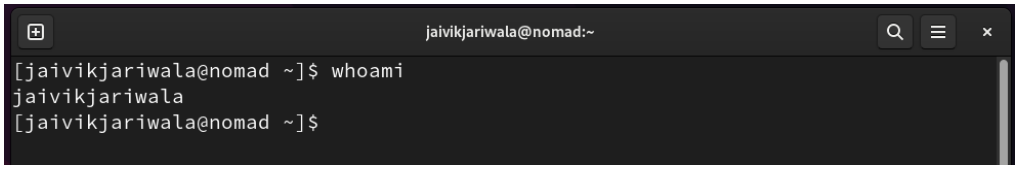
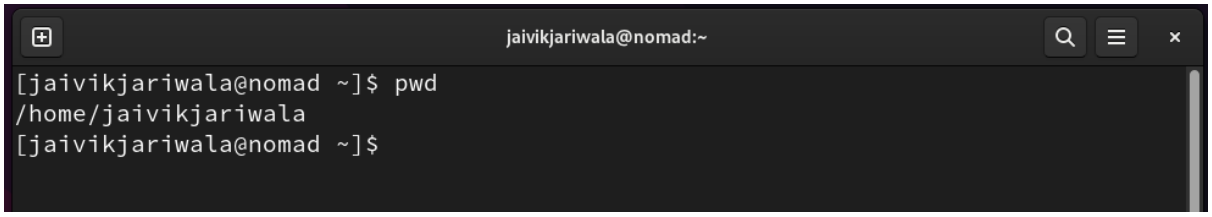
Lab – 1 Jaivik Jariwala 21BCP004

