CREATING A BASH LIKE SHELL IN LINUX FROM SCRATCH

By-
SWATI SINGH(18BCE1018)
R. HARINI(18BCE1010)
SHRADDHA NAIR(18BCE1070)

A project report submitted to-
Dr. RENUKA DEVI
SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
in partial fulfillment of the requirements for the course of
CSE2005 –OPERATING SYSTEMS
in
B.Tech. COMPUTER SCIENCE ENGINEERING

CONTENT

ABSTRACT:

This project aims at creating a bash like shell. GNU Bash or simply Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell which was first released in the year 1989. This project aims at creating a bash like shell that has its own commands and functionalities which have been created from scratch using the C programming language. The commands include various file handling commands, like touch and wc, process commands like ps, logging commands like mtime and log and some mathematical computation commands like expt and fact amongst others. The shell reads in the commands from the terminal and performs the required task by splitting the line into tokens and comparing the tokens with the names of the commands. This is carried out by the execution functions included in the C code. Overall, the shell gives a bash like experience, with about thirty executable commands.

INTRODUCTION:

The BASH (Bourne Again Shell) is a Unix shell and command language which is used by people all over the world. Our project aims at recreating the bash shell with many of its old functionalities as well as some new ones. This shell makes use of the C programming language for creating functions which carry out the work of the commands. The shell implements almost all the important commands of the original BASH shell. The commands when entered in the shell are read and broken down into tokens, these tokens are then sent to command functions based on the name of the command entered, and are executed based on the arguments entered. This is taken care of by the lsh_read_line, lsh_split_line, lsh_launch, and lsh_execute functions which carry out the process step by step. Some of the commands implemented include:Mtime, Find, Lsh_wc, Lsh_uniq, Lsh_expt, Lsh_diff.

MODULE DESCRIPTION:

Bash is a command processor that typically runs in a text window where the user types commands that cause actions.

Some of the commands implemented are:

1. **Cat command:** One of the most frequently used command in Linux. It allows to create single or multiple files, view content of a file and append text to the file. It is also used to create a file and add text.

2. **Add command:** This command adds the content of the first file to the second file.

3. **Grep command:** The grep command is used to search a specific text in the given file. It outputs the entire line where the word has been found.

4. **Find command:** The entire Linux filesystem is in the form of a tree structure. Using a recursive function, one can implement the command to find a file in the given directory. It outputs the relative path of the file from the given directory as well.

5. **Mv command:** This command moves the contents of one file to another one and deletes the first file. This is also used to rename a file.

6. **Ps command:** On parsing the /proc directory, the command can be used to display information about all the processes on the system such an Pid, Name, Status, etc.

7. **Touch command:** This command creates a new file without adding any content. The file is created for later purposes.

8. **Mtime command:** The command is used to modify the timestamp of a particular file.

9. **Log command:** This command maintains history of all the commands entered in the shell along with the timestamp and displays it to the user. Before exiting the shell, it stores the buffer content to a log file.

10. **Cd command:** It is used to change the working directory to the given directory.

11. **Help command:** It displays information about all the commands available in the shell.

12. **Echo command:** The echo command displays any text that was entered with the command back to the user.

13. **Ls command:** This command lists out all the files present in the working directory.

14. **Wc command:** The command is used for printing line, word,character and byte counts for file. It can return the number of lines in a file, the number of characters in a file and the number of words in a file. It can also return the above information for two specified files and print the total count. The longest line of the file can also be printed by this command.

15. **Sort command:** The sort command is used to sort the contents of the given file in ascending or descending order and accordingly it will either display the result on the shell or write it into the specified file.

16. **Rm command:** This command is used to remove a file from the working directory.

17. **Clear command:** The clear commands clears the entire shell console.

18. **Cmp command:** This command is used to compare the contents of two files. It also displays the difference in the characters and byte count.

19. **Clclog command:** This command clears the log buffer and the log file content. It can also clear the log content of a particular date.

20. **Viewlog Command:** This command allows the user to view the entire content of the log file or the contents of a specific date.

21. **Uniq command:** It is used to report or filter repeated lines in a file. It also shows the count of occurrences of the lines in the file.

22. **Head command:** The head command displays the specified number of lines from the start of the file.

23. **Tail command:** The head command displays the specified number of lines from the end of the file.

24. **Diff command:** This command is used to display the differences in the files by comparing the files line by line. Unlike cmp, it tells us which lines in one file has to be changed to make the two files identical.

25. **Expt command:** This command is used to solve a simple numerical expression. This is evaluated by converting the expression to postfix expression.

26. **Fact command:** This command outputs all the prime factors of the given number.

27. **Tac command:** The tac command outputs the contents of a file in the reverse order.

28. **Exit command:** This command is used to exit the created shell.

SOFTWARE USED:

Bash is a UNIX shell which is used widely as the default login shell for most Linux distributions. Hence in order to implement this project, we require any version of Linux OS installed on the device using any VMware.

The commands of the shell are implemented using a large C program with various functions for each command. This requires basic knowledge of C programming and system calls. Since the Linux distribution comes with an inbuilt C compiler, one does not need to install it separately.

The Linux terminal is required to run the C program and view and interact with the implemented shell as well.

SAMPLE CODING:-

1) cd

```
int lsh_cd(char **args){
  if (args[1] == NULL) {        fprintf(stderr, "OS_Shell: expected argument to \"cd\"\n");
  } else {          if (chdir(args[1]) != 0) {perror("OS_Shell");       }  }
  return 1;}
```

2) ls

```
int lsh_ls(char ** args){
        struct dirent **namelist;
        int n;
        if (args[1]==NULL){            n=scandir(".", &namelist, NULL, alphasort);      }
        else{              n=scandir(args[1], &namelist, NULL, alphasort);          }
        if (n<0){         perror("scandir");
        exit(EXIT_FAILURE);       }
        else{              while (n--){              printf("%s\n", namelist[n]->d_name);
        free(namelist[n]);      }
        free(namelist);          }
        return 1;}
```

3) echo

```
int lsh_echo(char ** args){

        int k=1;

        while (args[k]!=NULL){      printf("%s ", args[k]);

        k++;    }

printf("\n");

return 1;}
```

4) help

```
int lsh_help(char **args){

  int i;

  printf("OS Project Shell\n");

  printf("Type program names and arguments, and hit enter.\n");

  printf("The following are built in:\n");

  for (i = 0; i < num_of_builtins(); i++) {     printf("  %s\n", builtin_str[i]);  }

   printf("Use the man command for information on other programs.\n");

  return 1;}
```

5) log

```
void log_entry(char* line, time_t t){

        strcpy(log_buffer[l][0], line);

        strcpy(log_buffer[l][1], ctime(&t));

        L++;}


int lsh_log(char ** args){
        printf("Displaying the log:\n");

        for (int i=0; i<=l;i++){        printf("%s\t %s\n",log_buffer[i][0], log_buffer[i][1]);}

return 1;}
```

6) touch

```
int touch(char ** args){ //touch filename

        int fdold, count;

        fdold=open(args[1], O_RDWR | O_APPEND);

        if (fdold==-1){         mode_t mode=S_IRUSR | S_IWUSR| S_IRGRP | S_IROTH;
//giving permissions to the file

        creat(args[1], mode);

        return 1;       }

        else{           printf("File already exists\n");        }

        return 1;}
```

7) mtime

```
int mtime(char ** args){ //mtime filename

        int retvalue;

        retvalue=utime(args[1],NULL); //changes access and modification times of the file

        if(retvalue==0){        printf("Timestamp modified\n");   }

        return 1;}
```

8) find

```
        void find_file(char *basePath, char *fileName){ //using a recursive function to
browse the hierarchical filesystem
```

```
        char path[1000];

        struct dirent *dp;

        DIR *dir=opendir(basePath);

        if (!dir){return;}

        while((dp=readdir(dir))!=NULL){          if(strcmp(dp->d_name,".")!=0 &&
strcmp(dp->d_name, "..")!=0){               if (strcmp(dp->d_name,
fileName)==0){printf("Path:%s\n",basePath);}

        else{           strcpy(path,basePath);

                strcat(path, "/");

                strcat(path, dp->d_name); //concat to browse the path

                find_file(path, fileName);          }       }       }

        closedir(dir);}


int find(char ** args){  find dirname filename

        find_file(args[1], args[2]); //call function to recursively browse

        return 1;}
```

9) ps

```
int ps(char ** args){

        DIR *dp ;

        struct dirent *entry; //creating obj to hold the contents of the directory

        if ((dp=opendir("/proc"))==NULL){       printf("Cannot open directory");

        return 0;       }

        chdir("/proc");       // the folder where info about the process is stored
        while ((entry=readdir(dp))!=NULL){        int flag=0;

        for (int i=0;entry->d_name[i]!='\0';i++){              if
(entry->d_name[i]=='0'||entry->d_name[i]=='1'||entry->d_name[i]=='2'||entry->d_name[i]=
='3'||entry->d_name[i]=='4'||entry->d_name[i]=='5'||entry->d_name[i]=='6'||entry->d_nam
e[i]=='7'||entry->d_name[i]=='8'||entry->d_name[i]=='9'){flag++;}

        }         //check if the file name contains only digits or not

        if (flag>0){    printf("Pid:%s\t", entry->d_name);

        int fd, r, j=0;

        char temp, line[100];

        strcat(entry->d_name, "/status");  //file where the info about each process is stored

        if ((fd=open(entry->d_name,O_RDONLY))!=-1){
while((r=read(fd,&temp, sizeof(char)))!=0 && temp!='\n'){

                        line[j]=temp;

                        j++;              }                 }

        printf("%s\n", line);           }                 }

        closedir(dp);

        return 1;}
```

10) add

```
int lsh_add(char ** args){    //add file1 file2

        int fdold1,fdold2,count;

        char buffer[2048];  //character buffer to store the bytes

        fdold1=open(args[1], O_RDWR);

        if(fdold1==-1)

        { printf("cannot open file1");    return 0;}
        fdold2=open(args[2], O_RDWR| O_APPEND);

        if(fdold2==-1)

        { printf("cannot open file2");   return 0; }

        while((count=read(fdold1, buffer, sizeof(buffer)))>0){ //printf("%s", buffer);

        size_t n=strlen(buffer);

        if ((write(fdold2, buffer , n))==-1){printf("Error while writing\n");  return 1; } }

        close(fdold2);

        fdold2=open(args[2], O_RDWR| O_APPEND); //pointer at the start again

        if(fdold2==-1)

        { printf("cannot open file2");  return 0; }

        while((count=read(fdold2, buffer, sizeof(buffer)))>0){

        printf("%s", buffer); }

        close(fdold1);

        close(fdold2);

        return 1; }
```

11) grep

```
int compare(const char* X, const char *Y){

        while( *X && *Y){

        if(*X!=*Y)return 0;

        X++;

        Y++; }

        return (*Y == '\0'); }
```

```
const char* str_in_str(const char* X, const char* Y){   while(*X != '\0'){

          if ((*X == *Y) && compare(X, Y))return X;

          X++;  }

          return NULL;  }

int lsh_grep( char ** args){  //grep expr filename

          int fd, r, j=0;

          char temp, line[100];

          if ((fd=open(args[2], O_RDONLY))!=-1){  while((r=read(fd,&temp,
sizeof(char)))!=0){ if (temp!='\n'){    line[j]=temp;

          j++;}   else  {     if(str_in_str(line, args[1])!=NULL)          //checking for a string
inside a string

          printf("%s\n", line);

          memset(line, 0, sizeof(line));

          j=0;  }  }   }  return 1;  }
```

