**OS\_LAB\_08\_Assignment**

**CE\_054**

Aim :- Program to Implement a Basic shell in C programming Language.

Shell Code :-

    // Author : Dhruv B Kakadiya

// CE\_054

// shell mini shell

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

#include<readline/readline.h>

#include<readline/history.h>

#define WRDLENMAX 200

#define CMDLENMAX 500

int getIpstr(char\* ipstringarg)

{

    char\* line;

    char cwd[512];

    getcwd(cwd, sizeof(cwd));

    printf("\n%s", cwd);

    line = readline(" $ ");

    if (strlen(line) != 0)

    {

        add\_history(line);

        strcpy(ipstringarg, line);

        return 0;

    }

    else

    {

        return 1;

    }

}

void cmdExecutionWithoutPipe(char\*\* parBeforePipeArgument)

{

    int pid = fork();

    int wst;

    if (pid == -1)

    {

        printf("\nError while creaating the child.\n");

        return;

    }

    else if (pid == 0)

    {

        if (execvp(parBeforePipeArgument[0], parBeforePipeArgument) < 0)

        {

            printf("\nError while executing the command.");

        }

        exit(0);

    }

    else

    {

        wait(&wst);

        return;

    }

}

void cmdExecutionWithPipe(char\*\* parBeforePipeArgument, char\*\* parAfterPipeArgument)

{

    int pid1, pid2, wst, pret, pipefd[2];

    pret = pipe(pipefd);

    if (pret < 0)

    {

        printf("\nError while creating the pipe.\n");

        return;

    }

    pid1 = fork();

    if (pid1 < 0)

    {

        printf("\n1\_Error while creaating the child.");

        return;

    }

    if (pid1 == 0)

    {

        close(pipefd[0]);

        dup2(pipefd[1], STDOUT\_FILENO);

        close(pipefd[1]);

        if (execvp(parBeforePipeArgument[0], parBeforePipeArgument) < 0)

        {

            printf("\n1\_Error while executing the command.");

            exit(0);

        }

    }

    else

    {

        pid2 = fork();

        if (pid2 < 0)

        {

            printf("\n2\_Error while creaating the child.");

            return;

        }

        if (pid2 == 0)

        {

            close(pipefd[1]);

            dup2(pipefd[0], STDIN\_FILENO);

            close(pipefd[0]);

            if (execvp(parAfterPipeArgument[0], parAfterPipeArgument) < 0)

            {

                printf("\n2\_Error while executing the command.");

                exit(0);

            }

        }

        else

        {

            wait(&wst);

            wait(&wst);

        }

    }

}

int pipeParsingFunc(char\* ipstringarg, char\*\* befAftPipeDuoArg)

{

    befAftPipeDuoArg[0] = strsep(&ipstringarg, "|");

    if (befAftPipeDuoArg[0] != NULL)

    {

        befAftPipeDuoArg[1] = strsep(&ipstringarg, "|");

    }

    if (befAftPipeDuoArg[1] != NULL)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

void parseWords(char\* ipstringarg, char\*\* parPipeArg)

{

    int i;

    for (i = 0; i < WRDLENMAX; i++)

    {

        parPipeArg[i] = strsep(&ipstringarg, " ");

        if (parPipeArg[i] == NULL)

        {

            break;

        }

        if (strlen(parPipeArg[i]) == 0)

        {

            i--;

        }

    }

}

int strProcess(char\* ipstringarg, char\*\* parBeforePipeArg, char\*\* parAfterPipeArg)

{

    char\* befAftPipeDuo[2];

    int cmdTypeRet = 0;

    cmdTypeRet = pipeParsingFunc(ipstringarg, befAftPipeDuo);

    if (cmdTypeRet)

    {

        parseWords(befAftPipeDuo[0], parBeforePipeArg);

        parseWords(befAftPipeDuo[1], parAfterPipeArg);

    }

    else

    {

        parseWords(ipstringarg, parBeforePipeArg);

    }

    return cmdTypeRet;

}

int main()

