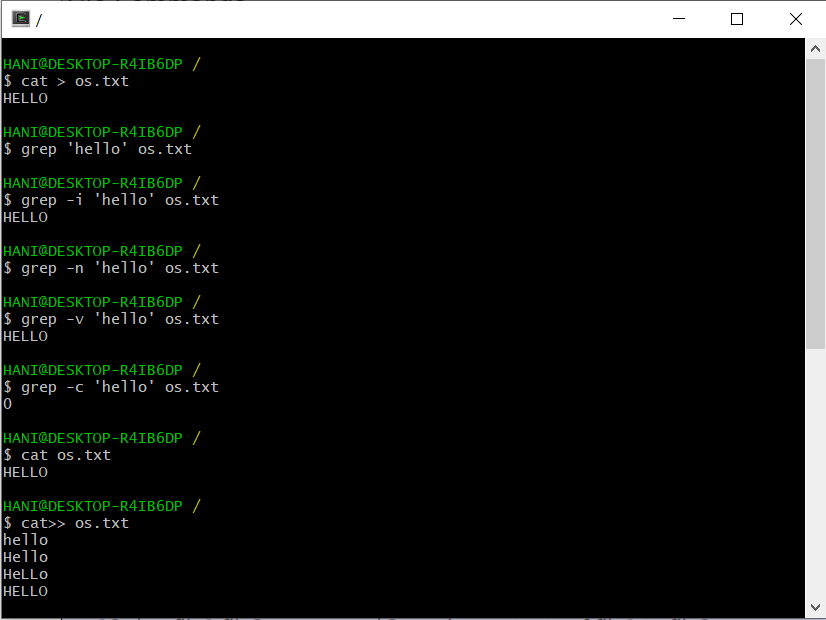




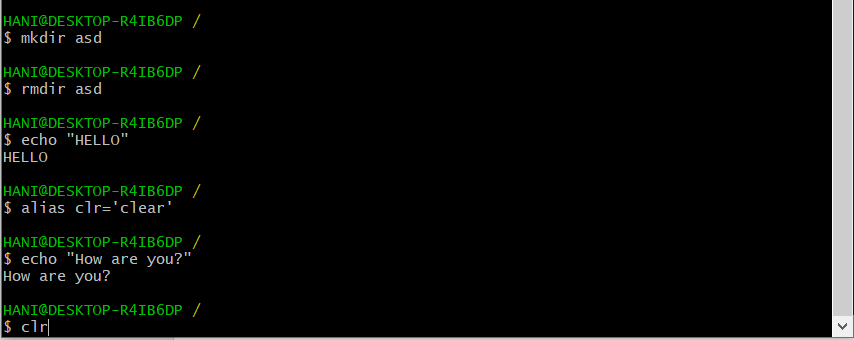
1. Process Management



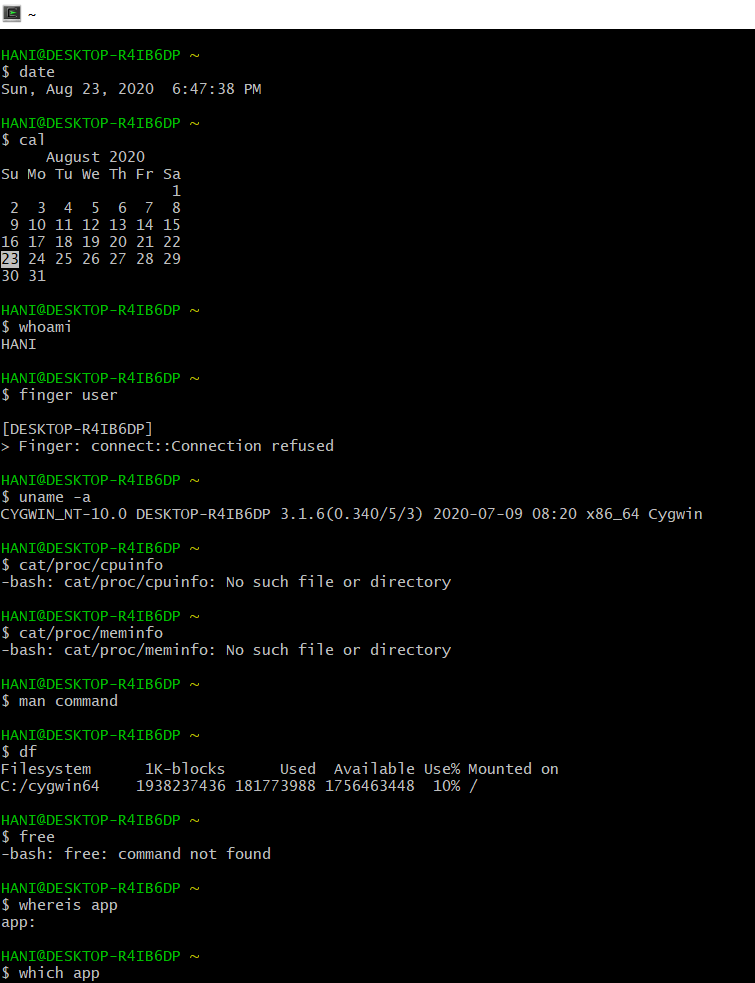