12) mv

```
int lsh_mv( char ** args){    //mv file1 file2

          int i, fd1, fd2;

          char *file1, *file2, buf[2];

          file1=args[1];

          file2=args[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);
        return 1;  }
```

13) cat

```
int cat_read(char ** args){
        int fdold,count;
        char buffer[2];
        fdold=open(args[1], O_RDONLY);
        if (fdold==-1){
        printf("Cannot open file\n");
        return 1; }
        while((count=read(fdold, buffer, 1))>0){     printf("%s", buffer);      }
        printf("\n");
        return 1;  }
int cat_append( char ** args){        //cat a filename
        int fdold, count;
        fdold=open(args[2], O_RDWR | O_APPEND);
        if (fdold==-1){   mode_t mode=S_IRUSR | S_IWUSR| S_IRGRP | S_IROTH;
```

```
          creat(args[2], mode);

          fdold=open(args[2], O_RDWR | O_APPEND);

          if (fdold==-1){   printf("Cannot open the created file");

          return 1;}  }

          char buffer[1024];

          printf("Enter the data to be appended:");

          scanf("%s", buffer);

          size_t n=strlen(buffer);

          if ((write(fdold, buffer,n))==-1){  printf("Error while writing\n");}

          else{    close(fdold);   }

          return 1;  }

int lsh_cat(char ** args){     //cat filename OR cat a filename

          if (args[1]==NULL){  printf("Please enter a file name\n");        }

          else if (strcmp(args[1], "a")==0){  cat_append(args);   }

          else{   cat_read(args);   }  return 1;  }
```

14) tac

```
int lsh_tac(char ** args)

{         FILE *fp1;     int cnt = 0;

          int i   = 0;

          fp1 = fopen(args[1],"r");

          if( fp1 == NULL )  {   printf("\n%s File can not be opened : \n",args[1]);   return -1;
}
```

```
            //moves the file pointer to the end.

           fseek(fp1,0,SEEK_END);

                      //get the position of file pointer.

           cnt = ftell(fp1);

           while( i < cnt ){   i++;

            fseek(fp1,-i,SEEK_END);

            printf("%c",fgetc(fp1));  }

           printf("\n");

           fclose(fp1);

           return 1;  }
```

15) fact

```
int lsh_fact(char ** args)

{  int n;

           sscanf(args[1], "%d", &n);   //converting string to integer

                                                                     // Print the number of 2s
that divide n

           while (n % 2 == 0) {   printf("%d ", 2);

            n = n / 2;  }

   printf("\n");

  int i;

           // n must be odd at this point.  So we can skip

           // one element (Note i = i +2)
```

```
          for (i = 3; i <= sqrt(n); i = i + 2) {

           // While i divides n, print i and divide n

           while (n % i == 0) {  printf("%d ", i);

         n = n / i;   }  }

            // This condition is to handle the case when n

            // is a prime number greater than 2

          if (n > 2)     printf("%d ", n);

printf("\n");

return 1;  }
```

16) expt

```
int precedence(char a,char b)

{                                          //returns true if precedence of operator a is more or equal to
than that of b

          if(((a=='+')||(a=='-'))&&((b=='*')||(b=='/')))

          return 0;

          else

          return 1;  }

int operate(int a,int b,char oper)

{  int res=0;

switch(oper)   {   case '+':res=a+b;break;

          case '-':res=a-b;break;

          case '*':res=a*b;break;
          case '/':res=a/b;break;  }

return res;                          //return result of evaluation   }

int lsh_expt(char ** args)  {char stack[20];                    //store the postfix expression

int top=-1;                          //top of postfix stack

int topOper=-1;                              //top of operator stack

char expr[20];

strcpy(expr,args[1]);

char topsymb,operatorStack[20];

int ctr=0;

while(expr[ctr]!='\0')                    //converting to postfix{                           //read till the
end of input

if(expr[ctr]>='0'&&expr[ctr]<='9')

stack[++top]=expr[ctr];                   //add numbers straightaway

else{   while(topOper>=0&&precedence(operatorStack[topOper],expr[ctr]))   {
          //check for the operators of higher  precedence and then add them to stack

topsymb=operatorStack[topOper--];

stack[++top]=topsymb;   }

operatorStack[++topOper]=expr[ctr];   }

ctr++;  }

while(topOper>=0)              //add remaining operators to stack

stack[++top]=operatorStack[topOper--];

char oper;

int operand1,operand2;
```

```
ctr=0;

int result[2];              //stack to keep storing results

int rTop=-1;                          //top of result stack

while(stack[ctr]!='\0')  {  oper=stack[ctr];

if(oper>='0'&&oper<='9')

result[++rTop]=(int)(oper-'0');                  //add numbers

else  {             //if an operator is encountered then pop twice and push the result of
operation to the stack

        operand1=result[rTop--];

        operand2=result[rTop--];

        result[++rTop]=operate(operand2,operand1,oper);  }

ctr++;  }

printf("%d\n",result[0]);

return 1;  }
```

17) head

```
int lsh_head(char ** args)

{  FILE  *file;

char line[100];

 int count = 0;

      // initialise file pointer to read

        file  =  fopen(args[1],"r");

       // read line by line

        while(fscanf(file , "%[^\n]\n" , line)!=EOF)  {  // break after 10 lines
```

```
                        if(count  == 10)  {   break;   }
                        else  {   printf("%s\n" ,  line);   }
                   count++;  }
         fclose(file);
return(1);  }
```

18) tail

```
struct stack {char strings[100];};
 int lsh_tail(char ** args)
{   // stucture initialisation
          struct stack s[100];
   FILE *file;
          char line[100];
          int n,count=0, i=0;
         file  = fopen(args[1] , "r");  // reading line by line and push to stack
          while(fscanf(file , "%[^\n]\n" , line)!=EOF)  {  strcpy(s[i].strings , line);
                        i++;     n=i;  }
          // pop line by line
          for(i=n;i>0;i--)  {  // last 10 lines    if(count == 10)       break;
                        else          printf("%s\n" , s[i].strings);
                        count++;  }
return(1);   }
```

19) diff

```
  int lsh_diff(char ** args)   {   struct lines st[20];
  FILE *file1,*file2;
  char line1[100],line2[100];
  int line_count1=0,flag2[10],flag1[10],line_count2=0;
  int i=0,j=0,k=0,n=0,m=0,o=0;
  file1 = fopen(args[1],"r");
  file2 = fopen(args[2],"r");
  while(1)   {   line_count1++;
line_count2++;
    if(fscanf(file1,"%[^\n]\n",line1)!=EOF && fscanf(file2,"%[^\n]\n",line2)!=EOF) {
    if(strcmp(line1,line2) == 0)    continue;
    else{   if(line1 != NULL){   strcpy(st[i].stack_lines1 , line1);
                        flag1[m]=line_count1;
                        m++;   }
              if(line2 != NULL){   strcpy(st[i].stack_lines2 , line2);
              flag2[o]=line_count2;
              o++;  } } }
     else if(fscanf(file1,"%[^\n]\n",line1)!=EOF){strcpy(st[i].stack_lines1 , line1);
    flag1[m]=line_count1;
    m++;   }
     else if(fscanf(file2,"%[^\n]\n",line2)!=EOF){   strcpy(st[i].stack_lines2 , line2);
    flag2[o]=line_count2;
```

```
            o++;  }   else       break;
             i++;
             n++; }
           for(i=0;i<m;i++){
            printf("%d,",flag1[i]);  }
           printf("c");
           for(i=0;i<o;i++)  {
            printf("%d,",flag2[i]);  }
           printf("\n");
           for(i=0;i<n;i++)  {
            printf("%s\n",st[i].stack_lines1);  }
           printf("---\n");
           for(i=0;i<n;i++)  {
            printf("%s\n",st[i].stack_lines2);  }
           fclose(file1);
           fclose(file2);
return(1);}
```

20) wc

```
int lsh_wc(char **args)  //Calculate the character,word,line,space,byte count in file  {
  int character_count=0;  //Initialiseing the counts
  int space_count=0;
  int word_count=0;
  int line_count=0;

  int byte_count=0;

  char ch;

int character_count1 = 0;

  int space_count1 = 0;

  int word_count1 = 0;

  int line_count1 = 0;

  int byte_count1=0;

  char ch1;

   FILE *fp;

  FILE *fp1;

  if(args[1] == NULL){   fprintf(stderr, "OS_Shell: expected argument to \"wc\"\n"); }

  else {  if(strstr(args[1],"txt")!=NULL)  //wc "filename".txt  {  fp = fopen(args[1],"r");

            while((ch=fgetc(fp))!=EOF)  //Reading each character of the file and finding count {

            character_count++;

            if(ch == ' ')  {

                        space_count++;

            word_count++;  }   if(ch == '\n')   {

                        line_count++;  } }   fclose(fp);

                        if(args[2]==NULL)  {  printf("Line_count = %d\n",line_count);

                        printf("Word_count = %d\n",word_count+2);

                        printf("Space_count = %d\n",space_count);
```

```
                printf("Character_count = %d\n",character_count);

                printf("Byte_count=%d\n",(character_count+space_count+1));

                printf("File_name= %s\n",args[1]);  }

        else if(strstr(args[2],"txt")!=NULL)  //wc "file1".txt "file2".txt  {  fp1 =
fopen(args[2],"r");    while((ch1=fgetc(fp1))!=EOF)   {  character_count1++;

        if(ch1 == ' ')   { space_count1++;

                word_count1++;  }   if(ch1 == '\n')  {

                line_count1++;  }  }  fclose(fp1);   printf("Line_count = %d\t",line_count);

                printf("Word_count = %d\t",word_count+2);

                printf("Space_count = %d\t",space_count);

                printf("Character_count = %d\t",character_count);

                printf("Byte_count=%d\t",(character_count+space_count+1));

                printf("File_name= %s\n",args[1]);

                printf("Line_count = %d\t",line_count1);

                printf("Word_count = %d\t",word_count1+2);

                printf("Space_count = %d\t",space_count1);

                printf("Character_count = %d\t",character_count1);

                printf("Byte_count=%d\t",(character_count1+space_count1+1));

                printf("File_name= %s\n",args[2]);

                printf("Total:\n");

                printf("Line_count = %d\t",line_count+line_count1);

                printf("Word_count = %d\t",word_count+4+word_count1);

                printf("Space_count = %d\t",space_count+space_count1);
```

```
                    printf("Character_count = %d\t",character_count+character_count1);
printf("Byte_count=%d\n",(character_count+space_count+1)+(character_count1+space_c
ount1+1));   }  }   else if(strstr(args[1],"txt")==NULL)  {   fp = fopen(args[2],"r");

 while((ch=fgetc(fp))!=EOF)   {

          character_count++;

          if(ch == ' ') {  space_count++;

          word_count++;  }   if(ch == '\n')  {    line_count++;   }    }

    fclose(fp);

     if(strcmp(args[1],"-l")==0)  //wc -l "file".txt (Print line count)  { printf("line_count =
%d\n",line_count);

 printf("File_name= %s\n",args[2]); }

 else if(strcmp(args[1],"-w")==0)  //wc -w "file".txt (Print word count)   {

                    printf("word_count = %d\n",word_count+2);

                    printf("File_name= %s\n",args[2]);   }

          else if(strcmp(args[1],"-m")==0)//wc -m "file".txt (Print character count)  {

                    printf("character_count = %d\n",character_count);

                    printf("File_name= %s\n",args[2]);   }

          else if(strcmp(args[1],"-s")==0) //wc -s "file".txt (Print space count)   {

                    printf("space_count = %d\n",space_count);

                    printf("File_name= %s\n",args[2]);   }

          else if(strcmp(args[1],"-L")==0)  //wc -L "file".txt (Print maximum length line) {

                    if(args[2] == NULL)   {

                    fprintf(stderr, "OS_Shell: expected argument to \"wc\"\n");   }

                    else  {   char line[1000];
```

```
                int maxc=-999;

                char ref[1000];

                fp = fopen(args[2],"r");

                while(fgets(line,sizeof(line),fp))   {   int len=strlen(line);

                        if(len>maxc){

                        maxc=len;

                        strcpy(ref,line);   }}   fclose(fp);

                printf("Biggest_line = %s",ref);

                printf("File_name= %s\n",args[2]); } } } }  return 1;  }
```