{

    char ipstring[CMDLENMAX], \*parBeforePipe[WRDLENMAX];

    char\* parAfterPipe[WRDLENMAX];

    int cmdType = 0;

        printf("\n\n:;:;Project:Shell;:;:\n\n");

    while (1)

    {

        if (getIpstr(ipstring))

        {

            continue;

        }

        cmdType = strProcess(ipstring, parBeforePipe, parAfterPipe);

        if(cmdType)

        {

                    cmdExecutionWithPipe(parBeforePipe, parAfterPipe);

        }

            else

        {

                    cmdExecutionWithoutPipe(parBeforePipe);

        }

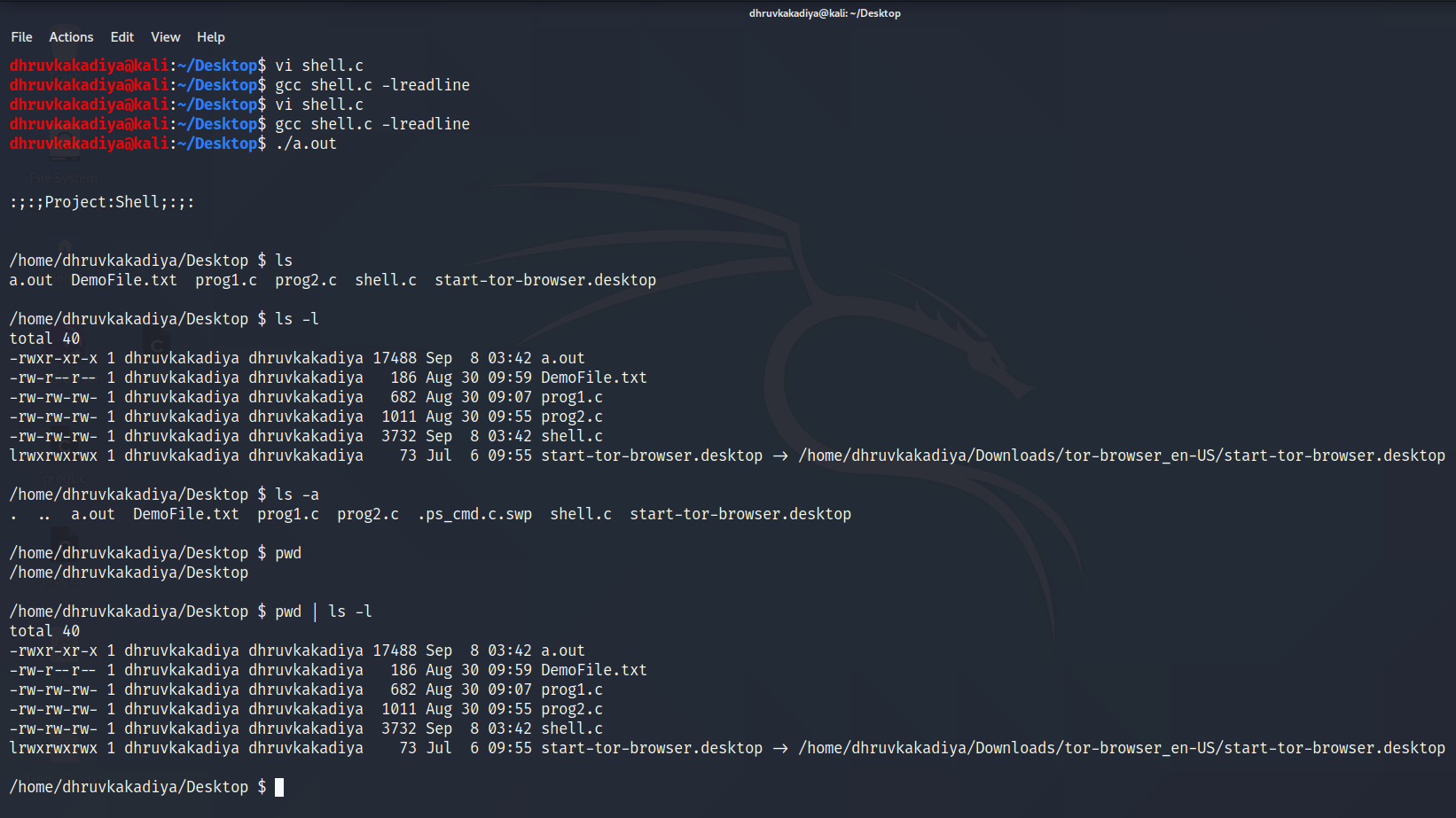
    }

    return 0;

