OPERATING SYSTEMS LAB

PAPER CODE (ETCS-352)

Faculty Name : Ms. Vandana Choudhary
Assistant Professor

Student name : Sarthak Bhatia
Roll No. : 07714803118
Semester : 6th
Group: I-4

Maharaja Agrasen Institute of Technology, PSP Area,
Sector – 22, Rohini, New Delhi – 110086

OPERATING SYSTEMS LAB

PRACTICAL RECORD

PAPER CODE : ETCS-352

Name of the student : Sarthak Bhatia

University Roll No. : 07714803118

Branch : IT

Section/ Group : 6I - 4

PRACTICAL DETAILS

Exp. No	Experiment Name	Date of performance	Date of checking	Remarks	Marks (10)
1	To write the introduction of linux operationg system and its basic commands	15/03/21	22/03/21		
2	Write a program to implement CPU scheduling for first come first serve.	22/03/21	19/04/21		
3	Write a program to implement CPU scheduling for shortest job first.	19/04/21	26/04/21		
4	Write a program to implement CPU scheduling for Round Robin.	26/04/21	03/05/21		
5	Write a program to perform priority scheduling.	03/05/21	09/05/21		
6a	Write a program for page replacement policy using LRU	17/05/21	24/05/21		
6b	Write a program for page replacement policy using FIFO.	24/05/21	31/05/21		
6c	Write a program for page replacement policy using Optimal.	24/05/21	31/05/21		
7	Write a program to implement first fit, best fit and worst fit algorithm for memory management.	31/05/21	07/06/21		
8	Write a program to implement reader/writer problem using semaphore.	07/06/21	14/06/21		
9	Write a program to implement Banker's algorithm for deadlock avoidance.	07/06/21	14/06/21		

EXPERIMENT-1 Date-15/3/21

- Aim - To write the Introduction of Linux Operating System and its basic components.

Linux Operating System is one of the most popular UNIX operating system. Its open source and a free to use operating system. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux system

1. Kernal - Kernal is the core part of Linux. It is responsible for all major activities of this system. It consists of various modules and interact directly with the underlying hardware. Kernal provides the required abstraction to hide low level hardware details to system or application programs.
2. System Library - System libraries are special functions or programs using which application programs access kernal's features. These libraries implement most of the functionalities of the operating system.
3. Special Utility - System Utility programs are responsible to do specialised, individual level tasks.

Kernal Mode vs User Mode

Kernal components code executes in a special mode called Kernal mode with full access to all resources of the computer. This code represents a single process, executes in a single address space and is very efficient.

User programs which has no access to system hardware and kernal code works in user mode. User programs uses system libraries to access kernal function.

Architecture

1. Hardware layer - Hardware consists of all peripheral devices (RAM/HDD/CPU).
2. Kernal - It is the core component of operating system,

interact directly with hardware and provides low level services to upper layer components.

3. Shell - An interface to kernal, hiding complexity of kernal's function from users.

4. Utilities - Programs that provide the user most of the functionality of an operating system.

Basic Features -

1. Portable
2. Open source
3. Multi User
4. Multi programming
5. Hierarchical File system
6. Secure

Linux Commands -

1. ls - The ls command is used to view the contents of a directory. By default, it shows contents of current working directory.
   syntax - $ ls [options] [file/dir]
   Commands - $ ls

2. pwd - The print working directory in the linux command to get the current working directory path.
   Syntax - $ pwd
   Commands - $ pwd

3. ls -l - The ls -l option flag lists all the files and folders of a directory with long listing format. It also gives details about read, write and execute rights of the files.
   syntax - $ ls -l [dir/file]
   command - $ ls -l

4. Cat > - This linux command is used to create a new file or combine 2 files into a new file
   Syntax - $ cat > [filename] [option]
   Command - $ cat > abc.txt

5. Cat - The cat command is used to display the contents of the file in terminal

Syntax - $cat [options] filename

Command - $cat abc.txt.

6. Edit - This command is used to edit the contents of existing file.

Syntax - $edit [options] filename

Command - $ edit abc.txt.

7. Chmod - The command is used to change the read, write and execute permissions of a file. It has fixed numbers 4-read, 2-write, 1-execute These numbers are added to give regular permissions.

Syntax - $ chmod [options] octal.code filename

Command - $chmod 774 abc.txt.

8. chmod - The command is used to change the file permissions of user, group and others by specifying their rights.

Syntax - $ chmod [options] reference = rwx filename

Command - $ chmod u=rwx, g=rx, o=r abc.txt.

9. chown - This command is used to change the ownership of a file or directory.

Syntax - $ chown user filename.

Command - $ chown user2 abc.txt.

10. SU - The switch user command is used to switch the current user with another user on the system.

Syntax - $ su [options] user

Command - $ su user2.

11. mkdir - The command is used to create a new directory in the given path. If path is not specified, it creates a new directory in the current working directory.

Syntax - $ mkdir [options] path/filename.

Command - $ mkdir test.txt.

12. Cd - This command is used to change the current working directory to the directory specified but the specified directory should be present inside the current working directory.
Syntax - $ cd [options] directory
Command - $ cd music.

13. rm - r - This command is used to delete directories and contents within them. We use - r if we want to delete the directory only.
syntax - $ rm - r directory.
Command - $ rm - r music.

14. who - It is used to get information about currently logged in user on system
syntax - $ who [options]
command - $ who

15. copy - The copy command is used to copy files from current directory to different directory.
Syntax - $ copy filename directory.
Command - $ copy abc.txt Music.

16. date - The date command is used to display the system date and time. It can also be used to change the system date and time.
syntax - $ date [options]
command - $ date.

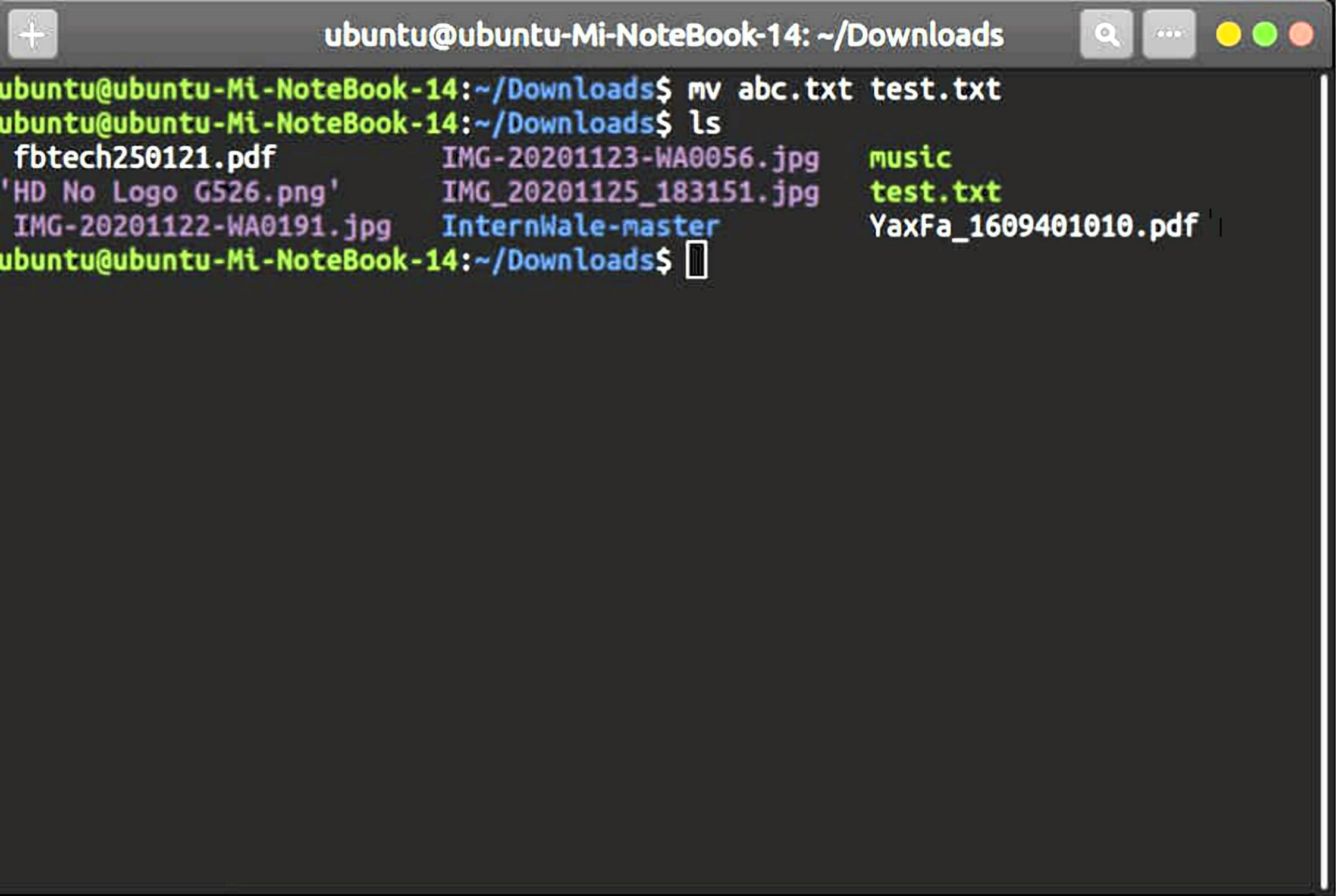
17. uname - It will print detailed information about linux system like machine name, operating system, kernal and so on.
Syntax - $ uname [options]
Command - $ uname.

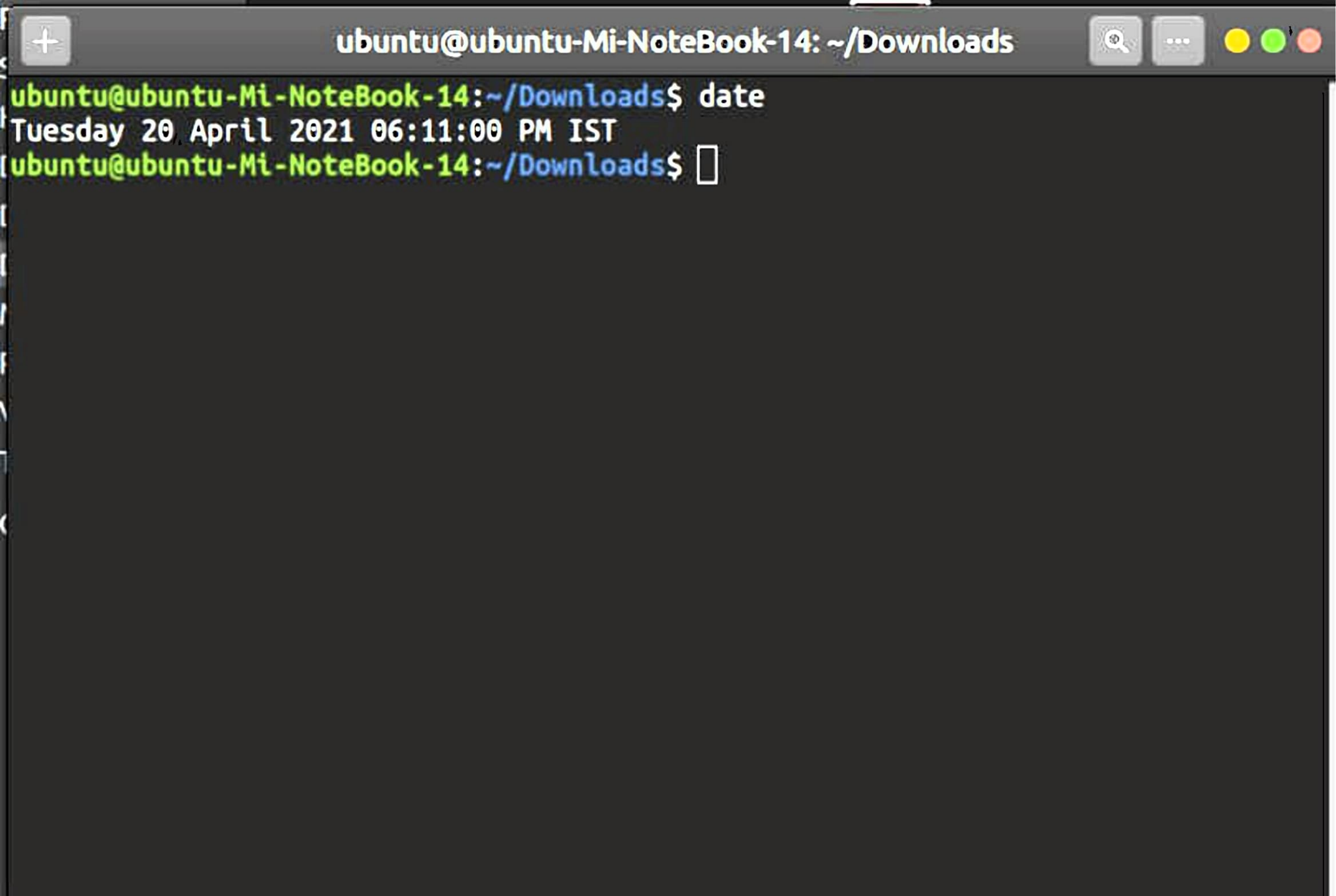
18. chgrp - The command is used to change group ownership of a file
syntax - $ chgrp [options] groupfile
Command - $ chgrp MyFile abc.txt.