Perform the following Linux Fedodra command operations

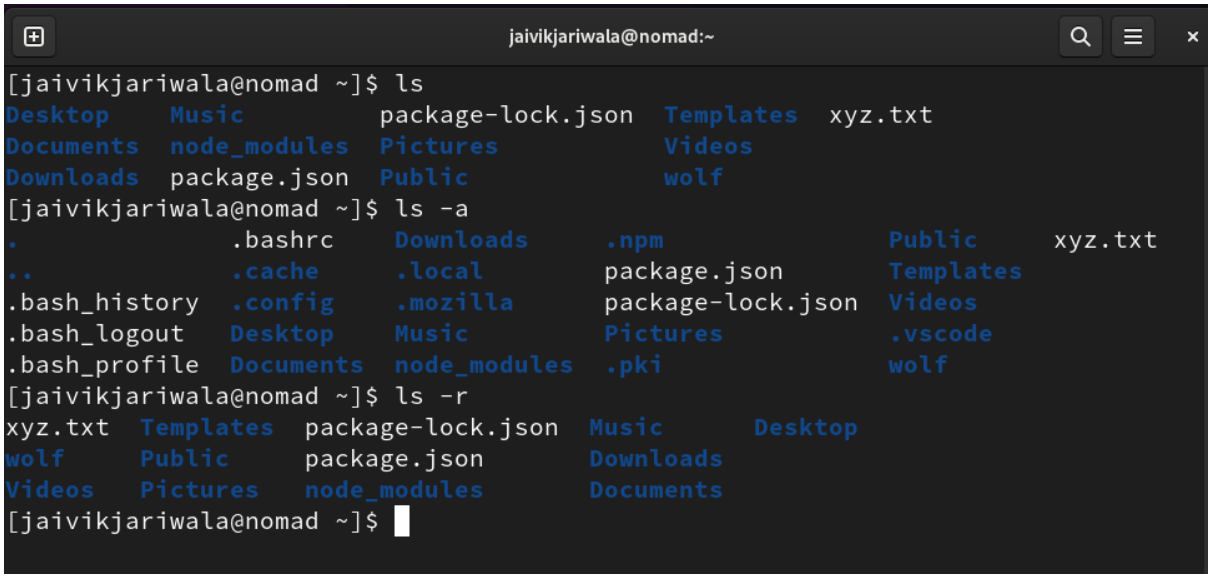
1. whoami



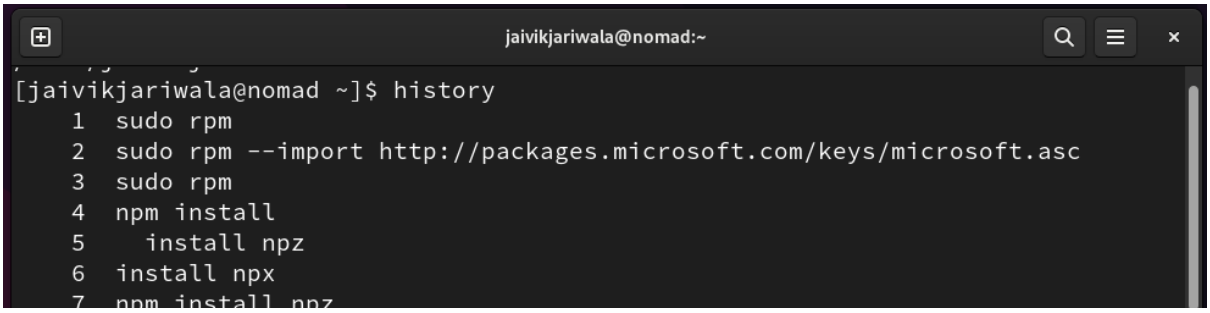
2. pwd



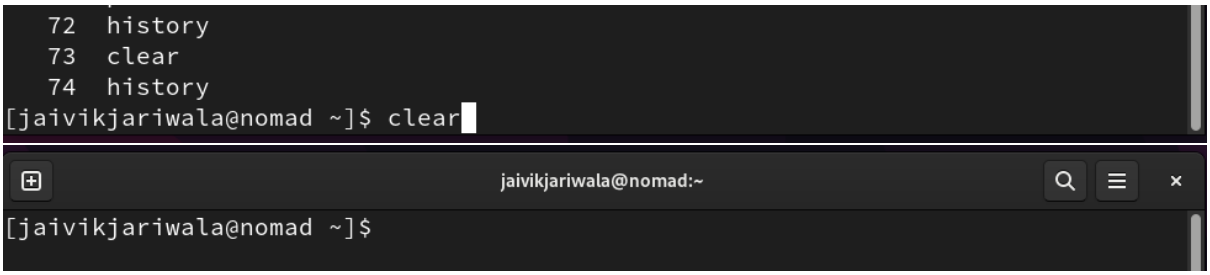
3. is
4. is -r
5. is -a



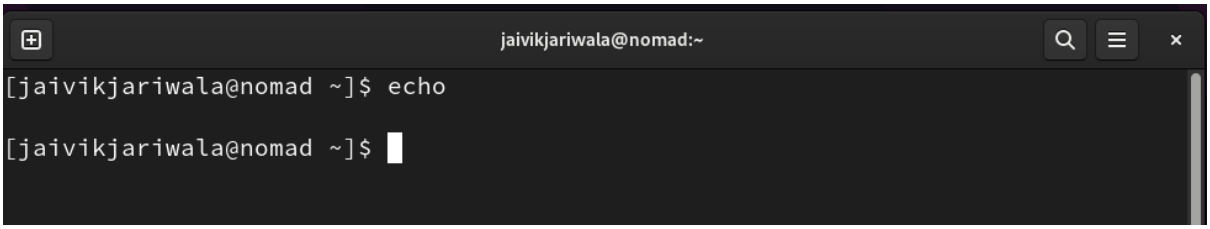
6. history



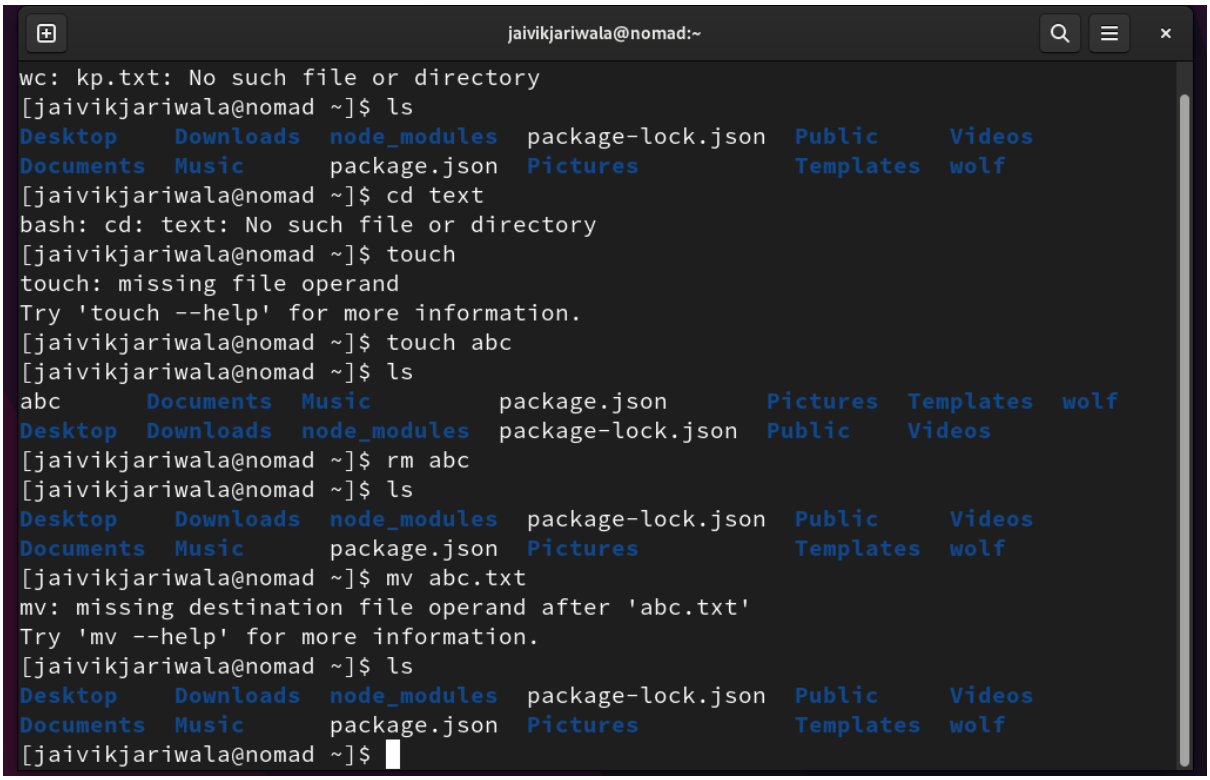
7. clear



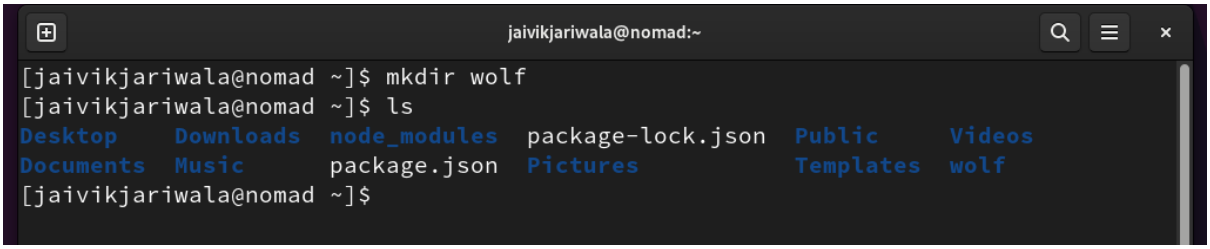
8. echo



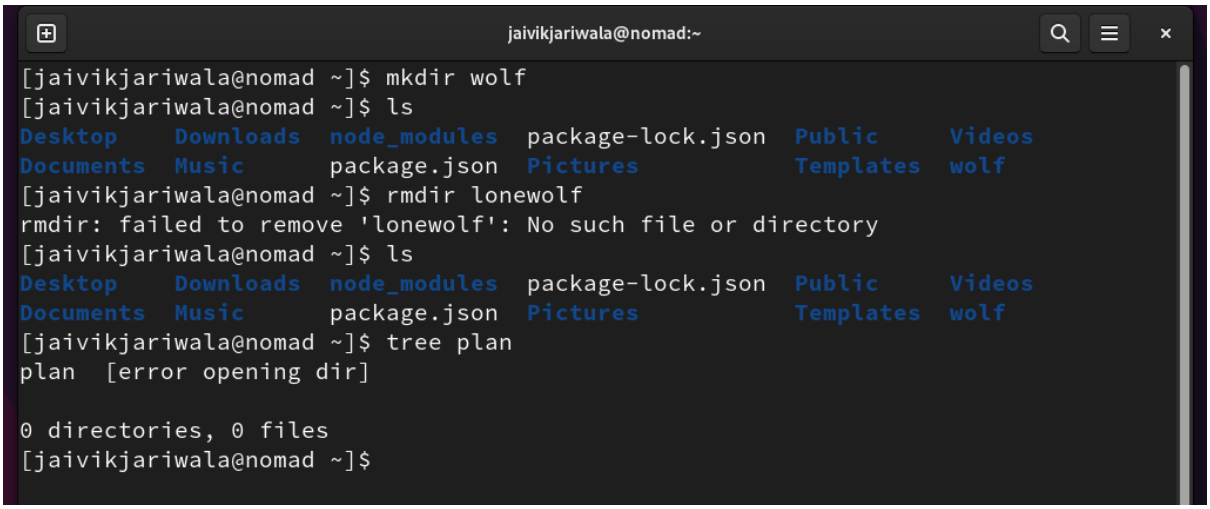
9. touch



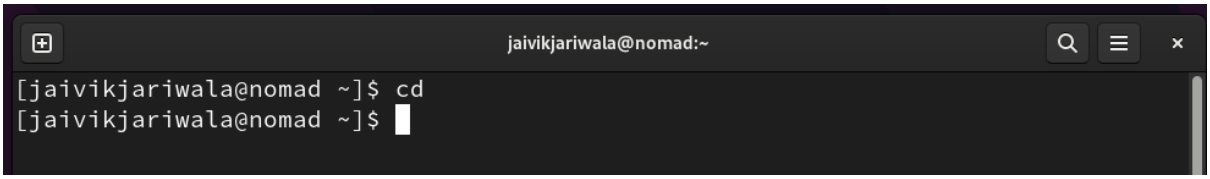
10. rm
11. mkdir



12. rmdir
13. mv



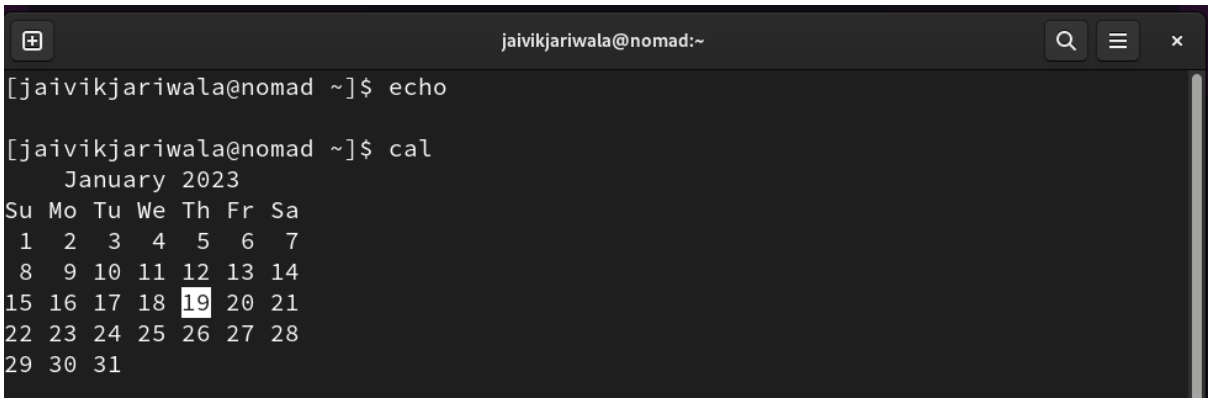
14. cd



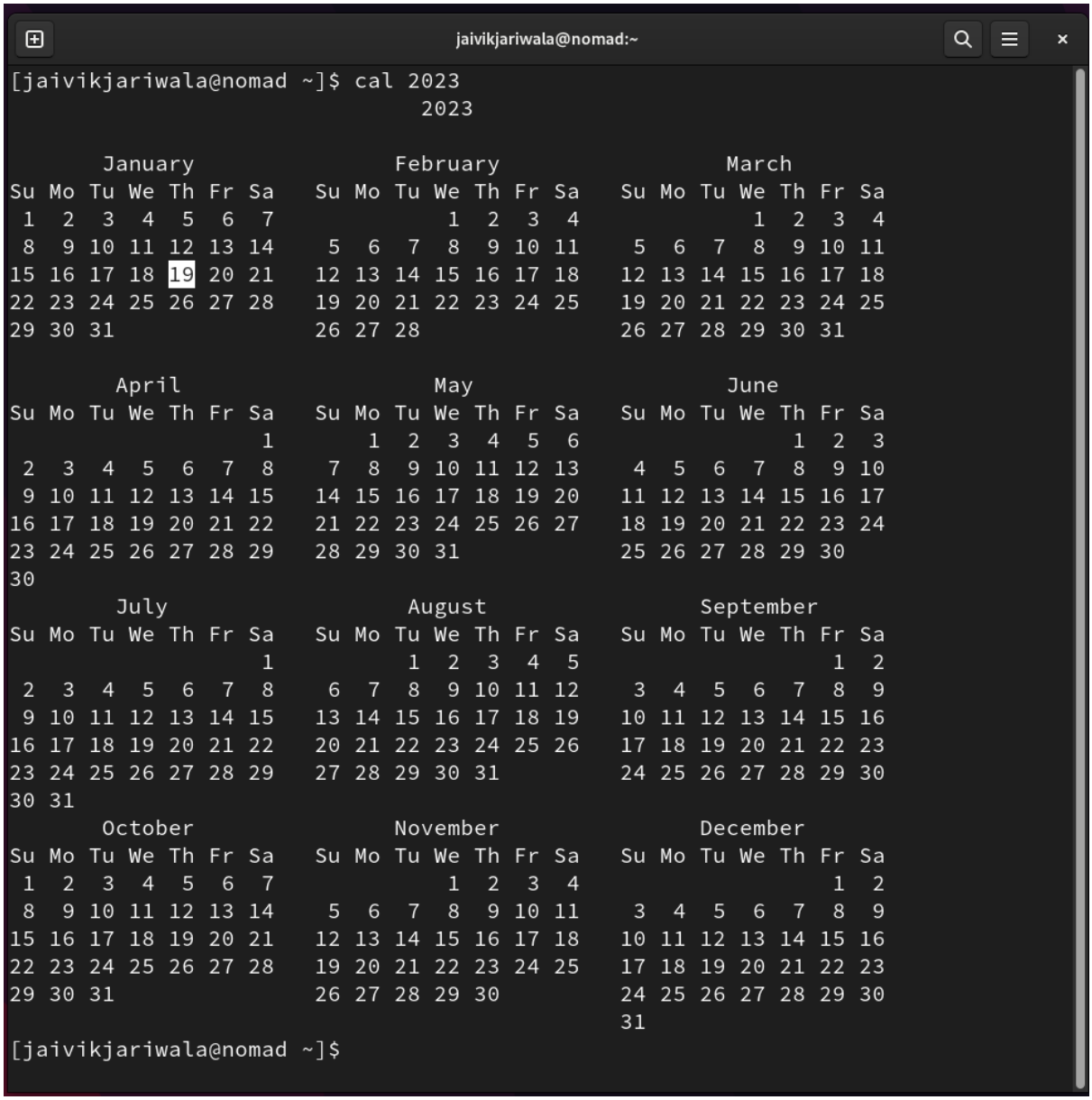
15. cmp

16. cat

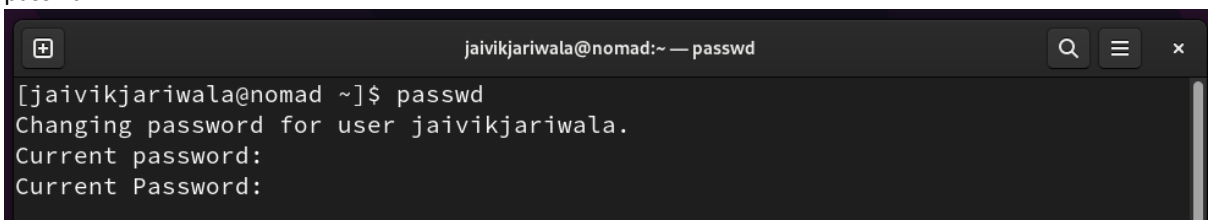
17. cal



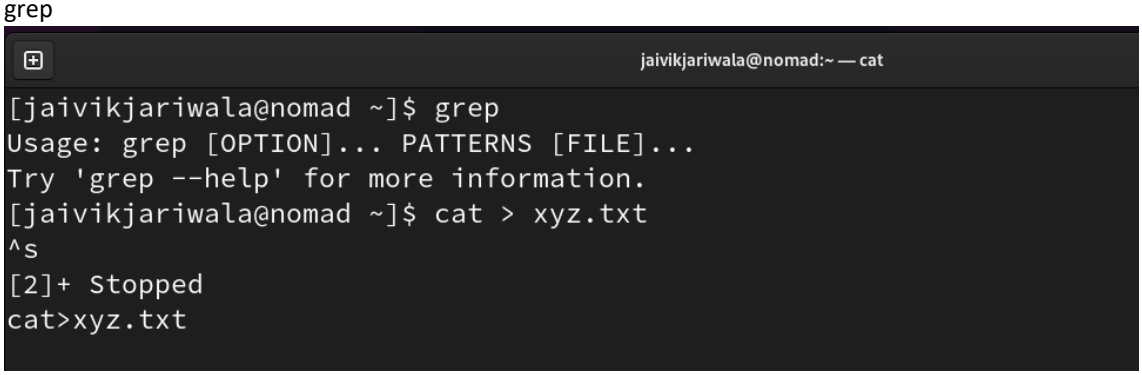
18. cal -ycal 2018



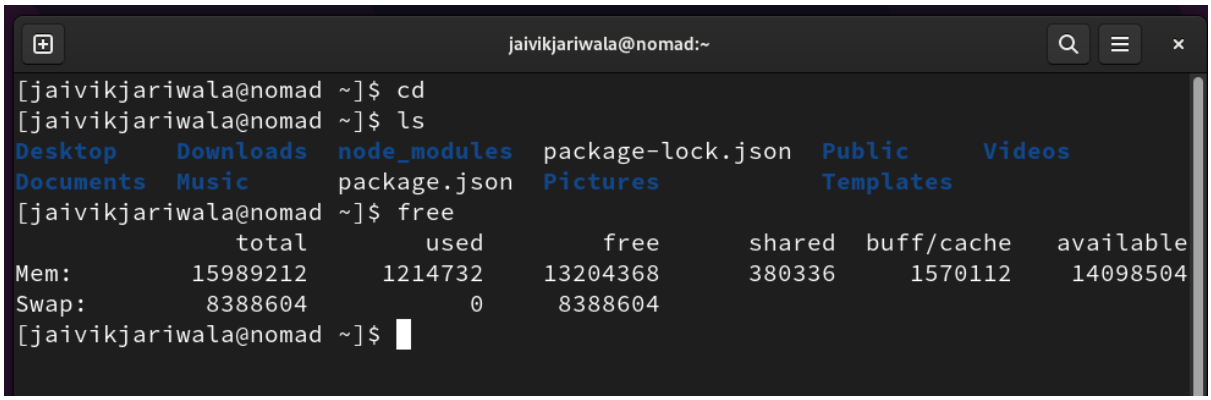
19. passwd



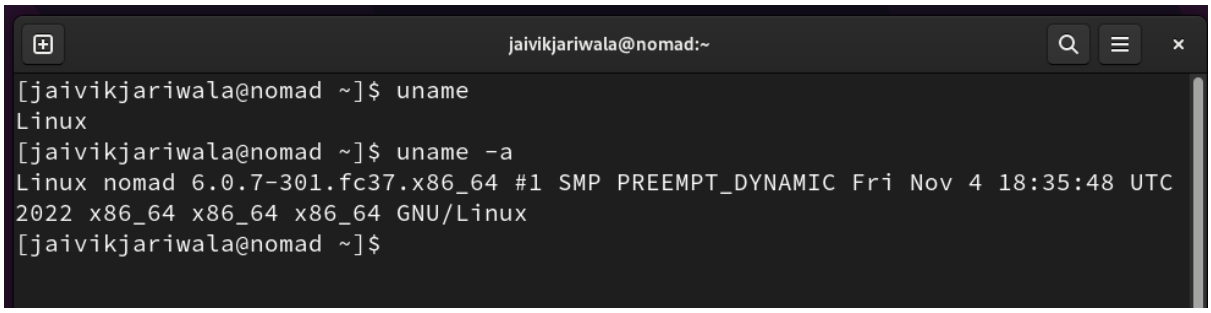
20. grep



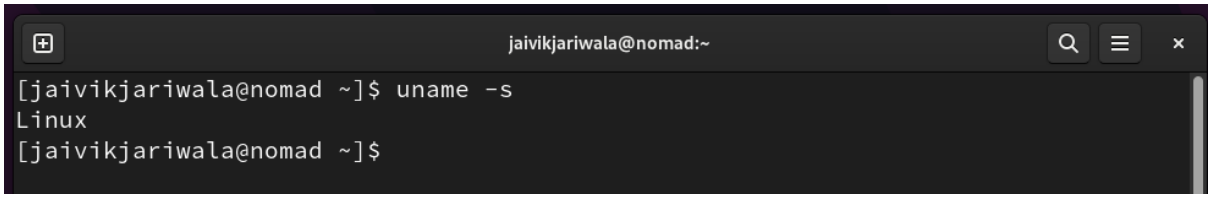
21. free



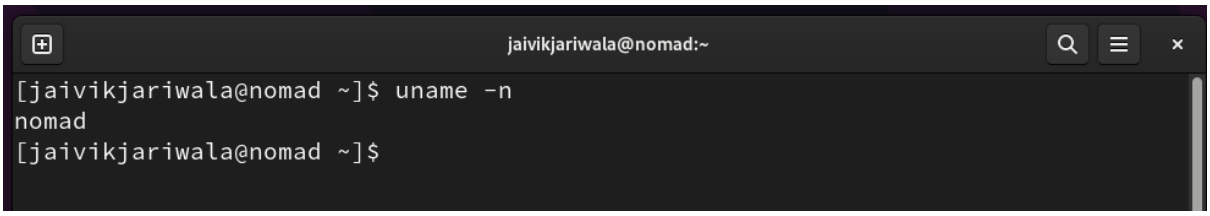
22. uname



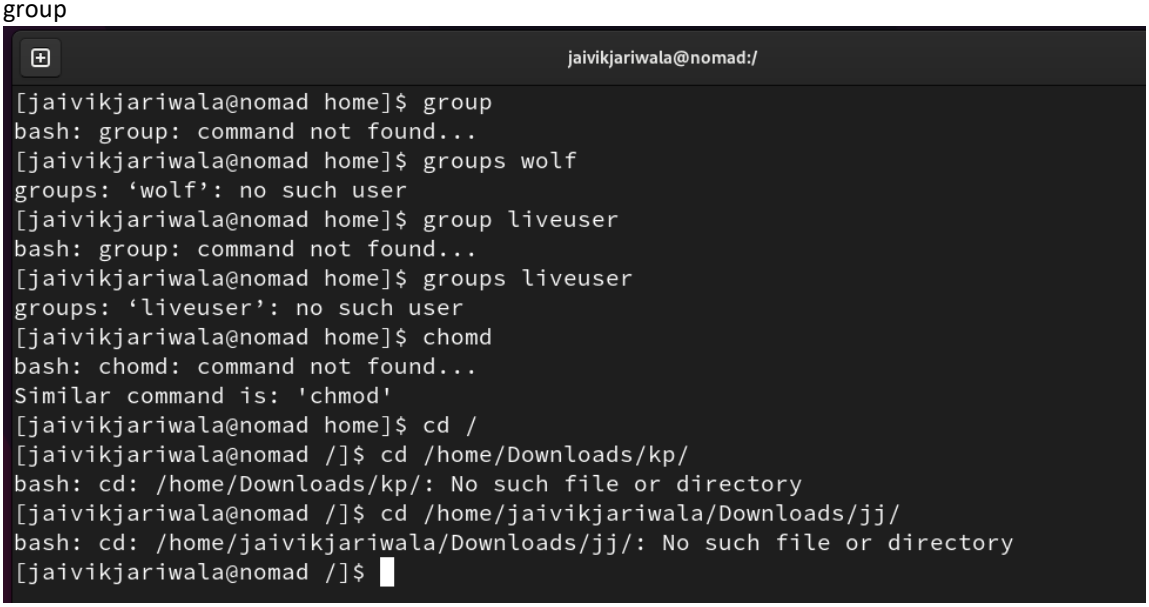
23. uname -a
24. uname -s



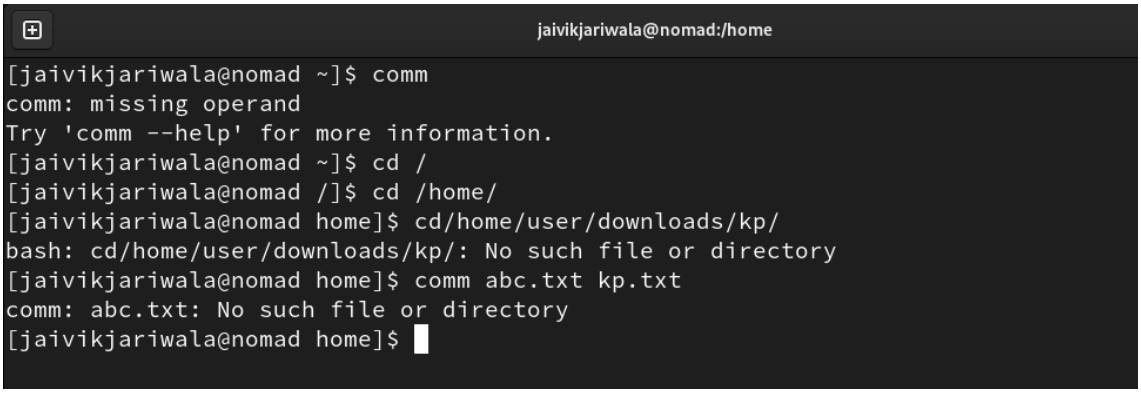
25. uname -n



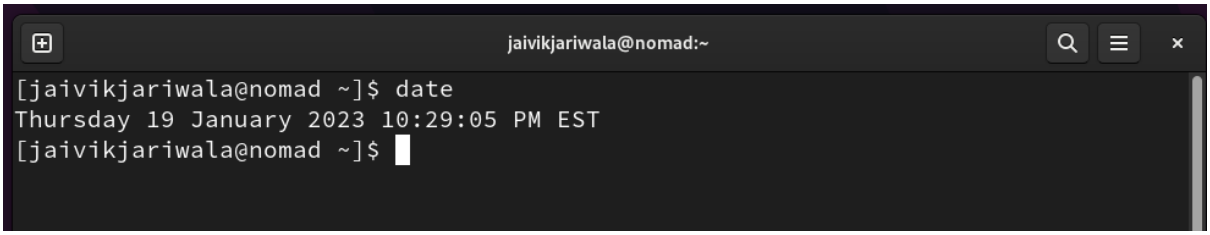
26. group



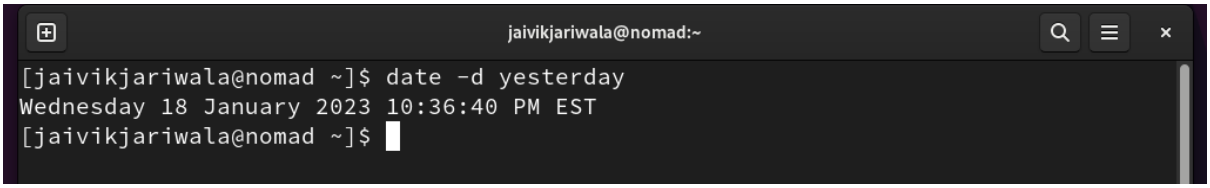
27. comm



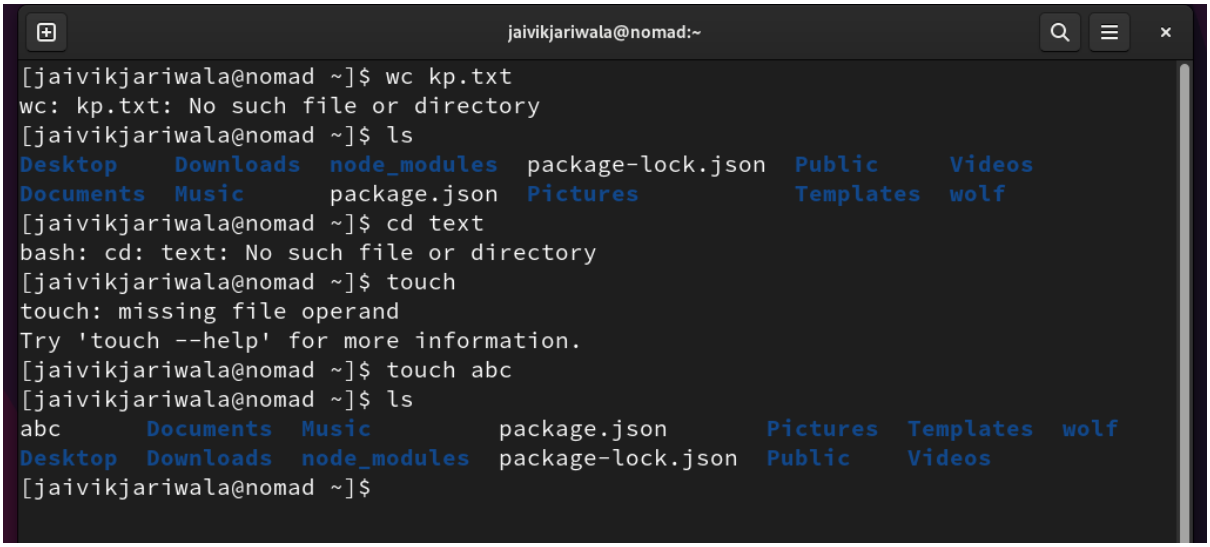