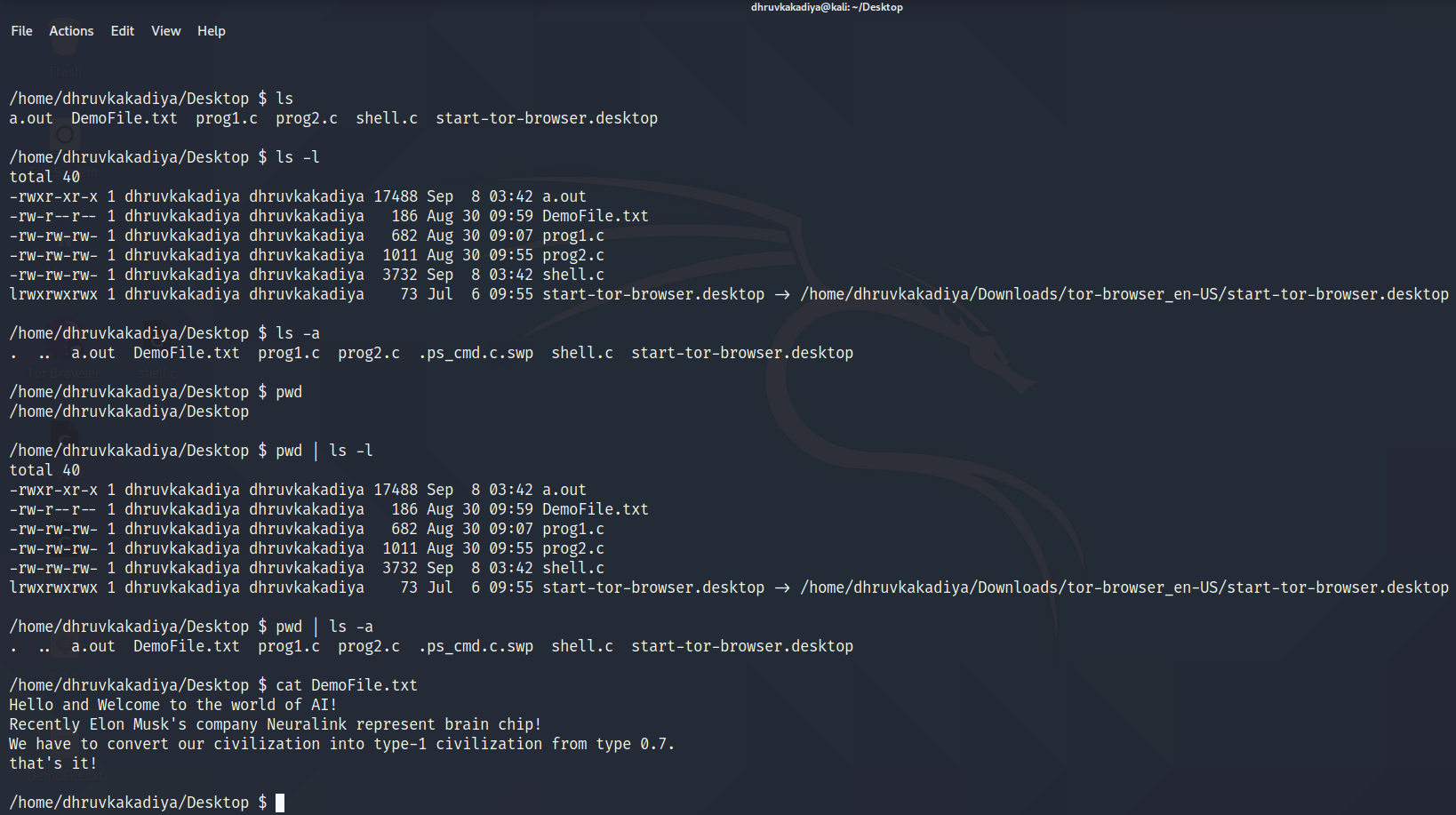
}

Outputs :-