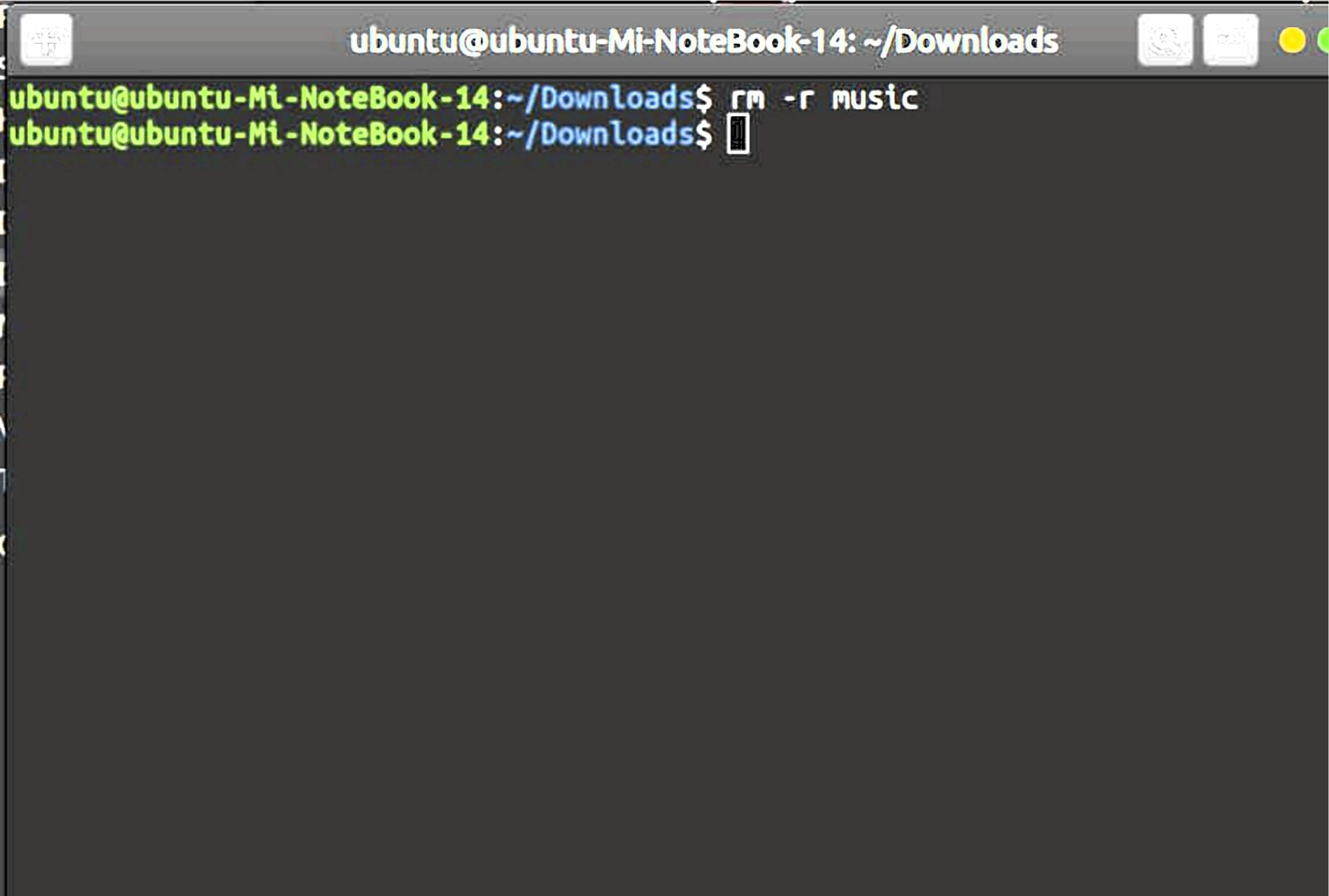
19. wc - The word count command is used to find no. of lines, words and character count in specified file.
syntax - $ wc [options] filename.
command - $ wc abc.txt.

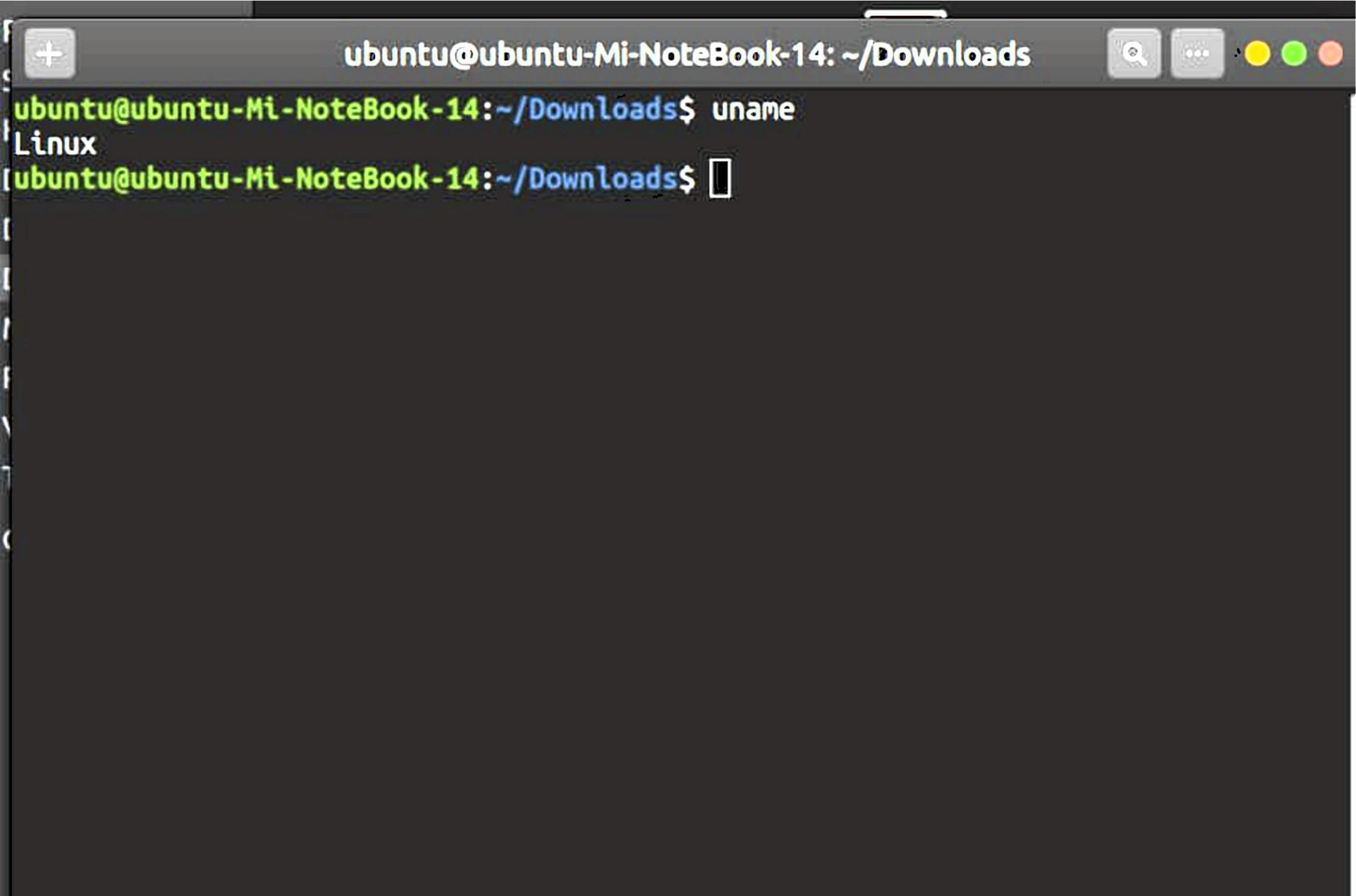
20. mall - The mv command is used to move one or more files from one place to another in a file system.
syntax - $ mv [options] source destination.
command - $ mv abc.txt test.txt.

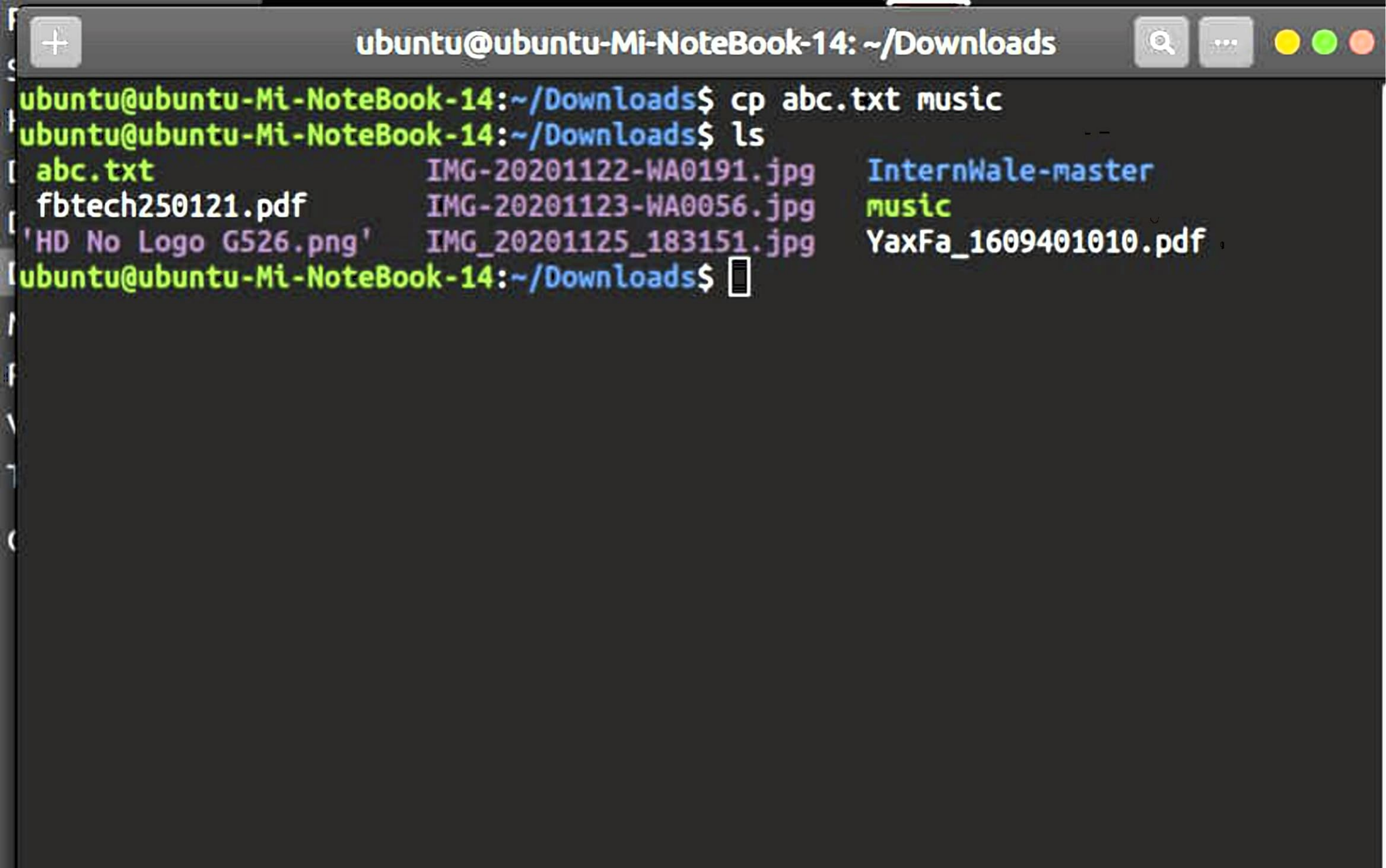

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ who
ubuntu :0 2021-04-20 23:24 (:0)
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