28. date



29. date -d



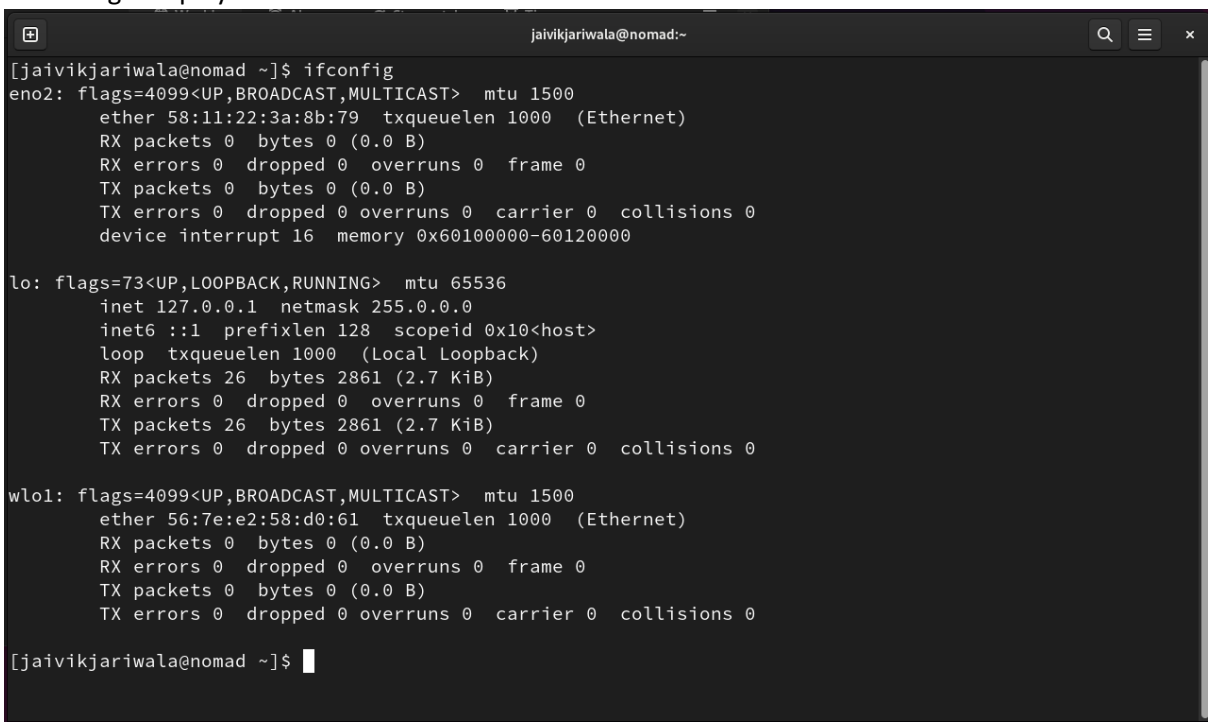
30. chmod
31. wc


jaivikjariwala@nomad:~
[jaivikjariwala@nomad ~]$ wc kp.txt
wc: kp.txt: No such file or directory
[jaivikjariwala@nomad ~]$ ls
Desktop Downloads node_modules package-lock.json Public Videos
Documents Music package.json Pictures Templates wolf
[jaivikjariwala@nomad ~]$ cd text
bash: cd: text: No such file or directory
[jaivikjariwala@nomad ~]$ touch
touch: missing file operand
Try 'touch --help' for more information.
[jaivikjariwala@nomad ~]$ touch abc
[jaivikjariwala@nomad ~]$ ls
abc Documents Music package.json Pictures Templates wolf
Desktop Downloads node_modules package-lock.json Public Videos
[jaivikjariwala@nomad ~]$

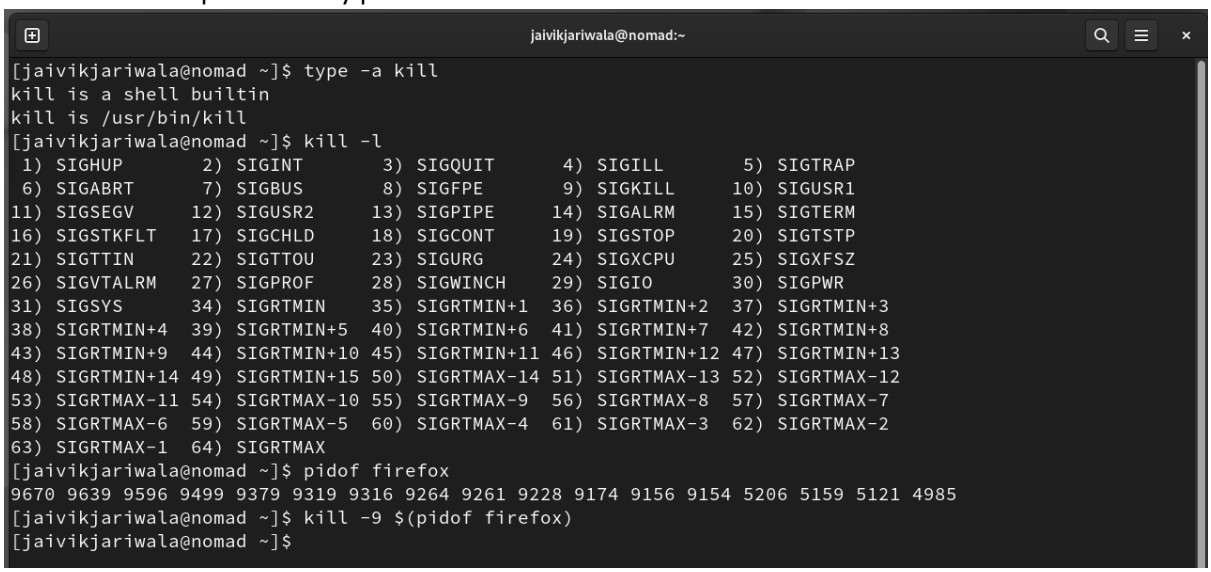
Lab – 2 Jaivik Jariwala 21BCP004

Perform the following Linux Fedodra command operations

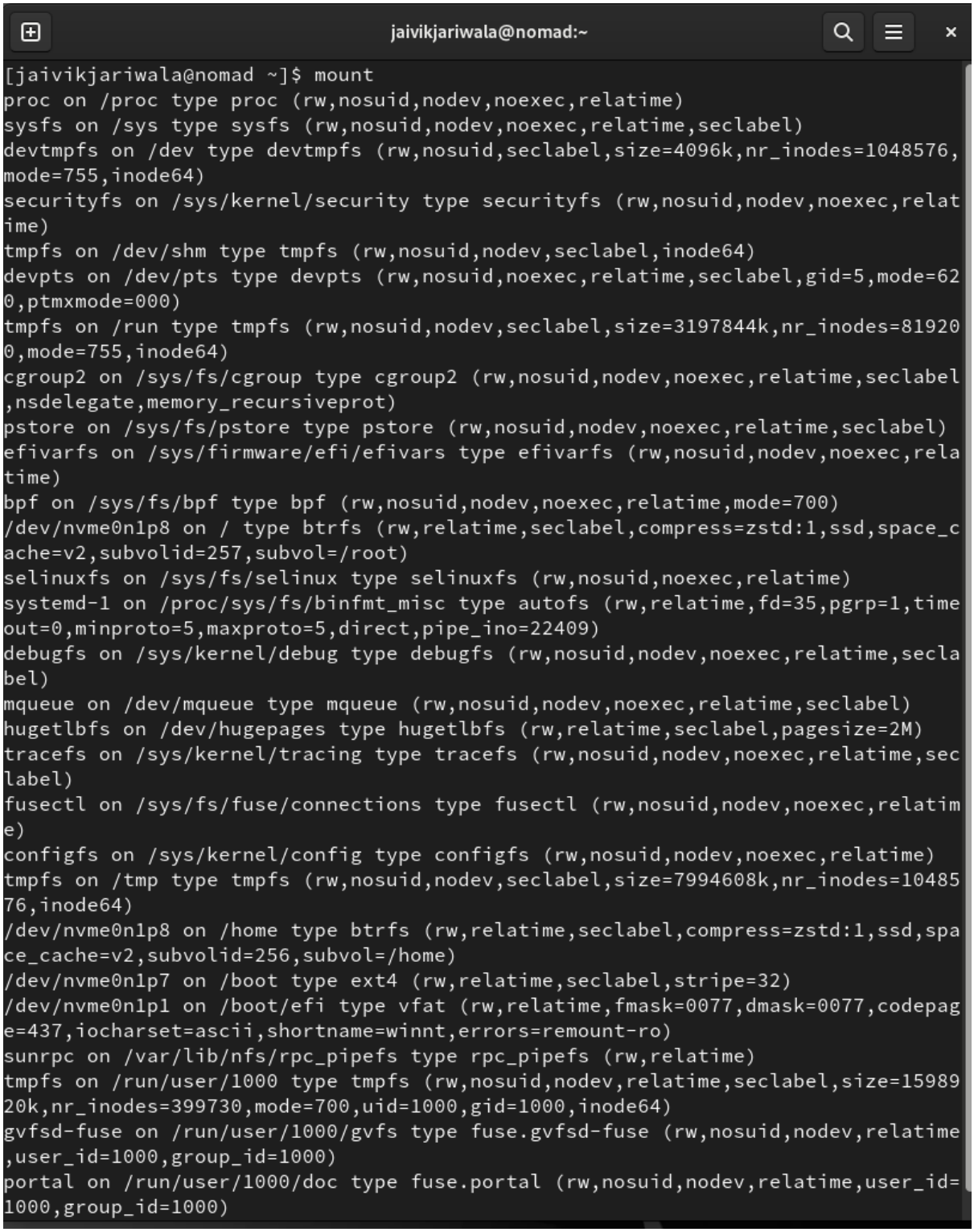
1. ifconfig : Display network interfaces and IP addresses.



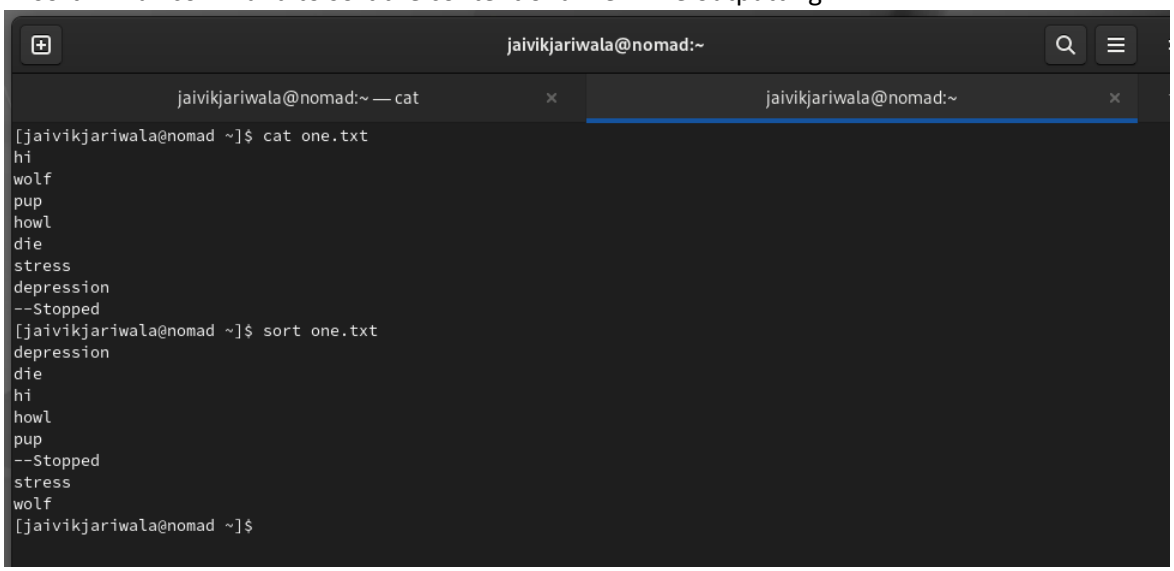
2. kill: Kill active processes by process ID or name



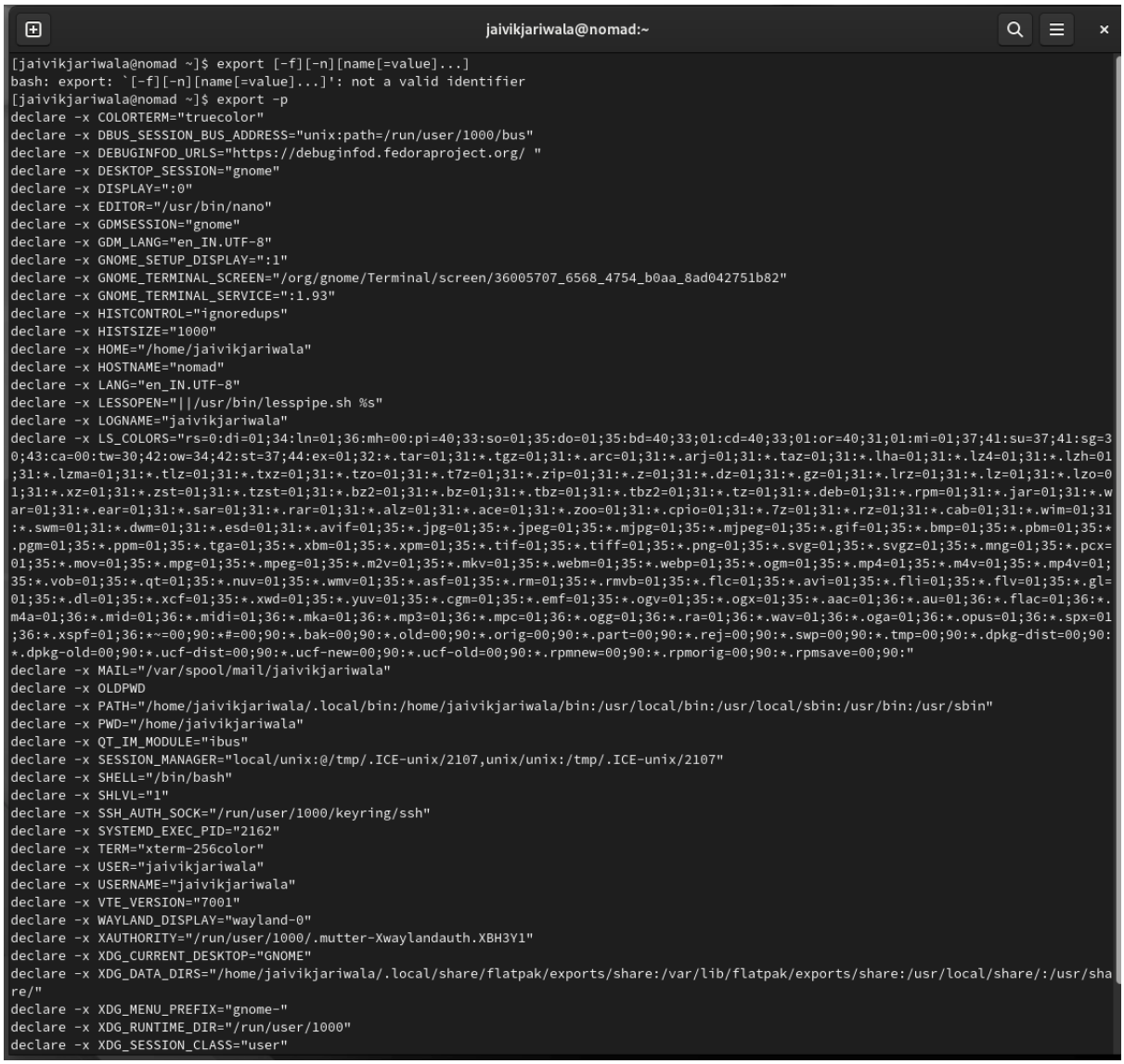
3. mount : Mount file systems in Linux



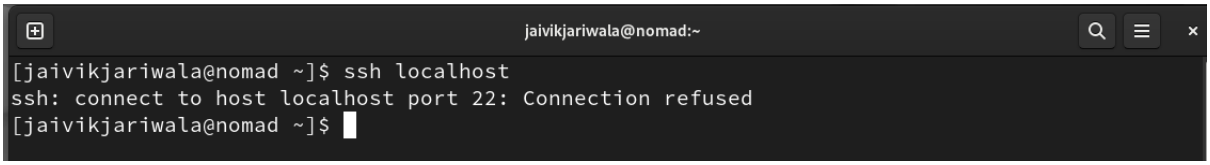
4. sort : Linux command to sort the content of a file while outputting



