**The National University of Lesotho**

**Department of Mathematics and Computer Science**

**Faculty of Science and technology**



**CS3521: Operating Systems**

**Task:** Assignment Report

**Due:** 08/05/24

**Participants:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Student Number** | **Surname, initials** | **Major** |
| 1. | 202200016 | Seutloali. K | B.Eng. Systems & Networks |
| 2. | 202101740 | Konyana. R | B.Eng. Electronics |
| 3. | 202100117 | Melato. R | B.Eng. Systems & Networks |
| 4. | 202100087 | Mokuku. L | B.Eng. Systems & Networks |
|  |  |  |  |

[1. Introduction 3](#_Toc166050938)

[1.1. What to expect? 3](#_Toc166050939)

[2. Aim and Progress 3](#_Toc166050940)

[2.1. Objective Overview 3](#_Toc166050941)

[2.2. Achievements 3](#_Toc166050942)

[3. Functions and Libraries 4](#_Toc166050943)

[3.1. Libraries 4](#_Toc166050944)

[3.2. What do the libraries do? 4](#_Toc166050945)

[3.3. Function Declarations 6](#_Toc166050946)

[3.4. Function Descriptions & Pseudo Code (How do they work?) 6](#_Toc166050947)

[4. Implementation 10](#_Toc166050948)

[4.1. How does the program work? 10](#_Toc166050949)

[4.2. Flowchart 11](#_Toc166050950)

[5. Additional Functionalities 12](#_Toc166050951)

[5.1. Functions, their use and importance 12](#_Toc166050952)

[a) Tab Completion 12](#_Toc166050953)

[b) Signal Handling 12](#_Toc166050954)

[c) mytwo() 12](#_Toc166050955)

[5.2. Bonus Question 12](#_Toc166050956)

[6. Myshell Execution 13](#_Toc166050957)

[6.1. myhelp() 13](#_Toc166050958)

[6.2. mycd() 13](#_Toc166050959)

[6.3. myenviron() 13](#_Toc166050960)

[6.4. mydir() 13](#_Toc166050961)

[6.5. myclr() 13](#_Toc166050962)

[6.6. myone() 13](#_Toc166050963)

[6.7. myecho() 13](#_Toc166050964)

[6.8. mypause() 13](#_Toc166050965)

[6.9. myquit() 13](#_Toc166050966)

[6.10. \*Bonus 13](#_Toc166050967)

[6.11. \*mytwo() 13](#_Toc166050968)

# 

# 

# Introduction

### What to expect?

This report provides an insightful overview of our group's journey in developing the MyShell command line interpreter. The documentation provides a detailed exploration of the objectives, achievements, functionalities, and implementation strategies of the MyShell assignment.

# Aim and Progress

### Objective Overview

The primary objective of the MyShell assignment is to provide hands-on experience in developing a simple Linux shell or command line interpreter. By undertaking this task, our group gained a deeper understanding of operating system principles and the functionality of shells as fundamental user interfaces. Through the implementation of essential internal commands such as directory navigation (‘mycd’), screen clearing (‘myclr’), directory listing (‘mydir’), and environment variable display (‘myenviron’), the group learned key concepts in system programming and command execution. Moreover, the assignment challenged us to design and implement a new command (‘myone’) with unique functionality, encouraging creativity and problem-solving skills. Overall, the assignment aims to foster practical skills in C programming, system call usage, error handling, and software design while reinforcing theoretical knowledge of operating system fundamentals.

### Achievements

The MyShell environment has successfully fulfilled all its expected functions and even went a step further by adding extra features to make it more user-friendly. Our team worked hard to make sure that the shell not only does what it's supposed to but also goes above and beyond to improve the user experience.

