# USP LAB EXAM LAB 1

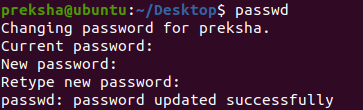
*Enter these commands at the UNIX*

*prompt, and take snapshot of the output and submit:*

### Echo hello world



*Passwd*



### Date



*Hostname*



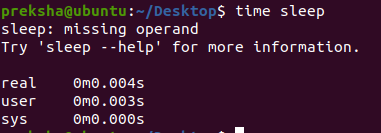
### Arch



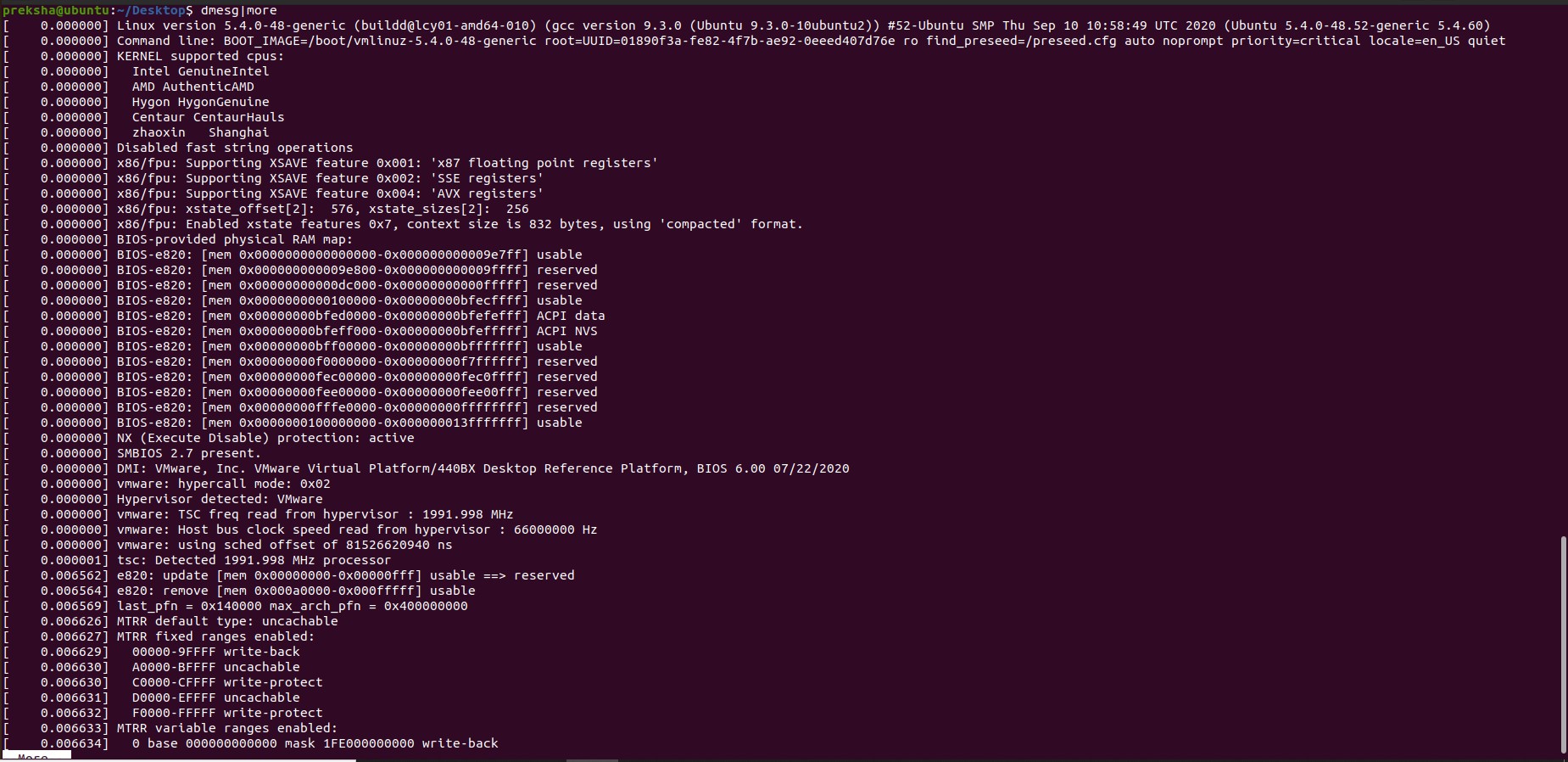
*Uname-a*



### Time sleep



*dmesg | more (you may need to press q to quit)*



### Uptime



*Who am i*



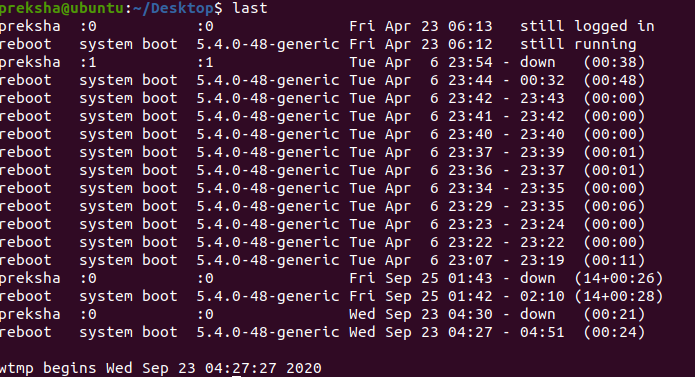
### Who



*Id*



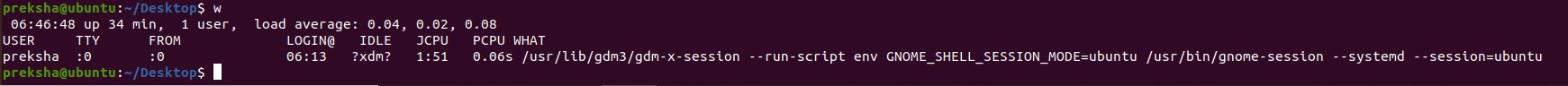
### Last



*Finger*



### W



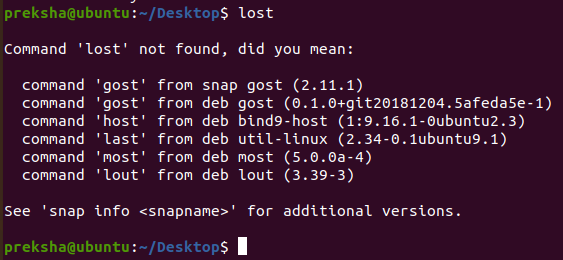
*Echo*



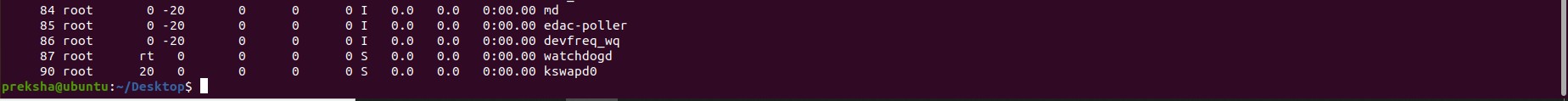
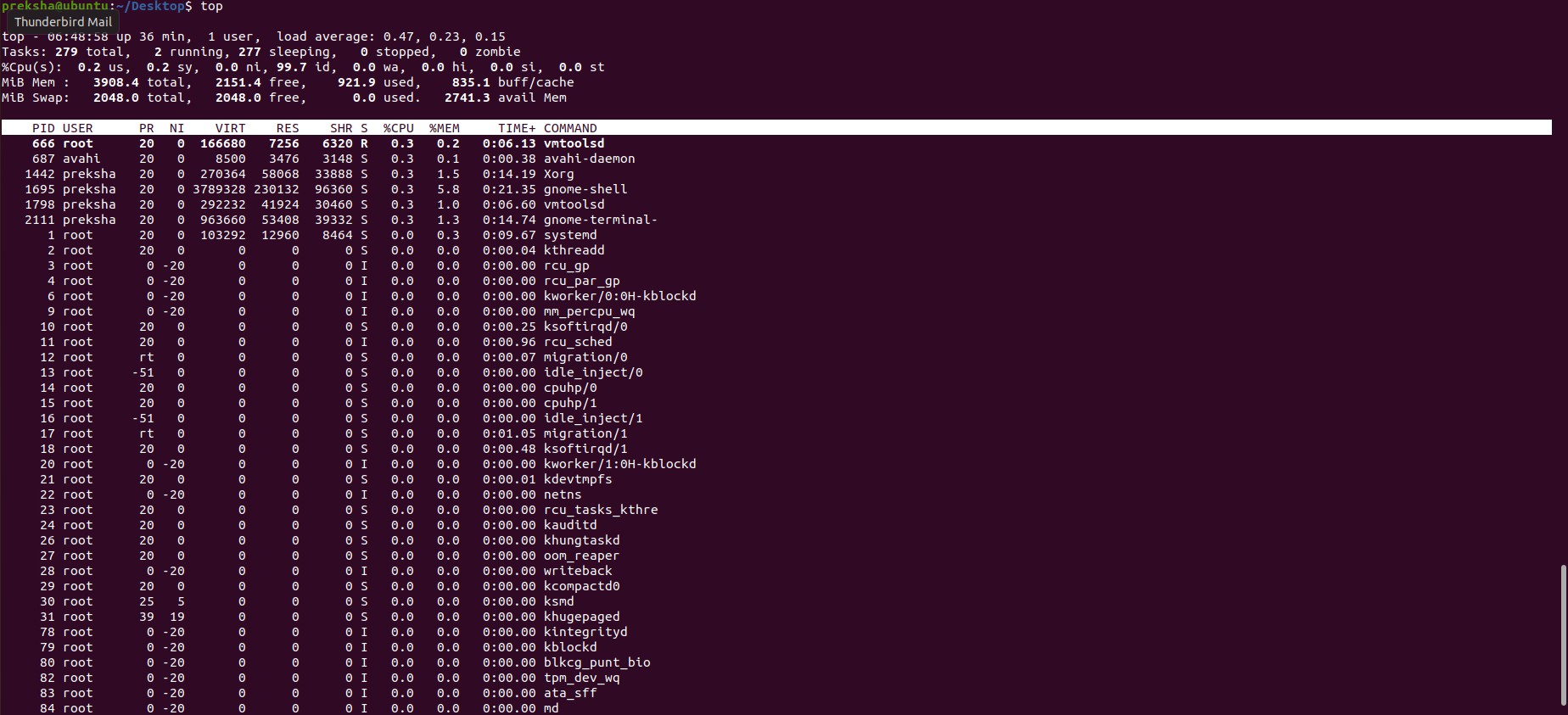
### Man “automatic door”



