Custom Shell Implementation

Capstone Project - Linux OS

Submitted by: Nikhil Upadhyay

Registration no.: 2241021004

Wipro Batch: 10

Email: nikhil.up2004@gmail.com

Date: 09/11/2025

Assignment 2 (Linux OS)

Custom Shell Implementation - Project Report

Objective:

Build a simple shell in C++ that can execute commands, manage processes, and handle redirection, piping and basic job control (background execution).

Day-wise Tasks:

- Day 1: Plan the shell features and parse user input.
- Day 2: Implement execution of basic commands through the shell.
- Day 3: Add support for process management (foreground, background processes).
- Day 4: Implement piping and redirection features.
- Day 5: Incorporate job control (listing jobs, bringing jobs to foreground/background).

Implementation Details:

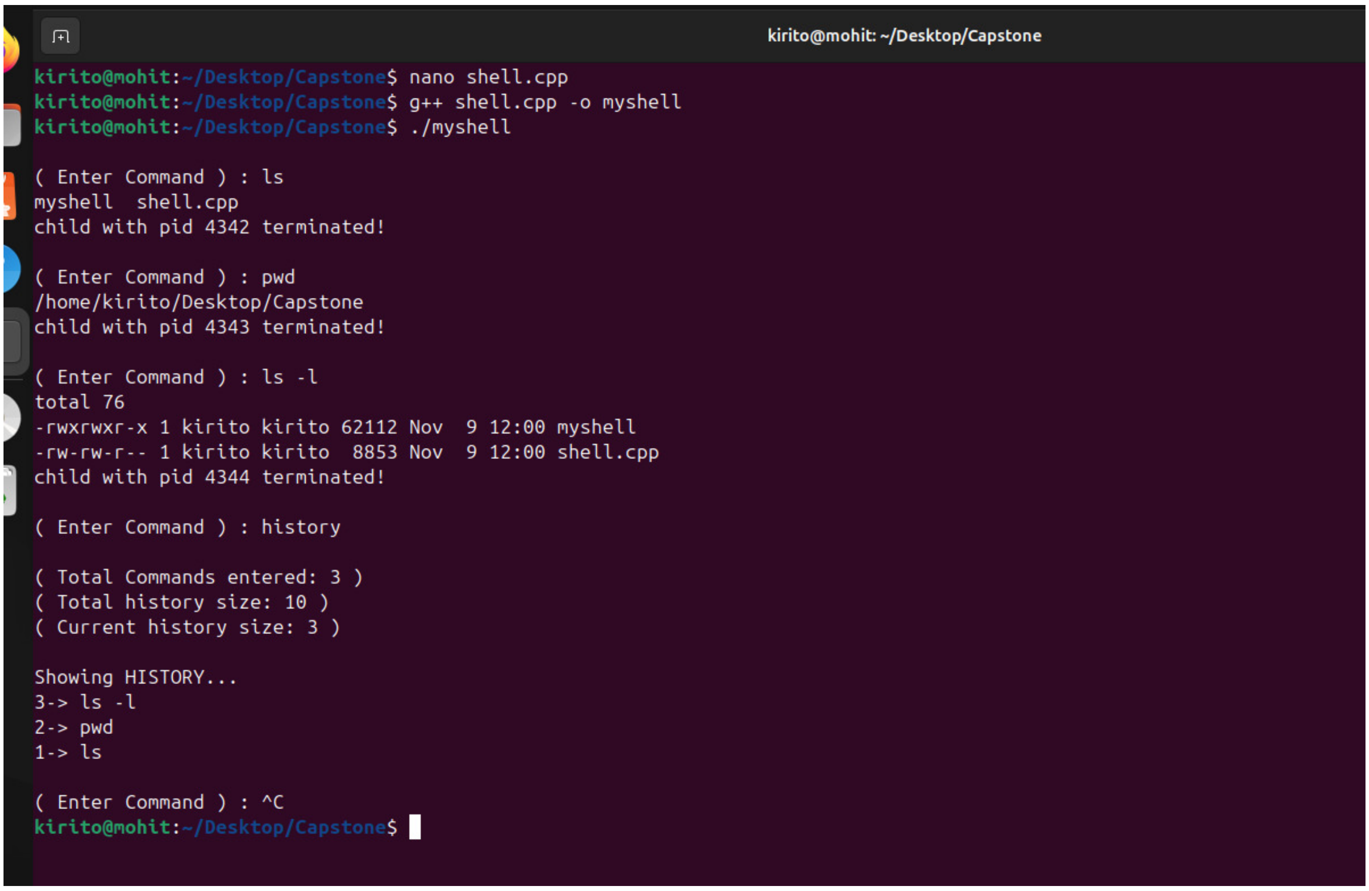
This shell implementation supports:
- Parsing commands and tokenizing by spaces while handling '&' for background execution.
- Builtin commands: cd, mkfifo, history, !! and !N support.
- Redirection: input '<', output '>' and stderr redirection '2>'.
- Piping between multiple commands using '|'.
- Background execution using '&' at end of command.

