NAME - SHREERAM

REG NO - 20214033

ASSIGNMENT 1

QUESTION 1:

Study of Unix/Linux general purpose utility command list : man, who,cat, cd, cp, ps, ls, mv,rm, mkdir,rmdir, echo, ccommands.

- man : man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS .

- who : who command is used to find out Time of last system boot , Current run level of the system , List of logged in users and more.

- cat:It is generally used to concatenate the files. It gives the output on the standard output

- cd:Use the "cd" command to change directory. For example, if you are in the home folder, and you want to go to the downloads folder, then you can type in “cd Downloads”.

- Cp: cp stands for copy. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name. cp command require at least two filenames in its arguments.

- Ps:The ps command reports information on current running processes, outputting to standard output. It is frequently used to find process identifier numbers. It supports searching for processes by user, group, process id or executable name.

- Ls:If you want to see the list of files on your UNIX or Linux system, use the 'ls' command. It shows the files /directories in your current directory. The ls command will show you the list of files in your current directory.

- Mv:mv stands for move. mv is used to move one or more files or directories from one place to another in file system like UNIX. It has two distinct functions:
  (i) It rename a file or folder.
  (ii) It moves group of files to different directory.

- Rm:Use the rm command to delete files and directories.

- Mkdir:Use the mkdir command when you need to create a folder or a directory.

- Rmkdir:Use rmdir to delete a directory. But rmdir can only be used to delete an empty

directory.

- Echo:echo command in linux is used to display line of text/string that are passed as an argument . This is a built in command that is mostly used in shell scripts and batch files to output status text to the screen or a file.

- more:more command is used to view the text files in the command prompt, displaying one screen at a time in case the file is large (For example log files). The more command also allows the user do scroll up and down through the page.

- date:The `date` command displays the current day, date, time, and year.

- time: The time command in Linux and Unix-like operating systems **lets you determine how long a specific command will take to run.** Usually, it's used to measure the performance of scripts or commands. The faster a command finishes its task, the better its performance.

- kill:Use this command as a last resort to destroy any jobs or programs that you suspended and are unable to restart. Use the jobs command to see a list of suspended jobs.

- history: ***history*** command is used to view the previously executed command. This feature was not available in the Bourne shell. Bash and Korn support this feature in which every command executed is treated as the event and is associated with an event number using which they can be recalled and changed if required. These commands are saved in a history file. In Bash shell **history** command shows the whole list of the command.***history*** command is used to view the previously executed command. This feature was not available in the Bourne shell. Bash and Korn support this feature in which every command executed is treated as the event and is associated with an event number using which they can be recalled and changed if required. These commands are saved in a history file. In Bash shell **history** command shows the whole list of the command.

- chmod: This command changes the permission information associated with a file. Every file (including directories, which Unix treats as files) on a Unix system is stored with records indicating who has permission to read, write, or execute the file, abbreviated as r, w, and x. These permissions are broken down for three categories of user: first, the owner of the file; second, a group with which both the user and the file may be associated; and third, all other users. These categories are abbreviated as u for owner (or user), g for group, and o for other.

- chown: The chown command **changes the owner of the file or directory specified by the File or Directory parameter to the user specified by the Owner parameter.** The value of the Owner parameter can be a user name from the user database or a numeric user ID. Optionally, a group can also be specified.

- pwd:This command reports the current directory path.

- cal: This command will print a calendar for a specified month and/or year.

- logout: Logging out of UNIX may be achieved simply by typing **logout, or <ctrl-D> or exit.** All three terminate the login shell and , in the former case, the shell performs commands from the . bash_logout file in your home directory. Exit is a C function that kills the calling process and circumvents all cleanup.

- shutdown: **shutdown** brings the system down in a secure way. All logged-in users are notified that the system is going down, and **login**(1) is blocked. It is possible to shut the system down immediately or after a specified delay.