# Functions and Libraries

### Libraries

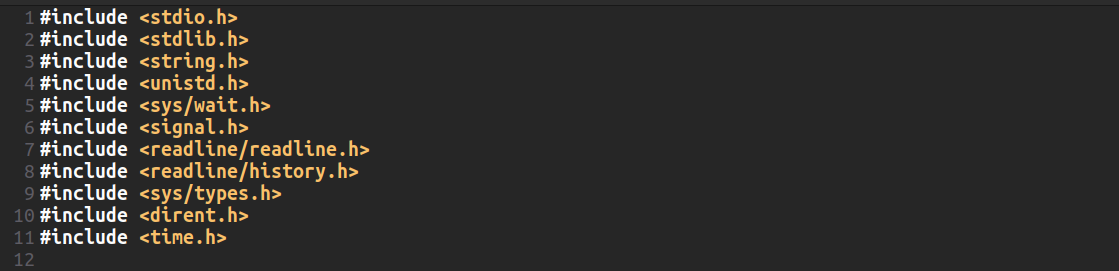


Figure 3. 1 Libraries included

### What do the libraries do?

1. **stdio.h**: This library provides input and output functionalities, such as ‘printf’ and ’scanf’, for reading from and writing to files and the console.  
   **Example Use**: It's used for standard input/output operations, such as printing messages to the console using ’printf’ and reading user input using ‘scanf’.
2. **stdlib.h**: This library contains general utility functions, such as memory allocation (’malloc, free’), random number generation (’rand, srand’), and program termination (’exit’).  
   **Example Use**: It's used for dynamic memory allocation (’malloc’), freeing allocated memory (’free’), and exiting the program (’exit’).
3. **string.h**: This library provides various string manipulation functions, such as copying strings (’strcpy’), concatenating strings (’strcat’), and comparing strings (‘strcmp’).  
   **Example Use**: It's used for manipulating strings, such as comparing command names, arguments, and environment variables.
4. **unistd.h**: This library provides access to various POSIX operating system API functions, such as file handling (’open, close’), directory manipulation (’chdir’, ‘mkdir’), and process management (‘fork’, ‘exec’).  
   **Example Use**: It's used for system calls like ’chdir’ to change the current working directory and fork to create child processes.
5. **sys/wait.h**: This library contains functions and macros related to process termination and status reporting, particularly for handling child processes.  
   **Example Use**: It's used for waiting for child processes to terminate using ’waitpid’.
6. **signal.h**: This library provides facilities for handling signals, such as registering signal handlers (’signal’) and sending signals to processes (’kill’).  
   **Example Use**: It's used for handling signals like SIGINT (Ctrl+C) and SIGTSTP (Ctrl+Z) by registering signal handlers.
7. **readline/readline.h**: This library provides facilities for line editing during command-line input, including history management and tab completion.  
   **Example Use**: It's used for reading user input with line editing capabilities and maintaining command history.
8. **readline/history.h**: This library works in conjunction with ’readline.h’ and provides functions for managing command history.  
   **Example Use**: It's used for adding commands to the history list and accessing command history during tab completion.
9. **sys/types.h**: This library provides various data types used in system calls and other system-related functions.  
   **Example Use**: It's used for data types like ’pid\_t’ (process ID) and ’DIR’ (directory stream).
10. **dirent.h**: This library provides functions and data types for directory handling, such as opening directories (’opendir’), reading directory entries (’readdir’), and closing directories (’closedir’).  
    **Example Use**: It's used for listing directory contents with functions like ’opendir’, ’readdir’, and ’closedir’.
11. **time.h**: This library provides functions for manipulating time, such as getting the current time (’time’), formatting time (’strftime’), and calculating time differences (’difftime’).  
    **Example Use**: It's used for generating timestamps and calculating execution times in the shell.

### Function Declarations

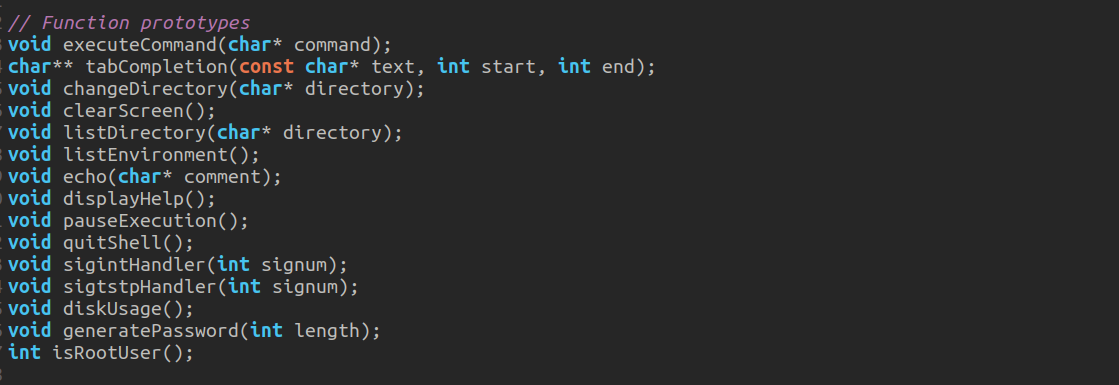


Figure 3. 2 Declared functions

### Function Descriptions & Pseudo Code (How do they work?)

1. mycd()

|  |
| --- |
| function changeDirectory(directory):  if directory is NULL:  // Display current directory  printCurrentDirectory()  else:  // Change directory  if directoryExists(directory):  setNewDirectoryAsCurrent(directory)  updateEnvironmentVariablePWD(directory)  printSuccessMessage(directory)  else:  printErrorMessage(directory) |

Figure 3. 3 mycd implemented by changeDirectory function

The changeDirectory function serves as the core mechanism for handling directory changes within the shell. When invoked with a directory path as its input, it attempts to change the current working directory to the specified path. If no directory path is provided, it displays the current working directory instead. To determine the existence of the specified directory, the function internally relies on the directoryExists helper function. Upon successfully changing the directory, it updates the environment variables PWD and OLDPWD to reflect the new and previous current working directories, respectively. This update ensures that the shell maintains a record of the previous directory for reference. Additionally, upon successful directory change, it may print a success message confirming the operation. However, if the specified directory does not exist, it prints an error message to inform the user of the issue.

1. myclr()

|  |
| --- |
| function clearScreen():  callSystemCommandToClearScreen() |

Figure 3. 4 myclr implemented by clearScreen function

The clearScreen function in the provided code is responsible for clearing the terminal screen. When invoked, it calls a system command, specifically "clear" which is a common Unix command used to clear the contents of the terminal screen. This command effectively removes all text and output from the terminal window.