*Lost*



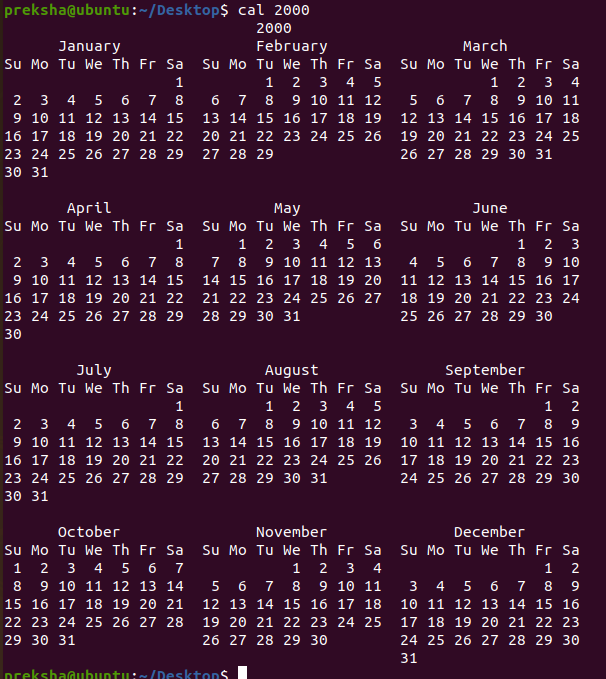
### Top



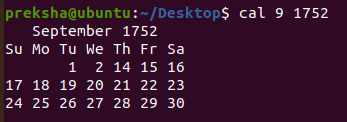
*Clear*



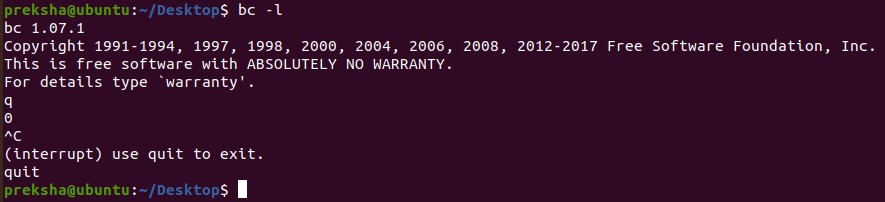
### Cal 2000



*Cal 9 1752*



### bc -l (type quit or press Ctrl-d to quit)



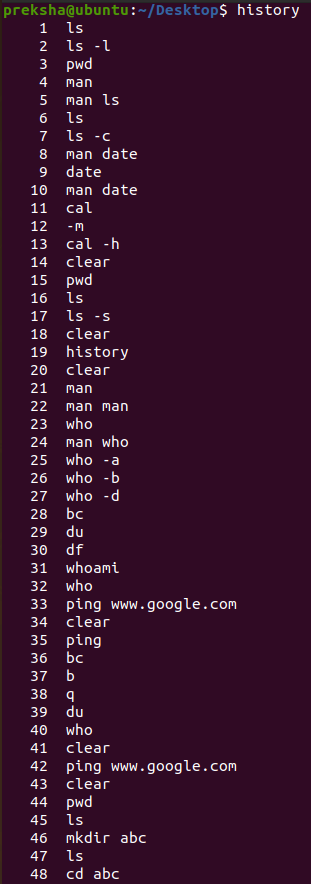
*echo 5+4 | bc –l*

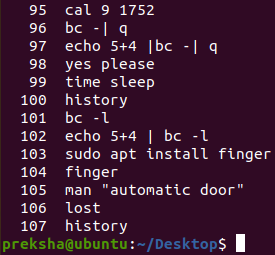
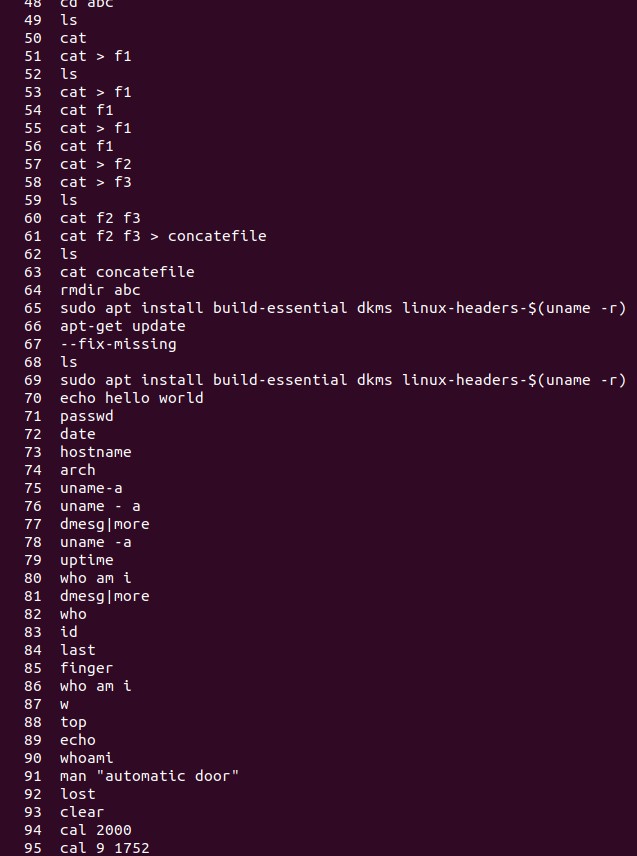


### Yes please



*History*





# LAB 2

*Try the following command sequence and submit with snapshot of output*

### cd

1. *pwd*

### cd ..

1. *pwd*

### ls –al

1. *cd ..*

### pwd (what happens now)

1. *cd /etc*

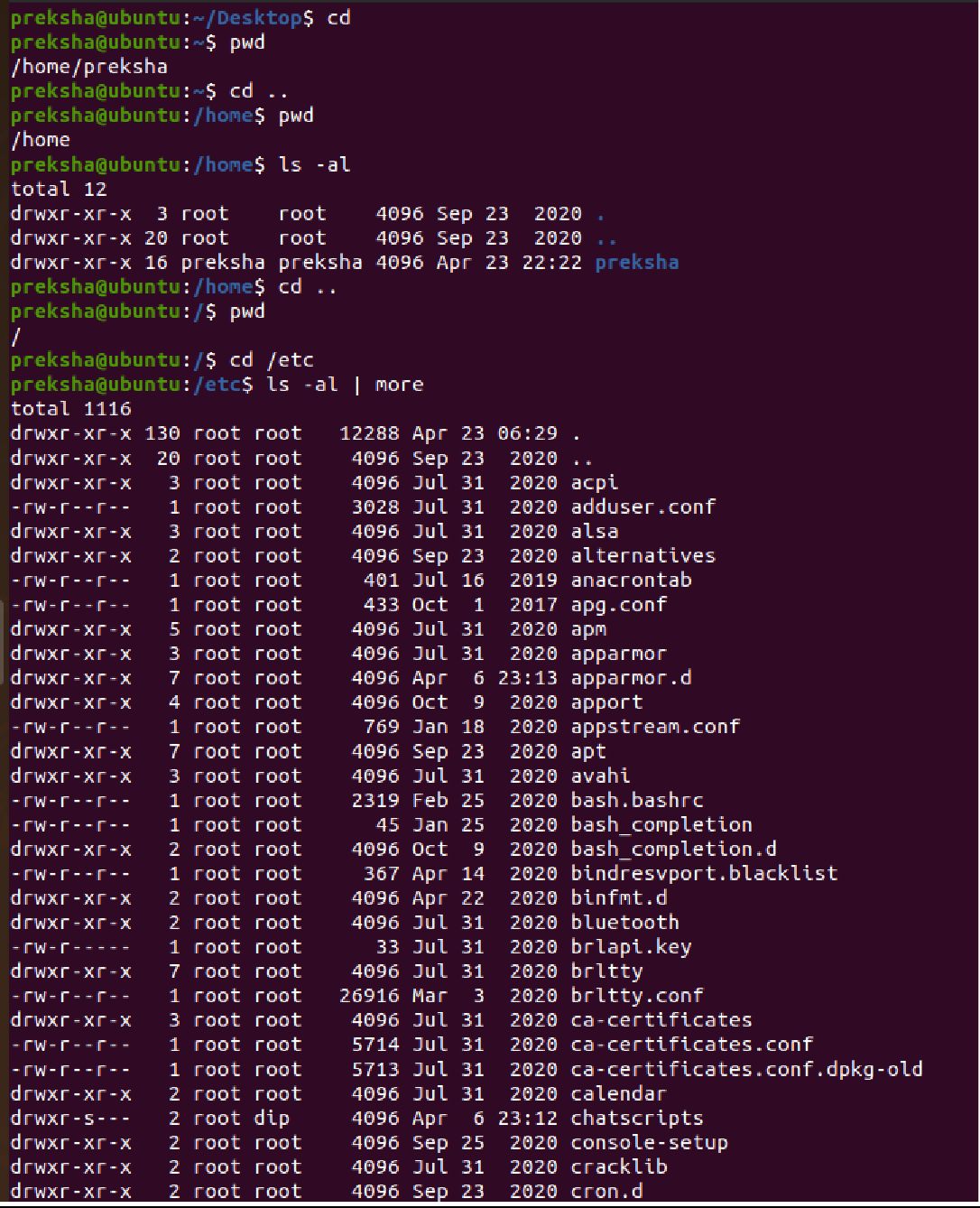
### ls –al | more

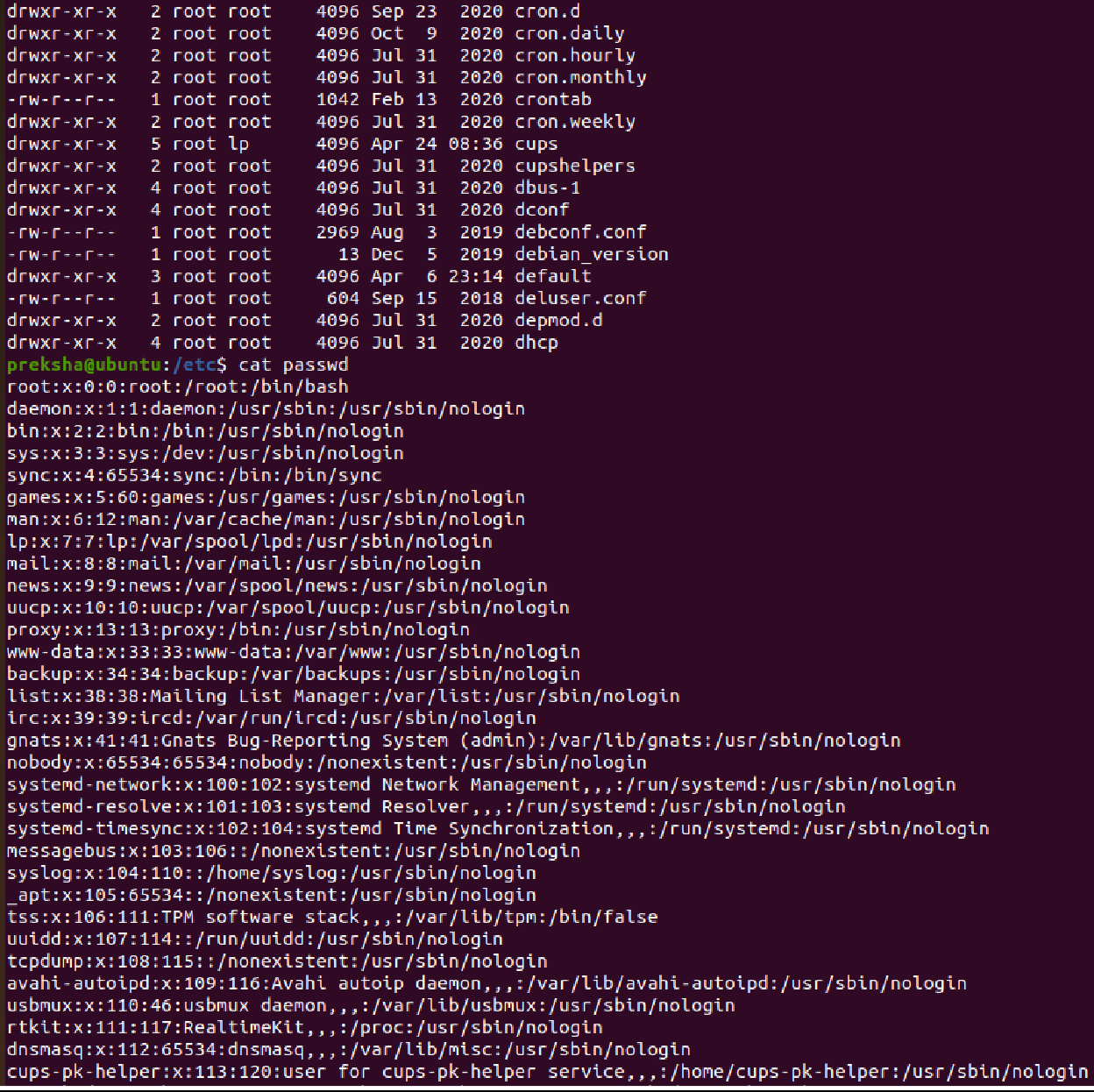
1. *cat passwd*

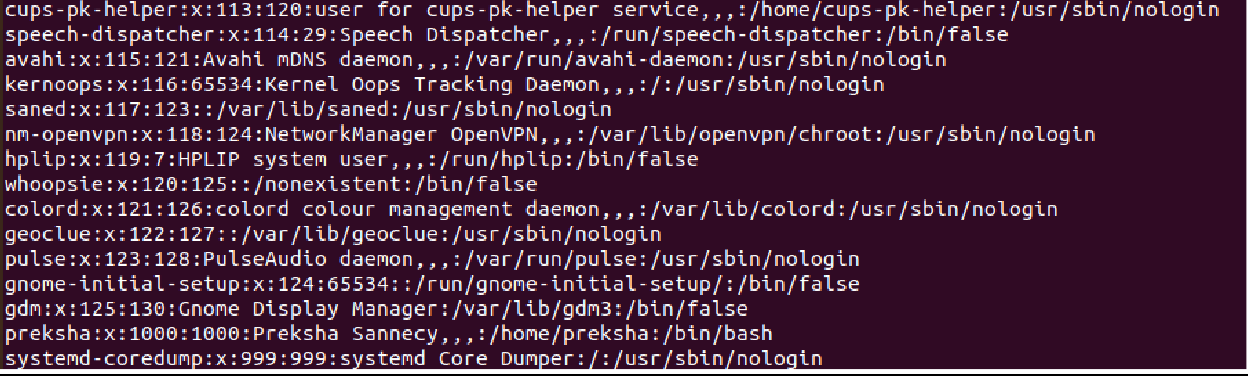
### cd ~

1. *pwd*

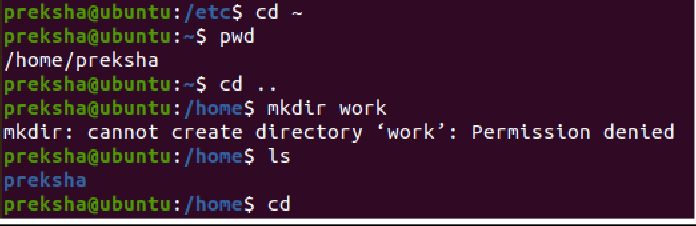
*COMBINED OUTPUT OF THE COMMANDS:*



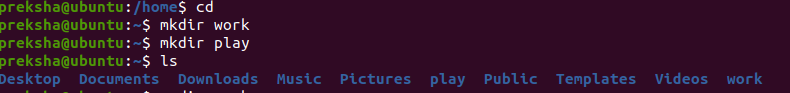




### Change back into your home directory.



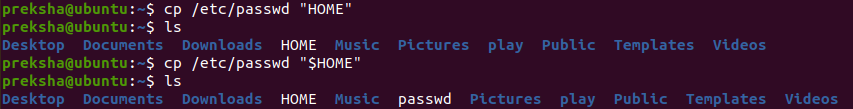
1. *Make subdirectories called work and play.*



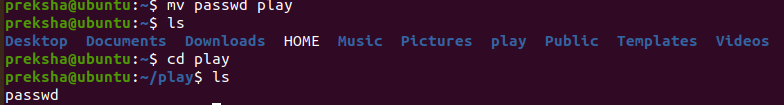
### Delete the subdirectory called work.