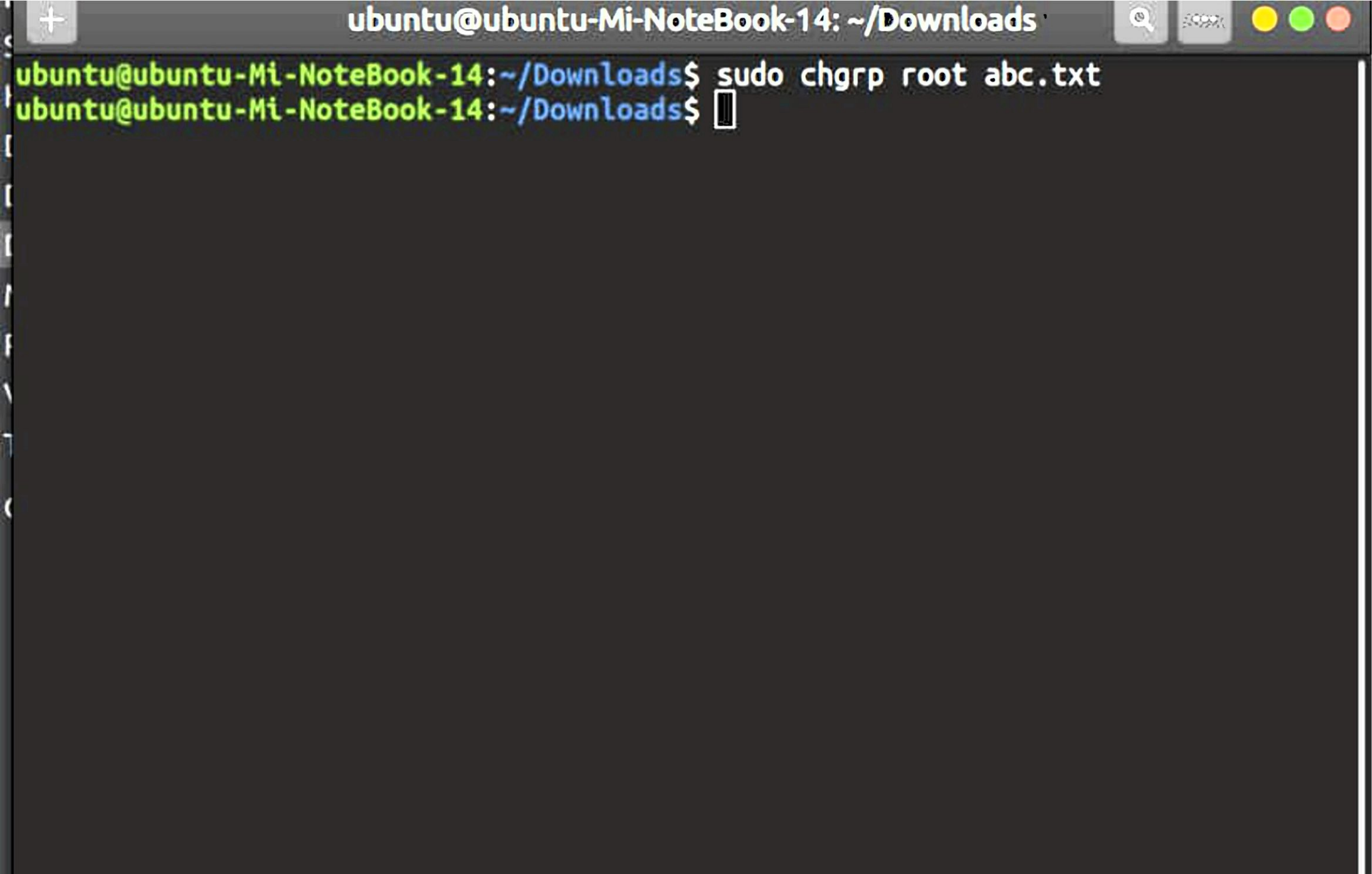

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ mv abc.txt test.txt
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ ls
fbtech250121.pdf
'HD No Logo G526.png'
IMG-20201122-WA0191.jpg
IMG-20201123-WA0056.jpg
IMG_20201125_183151.jpg
InternWale-master
music
test.txt
YaxFa_1609401010.pdf
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$


ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ date
Tuesday 20 April 2021 06:11:00 PM IST
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

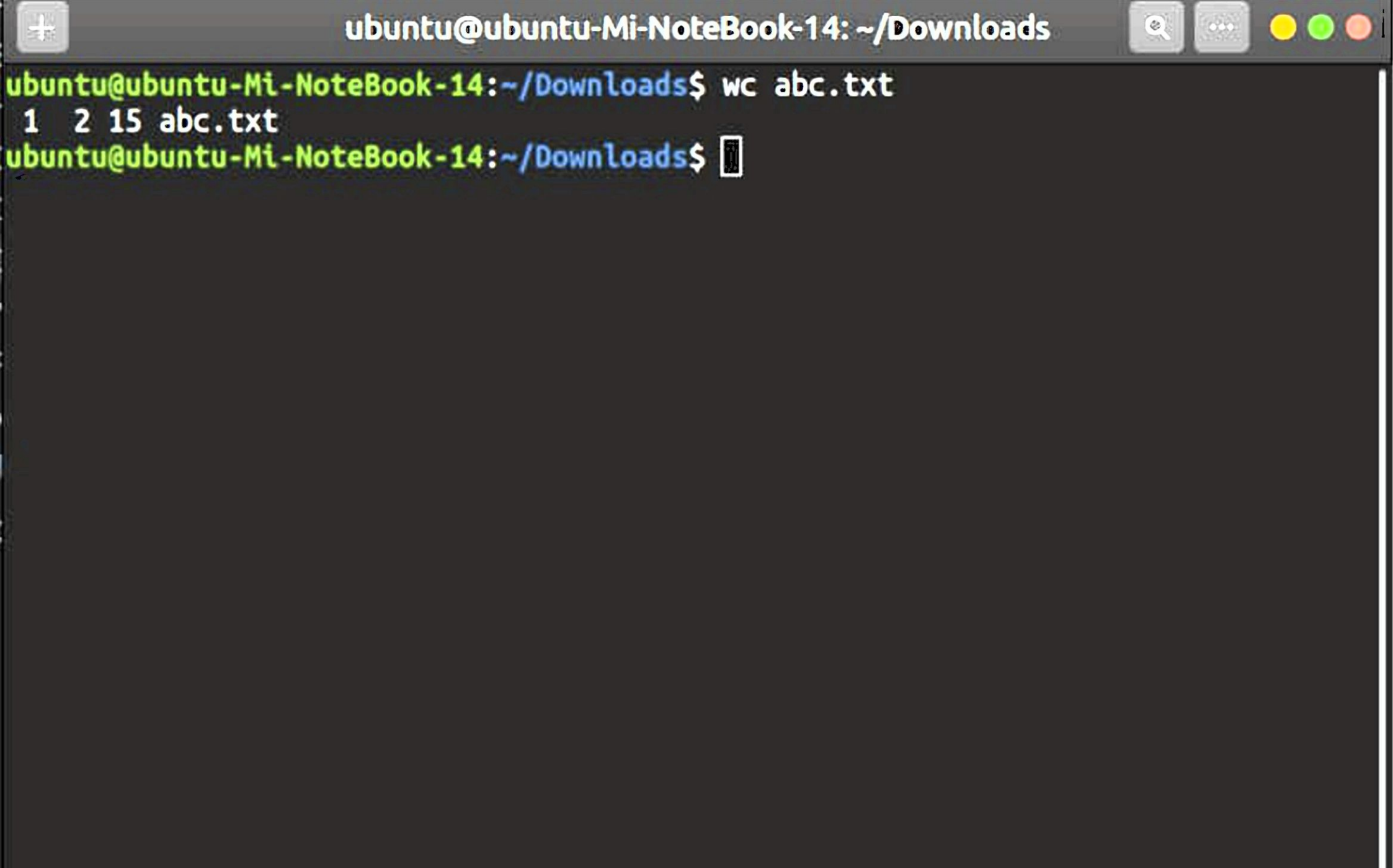

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ rm -r music
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

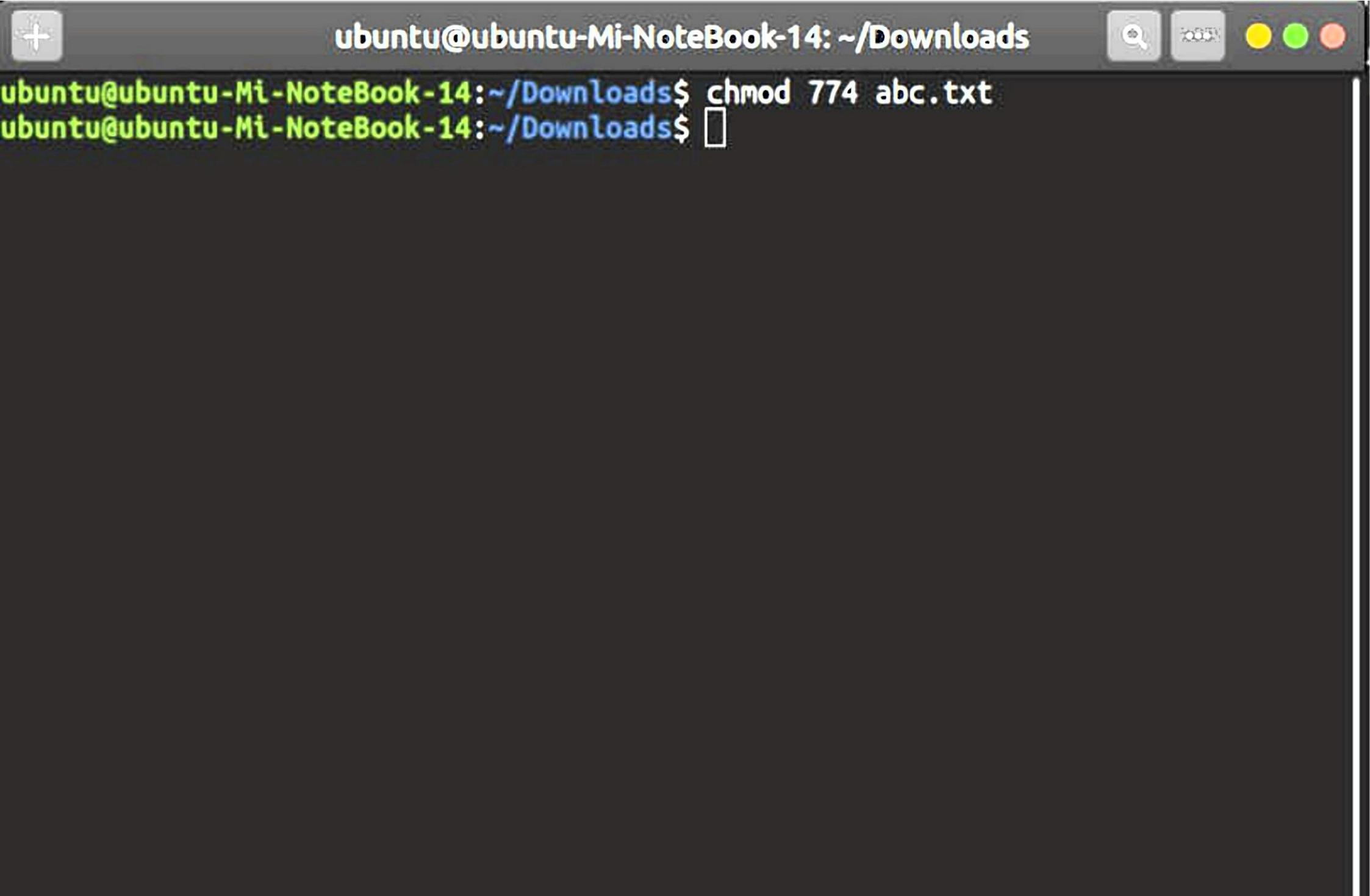

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ uname
Linux
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

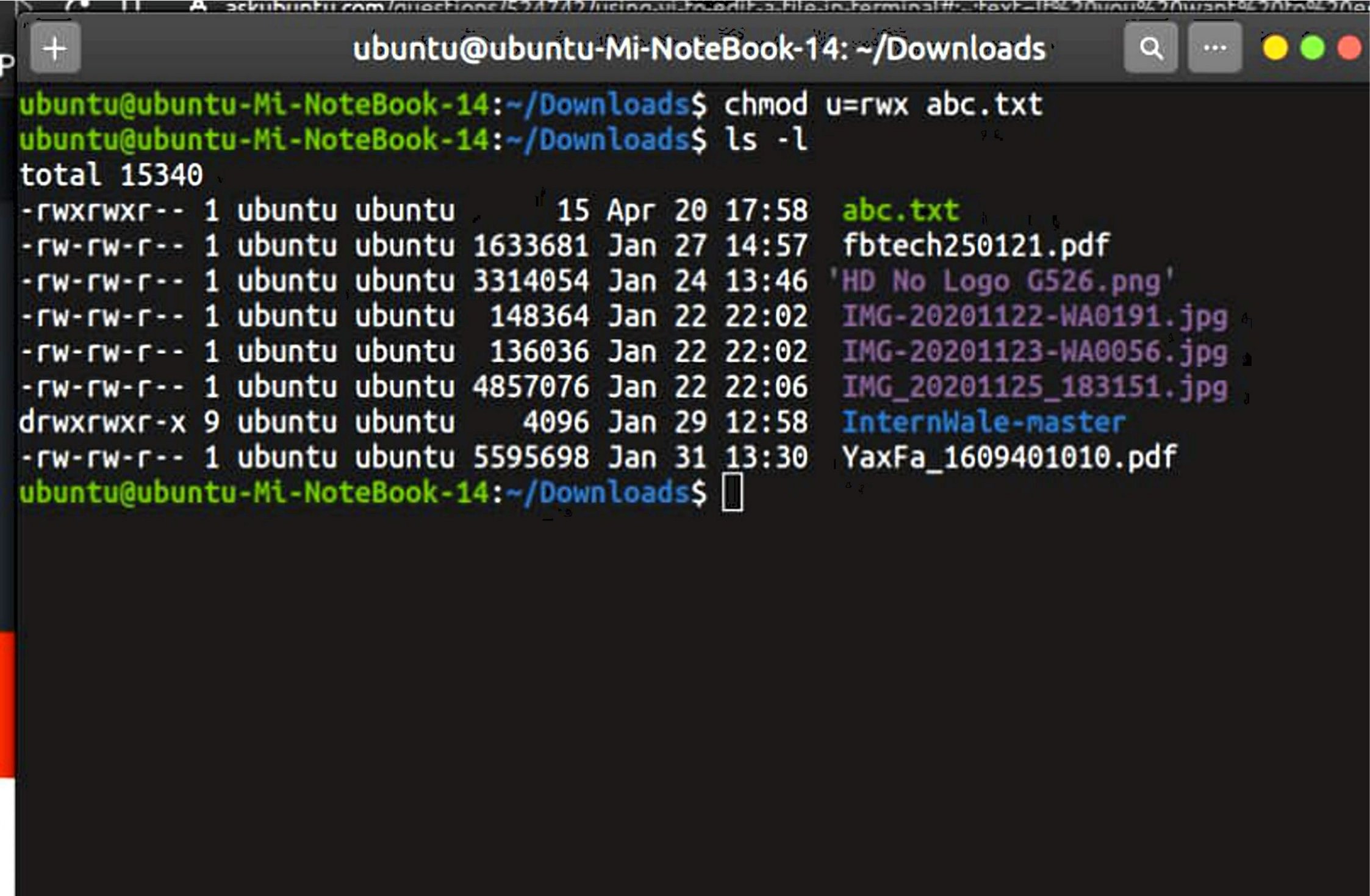

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ cp abc.txt music
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ ls
abc.txt
fbtech250121.pdf
'HD No Logo G526.png'
IMG-20201122-WA0191.jpg
IMG-20201123-WA0056.jpg
IMG_20201125_183151.jpg
InternWale-master
music
YaxFa_1609401010.pdf
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