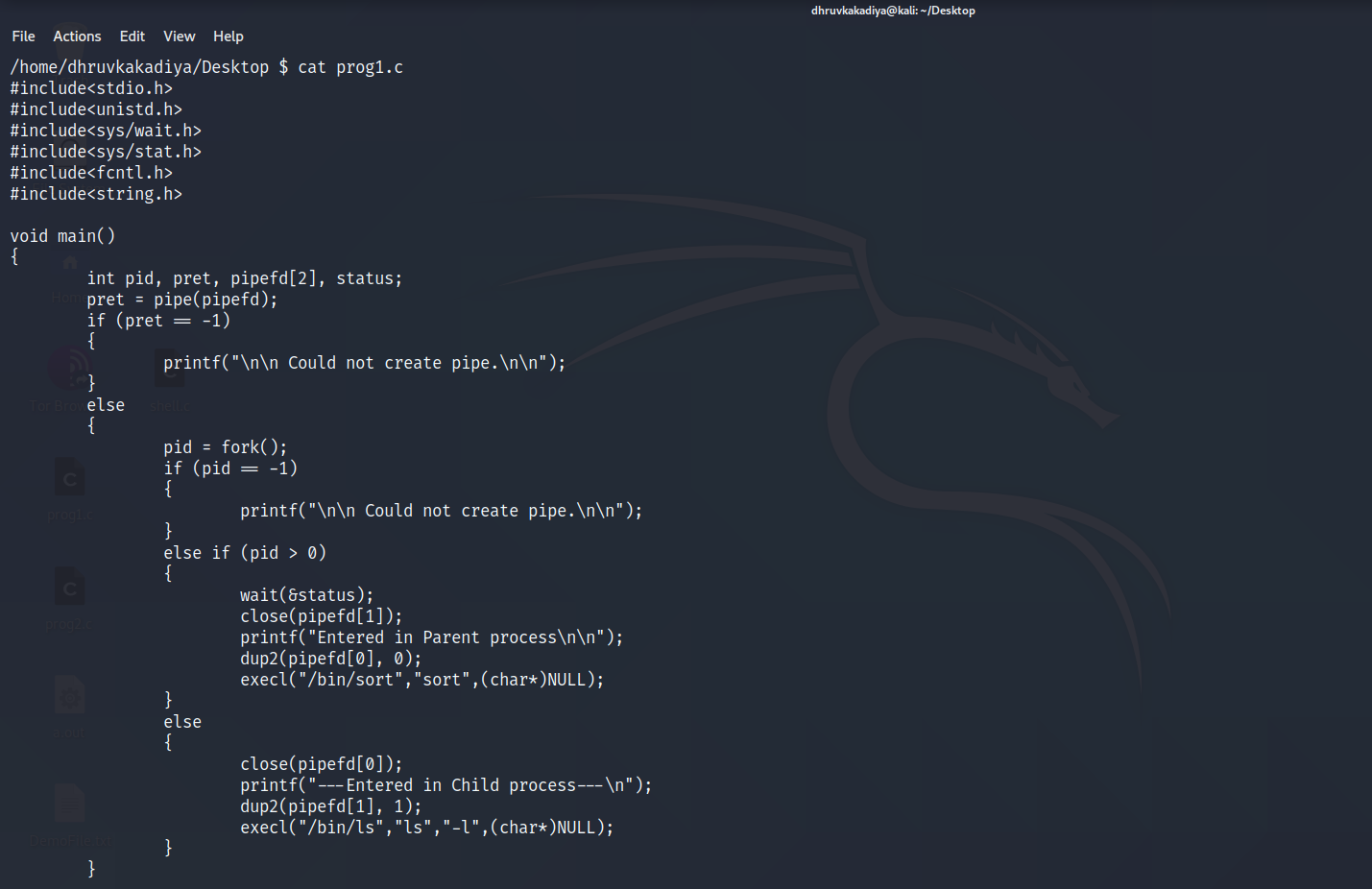
1.