1. *Copy file /etc/passwd into your home directory.*

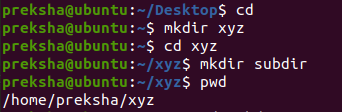


1. *Move it into the subdirectory play*

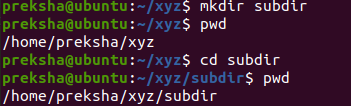


# LAB 3

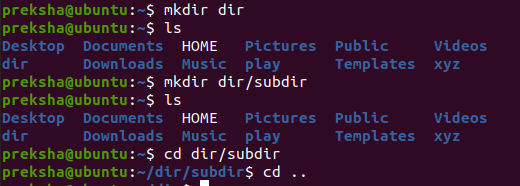
1. *Create a directory.*



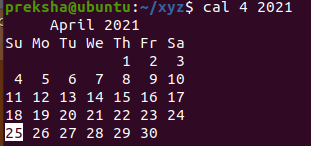
1. *Create a subdirectory in the directory created.*



1. *Change your current directory to the subdirectory.*



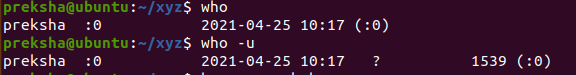
1. *Display the calendar of the month.*



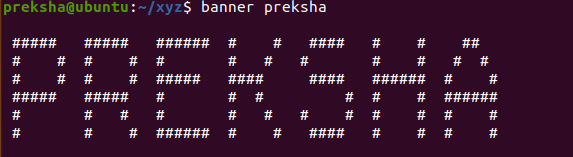
1. *Get a directory listing the parent directory.*



1. *How many users were logged onto your device.*



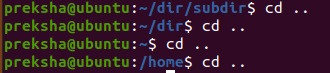
1. *Display your name in the form of a banner.*



1. *Display the name of device name of your terminal.*

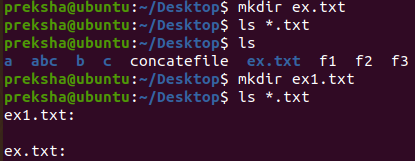


1. *Move to the root directory.*



# LAB 4

### List all the processes that are presently running.

1. *List the text files in your current directory.*

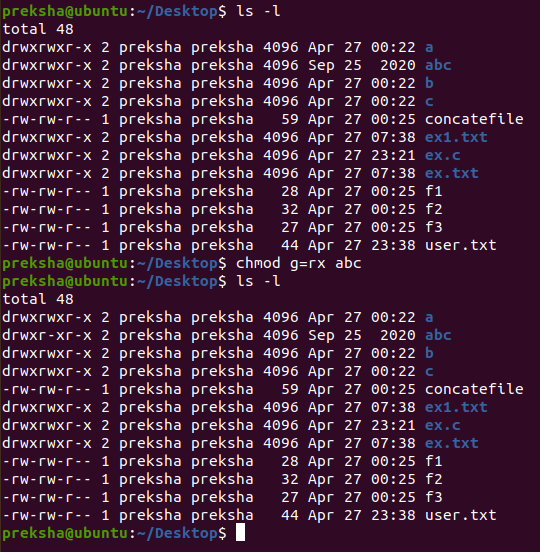
### Make a copy of any text file.



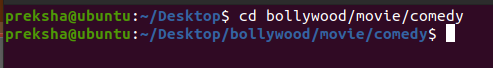
1. *Rename one of your text files in the current directory.*
2. *Delete an unneeded copy of a file.*

*LAB 5:*

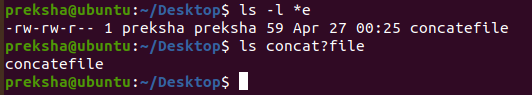
*Set permissions on all of your files and directories to those that you want. You may want to give read permission on some of your files and directories to members of your group.*



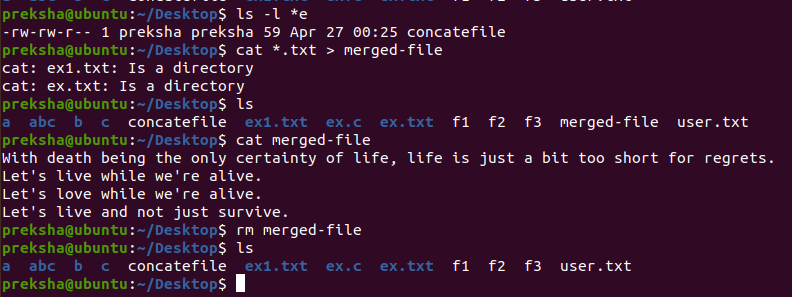
*Create a number of hierarchically related directories and navigate through them using a combination of absolute path names (starting with "/") and relative pathnames.*



*Try using wildcards (“\*” and possibly “?”).*

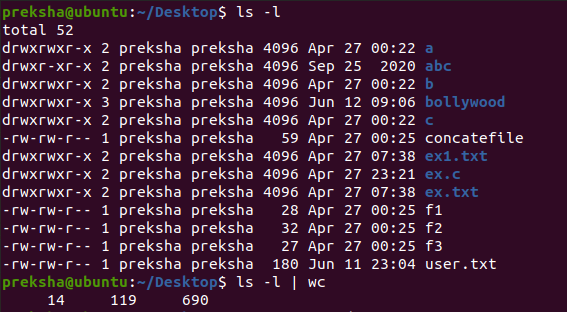


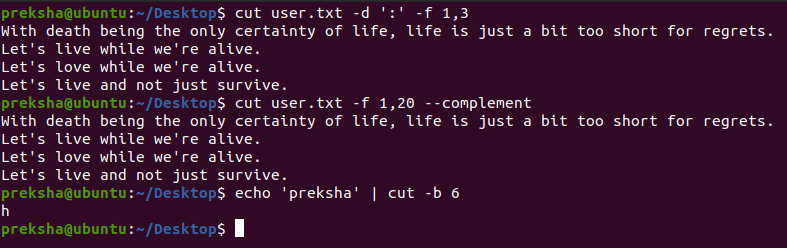
*Put a listing of the files in your directory into a file called file list. (Then delete it!)*

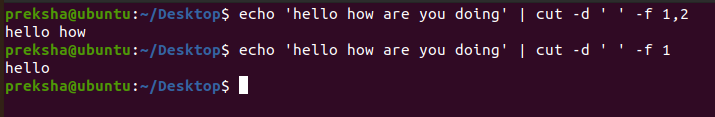


*LAB 6:*

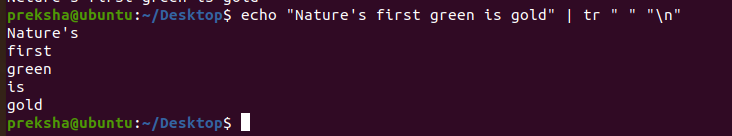
*Type the command ls -l and examine the format of the output. Pipe the output of the command ls -l to the word count wc to obtain a count of the number of files in your directory.*



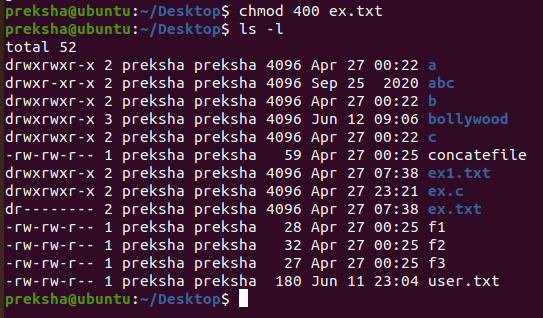
*Use cut to strip away the reference material and leave just the text field.*



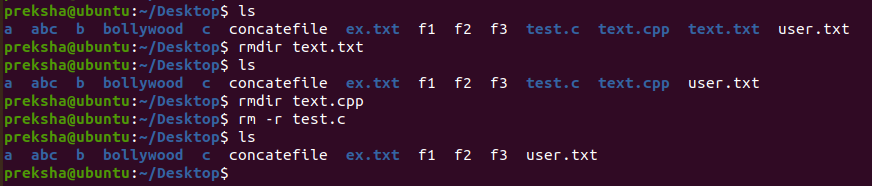
*Use tr to strip away any tags that are actually in the text (e.g., attached to the words), so that you are left with just the words.*



*Set a file to be read-only with the chmod (from change mode) command. Interpret the file permissions displayed by the ls –l command.*



*Delete one or more directories with the rmdir (from remove directory) command. See what happens if the directory is not empty. Experiment (carefully!) with the rm -r command to delete a directory and its content.*

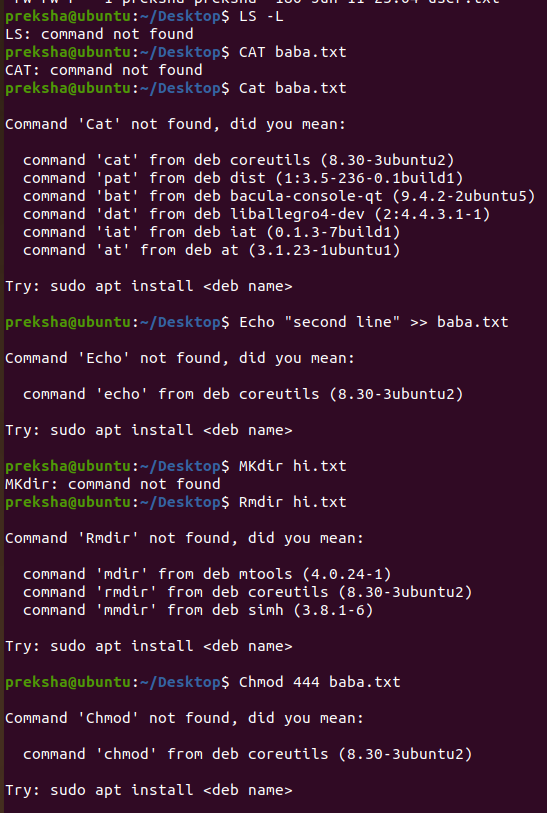


*Experiment with redirecting command output (e.g., ls -l >file1). Try ">>" instead of " >"*

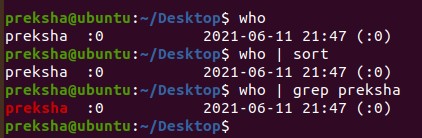
*with an existing text file as the output.*



*See whether upper-case versions of any of these commands work as well as the lower-case versions.*

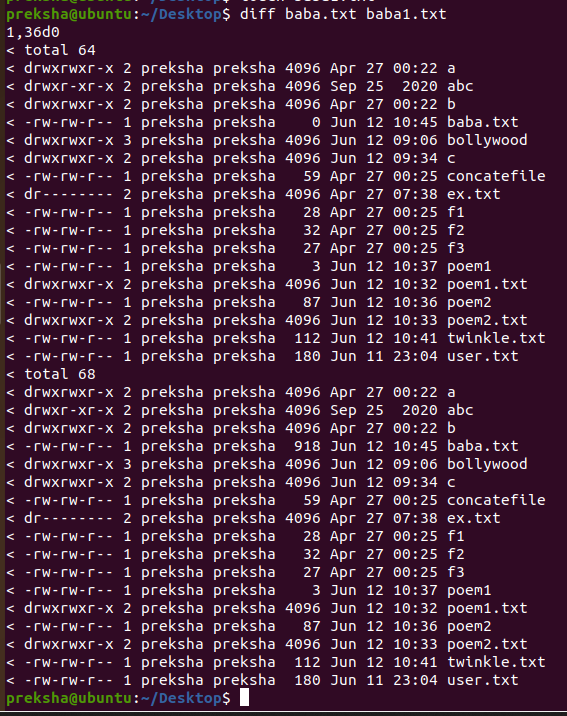


*Use the who command to see users logged into the system.Pipe the output of the who command to the sort command.Search for your login name in who file using the grep command.*