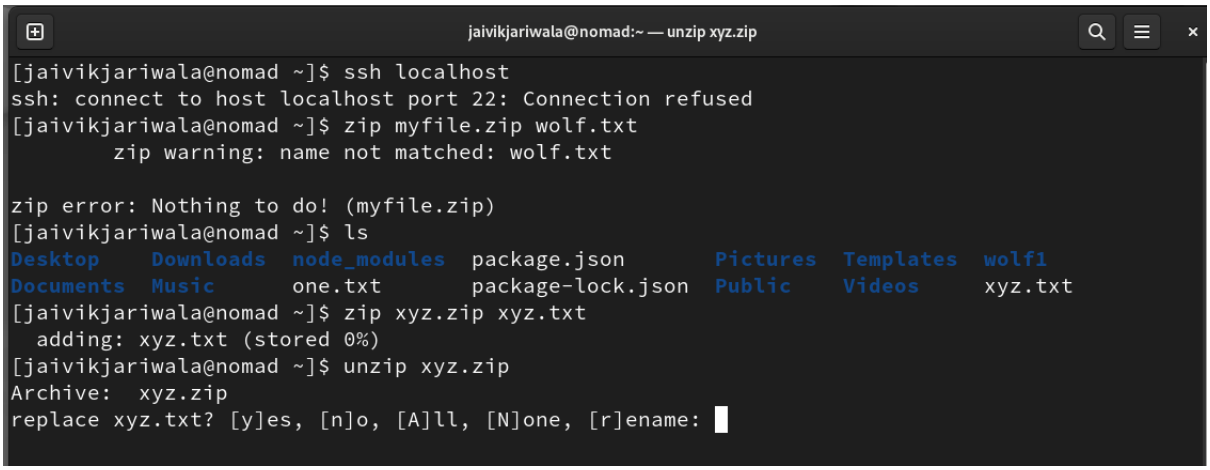
5. export : Export environment variables in Linux



6. ssh : Secure Shell command in Linux



7. zip : Zip files in Linux

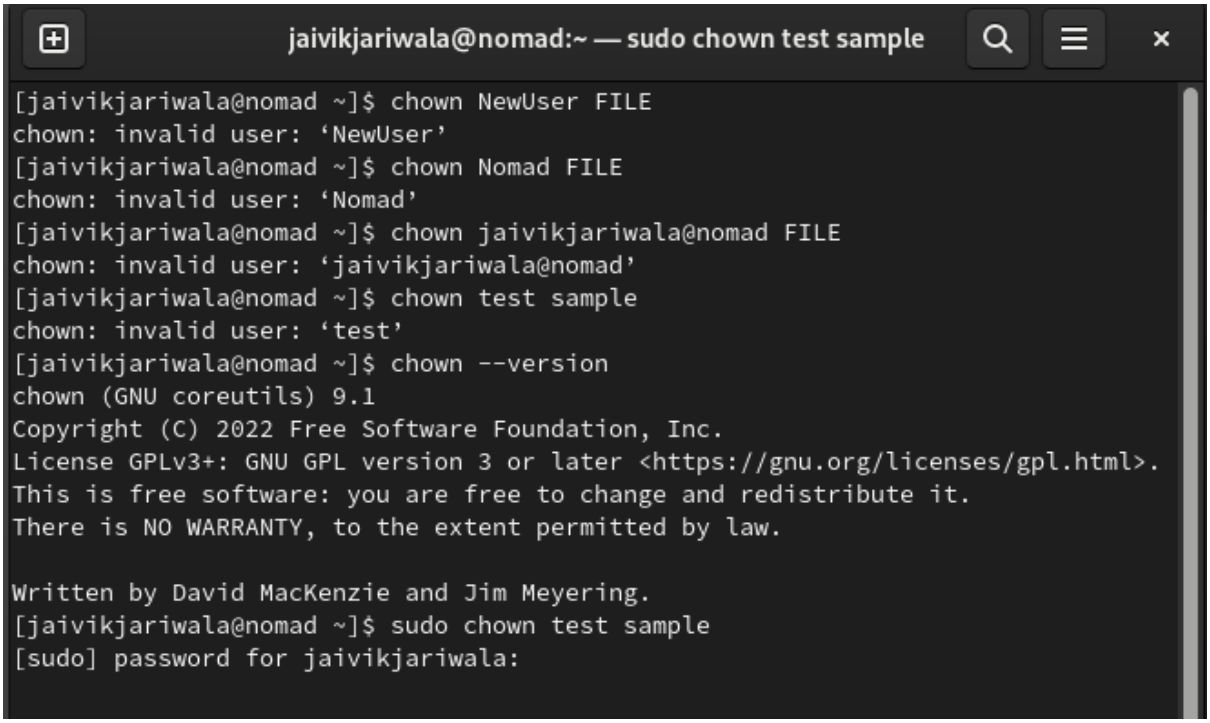


8. unzip : Unzip files in Linux

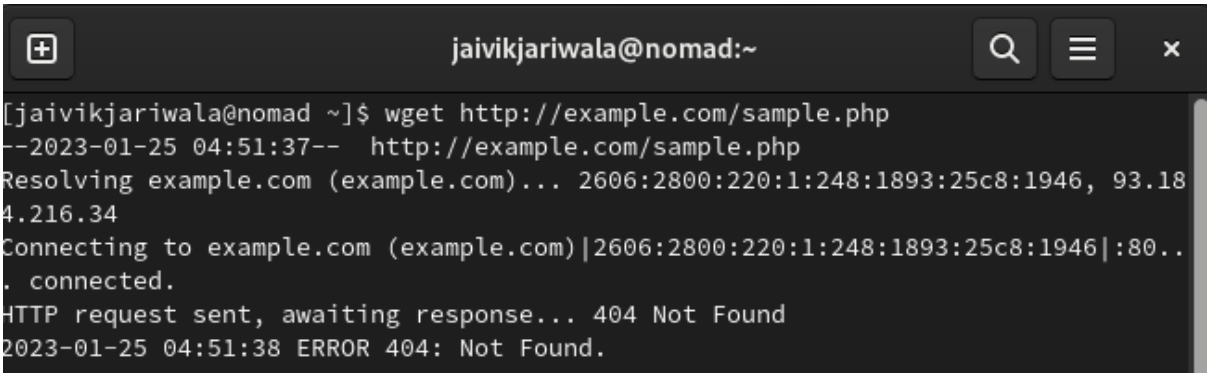
9. ps : Display active processes

10. uname : Linux command to get basic information about the OS

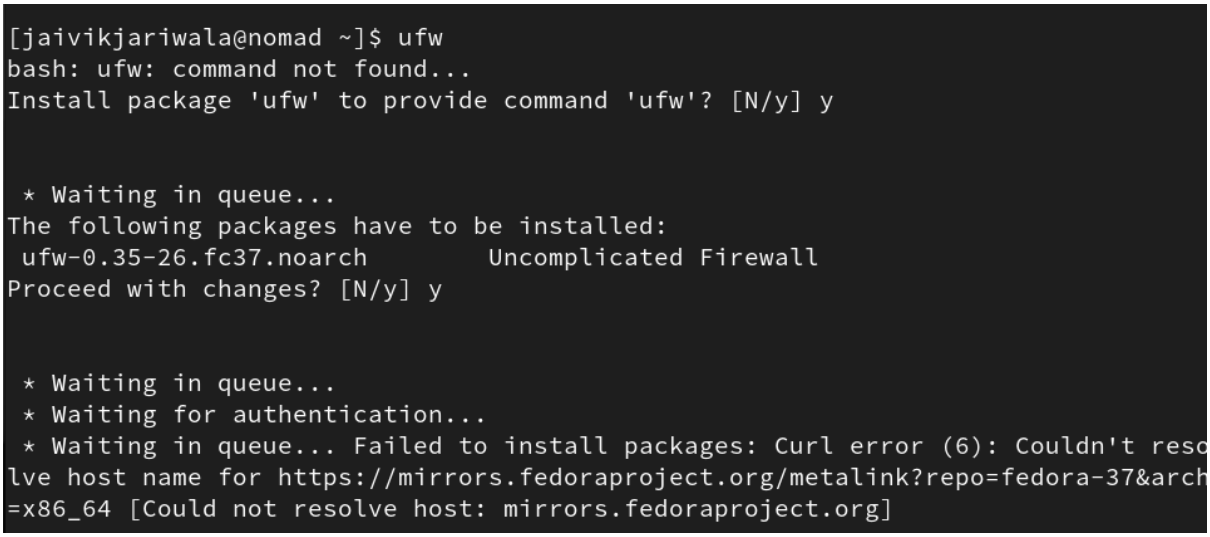
11. chown : Command for granting ownership of files or folders



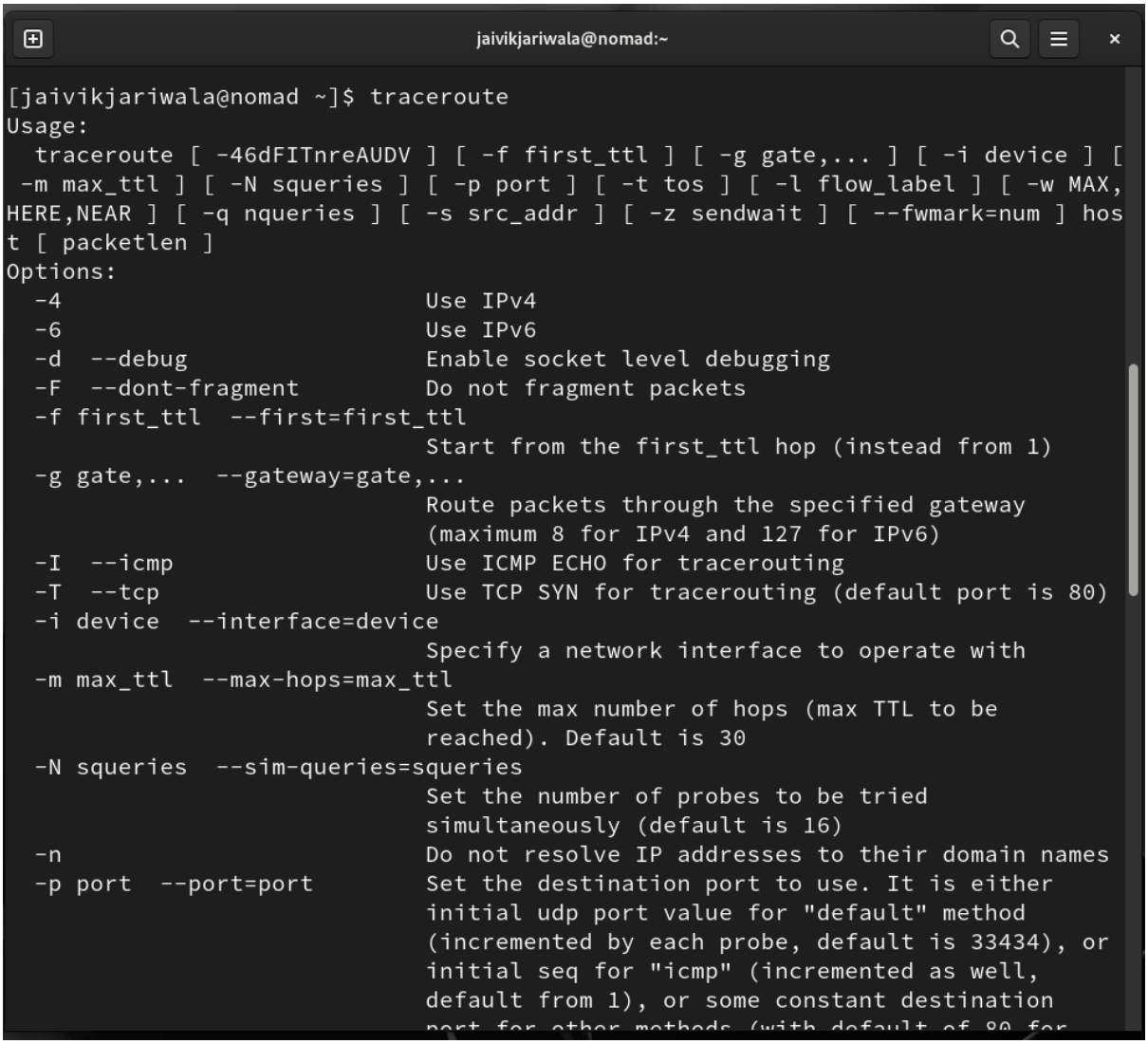
12. wget : Direct download files from the internet



13. ufw : Firewall command



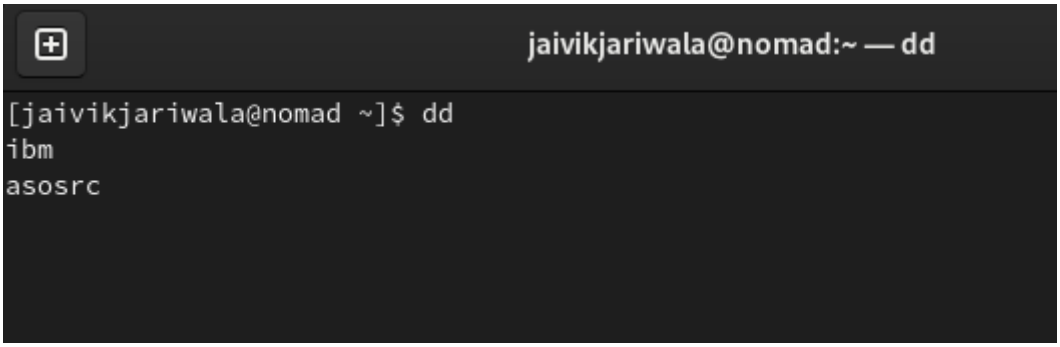
14. traceroute : Trace all the network hops to reach the destination