1. Grep Command

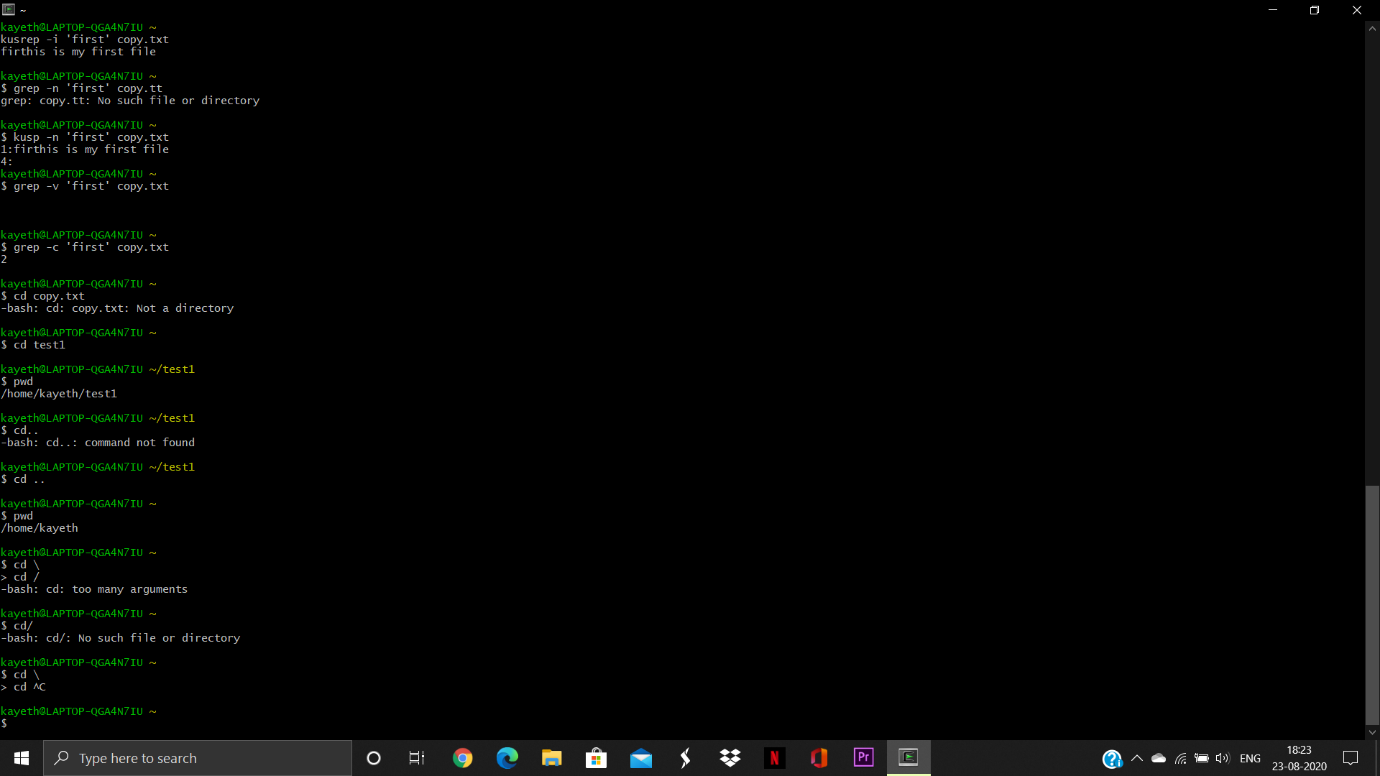
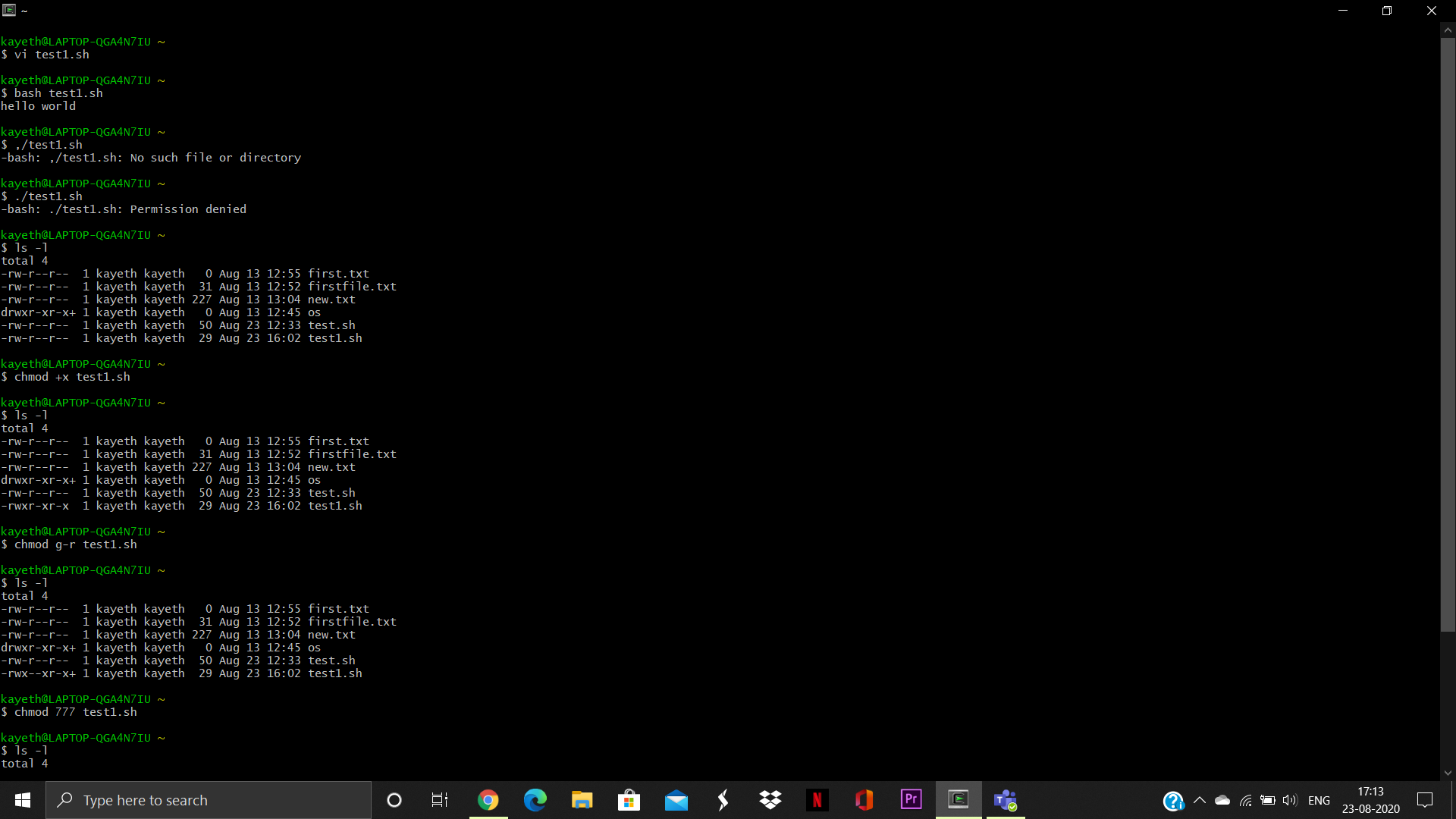
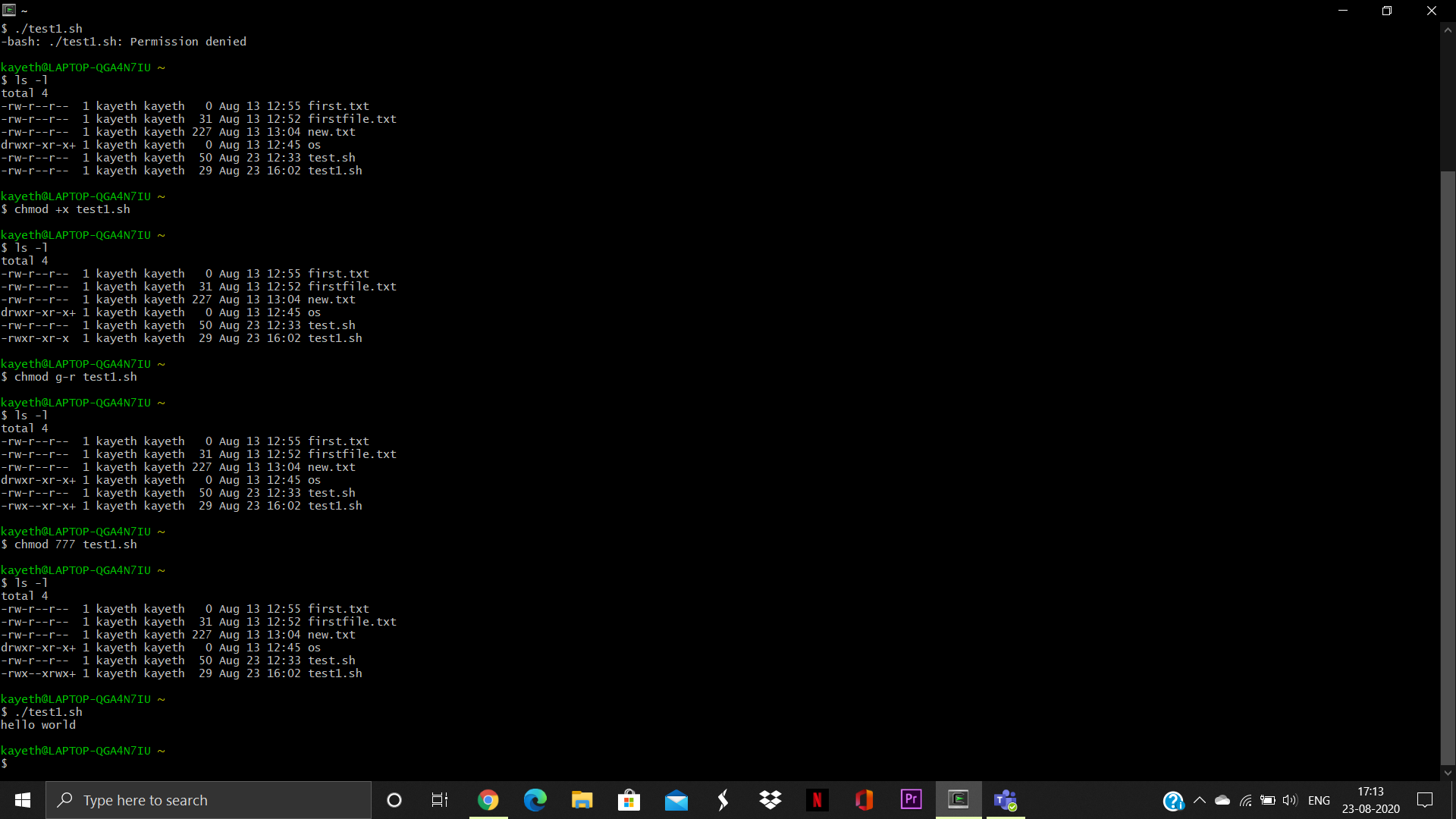