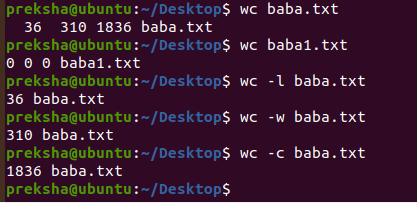


## LAB 7:

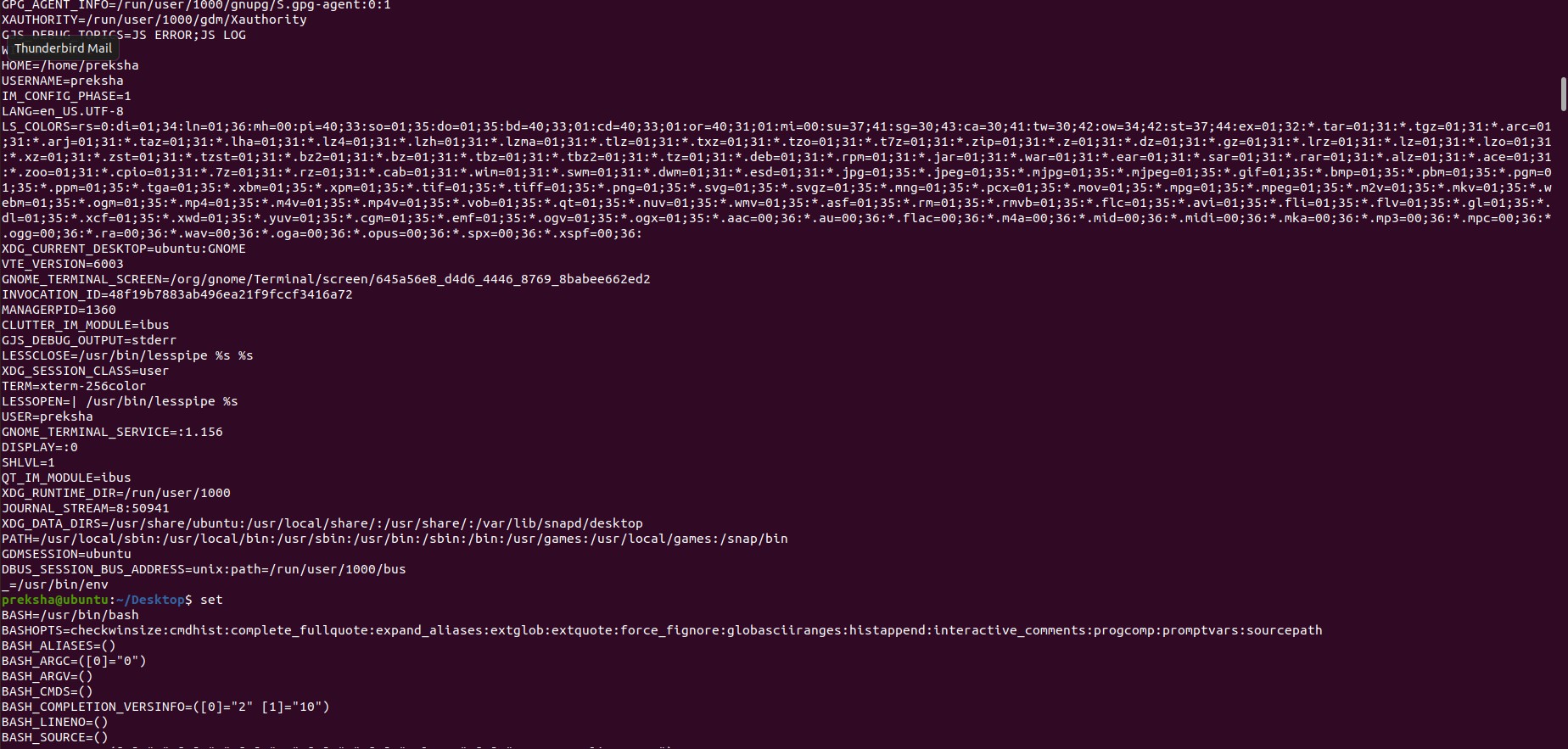
*Compare two text files with the diff command.*

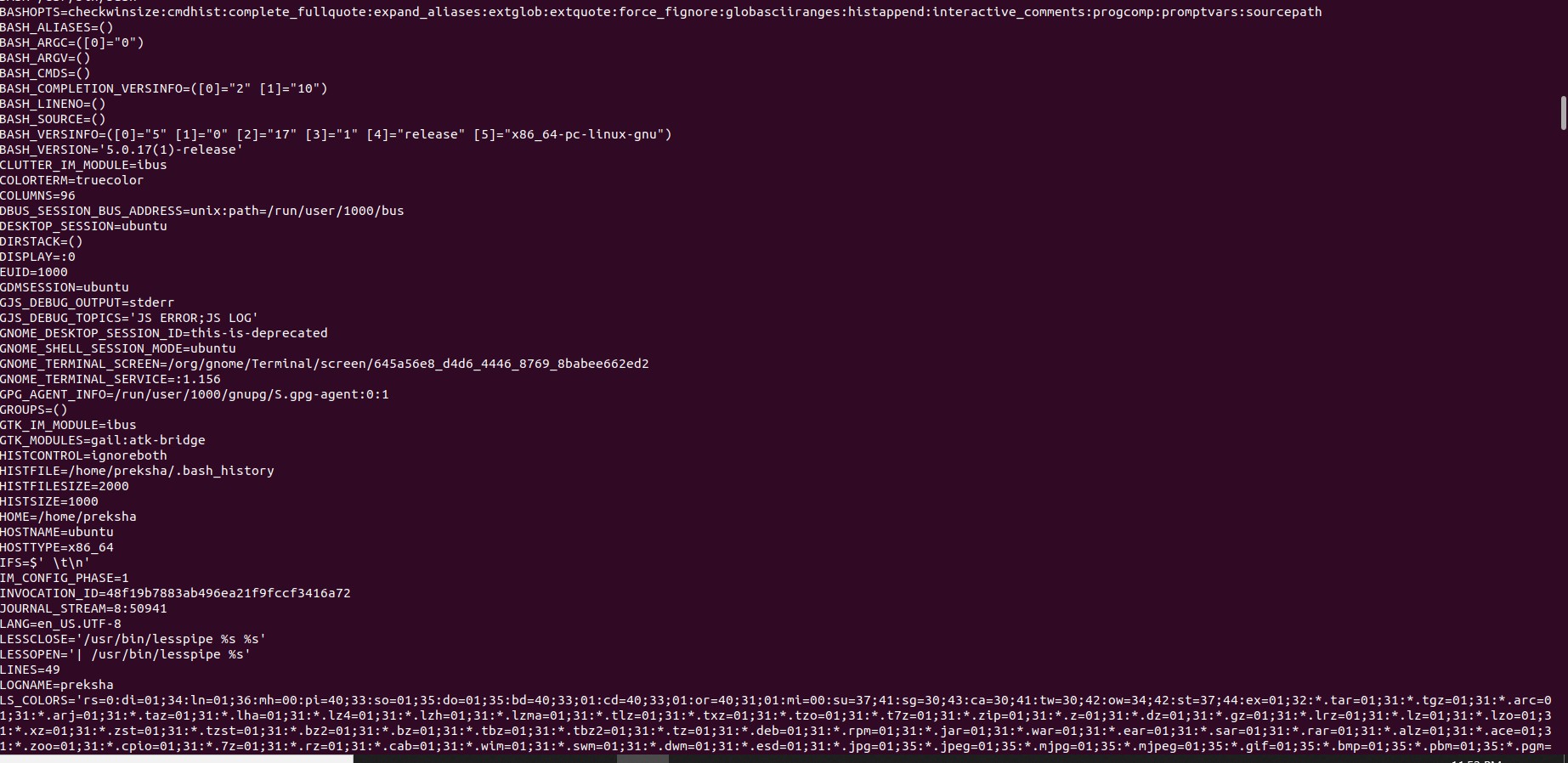


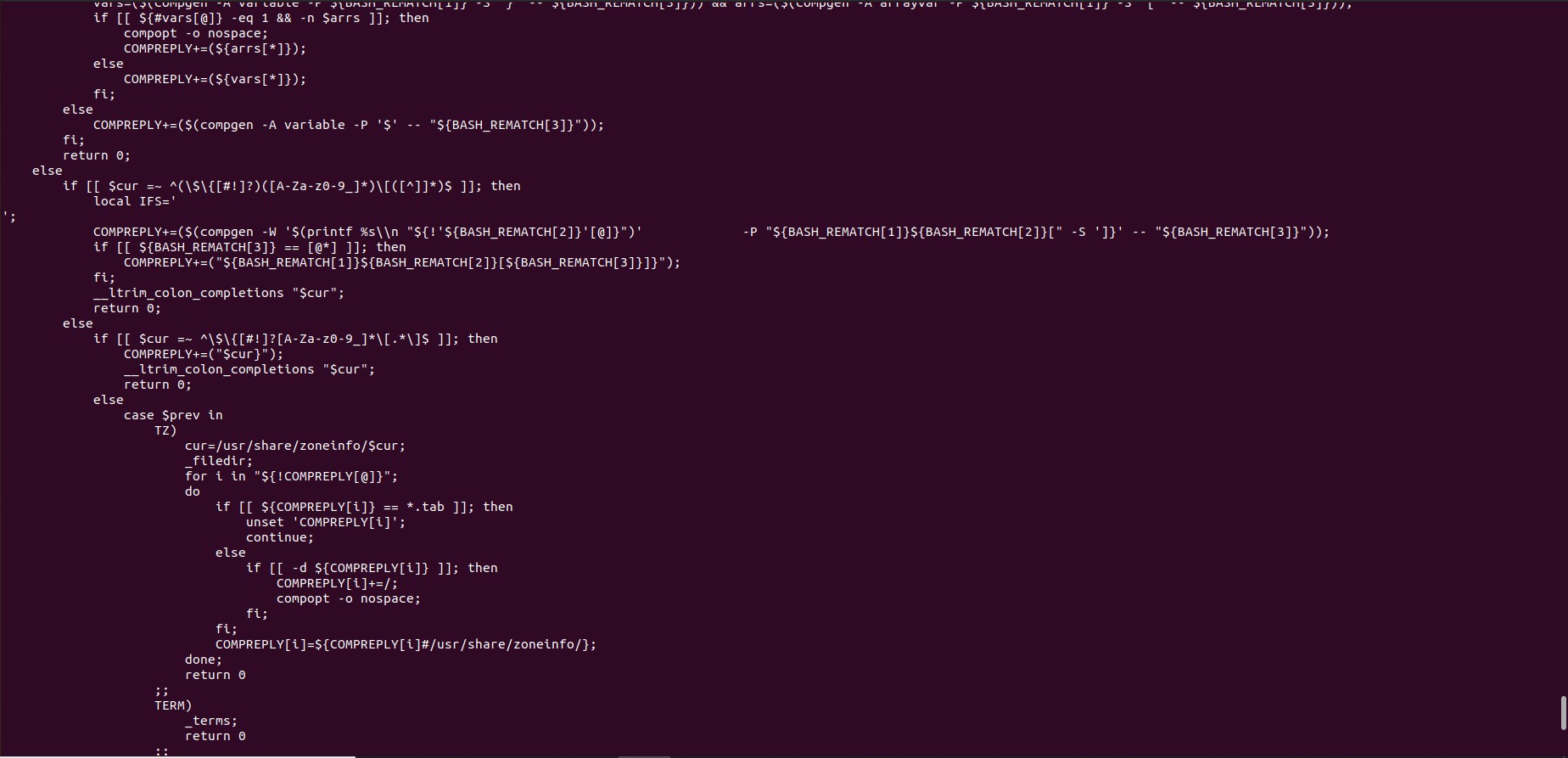
*Count lines, words, and characters in a file with the wc command.*



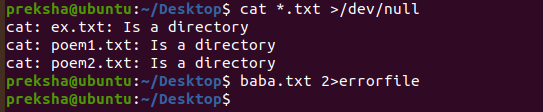
*Display your current environment variables with the following command: set or env.*



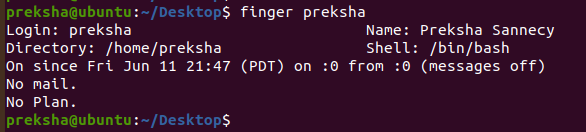




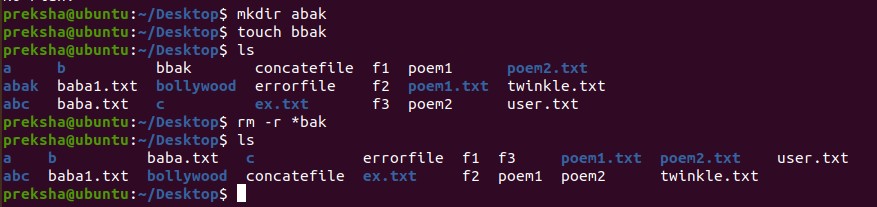
*Concatenate all files in a directory redirected to /dev/null and redirecting standard error to “errorFile”?*

