Linux Shell Implementation in C

Name : Yash Aggarwal
2019480

This assignment is about the implementation of the shell using the c language.

The commands implemented are of two type :

1) Internal commands
2) External commands

The Internal commands implemented are as follows :

1) cd : This command is used to change the current directory to the specified directory. If a directory exists at the given location, then we see that the cd <directory name> command will take us to the specified directory name.

   Implementation In C :

   To implement this command in C, we use the chdir system call. This system call takes us to the specified directory. chdir is declared in the unistd.h header file.

Simple Implementation :

```
terminal >>> :
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  cd ..
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480 >>>  cd question_1
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_1 >>>
```

The flags implemented are -P. This flag allows us to axis the physical location given to us.
Error Handling :

1) In case there is no space after the command, the program returns the error that the command is not found.
2) Also, if there is no space after the flags, then a similar command error() is thrown.
3) If any other flag is used, then the command returns that this “flag does not exist”.

```
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480 >>>  cd -L
this flag is not supported
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480 >>>
```

Using the cd ~ command it goes back to the /home/yash/ directory :

```
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480 >>>  cd ~
/home/yash >>>
```

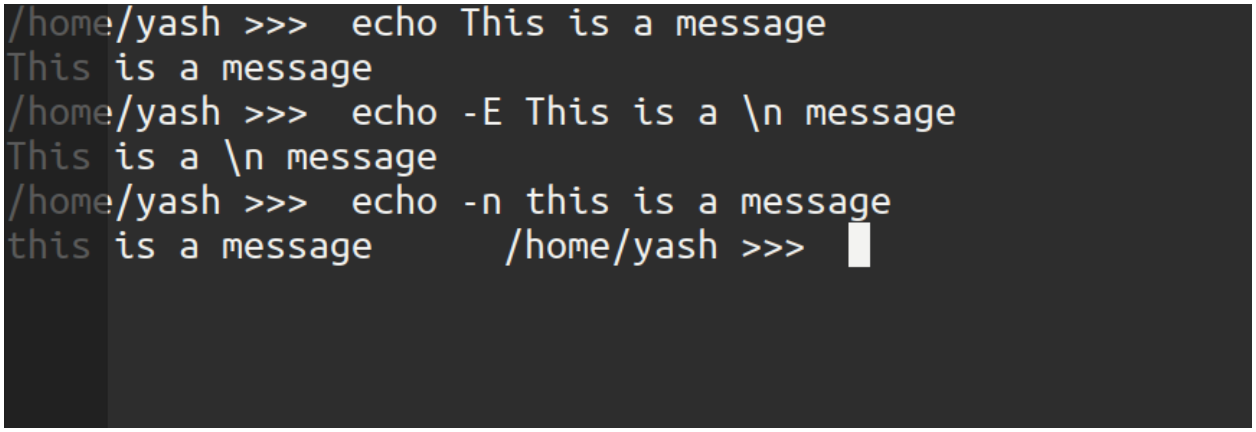
If the directory does not exist, then it prints the error using the errno.h directory using the following commands :

```
int val;
val = chdir(command);
if(val != 0){
     printf("%s\n", strerror(errno));
}
else{
      continue;
}
```

For example :

/home/yash >>> cd this_does_not_exist
No such file or directory
/home/yash >>>

2) Echo : This command is used to print the message that is sent along with it. The flags implemented using the echo command are -E and -n. -E flag is used to ignore the escape sequences that are passed in the d message that is to be printed. -n flag is used to prevent the printing of the new line in the terminal.