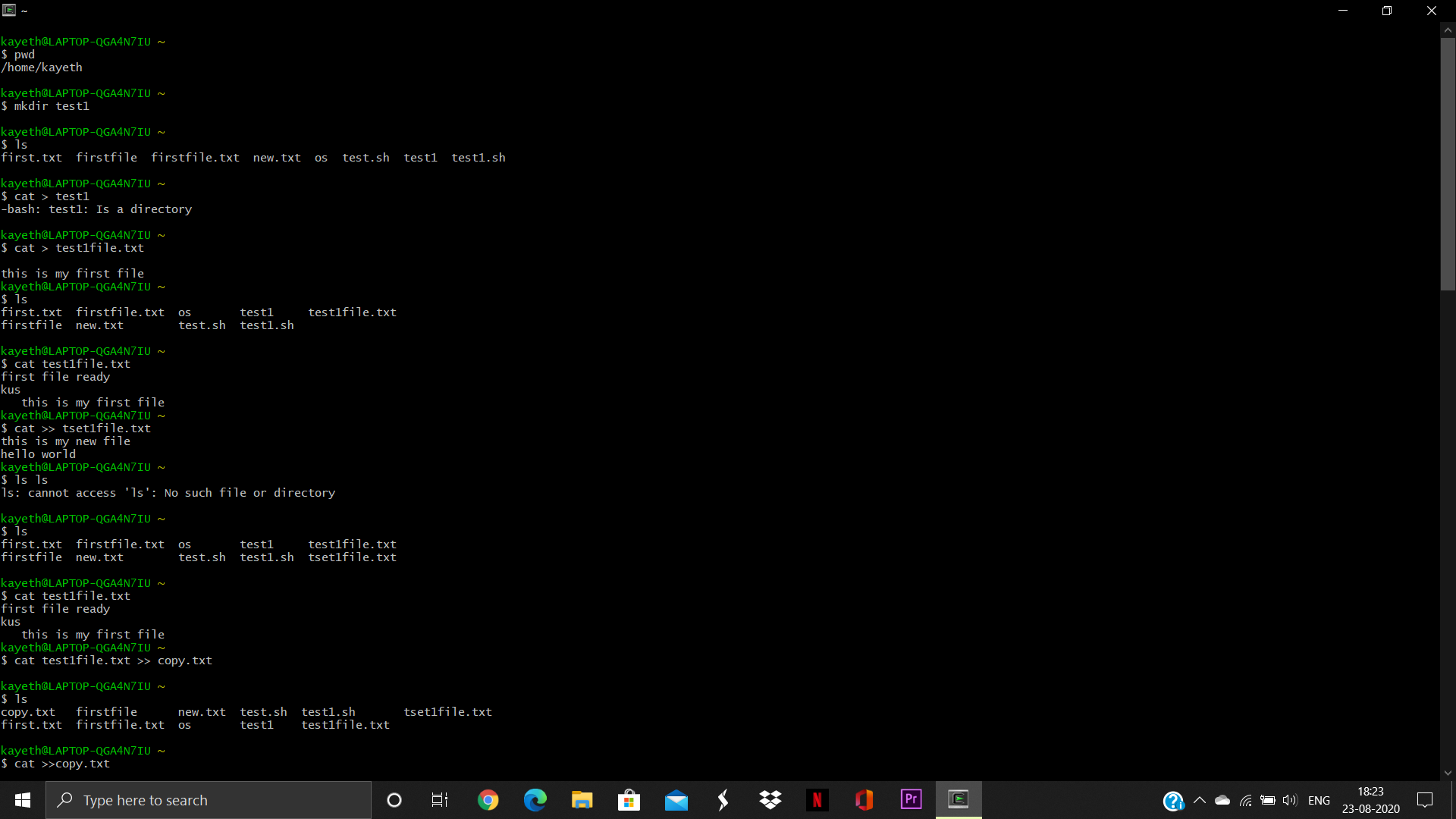
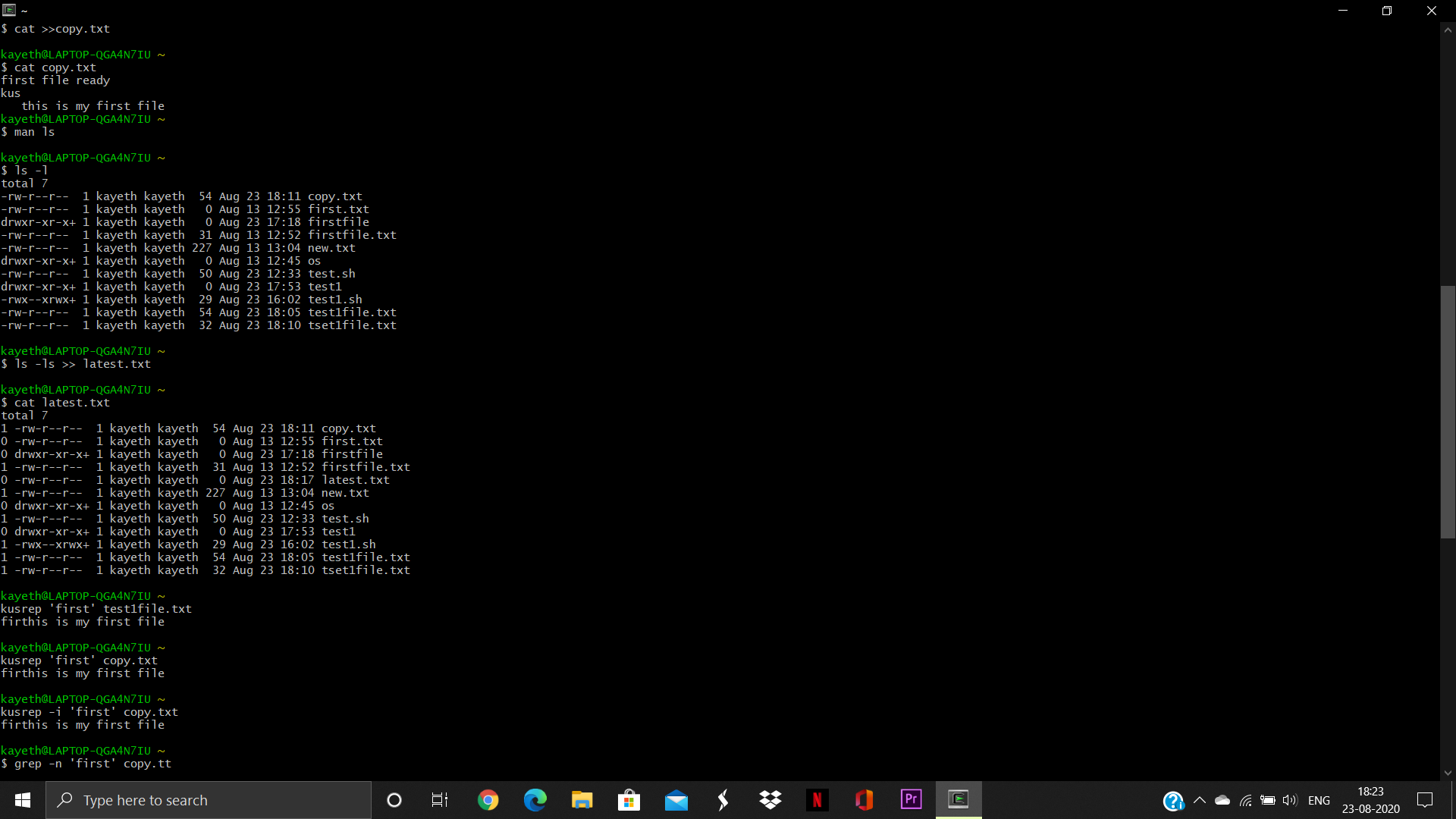
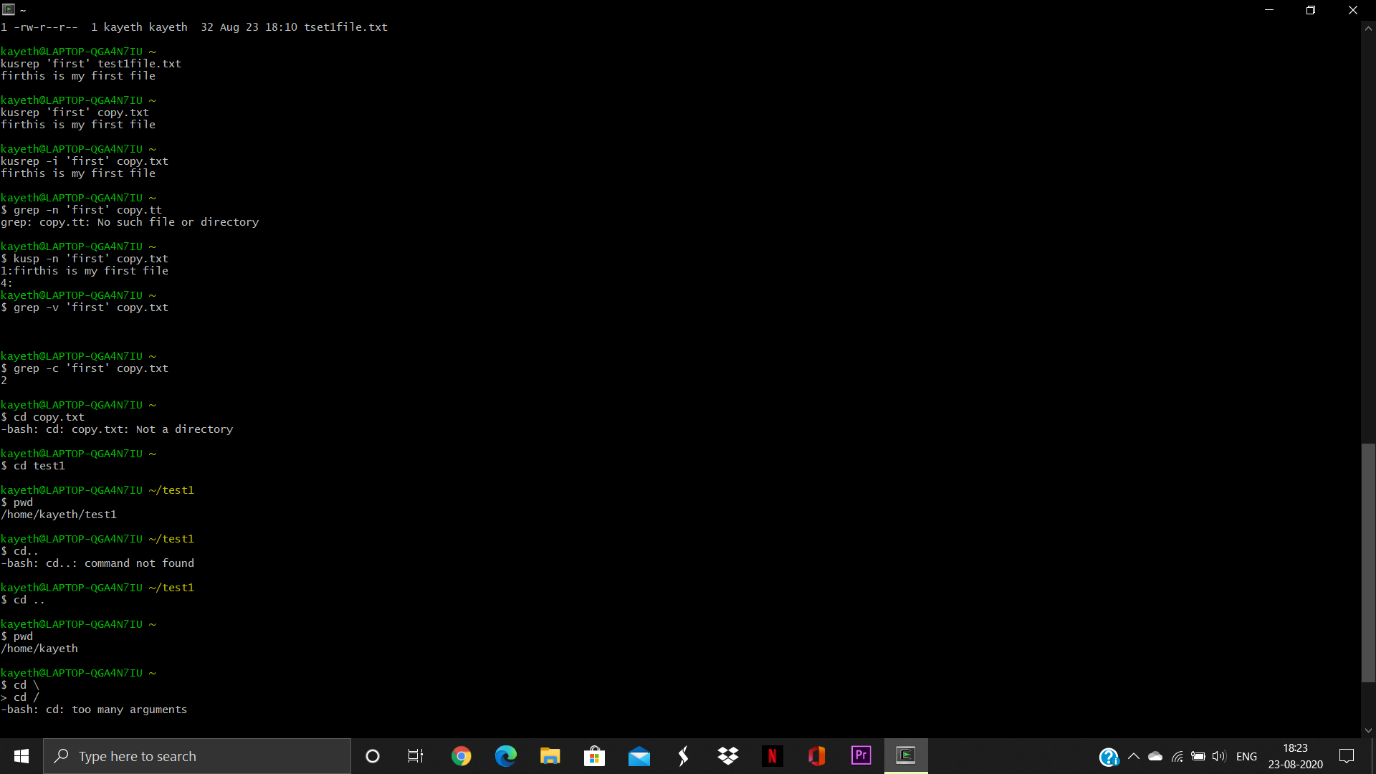


