Project Report

On

Simple Shell Development

Submitted in partial fulfillment of the requirements for the award of degree of

# BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE & ENGINEERING

# (Artificial Intelligence & Machine Learning)

by

|  |  |  |
| --- | --- | --- |
| Ms. B.BINDU  Ms. D.MANASA  Ms. K.GEETHA REDDY  Ms. R.AASHRITHA | -  -  - | 23WH1A6608  23WH1A6611  23WH1A6651  23WH1A6652 |

Under the esteemed guidance of

Ms. S Annapoorna

Assistant Professor, CSE(AI&ML)



BVRIT HYDERABAD College of Engineering for Women

(UGC Autonomous Institution | Approved by AICTE | Affiliated to JNTUH)

(NAAC Accredited - A Grade | NBA Accredited B.Tech. (EEE, ECE, CSE and IT)

Bachupally, Hyderabad – 500090

2024-25

## BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with A Grade

Bachupally, Hyderabad – 500090 2024-25

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)



## **CERTIFICATE**

This is to certify that the Project Report entitled **“Simple Shell Development”** is a bonafide work carried out by **Ms. B.BINDU (23WH1A6608), Ms. D.MANASA**

**(23WH1A6611), Ms. K.GEETHA REDDY (23WH1A6651), Ms. R.AASHRITHA (23WH1A6652)** in partial fulfillment for the award of B. Tech degree in **Computer Science & Engineering (AI&ML), BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision. The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma

|  |  |
| --- | --- |
| Supervisor  Ms.S.Annapoorna  Assistant Professor  Dept of CSE(AI&ML) | Head of the Department  Dr. B. Lakshmi Praveena  HOD & Professor  Dept of CSE(AI&ML) |

## 

## **DECLARATION**

We hereby declare that the work presented in this project entitled “**Simple Shell Development**” submitted towards completion of Project work in a Year of B.Tech of CSE(AI&ML) at BVRIT HYDERABAD College of Engineering for Women, Hyderabad is an authentic record of our original work carried out under the guidance of Ms.S.Annapoorna**,** Assistant Professor, Department of CSE(AI&ML).

**Sign with Date:**

**B.Bindu (23WH1A6608)**

**Sign with Date:**

**N. Manasa (23WH1A6611)**

**Sign with Date:**

**K.Geetha Reddy**

**(23WH1A6651)**

**Sign with Date:**

**R.Aashritha**

**(23WH1A6652)**

**ACKNOWLEDGEMENT**

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha, Principal, BVRIT HYDERABAD College of Engineering for Women**, for her support by providing the working

facilities in the college

Our sincere thanks and gratitude **to Dr. B. Lakshmi Praveena, Head of the Department,**

**Department of CSE(AI&ML), BVRIT HYDERABAD College of Engineering for Women,** for all timely support and valuable suggestions during the period of our project.

We are extremely thankful to our Internal Guide, **Ms.** S.Annapoorna**, Assistant Professor**

**CSE(AI&ML), BVRIT HYDERABAD College of Engineering for Women,** for her constant guidance and encouragement throughout the project.

Finally, we would like to thank our RealTime Project Coordinator, all Faculty and Staff of

CSE(AI&ML) department who helped us directly or indirectly. Last but not least, we wish to acknowledge our **Parents** and **Friends** for giving moral strength and constant encouragement.

B.BINDU(23wh1a6608)

D.MANASA(23wh1a6611)

K.GEETHAREDDY(23wh1a6651)

R.AASHRITHA(23wh1a6652)

## 

**AIM:**

To design and implement a lightweight, user-friendly command-line shell named SlimpleShell, capable of executing basic Unix commands, handling process creation, and demonstrating core operating system concepts such as input/output redirection, piping, and background execution.

**Procedure:**

1. Start the shell with a welcome message and enter a continuous input loop.

2. Read a line of input using fgets() or getline().

3. Remove newline character and check for exit to terminate.

4. Split the input into tokens using strtok() (parse the command and arguments).

5. Check for redirection symbols (>, <) and set flags.

6. Check for pipe symbol (|) and divide the input into separate commands.

7. For redir rection:

                         Open the required files using open().

                         Replace STDIN or STDOUT using dup2().

8. For piping

                               Create pipes using pipe().

  Use multiple fork() calls for each command.

  Pass file descriptors between processes.