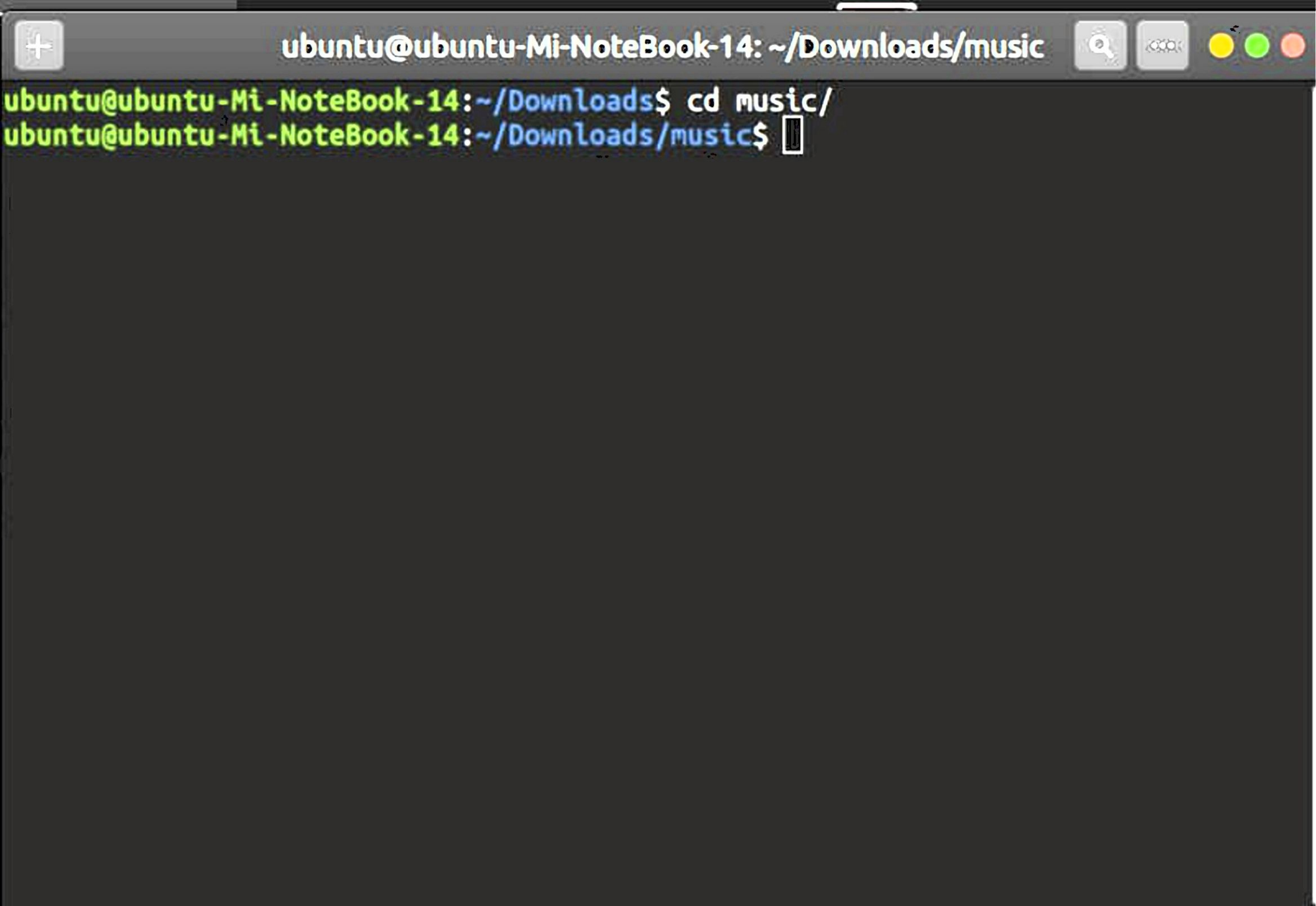

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ sudo chgrp root abc.txt
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

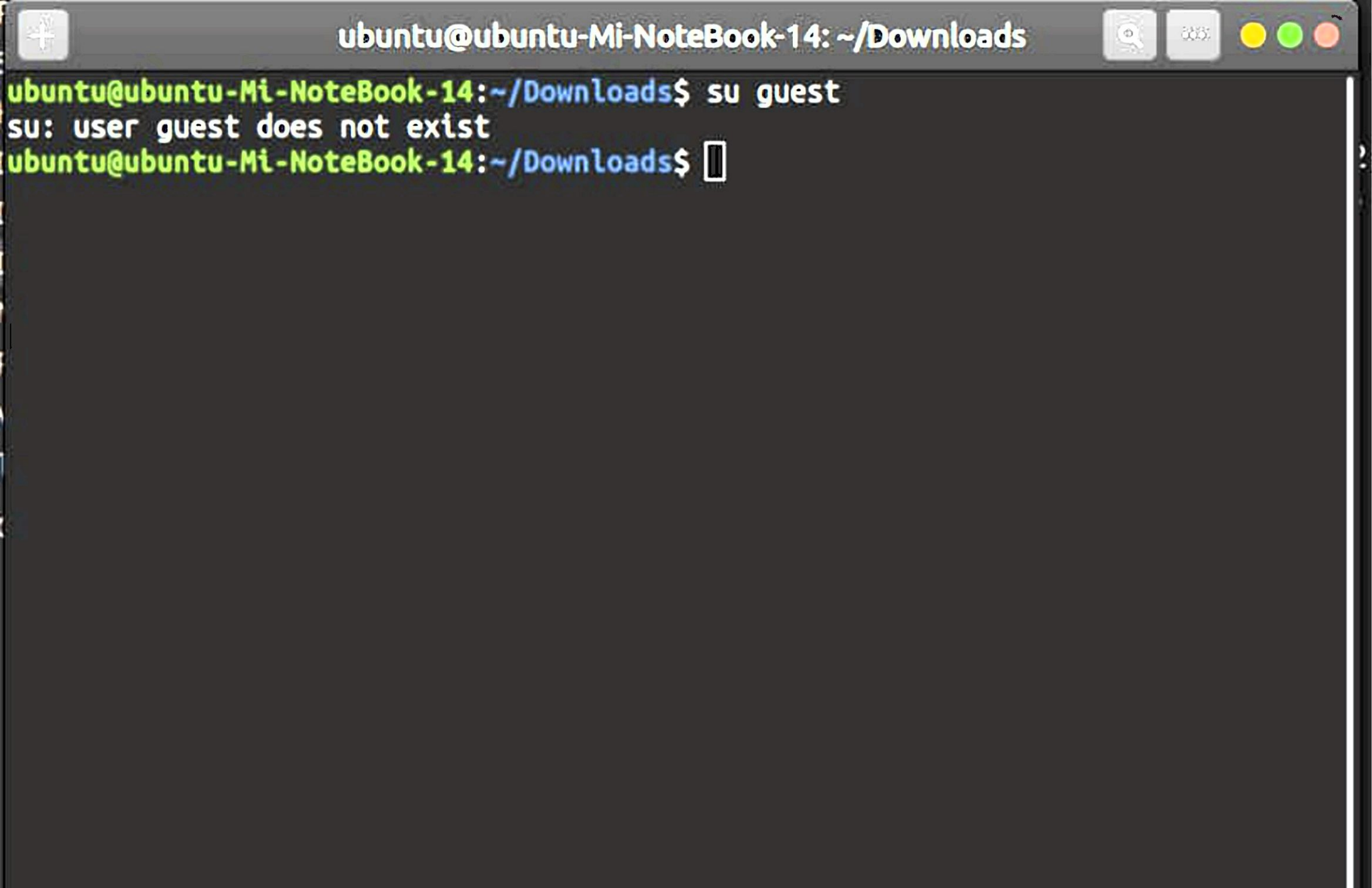

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ cat abc.txt
sarthak bhatia
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

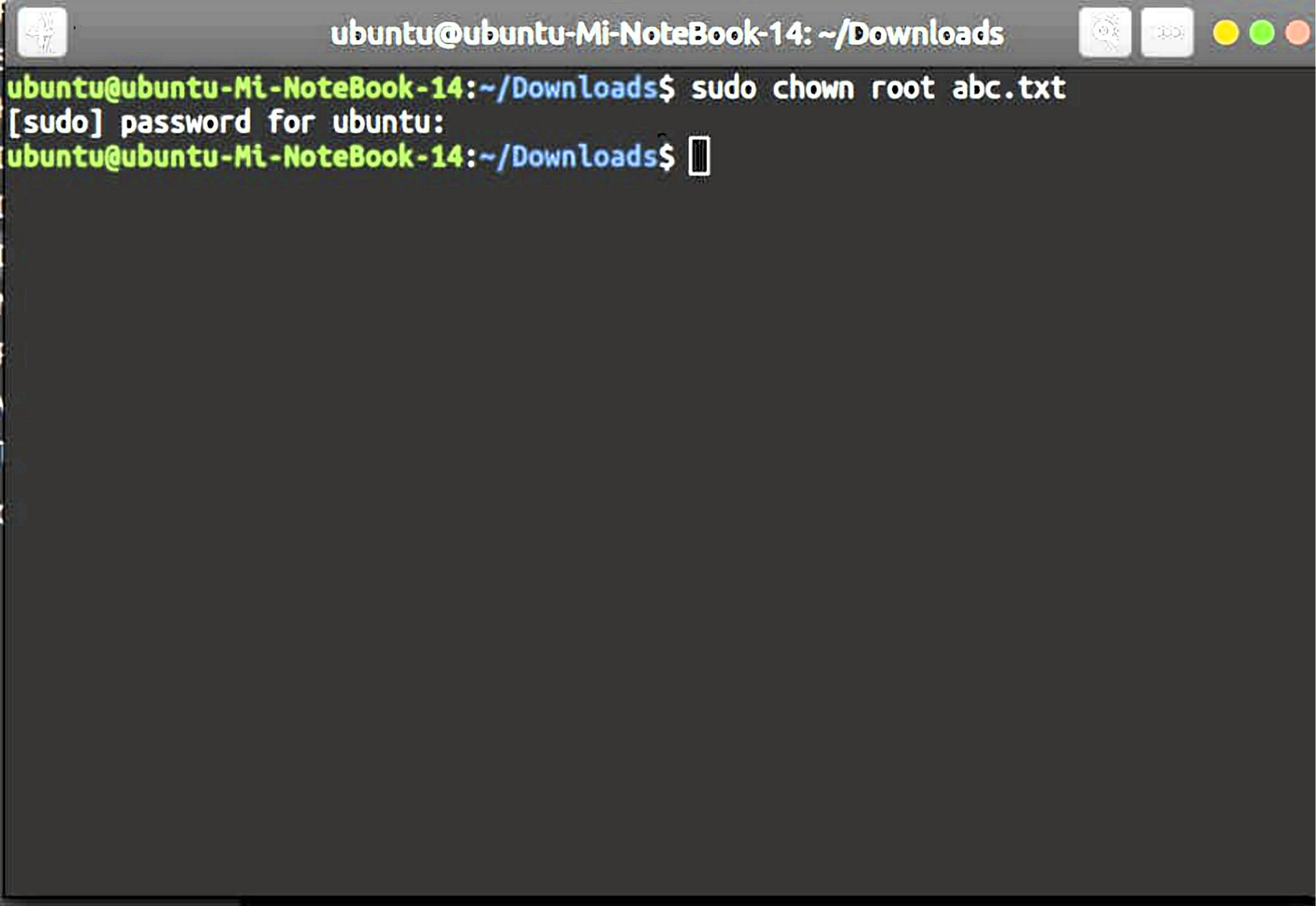

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ wc abc.txt
1 2 15 abc.txt
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