1. Echo & Alias Function



1. System info





**Experiment-3**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students with Linux Operating System Commands.

**Program Outcome**

* The students will be able to perform Linux commands .

**Problem Statement**

Write a shell program to find factorial of a number.

**Algorithm/ flowchart:**

START

Step 1 → Take integer variable A

Step 2 → Assign value to the variable

Step 3 → From value A upto 1 multiply each digit and store

Step 4 → the final stored value is factorial of A

STOP

**Code :**

#!bin/sh

Echo “enter a number”

Read num

Fact = 1

For((i=2;i<=num;i++))

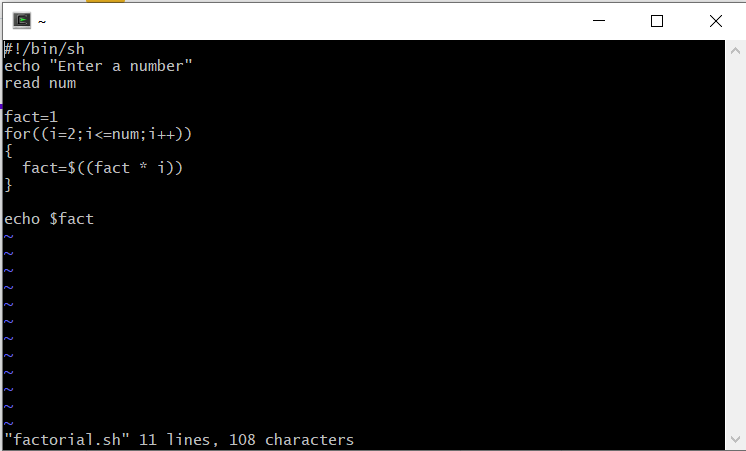
{

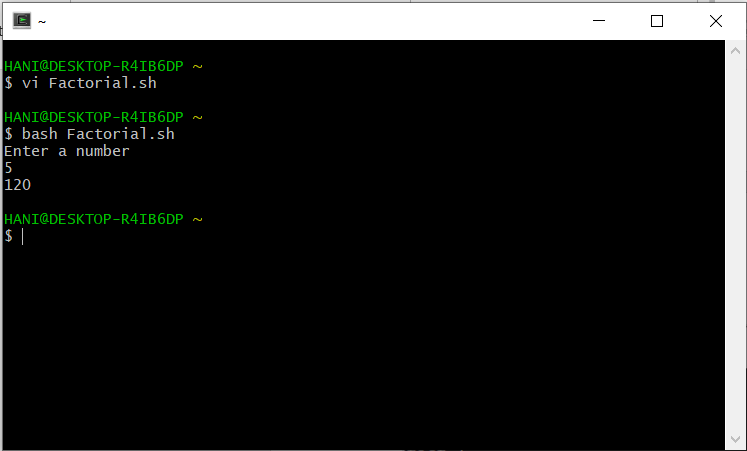
Fact=$((fact \* i))

}

Echo $fact

**Output:**





**Experiment-4**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students with Linux Operating System Commands.

**Program Outcome**

* The students will be able to perform Linux commands .

**Problem Statement**

Write a shell program to find gross salary.

**Algorithm/ flowchart:**

i.If basic salary is < 1500 then HRA =10% of the basic and DA =90% of the basic.

II. If basic salary is >=1500 then HRA =Rs500 and DA=98% of the basic

The basic salary is entered interactively through the key board

**Code :**

vi gsalary.sh

echo "enter the basic salary:"

read bsal

if [ $bsal -lt 1500 ]

then gsal=$((bsal+((bsal/100)\*10)+(bsal/100)\*90))

echo "The gross salary : $gsal"

fi

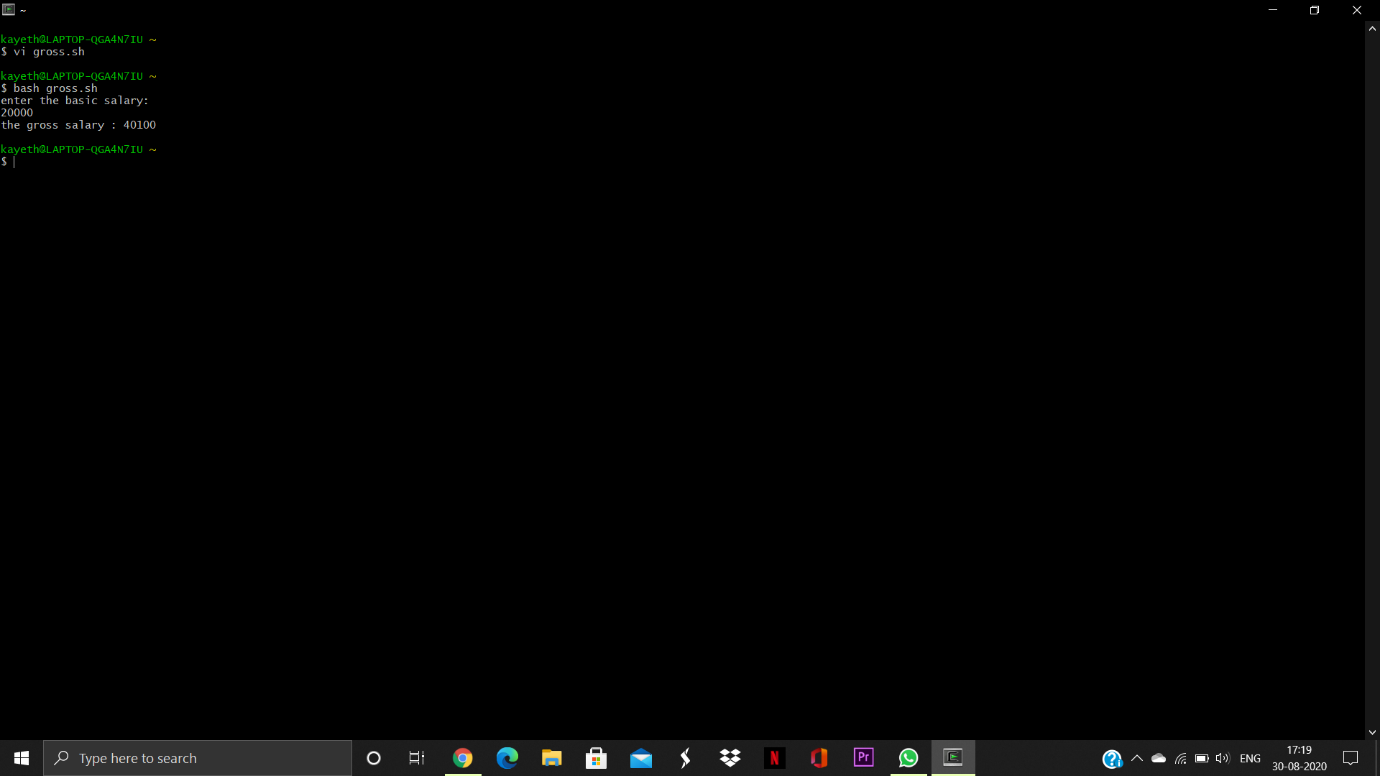
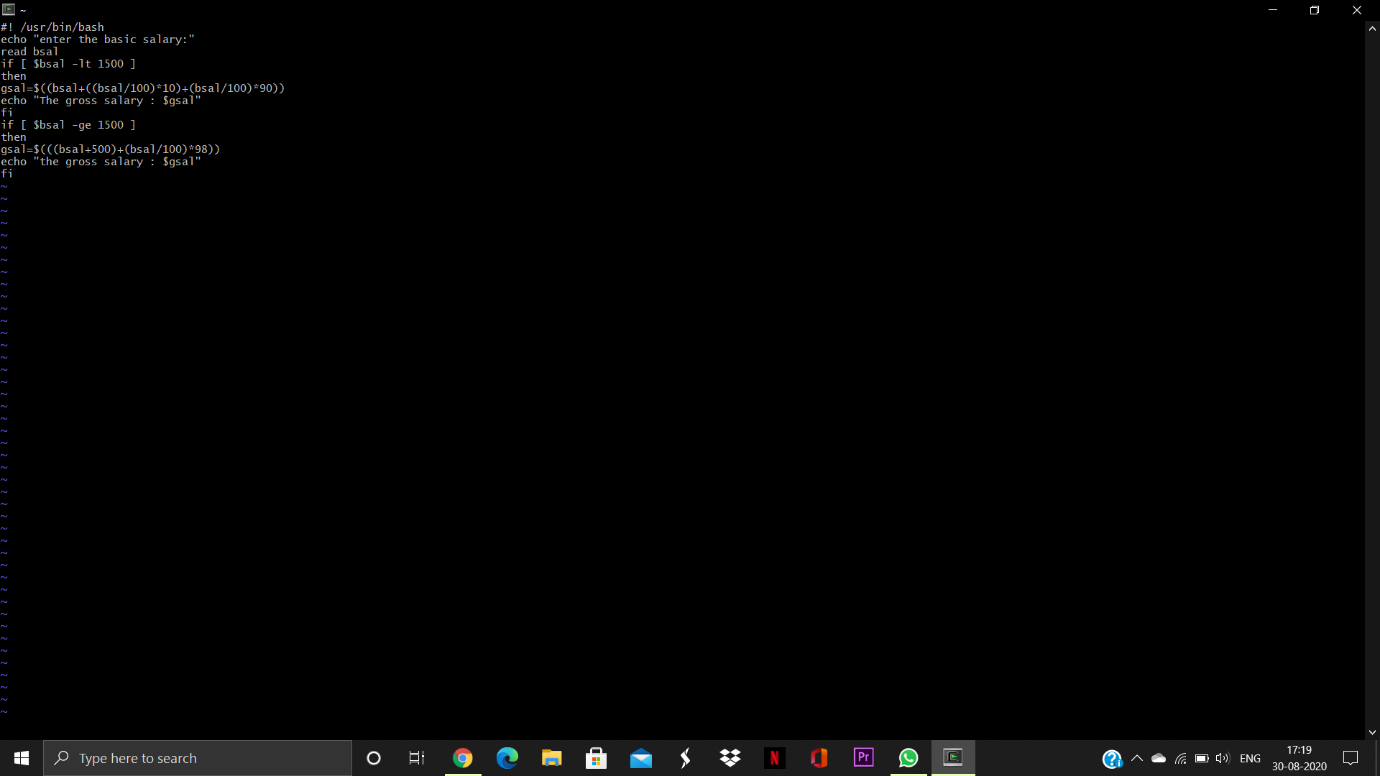
if [ $bsal -ge 1500 ]

then

gsal=$(((bsal+500)+(bsal/100)\*98))

echo "the gross salary : $gsal"