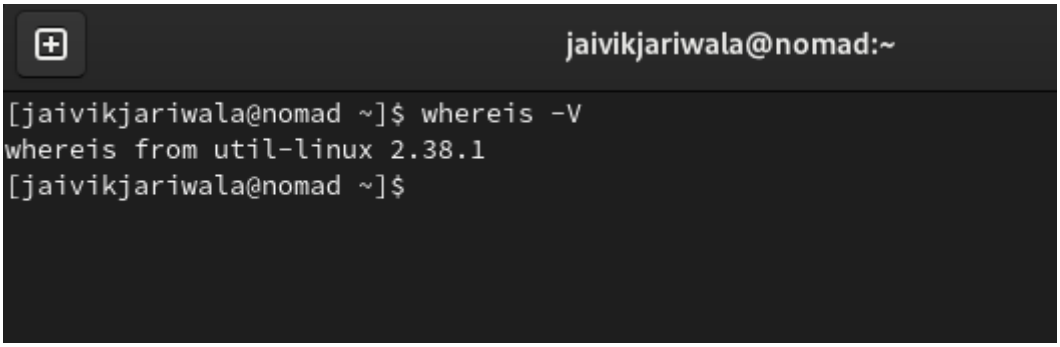
15. service : Linux command to start and stop services

16. alias : Create custom shortcuts for your regularly used commands

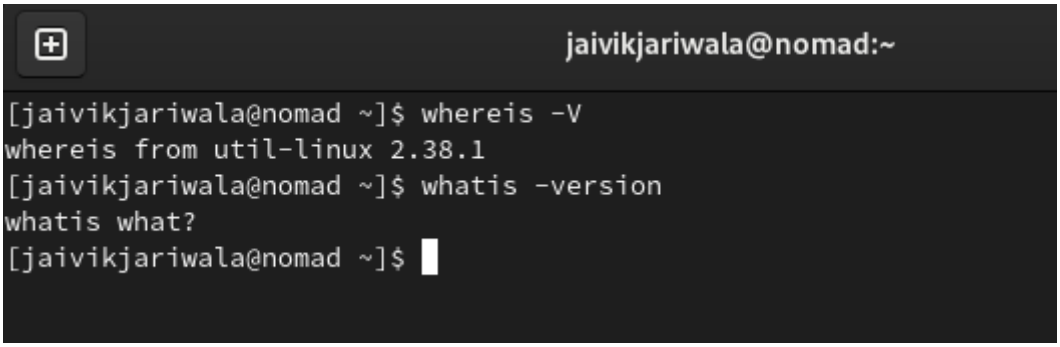
17. dd : Majorly used for creating bootable USB sticks



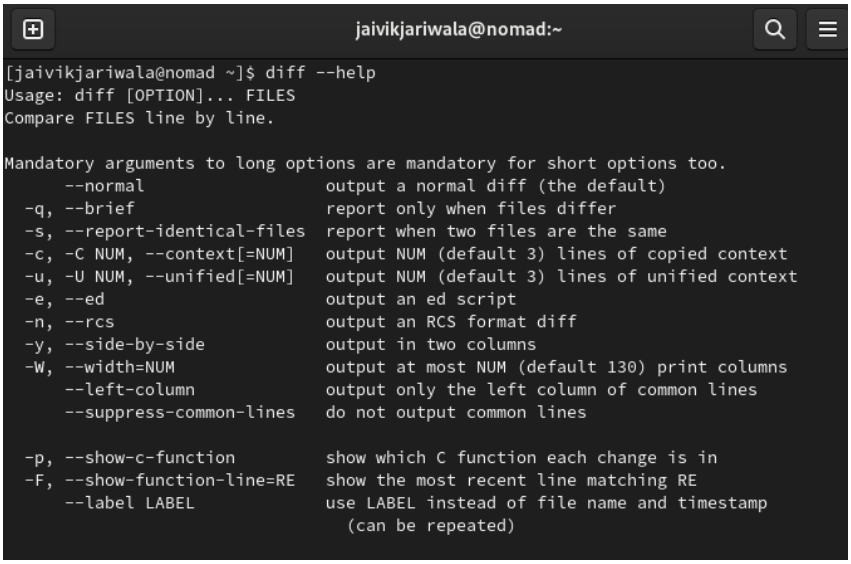
18. whereis : Locate the binary, source, and manual pages for a command



19. whatis : Find what a command is used for



20. diff : Find the difference between two files



21. ln : Create symbolic links (shortcuts) to other files

```
[jaivikjariwala@nomad ~]$ In
bash: In: command not found...
[jaivikjariwala@nomad ~]$
```

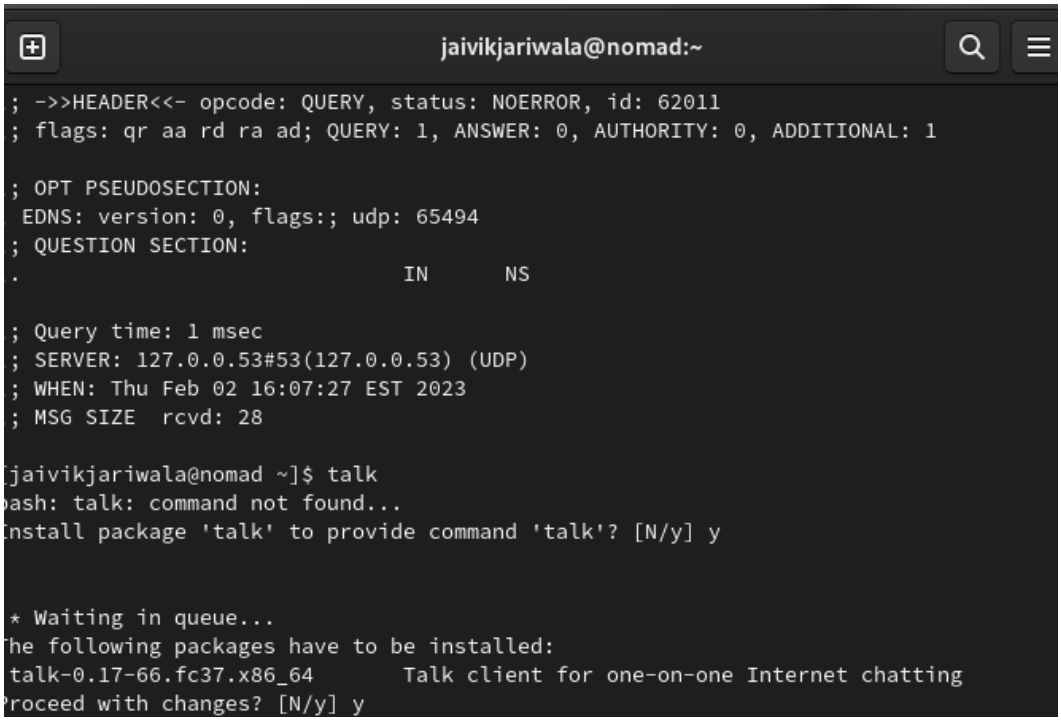
22. top : View active processes live with their system usage

jaivikjariwala@nomad:~ — top

```
top - 14:28:46 up 13 min,  1 user,  load average: 0.24, 0.29, 0.22
Tasks: 384 total,   1 running, 383 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.2 sy,  0.0 ni, 99.2 id,  0.0 wa,  0.2 hi,  0.0 si,  0.0 st
MiB Mem :  15614.5 total,  12031.6 free,   1263.8 used,   2319.1 buff/cache
MiB Swap:   8192.0 total,   8192.0 free,      0.0 used.  13614.9 avail Mem
```

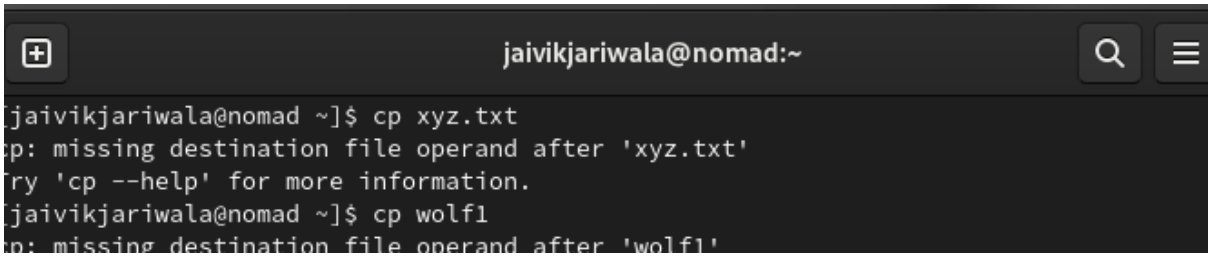
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3971	jaivikj+	20	0	772152	57936	44636	S	5.3	0.4	0:10.18	gnome-terminal-
2146	jaivikj+	20	0	8018936	216588	125552	S	3.7	1.4	1:09.19	gnome-shell
513	root	-51	0	0	0	0	S	1.0	0.0	0:02.76	irq/79-ASUP1205:00
901	systemd+	20	0	16156	7932	7004	S	0.3	0.0	0:02.21	systemd-oomd
2415	root	20	0	259640	29744	8056	S	0.3	0.2	0:00.58	sssd_kcm
5120	**jaivikj+**	**20**	**0**	**224996**	**3880**	**3040**	**R**	**0.3**	**0.0**	**0:00.03**	**top**
1	root	20	0	172164	17840	11060	S	0.0	0.1	0:01.74	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_flushwq
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-events_highpri
9	root	20	0	0	0	0	I	0.0	0.0	0:01.48	kworker/u40:0-btrfs-endio-write
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
12	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_kthread
13	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_rude_kthread
14	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_trace_kthread
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
16	root	20	0	0	0	0	I	0.0	0.0	0:00.99	rcu_preempt
17	root	rt	0	0	0	0	S	0.0	0.0	0:00.01	migration/0
19	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
20	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
21	root	rt	0	0	0	0	S	0.0	0.0	0:00.11	migration/1

23. useradd : Add new user or change existing users data



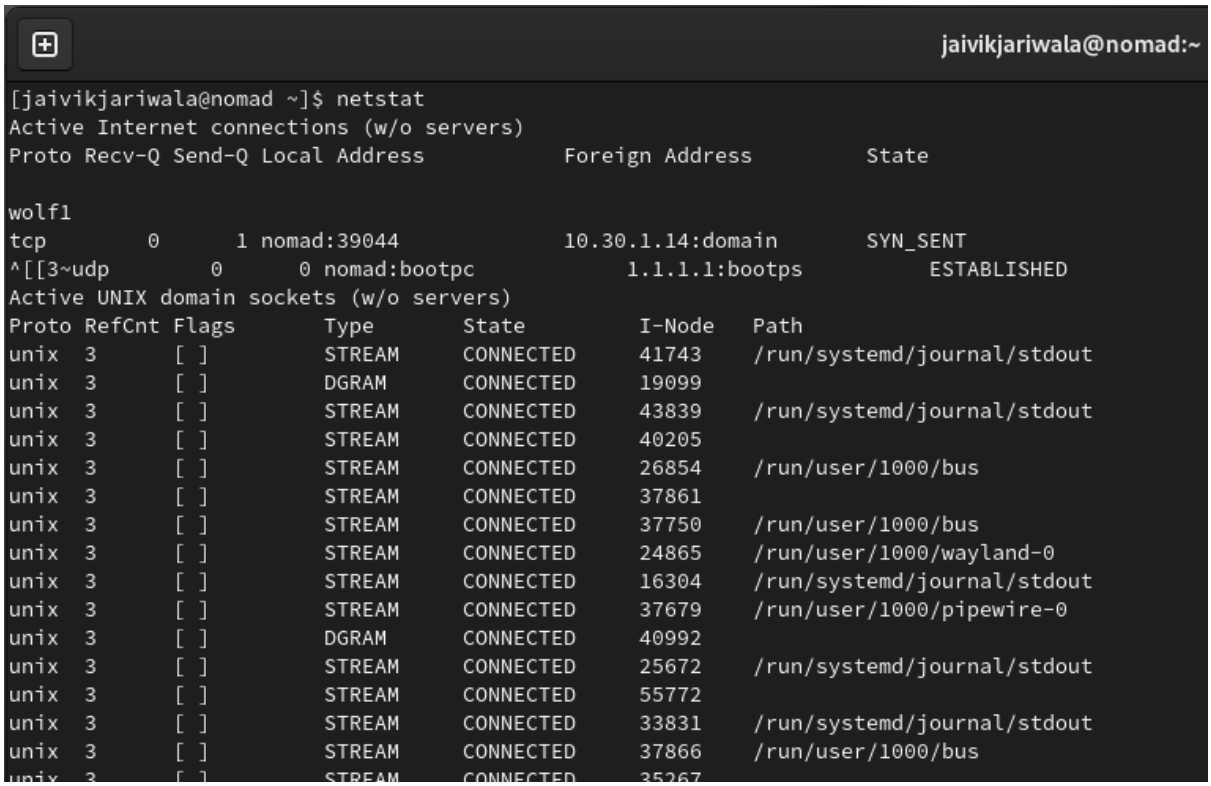
24. man : Access manual pages for all Linux commands

25. cp : Similar usage as mv but for copying files in Linux

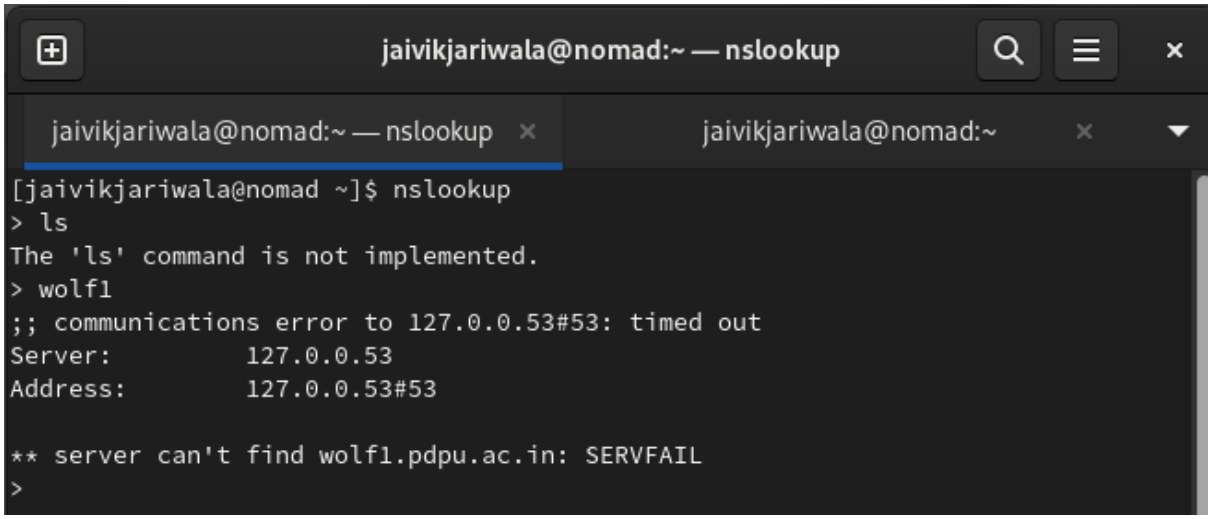


26. ln : Create symbolic links (shortcuts) to other files

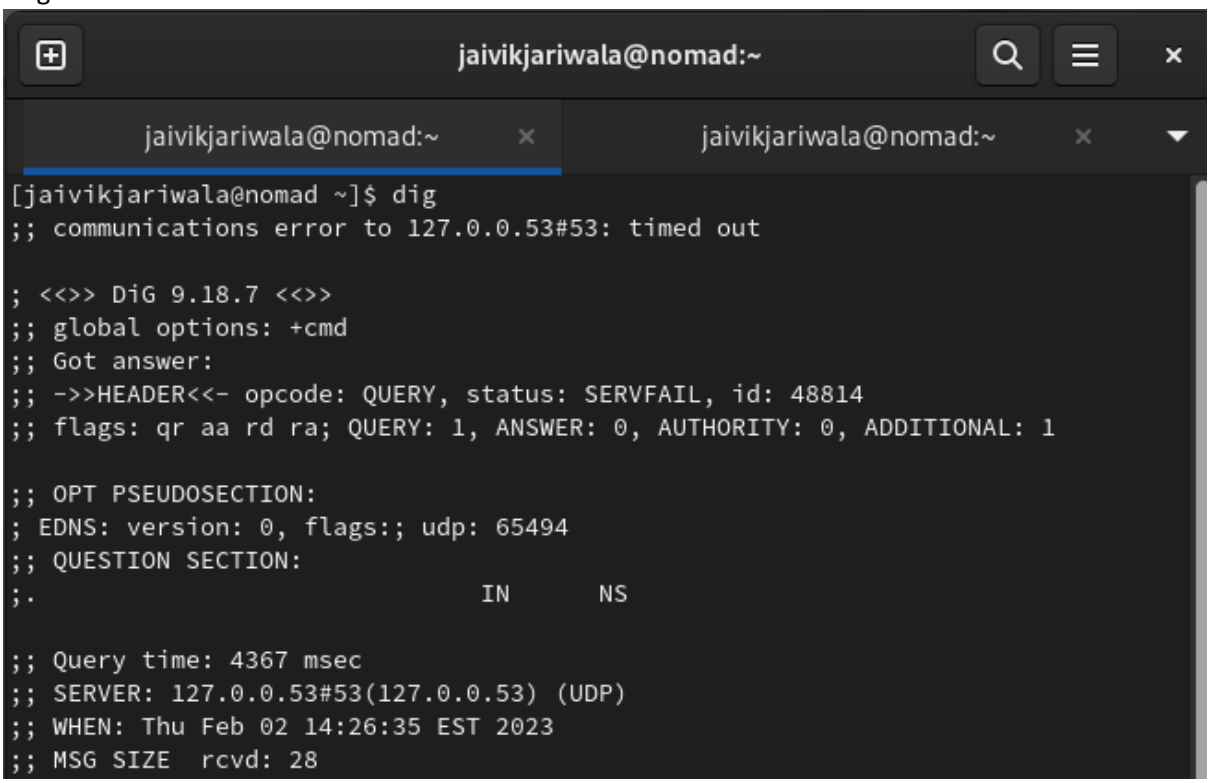
27. netstat : netstat command displays various network related information such as network connections, routing tables, interface statistics, masquerade connections, multicast memberships etc.