1. myenviron()

|  |
| --- |
| function listEnvironment():  for each environmentVariable in environment:  print environmentVariable.name + "=" + environmentVariable.value |

Figure 3. 5 myenviron implemented by listEnvironment function

The listEnvironment function within the provided code is designed to facilitate the display of environment variables within the shell environment. Upon invocation, this function systematically iterates through all available environment variables, accessing each variable's name and corresponding value. Utilizing this information, the function then proceeds to print out a comprehensive list, wherein each entry showcases the name-value pair of the respective environment variable.

1. myecho()

|  |
| --- |
| function echo(comment):  if comment is not NULL:  Initialize an index variable i to 0 and a new index variable j to 0.  Iterate through each character of the comment:  Check if the current character is not a space or a tab.  If true, copy the character to the output string at index j and increment j.  Check if the current character is a space or a tab.  If true, check if it is preceded by another space or tab or if it's the first character.  If false, replace the consecutive spaces or tabs with a single space in the output string at index j and increment j.  Move to the next character in the comment.  Null-terminate the output string to ensure it ends properly.  Print the modified comment.  else:  Print a new line. |

Figure 3. 6 myecho implemented by echo function

Upon invocation, this function accepts a comment or message as input. If the input comment is not NULL, indicating the presence of a message, the function proceeds to print out the comment to the terminal. However, before printing, the echo function incorporates a mechanism to enhance readability by ensuring that multiple consecutive spaces or tabs within the comment are reduced to a single space. In cases where the input comment is NULL, implying the absence of a specific message, the echo function gracefully handles the situation by printing a new line to the terminal.

1. mydir()

|  |
| --- |
| function listDirectory(directory):  if directory is NULL:  callSystemCommandToListContentsOfCurrentDirectory()  else:  if directoryExists(directory):  callSystemCommandToListContentsOfDirectory(directory)  else:  printErrorMessage(directory) |

Figure 3. 7 mydir implemented by listDirectory function

Upon invocation, this function is capable of accommodating two distinct scenarios based on the presence or absence of a directory path provided as input. When provided with a directory path, the function first verifies the existence of the specified directory. If the directory exists, it proceeds to execute a system command to list the contents of the specified directory, thereby providing users with a comprehensive view of its contents. However, in cases where no directory path is provided, the function defaults to listing the contents of the current directory by executing the ‘ls’ system command. Moreover, to ensure robust error handling, the function incorporates a mechanism to detect non-existent directories. In the event that the specified directory does not exist, the function promptly notifies the user by printing an error message to the terminal.

1. mypause()

|  |
| --- |
| function pauseExecution():  print "Press Enter to continue..."  flushOutputBuffer()  waitForEnterKeyPress() |

Figure 3. 8 mypause implemented by pauseExecution function

Upon invocation, this command initiates a pause in the execution flow, effectively suspending any further processing until a specific user action is taken. Specifically, the shell prompts the user with a message instructing them to press the Enter key to resume execution. During this pause, the shell remains inactive, awaiting user input. Users cannot interact with other commands or perform additional actions until they press Enter to resume execution. Once the Enter key is pressed, signalling their readiness to continue, the shell gracefully resumes its execution flow, enabling the continuation of any pending operations or tasks.

1. myone()

|  |
| --- |
| function generatePassword(length):  if length > 0:  charset = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&\*()-\_=+"  seedRandomNumberGenerator()  print "Generated password: "  for i from 1 to length:  print randomCharacterFromCharset(charset)  print newLine  else:  print "Invalid password length" |

Figure 3. 9 myone implemented by generatePassword function

Upon invocation, users specify the desired length for the password they wish to generate. Subsequently, the command utilizes a predefined character set, consisting of alphanumeric characters and special symbols, from which it randomly selects characters to construct the password. To achieve this, the command employs a pseudorandom number generator seeded with the current system time, ensuring a different sequence of characters is generated with each invocation. The length specified by the user determines the number of characters to be included in the generated password. Additionally, the index calculations for selecting characters from the character set are determined by the random numbers generated by the pseudorandom number generator. This process ensures that each character in the password is independently and randomly chosen from the character set, enhancing the security and unpredictability of the generated passwords. Once the password is constructed, it is displayed to the user, providing a convenient and efficient means of generating strong, unique passwords for various applications.

1. myquit()

|  |
| --- |
| function quitShell():  print "Exiting MyShell..."  exit(0) |

Figure 3. 10 myquit implemented by quitShell function

Upon invocation, this command initiates the termination process, prompting the shell to gracefully shut down and relinquish control back to the parent environment. This action effectively halts any ongoing operations or tasks within the shell, ensuring that all resources are properly released and any open connections or processes are appropriately terminated. Once the termination process is complete, the user is seamlessly transitioned out of the shell environment and returned to the parent environment.

# Implementation

### How does the program work?

This is a custom shell implementation which provides users with a basic command-line interface. It allows users to execute various predefined commands and provides features such as tab completion, signal handling for Ctrl+C and Ctrl+Z, and coloured prompts to indicate whether the user is root or not. These prompts include the user’s username, hostname, and current directory. The prompt is initialized by the main function, which also sets up tab completion by assigning the tabCompletion() function to the rl\_attempted\_completion\_function. The main function then enters an infinite loop to read user commands using the readline() function. Upon receiving a command, the executeCommand() function is invoked. This function tokenizes the entered command by splitting it based on spaces and newlines, and then identifies the command by comparing it with the predefined commands stored in the predefinedCommands array. If a match is found, the corresponding function is called to execute the command's functionality. If the command is not recognized, an 'Unknown command' message is displayed. Once the command execution is complete, the prompt is displayed again, and the loop continues until the user decides to quit the shell.