21) cmp

```
int lsh_cmp(char **args)  //compare the content of two files  { FILE *fx,*fx1;

   FILE *fz,*fz1;

int count=0,flag=0,linec=1;

   char c;

   char ch;

   if(strstr(args[1],"txt")!=NULL)  //cmp "file1".txt "file2".txt (Print byte and line number
if files don't match){   fx=fopen(args[1],"r");

     fx1=fopen(args[2],"r");

     while((c=getc(fx))!=EOF) {    count++;

          ch=getc(fx1);

          if(c=='\n')   {     linec++;    }

          if(ch!=c)  //Comparing each character of two files    {    flag=1;

          break;  }   }
     if(flag==1) {   printf("%s and %s do not match at byte %d line
%d\n",args[1],args[2],count,linec);

          return 1;  }  else  {   printf("%s and %s perfectly match\n",args[1],args[2]);

          return 1;   }

     fclose(fx);

     fclose(fx1);  }    else if(strcmp(args[1],"-b")==0)  //cmp -b "file1".txt "file2".txt (Print
character along with byte and line number at which file differ)  {

     fz=fopen(args[2],"r");

     fz1=fopen(args[3],"r");

     if(fz==NULL || fz1==NULL)   {   perror("Failed:");

          return 1;  }   while((c=getc(fz))!=EOF)   {  count++;

          ch=getc(fz1);

          if(c=='\n')  {   linec++;   }

          if(ch!=c)   {  flag=1;

          break; }   }

     if(flag==1)   {  printf("%s and %s differ at byte %d and line %d where %c in %s and
%c in %s differ\n",args[2],args[3],count,linec,c,args[2],ch,args[3]);

          return 1; }  else  { printf("%s and %s perfectly match\n",args[2],args[3]);

          return 1;  }

     fclose(fz);

     fclose(fz1);}  }
```

22) sort

```
int lsh_sort(char **args)  //sort the data in a file

{ FILE *f1;
```

```
   char sorted[1000][1000];

   char value[1000];

         int c=0;

         int i,j;

   if(strstr(args[1],"txt")!=NULL) //sorting in ascending order  {

     f1=fopen(args[1],"r");

     while(fgets(value,sizeof(value),f1)!=NULL) //Getting each line from file  {

          if(strchr(value,'\n'))  {

          value[strlen(value)-1]='\0'; }

          strcpy(sorted[c],value);  //Adding each line in the sorted array

          c++;   }

//Sorting the sorted array using bubble sort

     for(i=0;i<c-1;++i)   {

          for(j=0;j<(c-i-1);++j)  {

                   if(strcmp(sorted[j],sorted[j+1])>0)   {

                   strcpy(value,sorted[j]);

                   strcpy(sorted[j],sorted[j+1]);

                   strcpy(sorted[j+1],value);   }   }  }

     if(args[2]==NULL)  //Printing the sorted array  {

          printf("Sorted Data:");

          for(i=0;i<c;i++)   {

          printf("%s\n",sorted[i]);  }    }
```

```
 else if(strstr(args[2],"txt")!=NULL)  //Storing sorted array in new file   {
       FILE *f2;
       f2=fopen(args[2],"w");
       for(i=0;i<c;i++)  {
       fprintf(f2,"%s\n",sorted[i]);   }  }  }
else if(strcmp(args[1],"-r")==0)  //sorting in reverse order{
 f1=fopen(args[2],"r");
 while(fgets(value,sizeof(value),f1)!=NULL)   {
       if(strchr(value,'\n'))  {  value[strlen(value)-1]='\0'; }
       strcpy(sorted[c],value);
       c++;  }
 for(i=0;i<c-1;++i) {
       for(j=0;j<(c-i-1);++j)   { if(strcmp(sorted[j+1],sorted[j])>0)  {
                 strcpy(value,sorted[j]);
                 strcpy(sorted[j],sorted[j+1]);
                 strcpy(sorted[j+1],value);   }   }   }
 if(args[3]==NULL)  {
       printf("Sorted Data:");
       for(i=0;i<c;i++)  {
       printf("%s\n",sorted[i]);   }   }
 else if(strstr(args[3],"txt")!=NULL)   {
       FILE *f2;
```

```
        f2=fopen(args[3],"w");

        for(i=0;i<c;i++)   {  fprintf(f2,"%s\n",sorted[i]);   }  }  }  return 1;  }
```

23) rm

```
int lsh_rm(char **args)  //Delete a file from directory{int del=remove(args[1]);

  if(!del) {  printf("File deleted successfully\n");}  else{  printf("Deletion failed\n");  }

  return 1;  }
```

24) uniq

```
int lsh_uniq(char **args)  // Delete the duplicated line {

  if(args[1]==NULL) {

    fprintf(stderr, "OS_Shell: expected argument to \"uniq\"\n"); } else  {

  FILE *fp,*fp1;

  char line[100],line1[100];

  fp=fopen(args[1],"r");

  fp1=fopen("test.txt","w");

  fgets(line,sizeof(line),fp);

  fputs(line,fp1);

  while(fgets(line1,sizeof(line1),fp)) {

    if(strcmp(line,line1)==0) //Comparing each line of file with all the other lines  {continue;
}else { fputs(line1,fp1);

        strcpy(line,line1);}}

  fclose(fp);

  fclose(fp1);

  remove(args[1]);
```

```
    rename("test.txt",args[1]);}

    return 1;  }
```

25) exit

```
int lsh_exit(char **args)  { time_t t = time(NULL);

  struct tm tm = *localtime(&t);

  FILE *f = fopen("log2.txt", "a");

  fprintf(f,"%d-%02d-%02d %02d:%02d:%02d : ",tm.tm_year + 1900, tm.tm_mon + 1,
tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);

  for (int i=0; i<=l;i++){  char x[1000];

      strcpy(x,log_buffer[i][0]);

      int k=strlen(x);

      for(int j=0;j<k;j++)  {

                    if(x[j]=='\0' || x[j]=='\00'){   continue;   }

                    else {   fputc(x[j],f);  }  }

      fputs(",",f);   }

  fputs("\n",f);

    fclose(f);

  return 0;   }
```

26) clclog

```
int lsh_clclog(char **args)  //Clear the log  {

    char dat[20];

    char line[100];
```

```
   if(args[2]==NULL)  //clear entire log  {

     fclose(fopen("log2.txt","w"));

     return 1;  }

   else              //clear log on the specified date  {  FILE *fp,*fp1;

     int flag=0;

     fp=fopen("log2.txt","r");

     fp1=fopen("replica.txt","w");

     strcpy(dat,args[2]);

     while(fgets(line,sizeof(line),fp)) {

            for(int i=0;i<strlen(dat);i++)   {

            if(dat[i]!=line[i])  //Finding the log of the day other than specified date  {

                     flag=1;

                     break;   }   }

            if(flag==1)  {  fputs(line,fp1);  //Copying the entire log other than the specified date
in a demo file

            flag=0;   }   }

     fclose(fp);

     fclose(fp1);

//Deleting log and renaming the demo file to log name

     remove("log2.txt");

     rename("replica.txt","log2.txt");

     return 1;   }  }
```

27) viewlog

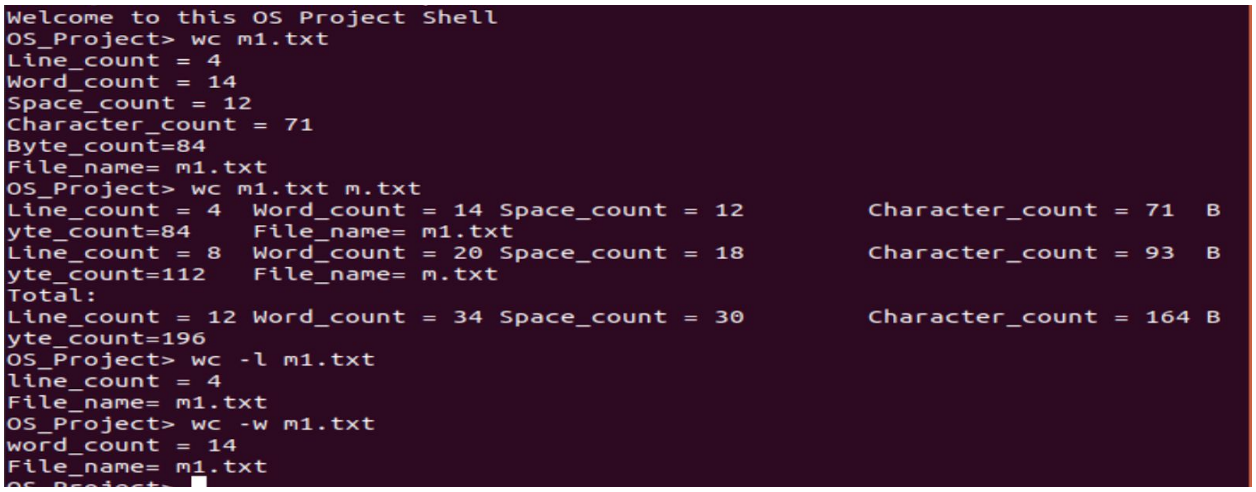
```
int lsh_viewlog(char **args)  //Used to view log of specific date {   if(args[1]==NULL){
      fprintf(stderr, "OS_Shell: expected argument to \"viewlog\"\n");}
    else  {  char dat[20];
    char line[100];
    FILE *fp;
            int flag=0;
    fp=fopen("log2.txt","r");
    strcpy(dat,args[1]);
    while(fgets(line,sizeof(line),fp))  {  for(int i=0;i<strlen(dat);i++)   {
             if(dat[i]!=line[i])  //Finding log of specified date from file   {
             flag=1;
             break;   }    }
      if(flag==0)  {   printf("%s",line);  //Printing the log of that day }
      else  { flag=0;  }  }
    fclose(fp);  }
    return 1;  }
```