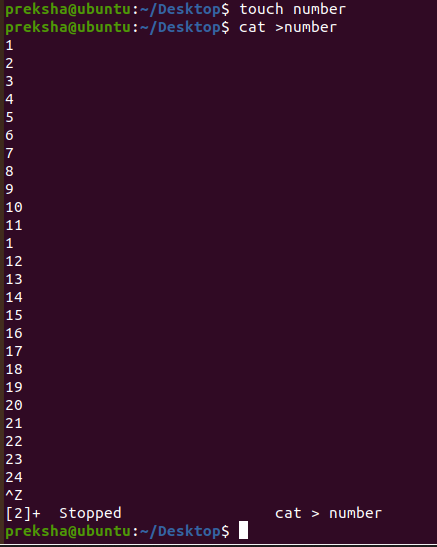


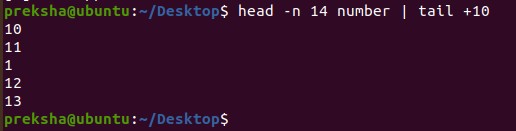
*Display information on yourself or another user with the finger command.*

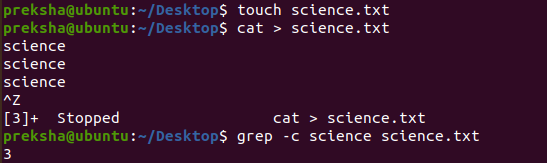


*If you wish, experiment with sending and receiving mail using the pine email program. Delete all the files in the current directory whose name ends in “.bak”.*

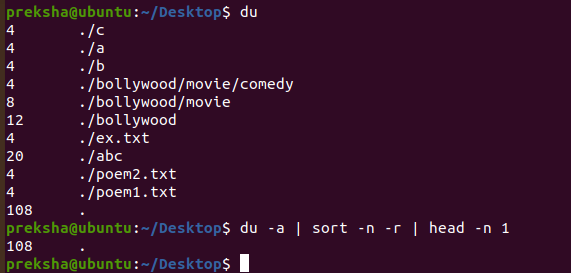


*Display lines 10 to 14 of any file which contains 25 lines.*



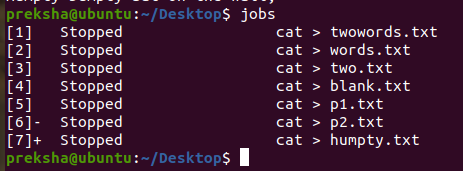
*Count how many lines contain the word science in a word file science.txt.*

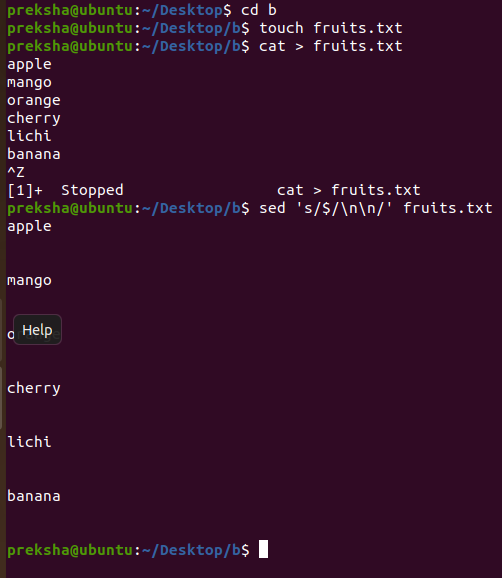
*List the statistics of the largest file (and only the largest file) in the current directory.*



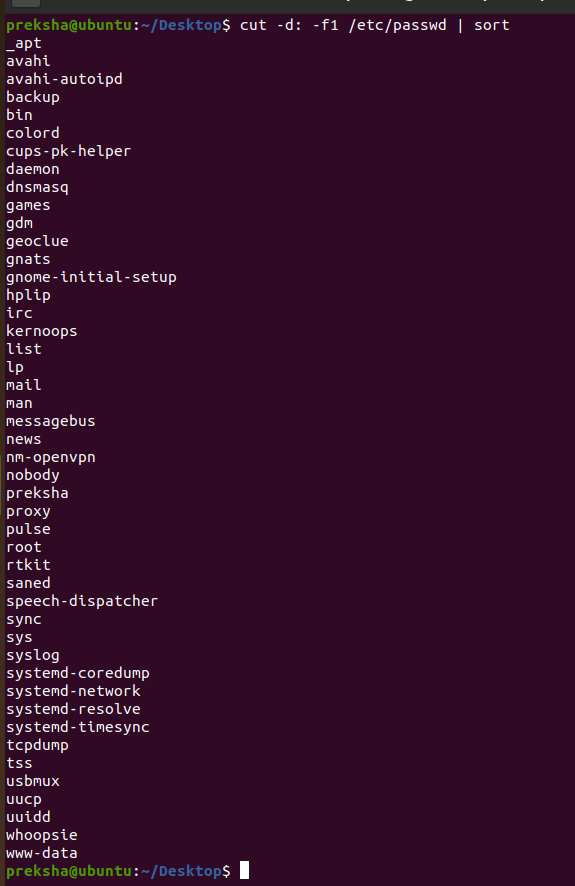
*LAB 8:*

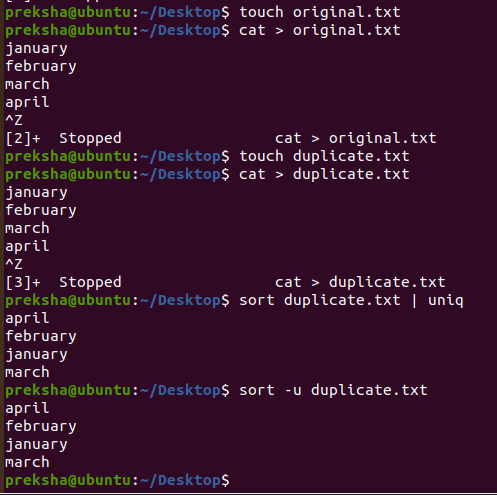
*Kill any process with the help of the PID and run any process at the background.*



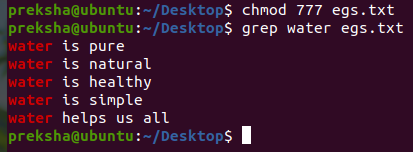
*Select a text file and double space the lines.*

*List all the users from /etc/passwd in the alphabetically sorted order.*

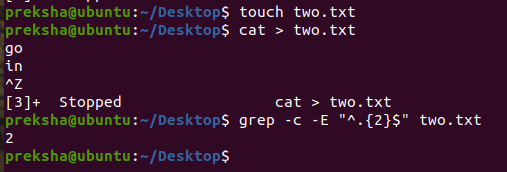


*Create a file with duplicate records and delete duplicate records for that file.*

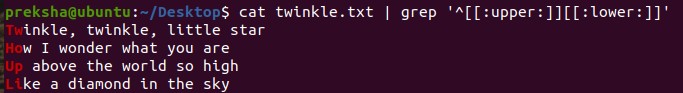
*Use the grep command to search the file example1 for occurrences of the string*

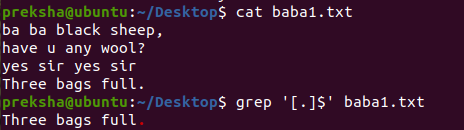
*“water”.*

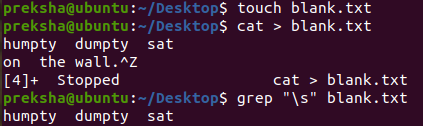
*Write grep commands to do the following activities:*

*To select the lines from a file that have exactly two characters.*

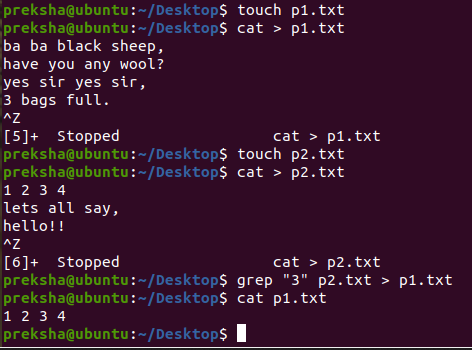
*To select the lines from a file that start with the upper case letter.*



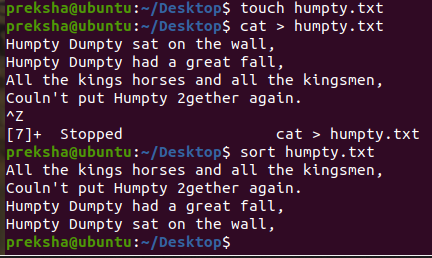
*To select the lines from a file that end with a period.*

*To select the lines in a file that has one or more blank spaces.*

*To select the lines in a file and direct them to another file which has digits as one of*

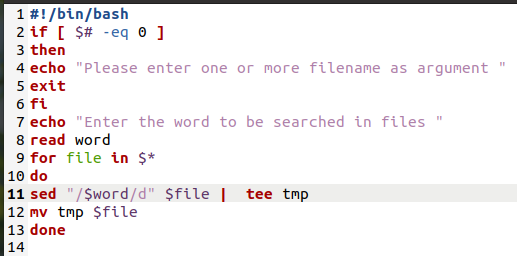
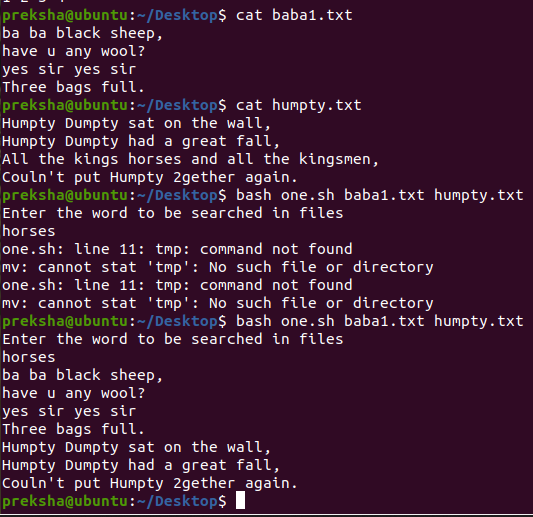
*the characters in that line.*

*Make a sorted wordlist from the file.*

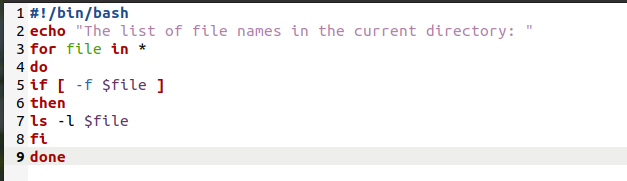


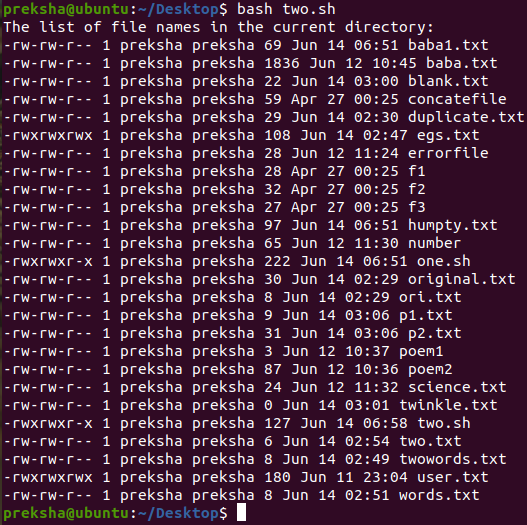
## LAB 9,10,11 SHELL SCRIPTS:

1. *Write a shell script that delete all lines containing a specified word*

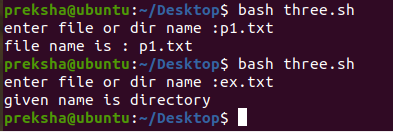
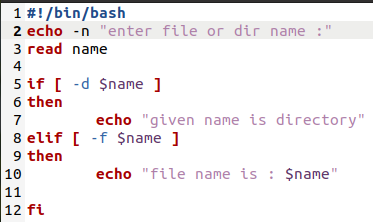


1. *Write a shell script that displays a list of all the files in the current directory*