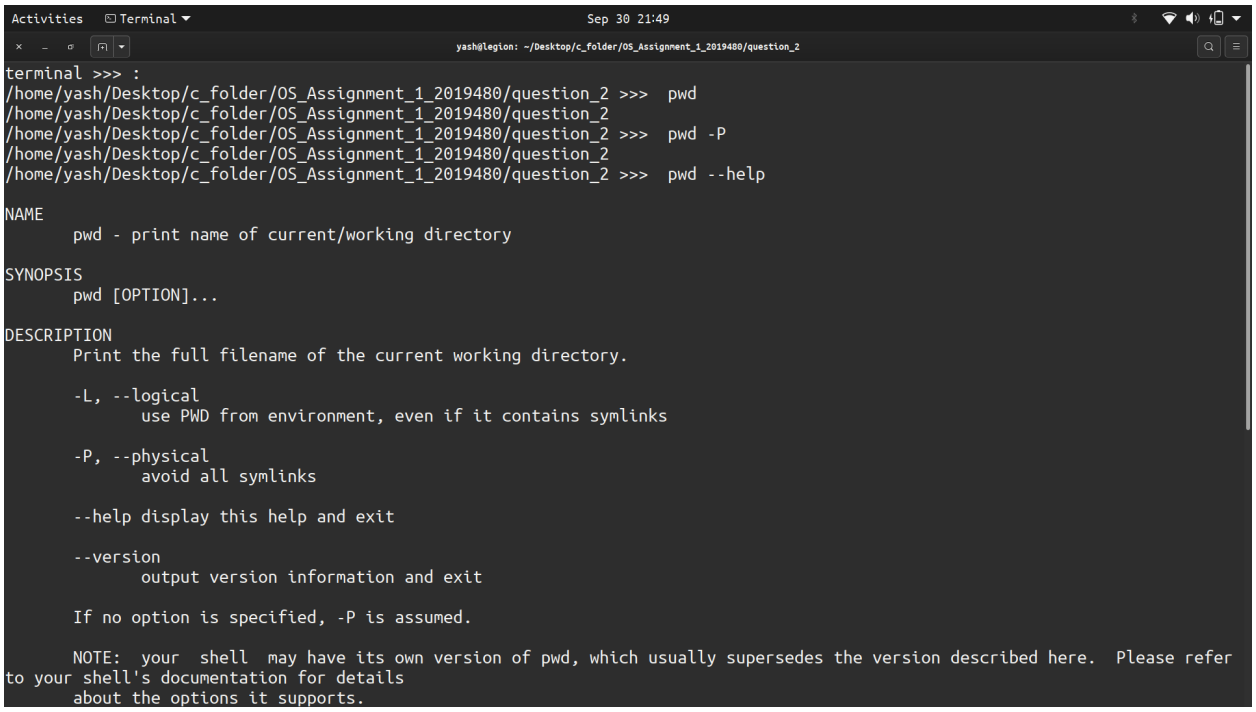
We also check for the following bugs :

i) there is no space between the command and the message.
ii) more than one space left between the command and the message.
iii) proper handling of the flags. If an unauthorized flag is passed, it raises an error and passes elegantly.

3) Pwd command : This command is used to print the current working directory. The -P flag is implemented, which simply prints the physical memory location of the current working directory.To print the directory, we use the getcwd() system call. It helps in printing the location of the current directory that we are working on.

Also, the --help flag is implemented which allows us to check the help for the given pwd command.

Here is a sample of the working of the pwd command :