28) clear

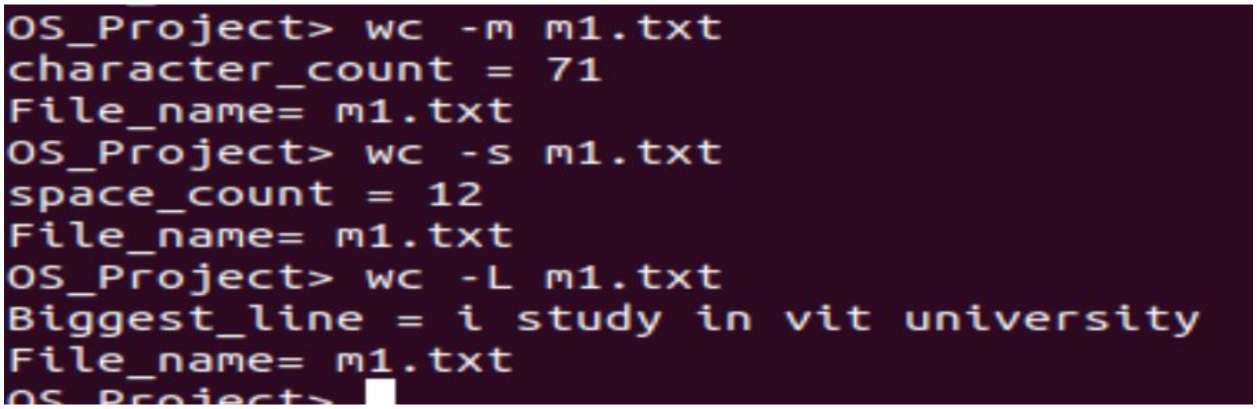
```
int lsh_clear(char **args)    //Clear the console window  {
    const char* blank="\e[1;1H\e[2J";
    write(STDOUT_FILENO,blank,12);
    return 1;  }
```

SCREENSHOTS:

WC-

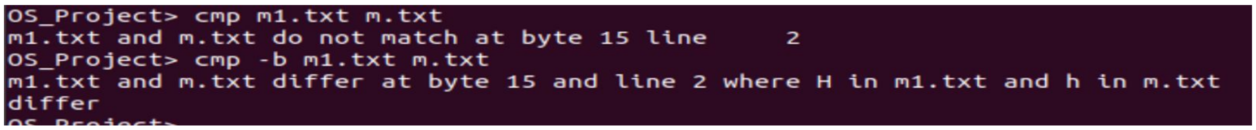


```
Welcome to this OS Project Shell
OS_Project> wc m1.txt
Line_count = 4
Word_count = 14
Space_count = 12
Character_count = 71
Byte_count=84
File_name= m1.txt
OS_Project> wc m1.txt m.txt
Line_count = 4  Word_count = 14 Space_count = 12        Character_count = 71  B
yte_count=84    File_name= m1.txt
Line_count = 8  Word_count = 20 Space_count = 18        Character_count = 93  B
yte_count=112   File_name= m.txt
Total:
Line_count = 12 Word_count = 34 Space_count = 30        Character_count = 164 B
yte_count=196
OS_Project> wc -l m1.txt
line_count = 4
File_name= m1.txt
OS_Project> wc -w m1.txt
word_count = 14
File_name= m1.txt
```



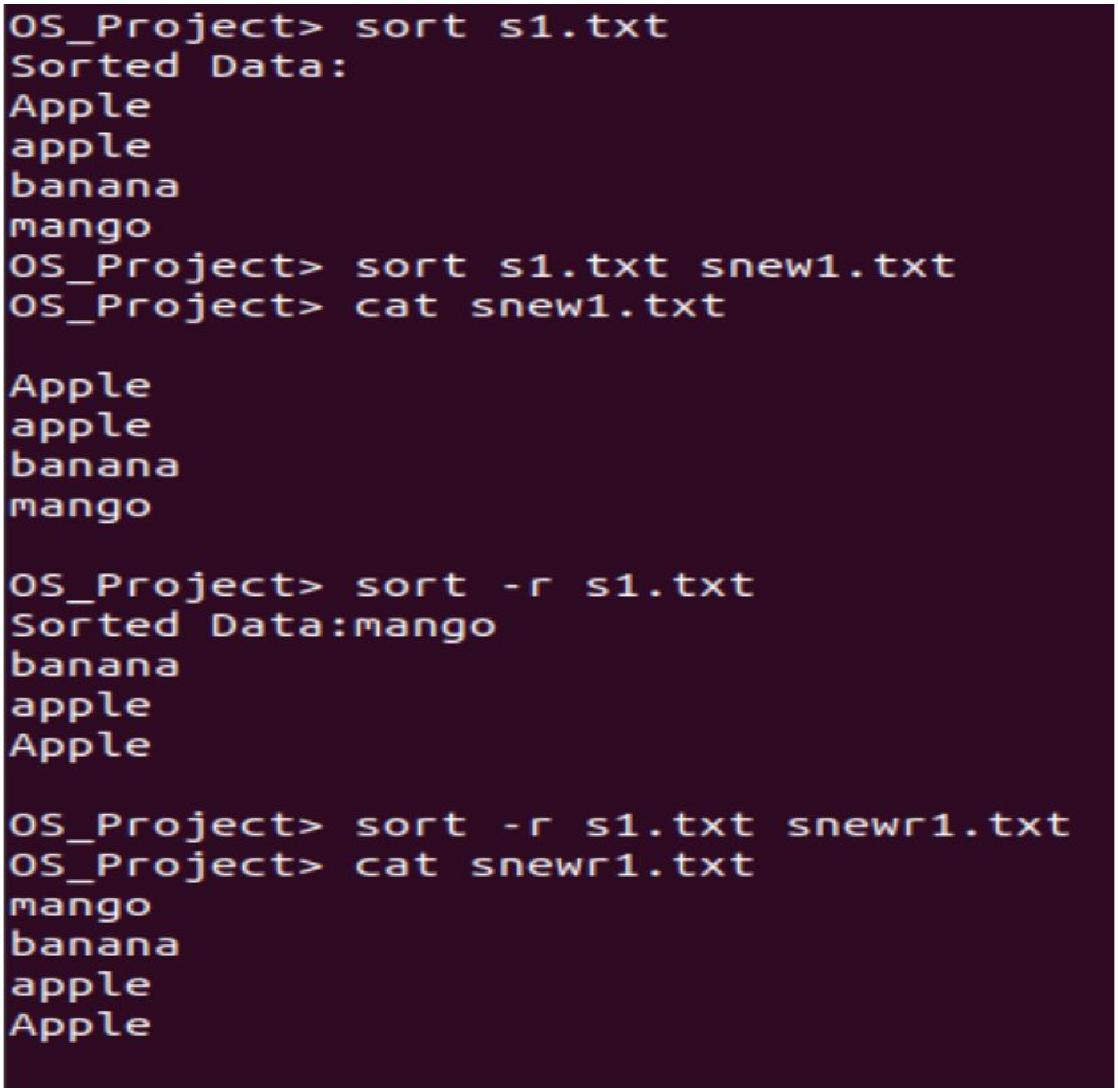
```
OS_Project> wc -m m1.txt
character_count = 71
File_name= m1.txt
OS_Project> wc -s m1.txt
space_count = 12
File_name= m1.txt
OS_Project> wc -L m1.txt
Biggest_line = i study in vit university
File_name= m1.txt
```

CMP-



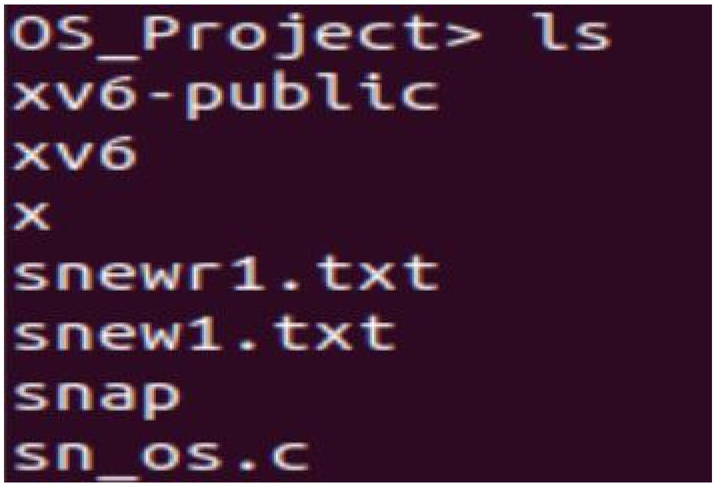
```
OS_Project> cmp m1.txt m.txt
m1.txt and m.txt do not match at byte 15 line      2
OS_Project> cmp -b m1.txt m.txt
m1.txt and m.txt differ at byte 15 and line 2 where H in m1.txt and h in m.txt
differ
```

SORT-

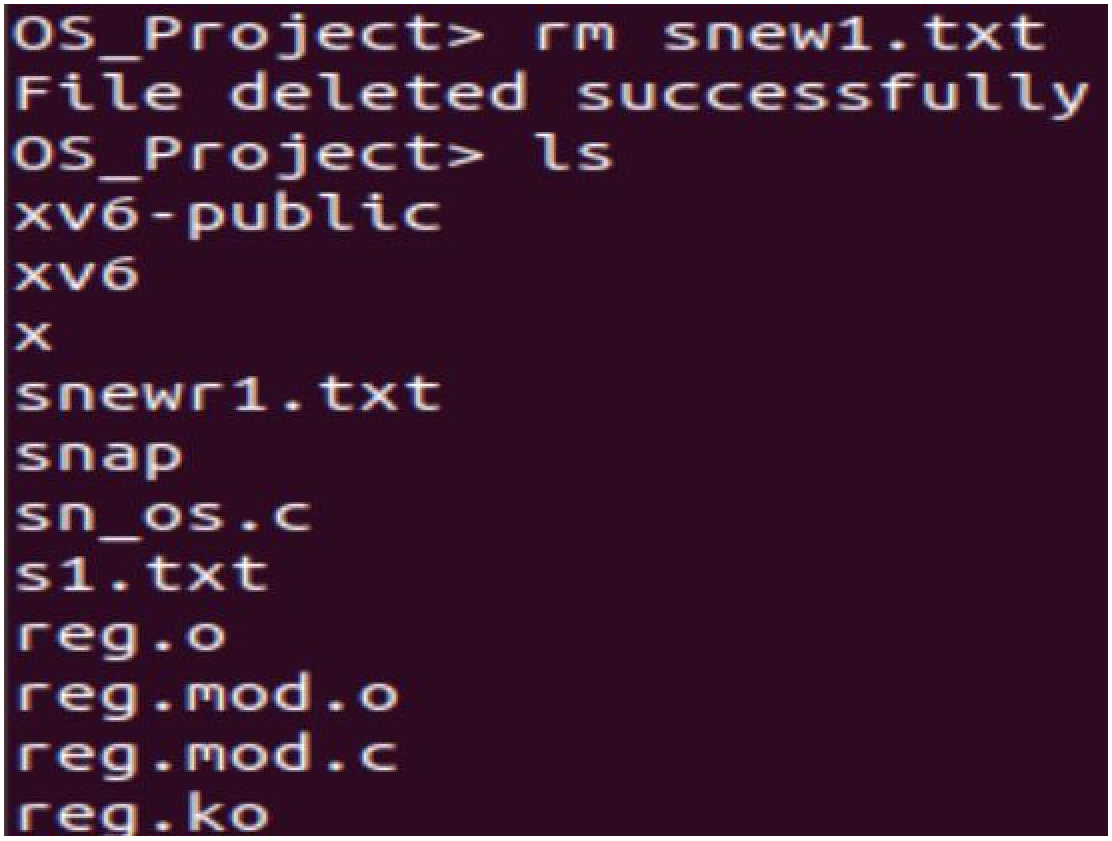


```
OS_Project> sort s1.txt
Sorted Data:
Apple
apple
banana
mango
OS_Project> sort s1.txt snew1.txt
OS_Project> cat snew1.txt

Apple
apple
banana
mango

OS_Project> sort -r s1.txt
Sorted Data:mango
banana
apple
Apple

OS_Project> sort -r s1.txt snewr1.txt
OS_Project> cat snewr1.txt
mango
banana
apple
Apple
```