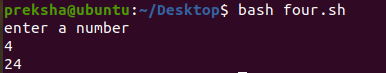


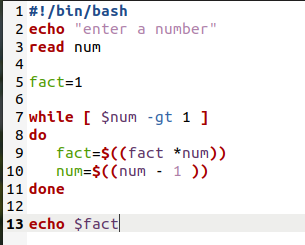


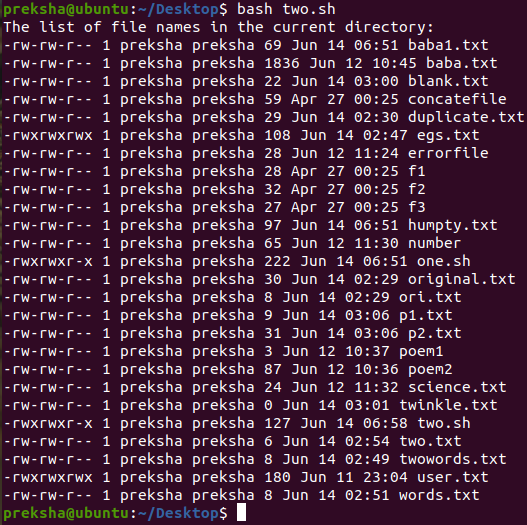
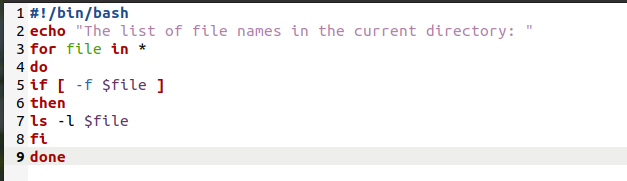
1. *Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or a directory and reports accordingly. Whenever the argument is a file or directory.*

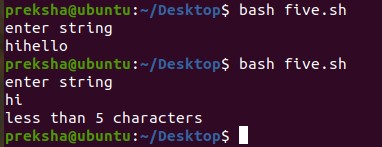
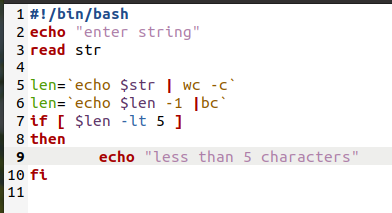


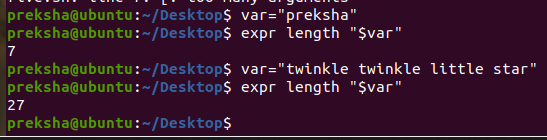
1. *Write a shell script to find the factorial of given integer*





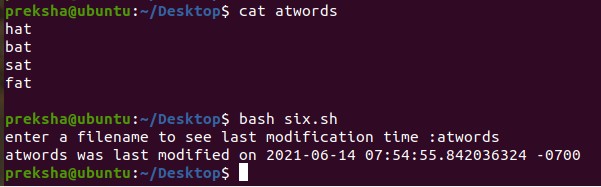
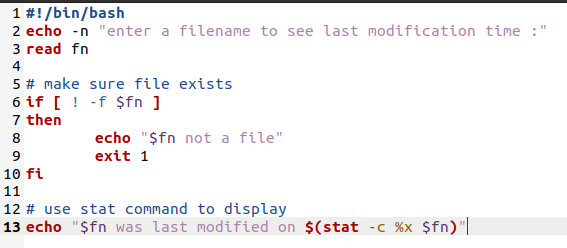
1. *write a shell script that list the all files in a directory.*
2. *Write a shell script that accepts a string from the terminal and echo a suitable message if it doesn’t have at least 5 characters including the other symbols.*

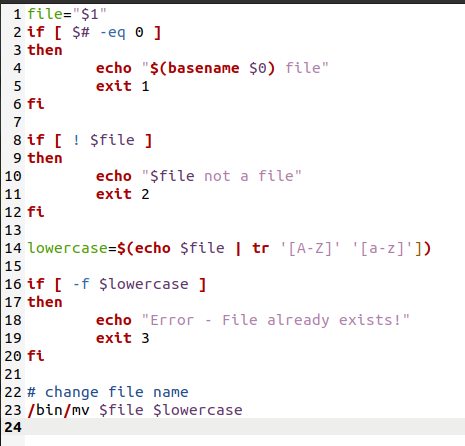


1. *Write a shell script to echo the string length of the given string as argument.*
2. *Write a shell script that accepts two directory names as arguments and deletes those files in the first directory which are similarly named in the second directly.Note: Contents should also match inside the files.*

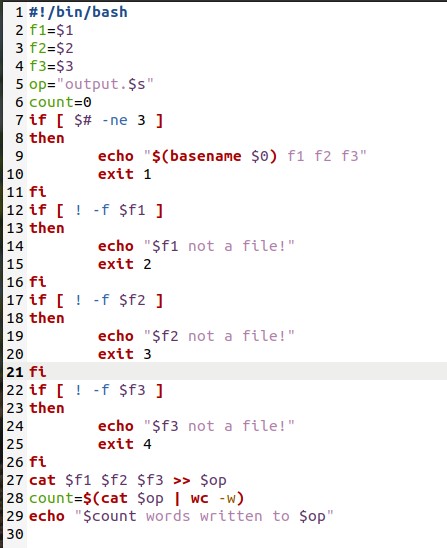
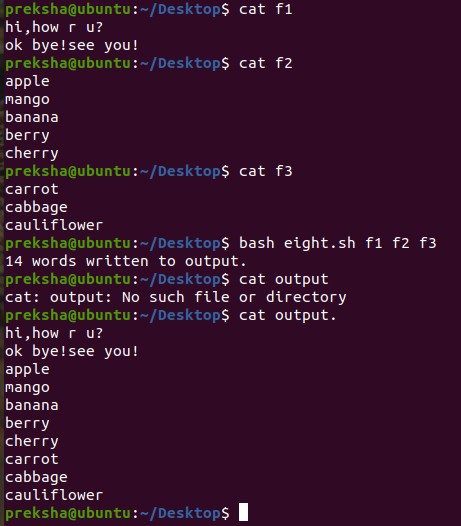


*EXTRA PROGRAMS:*

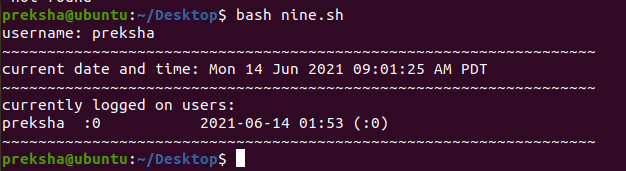
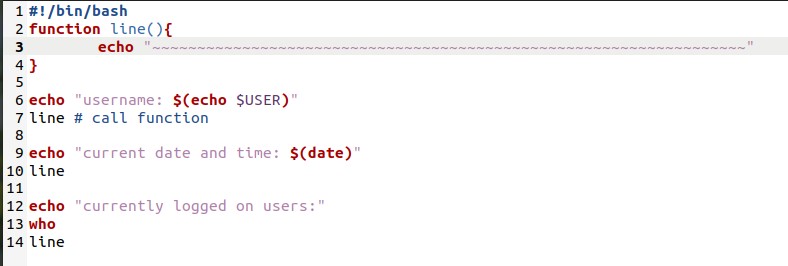
1. *Write a shell script that displays the last modification time of any file.*
2. *Write a shell script that changes the name of the files passed as arguments to lowercase.*



1. *Write a shell script to combine any three text files into a single file (append them in the order as they appear in the arguments) and display the word count.*



1. *Write a script called hello which outputs the following:*

* *your username*
* *the time and date*
* *who is logged on*

# FILE API ASSIGNMENT

### Program to retrieve the permission of a file

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h>

int main(int argc, char \*\*argv)

{

int result;

char \*filename = (char \*)malloc(512); if (argc < 2)

{

}

else

{

}

strcpy(filename, "/usr/bin/adb");

strcpy(filename, argv[1]);

result = access (filename, R\_OK); if ( result == 0 )

{

printf("%s has read permissions\n",filename);

}

else

{

printf("%s has no read permissions\n",filename);

}

result = access (filename, W\_OK); if ( result == 0 )

{

}

else

{

}

printf("%s has write permissions\n",filename);

printf("%s has no write permission\n",filename);

result = access (filename, X\_OK);

if ( result == 0 )

{

printf("%s is executable\n",filename);

}

else

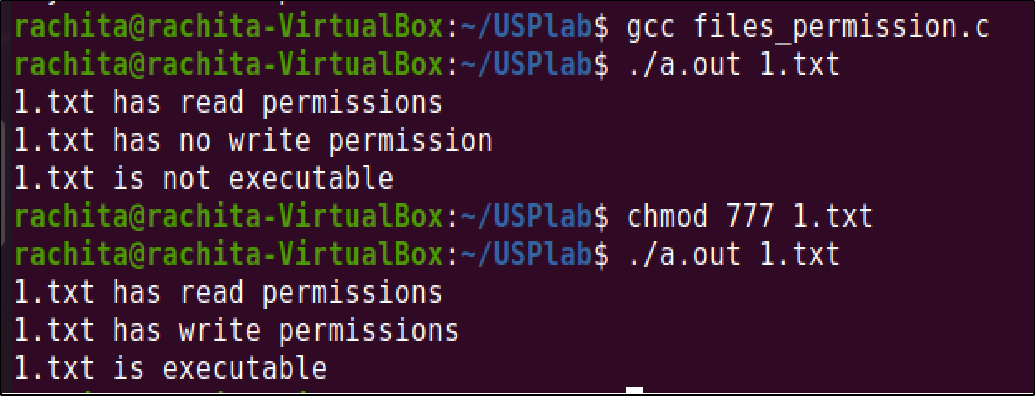
{

printf("%s is not executable\n",filename);

}

free(filename); return 0;

}



### Program to find the number of links

#include <stdio.h> #include <sys/stat.h>

int main ( int argc, char \*\* argv )

{

int i;

struct stat st; if(argc<=1)

{

}

else

{

printf("enter file name in the command line\n");

for (i = 1; i < argc; ++i)

{

if (stat(argv[i], &st) == -1)

{

}

else

{

}

}

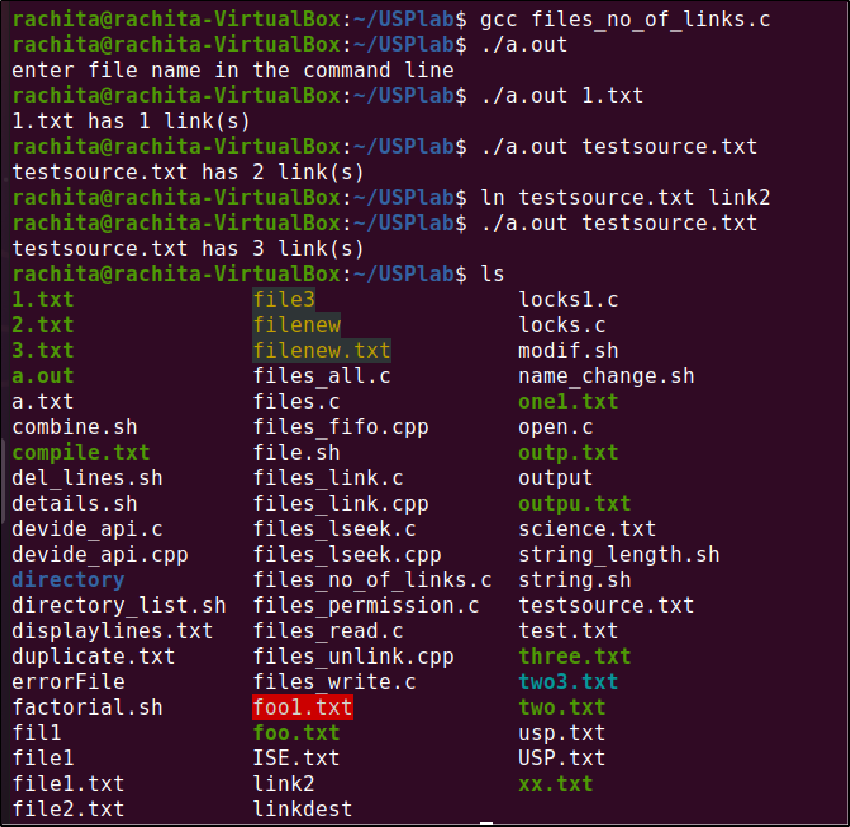
}

perror(argv[i]);

printf("%s has %ld link(s)\n", argv[i], st.st\_nlink);

return 0;

}



### Program to find the user and group name

#include <pwd.h> #include <grp.h> #include <sys/stat.h> #include<stdio.h>

int main(int argc, char \*\* argv )

{

struct stat info; if(argc<=1)

{

printf("enter file name in the command line\n");

}

else

{

for (int i = 1; i < argc; ++i)

{

stat(argv[i], &info); // Error check omitted struct passwd \*pw = getpwuid(info.st\_uid); struct group \*gr = getgrgid(info.st\_gid); printf("For file %s : \n",argv[i]);