Limitations & Notes:
- Quoting handling is minimal (single quotes are ignored in tokenization).
- Advanced job control (fg/bg builtins, job listing) is basic; processes launched in background are not tracked beyond reporting PID.

C++ Code (complete):

```
#include <iostream>
#include <string>
#include <vector>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/stat.h>

using namespace std;

// file_descriptor: 0-> read, 1-> write, 2-> error

vector<string> tokenizeCommand(const string& str, char delim) {
    vector<string> strTokens;
    string tmp = "";

    for (size_t i = 0; i < str.length(); i++) {
        if (str[i] == delim) {
            if (!tmp.empty()) {
                strTokens.push_back(tmp);
                tmp.clear();
            }
        } else if (str[i] == '&') {
            if (!tmp.empty()) {
                strTokens.push_back(tmp);
            }
            strTokens.push_back("bgRunTriggered");
            return strTokens;
        } else if (str[i] == '\'') {
            // ignore single quotes (could be extended)
        } else {
            tmp.push_back(str[i]);
        }
    }
    if (!tmp.empty()) {
        strTokens.push_back(tmp);
    }
    return strTokens;
}

vector<string> separateCommands(const string& str, char delim) {
    vector<string> strTokens;
    string tmp = "";

    for (size_t i = 0; i < str.length(); i++) {
        if (str[i] == delim) {
            if (!tmp.empty()) {
                strTokens.push_back(tmp);
                tmp.clear();
            }
        } else {
            tmp.push_back(str[i]);
        }
    }
    if (!tmp.empty()) {
        strTokens.push_back(tmp);
    }
    return strTokens;
}

void shiftAndPop(vector<string>& vec) {
    if (vec.empty()) return;
    for (size_t i = 0; i + 1 < vec.size(); ++i) {
        vec[i] = vec[i + 1];
    }
    vec.pop_back();
}

int main() {
    string cmd;
    pid_t ret_Val;
    int childCount;
    vector<string> history;
    int historyCommandsCounter = 0;

    while (true) {
        cout << "\n( Enter Command ) : ";
        if (!getline(cin, cmd)) {
            // handle Ctrl-D or EOF
            break;
        }
        if (cmd == "exit") {
            break;
        }

        if (cmd == "history") {
            int srNo = history.size();
            cout << "\n( Total Commands entered: " << historyCommandsCounter << " )\n";
            cout << "( Total history size: 10 )\n";
            cout << "( Current history size: " << history.size() << " )\n";
            cout << "\nShowing HISTORY...\n";
            for (int i = history.size() - 1; i >= 0; i--) {
                cout << srNo-- << "-> " << history[i] << endl;
            }
            continue;
        }

        if (cmd == "!!") {
            if (history.empty()) {
                cout << "history is empty\n";
                continue;
            }
            cmd = history.back();
            cout << "most recent command was " << cmd << endl << endl;
        }

        if (cmd.size() > 1 && cmd[0] == '!' && cmd[1] != '!') {
            string numStr = cmd.substr(1);
            int no = 0;
            try {
                no = stoi(numStr);
            } catch (...) {
                cout << "Invalid history reference\n";
                continue;
            }
            if (no > (int)history.size() || no <= 0) {
                cout << " command - " << no << " not found in history! (//ERROR_404)\n";
                continue;
            }
            cout << "no " << no << " command will be going to execute...\n\n";
            cmd = history[no - 1];
        }

        historyCommandsCounter++;
        if ((int)history.size() == 10) {
            shiftAndPop(history);
        }
        history.push_back(cmd);

        vector<string> commands = separateCommands(cmd, '|');
        bool pipesFlg = (commands.size() > 1);

        int prev_pipe[2] = {-1, -1}, next_pipe[2];

        for (size_t i = 0; i < commands.size(); i++) {
            vector<string> tokens = tokenizeCommand(commands[i], ' ');
            bool bgRunFlag = false;
            if (!tokens.empty() && tokens.back() == "bgRunTriggered") {
                bgRunFlag = true;
                tokens.pop_back();
            }
            if (tokens.empty()) continue;

            if (tokens[0] == "cd") {
                if (tokens.size() < 2) {
                    cout << "Usage: cd <directory>" << endl;
                } else if (chdir(tokens[1].c_str()) != 0) {
                    perror("chdir failed");
                }
                continue;
            }

            if (tokens[0] == "mkfifo") {
                if (tokens.size() < 2) {
                    cout << "Usage: mkfifo <filename>" << endl;
                } else if (mkfifo(tokens[1].c_str(), 0666) != 0) {
                    perror("mkfifo failed");
                } else {
                    cout << "named pipe : " << tokens[1] << " created successfully!\n";