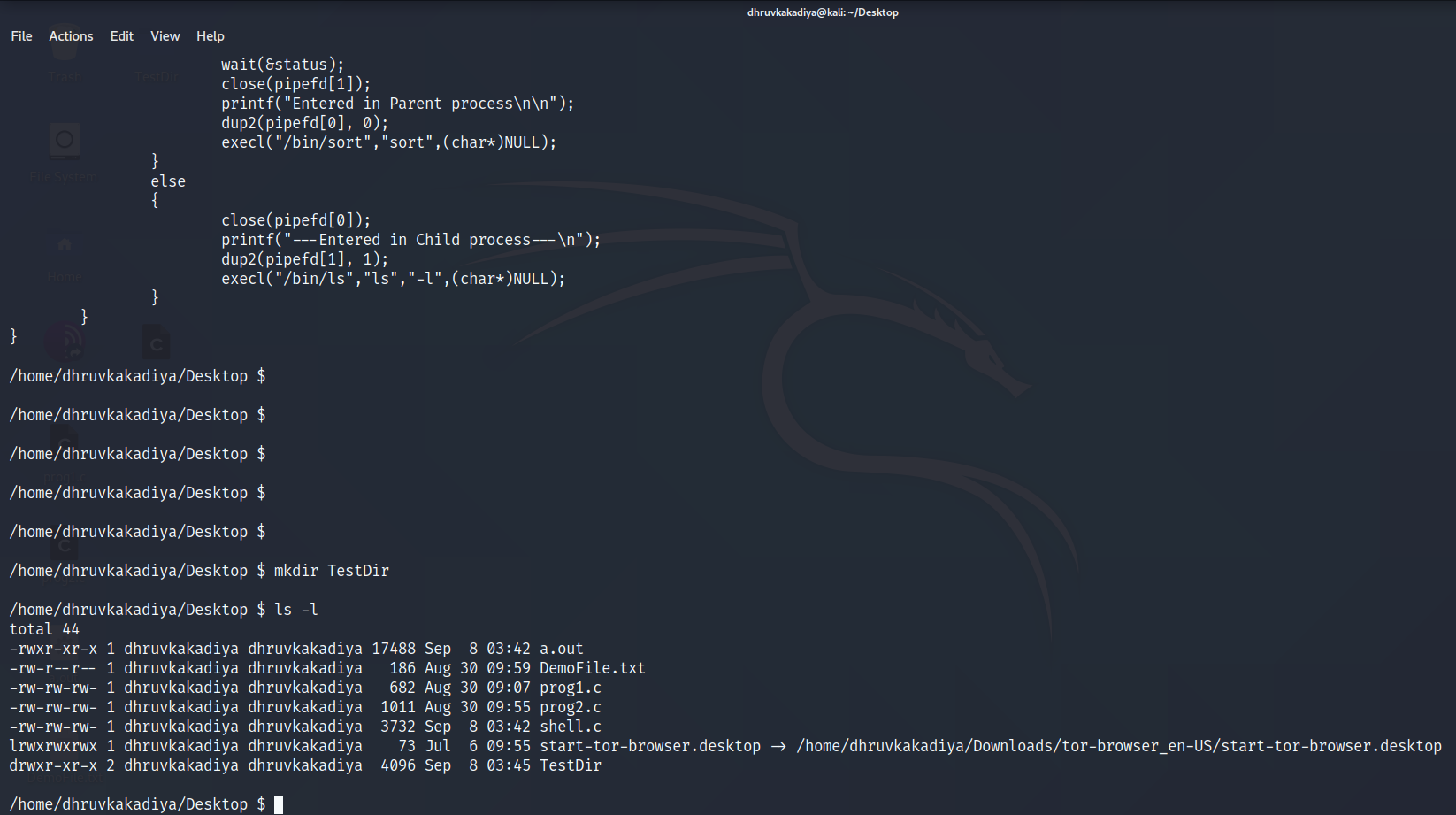
2.