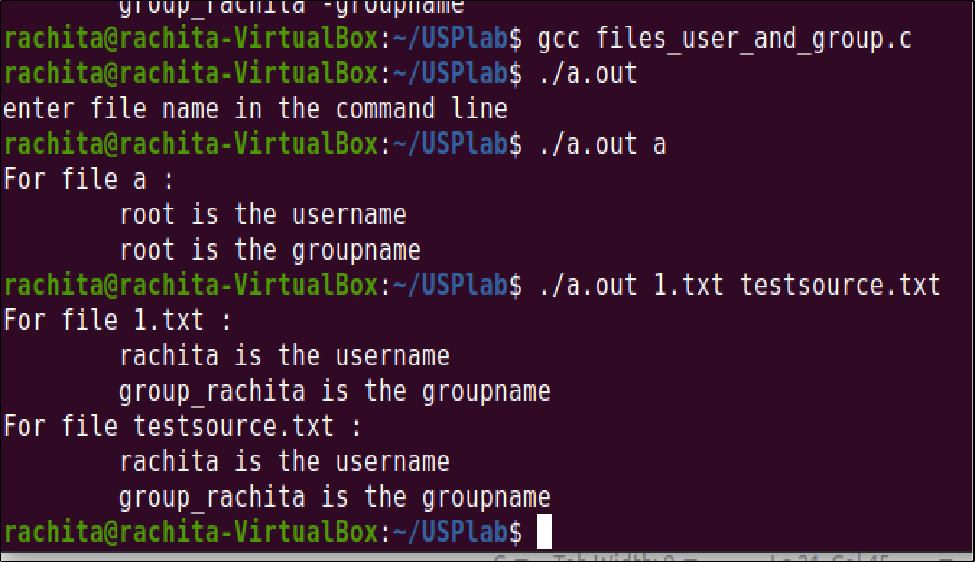
printf("\t%s is the username\n",pw->pw\_name); printf("\t%s is the groupname\n",gr->gr\_name);

}

}

return 0;

}



### Program to find the type of the file. The input will be given as a command line argument.

#include <sys/stat.h> #include <time.h>

#include <stdio.h> #include <stdlib.h>

int main(int argc, char \*argv[])

{

struct stat sb;

if(argc<=1)

{

}

else

{

printf("enter file name in the command line\n");

for (int i = 1; i < argc; ++i)

{

if (stat(argv[i], &sb) == -1)

{

perror(argv[i]); return 0;

}

printf("\nFor %s \n\ttype:",argv[i]);

switch (sb.st\_mode & S\_IFMT)

{

case S\_IFBLK: printf("block device\n");

break;

case S\_IFCHR: printf("character device\n");

break;

case S\_IFDIR: printf("directory\n"); break;

case S\_IFIFO: printf("FIFO/pipe\n"); break;

case S\_IFLNK: printf("symlink\n"); break;

case S\_IFREG: printf("regular file\n");

break;

case S\_IFSOCK: printf("socket\n"); break;

default: printf("unknown?\n");

exit(0);

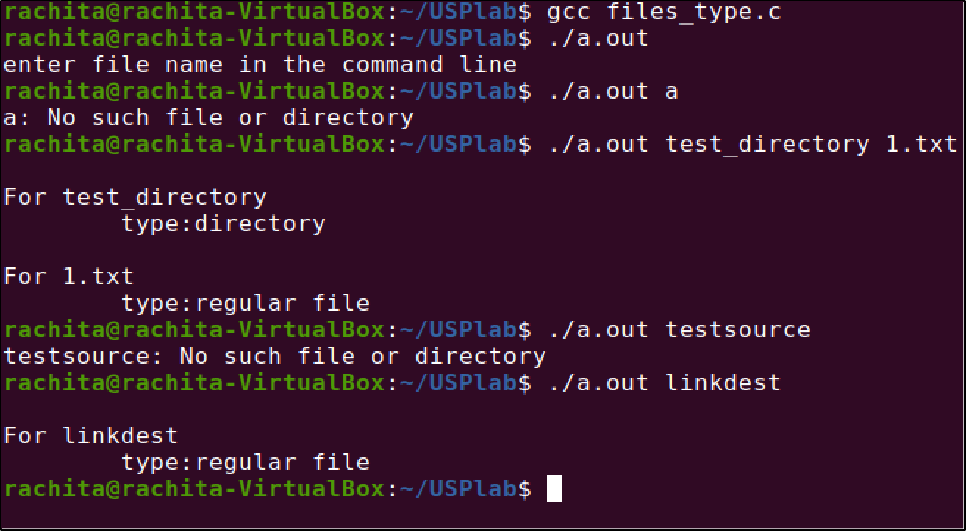
}

}

}

return 0;

}



### Program to create a file f1.txt. Write to the file “We are quarantined since one month“ and the move the file pointer to 50 bytes ahead and then write “we are not sure when lock down will be released”

Filename used-output1.txt #include<stdio.h> #include<unistd.h> #include<sys/types.h> #include<iostream> #include<fcntl.h> #include<cstring>

using namespace std; int main()

{

int fd;

char c1[]="We are quarantined since one month";

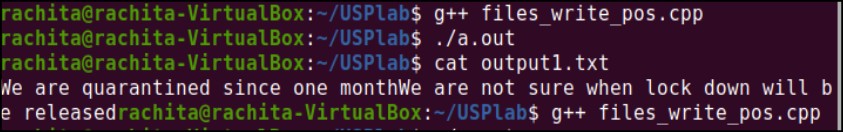
char c2[]="We are not sure when lock down will be released"; fd=creat("output1.txt",0777);

write(fd,c1,strlen(c1));

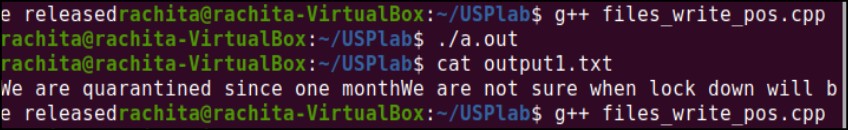
lseek(fd,50,SEEK\_CUR); write(fd,c2,strlen(c2)); return 0;

}

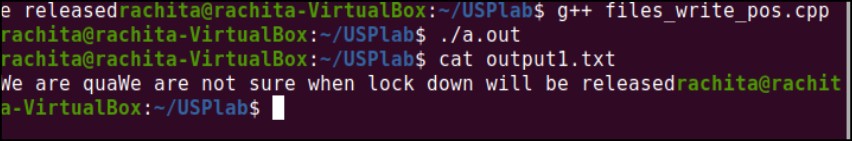
offset-50,SEEK\_CUR



offset-10,SEEK\_CUR



offset-10,SEEK\_SET



### Program to read the first 10 bytes of data from the file f1.txt and again read the file contents from 20 th byte.

#include<unistd.h> #include<fcntl.h> #include<sys/types.h> #include<sys/stat.h> #include<stdio.h>

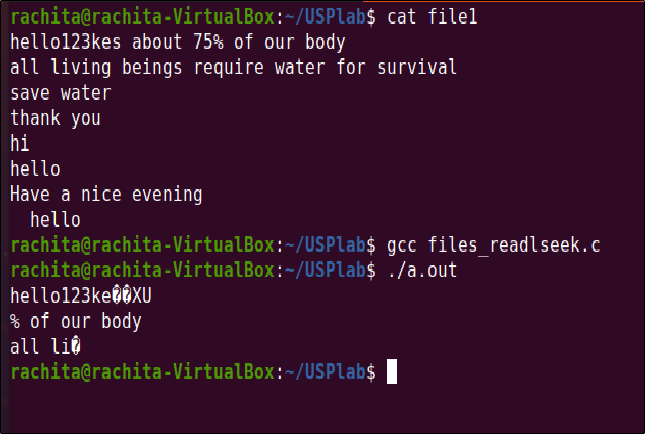
int main()

{

int n,f;

char buff[20],buff2[20]; f=open("file1",O\_RDWR); read(f,buff,10); printf("%s\n",buff); lseek(f,10,SEEK\_CUR); read(f,buff2,20); printf("%s\n",buff2); return 0;

}



# Problem based assignment on API's

#### Write a C program that takes one or more file/directory names as command line input and reports the following information on the file:

* *File type*

#### Number of links

* *Time of last access*

#### Read, write, and execute permission

#include <stdio.h> #include <stdlib.h> #include <time.h>

char \* error\_msg[] = {

"\nUsage: ./12filestat <file|dir> [<file|dir>]\n\n",

"\nFile does not exist !!\n\n",

"\nError doing 'stat' on file\n\n"

};

void print\_error(int msg\_num, int exit\_code, int exit\_flag);

int main(int argc, char \*argv[])

{

int i;

mode\_t file\_perm; struct stat file\_details;

char success\_msg[] = "\nCommand executed successfully\n\n"; if ( argc < 2 ) print\_error(0,2,1);

for ( i = 1; i < argc; i++ )

{

printf("\n%s\n%s\n%s\n","----------------",argv[i]," ");

if ( access(argv[i],F\_OK) == -1 )

{

print\_error(1,3,0);

continue; /\* Check the next file \*/

}

if ( lstat(argv[i],&file\_details) < 0 )

{

print\_error(2,4,0);

continue; /\* Check the next file \*/

}

if ( S\_ISREG(file\_details.st\_mode) )

printf("File type : Regular\n");

else if ( S\_ISDIR(file\_details.st\_mode) )

printf("File type : Directory\n");

else if ( S\_ISLNK(file\_details.st\_mode) )

printf("File type : Symbolic link\n");

else

printf("File type : Other");

printf("Number of links : %d\n", (int)file\_details.st\_nlink);

/\* Get the time of last access of the file \*/

printf("Time of last access : %s", ctime(&file\_details.st\_atime));

/\* Get the file permissions \*/

printf("File Permissions:\n");

file\_perm = file\_details.st\_mode & ~S\_IFMT;

printf("\tUser : ");

if ( file\_perm & S\_IRUSR ) printf("Readable, ");

else printf("Not readable, ");

if ( file\_perm & S\_IWUSR ) printf("Writable, ");

else printf("Not writable, ");

if ( file\_perm & S\_IXUSR ) printf("Executable\n");

else printf("Not executable\n");

printf("\tGroup : ");

if ( file\_perm & S\_IRGRP ) printf("Readable, ");

else printf("Not readable, ");

if ( file\_perm & S\_IWGRP ) printf("Writable, ");

else printf("Not writable, ");

if ( file\_perm & S\_IXGRP ) printf("Executable\n");

else printf("Not executable\n");

printf("\tOthers : ");

if ( file\_perm & S\_IROTH ) printf("Readable, ");

else printf("Not readable, ");

if ( file\_perm & S\_IWOTH ) printf("Writable, ");

else printf("Not writable, ");

if ( file\_perm & S\_IXOTH ) printf("Executable\n");

else printf("Not executable\n");

}

printf("%s", success\_msg); return 1;

}

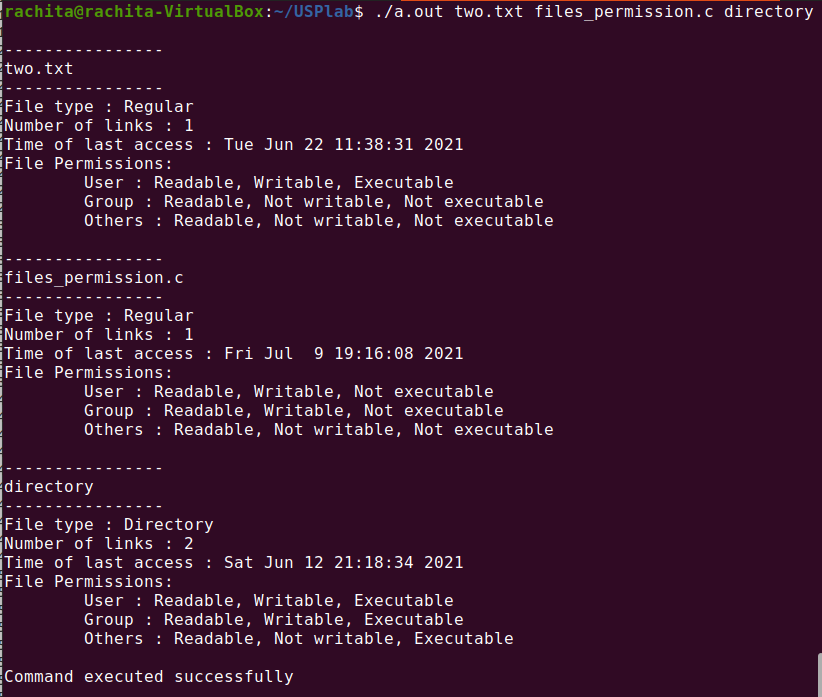
void print\_error(int error\_index, int exit\_code, int exit\_flag)

{

fprintf(stderr, "%s\n",error\_msg[error\_index]);

if (exit\_flag) exit(exit\_code);

}



1. *Write a program in C that illustrates how to execute two commands concurrently with a command pipe.*

#include<stdio.h> #include<fcntl.h> #include<stdlib.h> #include<unistd.h> void main()

{

int pfd[2],p; pipe(pfd); p=fork(); if(p==0)

{

}

else

{

}

}

close(pfd[0]); close(1); dup(pfd[1]);

execlp("ls","ls","-l",(char\*)0);

close(pfd[1]); close(0); dup(pfd[0]);

execlp("wc","wc",(char\*)0);



#### Write a C Program that makes a copy of a file using standard I/O and system calls

#include <stdio.h> #include <unistd.h> #include <fcntl.h> #include <stdlib.h>

int main(int argc, char \*argv[])