QUESTION 2:

Write C programs to simulate UNIX commands like ls, grep, etc.

```
#include<stdio.h>
#include<stdlib.h>
#include<dirent.h>
#define DATA_SIZE 1000
void createf()
{  char data[DATA_SIZE];
    char n[100];
    FILE * fPtr;
    int i;
    printf("create 2 files \nfile1: with data \nfile2: without
data for copying\n");
   for ( i=0;i<2;i++){
   printf("enter a file name:");
   gets(n);
    fPtr = fopen(n,"w");
    if(fPtr == NULL)
    {   printf("Unable to create file.\n");
        exit(EXIT_FAILURE);
    }
    printf("Enter contents to store in file : \n");
    fgets(data, DATA_SIZE, stdin);
    fputs(data, fPtr);
    fclose(fPtr);
    printf("File created and saved successfully. ?? \n");
}
}
void copyfun(){
```

```
char ch, source_file[20], target_file[20];
       FILE *source, *target;
       printf("Enter name of file to copy\n");
       gets(source_file);
       source = fopen(source_file, "r");
       if (source == NULL)
       {
          printf("Press any key to exit...\n");
          exit(EXIT_FAILURE);
       }
       printf("Enter name of target file\n");
       gets(target_file);
       target = fopen(target_file, "w");
       if (target == NULL)
       {
          fclose(source);
          printf("Press any key to exit...\n");
          exit(EXIT_FAILURE);
       }
       while ((ch = fgetc(source)) != EOF)
          fputc(ch, target);

       printf("File copied successfully.\n");
       fclose(source);
       fclose(target);
}
void lsandgrep(){
 char fn[10],pat[10],temp[200];
FILE *fp;
char dirname[10];
DIR*p;
struct dirent *d;
printf("Enter directory name\n");
scanf("%s",dirname);
p=opendir(dirname);
if(p==NULL)
  {
  perror("Cannot find directory");
  exit(0);
  }
while(d=readdir(p))
  printf("%s\n",d->d_name);
```

```
}

int main(){
 createf();
 copyfun();
 lsandgrep();
}
```

QUESTION 3:

Write a program to implement
1. Create a file
2. Read contents of a file
3. Write to a file
4. Link and unlink a file
5. Copy file
6. Read contents of a file in a reverse order

Using the system calls: open( ), close( ), read( ), write( ), lseek( ), link( ), unlink( ).

```
// C program to illustrate
// open system call
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

extern int errno;

int main()
{
    // if file does not have in directory
    // then file foo.txt is created.
    int fd = open("foo.txt", O_RDONLY | O_CREAT);

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

    if (fd == -1) {
            // print which type of error have in a code
            printf("Error Number % d\n", errno);

            // print program detail "Success or failure"
            perror("Program");
    }
```

```
    return 0;
}

// C program to illustrate close system Call
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    int fd1 = open("foo.txt", O_RDONLY);
    if (fd1 < 0) {
            perror("c1");
            exit(1);
    }
    printf("opened the fd = % d\n", fd1);

    // Using close system Call
    if (close(fd1) < 0) {
            perror("c1");
            exit(1);
    }
    printf("closed the fd.\n");
}

// C program to illustrate
// read system Call
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    int fd, sz;
    char* c = (char*)calloc(100, sizeof(char));

    fd = open("foo.txt", O_RDONLY);
    if (fd < 0) {
            perror("r1");
            exit(1);
    }

    sz = read(fd, c, 10);
    printf("called read(% d, c, 10). returned that"
```

```
            " %d bytes were read.\n",
            fd, sz);
    c[sz] = '\0';
    printf("Those bytes are as follows: % s\n", c);

    return 0;
}

// C program to illustrate
// write system Call
#include<stdio.h>
#include <fcntl.h>
main()
{
int sz;

int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (fd < 0)
{
    perror("r1");
    exit(1);
}

sz = write(fd, "hello geeks\n", strlen("hello geeks\n"));

printf("called write(% d, \"hello geeks\\n\", %d)."
    " It returned %d\n", fd, strlen("hello geeks\n"), sz);

close(fd);
}

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

    int main()
    {
       int fd;
       long position;

       fd = open("datafile.dat", O_RDONLY);
       if ( fd != -1)
          {
          position = lseek(fd, 0L, 2); /* seek 0 bytes from end-of-file */
          if (position != -1)
```

```
            printf("The length of datafile.dat is %ld bytes.\n", position);
          else
            perror("lseek error");
          }
        else
          printf("can't open datafile.dat\n");
        close(fd);
      }


    #include <stdio.h>

int main()
{
  if ((link("foo.old", "foo.new")) == -1)
    {
    perror(" ");
    exit (1);        /* return a non-zero exit code on error */
    }
  exit(0);
}

int main()
{
  if ((unlink("foo.bar")) == -1)
    {
    perror(" ");
    exit (1);        /* return a non-zero exit code on error */
    }
  exit (0);
}
```