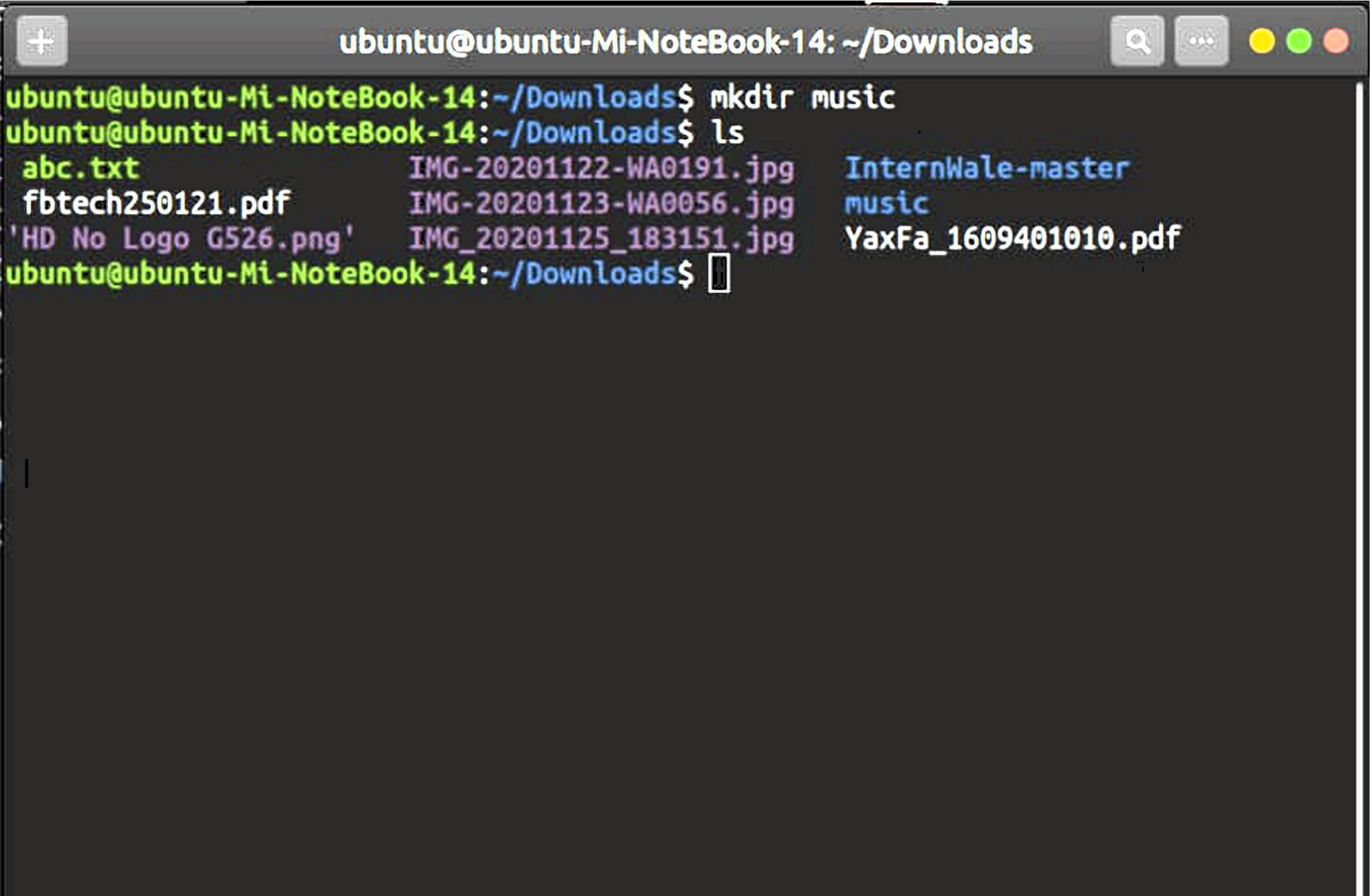

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ chmod 774 abc.txt
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

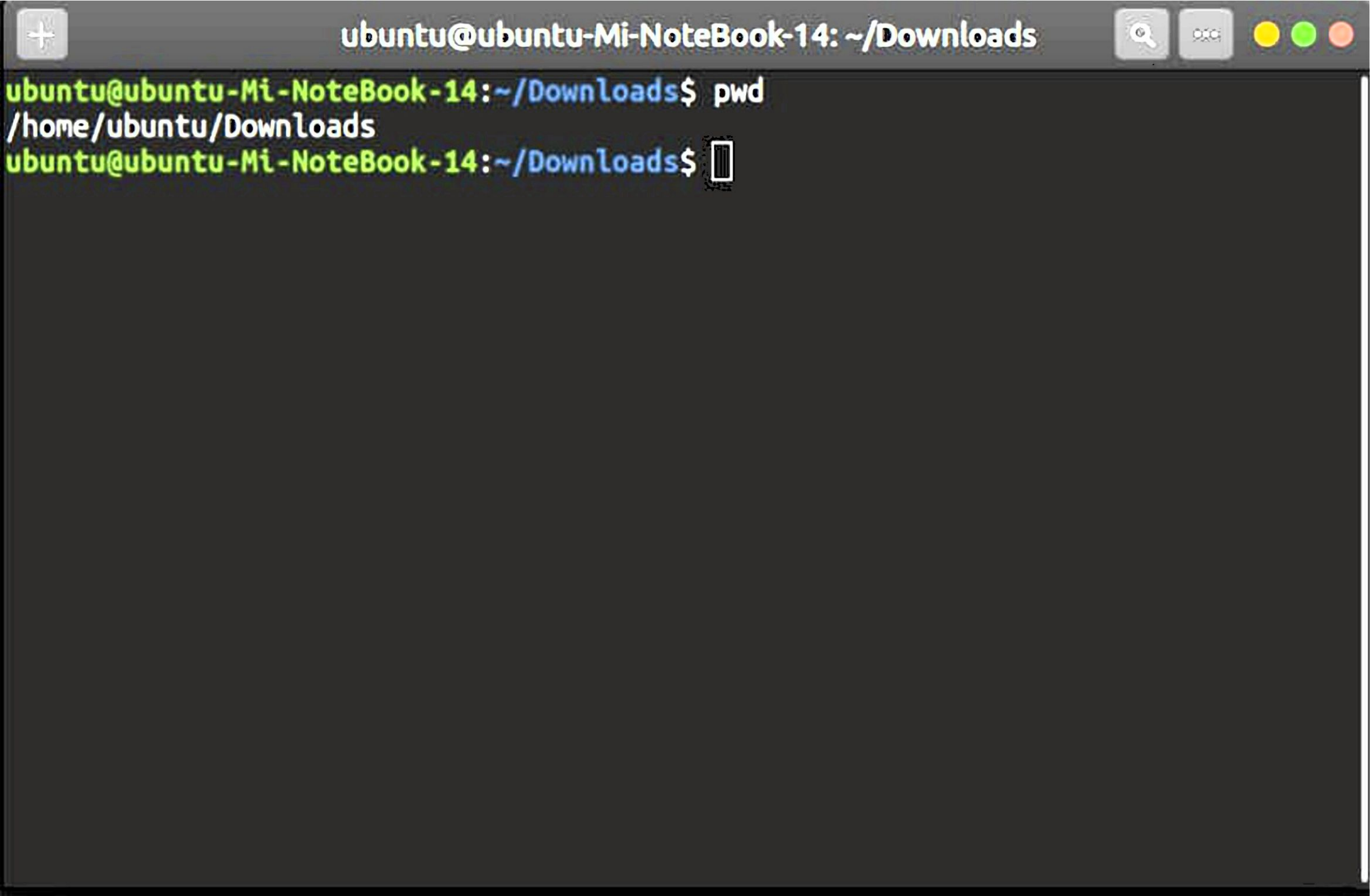

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ chmod u=rwx abc.txt
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ ls -l
total 15340
-rwxrwxr-- 1 ubuntu ubuntu 15 Apr 20 17:58 abc.txt
-rw-rw-r-- 1 ubuntu ubuntu 1633681 Jan 27 14:57 fbtech250121.pdf
-rw-rw-r-- 1 ubuntu ubuntu 3314054 Jan 24 13:46 'HD No Logo G526.png'
-rw-rw-r-- 1 ubuntu ubuntu 148364 Jan 22 22:02 IMG-20201122-WA0191.jpg
-rw-rw-r-- 1 ubuntu ubuntu 136036 Jan 22 22:02 IMG-20201123-WA0056.jpg
-rw-rw-r-- 1 ubuntu ubuntu 4857076 Jan 22 22:06 IMG_20201125_183151.jpg
drwxrwxr-x 9 ubuntu ubuntu 4096 Jan 29 12:58 InternWale-master
-rw-rw-r-- 1 ubuntu ubuntu 5595698 Jan 31 13:30 YaxFa_1609401010.pdf
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

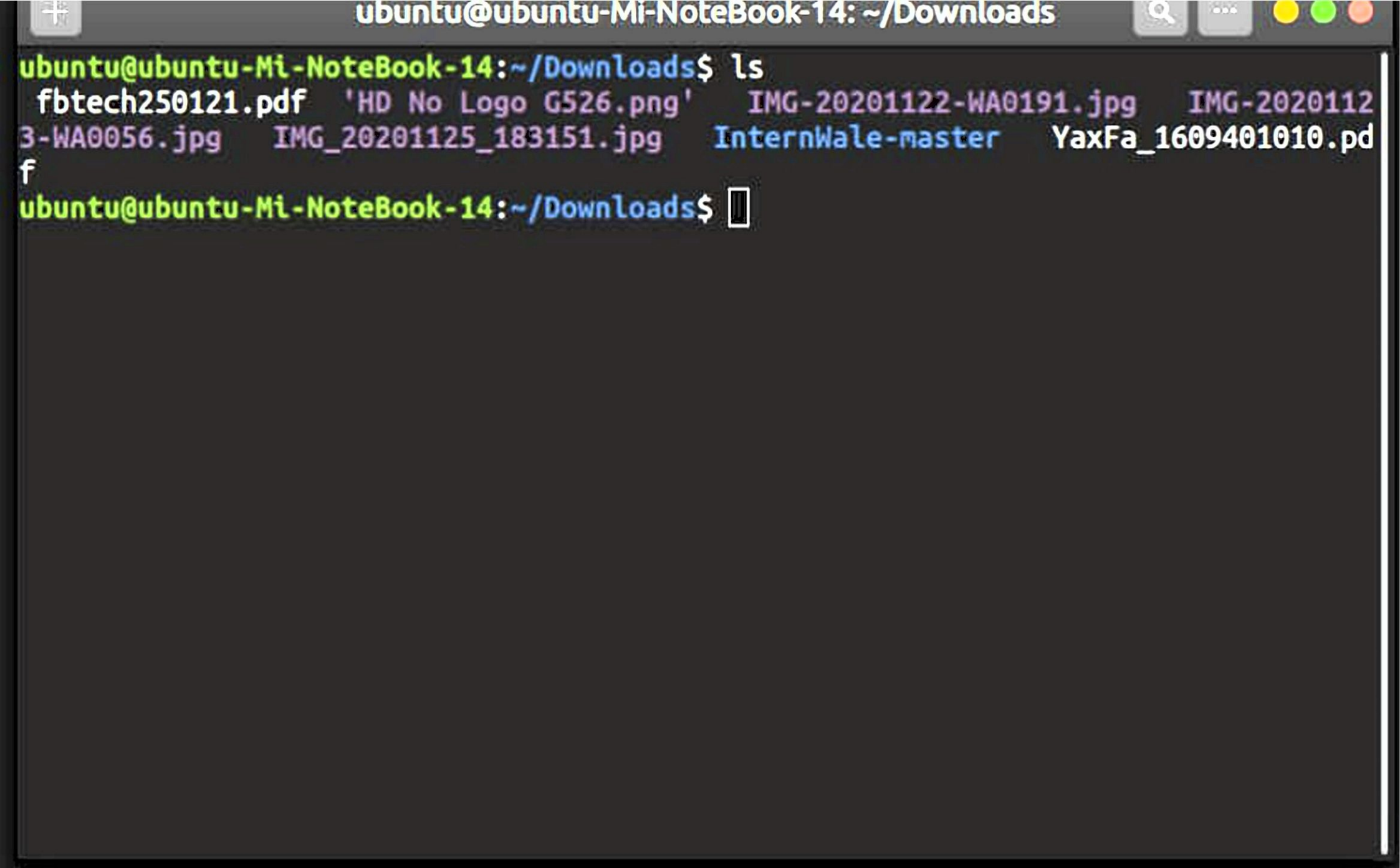

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads/music
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ cd music/
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads/music$

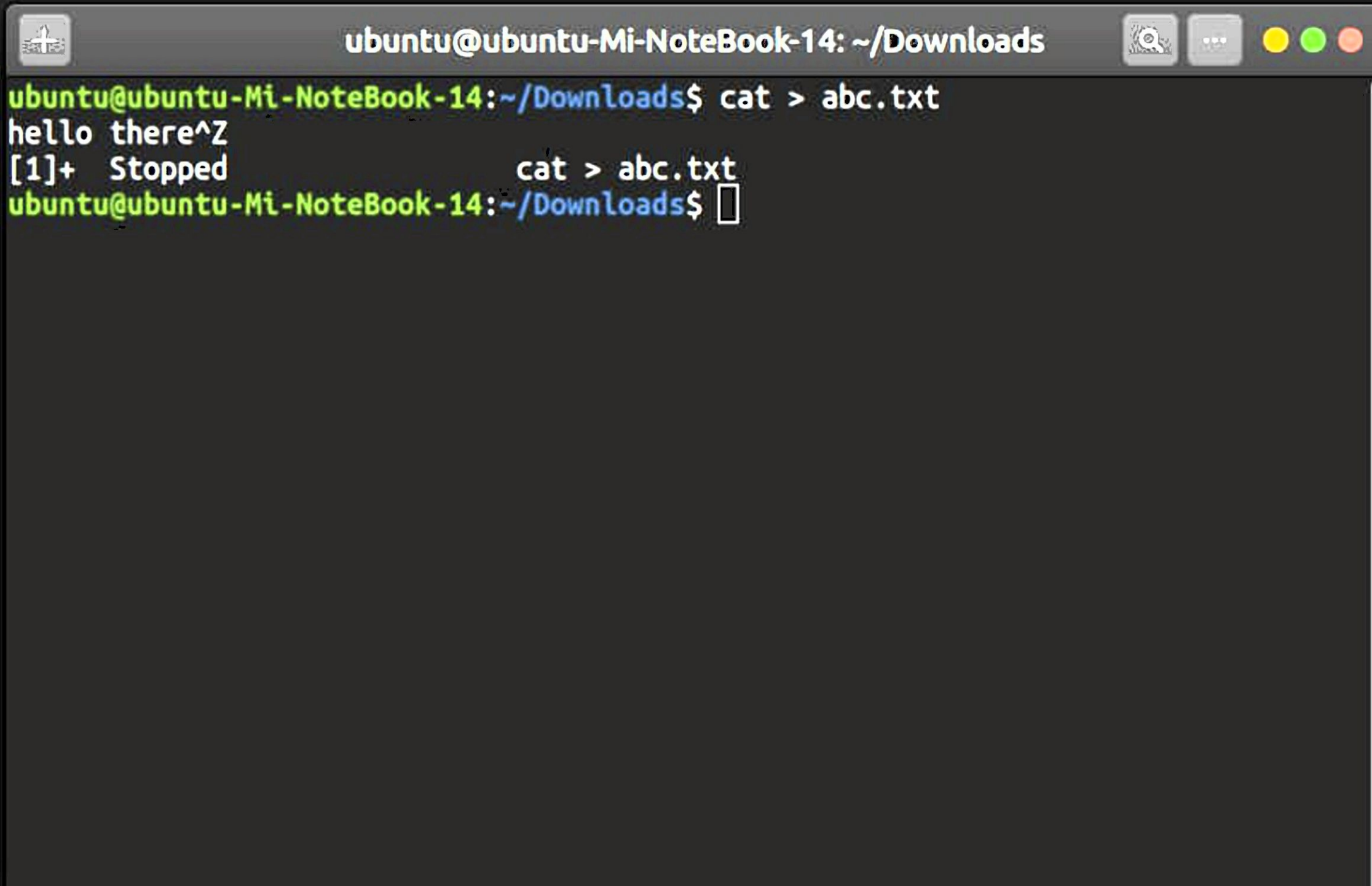

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ su guest
su: user guest does not exist
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