{

int f1, f2; char buff[50]; long int n;

if(((f1 = open(argv[1], O\_RDONLY)) == -1 || ((f2=open(argv[2], O\_CREAT | O\_WRONLY | O\_TRUNC,

0700))== 1)))

{

perror("problem in file"); exit(1);

}

while((n=read(f1, buff, 50))>0) if(write(f2, buff, n)!=n)

{

perror("problem in writing"); exit(3);

}

if(n==-1)

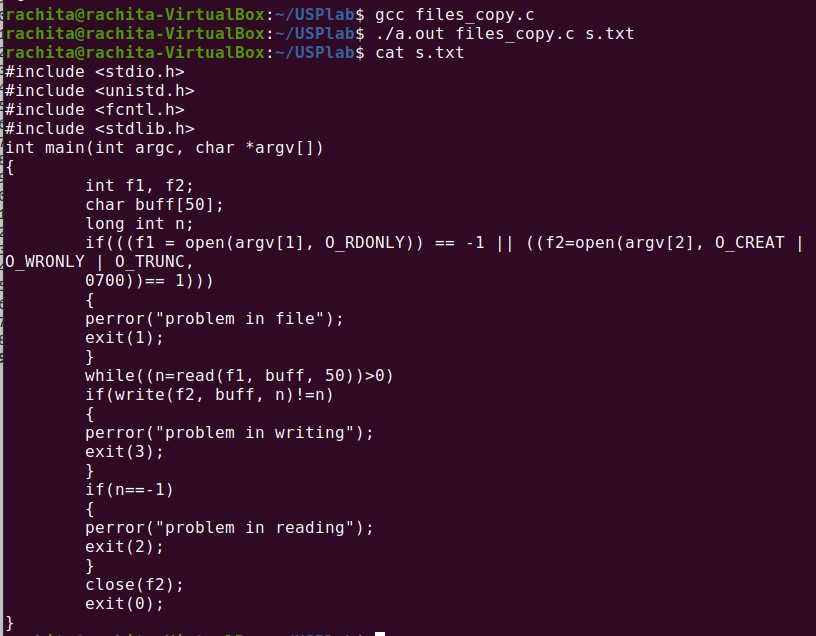
{

perror("problem in reading"); exit(2);

}

close(f2); exit(0);

}



### Implement in C the following Unix commands using system calls A)Cat

#include<sys/types.h> #include<sys/stat.h> #include<stdio.h> #include<unistd.h> #include<fcntl.h>

void main( int argc,char \*argv[3] )

{

int fd,i; char buf[2];

fd=open(argv[1],O\_RDONLY,0777); if(fd==-argc)

{

}

else

{

printf("file open error");

while((i=read(fd,buf,1))>0)

{

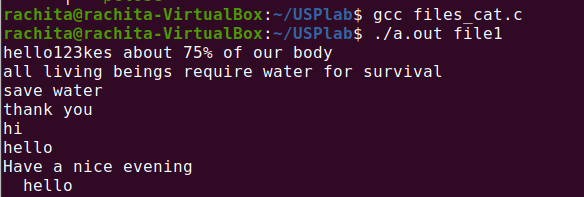
printf("%c",buf[0]);

}

close(fd);

}

}



*B)mv*

#include<sys/types.h> #include<sys/stat.h> #include<stdio.h> #include<fcntl.h> #include<unistd.h>

void main( int argc,char \*argv[] )

{

int i,fd1,fd2;

char \*file1,\*file2,buf[2]; file1=argv[1]; file2=argv[2];

printf("file1=%s file2=%s",file1,file2); fd1=open(file1,O\_RDONLY,0777); fd2=creat(file2,0777); while(i=read(fd1,buf,1)>0)

write(fd2,buf,1); remove(file1);

close(fd1); close(fd2);

}

### cp

#include<sys/types.h> #include<sys/stat.h> #include<stdio.h> #include<fcntl.h> #include<unistd.h>

void main( int argc,char \*argv[] )

{

int i,fd1,fd2;

char \*file1,\*file2,buf[2]; file1=argv[1]; file2=argv[2];

printf("file1=%s file2=%s",file1,file2); fd1=open(file1,O\_RDONLY,0777); fd2=creat(file2,0777); while(i=read(fd1,buf,1)>0)

write(fd2,buf,1);

close(fd1); close(fd2);

}



### ln

#include<stdio.h> #include<unistd.h> #include<iostream> using namespace std;

int main(int argc,char\*argv[])

{

if(argc!=3)

{

cerr<<"usage:"<<argv[0]<<"<src\_file><dest\_file>\n"; return 0;

}

if(link(argv[1],argv[2])==-1)

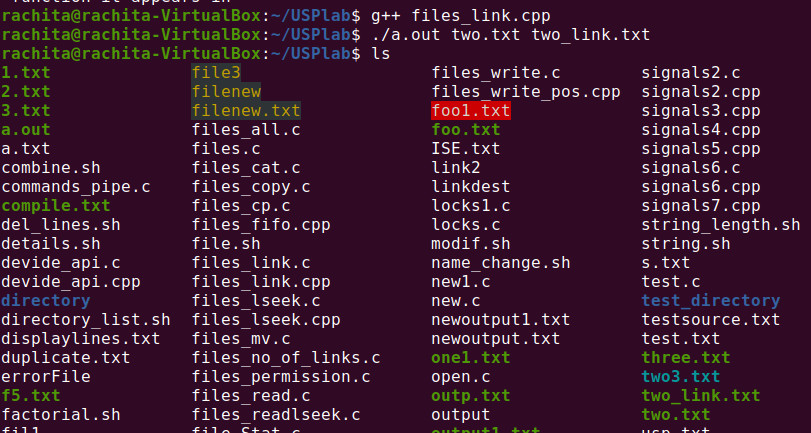
{

perror("link"); return 1;

}

return 0;

}



### Write a C program to list for every file in a directory, its inode number and file name.

#include<stdlib.h> #include<stdio.h> #include<string.h>

void main(int argc, char \*argv[])

{

char d[50]; if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"-i ");

strcat(d,argv[1]);

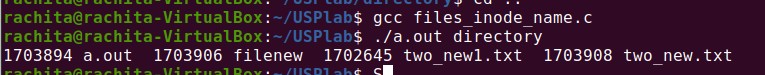
system(d);

}

else

printf("\nInvalid No. of inputs");

}



### Write a C Program that demonstrates redirection of standard output to a file .EX:ls>f1.

#include<stdlib.h> #include<stdio.h> #include<string.h>

void main(int argc, char \*argv[])

{

char d[50]; if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"> ");

strcat(d,argv[1]);

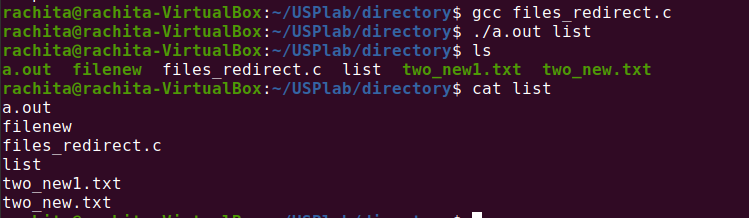
}

else

}

system(d);

printf("\nInvalid No. of inputs");



### Write a C program to create a child process and allow the parent to display “parent” and the child to display “child” on the screen

#include<stdio.h> #include<sys/wait.h> #include<stdlib.h> #include<unistd.h> int main(void)

{

int pid;

int status;

printf("Printing based on pid:\n"); pid = fork();

if(pid == -1)

{

perror("bad fork"); exit(1);

}

if (pid == 0)

printf("Child\n");

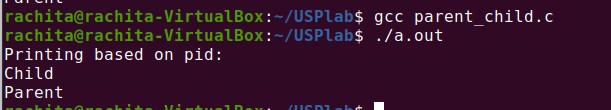
else

{

}

}

wait(&status); printf("Parent\n");



### Write a C program to create a Zombie process.

#include <stdlib.h> #include <sys/types.h> #include<stdio.h> #include <unistd.h>

int main ()

{

int pid\_t,child\_pid;

child\_pid = fork (); if (child\_pid > 0) { sleep (60);

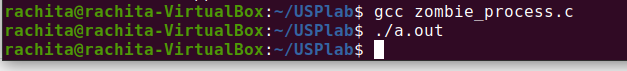
}

else { exit (0);

}

return 0;

}



### Write a C program that illustrates how an orphan is created.

#include<stdio.h> #include<stdlib.h> #include<unistd.h> int main()

{

int pid; pid=fork(); if(pid==0)

{

printf("IAM CHILD PROCESS,MY PROCESS ID IS: %d",getpid());

printf("\n THE CHILDS'S PARENT PROCESSS ID IS : %d\n",getppid()); sleep(10);

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*AFTER 10 SECUNDS\*\*\*\*\*\*\*\*\*\*"); printf("\n IAM CHILD PROCESS,MY PROCESS ID IS: %d",getpid());

printf("\n THE CHILDS'S PARENT PROCESS ID IS :%d\n",getppid());

}

else

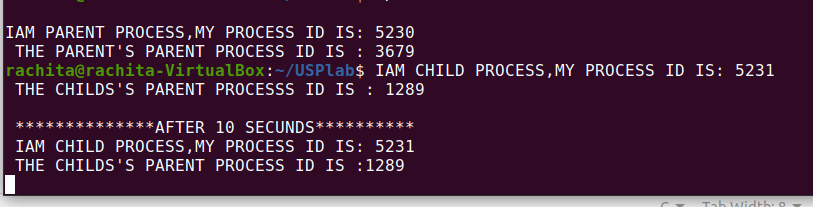
{

printf("\nIAM PARENT PROCESS,MY PROCESS ID IS: %d",getpid());

printf("\n THE PARENT'S PARENT PROCESS ID IS : %d\n",getppid());

}

}



### Write a program that illustrates how to execute two commands concurrently with a command

#include<stdio.h> #include<fcntl.h> #include<stdlib.h> #include<unistd.h>

void main()

{

int pfd[2],p; pipe(pfd); p=fork(); if(p==0)

{

}

else

{

}

}

close(pfd[0]); close(1); dup(pfd[1]);

execlp("ls","ls","-l",(char\*)0);

close(pfd[1]); close(0); dup(pfd[0]);

execlp("wc","wc",(char\*)0);



### Write a C programs that illustrate communication between two unrelated processes using named pipe.

#include<stdio.h>

#include<stdlib.h> #include<errno.h> #include<unistd.h> int main()

{

int pfds[2]; char buf[30];

if(pipe(pfds)==-1)

{

perror("pipe"); exit(1);

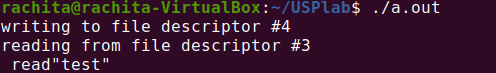
}

printf("writing to file descriptor #%d\n", pfds[1]); write(pfds[1],"test",5);

printf("reading from file descriptor #%d\n ", pfds[0]); read(pfds[0],buf,5);

printf("read\"%s\"\n" ,buf);

}



#### Write a C program that illustrates suspending and resuming processes using signals.

#include<stdio.h> #include<signal.h>

#include<stdlib.h> #include<unistd.h> void alarms(int signo)

{

printf("alarm function. signo = %d\n,",signo);

}

int main()

{

if(signal(SIGALRM,alarms)==SIG\_ERR) printf("Signal error");

alarm(5);

for(int i=0;i<10;i++)

printf(" i=%d \n",i); printf("main program terminated\n"); return 0;

}



#### Write a separate program using signal system call to catch the following signals.

* *SIGSEGV*

#### SIGINT

* *SIGFPE*

#### SIGALRM (use alarm system call)

* *SIGALRM (use setitimer system call)*

#### SIGVTALRM (use setitimer system call)

* *SIGPROF (use setitimer system call)*

#include <iostream> #include <signal.h> using namespace std; int main()

{

sigset\_t sigset; sigemptyset(&sigset); sigaddset(&sigset, SIGINT); sigaddset(&sigset, SIGUSR1); sigaddset(&sigset, SIGQUIT); sigaddset(&sigset, SIGALRM); sigaddset(&sigset, SIGFPE); sigaddset(&sigset, SIGSEGV);

if (sigprocmask(SIG\_SETMASK, &sigset, 0) == -1)

perror("sigprocmask");

/\*initialize set\*/

if (sigprocmask(0, NULL, &sigset) < 0) perror("sigprocmask error");

if (sigismember(&sigset, SIGINT)) cout<<"SIGINT"<<endl; if (sigismember(&sigset, SIGFPE)) cout<<"SIGFPE "<<endl;

if (sigismember(&sigset, SIGQUIT)) cout<<"SIGQUIT "<<endl; if (sigismember(&sigset, SIGUSR1)) cout<<"SIGUSR1 "<<endl; if (sigismember(&sigset, SIGUSR1)) cout<<"SIGUSR1 "<<endl;

if (sigismember(&sigset, SIGALRM)) cout<<"SIGALRM "<<endl; if (sigismember(&sigset, SIGSEGV)) cout<<"SIGSEGV "<<endl;

}