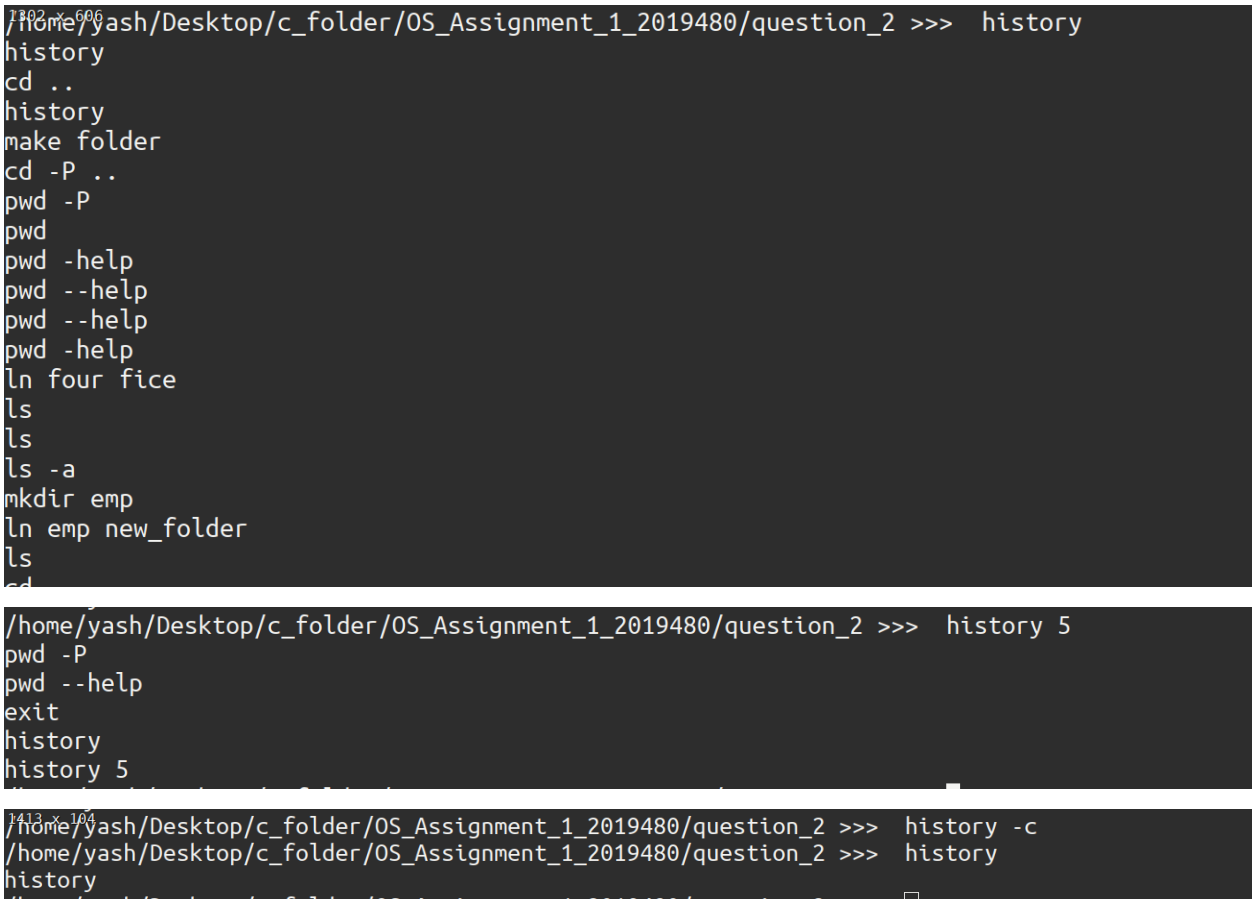


The errors and bugs handled are :
i) if there is no space command, an error is raised.
ii) while opening the pwd_help file, open() system call is used. If it cannot open the file then it raises an error.
iii) flags are also checked for any error.

4) Exit : This command is used to exit the shell. No flag is implemented along with this command. The command works as follows :

5) History : This command is used to print the history of all the previous commands that we have executed in our directory upto this point. The flags implemented are -a and -c. Also, if we pass a number along with the history command then those many previous commands are printed. The errors handled are :
    1) Handling the error for the open() command while opening the history.txt file, where the previous commands are stored.
    2) Handling flags properly.
    3) Handling the spacing of the commands and the flags.
    4) Handling alpha numerics while printing the no.of previous commands.