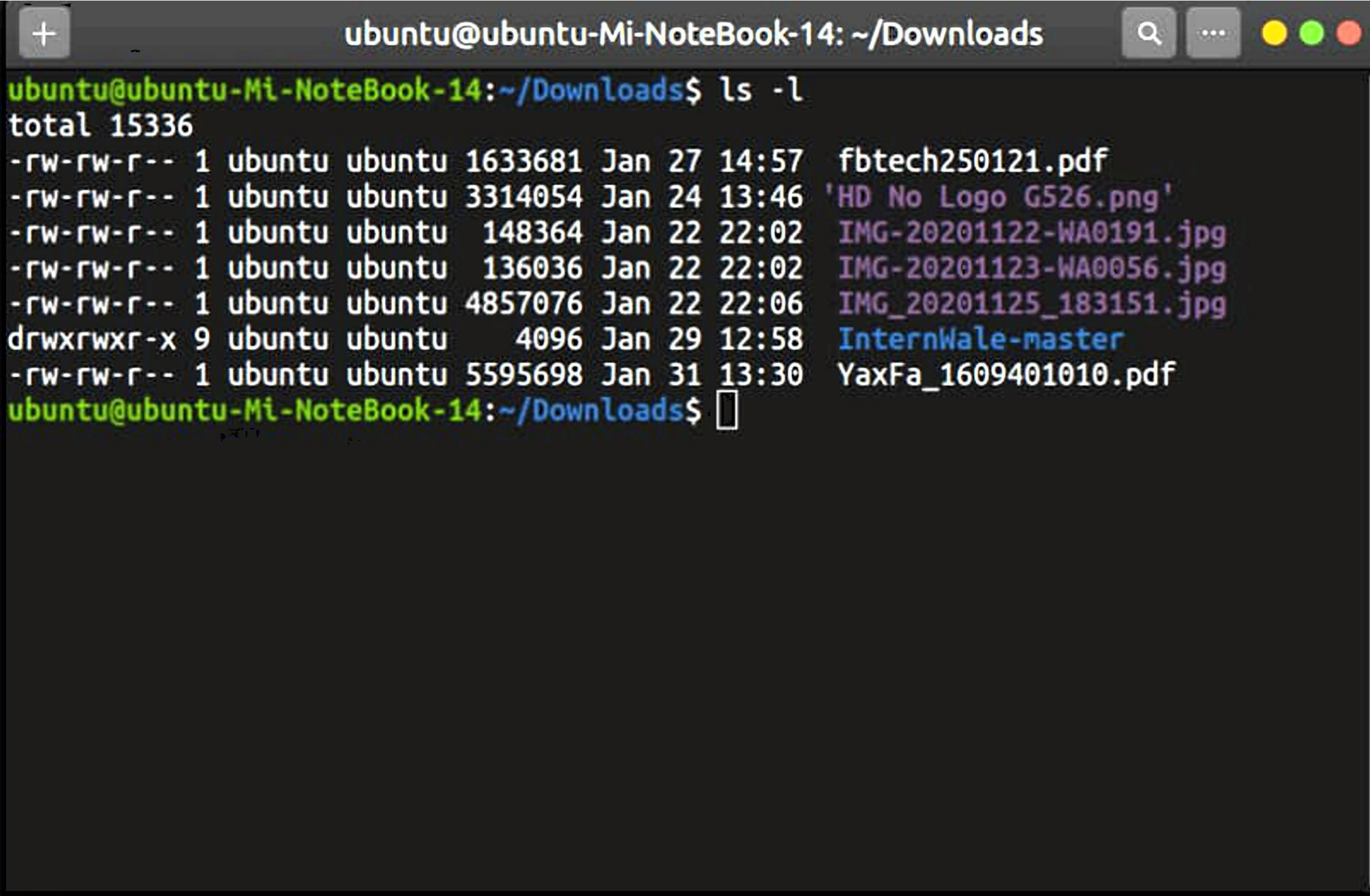

ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ sudo chown root abc.txt
[sudo] password for ubuntu:
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$


ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ mkdir music
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ ls
abc.txt
fbtech250121.pdf
'HD No Logo G526.png'
IMG-20201122-WA0191.jpg
IMG-20201123-WA0056.jpg
IMG_20201125_183151.jpg
InternWale-master
music
YaxFa_1609401010.pdf
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$


ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ pwd
/home/ubuntu/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$


ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ ls
fbtech250121.pdf 'HD No Logo G526.png' IMG-20201122-WA0191.jpg IMG-20201123-WA0056.jpg IMG_20201125_183151.jpg InternWale-master YaxFa_1609401010.pdf
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$


ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ cat > abc.txt
hello there^Z
[1]+ Stopped cat > abc.txt
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$


ubuntu@ubuntu-Mi-NoteBook-14: ~/Downloads
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$ ls -l
total 15336
-rw-rw-r-- 1 ubuntu ubuntu 1633681 Jan 27 14:57 fbtech250121.pdf
-rw-rw-r-- 1 ubuntu ubuntu 3314054 Jan 24 13:46 'HD No Logo G526.png'
-rw-rw-r-- 1 ubuntu ubuntu 148364 Jan 22 22:02 IMG-20201122-WA0191.jpg
-rw-rw-r-- 1 ubuntu ubuntu 136036 Jan 22 22:02 IMG-20201123-WA0056.jpg
-rw-rw-r-- 1 ubuntu ubuntu 4857076 Jan 22 22:06 IMG_20201125_183151.jpg
drwxrwxr-x 9 ubuntu ubuntu 4096 Jan 29 12:58 InternWale-master
-rw-rw-r-- 1 ubuntu ubuntu 5595698 Jan 31 13:30 YaxFa_1609401010.pdf
ubuntu@ubuntu-Mi-NoteBook-14:~/Downloads$

EXPERIMENT-2 Date - 22/3/21

- **AIM** - Write a program to implement CPU scheduling for First come First serve Algorithm.
- **THEORY** - First come First serve is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest scheduling algorithm. In this, the processes which request the CPU first, gets the CPU allocation first. This is managed with the help of a queue. The algorithm supports non-preemptive scheduling. The method is poor in performance and the average waiting time is quite high.
- **Code** :

```
#include <iostream>
using namespace std;
int main(){
    int n;
    cin >> n;
    int task[n];
    int burstTime[n];
    int start[n];
    for(int i=0; i<n; i++){
        cin >> task[i];
        cin >> burstTime[i];
        cin >> start[i];
    }
    int total=0;
    int waitTime[n];
    int turnAround[n];
    cout << "Name - Sarthak Bhatia" << endl << "Rollnumber - 0771CS203118" << endl << "output" << endl.
```

```
for (int i=0; i<n; i++){
    waitTime[i] = total - start[i];
    total += burstTime[i];
    turnAround[i] = total - start[i];
    cout << "waiting time for task" << task[i] << " is
      << waitTime[i] << endl;
    cout << "turn aground time for task" << task[i]
      << " is " << turnAround[i] << endl;
  }
  return 0;
}
```

Output:

```
Name- sarthak bhatia
Roll number- 07714803118
    output
waiting time for task 1 is 0
turn around time for task 1 is 5
waiting time for task 2 is 4
turn around time for task 2 is 19
waiting time for task 3 is 18
turn around time for task 3 is 28
```

EXPERIMENT-2 Date - 19/4/21

- AIM - Write a program to implement CPU scheduling for shortest job first.

A) Non-Preemptive Shortest Job First

- THEORY - Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

  In non-preemptive scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches a waiting state or terminated.

- PROGRAM -

```
#include <iostream>
using namespace std;
int mat[10][6];

void swap(int* a, int* b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
void arrangeArrival(int num, int mat[][6]){
    for(int i=0; i<num; i++){
        for(int j=0; j<num-i-1; j++){
            if(mat[j][1] > mat[j+1][1]){
                for(int k=0; k<5; k++){
                    swap(mat[j][k], mat[j+1][k]);
                }
            }
        }
    }
}
```

```
void completionTime(int num, int mat[][6]) {
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];
    for(int i=1; i<num; i++) {
        temp = mat[i-1][3];
        int low = mat[i][2];
        for(int j=i; j<num; j++) {
            if(temp >= mat[j][1] && low >= mat[j][2]){
                low = mat[j][2];
                val = j;
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for(int k=0; k<6; k++){
            swap(mat[val][k], mat[i][k]);
        }
    }
}
int main() {
    int num, temp;
    cout << "Name - Sarthak Bhatia\n Roll No - 0771480318";
    cout << "Enter number of processes:";
    cin >> num;
    for(int i=0; i<num; i++){
        cout << "Process" << i+1 << "\n";
        cout << "Enter Process Id: ";
        cin >> mat[i][0];
```

```
        cout << "Enter arrival Time: ";
        cin >> mat[i][1];
        cout << "Enter Burst Time: ";
        cin >> mat[i][2];
    }

arrangeArrival(num, mat);
completionTime(num, mat);
cout << "Result - \n";
cout << "ProcessID \t Arrival Time \n Burst Time
 \t Waiting Time \t Turnaround Time \n";
for(int i=0; i< num; i++){
    cout << mat[i][0] << "\t\t" << mat[i][1] <<
    "\t\t" << mat[i][2] << "\t\t" << mat[i][4]
    << "\t\t" << mat[i][5] << "\n";
}
}
```

- OUTPUT -

```
Name-Sarthak Bhatia
Roll no-07714803118
Enter number of Process: 4
Process 1
Enter Process Id: 1
Enter Arrival Time: 2
Enter Burst Time: 3
Process 2
Enter Process Id: 2
Enter Arrival Time: 0
Enter Burst Time: 4
Process 3
Enter Process Id: 3
Enter Arrival Time: 4
Enter Burst Time: 2
Process 4
Enter Process Id: 4
Enter Arrival Time: 5
Enter Burst Time: 4
Result-
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
2               0               4               0               4
3               4               2               0               2
1               2               3               4               7
4               5               4               4               8
```

B) Preemptive Shortest Job First

- THEORY - In preeptive SJF scheduling, jobs are put into the ready queue as they come. A process with shortest burst time begins execution. If a process with even a shorter time arrives, the current process is removed or preemptive from execution, and shorter Job is allocated CPU cycle.
- PROGRAM -

```
#include <bits/stdc++.h>
using namespace std;
void WaitingTime(int mat[][3], int n, int wt[]){
    int rt[n];
    for(int i=0; i<n; i++){
        rt[i] = mat[i][1];
    }
    int complete=0, t=0, minm = INT_MAX;
    int shortest=0, finish_time;
    bool check = false;
    while(complete!=n){
        for(int j=0; j<n; j++){
            if((mat[j][2] <= t) && (rt[j]<minm) && rt[j]>0){
                minm = rt[j]
                shortest = j
                check = true;
            }
        }
        if(check == false){
            t++;
            continue;
        }
        rt[shortest]--;
        minm = rt[shortest];
```

```
if(minm == 0)
    minm = INT_MAX;
if (rt[shortest]==0) {
    complete++;
    check = false;
    finish_time = t+1;
    wt[shortest] = finish_time - mat[shortest][1] - mat[shorte
                                                        [2
    if (wt[shortest] < 0)
        wt[shortest] = 0;
  }
  t++;
 }
}
void TurnaroundTime(int mat[][3], int n, int wt[], int tat[]){
    for(int i=0; i<n; i++){
      tat[i] = mat[i][1] + wt[i];
    }
}
void AvgTime(int mat[][3], int n){
  int wt[n], tat[n], total_wt=0, total_tat=0;
   WaitingTime(mat, n, wt);
   TurnaroundTime(mat, n, wt, tat);
   cout << endl;
   cout << "Result - \n";
   cout << "Process" << "Burst Time" << "waiting
   Time" << "Turn Around Time \n";
   for(int i=0; i<n; i++){
       total_wt = total_wt + wt[i];
       total_tat = total_tat + tat[i];
```

```
cout << " " mat[i][0] << " \t \t" << mat[i][1] << " \t \t
<< wt[i] << " \t \t" << tat[i] << endl;
}

cout << "\n Average waiting Time = " << (float)total_wt/n;
cout << "\n Average turn Around Time = " << (float) Total_tat/n;
}

int main() {
    int num, temp;
    cout << "Name - Sarthak Bhatia \n Roll No - 0771480 3118";
    cout << "Enter number of Processes: ";
    cin >> num;
    int mat[num][3];
    for(int i=0; i<num, i++){
      cout << "Process " << i+1 << " \n";
      cout << "Enter Process ID:";
       cin >> mat[i][0];
       cout << "Enter Burst Time:";
       cin >> mat[i][1];
       cout << "Enter Arrival Time:";
       cin >> mat[i][2];
      }
     AvgTime(mat, num);
      return 0;
}
```