28. nslookup : A network utility program used to obtain information about Internet servers. As its name suggests, the utility finds name server information for domains by querying DNS.

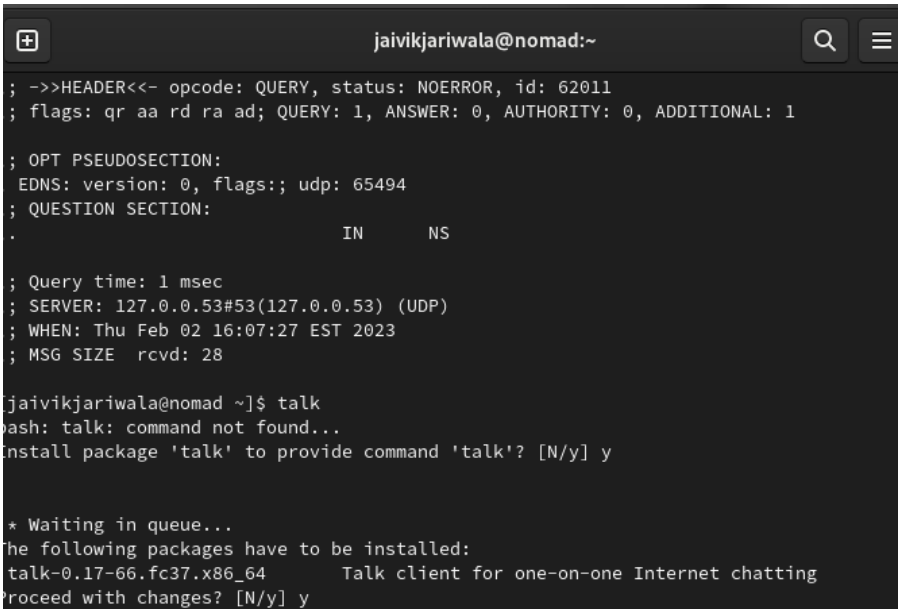


29. dig : dig is a tool for querying DNS nameservers for information about host addresses, mail exchanges, nameservers, and related information. This tool can be used from any Linux (Unix) or Macintosh OS X operating system. The most typical use of dig is to simply query a single host.

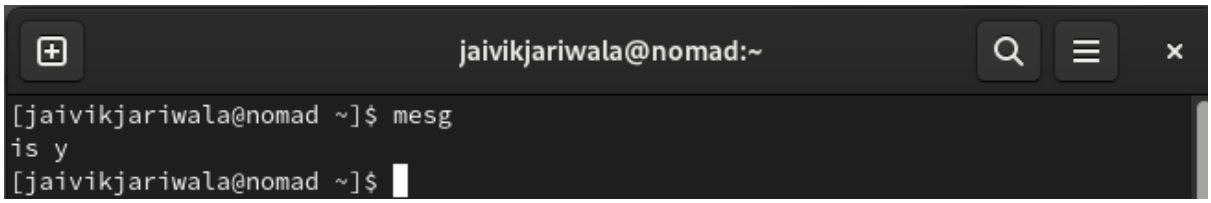


30. uptime : You have just connected to your Linux Server Machine and founds Something unusual or malicious, what you will do? Guessing…. NO, definitely not you could run uptime to verify what happened actually when the server was unattended.

31. wall : one of the most important command for administrator, wall sends a message to everybody logged in with their mesg permission set to “yes“. The message can be given as an argument to wall, or it can be sent to wall’s standard input.

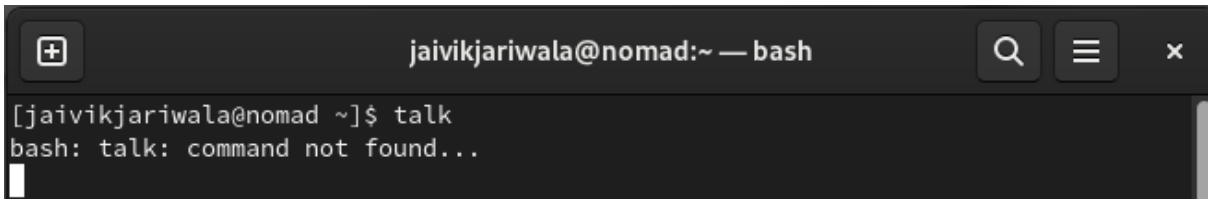


32. mesg : Lets you control if people can use the “write” command, to send text to you over the screen.



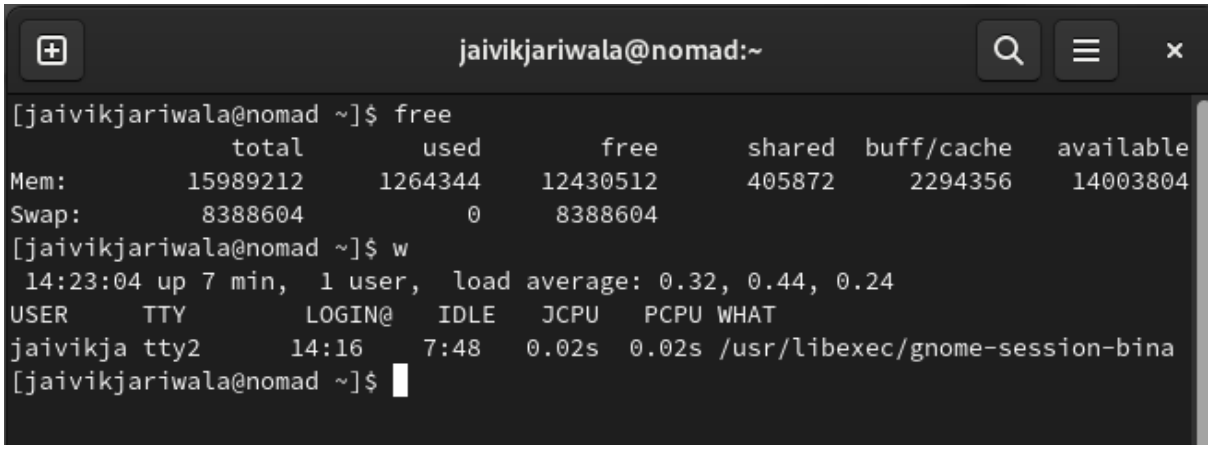
33. write : Let you send text directly to the screen of another Linux machine if ‘mesg’ is ‘y’.

34. talk : An enhancement to write command, talk command lets you talk to the logged in users.



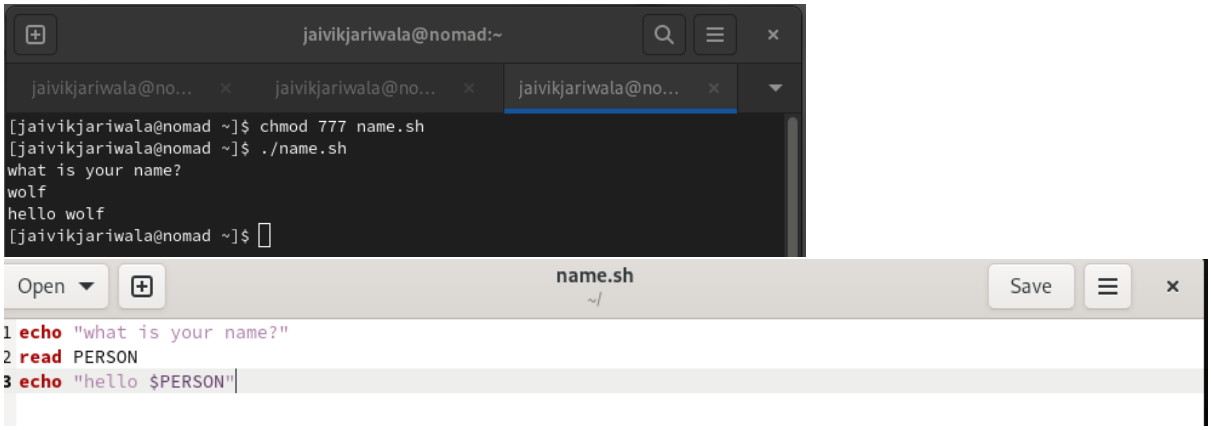
35. w : what command ‘w’ seems you funny? But actually it is not. t’s a command, even if it’s just one letter long! The command “w” is a combination of uptime and who commands given one immediately after the other, in that order.

36. rename : As the name suggests, this command rename files. rename will rename the specified files by replacing the first occurrence from the file name.

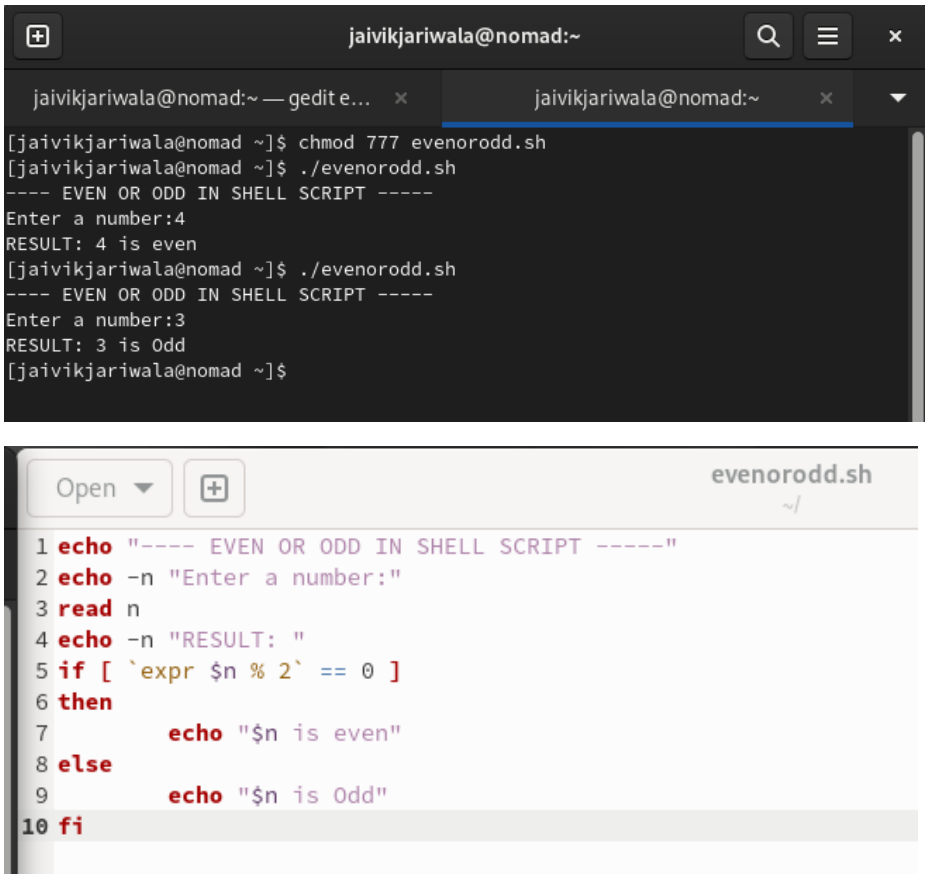


37. free : Keeping track of memory and resources is as much important, as any other task performed by an administrator, and ‘free‘ command comes to rescue here.

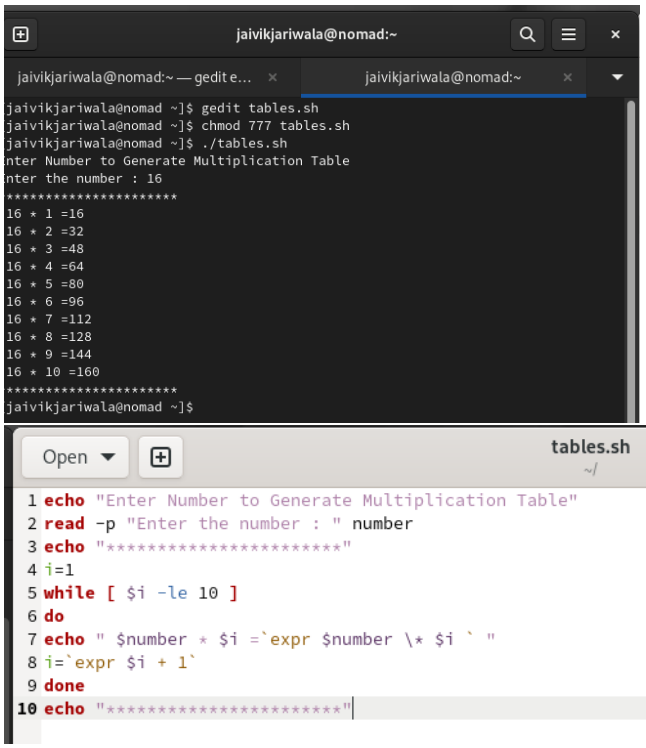
1) Write a shell script to print your name.



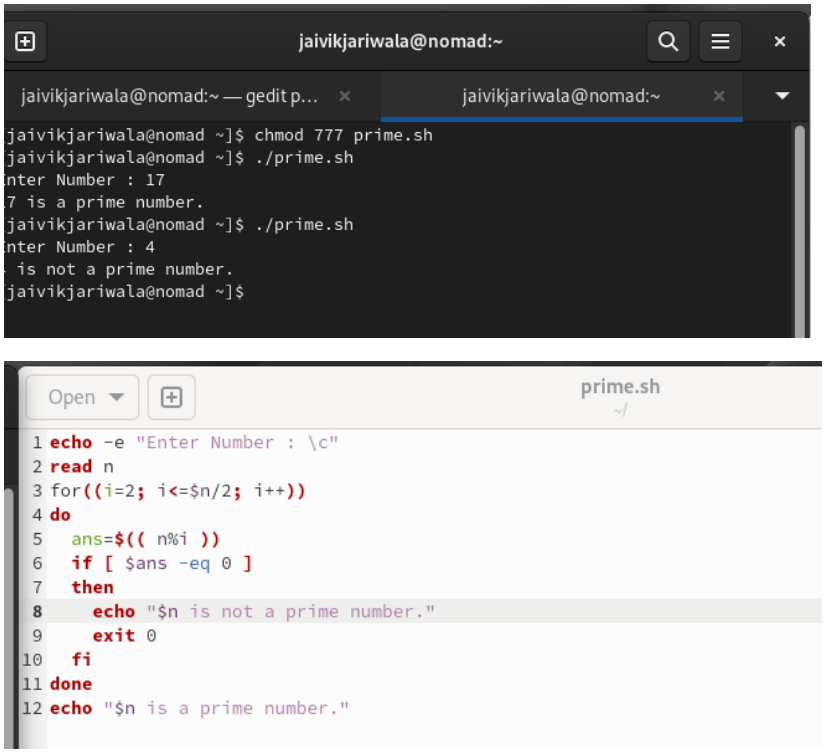
2) Write a shell script to find whether a number is even or odd.