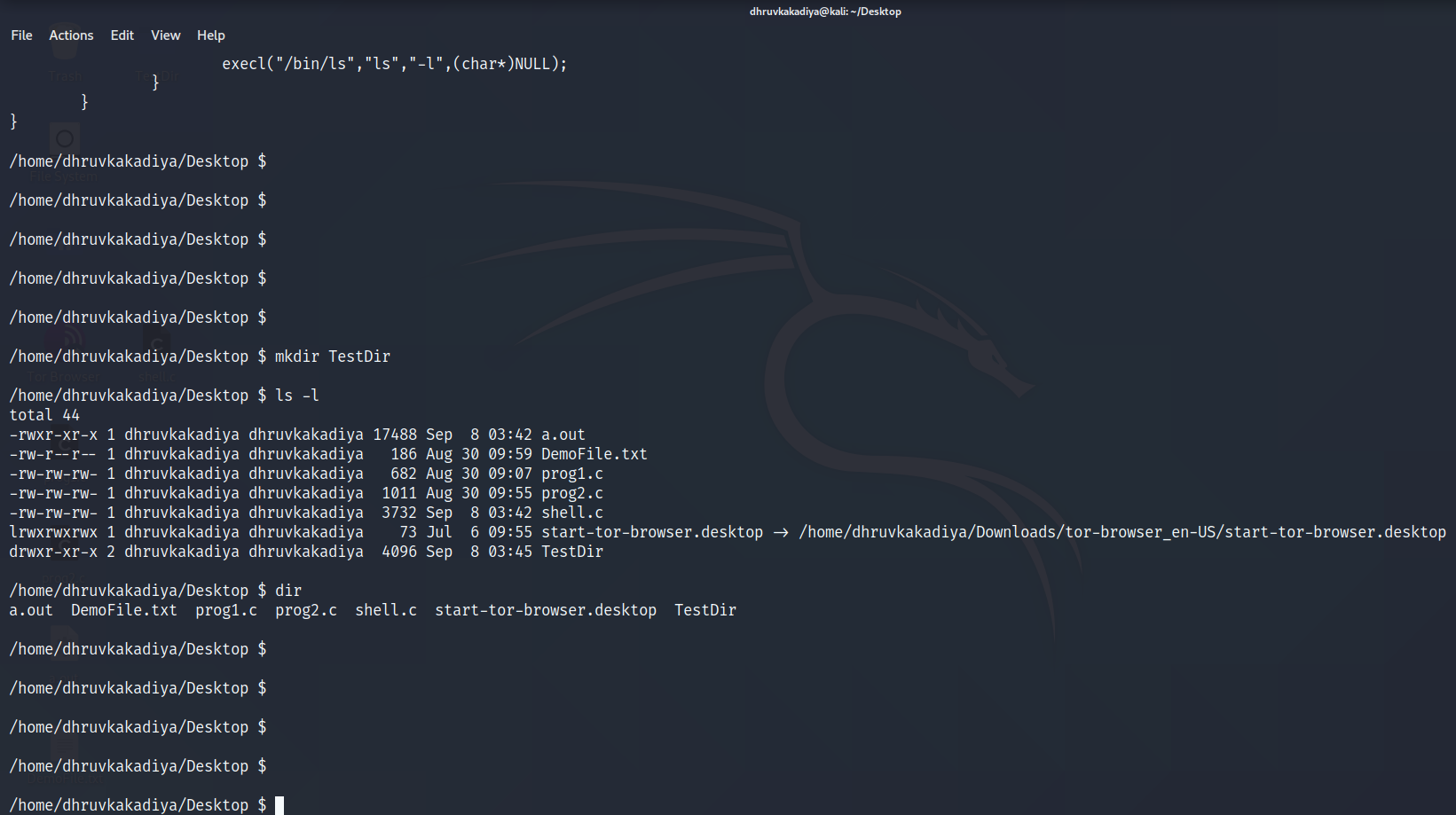
3.



4.



5.



6.