- OUTPUT-

```
Name-Sarthak Bhatia
Roll no-07714803118
Enter number of Process: 4
Process 1
Enter Process Id: 1
Enter Burst Time: 6
Enter Arrival Time: 1
Process 2
Enter Process Id: 2
Enter Burst Time: 8
Enter Arrival Time: 1
Process 3
Enter Process Id: 3
Enter Burst Time: 7
Enter Arrival Time: 2
Process 4
Enter Process Id: 4
Enter Burst Time: 3
Enter Arrival Time: 3

Result-
Processes  Burst time  Waiting time  Turn around time
 1              6              3              9
 2              8              16             24
 3              7              8              15
 4              3              0              3

Average waiting time = 6.75
Average turn around time = 12.75
```

EXPERIMENT- 4 Date - 26/04/21

- AIM- To implement CPU scheduling for Round Robin.
- THEORY - Round-Robin is one of the algorithms employed by process and network schedulers in computing. As the term generally used, time slices are assigned to each process in equal portions and in circular order, handling all processes without priority. Round-Robin scheduling is simple, easy to implement and starvation-free. Round-Robin scheduling can be applied to other scheduling problems, such as data packet scheduling in computer networks. The name of the algorithm comes from the round-robin principle known from other fields, where each person takes an equal share of something in turn.
- PROGRAM-

```
# include <iostream>
  using namespace std;

  void waitingTime(int process [], int n, int burstTime [],
   int [] wt , int timePeriod) {

      int remBurstTime [n];
      for (int i=0; i<n; i++) {
        remBurstTime[i] = burstTime[i];
      }
      int t = 0;
      while (true) {
          bool done = true;
          for (int i=0; i<n; i++) {
            if (remBurstTime [i] >0) {
                done = false;
```

```
        if (remBurstTime[i] > timePeriod) {
            t+ = timePeriod;
            remBurstTime[i] -= timePeriod;
        }
        else {
            t = t + remBurstTime[i];
            t+ = timePeriod;
            wt[i] = t - burstTime[i];
            remBurstTime = 0;
        }
    }
  }
  if (done == true) break;
  }
}
void turnAroundTime(int [] process, int n, int burstTime [], int
    wt [], int tat[]) {
      for(int i=0; i < n; i++) {
          tat[i] = burstTime[i] + wt[i];
      }
}
void solve (int process[], int n, int burstTime[], int timePeriod) {
    int wt[n], tat[n], totalWt=0, totalTAT=0;
    waitingTime(process, n, burstTime, wt, timePeriod);
    turnAroundTime(process, n, burstTime, wt, tat) &;
    cout << "Process " << " BurstTime " << " Waiting
    Time " << " Turn Around Time \n";
```

```
for(int i=0; i<n; i++){
    totalWT += wt[i];
    totalTAT += tat[i];
    Cout << "  (i+1) " << "\t\t" << burstTime[i] <<
    "\t\t" << wt[i] << "\t\t" << tat[i] << endl;
}
Cout << "Average waiting Time = " << (float) totalWT /n;
Cout << "Average TurnAround Time = " << (float) totalTAT/n;
}
int main(){
    int n;
    Cout << "Name - Sarthak Bhatia \n Roll No. - 0771480318"
    Cout << "Enter Number of Processes - ";
    cin >> n;
    int process[n];
    int burstTime[n];
    for(int i=0; i<n; i++){
        Cout << "Process " << i+1 << "\n";
        Cout << "Enter process ID: ";
        cin >> process[i];
        Cout << "Enter Burst time: ";
        cin >> burstTime[i];
    }
    int timePeriod=2;
    solve(process, n, burstTime, timePeriod);
    return 0;
}
```

• OUTPUT-

```
Name-Sarthak Bhatia
Roll no-07714803118
Enter number of Process: 3
Process 1
Enter Process Id: 1
Enter Burst Time: 10
Process 2
Enter Process Id: 2
Enter Burst Time: 5
Process 3
Enter Process Id: 3
Enter Burst Time: 8
Processes  Burst time  Waiting time  Turn around time
 1              10          13              23
 2              5           10              15
 3              8           13              21
Average waiting time = 12
Average turn around time = 19.6667
```

EXPERIMENT-5

- AIM - Write a program to perform Priority scheduling.
- THEORY - Priority scheduling is a non-primitive algorithm and one of the most common scheduling algorithm in batch systems. Each process is assigned first arrival time (less arrival process first) if two processes have same arrival time, then compare to priorities (highest process first). Also, if two processes have same priority then compare to process number (less process number first). This process is repeated while all process get executed.
- PROGRAM -

```
#include<iostream>
using namespace std;
int main(){
 cout<<" Name - Sarthak Bhatia \n Roll No. - 07714803118";
 int a[10], b[10], x[10], pr[10]={0};
 int waiting[10], turnaround[10], completion[10];
 int i, j, smallest, count=0, time, n;
 double avg=0, tt=0, end;
 cout<<"\n Enter number of Processes:";
  cin>>n;
 cout<<"\n Enter arrival times of Processes:";
  for(int i=0; i<n; i++){
     cin>>a[i];
  }
  cout<<"\n Enter burst time of Processes:";
   for(int i=0; i<n; i++){
      cin>>b[i];
   }
   cout<<"\n Enter priority of Processes:";
```

```
for (int i=0; i<n; i++){
    cin>>pr[i];
}
for(int i=0; i<n; i++){
    x[i]= b[i];
}
pr[9]=-1
for(time=0; count!=n; time++){
        smallest=9;
        for( i=0; i<n; i++){
        if(a[i]<=time && pr[i]>pr[smallest] && b[i]>0)
            smallest=i;
        }
        time+= b[smallest]-1;
        b[smallest]= -1
        count++;
        end = time+1;
        completion[smallest]=end;
        waiting[smallest]=end-a[smallest]-x[smalbst];
        turnaround[smallest]=end-a[smallest];
    }

cout<<"Process"<<"\t\t"<<"burst-time"<<"\t\t"
<<"arrival-time"<<"\t\t"<<"waiting-time"<<
"\t\t"<<"turnaround-time"<<"\t\t"<<"completion-
time"<<"\t\t"<<"Priority"<<endl;
for(int i=0; i<n; i++){
cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<a[i]<<
"\t\t"<<waiting[i]<<"\t\t"<<turnaround[i]<<"\t\t"
<<completion[i]<<"\t\t"<<pr[i]<<endl;
        avg+= waiting[i];
        tt= tt+ turnaround[i];
}
cout<<"\n\n Average waiting time="<<avg;
cout<<"\n Average Turnaround time="<<tt; }
```

• OUTPUT-

```
Name- Sarthak Bhatia
Roll No.- 07714803118

Enter the number of Processes: 5

Enter arrival time of processes:
0
1
2
3
4

Enter burst time of processes:
4
3
1
5
2

Enter priority of processes:
2
3
4
5
5
Process burst-time  arrival-time    waiting-time    turnaround-time completion-time Priority
p1          4           0               0               4               4               2
p2          3           1               11              14              15              3
p3          1           2               9               10              12              4
p4          5           3               1               6               9               5
p5          2           4               5               7               11              5


Average waiting time =5.2  Average Turnaround time =8.2
```

EXPERIMENT-6

- **AIM** - Write a program to implement page replacement policy using FIFO.
- **THEORY** - In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new pages come in.

First In First Out Algorithm - This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in front of the queue. When a page needs to be replaced page in front of queue is selected for removal.

- **PROGRAM** -

```
# include <bits/stdc++.h>
using namespace std;
int main(){
  int n, m, i, j, k, hit=0;
  cout << "Enter number of Frames: \n";
  cin >> n;
  cout << "Enter number of Processes: \n";
  cin >> m;
  vector <int> p(m);
  vector <int> hi(m);
  cout << "Enter processes";
  for(int i=0; i< m; i++){
      cin >> p[i];
  }
  vector < vector <int >> a(n);
  for(int i=0; i<n ; i++){
    a[i] = vector <int> (m+1); }
```

```
map<int, int> mp;
for (i=0; i<m; i++) {
    vector<pair<int, int>> C;
    for (auto v: mp) {
        C.push_back({v.second, v.first});
    }
    sort(C.begin(), C.end());
    bool hasrun = false;
    for (j=0; j<n; j++) {
        if (a[j][i] == p[i]) {
            hit++;
            hi[i] = 1;
            mp[p[i]]++;
            hasrun = true;
            break;
        }
        if (a[j][i] == -1) {
            for (k=i; k<m; k++) {
                a[j][k] = p[i];
            }
            mp[p[i]]++;
            hasrun = true;
            break;
        }
    }
    if (j == n || hasrun == false) {
        for (j=0; j<n; j++) {
            if (a[j][i] == C[C.size()-1].second) {
                mp.erase(a[j][i]);
                for (k=i; k<m; k++) {
                    a[j][k] = p[i];
                }
                mp[p[i]]++;
                break;
            }
        }
    }
```

```
for(auto v : mp){
    if(v.first != p[i]){
        mp[v.first]++;
    }
  }
}
cout << "Process ";
for(i=0; i<m; i++){
  cout << p[i] << " ";
}
cout << "\n";
for(i=0; i<n; i++){
   cout << "Frame" << i << " ";
   for(j=0; j<m; j++){
     if(a[i][j] == -1){
        cout << "E ";
     }
     else
       cout << a[i][j] << " ";
   }
   cout << "\n";
}
for(i=0; i<m; i++){
    if(hi[i] == 0)
      cout << " ";
    else
     cout << hi[i] << " ";
   }
cout << "\n";
cout << "Hit" << hit << "\n" << "Page Fault" << m-hit;
return 0; }
```

• OUTPUT-

```
Name- Sarthak bhatia
Roll No- 07714803118
Enter number of frames
3
Enter number of processes
12
Enter processes
1 2 3 4 1 2 5 1 2 3 4 5
Process 1 2 3 4 1 2 5 1 2 3 4 5
Frame 0 1 1 1 4 4 4 5 5 5 5 5 5
Frame 1 E 2 2 2 1 1 1 1 1 3 3 3
Frame 2 E E 3 3 3 2 2 2 2 2 4 4
                1 1     1
Hit 3
Page Fault 9


...Program finished with exit code 0
Press ENTER to exit console.
```

24/5/21 EXPERIMENT-~~7~~6(B)

Sarthak Bhatia
07714803118

- AIM - Write a program to implement page replacement policy using ~~FIFO~~ LRU approach.

- THEORY - Page replacement Algorithm is needed to decide which page needs to be replaced when a new page comes in. Page replacement happens when a requested page is not present in memory and available space is insufficient for allocation to requested page. The page replacement algorithm tries to select which ~~pg~~ pages should be ~~rep~~ replaced as to minimize total number of page misses. There are different algorithms which are evaluated on a particular string of memory reference and the no. of page faults are computed. Fewer the page faults, better the algorithm.

  Least Recently Used (LRU): It is a greedy algorithm in which the page to be replaced is the least recently used. The idea ~~of~~ is based on locality of reference, the least recently used page is not likely.

  Advantage - 1. It is open for full analysis.
  2. It is free from Belady's Anomly.
  3. Easy to chose page which has faulted.

  Disadvantages - 1. Hardware assistance is high.
  2. It requires additional data structure to be implemented.

- PROGRAM -

```
#include <bits/stdc++.h>
using namespace std;
int pageFaults(int pages[], int n, int cap){
    unordered_set<int> s;
    unordered_map<int, int> indexes;
```

```
    int pagefaults = 0;
for (int i = 0; i < n; i++){
    if (s.size() < cap) {
        if (s.find(pages[i]) == s.end()){
            s.insert(pages[i]);
            pagefaults++;
        }
        indexes[pages[i]] = i;
    }
    else {
        if (s.find(pages[i]) == s.end()){
            int lru = INT_MAX, val;
            for (auto it = s.begin; it != s.end(); i++){
                if (indexes[*it] < lru){
                    lru = indexes[*it];
                    val = *it;
                }
            }
            s.erase(val);
            s.insert(pages[i]);
            pagefaults++;
        }
        indexes[pages[i]] = i;
    }
}
    return pagefaults;
}
int main() {
    int n, cap;
    cout << "Sarthak Bhatia \n 07714803118";
    cout << "\n Enter no of pages: ";
    cin >> n;
    int pages[n];
```

```
cout << " Enter the no. of sequence of pages : ";
for (int i=0; i<n; i++)
        cin >> pages[i];
cout << " Enter the no. of frames : ";
cin >> cap;
int pagefaults = pageFaults(pages, n, cap);
cout << " No. of hits : " << n-pagefaults;
cout << " No. of page faults : " << pagefaults;
return 0;
}
```

- OUTPUT -

```
Sarthak Bhatia
 07714803118
Enter number of pages: 8
Enter the number of sequences of pages: 0 1 2 3 1 2 0 2
Enter the number of frames: 3
Number of hits: 3
Number of page faults: 5

...Program finished with exit code 0
Press ENTER to exit console.
```

24/5/2021 EXPERIMENT-6C

- AIM - Write a program to implement page replacement policy using optimal page replacement algorithm.
- THEORY - A page replacement Algorithm is needed to decide which page needs to be replaced when a new page comes in. Page replacement happens when a requested page is not present in memory and available space is insufficient for allocation to requested page.