### Flowchart

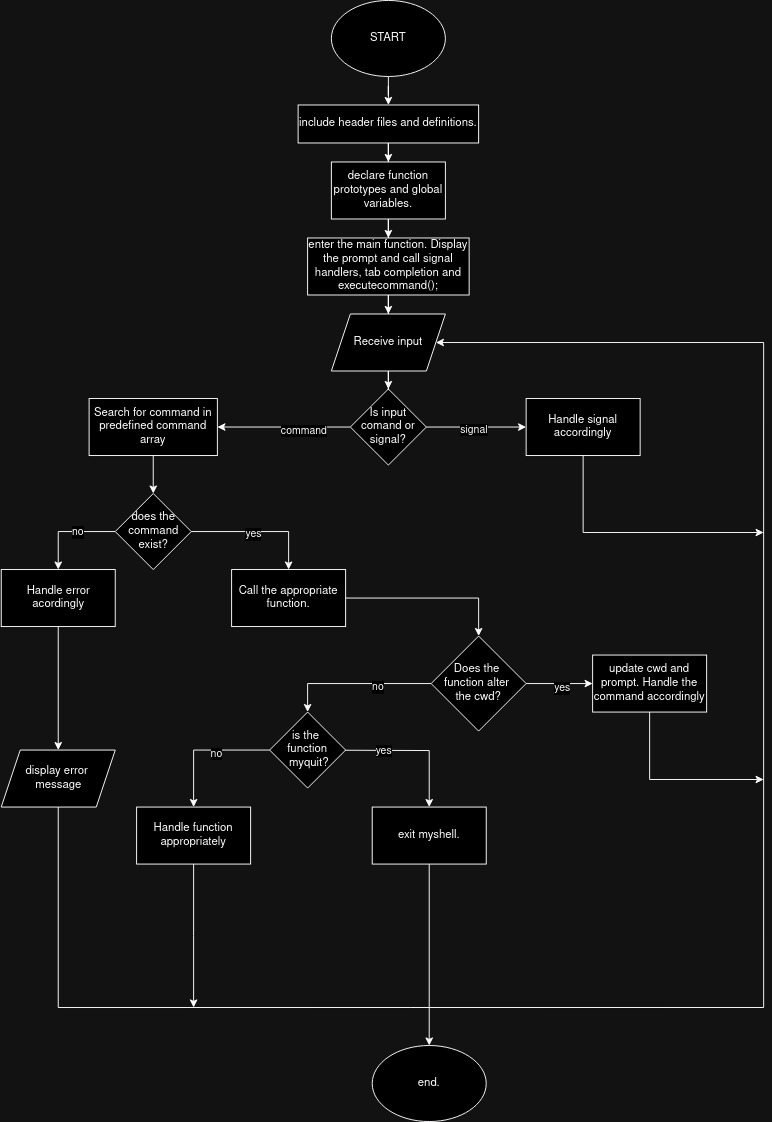


Figure 3. 11 Flow chart of how the program works

# Additional Functionalities

### Functions, their use and importance

#### Tab Completion

Tab completion is a feature implemented within the shell environment to enhance user productivity and efficiency during command-line input. When enabled, tab completion allows users to expedite the process of entering commands, file names, directory paths, and other input by automatically suggesting or completing partially entered text based on available options. Upon pressing the Tab key, the shell evaluates the current input context and provides a list of possible completions or suggestions. Users can then select from the available options or continue typing to refine their input further. Tab completion significantly reduces the likelihood of typing errors, accelerates command entry, and improves user interaction within the shell environment.

#### Signal Handling

Signal handlers are mechanisms implemented within the shell environment to manage and respond to various signals sent by the operating system or other processes. Signals represent asynchronous notifications or events that occur within the system, such as user interruptions (e.g., Ctrl+C for ’SIGINT’) or process status changes (e.g., termination signals like ’SIGTSTP’ (Ctrl+Z)). Signal handlers are functions programmed to intercept specific signals and execute predefined actions or routines in response. For example, signal handlers may be configured to handle signals like SIGINT (Ctrl+C) by terminating the current operation gracefully or ’SIGTSTP’ (Ctrl+Z) by suspending the execution of the current process. By utilizing signal handlers, the shell can effectively manage signal-based events, maintain system stability, and ensure a responsive and robust user experience.

#### mytwo()

The ’mytwo’ command in the shell facilitates the display of disk usage information, offering users insight into the utilization of available storage space across various file systems. Upon invocation, this command executes a system command (df -h) that retrieves and presents disk usage statistics in a readable format. The output typically includes details such as the total size, used space, available space, and usage percentage for each mounted file system. By providing this information, the ’mytwo’ command enables users to monitor disk usage levels, identify potential storage constraints, and make informed decisions regarding file management and resource allocation within the system.