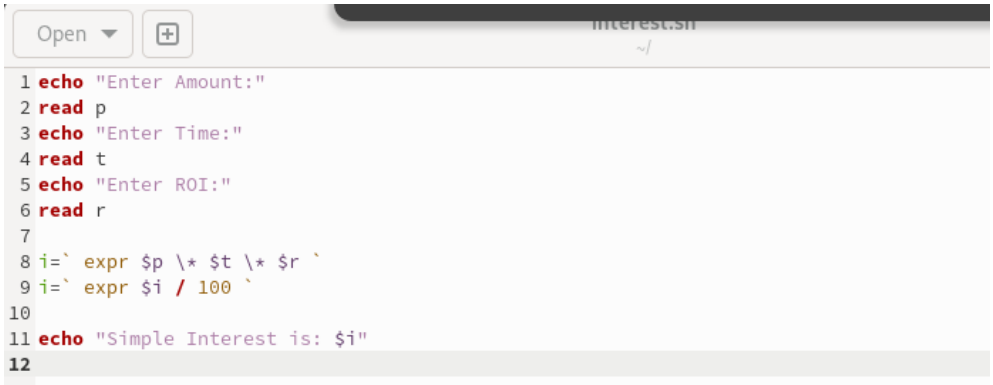
3) Write a script to print a table of a given number.



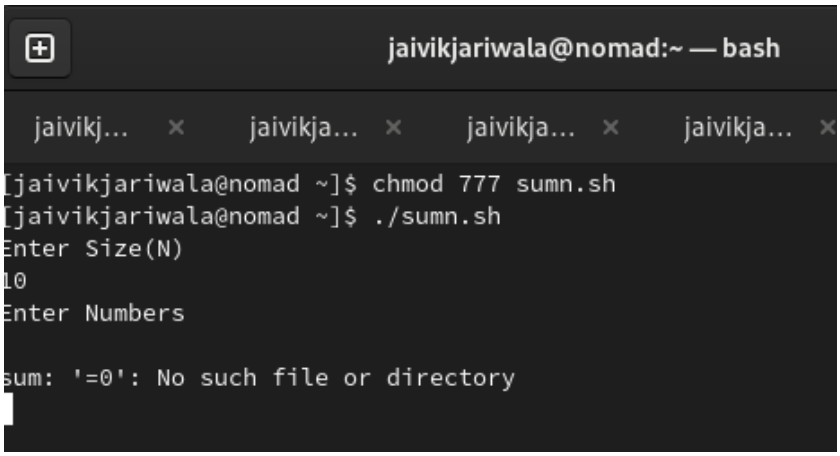
4) Write a shell script to check whether a given no. is prime or not.



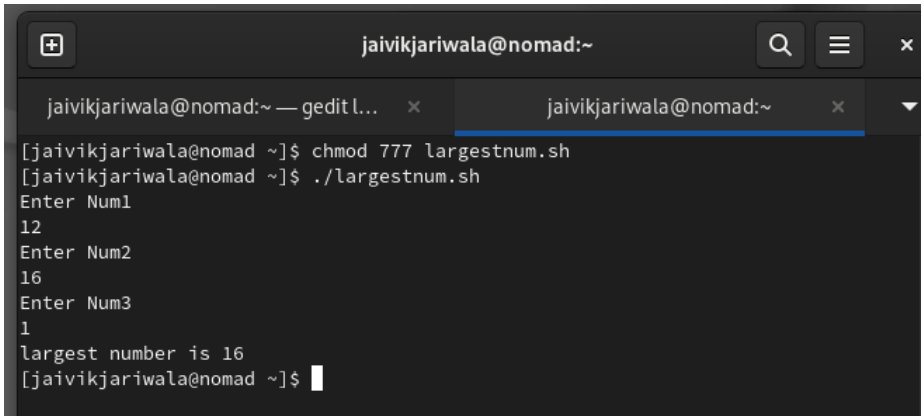
5) Write a shell script to find the simple interest.



6) Write a shell script to find sum of “n” numbers.



7) Write a shell script to find the largest number of three numbers.



Open

largestnum.sh

~/

```
1 echo "Enter Num1"
2 read num1
3 echo "Enter Num2"
4 read num2
5 echo "Enter Num3"
6 read num3
7
8 if [ $num1 -gt $num2 ] && [ $num2 -gt $num3 ]
9 then
10         echo "largest number is" $num1
11
12 elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
13 then
14         echo "largest number is" $num2
15
16 else
17         echo "largest number is" $num3
18
19 fi
```

Lab – 4 Jaivik Jariwala 21BCP004

Write a menu driven shell script, which will print the following menu and execute the given task.

- ❖ Display a calendar of current month
- ❖ Display today's date and time
- ❖ Display username those are currently logged in the system
- ❖ Display your name at the given x,y position.
- ❖ Display your terminal number

```
echo "Menu"
echo "1) Display Calendar of current month"
echo "2) Display Today's date and time"
echo "3) Display Username who is currently logged in"
echo "4) Display your name at given x,y position"
echo "5) Display your terminal number"
echo "6) exit"
echo "Enter your choice:"
read ch
case Sch in
1)cal;;
2) date;;
3) who;;
4) row=$(tput lines)
col=$(tput cols)
echo "Number of rows $row Cols=$col"
echo "Enter desired x,y position"
echo "x position="
read x
echo "y position="
read y
```

```
Menu
1) Display Calendar of current month
2) Display Today's date and time
3) Display Username who is currently logged in"
4) Display your name at given x,y position
5) Display your terminal number
6) exit"
Enter your choice:
1
```

Write a shell script which will generate first n Fibonacci numbers such as :1, 1, 2, 3, 5, 13,….etc

```
fibo()
{
    no=$1
    if [ $no -eq 1 ]; then
        return 0
    elif [ $no -eq 2 ]; then
        return 1
    else
        a1=`expr $no - 1`
        fibo $a1
        a=$(echo $?)

        b1=`expr $no + 1`
        fibo $b1
        b=$(echo $?)

        c=`expr $a + $b`
        return $c
    fi
}
```

```

clear

echo -e "Enter number of terms : \c"
read n

if [ $n -gt 0 ]; then
    for (( i=1; i<=$n; i++ ))
    do
        fibo $i
        echo $?
    done
else
    echo -e "Invalid input."
fi
```

```
Enter  number of terms : 6
0 1 1 2 3 5
```

Shell Script to print half pyramids using numbers

```
rows=5
for((i=rows; i>=1; i--))
do
  for((j=1; j<=i; j++))
  do
    echo -n "* "
  done
  echo
done
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Write a shell script to find the reverse of a given number.

```
read -p "Enter a number: " number
temp=$number
while [ $temp -ne 0 ]
do
    reverse=$reverse$((temp%10))
    temp=$((temp/10))
done
echo "Reverse of $number is $reverse"
```

```
Enter a number: 143
Reverse of 143 is 341
```

Write a shell script to find the sum of two floating point numbers

```
echo enter a and b
read a b
c=`echo $a+$b | bc
echo $c
```

```
enter a and b
4.5 6.7
11.2
```

Write a shell script that changes text to uppercase

```
echo "enter the string:"
read string
echo "the string in uppercase is:"
echo ${string^^}
```

```
enter the string:
rashida
the string in uppercase is:
RASHIDA
```

Write a shell script which prints “invalid no. of arguments” if more than 5 command line arguments otherwise print “valid no. of arguments”

```
6 7 8 5 4
valid no. of arguments
```

```
echo $#

if [ $# -le 5 ]
then
    echo "valid no of arguments"
else
    echo "Invalid no of arguments"

fi
```

.

Write a shell script to make the following operations menu based:

❖ Addition ❖ Subtraction ❖ Multiplication ❖ Division

```
echo -e "Addition: \nSubtraction: -\nMultiplication: x\nDivision: /"
read op

        case Sop in
         +) c expr $a $b
            echo "Sum of $a and $b is $c";;

         -) c= expr $a - sb
           echo "Difference of $a and $b is $c";;

         x) c="expr $a - sb"
           echo "Product of $a and $b is $c";;

         /) c="expr $a / $b";;
           echo "Division of $a and $b is $c";;

         .) echo "Invalid Operator"
            exit;;
        esac
```

```
Enter the first number

23

Enter the second number

45

Enter the operator:

Addition: +

Subtraction:

Multiplication: x

Division: /

Sum of 23 and 45 is 68
```

Write a shell script to find the sum of all digits for a given number

```
echo -n "Enter number: "
read n
sd=0
sum=0
while [ $n -gt 0 ]
do
    sd=$(( $n % 10 ))
    n=$(( $n / 10 ))
    sum=$(( $sum + $sd ))

done
echo "Sum of all digit is $sum"
```

```
Enter number: 67

Sum of all digit is 13
```

Write a shell script to find the factorial of a given number.

```
echo "Enter a number"
read num
fact=1
while [ $num -gt 1 ]
do
fact=$((fact * num))
num=$((num - 1))
done
echo Factorial=$fact
```

```
Enter a number: 7
Factorial=5040
```

Lab – 5 Jaivik Jariwala 21BCP004

Scheduling Algorithm

```
def fcfs(processes):
    #initialize the waiting time and turn around time
    time=0
    waiting_time=0
    completion_time=0
    turn_around_time=0

    #execute the processes one by one in fcfs
    for p in processes:
        #update waiting time for cuttent process
        waiting_time += time
        #execute the current process
        print("executing process", p.pid, "with burst time" , p.burst_time ,
"with arrival time", p.pid)
        time += p.burst_time
        #calulate turnaround time for current processs
        turn_around_time += time
        #calculate completion time for current process
        completion_time += time
        #print the current process details
        print("waiting time", waiting_time)
        print("turn around time", turn_around_time)
        print("completion time", completion_time)

    #print the total burst and total average time
    total_waiting_time = waiting_time
    average_waiting_time = total_waiting_time / len(processes)
    print("total waiting time", total_waiting_time)
    print("avarage waiting time", average_waiting_time)
```

```
fcfs(processes)
```

```
executing process 1 with burst time 10 with arrival time 1
waiting time 0
turn around time 10
completion time 10
executing process 2 with burst time 5 with arrival time 2
waiting time 10
turn around time 25
completion time 25
executing process 3 with burst time 8 with arrival time 3
waiting time 25
turn around time 48
completion time 48
executing process 4 with burst time 3 with arrival time 4
waiting time 48
turn around time 74
completion time 74
total waiting time 48
avarage waiting time 12.0
```

```
# Define a function to execute the processes using SJF scheduling algorithm
def sjf(processes):
    # Sort the processes by their burst time in non-decreasing order
    processes.sort(key=lambda p: p.burst_time)

    # Initialize the time and waiting time variables
    time = 0
    waiting_time = 0
    completion_time = 0
    turn_around_time = 0

    # Execute the processes one by one in SJF order
    for p in processes:
        #update waiting time for cuttent process
        waiting_time += time
        #execute the current process
        print("executing process", p.pid, "with burst time" , p.burst_time ,
"with arrival time", p.pid)
        time += p.burst_time
        #calulate turnaround time for current processs
        turn_around_time += time
        #calculate completion time for current process
        completion_time += time
        #print the current process details
        print("waiting time", waiting_time)
        print("turn around time", turn_around_time)
        print("completion time", completion_time)

    # Print the total waiting time and average waiting time
    total_waiting_time = waiting_time
    average_waiting_time = total_waiting_time / len(processes)
    print("Total waiting time:", total_waiting_time)
    print("Average waiting time:", average_waiting_time)


sjf(processes)
```

```
executing process 4 with burst time 3 with arrival time 4
waiting time 0
turn around time 3
completion time 3
executing process 2 with burst time 5 with arrival time 2
waiting time 3
turn around time 11
completion time 11
executing process 3 with burst time 8 with arrival time 3
waiting time 11
turn around time 27
completion time 27
executing process 1 with burst time 10 with arrival time 1
waiting time 27
turn around time 53
completion time 53
Total waiting time: 27
Average waiting time: 6.75
```

```
# Define a function to execute the processes using Round Robin scheduling algorithm
def round_robin(processes, quantum):
    # Initialize the waiting time, turnaround time, completion time, and time variables
    waiting_time = [0] * len(processes)
    turnaround_time = [0] * len(processes)
    completion_time = [0] * len(processes)
    time = 0

    # Create a copy of the processes list to keep track of the remaining burst time of each process
    remaining_burst_time = [p.burst_time for p in processes]

    # Execute the processes one by one in Round Robin order
    while True:
        all_processes_completed = True
        for i, p in enumerate(processes):
            if remaining_burst_time[i] > 0:
                # Calculate the time slice for the current process
                time_slice = min(remaining_burst_time[i], quantum)

                # Execute the current process for the time slice
                print("Executing process", p.pid, "for", time_slice, "units of time")
                remaining_burst_time[i] -= time_slice
                time += time_slice

                # Check if the current process has completed
                if remaining_burst_time[i] == 0:
                    # Calculate turnaround time and completion time for the current process
                    turnaround_time[i] = time - p.arrival_time
                    completion_time[i] = time

                    # Print the waiting time, turnaround time, and completion time for the current process
                    print("Process", p.pid, "completed at time", completion_time[i])
                    print("Waiting time for process", p.pid, "is", waiting_time[i])
                    print("Turnaround time for process", p.pid, "is", turnaround_time[i])
                    print("Completion time for process", p.pid, "is", completion_time[i])
```

```
                else:
                    # Update the waiting time for the current process
                    waiting_time[i] = time - p.arrival_time

                all_processes_completed = False

        # Check if all processes have completed
        if all_processes_completed:
            break

    # Calculate the total waiting time and average waiting time for all
processes
    total_waiting_time = sum(waiting_time)
    average_waiting_time = total_waiting_time / len(processes)

    # Print the total and average waiting time
    print("Total waiting time:", total_waiting_time)
    print("Average waiting time:", average_waiting_time)
```

```
quantum = 2
round_robin(processes, quantum)
```

```
Executing process 4 for 2 units of time
Executing process 2 for 2 units of time
Executing process 3 for 2 units of time
Executing process 1 for 2 units of time
Executing process 4 for 1 units of time
Process 4 completed at time 9
Waiting time for process 4 is -1
Turnaround time for process 4 is 6
Completion time for process 4 is 9
Executing process 2 for 2 units of time
Executing process 3 for 2 units of time
Executing process 1 for 2 units of time
Executing process 2 for 1 units of time
Process 2 completed at time 16
Waiting time for process 2 is 10
Turnaround time for process 2 is 15
Completion time for process 2 is 16
Executing process 3 for 2 units of time
Executing process 1 for 2 units of time
Executing process 3 for 2 units of time
Process 3 completed at time 22
Waiting time for process 3 is 16
Turnaround time for process 3 is 20
Completion time for process 3 is 22
Executing process 1 for 2 units of time
Turnaround time for process 1 is 26
Completion time for process 1 is 26
Total waiting time: 49
Average waiting time: 12.25
```

```
def srtf(processes):
    # Initialize the waiting time, turnaround time, completion time, and time
variables
    waiting_time = [0] * len(processes)
    turnaround_time = [0] * len(processes)
    completion_time = [0] * len(processes)
    time = 0

    # Create a copy of the processes list to keep track of the remaining burst
time of each process
    remaining_burst_time = [p.burst_time for p in processes]

    # Execute the processes one by one in Shortest Remaining Time First order
    while True:
        # Find the process with the shortest remaining burst time
        shortest_time = float('inf')
        shortest_index = None
        for i, p in enumerate(processes):
            if remaining_burst_time[i] > 0 and remaining_burst_time[i] <
shortest_time:
                shortest_time = remaining_burst_time[i]
                shortest_index = i

        if shortest_index is None:
            # All processes completed
            break

        # Execute the process with the shortest remaining burst time
        p = processes[shortest_index]
        print("Executing process", p.pid, "for",
remaining_burst_time[shortest_index], "units of time")
        remaining_burst_time[shortest_index] = 0
        time += remaining_burst_time[shortest_index]

        # Calculate turnaround time and completion time for the current
process
        turnaround_time[shortest_index] = time - p.arrival_time
        completion_time[shortest_index] = time

        # Print the waiting time, turnaround time, and completion time for the
current process
        print("Process", p.pid, "completed at time",
completion_time[shortest_index])
        print("Waiting time for process", p.pid, "is",
waiting_time[shortest_index])
        print("Turnaround time for process", p.pid, "is",
turnaround_time[shortest_index])
```

```
        print("Completion time for process", p.pid, "is",
completion_time[shortest_index])