  9. Use execvp() in each child process to execute the command.

 10. Use wait() in the parent process to wait for child completion.

 11. Loop back for the next command.

**Algorithm:**

 1. Start the program.

2. Display prompt (e.g., myshell$).

3. Read command input.

4. Parse command into arguments.

5. If command contains:

  >: Redirect output

  <: Redirect input

  |: Create pipe

  6. fork() a child process.

  7. In the child:

                Handle redirection/piping

                Use execvp() to execute command

  8. In the parent:

                   Use wait() to wait for the child

 9. Repeat until exit command.

10. End

# **SOURCE CODE**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <pwd.h>

#include <errno.h>

#define MAX\_CMD\_LEN 1024

#define MAX\_ARGS 100

#define MAX\_CMDS 20

#define HISTORY\_FILE ".my\_shell\_history"

int command\_count = 1;

void save\_history(const char \*cmd) {

char path[256];

snprintf(path, sizeof(path), "%s/%s", getenv("HOME"), HISTORY\_FILE);

FILE \*fp = fopen(path, "a");

if (fp) {

fprintf(fp, "%s\n", cmd);

fclose(fp);

}

}

void show\_history() {

char path[256];

snprintf(path, sizeof(path), "%s/%s", getenv("HOME"), HISTORY\_FILE);

FILE \*fp = fopen(path, "r");

if (!fp) {

perror("history");

return;

}

char line[1024];

int num = 1;

while (fgets(line, sizeof(line), fp)) {

printf("%d %s", num++, line);

}

fclose(fp);

}

void parse\_command(char \*cmd, char \*\*args) {

int i = 0;

args[i] = strtok(cmd, " \t\r\n");

while (args[i] != NULL && i < MAX\_ARGS - 1) {

args[++i] = strtok(NULL, " \t\r\n");

}

args[i] = NULL;

}

void execute\_pipeline(char \*line) {

char \*commands[MAX\_CMDS];

int num\_cmds = 0;

commands[num\_cmds] = strtok(line, "|");

while (commands[num\_cmds] != NULL && num\_cmds < MAX\_CMDS - 1) {

commands[++num\_cmds] = strtok(NULL, "|");

}

int pipefd[2 \* (num\_cmds - 1)];

for (int i = 0; i < num\_cmds - 1; ++i) {

if (pipe(pipefd + i \* 2) < 0) {

perror("pipe");

exit(EXIT\_FAILURE);

}

}

for (int i = 0; i < num\_cmds; ++i) {

char \*args[MAX\_ARGS];

parse\_command(commands[i], args);

pid\_t pid = fork();

if (pid == 0) {

if (i > 0)

dup2(pipefd[(i - 1) \* 2], STDIN\_FILENO);

if (i < num\_cmds - 1)

dup2(pipefd[i \* 2 + 1], STDOUT\_FILENO);

for (int j = 0; j < 2 \* (num\_cmds - 1); ++j)

close(pipefd[j]);

execvp(args[0], args);

perror("execvp");

exit(1);

}

}

for (int i = 0; i < 2 \* (num\_cmds - 1); ++i)

close(pipefd[i]);

for (int i = 0; i < num\_cmds; ++i)

wait(NULL);

}

void execute\_command(char \*line) {

char \*args[MAX\_ARGS];

char \*input\_file = NULL, \*output\_file = NULL;

char \*in = strchr(line, '<');

char \*out = strchr(line, '>');

if (in) {

\*in = '\0';

input\_file = strtok(in + 1, " \t\r\n");

}

if (out) {

\*out = '\0';

output\_file = strtok(out + 1, " \t\r\n");

}

parse\_command(line, args);

if (args[0] == NULL) return;

pid\_t pid = fork();

if (pid == 0) {

if (input\_file) {

int fd = open(input\_file, O\_RDONLY);

if (fd < 0) { perror("input"); exit(1); }

dup2(fd, STDIN\_FILENO);

close(fd);

}

if (output\_file) {

int fd = open(output\_file, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd < 0) { perror("output"); exit(1); }

dup2(fd, STDOUT\_FILENO);

close(fd);

}

execvp(args[0], args);

perror("execvp");

exit(1);

} else {

wait(NULL);

}

}

int run\_command(char \*line) {

int status = 0;