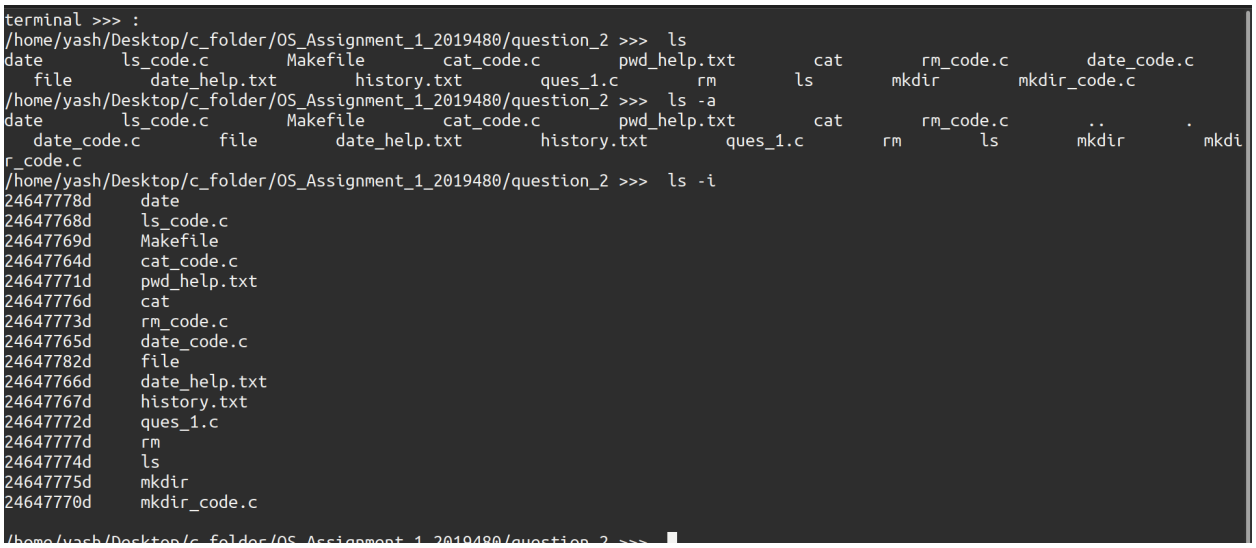
    The implementation is as follows :



2) External Commands : The external commands implemented are :

1) Ls : This command is used to print all the files and directories existing in the given directory. The flags implemented are -a and -i. The -a flag is used to print all the hidden files in the given directory. The -i flag is used to print the index of each file.
   Error handling is done as follows :
   1) Error handling is done while using the opendir() system call. This system call is used to open the given directory so that the directories and files in it can be opened. It is a part of the dirent.h header file.
   2) Error handling is done while dealing with the flags used for this command.
   3) Error handling is done while using the fork command. If forking is not successful, then a message is printed and the program moves to the next command.

   An implementation of this program is as follows :



2) mkdir command : This command is used to create new directories in the given directory. The flags used for the mkdir command are -m and -v. The -m flag is used to define the mode in which the directory is created and -v flag is used to print a message when the directory is created.

Error handling is done as follows :

1) While forking(), if the child process is not created (pid value < 0), then the program prints a message and moves to the next command.
2) Error handling is done while parsing the flags.
3) For creating multiple folders, bugs are handled.

Implementation of mkdir is as follows :

```
terminal >>> :
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  mkdir one
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  mkdir -v two
mkdir : directory created two
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  mkdir -m three
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  ls
date        ls_code.c        three        Makefile        cat_code.c        pwd_help.txt        cat        rm_code.c        dat
e_code.c        file        date_help.txt        history.txt        ques_1.c        rm        ls        mkdir        two
 one        mkdir_code.c
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>
```

3) rm command : This command is used to remove the existing files of a directory. Flags implemented are as follows :

1) -v : this flag is used to print a message when a file is deleted
2) -i : this is used to ask for permission before actually trying to remove a file.

Error handling is done as follows :

i) error handling is done to find if a file has already been deleted or not. It is done using the remove() command available in c. If the file is removed it returns 0 . Else, error is printed.

ii) error handling is also done while forking.

iii) The given flags can be used as a pair and the errors are handled for the same.