RM-

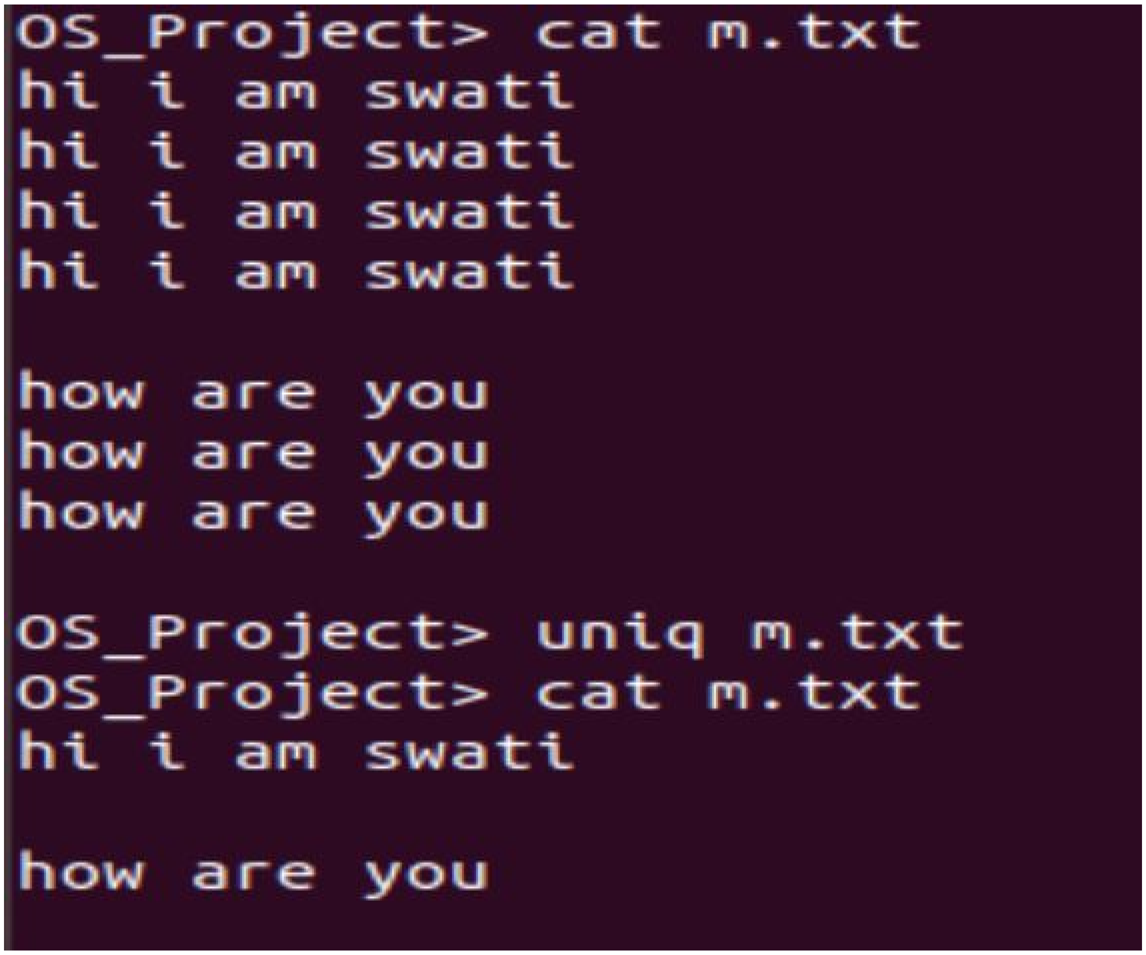


```
OS_Project> ls
xv6-public
xv6
x
snewr1.txt
snew1.txt
snap
sn_os.c
```



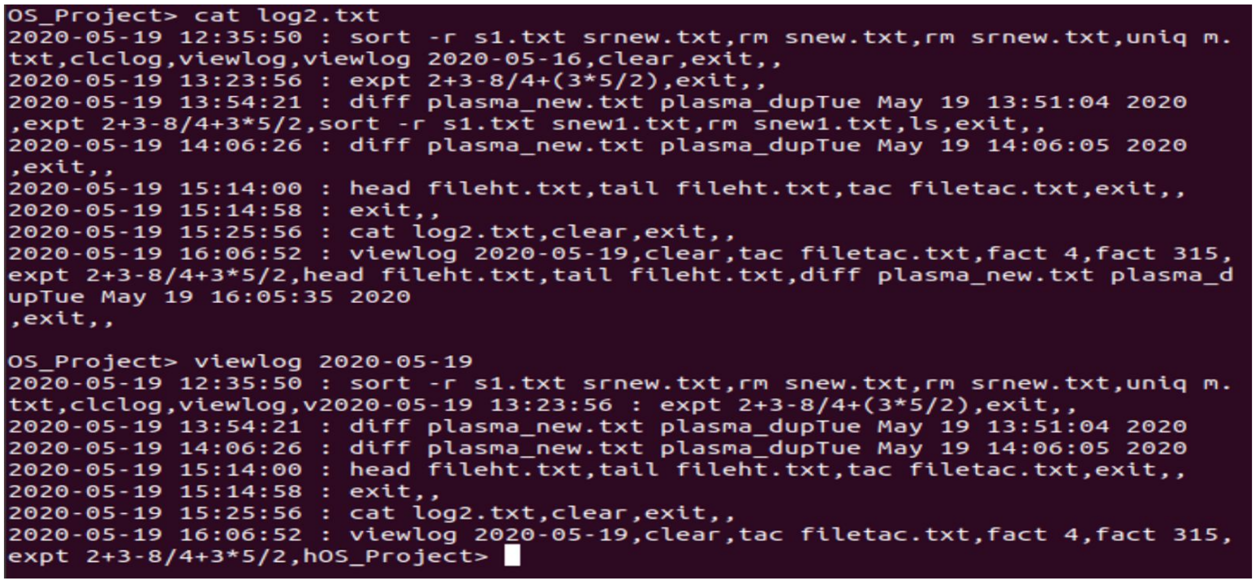
```
OS_Project> rm snew1.txt
File deleted successfully
OS_Project> ls
xv6-public
xv6
x
snewr1.txt
snap
sn_os.c
s1.txt
reg.o
reg.mod.o
reg.mod.c
reg.ko
```

UNIQ-



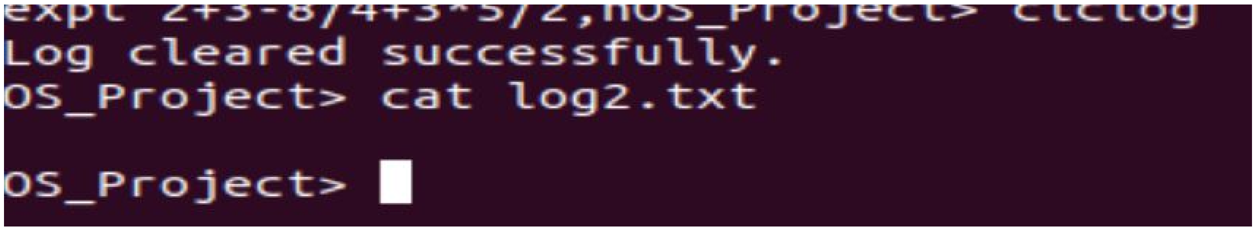
```
OS_Project> cat m.txt
hi i am swati
hi i am swati
hi i am swati
hi i am swati

how are you
how are you
how are you

OS_Project> uniq m.txt
OS_Project> cat m.txt
hi i am swati

how are you
```

VIEWLOG-



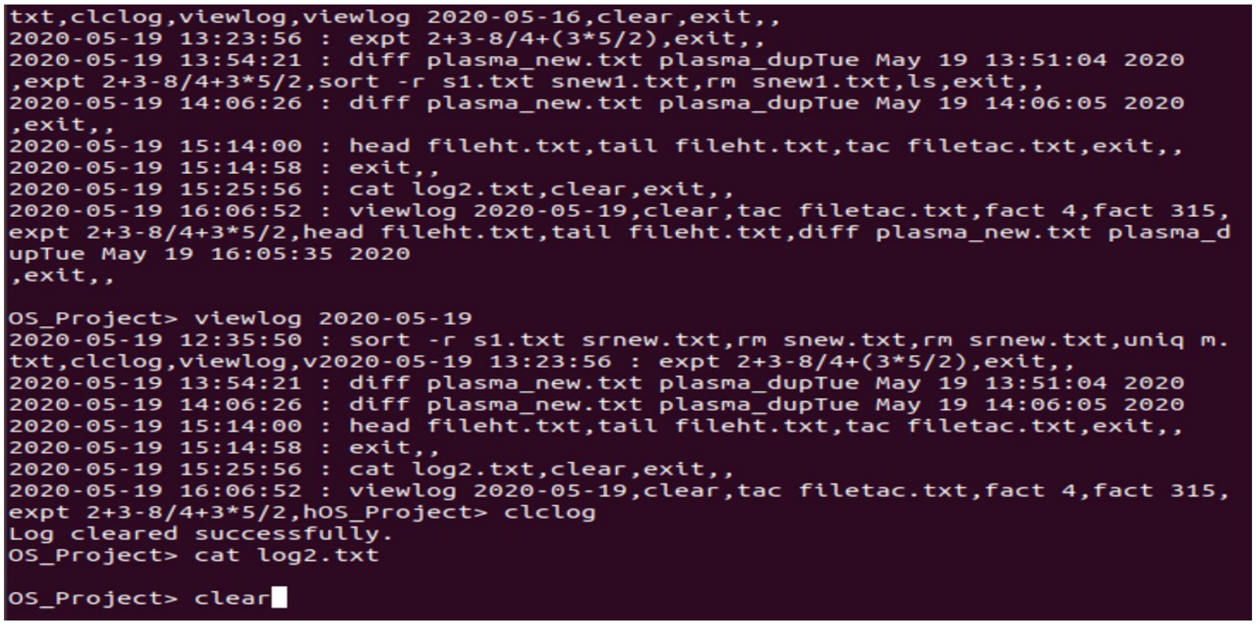
```
OS_Project> cat log2.txt
2020-05-19 12:35:50 : sort -r s1.txt srnew.txt,rm snew.txt,rm srnew.txt,uniq m.
txt,clclog,viewlog,viewlog 2020-05-16,clear,exit,,
2020-05-19 13:23:56 : expt 2+3-8/4+(3*5/2),exit,,
2020-05-19 13:54:21 : diff plasma_new.txt plasma_dupTue May 19 13:51:04 2020
,expt 2+3-8/4+3*5/2,sort -r s1.txt snew1.txt,rm snew1.txt,ls,exit,,
2020-05-19 14:06:26 : diff plasma_new.txt plasma_dupTue May 19 14:06:05 2020
,exit,,
2020-05-19 15:14:00 : head fileht.txt,tail fileht.txt,tac filetac.txt,exit,,
2020-05-19 15:14:58 : exit,,
2020-05-19 15:25:56 : cat log2.txt,clear,exit,,
2020-05-19 16:06:52 : viewlog 2020-05-19,clear,tac filetac.txt,fact 4,fact 315,
expt 2+3-8/4+3*5/2,head fileht.txt,tail fileht.txt,diff plasma_new.txt plasma_d
upTue May 19 16:05:35 2020
,exit,,

OS_Project> viewlog 2020-05-19
2020-05-19 12:35:50 : sort -r s1.txt srnew.txt,rm snew.txt,rm srnew.txt,uniq m.
txt,clclog,viewlog,v2020-05-19 13:23:56 : expt 2+3-8/4+(3*5/2),exit,,
2020-05-19 13:54:21 : diff plasma_new.txt plasma_dupTue May 19 13:51:04 2020
2020-05-19 14:06:26 : diff plasma_new.txt plasma_dupTue May 19 14:06:05 2020
2020-05-19 15:14:00 : head fileht.txt,tail fileht.txt,tac filetac.txt,exit,,
2020-05-19 15:14:58 : exit,,
2020-05-19 15:25:56 : cat log2.txt,clear,exit,,
2020-05-19 16:06:52 : viewlog 2020-05-19,clear,tac filetac.txt,fact 4,fact 315,
expt 2+3-8/4+3*5/2,hOS_Project> █
```