QUESTION 4:

Determine the size of a file using the lseek command. Once you found out the size, calculate the
number of blocks assigned for the file. Compare these results with the similar results obtained when
using the function stat.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
```

```
#include <unistd.h>

int main() {
        const char *filename = "example.txt";
        int fd = open(filename, O_RDONLY);

        if (fd == -1) {
        perror("Error opening file");
        return 1;
        }

        // Using lseek to find the size of the file
        off_t file_size = lseek(fd, 0, SEEK_END);

        if (file_size == -1) {
        perror("Error seeking file");
        close(fd);
        return 1;
        }

        printf("Using lseek:\n");
        printf("File size: %ld bytes\n", (long)file_size);
        printf("Number of blocks assigned: %ld\n", (long)file_size / 512); // Assuming a block size
of 512 bytes

        struct stat file_info;
        if (stat(filename, &file_info) == -1) {
        perror("Error getting file info");
        close(fd);
        return 1;
        }

        printf("Using stat:\n");
        printf("File size from stat: %ld bytes\n", (long)file_info.st_size);
        printf("Number of blocks from stat: %ld\n", (long)file_info.st_blocks);

        close(fd);
        return 0;
}
```

Output:

```
Using lseek:
```

```
File size: 1024 bytes
Number of blocks assigned: 2
Using stat:
File size from stat: 1024 bytes
Number of blocks from stat: 8
```

QUESTION 5:

Write a program to change current working directory and display the inode details for each file in the
new directory using the system calls: opendir( ), readdir( ), closedir( ), getcwd( ), chdir( ).

```
#include <sys/types.h>

#include <sys/stat.h>

#include <dirent.h>

void listDir(char *dirName)

{

  DIR* dir;

  struct dirent *dirEntry;

  struct stat inode;

  char name[1000];

  dir = opendir(dirName);

    if (dir == 0) {

        perror ("Eroare deschidere fisier");

        exit(1);

    }

    while ((dirEntry=readdir(dir)) != 0) {

        sprintf(name,"%s/%s",dirName,dirEntry->d_name);
```

```
        lstat (name, &inode);


         // test the type of file
        if (S_ISDIR(inode.st_mode))
                printf("dir ");
        else if (S_ISREG(inode.st_mode))
        printf ("fis ");
                else
                if (S_ISLNK(inode.st_mode))
                printf ("lnk ");
        else;
        printf(" %s\n", dirEntry->d_name);
        }
}


#define _POSIX_SOURCE
#include <dirent.h>
#include <sys/types.h>
#undef _POSIX_SOURCE
#include <stdio.h>


main() {
  DIR *dir;
  struct dirent *entry;
```

```
  int count;

  if ((dir = opendir("/")) == NULL)
    perror("opendir() error");
  else {
    count = 0;
    while ((entry = readdir(dir)) != NULL) {
      printf("directory entry %03d: %s\n", ++count, entry->d_name);
    }
    closedir(dir);
  }
}
#define _POSIX_SOURCE
#include  <unistd.h>
#undef _POSIX_SOURCE
#include <stdio.h>

main() {
  char cwd[256];

  if (chdir("/tmp") != 0)
    perror("chdir() error()");
  else {
    if (getcwd(cwd, sizeof(cwd)) == NULL)
```

```
      perror("getcwd() error");
    else
      printf("current working directory is: %s\n", cwd);
  }
}

int main(int argc, char **argv)
{
            if (argc != 2) {
            printf ("UTILIZARE: %s nume_dir\n", argv[0]);
            exit(0);
            }
    printf(\94Continutul directorului este:\n\94);
    listDir(argv[1]);
}

#include<stdio.h>

// chdir function is declared
// inside this header
#include<unistd.h>
int main()
{
    char s[100];

    // printing current working directory
    printf("%s\n", getcwd(s, 100));

    // using the command
    chdir("..");

    // printing current working directory
```

```
    printf("%s\n", getcwd(s, 100));

    // after chdir is executed
    return 0;
}
```