### Bonus Question

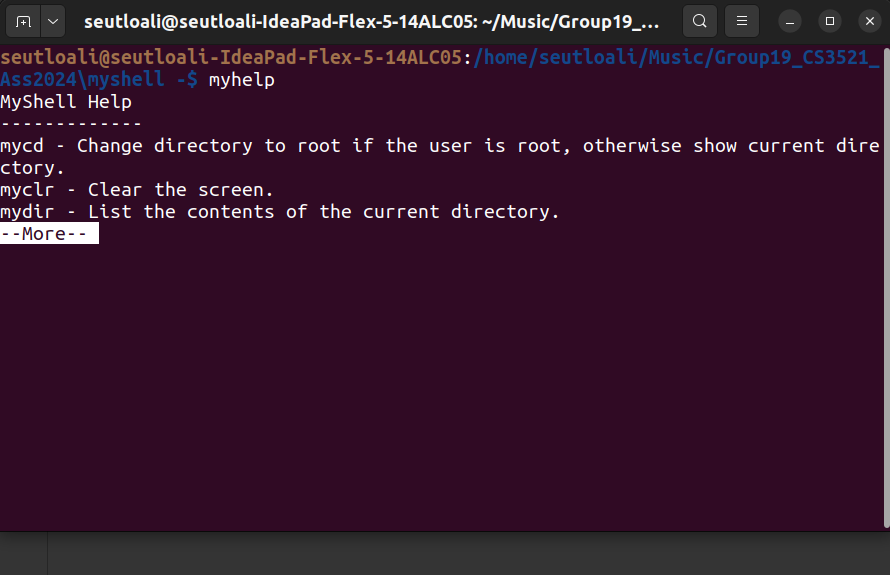
|  |
| --- |
| function executeCommand(command):  lowercaseCommand = convertToLowerCase(command) // Convert command to lowercase  // Remaining code executes as before, but with lowercaseCommand used for comparison |

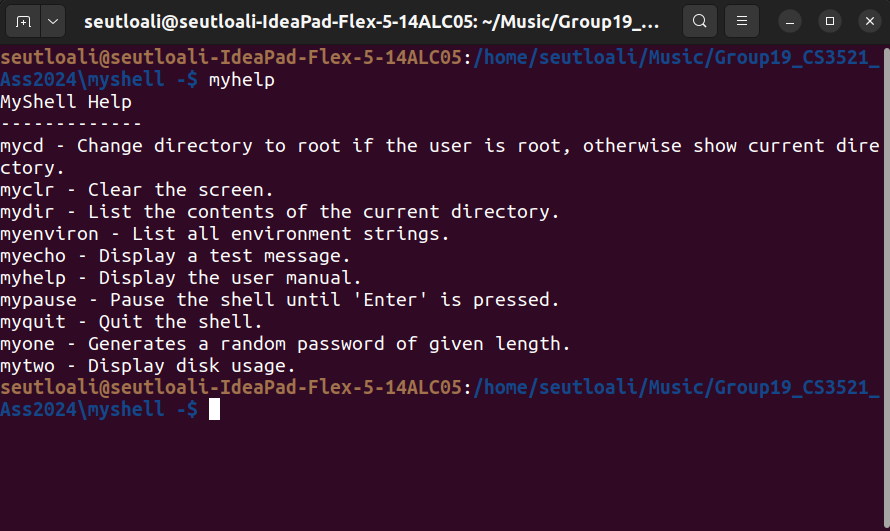
Figure 3. 12 case sensitivity implemented by executeCommand function

In the existing shell implementation, commands are directly compared against predefined commands using string comparison functions like ’strcmp’. To introduce optional case sensitivity, all incoming commands are converted to lowercase before processing. This conversion ensures uniformity in command comparison, regardless of user input case variations. Upon receiving a command, the shell employs a ‘tolower’ function and iterates through the command string, converting uppercase characters to lowercase. Subsequently, the converted command is compared against predefined commands. This approach allows users to enter commands in any case preference, ensuring consistent execution while enhancing usability and user experience within the shell environment.