CLCLOG-

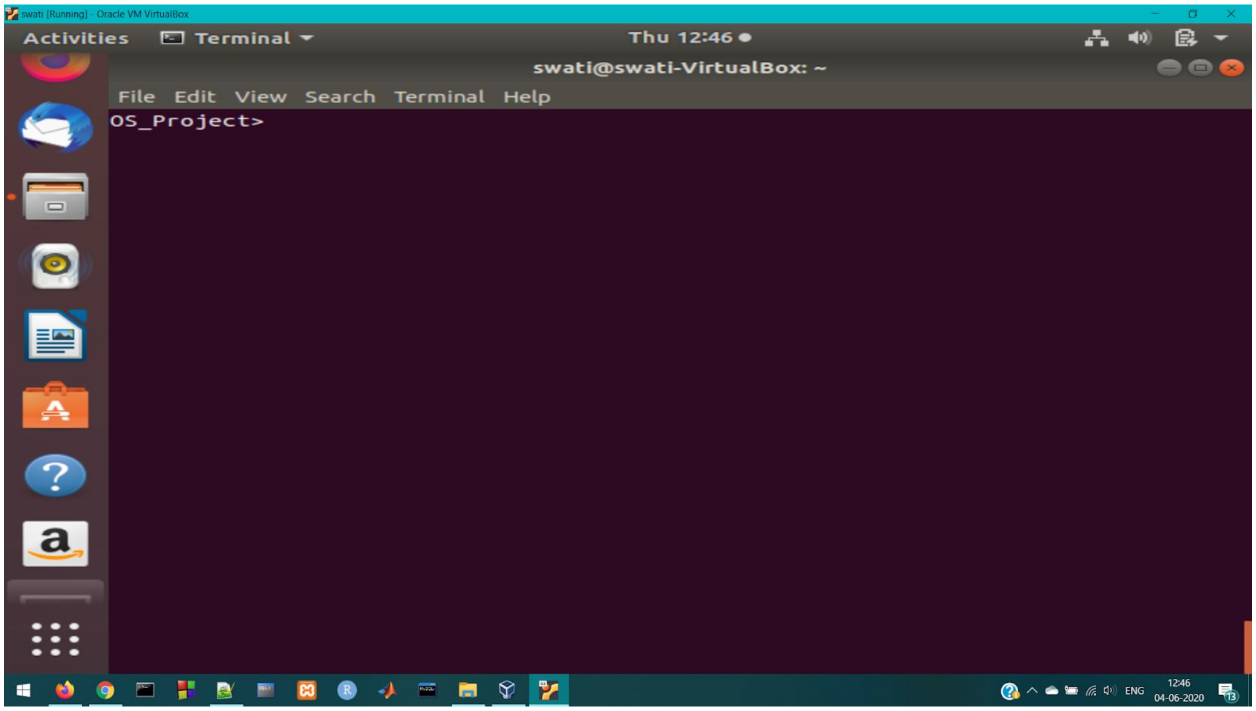


```
expt 2+3-8/4+3*5/2,hOS_Project> clclog
Log cleared successfully.
OS_Project> cat log2.txt

OS_Project> 
```

CLEAR-

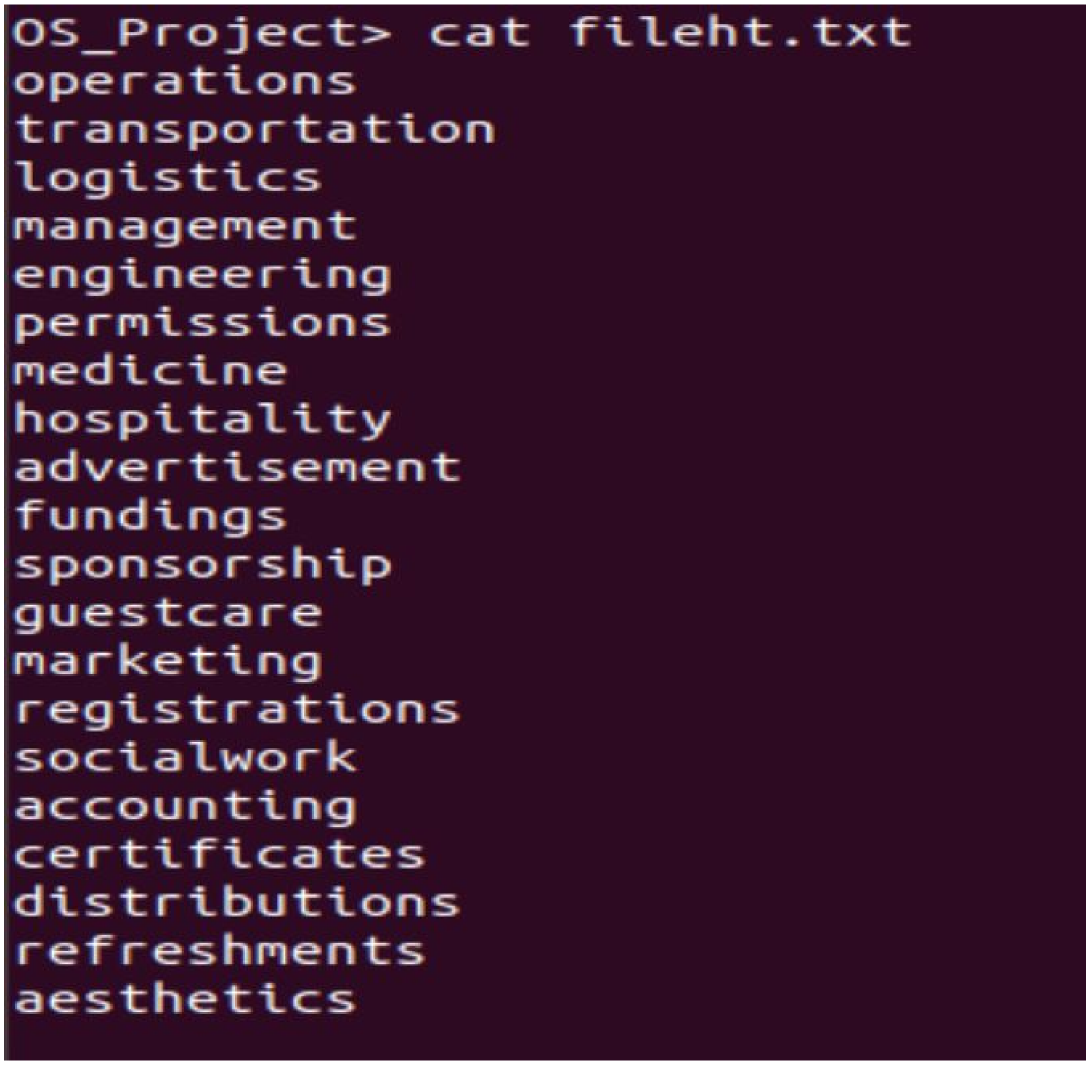


```
txt,clclog,viewlog,viewlog 2020-05-16,clear,exit,,
2020-05-19 13:23:56 : expt 2+3-8/4+(3*5/2),exit,,
2020-05-19 13:54:21 : diff plasma_new.txt plasma_dupTue May 19 13:51:04 2020
,expt 2+3-8/4+3*5/2,sort -r s1.txt snew1.txt,rm snew1.txt,ls,exit,,
2020-05-19 14:06:26 : diff plasma_new.txt plasma_dupTue May 19 14:06:05 2020
,exit,,
2020-05-19 15:14:00 : head fileht.txt,tail fileht.txt,tac filetac.txt,exit,,
2020-05-19 15:14:58 : exit,,
2020-05-19 15:25:56 : cat log2.txt,clear,exit,,
2020-05-19 16:06:52 : viewlog 2020-05-19,clear,tac filetac.txt,fact 4,fact 315,
expt 2+3-8/4+3*5/2,head fileht.txt,tail fileht.txt,diff plasma_new.txt plasma_d
upTue May 19 16:05:35 2020
,exit,,

OS_Project> viewlog 2020-05-19
2020-05-19 12:35:50 : sort -r s1.txt srnew.txt,rm snew.txt,rm srnew.txt,uniq m.
txt,clclog,viewlog,v2020-05-19 13:23:56 : expt 2+3-8/4+(3*5/2),exit,,
2020-05-19 13:54:21 : diff plasma_new.txt plasma_dupTue May 19 13:51:04 2020
2020-05-19 14:06:26 : diff plasma_new.txt plasma_dupTue May 19 14:06:05 2020
2020-05-19 15:14:00 : head fileht.txt,tail fileht.txt,tac filetac.txt,exit,,
2020-05-19 15:14:58 : exit,,
2020-05-19 15:25:56 : cat log2.txt,clear,exit,,
2020-05-19 16:06:52 : viewlog 2020-05-19,clear,tac filetac.txt,fact 4,fact 315,
expt 2+3-8/4+3*5/2,hOS_Project> clclog
Log cleared successfully.
OS_Project> cat log2.txt

OS_Project> clear
```