// Handle && and ||

if (strstr(line, "&&") || strstr(line, "||")) {

char \*op = strstr(line, "&&") ? "&&" : "||";

char \*cmd1 = strtok(line, op);

char \*cmd2 = strtok(NULL, "");

if (!cmd1 || !cmd2) return 1;

status = run\_command(cmd1);

if ((strstr(op, "&&") && status == 0) || (strstr(op, "||") && status != 0)) {

return run\_command(cmd2);

}

return status;

}

// Background execution

int bg = 0;

int len = strlen(line);

if (len > 0 && line[len - 1] == '&') {

bg = 1;

line[len - 1] = '\0';

}

// Built-in cd

if (strncmp(line, "cd ", 3) == 0) {

char \*path = line + 3;

if (chdir(path) != 0) {

perror("cd failed");

return 1;

}

return 0;

}

// Built-in history

if (strcmp(line, "history") == 0) {

show\_history();

return 0;

}

if (strchr(line, '|')) {

execute\_pipeline(line);

} else {

execute\_command(line);

}

return bg ? 0 : 0;

}

int main() {

char line[MAX\_CMD\_LEN];

while (1) {

// Get current directory and user

char cwd[256];

getcwd(cwd, sizeof(cwd));

struct passwd \*pw = getpwuid(getuid());

char \*user = pw->pw\_name;

// Display prompt with count

printf("💻 [%d] %s:%s$ ", command\_count++, user, cwd);

fflush(stdout);

if (!fgets(line, sizeof(line), stdin)) break;

line[strcspn(line, "\n")] = 0;

if (strlen(line) == 0) continue;

if (strcmp(line, "exit") == 0) break;

save\_history(line);

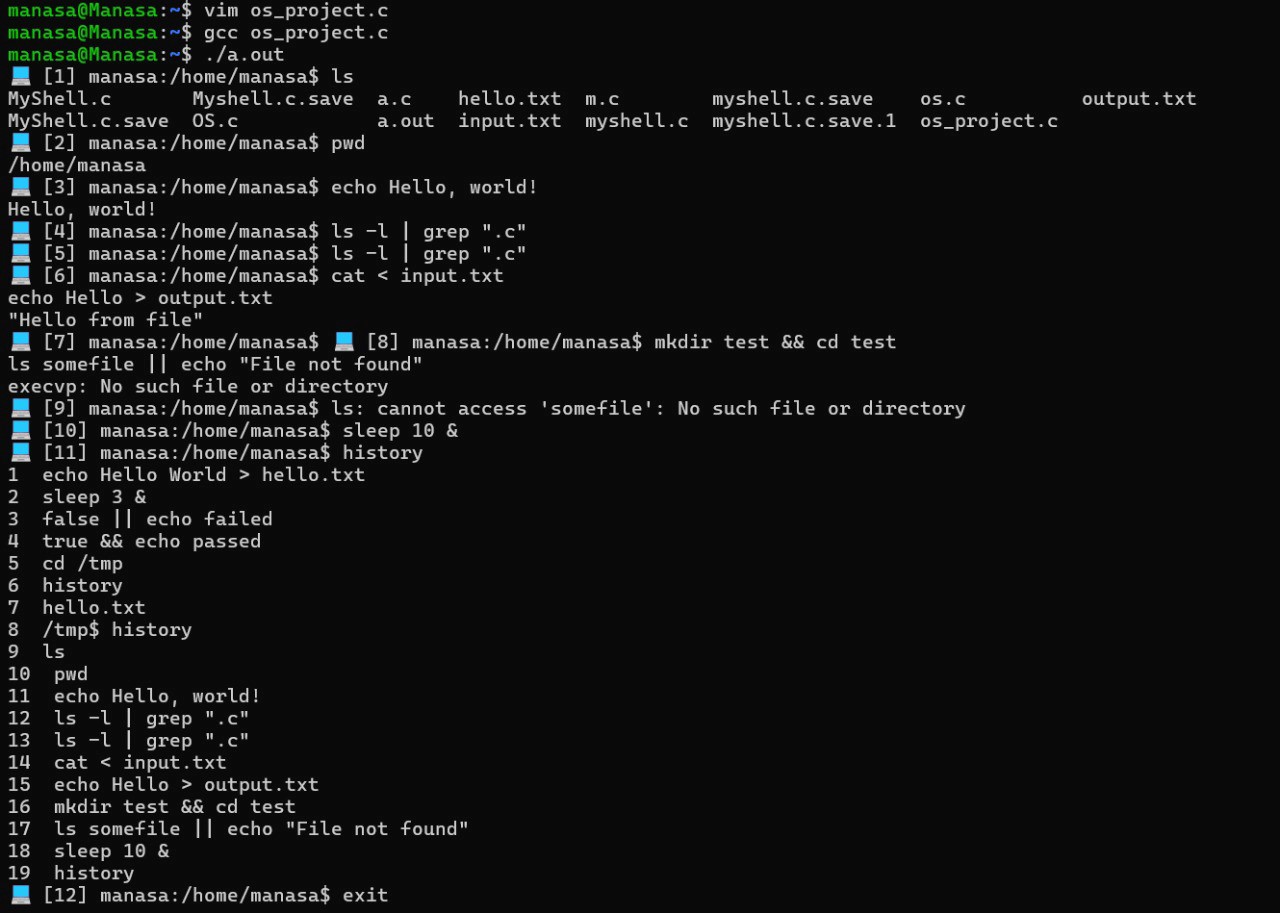
run\_command(line);