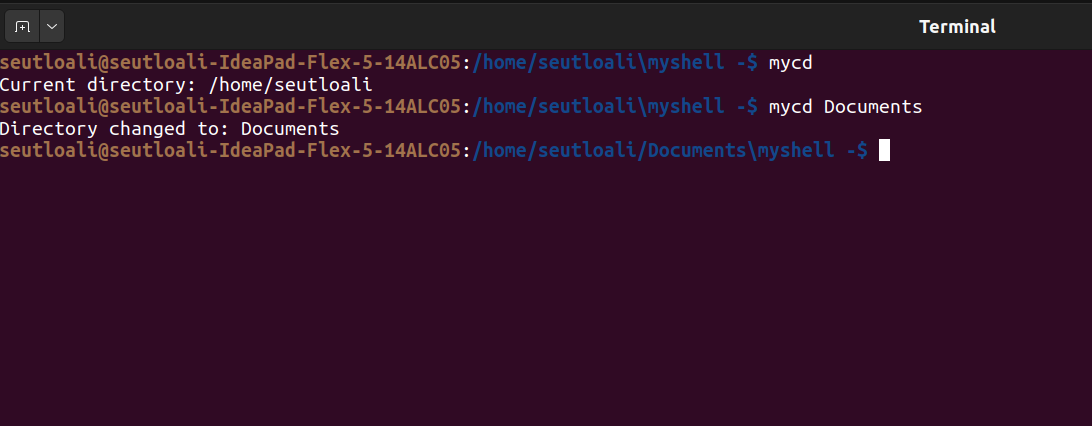
# Myshell Execution

### Myhelp()

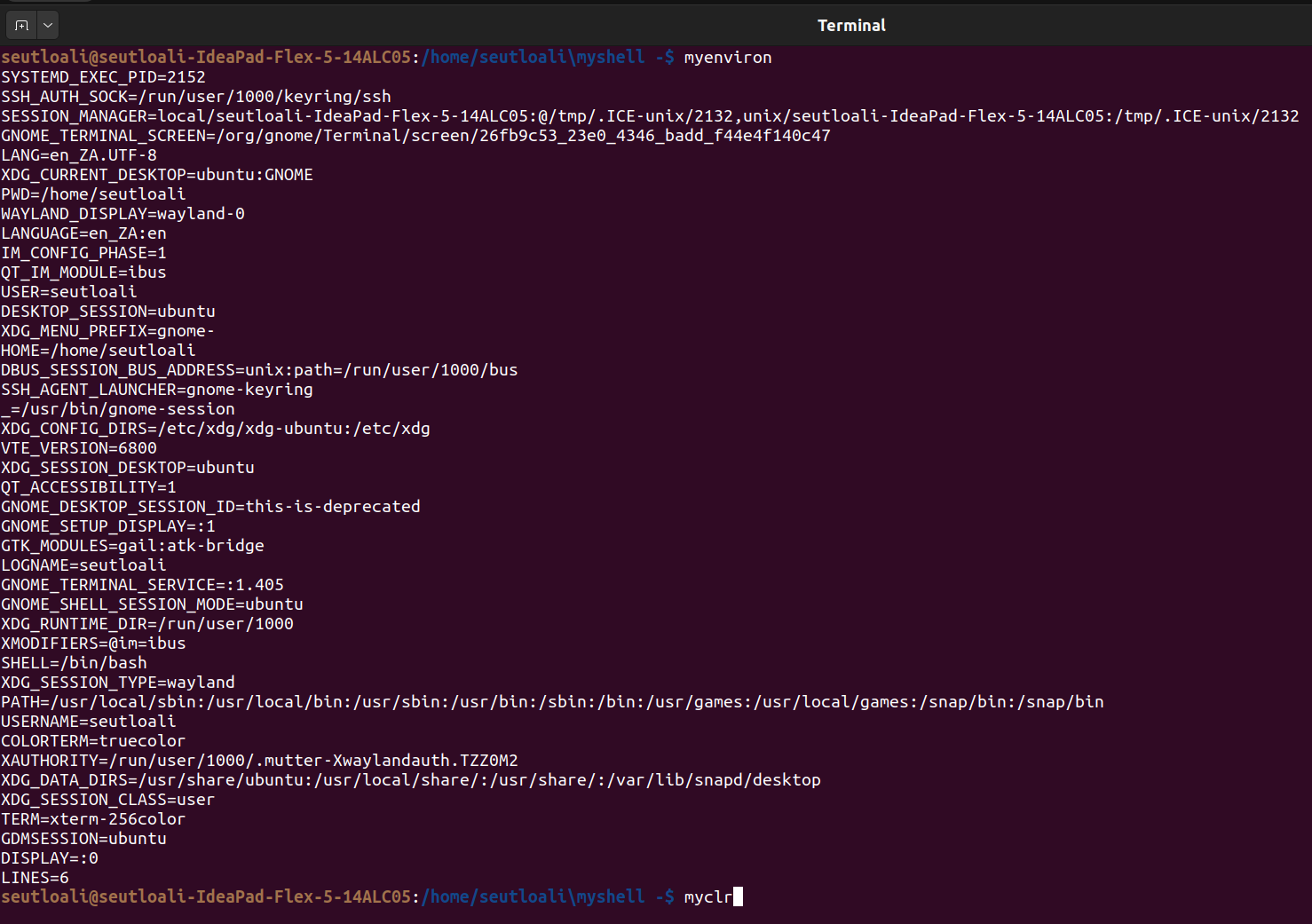




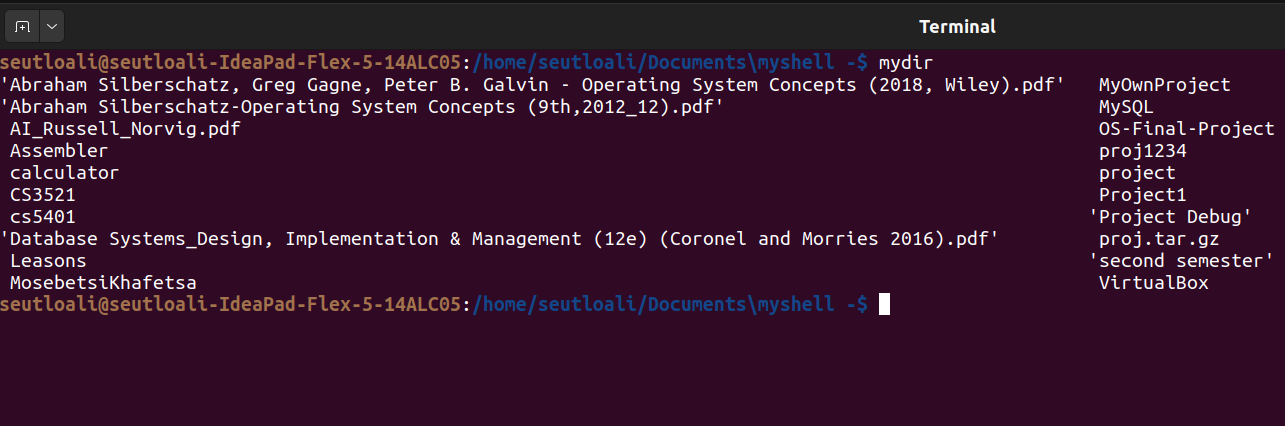
### mycd()