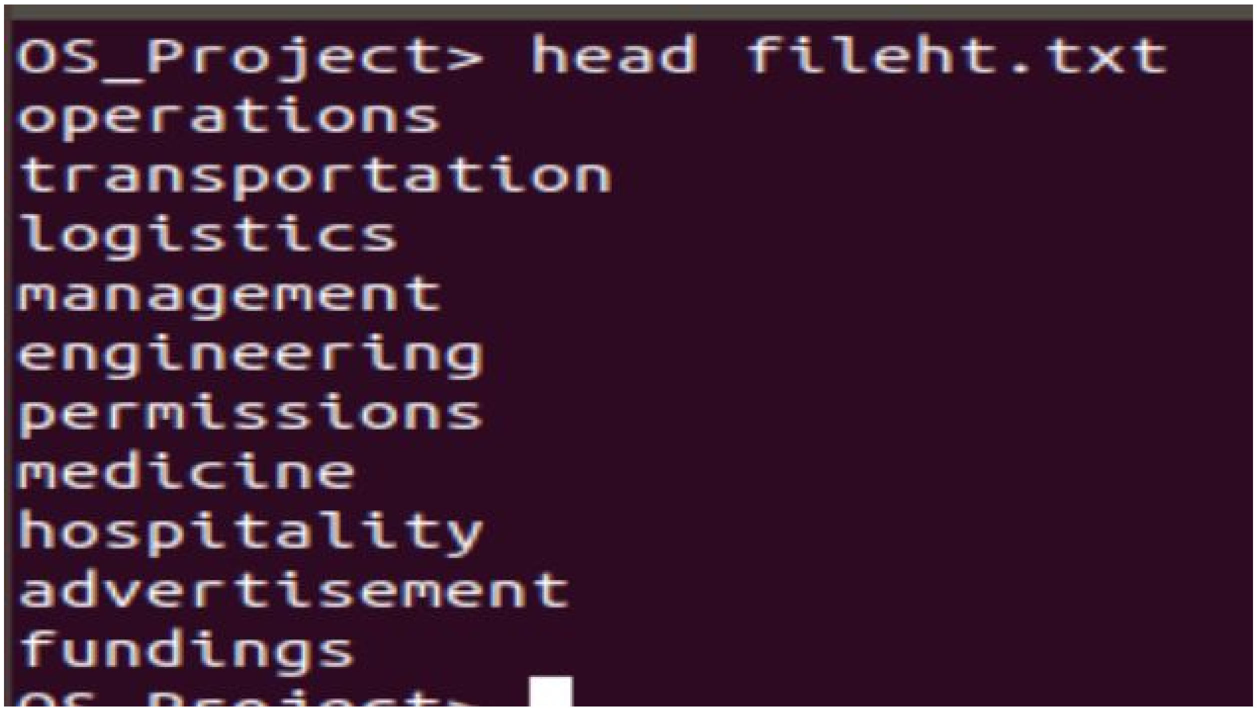


```
OS_Project>
```

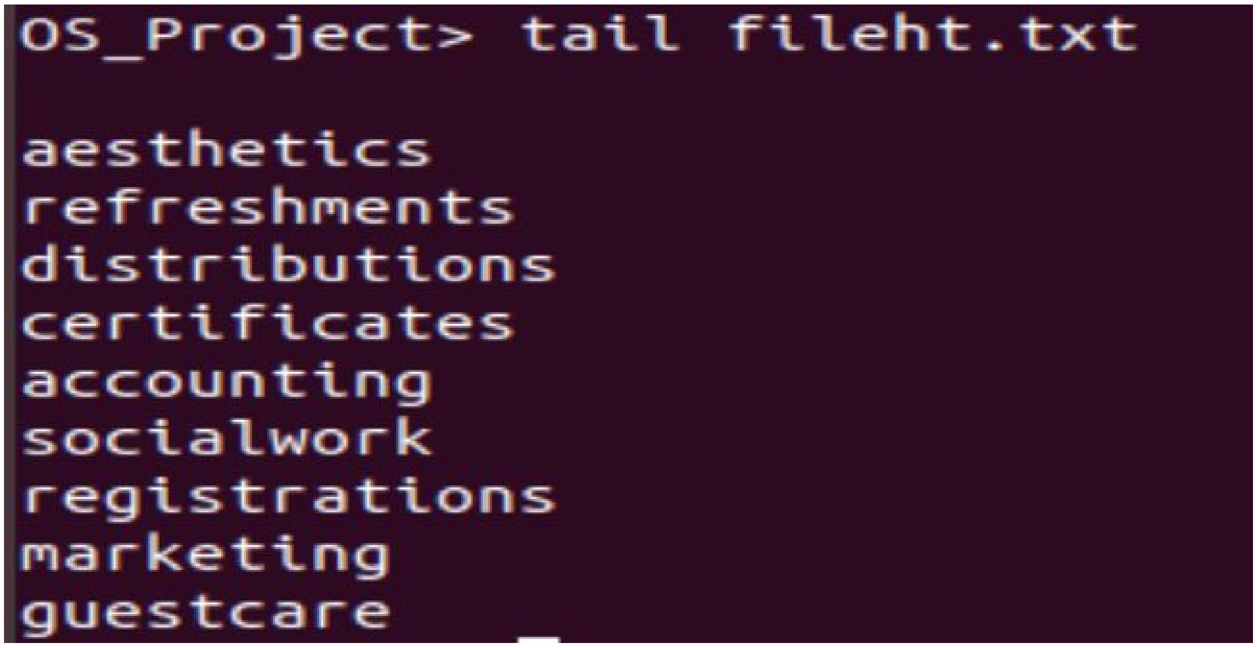
HEAD AND TAIL-



```
OS_Project> cat fileht.txt
operations
transportation
logistics
management
engineering
permissions
medicine
hospitality
advertisement
fundings
sponsorship
guestcare
marketing
registrations
socialwork
accounting
certificates
distributions
refreshments
aesthetics
```

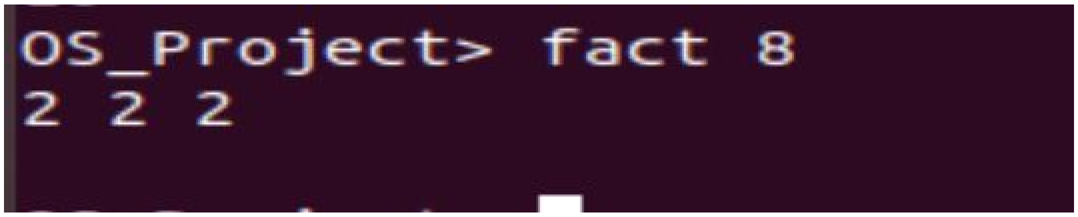


```
OS_Project> head fileht.txt
operations
transportation
logistics
management
engineering
permissions
medicine
hospitality
advertisement
fundings
```



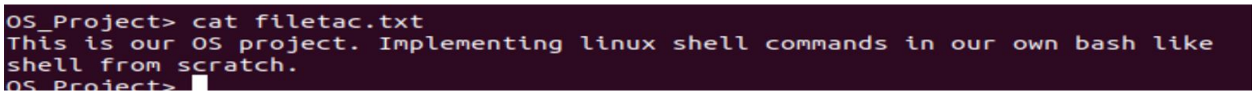
```
OS_Project> tail fileht.txt

aesthetics
refreshments
distributions
certificates
accounting
socialwork
registrations
marketing
guestcare
```

FACT-

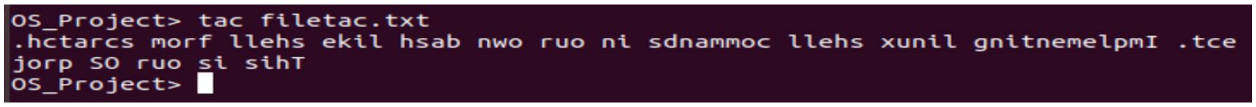


```
OS_Project> fact 8
2 2 2
```

TAC-

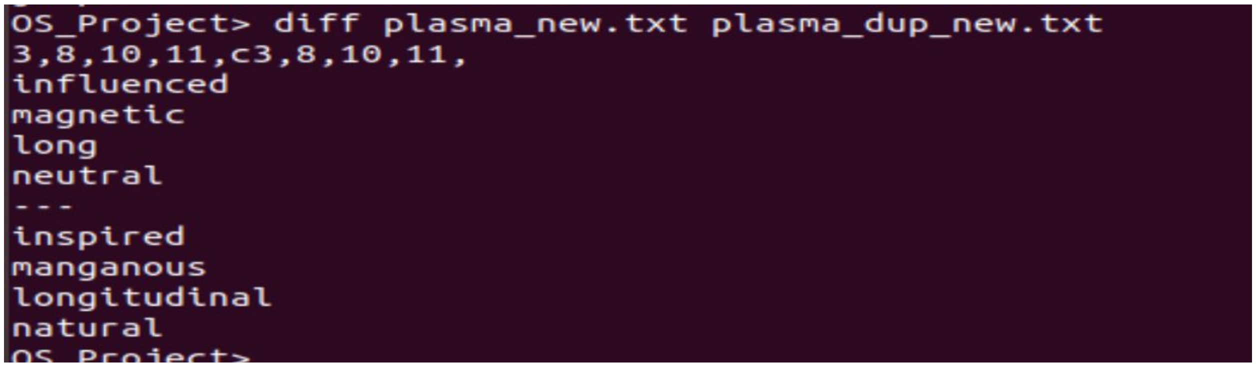


```
OS_Project> cat filetac.txt
This is our OS project. Implementing linux shell commands in our own bash like
shell from scratch.
OS_Project>
```



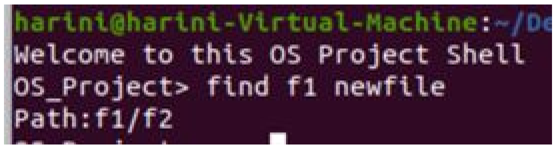
```
OS_Project> tac filetac.txt
.hctarcs morf llehs ekil hsab nwo ruo ni sdnammoc llehs xunil gnitnemelpmI .tce
jorp SO ruo si sihT
OS_Project>
```

DIFF-



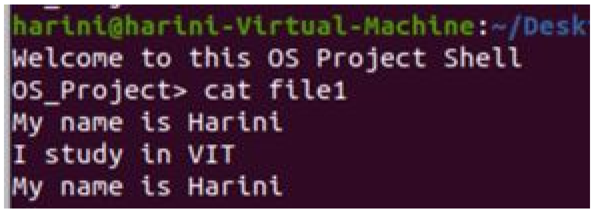
```
OS_Project> diff plasma_new.txt plasma_dup_new.txt
3,8,10,11,c3,8,10,11,
influenced
magnetic
long
neutral
---
inspired
manganous
longitudinal
natural
OS_Project>
```

Find <directory name> <filename>:



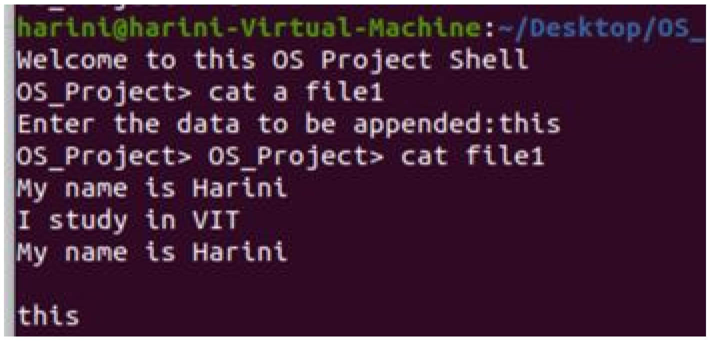
```
harini@harini-Virtual-Machine:~/De
Welcome to this OS Project Shell
OS_Project> find f1 newfile
Path:f1/f2
```