        # Update the waiting time for all remaining processes
        for i, p in enumerate(processes):
            if i != shortest_index and remaining_burst_time[i] > 0:
                waiting_time[i] += remaining_burst_time[shortest_index]

    # Calculate the total waiting time and average waiting time for all
processes
    total_waiting_time = sum(waiting_time)
    average_waiting_time = total_waiting_time / len(processes)

    # Print the total and average waiting time
    print("Total waiting time:", total_waiting_time)
    print("Average waiting time:", average_waiting_time)
```

```
srtf(processes)
```

```
Executing process 4 for 3 units of time
Process 4 completed at time 0
Waiting time for process 4 is 0
Turnaround time for process 4 is -3
Completion time for process 4 is 0
Executing process 2 for 5 units of time
Process 2 completed at time 0
Waiting time for process 2 is 0
Turnaround time for process 2 is -1
Completion time for process 2 is 0
Executing process 3 for 8 units of time
Process 3 completed at time 0
Waiting time for process 3 is 0
Turnaround time for process 3 is -2
Completion time for process 3 is 0
Executing process 1 for 10 units of time
Process 1 completed at time 0
Waiting time for process 1 is 0
Turnaround time for process 1 is 0
Completion time for process 1 is 0
Total waiting time: 0
Average waiting time: 0.0
```

```
class Process:
    def __init__(self, pid, arrival_time, burst_time, priority):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.priority = priority
        self.remaining_time = burst_time

    def __lt__(self, other):
        return self.priority < other.priority

def priority_scheduling(processes):
    time = 0
    n = len(processes)
    completed = 0
    waiting_time = 0
    turnaround_time = 0
    response_time = [0] * n
    remaining_time = [p.burst_time for p in processes]

    while completed != n:
        highest_priority_process = None
        for i in range(n):
            if processes[i].arrival_time <= time and remaining_time[i] > 0:
                if highest_priority_process is None or processes[i] <
processes[highest_priority_process]:
                    highest_priority_process = i

        if highest_priority_process is None:
            time += 1
            continue

        response_time[highest_priority_process] = time -
processes[highest_priority_process].arrival_time

        remaining_time[highest_priority_process] -= 1
        time += 1

        if remaining_time[highest_priority_process] == 0:
            completed += 1
            waiting_time += time -
processes[highest_priority_process].arrival_time -
processes[highest_priority_process].burst_time
            turnaround_time += time -
processes[highest_priority_process].arrival_time

    print("Average waiting time =", waiting_time / n)
    print("Average turnaround time =", turnaround_time / n)
```

```
    print("Average response time =", sum(response_time) / n)


if __name__ == '__main__':
    processes = [
        Process(1, 0, 7, 2),
        Process(2, 2, 4, 3),
        Process(3, 4, 1, 1),
        Process(4, 5, 4, 4),
        Process(5, 6, 3, 2),
    ]
    priority_scheduling(processes)
```

```
Average waiting time = 4.4
Average turnaround time = 8.2
Average response time = 7.2
```

Disk Scheduling Algorithm

```python
import matplotlib.pyplot as plt

# FCFS disk scheduling
def fcfs(head, queue):
    total_movement = 0
    for i in range(len(queue)):
        diff = abs(head - queue[i])
        total_movement += diff
        head = queue[i]
    return total_movement

# SSTF disk scheduling
def sstf(head, queue):
    total_movement = 0
    while queue:
        queue.sort(key=lambda x: abs(x-head))
        diff = abs(head - queue[0])
        total_movement += diff
        head = queue[0]
        queue.pop(0)
    return total_movement

# SCAN disk scheduling
def scan(head, queue):
    queue.sort()
    total_movement = 0
    if head not in queue:
        print("Head not in queue.")
        return 0
    for i in range(queue.index(head) + 1):
        diff = abs(head - queue[i])
        total_movement += diff
        head = queue[i]
    total_movement += abs(head - queue[0])
    head = queue[0]
    for i in range(queue.index(head), len(queue)):
        diff = abs(head - queue[i])
        total_movement += diff
        head = queue[i]
    return total_movement
```

```python
# C-SCAN disk scheduling
def cscan(head, queue):
    queue.sort()
    total_movement = 0
    if head not in queue:
        print("Head not in queue.")
        return 0
    for i in range(queue.index(head) + 1):
        diff = abs(head - queue[i])
        total_movement += diff
        head = queue[i]
    total_movement += abs(head - queue[0])
    head = queue[0]
    total_movement += abs(head - queue[-1])
    head = queue[-1]
    for i in range(0, queue.index(head)):
        diff = abs(head - queue[i])
        total_movement += diff
        head = queue[i]
    return total_movement

# LOOK disk scheduling
def look(head, queue):
    queue.sort()
    total_movement = 0
    if head not in queue:
        print("Head not in queue.")
        return 0
    for i in range(queue.index(head) + 1):
        diff = abs(head - queue[i])
        total_movement += diff
        head = queue[i]
    for i in range(queue.index(head), len(queue)):
        diff = abs(head - queue[i])
        total_movement += diff
        head = queue[i]
    for i in range(queue.index(head) - 1, -1, -1):
        diff = abs(head - queue[i])
        total_movement += diff
        head = queue[i]
    return total_movement
```

```
# C-LOOK disk scheduling
def clook(head, queue):
    queue.sort()
    total_movement = 0
    for i in range(len(queue)):
        if queue[i] >= head:
            total_movement += abs(head - queue[i])
            head = queue[i]
            break
    for i in range(len(queue)):
        if queue[i] < head:
            total_movement += abs(head - queue[i])
            head = queue[i]
    for i in range(len(queue)-1, -1, -1):
        if queue[i] >= head:
            total_movement += abs(head - queue[i])
            head = queue[i]
    for i in range(len(queue)-2, -1, -1):
        if queue[i] < head:
            total_movement += abs(head - queue[i])
            head = queue[i]
    return total_movement

# Sample disk queue and head position
queue = [86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130]
head = 143

 # Calculate total head movement for each algorithm
fcfs_total = fcfs(head, queue)
sstf_total = sstf(head, queue.copy())
scan_total = scan(head, queue.copy())
cscan_total = cscan(head, queue.copy())
look_total = look(head, queue.copy())
clook_total = look(head, queue.copy())

print("FCFS total head movement:", fcfs_total)
print("SSTF total head movement:", sstf_total)
print("SCAN total head movement:", scan_total)
print("C-SCAN total head movement:", cscan_total)
print("LOOK total head movement:", look_total)
print("C-LOOK total head movement:", clook_total)

plt.bar(["FCFS", "SSTF", "SCAN", "C-SCAN", "LOOK", "C-LOOK"],
[fcfs_total, sstf_total, scan_total, cscan_total, look_total, clook_total])
plt.title("Disk Scheduling Algorithms")
plt.ylabel("Total Head Movement")
plt.show()
```

Output :

```
Head not in queue.
Head not in queue.
Head not in queue.
Head not in queue.
FCFS total head movement: 7081
SSTF total head movement: 1745
SCAN total head movement: 0
C-SCAN total head movement: 0
LOOK total head movement: 0
C-LOOK total head movement: 0
```

Lab – 7 Jaivik Jariwala 21BCP004

Page Replacement

FIFO Algorithm

```python
def fifo_page_replacement(pages, frames):
    # initialize page fault counter
    page_faults = 0
    # initialize list of frames
    frame_list = []
    # iterate over pages
    for page in pages:
        # if page is not in frames
        if page not in frame_list:
            # if there is still space in frames
            if len(frame_list) < frames:
                frame_list.append(page)
            # if there is no space in frames
            else:
                # remove the oldest page in frames
                frame_list.pop(0)
                # add the new page to frames
                frame_list.append(page)
            # increment page fault counter
            page_faults += 1
    return page_faults
```

```python
pages = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
frames = 3
page_faults = fifo_page_replacement(pages, frames)
print("Number of page faults:", page_faults)
```

```
Number of page faults: 10
```

Optimal Algorithm

```
def optimal_page_replacement(pages, frames):
    # initialize page fault counter
    page_faults = 0
    # initialize list of frames
    frame_list = []
    # iterate over pages
    for page in pages:
        # if page is not in frames
        if page not in frame_list:
            # if there is still space in frames
            if len(frame_list) < frames:
                frame_list.append(page)
            # if there is no space in frames
            else:
                # initialize dictionary of indices
                indices = {}
                # iterate over frames
                for f in frame_list:
                    # if page is not in the remaining pages
                    if f not in pages[pages.index(page):]:
                        # store the index
                        indices[f] = 1000000
                    # if page is in the remaining pages
                    else:
                        # store the index of its next occurrence
                        indices[f] = pages.index(f, pages.index(page))
                # remove the frame with the maximum index
                frame_list.remove(max(indices, key=indices.get))
                # add the new page to frames
                frame_list.append(page)
            # increment page fault counter
            page_faults += 1
    return page_faults
```

```
pages = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
frames = 3
page_faults = optimal_page_replacement(pages, frames)
print("Number of page faults:", page_faults)
```

Number of page faults: 10

LRU

```
from collections import deque

def lru_page_replacement(pages, frames):
    # initialize page fault counter
    page_faults = 0
    # initialize list of frames
    frame_list = []
    # initialize queue to keep track of the least recently used pages
    page_queue = deque()
    # iterate over pages
    for page in pages:
        # if page is not in frames
        if page not in frame_list:
            # if there is still space in frames
            if len(frame_list) < frames:
                frame_list.append(page)
            # if there is no space in frames
            else:
                # remove the least recently used page from frames
                lru_page = page_queue.popleft()
                frame_list.remove(lru_page)
                # add the new page to frames
                frame_list.append(page)
            # add the new page to the least recently used queue
            page_queue.append(page)
            # increment page fault counter
            page_faults += 1
        # if page is already in frames
        else:
            # remove page from the least recently used queue
            page_queue.remove(page)
            # add page to the end of the least recently used queue
            page_queue.append(page)
    return page_faults
```

```
pages = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
frames = 3
page_faults = lru_page_replacement(pages, frames)
print("Number of page faults:", page_faults)
```

Number of page faults: 10

LFU

```
from collections import defaultdict

def lfu_page_replacement(pages, frames):
    # initialize page fault counter
    page_faults = 0
    # initialize list of frames
    frame_list = []
    # initialize dictionary to keep track of the frequency of pages in frames
    page_freq = defaultdict(int)
    # iterate over pages
    for page in pages:
        # if page is not in frames
        if page not in frame_list:
            # if there is still space in frames
            if len(frame_list) < frames:
                frame_list.append(page)
            # if there is no space in frames
            else:
                # find the least frequently used page in frames
                min_freq = min(page_freq.values())
                lfu_pages = [p for p in frame_list if page_freq[p] ==
min_freq]
                lfu_page = lfu_pages[0]
                for p in lfu_pages:
                    if page_freq[p] < page_freq[lfu_page]:
                        lfu_page = p
                frame_list.remove(lfu_page)
                # add the new page to frames
                frame_list.append(page)
                page_freq.clear()
            # increment frequency of new page
            page_freq[page] += 1
            # increment page fault counter
            page_faults += 1
        # if page is already in frames
        else:
            # increment frequency of page
            page_freq[page] += 1
    return page_faults
```

```
pages = [1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5]
frames = 3
page_faults = lfu_page_replacement(pages, frames)
print("Number of page faults:", page_faults)
```

Number of page faults: 8

Lab – 8 Jaivik Jariwala 21BCP004

Deadlocks and Concurrency

Producer Consumer

```
import threading
import time
import random
from queue import Queue

# Define the shared queue with a maximum size
queue = Queue(maxsize=5)

# Define a function to simulate the producer thread
def producer():
    while True:
        # Generate a random number and add it to the queue
        item = random.randint(1, 10)
        queue.put(item)
        print(f"Producer added {item} to the queue")
        # Sleep for a random amount of time
        time.sleep(random.uniform(0.5, 1.5))

# Define a function to simulate the consumer thread
def consumer():
    while True:
        # Get an item from the queue
        item = queue.get()
        print(f"Consumer removed {item} from the queue")
        # Sleep for a random amount of time
        time.sleep(random.uniform(0.5, 1.5))
        # Indicate that the task is complete
        queue.task_done()

# Create and start the producer and consumer threads
producer_thread = threading.Thread(target=producer)
consumer_thread = threading.Thread(target=consumer)
producer_thread.start()
consumer_thread.start()

# Wait for the threads to complete (which they never will in this example)
producer_thread.join()
consumer_thread.join()
```

Output exceeds the size limit. Open the full output data in a text editor
Producer added 1 to the queue
Consumer removed 1 from the queue
Producer added 9 to the queueConsumer removed 9 from the queue

Producer added 4 to the queue
Consumer removed 4 from the queue
Producer added 2 to the queue
Consumer removed 2 from the queue
Producer added 3 to the queue
Consumer removed 3 from the queue
Producer added 2 to the queueConsumer removed 2 from the queue

Producer added 4 to the queueConsumer removed 4 from the queue

Producer added 5 to the queue
Consumer removed 5 from the queue
Producer added 10 to the queueConsumer removed 10 from the queue

Producer added 4 to the queue
Consumer removed 4 from the queue
Producer added 2 to the queue
Consumer removed 2 from the queue
Producer added 2 to the queue
Consumer removed 2 from the queue
Producer added 2 to the queue
...
Producer added 6 to the queue
Consumer removed 6 from the queue
Producer added 4 to the queue
Producer added 1 to the queue

Banker's Algorithm

```
# Define the available resources and maximum resource needs for each process
available = [3, 3, 2]
maximum = [
    [7, 5, 3],
    [3, 2, 2],
    [9, 0, 2],
    [2, 2, 2],
    [4, 3, 3]
]

# Define the currently allocated resources for each process
allocation = [
    [0, 1, 0],
    [2, 0, 0],
    [3, 0, 2],
    [2, 1, 1],
    [0, 0, 2]
]

# Define the remaining need for each process
need = [
    [7, 4, 3],
    [1, 2, 2],
    [6, 0, 0],
    [0, 1, 1],
    [4, 3, 1]
]

# Define a function to check if a state is safe
def is_safe(available, allocation, need):
    # Create copies of the available, allocation, and need matrices
    work = available.copy()
    finish = [False] * len(allocation)
    allocation = [row.copy() for row in allocation]
    need = [row.copy() for row in need]

    # Iterate through the processes
    while True:
        # Find a process that can be completed
        found = False
        for i in range(len(allocation)):
            if not finish[i] and all(need[i][j] <= work[j] for j in
range(len(available))):
                found = True
                # Release the allocated resources
                for j in range(len(available)):
                    work[j] += allocation[i][j]
```

```
                finish[i] = True
                break
        # If no process can be completed, exit the loop
        if not found:
            break

    # If all processes have been completed, the state is safe
    return all(finish)

# Define a function to request resources
def request_resources(process, request, available, allocation, need):
    # Check if the request is valid
    if any(request[i] > need[process][i] or request[i] > available[i] for i in
range(len(available))):
        return False

    # Attempt to allocate the resources
    for i in range(len(available)):
        available[i] -= request[i]
        allocation[process][i] += request[i]
        need[process][i] -= request[i]

    # Check if the new state is safe
    if is_safe(available, allocation, need):
        return True
    else:
        # If the new state is not safe, revert the allocation
        for i in range(len(available)):
            available[i] += request[i]
            allocation[process][i] -= request[i]
            need[process][i] += request[i]
        return False

# Test the request_resources function with a sample request
print(f"Initial state: Available = {available}, Allocation = {allocation},
Need = {need}")
if request_resources(1, [1, 0, 2], available, allocation, need):
    print("Request granted")
    print(f"New state: Available = {available}, Allocation = {allocation},
Need = {need}")
else:
    print("Request denied")
    print(f"State unchanged: Available = {available}, Allocation =
{allocation}, Need = {need}")
```

Initial state:

Available = [3, 3, 2],

Allocation = [[0, 1, 0], [2, 0, 0], [3, 0, 2], [2, 1, 1], [0, 0, 2]],

Need = [[7, 4, 3], [1, 2, 2], [6, 0, 0], [0, 1, 1], [4, 3, 1]]

Request granted

New state:

Available = [2, 3, 0],

Allocation = [[0, 1, 0], [3, 0, 2], [3, 0, 2], [2, 1, 1], [0, 0, 2]],

Need = [[7, 4, 3], [0, 2, 0], [6, 0, 0], [0, 1, 1], [4, 3, 1]]