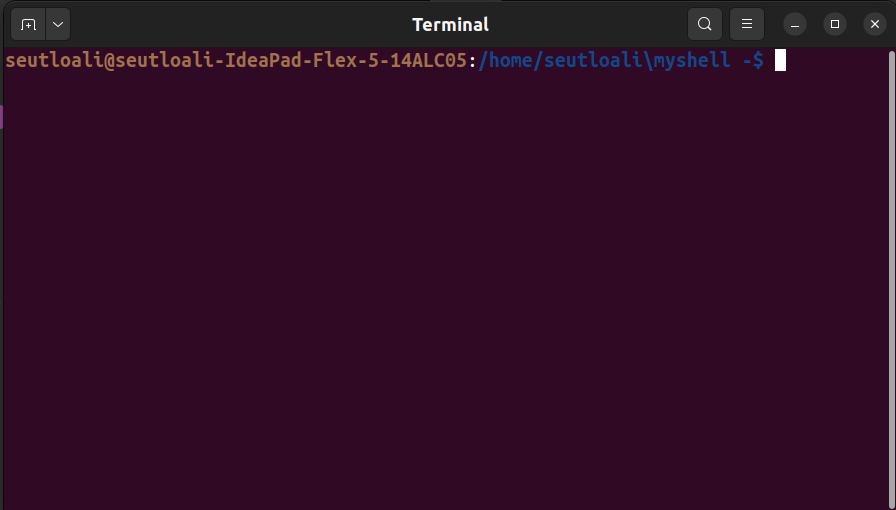
### myenviron() & myclr()



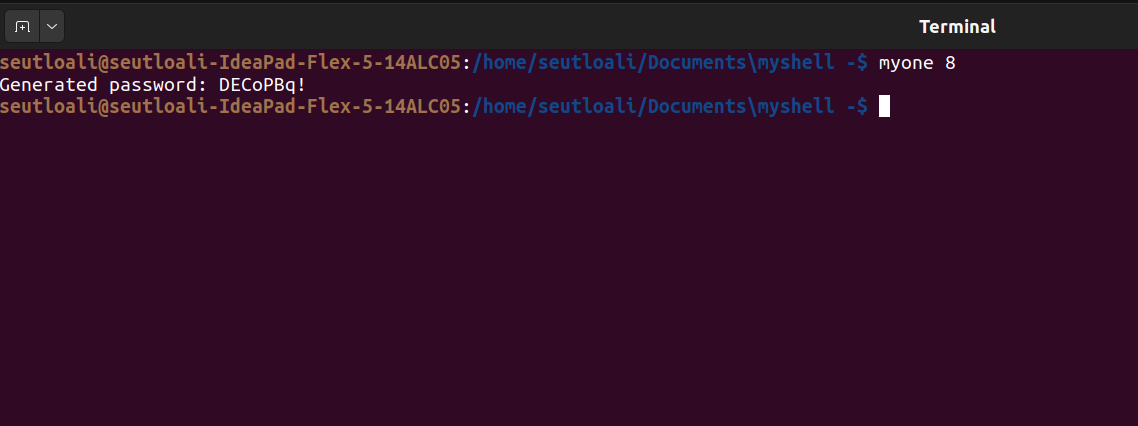
### mydir()



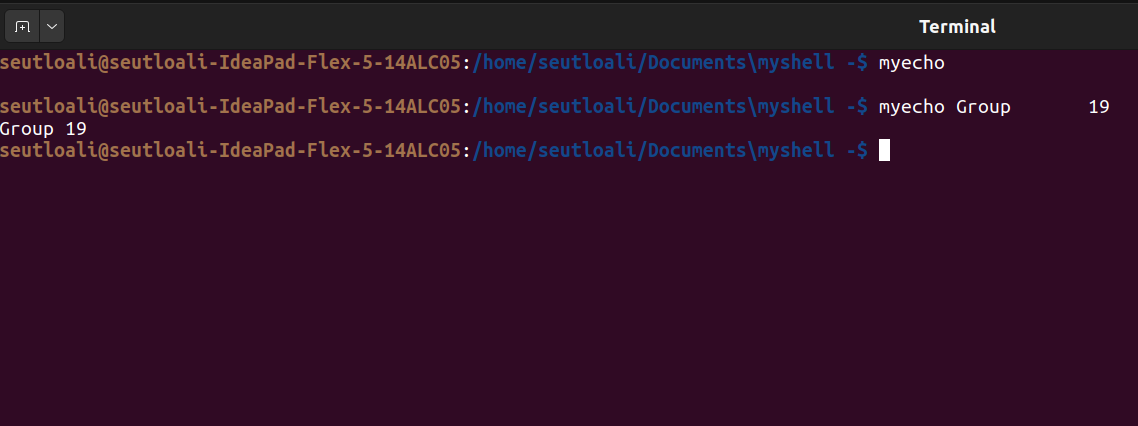
### myclr()