fi

**Output:** 

**Experiment-5**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students with Linux Operating System Commands.

**Program Outcome**

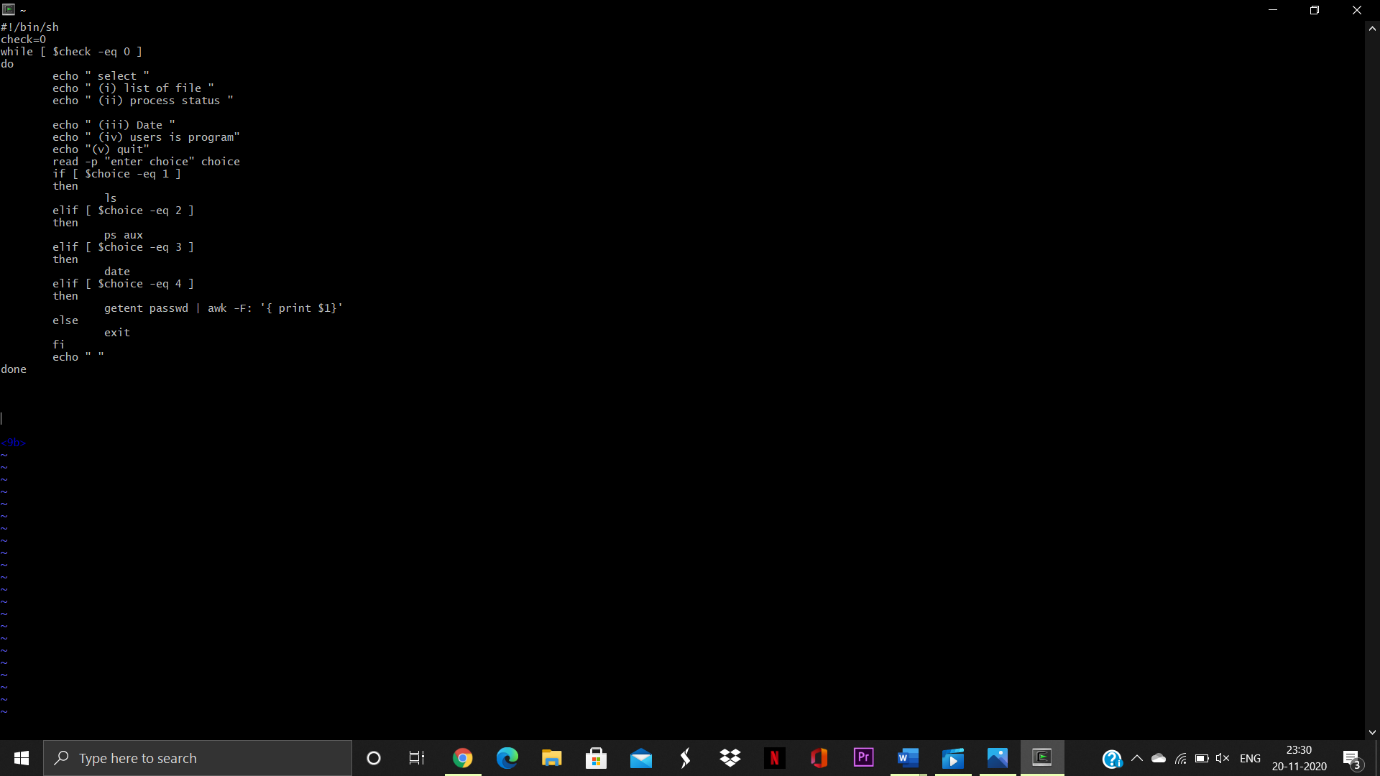
* The students will be able to perform Linux commands .

**Problem Statement**

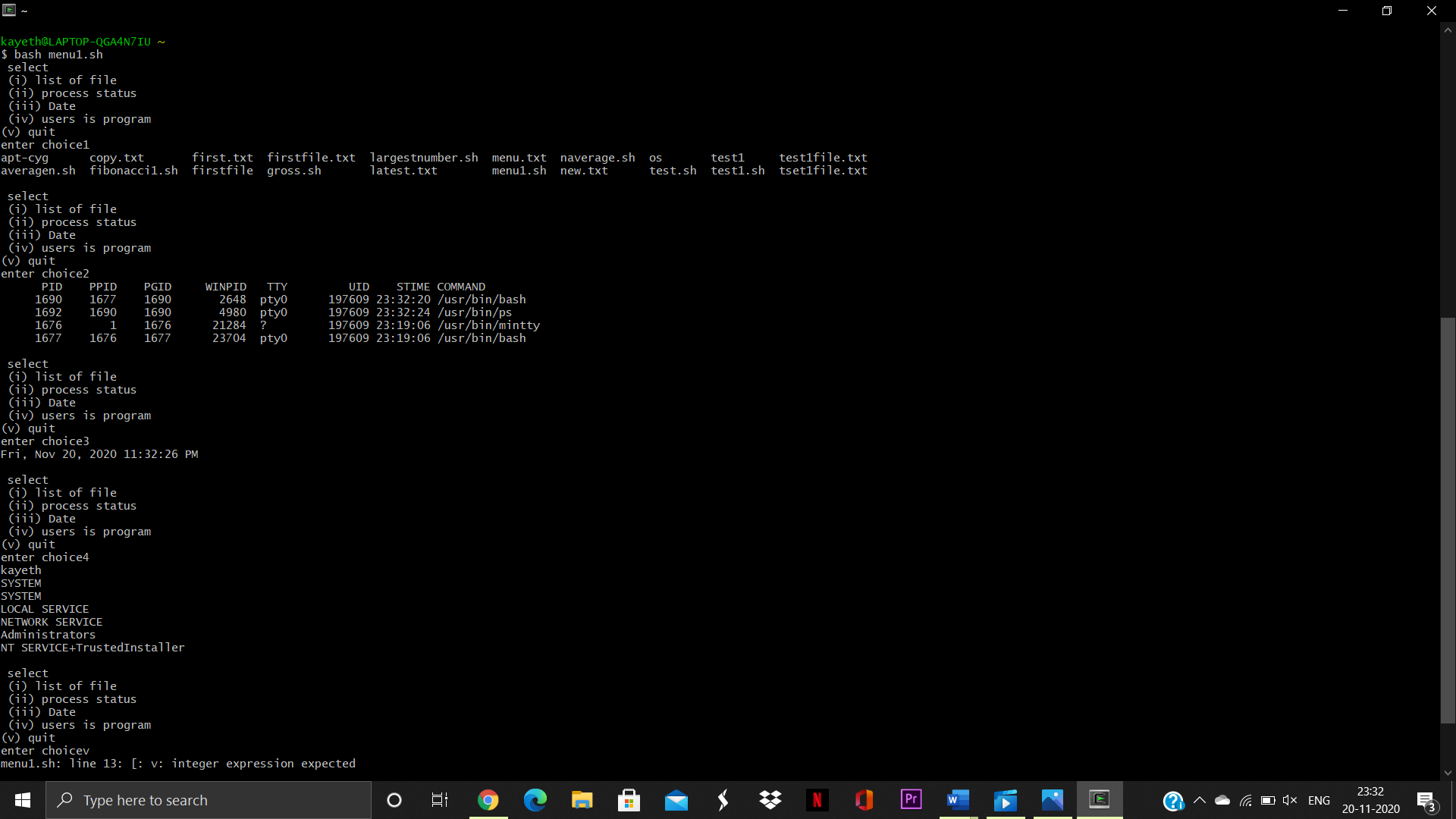
Write a shell program to display menu and execute instructions.

**Algorithm/ flowchart:**

**Code:**



**Output :**



**Experiment-6**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students with Linux Operating System Commands.

**Program Outcome**

* The students will be able to perform Linux commands .

**Problem Statement :**

Write a shell program to find Fibonacci series.

**Algorithm/ flowchart:**

* Start
* Declare variables i, a,b , show
* Initialize the variables, a=0, b=1, and show =0
* Enter the number of terms of Fibonacci series to be printed
* Print First two terms of series
* Use loop for the following steps  
  ->show=a+b  
  ->a=b  
  ->b=show  
  ->increase value of i each time by 1  
  -> print the value of show
* End

**Code:**

# Static input fo N

N=6

# First Number of the

# Fibonacci Series

a=0

# Second Number of the

# Fibonacci Series

b=1

echo "The Fibonacci series is : "

for (( i=0; i<N; i++ ))

do

    echo -n "$a "

    fn=$((a + b))