                }
                continue;
            }

            if (i < commands.size() - 1 && pipesFlg) {
                if (pipe(next_pipe) == -1) {
                    perror("pipe creation failed");
                    exit(EXIT_FAILURE);
                }
            }

            ret_Val = fork();
            if (ret_Val < 0) {
                cerr << "\nFork() failed!\n";
                exit(EXIT_FAILURE);
            } else if (ret_Val == 0) {
                // CHILD
                // handle redirections
                for (size_t j = 0; j < tokens.size(); j++) {
                    if (tokens[j] == "<" && j + 1 < tokens.size()) {
                        int fd_read = open(tokens[j + 1].c_str(), O_RDONLY);
                        if (fd_read < 0) { perror("open for < failed"); exit(EXIT_FAILURE); }
                        dup2(fd_read, STDIN_FILENO);
                        close(fd_read);
                        tokens.erase(tokens.begin() + j, tokens.begin() + j + 2);
                        break;
                    }
                }
                for (size_t j = 0; j < tokens.size(); j++) {
                    if (tokens[j] == ">" && j + 1 < tokens.size()) {
                        int fd_write = open(tokens[j + 1].c_str(), O_WRONLY | O_CREAT |
O_TRUNC, 0644);
                        if (fd_write < 0) { perror("open for > failed"); exit(EXIT_FAILURE); }
                        dup2(fd_write, STDOUT_FILENO);
                        close(fd_write);
                        tokens.erase(tokens.begin() + j, tokens.begin() + j + 2);
                        break;
                    }
                }
                for (size_t j = 0; j < tokens.size(); j++) {
                    if (tokens[j] == "2>" && j + 1 < tokens.size()) {
                        int fd_error = open(tokens[j + 1].c_str(), O_WRONLY | O_CREAT |
O_APPEND, 0644);
                        if (fd_error < 0) { perror("open for 2> failed"); exit(EXIT_FAILURE); }
                        dup2(fd_error, STDERR_FILENO);
                        close(fd_error);
                        tokens.erase(tokens.begin() + j, tokens.begin() + j + 2);
                        break;
                    }
                }

                if (pipesFlg) {
                    if (i < commands.size() - 1) {
                        dup2(next_pipe[1], STDOUT_FILENO);
                    }
                    if (i > 0) {
                        dup2(prev_pipe[0], STDIN_FILENO);
                    }
                    // close unused fds
                    if (prev_pipe[0] != -1) close(prev_pipe[0]);
                    if (prev_pipe[1] != -1) close(prev_pipe[1]);
                    close(next_pipe[0]);
                    close(next_pipe[1]);
                }
                // build exec args
                vector<char*> args;
                for (size_t j = 0; j < tokens.size(); j++) {
                    args.push_back(const_cast<char*>(tokens[j].c_str()));
                }
                args.push_back(NULL);

                if (execvp(args[0], args.data()) == -1) {
                    perror("execvp failed");
                    exit(EXIT_FAILURE);
                }
            } else {
                // PARENT
                if (pipesFlg) {
                    if (i < commands.size() - 1) {
                        close(next_pipe[1]);
                    }
                    if (i > 0) {
                        close(prev_pipe[0]);
                    }
                    prev_pipe[0] = next_pipe[0];
                    prev_pipe[1] = next_pipe[1];
                }
                if (!bgRunFlag) {
                    int status;
                    waitpid(ret_Val, &status, 0);
                    if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
                        cout << "child with pid " << ret_Val << " terminated!\n";
                    }
                } else {
                    cout << "\nchild running in background! pid = " << ret_Val << "\n";
                }
            }
        }
    }

    cout << "\n***** shell terminated! *****\n";
    return 0;
}
```

Screenshot 1: Shell execution output



```
kirito@mohit:~/Desktop/Capstone$ nano shell.cpp
kirito@mohit:~/Desktop/Capstone$ g++ shell.cpp -o myshell
kirito@mohit:~/Desktop/Capstone$ ./myshell

( Enter Command ) : ls
myshell  shell.cpp
child with pid 4342 terminated!

( Enter Command ) : pwd
/home/kirito/Desktop/Capstone
child with pid 4343 terminated!

( Enter Command ) : ls -l
total 76
-rwxrwxr-x 1 kirito kirito 62112 Nov  9 12:00 myshell
-rw-rw-r-- 1 kirito kirito  8853 Nov  9 12:00 shell.cpp
child with pid 4344 terminated!

( Enter Command ) : history

( Total Commands entered: 3 )
( Total history size: 10 )
( Current history size: 3 )

Showing HISTORY...
3-> ls -l
2-> pwd
1-> ls

( Enter Command ) : ^C
kirito@mohit:~/Desktop/Capstone$ 
```