The page replacement algorithm tries to select which pages should be replaced so that no. of page misses are minimum. There are different algorithms which are evaluated on a particular string of memory reference and no. of page faults are compared. Fewer the page fault, better is the algorithm.

Optimal Page Replacement - In this algorithm, OS replaces the pages that will not be used for the longest period of time in near future. Although it can't be practically implemented, but it is most optimal and have minimal miss.

Advantages - 1. Assistance needed is low.
2. Complexity is less and easy to implement.

Disadvantages - 1. Error handling is tough.
2. Not possible as OS cannot know future requests.

- PROGRAM -

```
#include<bits/stdc++.h>
using namespace std;
bool search(int key, vector<int>&fr){
    for(int i=0; i<fr.size(); i++){
        if(fr[i]==key)
            return true;
    }
    return false;}
```

```
int predict (int pg [], vector <int> &fr, int pn, int idx){
    int res = -1, farthest = idx;
    for (int i = 0; i < fr.size(); i++){
        for (int j = idx; j < pn; j++){
            if (fr[i] == pg[j]){
                if (j > farthest){
                    farthest = j;
                    res = i;
                }
                break;
            }
        }
        if (j == pn)
            return i;
    }
    return (res == -1) ? 0 : res;
}
void optimalPage (int [] pg, int pn, int fn) {
    vector <int> fr;
    int hit = 0;
    for (int i=0; i<pn; i++){
        if (search(pg[i], fr)){
            hit++;
            continue;
        }
        if (fr.size() < fn){
            fr.push_back(pg[i]);
        else {
            int j = predict (pg, fr, pn, i++);
            fr[j] = pg[i];
        }
    }
    cout << "\n No. of hits = " << hit;
    cout << "\n No of misses = " << pn - hit;
}
```

```
int main() {
   int n, fn;
   cout << "Sarthak Bhatia \n 07714803118";
   cout << "\n Enter no. of page requests: ";
   cin >> n;
   int pg[n];
   cout << "Enter the page requests: ";
    for(int i=0; i<n; i++)
          cin >> pg[i];
    cout << "Enter the frame size: ";
    cin >> fn;
    optimalPage(pg, n, fn);
    return 0;
  }
```

- OUTPUT-

```
Sarthak Bhatia
 07714803118
Enter number of page requests: 8
Enter the page requests: 0 1 2 0 3 0 4 2
Enter the frame size: 3
No. of hits = 3
No. of misses = 5



...Program finished with exit code 0
Press ENTER to exit console.
```

31/5/2021 EXPERIMENT- 7

- AIM- Write a program to implement first fit, best fit and worst fit algorithms for memory management.
- THEORY- For both fixed and dynamic memory allocation schemes, the operating system must keep list of each memory location noting which are free and which are busy. Then as new jobs come into the system, the free partitions must be allocated.
  These partitions may be allocated by 3 ways:-
  1. First-fit memory allocation.
  2. Best-fit memory allocation.
  3. Worst-fit memory allocation.

  First-Fit memory allocation - This method keeps the free/busy list of jobs organisld by memory-location. Here, first job claims the first available memory with space more than or equal to its size.

  Best-Fit memory allocation - In this method, the operating system first searches the whole of the memory according to the size of given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently.

  Worst-Fit memory allocation - Here, the process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition.
- PROGRAM-

```
#include<bits/stdc++.h>
using namespace std;
void firstFit(int blocks[], int n, int processes[], int m){
int allocation[m];
```

```
for (int i=0; i<m; i++) {
        allocation[i] = -1;
}

for (int i=0; i<m; i++){
    for (int j=0; j<n; j++) {
        if (blocks[j] >= processes[i]) {
            allocation[i] = j
            blocks[j] -= processes[i];
            break;
        }
    }
    cout << "------FirstFit------";
for (int i=0; i<m; i++) {
    if (allocation[i] != -1) {
        cout << "Process " << i+1 << " -> " << allocation[i]+1;
    }
    else {
        cout << "Process " << i+1 << " -> " << "Unallocated";
    }
}
}

void bestFit (int blocks[], int n, int processes[], int m){
    int allocation[m];
    for (int i=0; i<m; i++){
        allocation[i]=-1;
    }
    for (int i=0; i<m; i++) {
        int idx = -1;
        for (int j=0; j<n; j++) {
```

```
        if (blocks[j] >= processes[i]) {
            if (idx == -1)
                idx = j;
            else if (blocks[idx] > blocks[j])
                idx = j;
        }
    }
    if (idx != -1) {
        allocation[i] = idx;
        block[idx] -= processes[i];
    }
}
cout << "\n\n---- Best Fit ----\n";
for (int i=0; i<m; i++) {
    if (allocation[i] != -1) {
        cout << "Process " << i+1 << " -> " << allocation[i]+1;
    else {
        cout << "Process " << i+1 << " -> " << "Unallocated";
    }
}
}
void worstFit(int blocks[], int n, int processes[], int m) {
    int allocation[n];
    for (int i=0; i<m; i++) {
        allocation[i] = -1;
    }
    for (int i = 0; i<m; i++) {
        int idx = -1;
        for (int j=0; j<n; j++) {
            if (blocks[j] >= processes[i]) {
                if (idx == -1) idx = j;
```

```
        else if (blocks[idx] < blocks[j])
            idx = j;
    }
  }
    if (idx != -1) {
        allocation[i] = idx;
        blocks[idx] -= processes[i];
    }
  }
    cout << "----- Worst Fit -----";
  for(int i=0; i<m; i++){
    if(allocation[i] != -1) {
      cout << "Process " << i+1 << " -> " << allocation[i]+1;
    }
    else{
    cout << "Process " << i+1 << " -> " << "Unallocated";
    }
  }
}
int main(){
    int n, m;
    cout << "Sarthak Bhatia\n0771480318" << endl;
    cout << "Enter the number of blocks: ";
    cin >> n;
    cout << "Enter the number of processes: ";
    cin >> m;
    int blocks[n];
    int processes[m];
    cout << "Enter block sizes: " << endl;
    for(int i = 0; i<n; i++){
        cin >> blocks[i];
    }
```

```
cout << "Enter processes size : " << endl;
for(int i=0; i < m; i++){
    cin >> processes[i];
}
int option;
cout << "Enter 1 for First fit, 2 for Best fit and 3 for
  Worst fit algorithm";
cin >> option;
switch (option) {
  case 1:
    firstFit(blocks, n, processes, m);
    break;
  case 2:
    bestFit(blocks, n, processes, m);
    break;
  default:
    worstFit(blocks, n, processes, m);
}
return 0;
}
```

- OUTPUT -

```
---------First Fit---------
Sarthak bhatia
07714803118
Enter the number of blocks: 5

Enter the number of processes: 4
Enter block sizes:
500
100
300
600
200
Enter processes sizes:
212
590
350
150
Process 1 -> 1
Process 2 -> 4
Process 3 -> Unallocated
Process 4 -> 1
```

```
---------Best Fit---------
Sarthak bhatia
07714803118
Enter the number of blocks: 5

Enter the number of processes: 4
Enter block sizes:
500
100
300
600
200
Enter processes sizes:
212
590
350
150
Process 1 -> 3
Process 2 -> 4
Process 3 -> 1
Process 4 -> 1


...Program finish
Press ENTER to ex
```

```
---------Worst Fit---------
Sarthak bhatia
07714803118
Enter the number of blocks: 5

Enter the number of processes: 4
Enter block sizes:
500
100
300
600
200
Enter processes sizes:
212
590
350
150
Process 1 -> 4
Process 2 -> Unallocated
Process 3 -> 1
Process 4 -> 4
```

7/06/21

EXPERIMENT- 8

- AIM- Write a program to implement reader / writer problem using semaphore.
- THEORY- Suppose that a database is shared among several concurrent processes. Some of these processes may want only to read the database, where as others may want to update the database. Obviously, if two reader across the shared data simultaneously, no adverse effect will result. However if a writer and some other process (either a reader or a writer) access the database simultaneously, chaos may ensue. To ensure that these difficulties do not arise, we require that the writer have exclusive access to shared database while writing to database. This synchronisation problem is referred to as reader-writer problem. The reader writer problem has several variations, all involving priorities. The simplest one is that no reader be kept waiting unless a writer has obtained permission to use shared object.
- PROGRAM-

```
# include <pthread.h>
# include <semaphore.h>
# include <stdio.h>
  sem_t  wrt;
  pthread_mutex_t  mutex;
  int  numreader = 0;

void * writer (void * who) {
   sem_wait (& wrt);
   cnt = cnt * 2;
   printf ("writer %d modified cnt to %d \n",
      (*((int *) who)), cnt);
      sem_post (& wrt);
}
```

```
void * reader (void* rno) {
    pthread_mutex_lock (&mutex);
    numreader++;
    if (numreader == 1)
            sem_wait (&wrt);
    pthread_mutex_unlock (&mutex);
    printf ("Reader %d : read cnt as %d \n", *((int*)rno
        cnt);
    pthread_mutex_lock (&mutex);
    numreader--;
    if (numreader == 0)
        sempost (&wrt);
    pthread_mutex_unlock (&mutex);
}
int main () {
    int n, m;
    printf ("Enter the number of reader and writers \n");
    scanf ("%d", n);
    scanf ("%d", m);
    pthread_t read[n], write[m];
    pthread_mutex_init (&mutex, NULL);
    sem_init (&wrt, 0, 1);
    int a[n];
    for (int i=0; i<n; i++) {
        a[i] = i+1;
    }
    for (int i=0; i<n, i++) {
        pthread_create (&read[i], NULL, (void*) reader,
            (void*) &a[i]);
    }
    for (int i=0; i<m; i++) {
        pthread_create (&write[i], NULL, (void*) writer,
            (void*) &a[i]);  }
```

```
for(int i=0; i<n; i++){
    pthread_join(read[i], NULL);
}
for(int i=0; i<m; i++){
    pthread_join(write[i], NULL);
}
pthread_mutex_destroy(&mutex);
sem_destroy(&wrt);
return 0;
}
```

- OUTPUT-

```
Reader 1: read cnt as 1
Reader 2: read cnt as 1
Reader 3: read cnt as 1
Reader 4: read cnt as 1
Reader 5: read cnt as 1
Reader 7: read cnt as 1
Reader 8: read cnt as 1
Reader 9: read cnt as 1
Writer 1 modified cnt to 2
Writer 2 modified cnt to 4
Writer 3 modified cnt to 8
Writer 4 modified cnt to 16
Writer 5 modified cnt to 32
Reader 6: read cnt as 32
Reader 10: read cnt as 32


...Program finished with exit code 0
Press ENTER to exit console.
```

7/6/21

EXPERIMENT-9

- **AIM** - Write a program to implement Banker's algorithm for deadlock avoidance.
- **THEORY** - The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amount of all resources, then makes an 's-state' check to test for possible activities, before deciding whether allocation should be allowed to continue. This algorithm is named so because it is used to continue banking system to check whether loan can be sanctioned to a person or not.
- **PROGRAM** -

```
#include <stdio.h>
int main() {
  int n, m, k;
  printf("Enter number of processes: ");
  scanf("%d", n);
  printf("Enter number of resources: ");
  scanf("%d", m);
  int alloc[n][m], max[n][m], avail[m];
  printf("Enter allocation matrix \n");
  for(int i=0; i<n; i++){
     for(int j=0; j<m; j++) {
        scanf("%d", alloc[i][j]);
  }
  }
  printf("Enter max matrix \n");
  for(int i=0; i<n; i++){
     for(int j=0; j<m; j++) {
        scanf("%d", max[i][j]);
```

```
    }
  }
printf("Enter available resources \n");
for(int j=0;j<m;j++){
    scanf("%d", avail[j]);
}

int f[n], ans[n], ind=0;
for (int k=0; k<n; k++){
        f[k]=0;
}
int need[n][m];
for(int i=0; i<n; i++){
   for(int j=0; j<m; j++){
        need[i][j] = max[i][j] - alloc[i][j];
   }
}
int y=0;
for(k=0; k<n; k++){
    for(int i=0; i<n; i++) {
        if(f[i]==0){
            int flag=0;
            for(int j=0; j<m; j++) {
                if(need[i][j]>avail[j]){
                    flag=1;
                    break;
                }
            }
            if(flag[i]==0) {
                ans[ind++] = i;
```

→

```
            for(y=0; y<m; y++){
                avail[y] += alloc[i][y];
            }
            f[i]=1;
        }
    }
  }
}
printf("Following is safe sequence \n");
for(int i=0; i<n-1; i++){
    printf(" P%d ->", ans[i]);
}
printf("P%d", ~~ans[i]~~ ans[n-1]);
return 0;
}
```

OUTPUT-

```
'Enter number of processes : 5
'Enter number of resources : 3
Enter allocation matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter max matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available resources
3 3 2
Following is the SAFE Sequence P1 -> P3 -> P4 -> P0 -> P2

...Program finished with exit code 0
Press ENTER to exit console.
```