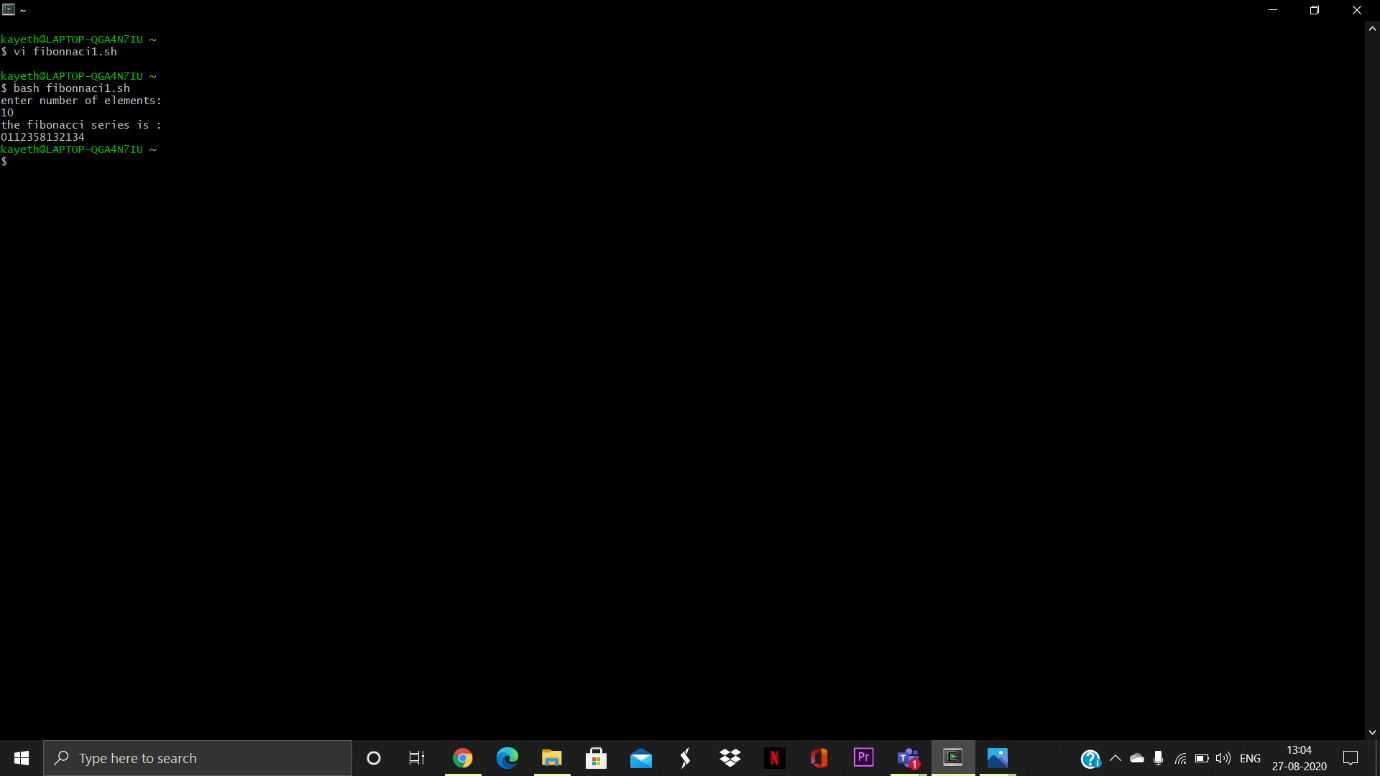
    a=$b

    b=$fn

done

# End of for loop

**Output :**



**Experiment-7**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To familiarize the students with Linux Operating System Commands.

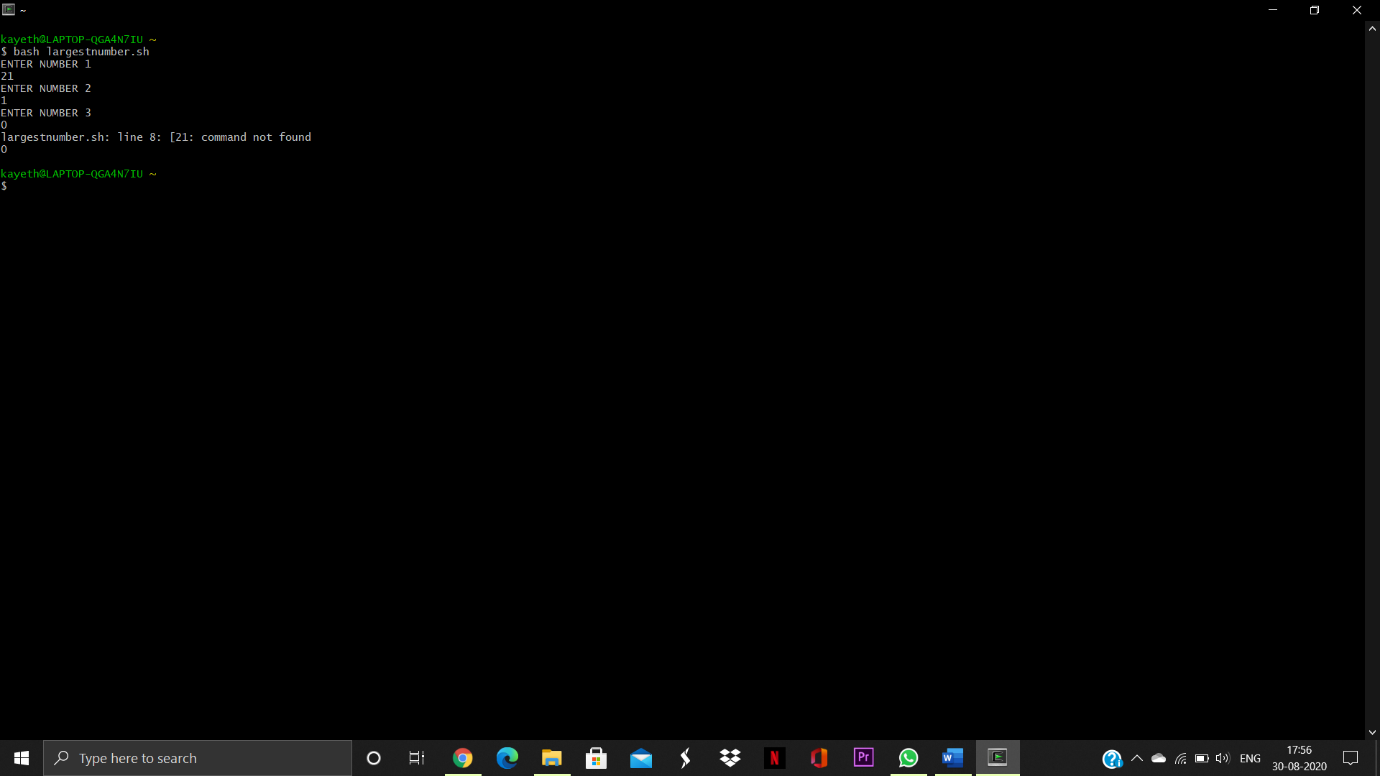
**Program Outcome**

* The students will be able to perform Linux commands .

**Problem Statement :**

Write a shell program to find Largest of 3 numbers.

**Code:** 

**Output :** 

**Experiment-8**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

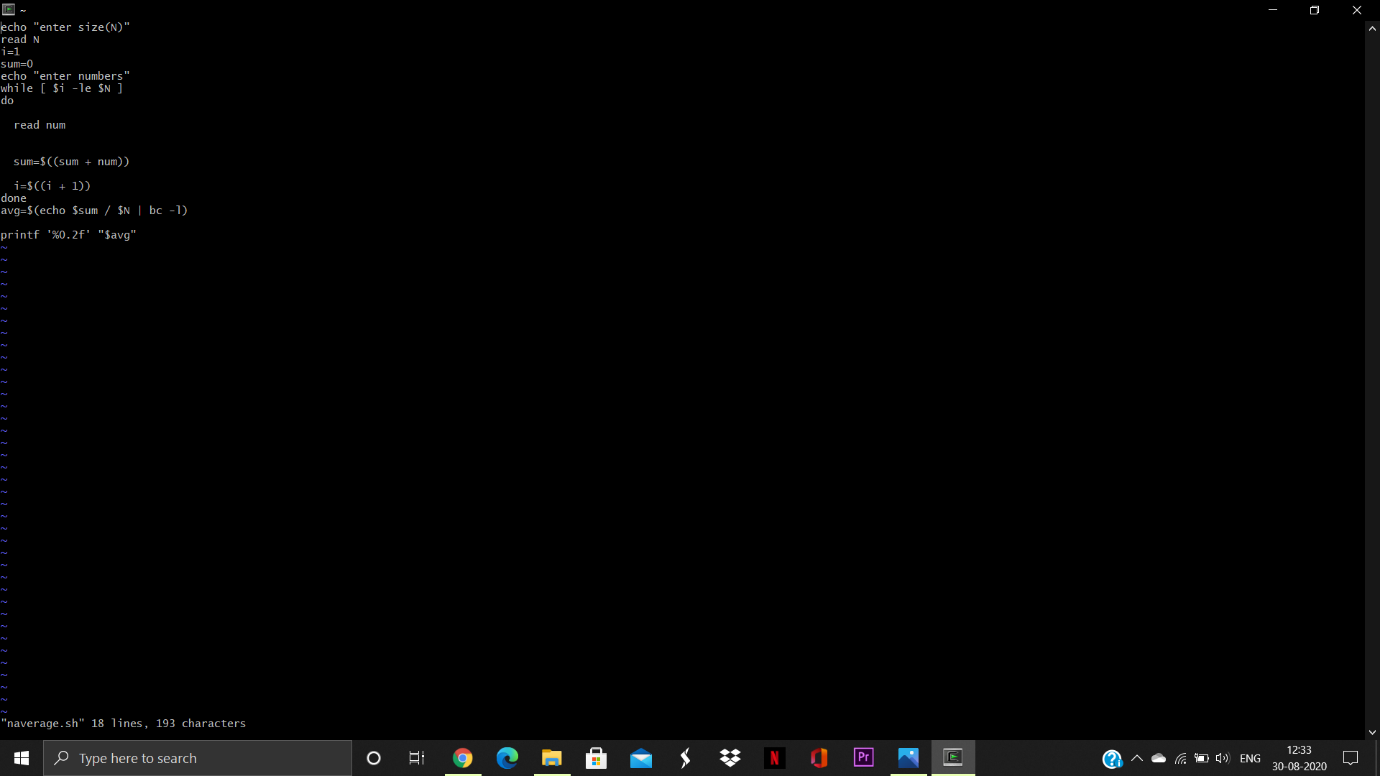
To familiarize the students with Linux Operating System Commands.

