# myShell (msh) Documentation

## *Introduction*

myShell (msh) is a custom shell written in C++ that serves as an alternative to existing command line interfaces like Bash and zsh. msh supports a whole range of commands like *cd, history, exit, alias, ls, cat, grep*. It also supports environment variable manipulation(*declare*), output redirection(>, >>), piping(|), command history(*history*), and command aliasing(*alias*).

## *Structure*

The main components of the project are:

- **msh.h**: The header file for the **simple_shell** class which defines the interface for the shell.
- **msh.cpp**: The implementation file for the **simple_shell** class. It implements the logic for parsing and executing commands.
- **main.cpp**: The driver program which creates an instance of the **simple_shell** class and enters a loop to read commands and execute them.

## *Design Decisions*

The design of myShell is simple yet extremely functional and modular. It divides the user input into command tokens which are then matched to the specific task mentioned in the user input. This makes the code easily extendable to recognize and execute more commands. myShell also borrows important features like output redirection, piping, aliasing which makes it easier and faster to execute complicated tasks.

myShell implements several key concepts of operating systems:

1. Process Management

   When a command is entered, the shell creates a child process to execute the command using the **fork()** system call. The parent process, i.e., the shell itself, waits for the child process to complete execution using the **wait()** system call. This ensures that the shell does not proceed to the next command until the current one has finished executing.

2. Inter-Process Communication

   myShell uses IPC to implement the piping functionality (command1 | command2). The output of **command1** is redirected to the write end of the pipe, and the input of **command2** is redirected from the read-end of the pipe. This allows the two processes to communicate and effectively pass data from one to the other, exemplifying the IPC concept.

3. File Management

   myShell implements File Management principles through the implementation of > and >> . When a redirection operator is used, the shell opens the specified file using the **open()** system call and then redirects the standard input or output to the file using the **dup2()** system call.

## *Command Descriptions*

- **cd**: The **cd** command changes the current directory of the shell to the directory specified as an argument. If the directory does not exist or is not accessible, an error message is displayed.
- **history**: The **history** command displays the list of commands that have been entered in the current shell session. The list is stored in a **deque**, and when the maximum size is reached, the oldest command is removed.
- **clear**: The **clear** command clears the screen to remove previous commands and outputs.
- **help**: The **help** command displays a list of available commands along with a brief description of what each command does.
- **declare**: The **declare** command is used to declare a variable and assign a value to it. If a value is not provided, it lists all variables. It also supports the creation of integer variables and read-only variables.
- **quit** or **exit**: These commands are used to terminate the shell session. When this command is executed, the shell program ends.
- **export**: The **export** command is used to make a shell variable available as an environment variable to child processes. This is useful when a variable defined in the shell needs to be accessed by a program invoked from the shell.
- **unset**: The **unset** command is used to delete the value of a shell variable or environment variable.
- **alias**: The **alias** command is used to create a shortcut for a longer command sequence. The alias and the command are stored in a map for quick lookup.
- **cat**: The **cat** command in the shell is used to display the contents of a file. The **cat** command in this shell also supports output redirection using **>** and **>>**. The **>** operator overwrites the file with the output, while **>>** appends the output to the existing content of the file.
- **Redirect ( >, >>)** : This function handles the redirection of output to a file using the **>** and **>>** operators. This function is invoked when the **cat** command is used with these operators.
- **Pipe ( | )** : This function handles the piping of output from one command to input of another command. This is done using the **|** operator in the command line.

## *User Guide*

Installation:

1. Clone the git repository using `git clone`
2. Upload it to the remote server using sftp
3. Run `make` to compile and build myShell
4. Run `./msh_app` to start the shell

Once the shell starts, you can enter commands at the prompt.

To exit the shell, simply enter the `exit` or `quit` command. Use `clear` to clear the screen.