}

return 0;

}

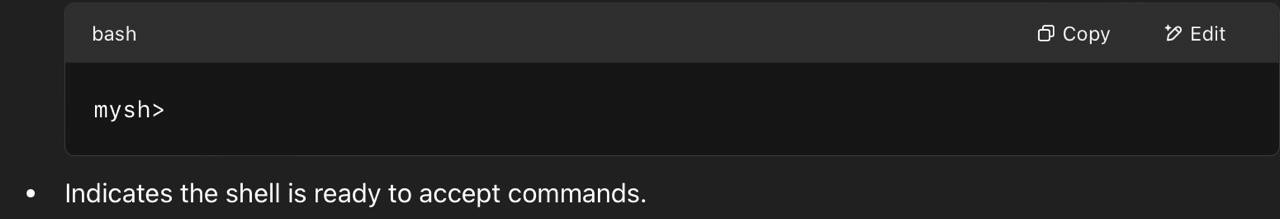
**OUTPUT:**



**FUNCTIONALITIES:**

1. **Prompt Display**

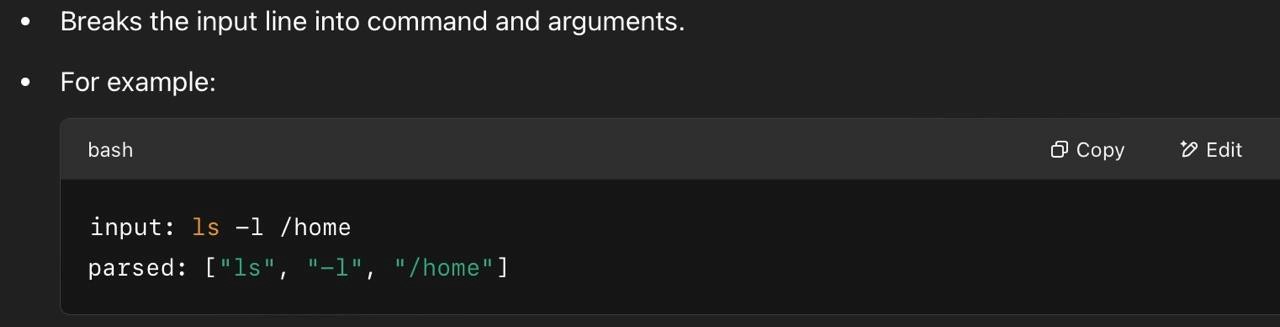
Shows a custom prompt like:



1. **Reading User Input**

Accepts a line of text input from the user (e.g., ls -l, cd Documents, etc.). Uses fgets() or getline() to read input.

1. **Command Parsing**



1. **Executing External Commands**

Executes standard system commands like ls, pwd, echo, etc.