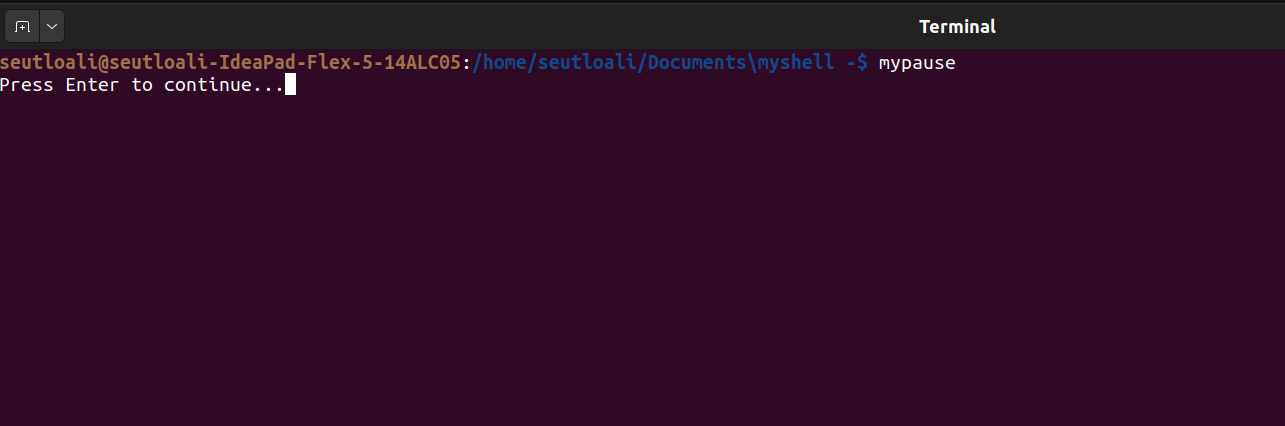
### myone()



### myecho()



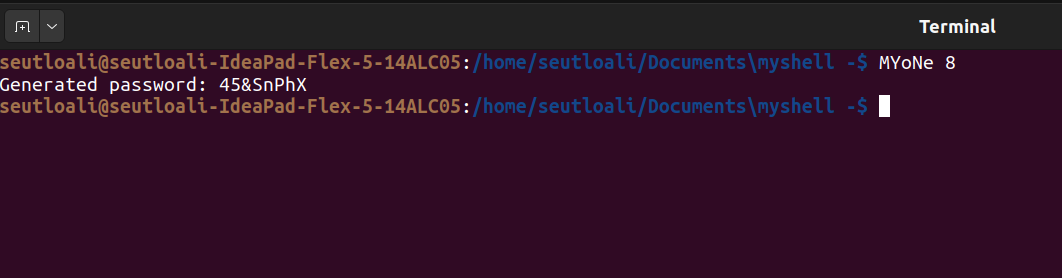
### mypause()



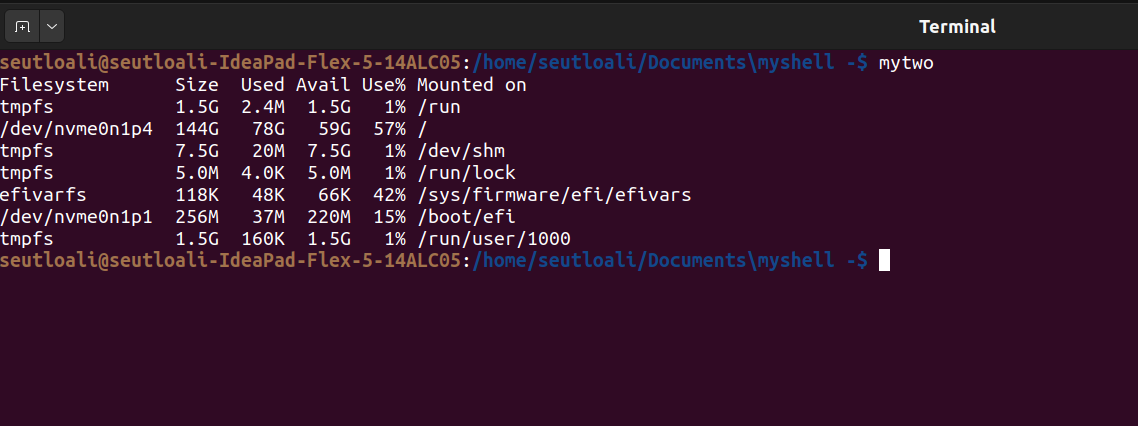
### myquit()

The shell simply closes.

### \*Bonus



### \*mytwo()



# How to Run?