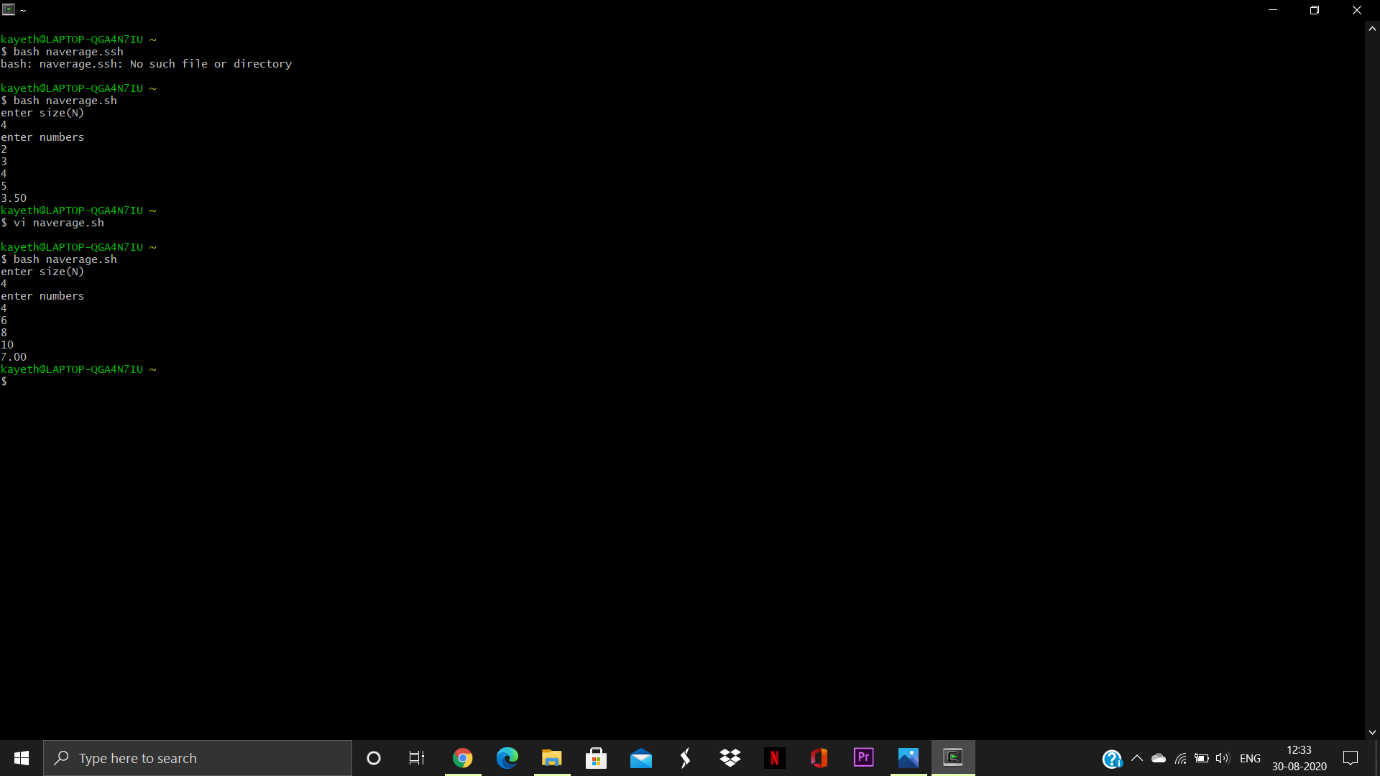
**Program Outcome**

* The students will be able to perform Linux commands .

**Problem Statement :**

Write a shell program to find average of n numbers .

**Code :** 

**Output :** 

**Experiment-9**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To learn about cpu scheduling algorithms

**Program Outcome**

* The students will be able to perform CPU Scheduling Algorithms

**Problem Statement :**

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time.

a) FCFS b) SJF c) Round Robin (pre-emptive) d) Priority **Code :**

1. To perform FCFS without Arrival Time

**Code(Python):**

def findWaitingTime(processes, n, bt, wt):

wt[0] = 0

for i in range(1, n ):

wt[i] = bt[i - 1] + wt[i - 1]

def findTurnAroundTime(processes, n, bt, wt, tat):

for i in range(n):

tat[i] = bt[i] + wt[i]

def findavgTime( processes, n, bt):

wt = [0] \* n

tat = [0] \* n

total\_wt = 0

total\_tat = 0

findWaitingTime(processes, n, bt, wt)

findTurnAroundTime(processes, n, bt, wt, tat)

# Display processes

print( "Processes Burst time " + " Waiting time " + " Turn around time")

# Calculate total waiting time

# and total turn around time

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" " + str(i + 1) + "\t\t" +

str(bt[i]) + "\t " +

str(wt[i]) + "\t\t " +

str(tat[i]))

print( "Average waiting time = "+

str(total\_wt / n))

print("Average turn around time = "+

str(total\_tat / n))

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

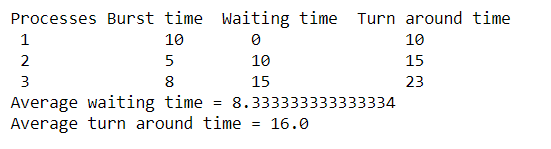
processes = [ 1, 2, 3]

n = len(processes)

burst\_time = [10, 5, 8]

findavgTime(processes, n, burst\_time)

**Output:**



To perform FCFS with Arrival Time

**Code(Python):**

def findWaitingTime(processes, n, bt, wt, at):

service\_time = [0] \* n

service\_time[0] = 0

wt[0] = 0

# calculating waiting time

for i in range(1, n):

service\_time[i] = (service\_time[i - 1] + bt[i - 1])

wt[i] = service\_time[i] - at[i]

if (wt[i] < 0):

wt[i] = 0

def findTurnAroundTime(processes, n, bt, wt, tat):

for i in range(n):

tat[i] = bt[i] + wt[i]

def findavgTime(processes, n, bt, at):

wt = [0] \* n

tat = [0] \* n

findWaitingTime(processes, n, bt, wt, at)

findTurnAroundTime(processes, n, bt, wt, tat)

# Display Processes

print("Processes Burst Time Arrival Time Waiting",

"Time Turn-Around Time Completion Time \n")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

compl\_time = tat[i] + at[i]

print(" ", i + 1, "\t\t", bt[i], "\t\t", at[i],

"\t\t", wt[i], "\t\t ", tat[i], "\t\t ", compl\_time)

print("Average waiting time = %.5f "%(total\_wt /n))

print("\nAverage turn around time = ", total\_tat / n)

# Driver Code

if \_\_name\_\_ =="\_\_main\_\_":

processes = [1, 2, 3]

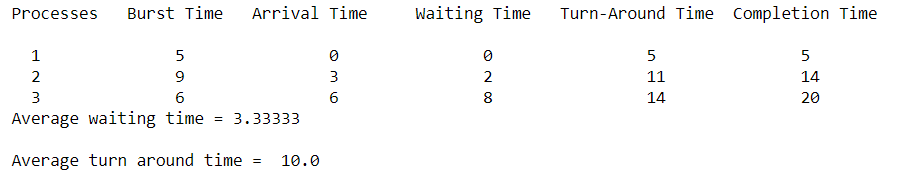
n = 3

burst\_time = [5, 9, 6]

arrival\_time = [0, 3, 6]

findavgTime(processes, n, burst\_time, arrival\_time)

**Output:**



**B)** To perform SJF-Non Preemptive

**Code (Python):**

bt=[] #bt stands for burst time

print("Enter the number of process: ")

n=int(input())

processes=[]

for i in range(0,n):

processes.insert(i,i+1)

print("Enter the burst time of the processes: \n")

bt=list(map(int, input().split()))

for i in range(0,len(bt)-1): #applying bubble sort to sort process according to their burst time

for j in range(0,len(bt)-i-1):

if(bt[j]>bt[j+1]):

temp=bt[j]

bt[j]=bt[j+1]

bt[j+1]=temp

temp=processes[j]

processes[j]=processes[j+1]

processes[j+1]=temp

wt=[] #wt stands for waiting time

avgwt=0 #average of waiting time

tat=[] #tat stands for turnaround time

avgtat=0 #average of total turnaround time

wt.insert(0,0)

tat.insert(0,bt[0])

for i in range(1,len(bt)):

wt.insert(i,wt[i-1]+bt[i-1])

tat.insert(i,wt[i]+bt[i])

avgwt+=wt[i]

avgtat+=tat[i]

avgwt=float(avgwt)/n

avgtat=float(avgtat)/n

print("\n")

print("Process\t Burst Time\t Waiting Time\t Turn Around Time")

for i in range(0,n):

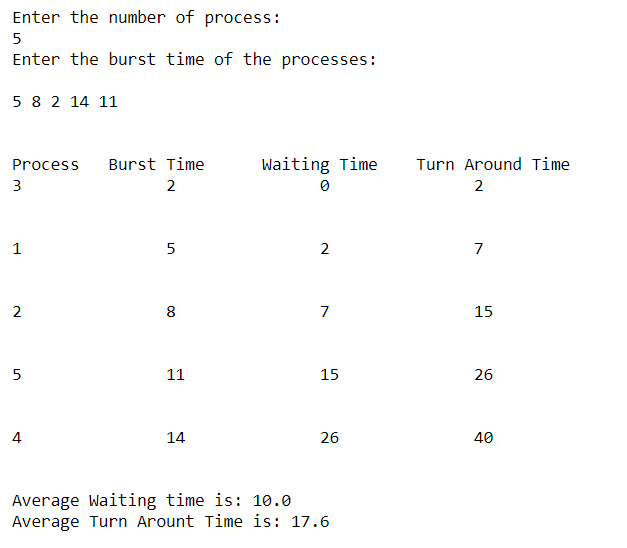
print(str(processes[i])+"\t\t"+str(bt[i])+"\t\t"+str(wt[i])+"\t\t"+str(tat[i]))

print("\n")

print("Average Waiting time is: "+str(avgwt))

print("Average Turn Arount Time is: "+str(avgtat))