Cat <filename>



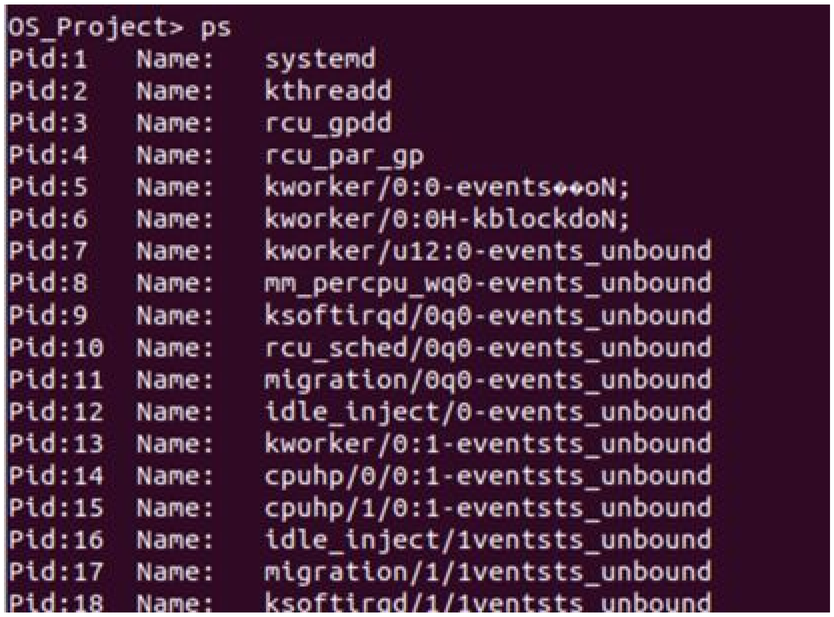
```
harini@harini-Virtual-Machine:~/Desk
Welcome to this OS Project Shell
OS_Project> cat file1
My name is Harini
I study in VIT
My name is Harini
```

Cat a <filename>:



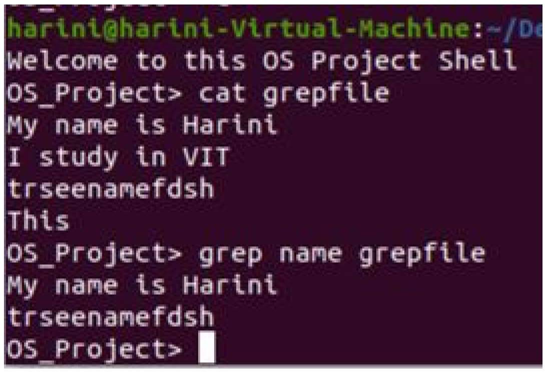
```
harini@harini-Virtual-Machine:~/Desktop/OS_
Welcome to this OS Project Shell
OS_Project> cat a file1
Enter the data to be appended:this
OS_Project> OS_Project> cat file1
My name is Harini
I study in VIT
My name is Harini

this
```

Ps:



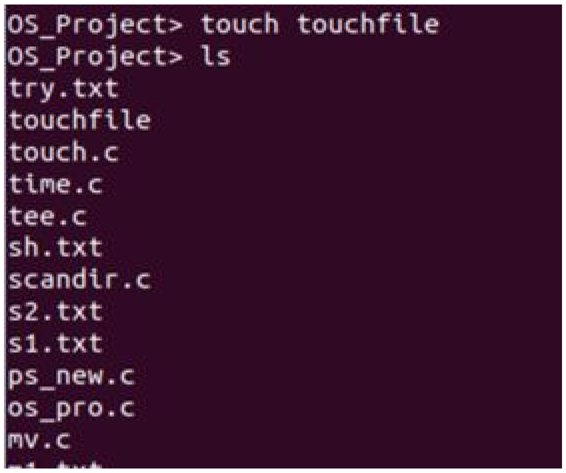
```
OS_Project> ps
Pid:1    Name:    systemd
Pid:2    Name:    kthreadd
Pid:3    Name:    rcu_gpdd
Pid:4    Name:    rcu_par_gp
Pid:5    Name:    kworker/0:0-events��oN;
Pid:6    Name:    kworker/0:0H-kblockdoN;
Pid:7    Name:    kworker/u12:0-events_unbound
Pid:8    Name:    mm_percpu_wq0-events_unbound
Pid:9    Name:    ksoftirqd/0q0-events_unbound
Pid:10   Name:    rcu_sched/0q0-events_unbound
Pid:11   Name:    migration/0q0-events_unbound
Pid:12   Name:    idle_inject/0-events_unbound
Pid:13   Name:    kworker/0:1-eventsts_unbound
Pid:14   Name:    cpuhp/0/0:1-eventsts_unbound
Pid:15   Name:    cpuhp/1/0:1-eventsts_unbound
Pid:16   Name:    idle_inject/1ventsts_unbound
Pid:17   Name:    migration/1/1ventsts_unbound
Pid:18   Name:    ksoftirqd/1/1ventsts_unbound
```

Grep <word to search> <filename>:



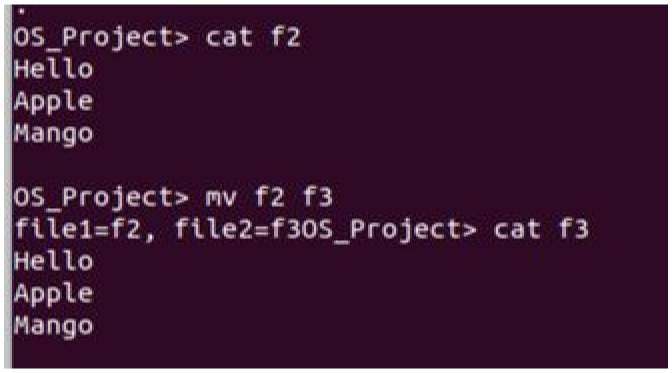
```
harini@harini-Virtual-Machine:~/De
Welcome to this OS Project Shell
OS_Project> cat grepfile
My name is Harini
I study in VIT
trseenamefdsh
This
OS_Project> grep name grepfile
My name is Harini
trseenamefdsh
OS_Project>
```

Touch <newfilename>



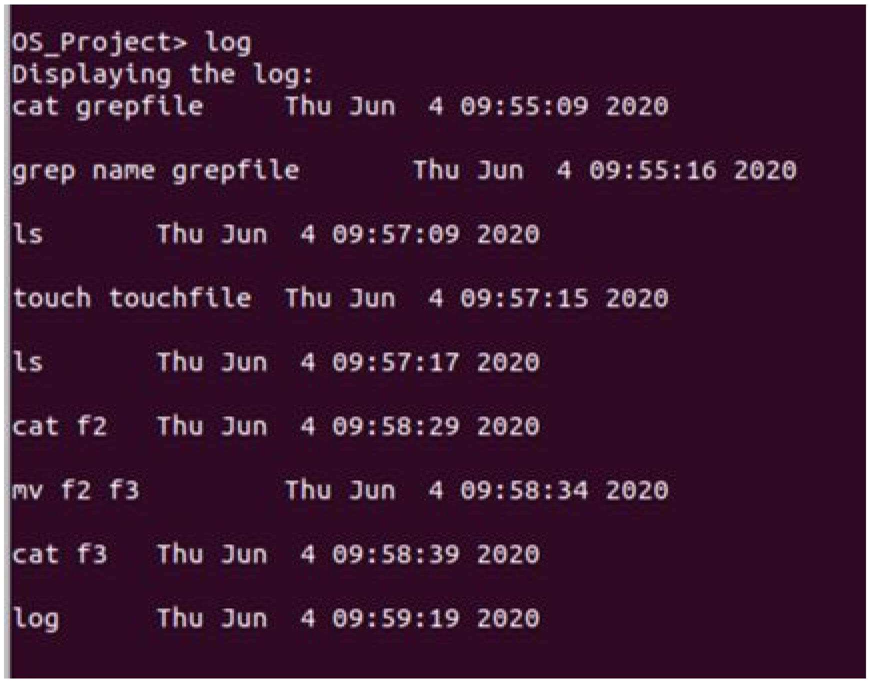
```
OS_Project> touch touchfile
OS_Project> ls
try.txt
touchfile
touch.c
time.c
tee.c
sh.txt
scandir.c
s2.txt
s1.txt
ps_new.c
os_pro.c
mv.c
```

Mv <file1> <file2>:



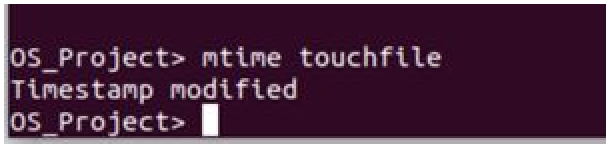
```
OS_Project> cat f2
Hello
Apple
Mango

OS_Project> mv f2 f3
file1=f2, file2=f3OS_Project> cat f3
Hello
Apple
Mango
```

Log:



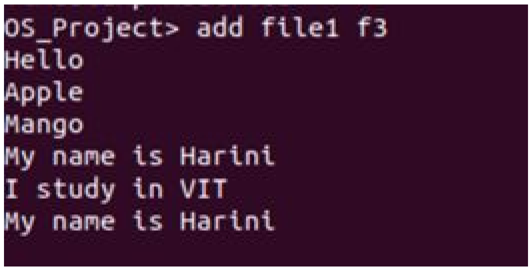
```
OS_Project> log
Displaying the log:
cat grepfile     Thu Jun  4 09:55:09 2020

grep name grepfile       Thu Jun  4 09:55:16 2020

ls       Thu Jun  4 09:57:09 2020

touch touchfile  Thu Jun  4 09:57:15 2020

ls       Thu Jun  4 09:57:17 2020

cat f2   Thu Jun  4 09:58:29 2020

mv f2 f3         Thu Jun  4 09:58:34 2020

cat f3   Thu Jun  4 09:58:39 2020

log      Thu Jun  4 09:59:19 2020
```

Mtime <filename>:



```
OS_Project> mtime touchfile
Timestamp modified
OS_Project> 
```

Add <file1> file2>:



```
OS_Project> add file1 f3
Hello
Apple
Mango
My name is Harini
I study in VIT
My name is Harini
```

RESULT:

The shell successfully implements around thirty shell commands. These commands are of different categories and use various user defined as well as some inbuilt C functions. The shell is a recreation of the bash shell, covering many of its important commands as well as some new commands. These new commands include finding the location of a process, changing the timestamp of commands in the log, solving a mathematical expression and many more. The project thus successfully creates a BASH like shell experience.This project can be extended to create a shell with all of the functionalities of the bash shell by incorporating more functions which make use of system calls and various inbuilt C functionalities.

REFERENCES:

https://en.wikipedia.org/wiki/Bash_(Unix_shell)#:~:text=GNU%20Bash%20or%20simply%20Bash,macOS%20Mojave%20and%20earlier%20versions.
https://www.tutorialspoint.com/unix/unix-useful-commands.htm
https://stackoverflow.com/questions/23607980/implementing-my-own-ps-command
https://studentboxoffice.in/jntuh/notes/linux-programming-and-data-mining-lab/write-in-c-the-following-unix-commands-using-system-calls-a-cat-b-ls-c-mv/66
https://www.thelearningpoint.net/computer-science/c-program-building-an-expression-evaluator
https://www.geeksforgeeks.org/c-program-for-efficiently-print-all-prime-factors-of-a-given-number/