An implementation is as follows :

```
1906 × 390
terminal >>> :
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  mkdir empty
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  ls
date        ls_code.c        Makefile        empty        cat_code.c        pwd_help.txt        cat
e_code.c        file        date_help.txt        history.txt        ques_1.c        rm        ls
.c
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  rm -v empty
removed 'empty'
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  rm -i empty
rm : remove regular file empty ? (y/n)    y
File cannot be removed/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>
```

4) cat command : This command is used to print the contents of a file on the terminal. The flags used are -n and -e. The -n flag is used to print the number of the line for each new line, whereas the -e flag is used to print $ at the end of each file.

This command follows similar error handling procedures as the previous commands. It is implemented as follows :

```
terminal >>> :
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  cat -n cat_code.c
1    #include<stdio.h>
2    #include<stdlib.h>
3    #include<sys/types.h>
4    #include<sys/stat.h>
5    #include<unistd.h>
6    #include<dirent.h>
7    #include<string.h>
8    #include <fcntl.h>
9    #include <stdarg.h>
10
11    void catError(){
12        printf("No file found\n");
13    }
14    void commandError(){
15        printf("Command not found\n");
16    }
17    void doFunc(int fd, int flag1, int flag2){
18        char buff;
19        int n = 1;
20        int k = 0;
21        int f= 1;
22        while(read(fd, &buff, 1) > 0){
23            if(flag1 == 1 && f == 1){
```

```
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  cat -e cat_code.c
#include<stdio.h> $
#include<stdlib.h> $
#include<sys/types.h> $
#include<sys/stat.h> $
#include<unistd.h> $
#include<dirent.h>$
#include<string.h>  $
#include <fcntl.h>$
#include <stdarg.h>$
$
void catError(){$
    printf("No file found\n"); $
}$
void commandError(){$
    printf("Command not found\n"); $
}$
void doFunc(int fd, int flag1, int flag2){$
    char buff; $
    int n = 1; $
    int k = 0; $
    int f= 1; $
```

```
terminal >>> :
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  cat cat_code.c date_code.c
#include<stdio.h>
```

5) Date : This command is used to print the current date, which is the local time. The flags used are :

1) -u -> it is used to print the GMT time
2) -h -> this is used to print the help for the date command. (as implemented in the original command line).

Error handling is done for :

1) Opening the file for reading the help (as implemented in the command line).
2) Handling the flags
3) Handling spaces.

The implementation is as follows :

```
terminal >>> :
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  date
Wed Sep 30 22:41:07 2020
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  date -u
30 - 9 - 2020    17 : 11 : 9 UTC
/home/yash/Desktop/c_folder/OS_Assignment_1_2019480/question_2 >>>  date -h
Usage: date [OPTION]... [+FORMAT]
  or:  date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
Display the current time in the given FORMAT, or set the system date.

Mandatory arguments to long options are mandatory for short options too.
  -d, --date=STRING          display time described by STRING, not 'now'
      --debug                annotate the parsed date,
                              and warn about questionable usage to stderr
  -f, --file=DATEFILE        like --date; once for each line of DATEFILE
  -I[FMT], --iso-8601[=FMT]  output date/time in ISO 8601 format.
                               FMT='date' for date only (the default),
                               'hours', 'minutes', 'seconds', or 'ns'
                               for date and time to the indicated precision.
                               Example: 2006-08-14T02:34:56-06:00
  -R, --rfc-email            output date and time in RFC 5322 format.
                               Example: Mon, 14 Aug 2006 02:34:56 -0600
      --rfc-3339=FMT         output date/time in RFC 3339 format.
                               FMT='date', 'seconds', or 'ns'
                               for date and time to the indicated precision.
                               Example: 2006-08-14 02:34:56-06:00
  -r, --reference=FILE       display the last modification time of FILE
  -s, --set=STRING           set time described by STRING
  -u, --utc, --universal     print or set Coordinated Universal Time (UTC)
      --help     display this help and exit
      --version  output version information and exit
```