**Output:**



To perform SJF- Preemptive

**Code (Python):**

def findWaitingTime(processes, n, wt):

rt = [0] \* n

# Copy the burst time into rt[]

for i in range(n):

rt[i] = processes[i][1]

complete = 0

t = 0

minm = 999999999

short = 0

check = False

while (complete != n):

for j in range(n):

if ((processes[j][2] <= t) and

(rt[j] < minm) and rt[j] > 0):

minm = rt[j]

short = j

check = True

if (check == False):

t += 1

continue

rt[short] -= 1

minm = rt[short]

if (minm == 0):

minm = 999999999

if (rt[short] == 0):

complete += 1

check = False

fint = t + 1

wt[short] = (fint - proc[short][1] - proc[short][2])

if (wt[short] < 0):

wt[short] = 0

t += 1

def findTurnAroundTime(processes, n, wt, tat):

for i in range(n):

tat[i] = processes[i][1] + wt[i]

def findavgTime(processes, n):

wt = [0] \* n

tat = [0] \* n

findWaitingTime(processes, n, wt)

findTurnAroundTime(processes, n, wt, tat)

# Display processes along with all details

print("Processes Burst Time Waiting Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", processes[i][0], "\t\t",

processes[i][1], "\t\t",

wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f "%(total\_wt /n) )

print("Average turn around time = ", total\_tat / n)

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

# Process id's

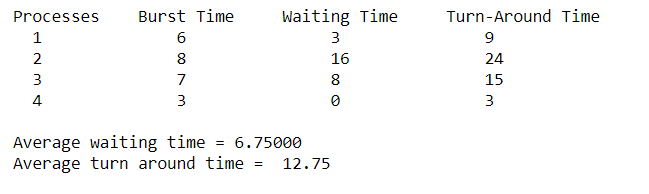
proc = [[1, 6, 1], [2, 8, 1],

[3, 7, 2], [4, 3, 3]]

n = 4

findavgTime(proc, n)

**Output:**



1. To perform Round Robin

**Code (Python):**

def findWaitingTime(processes, n, bt, wt, quantum):

rem\_bt = [0] \* n

for i in range(n):

rem\_bt[i] = bt[i]

t = 0

while(1):

done = True

for i in range(n):

if (rem\_bt[i] > 0) :

done = False

if (rem\_bt[i] > quantum) :

t += quantum

rem\_bt[i] -= quantum

else:

t = t + rem\_bt[i]

wt[i] = t - bt[i]

rem\_bt[i] = 0

if (done == True):

break

def findTurnAroundTime(processes, n, bt, wt, tat):

for i in range(n):

tat[i] = bt[i] + wt[i]

def findavgTime(processes, n, bt, quantum):

wt = [0] \* n

tat = [0] \* n

findWaitingTime(processes, n, bt, wt, quantum)

findTurnAroundTime(processes, n, bt, wt, tat)

print("Processes Burst Time Waiting Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", i + 1, "\t\t", bt[i], "\t\t", wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f "%(total\_wt /n) )

print("Average turn around time = %.5f "% (total\_tat / n))

if \_\_name\_\_ =="\_\_main\_\_":

# Process id's

proc = [1, 2, 3]

n = 3

# Burst time of all processes

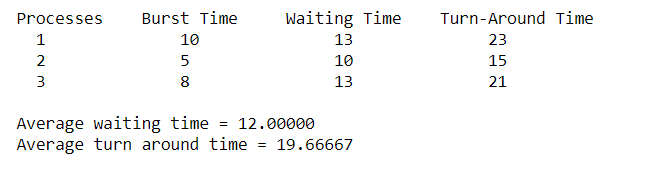
burst\_time = [10, 5, 8]

# Time quantum

quantum = 2;

findavgTime(proc, n, burst\_time, quantum)

**Output:**



D) To perform Priority Scheduling (with Arrival Time).

**Code (Python):**

totalprocess = 5

proc = []

for i in range(5):

l = []

for j in range(4):

l.append(0)

proc.append(l)

def get\_wt\_time( wt):

service = [0] \* 5

service[0] = 0

wt[0] = 0

for i in range(1, totalprocess):

service[i] = proc[i - 1][1] + service[i - 1]

wt[i] = service[i] - proc[i][0] + 1

if(wt[i] < 0) :

wt[i] = 0

def get\_tat\_time(tat, wt):

for i in range(totalprocess):

tat[i] = proc[i][1] + wt[i]

def findgc():

wt = [0] \* 5

tat = [0] \* 5

wavg = 0

tavg = 0

get\_wt\_time(wt)

get\_tat\_time(tat, wt)

stime = [0] \* 5

ctime = [0] \* 5

stime[0] = 1

ctime[0] = stime[0] + tat[0]

for i in range(1, totalprocess):

stime[i] = ctime[i - 1]

ctime[i] = stime[i] + tat[i] - wt[i]

print("Process\_no\tStart\_time\tComplete\_time",

"\tTurn\_Around\_Time\tWaiting\_Time")

# display the process details

for i in range(totalprocess):

wavg += wt[i]

tavg += tat[i]

print(proc[i][3], "\t\t", stime[i], "\t\t", end = " ")

print(ctime[i], "\t\t", tat[i], "\t\t\t", wt[i])

print("Average waiting time is : ", end = " ")

print(wavg / totalprocess)

print("average turnaround time : " , end = " ")

print(tavg / totalprocess)

if \_\_name\_\_ =="\_\_main\_\_":

arrivaltime = [1, 2, 3, 4, 5]

bursttime = [3, 5, 1, 7, 4]

priority = [3, 4, 1, 7, 8]

for i in range(totalprocess):

proc[i][0] = arrivaltime[i]

proc[i][1] = bursttime[i]

proc[i][2] = priority[i]

proc[i][3] = i + 1

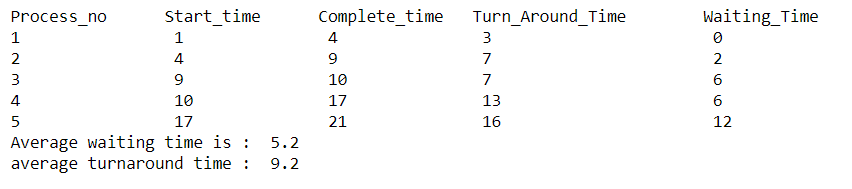
# Using inbuilt sort function

proc = sorted (proc, key = lambda x:x[2])

proc = sorted (proc)

findgc()

**Output:**



To perform Priority Scheduling (without Arrival Time).

**Code (Python):**

def findWaitingTime(processes, n, wt):

wt[0] = 0

for i in range(1, n):

wt[i] = processes[i - 1][1] + wt[i - 1]

def findTurnAroundTime(processes, n, wt, tat):

for i in range(n):

tat[i] = processes[i][1] + wt[i]

def findavgTime(processes, n):

wt = [0] \* n

tat = [0] \* n

findWaitingTime(processes, n, wt)

findTurnAroundTime(processes, n, wt, tat)

print("\nProcesses Burst Time Waiting Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", processes[i][0], "\t\t",

processes[i][1], "\t\t",

wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f "%(total\_wt /n))

print("Average turn around time = ", total\_tat / n)

def priorityScheduling(proc, n):

proc = sorted(proc, key = lambda proc:proc[2], reverse = True);

print("Order in which processes gets executed")

for i in proc:

print(i[0], end = " ")

findavgTime(proc, n)

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

# Process id's

proc = [[1, 10, 1],

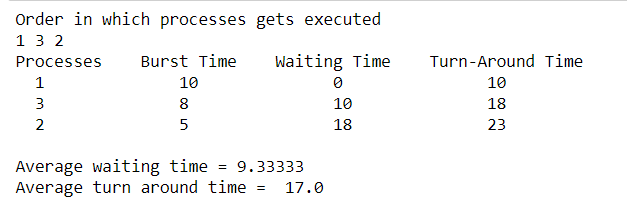
[2, 5, 0],

[3, 8, 1]]

n = 3

priorityScheduling(proc, n)

**Output:**



**Experiment-10**

Student Name and Roll Number:

Semester /Section:

Link to Code:

Date:

Faculty Signature:

Remarks:

**Objective**

To learn about Deadlock Management Technique

**Program Outcome**

* The students will be able to perform Bankers algorithm

**Problem Statement :**

Write a program to simulate Bankers algorithm for the purpose of deadlock avoidance

**Code:**

def main():

processes = int(input("number of processes : "))

resources = int(input("number of resources : "))

max\_resources = [int(i) for i in input("maximum resources : ").split()]

print("\n-- allocated resources for each process --")

currently\_allocated = [[int(i) for i in input(f"process {j + 1} : ").split()] for j in range(processes)]

print("\n-- maximum resources for each process --")

max\_need = [[int(i) for i in input(f"process {j + 1} : ").split()] for j in range(processes)]

allocated = [0] \* resources

for i in range(processes):

for j in range(resources):

allocated[j] += currently\_allocated[i][j]

print(f"\ntotal allocated resources : {allocated}")

available = [max\_resources[i] - allocated[i] for i in range(resources)]

print(f"total available resources : {available}\n")

running = [True] \* processes

count = processes

while count != 0:

safe = False

for i in range(processes):

if running[i]:

executing = True

for j in range(resources):

if max\_need[i][j] - currently\_allocated[i][j] > available[j]:

executing = False

break

if executing:

print(f"process {i + 1} is executing")

running[i] = False

count -= 1

safe = True

for j in range(resources):

available[j] += currently\_allocated[i][j]

break

if not safe:

print("the processes are in an unsafe state.")

break

print(f"the process is in a safe state.\navailable resources : {available}\n")

if \_\_name\_\_ == '\_\_main\_\_':

main()

**OUTPUT :**