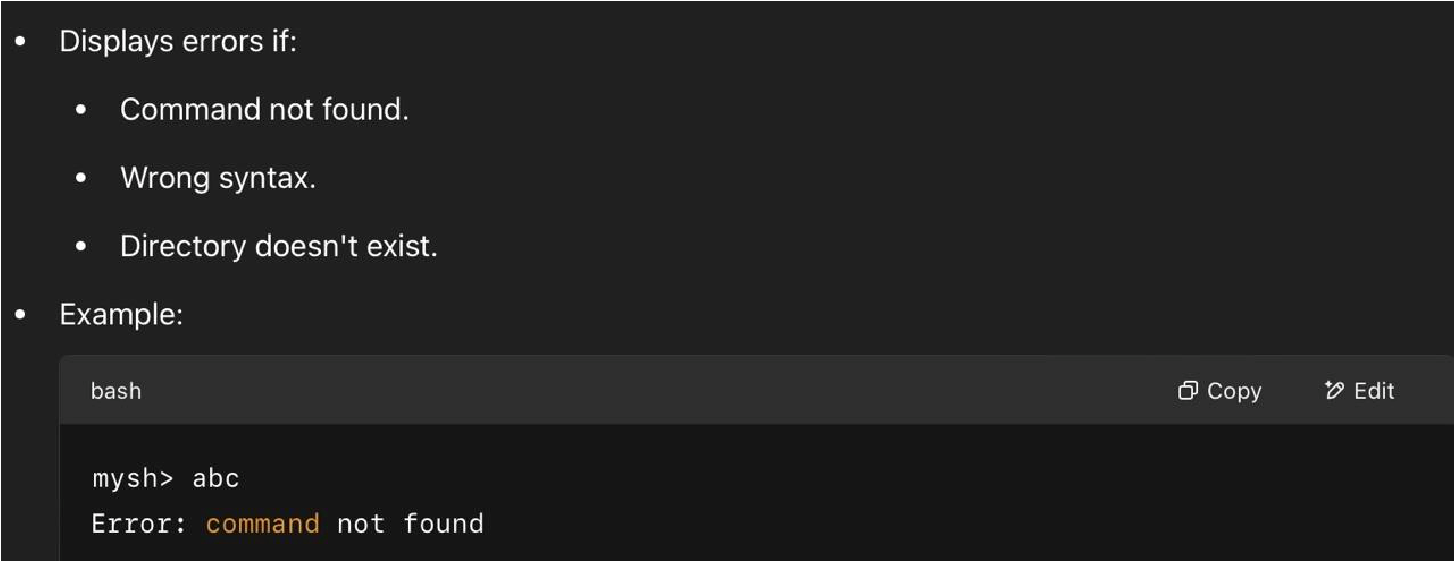
uses:

fork() – to create a child process.

execvp() – to run the command.

wait() or waitpid() – parent waits for the child to finish.

**5.Error Handling**



**ADVANTAGES:**

1. 🧠 Deep Understanding of Operating Systems

Covers system-level concepts like: Process creation (fork())

Command execution (execvp())

Process synchronization (wait())

1. 💻 Hands-On with System Calls

You gain real experience using important C system calls. Helps understand how user space talks to the kernel.

1. 🛠 Build Your Own CLI Tool

You create a working command-line interface from scratch. It’s a mini version of something you use daily — Bash!

1. 🔄Reinforces Core C Programming Skills

String handling

Pointers and arrays

Dynamic memory

Error handling

Looping and control structures

1. 🧩Extendable for More Features

You can easily build on top of it by adding:

Piping

I/O redirection

Background jobs

Environment variables Autocomplete, etc.

**7.** 🧪Helps with Debugging and Testing Practice

Debugging logic bugs

Testing with various shell commands

**DISADVANTAGES:**

1. 🔒 Lack of Security Features

* Simple shells don’t include advanced permission checks, input sanitization, or protection against malicious commands.
* Vulnerable to misuse if not carefully handled.

1. 🧠 Limited Functionality Compared to Real Shells
   * Command history
   * Tab auto-completion
   * Job control (fg, bg, kill)
   * Environment variable expansion ($HOME, $PATH) Not practical for real-world use.

🐞Error Handling Is Basic

Basic error messages (like "Command not found") don’t provide detailed debugging info.

May not gracefully recover from incorrect inputs or crashes.

1. 🕰 Can Be Time-Consuming to Extend

Adding features like pipes, redirection, or background execution requires deep understanding and careful handling. Project complexity grows fast.

**CONCLUSION:**

This project helped in understanding how a basic shell works by implementing core features like command execution, process handling, and built-in commands. It was a great hands-on way to learn system programming and the basics of how real shells operate.