User Documentation/Manual: Mini UNIX Shell

Version: 1.0
Author: Jaskaran Singh
Date: December 9, 2025

1. Overview

MyShell is a custom UNIX-style command-line interpreter implemented in C. It serves as an interface between the user and the operating system kernel, which allows users to execute commands and manage file systems, and orchestrate processes. It is designed to mimic the core functionality of standard shells.

2. System Requirements

- **Operating System:** Linux / UNIX-based OS (macOS used in development environment)
- **Compiler:** GCC or Clang (Clang used in development environment)
- **Build Tool:** GNU Make

3. Installation and Compilation

The shell is distributed as source code and must be compiled before use. A Makefile is provided to automate this process.

Step 1: Compilation

Open your terminal, navigate to the project directory, and run:

```
make
```

- *Success:* The compiler will generate an executable file named shell.
- *Cleanup:* To remove compiled files, run make clean.

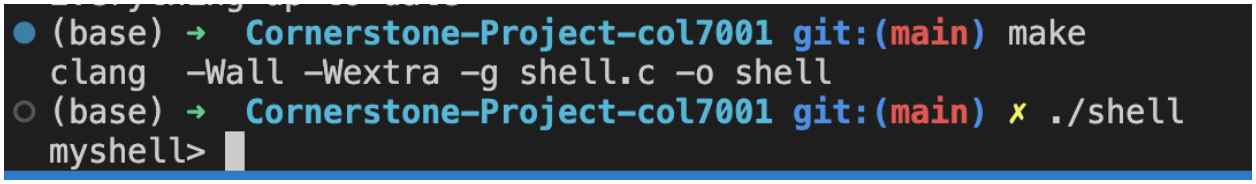
Step 2: Launching the Shell

Start the shell by running:

```
./shell
```

You will see the prompt:

myshell>



4. Feature Guide

4.1 executing Commands

You can run any standard executable found in the system's PATH.

Syntax: [command] [flags] [arguments]

- **Example:** ls -la /home

4.2 Built-in Commands

The shell includes specific commands that run internally without creating new processes.

- **cd [directory]**: Changes the current working directory.
  - *Example:* cd Documents or cd ..
  - *Error:* Returns an error if the directory does not exist.
- **exit**: Terminates the shell session.

4.3 Quoted Strings

Arguments containing spaces can be grouped using double quotes.

Syntax: command "arg with spaces"

- **Example:** echo "Hello World"
  - *Result:* Prints Hello World (without quotes).
- **Example:** mkdir "Project Alpha"
  - *Result:* Creates a single directory named Project Alpha.

4.4 I/O Redirection

You can redirect the standard input and output streams of commands using < and >.

- **Output Redirection (>)**: Writes the output of a command to a file. If the file exists, it is overwritten. If it does not exist, it is created.
  - *Syntax:* command > file.txt
  - *Example:* ls -l > filelist.txt
- **Input Redirection (<)**: Feeds the contents of a file into a command as input.
  - *Syntax:* command < file.txt
  - *Example:* wc -l < filelist.txt

4.5 Pipelines

The shell supports chaining two commands together using the pipe operator |. The output of the first command becomes the input of the second.

Syntax: command1 | command2

- **Example:** ls | grep .c
  - Lists all files ending in .c.
- **Limitation:** Currently supports single-stage pipelines (one pipe only).

4.6 Background Processing

To run a command asynchronously (in the background), append an ampersand &.

Syntax: command &

- **Behavior:** The shell will print the Process ID (PID) of the background job and immediately return the prompt myshell>, allowing you to continue typing commands while the job runs.
- **Example:** sleep 10 &

4.7 Signal Handling

- **Ctrl+C (SIGINT):** Interrupts the currently running foreground process but **does not** kill the shell itself. The shell will simply print a new line and a fresh prompt.
- **Zombie Cleanup:** Background processes are automatically cleaned up ("reaped" as mentioned in one of comments of my code) by the shell when they finish, preventing zombie processes from accumulating in the system memory.

5. Troubleshooting / FAQ

Q: I pressed Ctrl+C and the prompt didn't appear immediately.
A: The shell handles the signal and prints a new prompt. If a process was running, it has been terminated. Simply type your next command.

Q: Why does cd not work inside a pipeline?
A: In UNIX, pipelines run in subshells (child processes). Running cd inside a pipe changes the directory of that child process, which immediately exits. This is standard behavior for all UNIX shells.

Q: I see "Syntax error: expected file after >".
A: Ensure you provide a filename immediately after the redirection symbol. Do not leave it dangling at the end of a line.

Q: Can I use multiple pipes like `ls | grep a | wc -l`?
A: No. As per the current system requirements, MyShell supports **single-stage pipelines** only (e.g., `cmd1 | cmd2`). Chaining more than two commands will result in undefined behavior or the shell ignoring the subsequent commands.

Q: I ran a background job, but its output is messing up my prompt.
A: This is normal behavior. Background jobs (&) share the same terminal screen as the shell. If the background job prints text, it will appear wherever the cursor currently is. You can press `Enter` to get a clean prompt, or redirect the background job's output to a file.

Q: What happens if I leave a quote open (e.g., `echo "hello`)?
A: The shell's parser will treat the rest of the line (and potentially future input) as part of that string until it finds a closing quote. Always ensure quotes are paired correctly.

Q: Is there a limit to the length of commands I can type?
A: Yes. The shell is configured with a static buffer size of 1024 characters per command line (MAX_CMD_LEN). If you paste a command longer than this limit, the input will be truncated, which may result in a syntax error or the command executing with incomplete arguments.

Q: I tried to pass a large list of arguments, but it failed. Why?
A: The shell supports a maximum of 64 arguments per command (MAX_ARGS).This includes the command name itself and any flags. Example: ls -la file1 file2 ... file63 is the limit.
If you exceed this count (e.g., by using a wildcard * in a directory with hundreds of files), the shell will only process the first 64 items and ignore the rest.

6. Example Session

Below is a transcript of a typical session using MyShell:

```
(base) ➜  Cornerstone-Project-col7001 git:(main) ✗ ./shell
myshell> pwd
/Users/karan/Desktop/Cornerstone-Project-col7001
myshell> ls
I_O_redirection_test.txt        README.md                       shell.dSYM
Lab 1                           shell
Makefile                        shell.c
myshell> mkdir "my folder"
myshell> ls
I_O_redirection_test.txt        README.md                       shell.c
Lab 1                           my folder                       shell.dSYM
Makefile                        shell
myshell> rmdir "my folder"
myshell> ls
I_O_redirection_test.txt        README.md                       shell.dSYM
Lab 1                           shell
Makefile                        shell.c
myshell> echo "Critical System Log" > test_log.txt
myshell>
myshell> cat test_log.txt
Critical System Log
myshell> wc -w < test_log.txt
       3
myshell> ls -la | grep shell
-rwxr-xr-x   1 karan  staff  10328 Dec  9 09:00 shell
-rw-r--r--   1 karan  staff   8799 Dec  9 08:40 shell.c
drwxr-xr-x   3 karan  staff     96 Dec  9 08:40 shell.dSYM
myshell> sleep 5 &
[Started process 36101]
myshell> sleep 100 &
[Started process 36122]
myshell> ps
  PID TTY           TIME CMD
26257 ttys001    0:00.19 /bin/zsh -il
13834 ttys003    0:00.20 -zsh
26258 ttys004    0:01.61 /bin/zsh -il
35967 ttys004    0:00.01 ./shell
36122 ttys004    0:00.00 sleep 100
myshell>
myshell>
myshell>
myshell>
myshell>
```

<---------- END OF REPORT ---------->