A

Project Report

On

# Custom Shell

Submitted in partial fulfillment of the requirement for the degree of

**Bachelor of Technology**

**In**

**Computer Science and Engineering**

By


**Deepak Birkhani 2261165**

**Dinesh Joshi 2261189**

**Esha Mahara 2261204**

**Harshita Mehta 2261259**


**Under the Guidance of**

**Mr. Prince Kumar**

**ASSISTANT PROFESSOR**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS**

**SATTAL ROAD, P.O. BHOWALI,**

**DISTRICT- NAINITAL-263132**


**2024-2025**

# STUDENT'S DECLARATION

We, **Deepak Birkhani, Dinesh Joshi, Esha Mahara** and **Harshita Mehta** hereby declare the work, which is being presented in the project, entitled **Custom Shell** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of **Mr. Prince Kumar**. The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Deepak Birkhani

Dinesh Joshi

Esha Mahara

Harshita Mehta

# CERTIFICATE

**The project report entitled "Custom Shell" being submitted by Deepak Birkhani S/o Puran Chandra Birkhani, 2261165, Dinesh Joshi S/o Bhuwan Chandra Joshi, 2261189, Esha Mahara D/o Harish Singh Mahara, 2261204, Harshita Mehta D/o Gopal Mehta, 2261259 of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.**

**Mr. Prince Kumar**
**(Project Guide)**

**Dr. Ankur Singh Bisht**
**(Head, CSE)**

# ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **'Mr. Prince Kumar'** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

**Deepak Birkhani, 2261165**
**Dinesh Joshi, 2261189**
**Esha Mahara, 2261204**
**Harshita Mehta, 2261259**

# Abstract

The CustomShell project is a command-line based shell environment developed entirely in C++ for the Windows operating system. It aims to provide a simplified yet functional alternative to traditional terminal programs by replicating key features such as command parsing, process management, I/O redirection, built-in command execution, and command auto-completion.

The motivation behind this project lies in understanding the core concepts of operating systems such as system calls, process handling, piping, and memory management. By developing each module of the shell from scratch, the project emphasizes low-level system programming and hands-on implementation of functionalities typically abstracted away by high-level terminal applications.

CustomShell follows a modular design and is divided into multiple phases, including support for internal and external commands, background and foreground process handling (&, jobs, fg, bg), redirection operators (>, <), and basic command history with keyboard navigation. The shell also implements autocomplete logic using Windows API for better user interaction in the CLI environment.

Despite the absence of a graphical user interface (GUI), the shell provides a user-friendly experience through its efficient command-line prompt and responsive features. It is primarily intended for users and developers who wish to learn about operating system internals and custom shell development on Windows platforms.

The CustomShell project not only serves as a learning tool but also lays the foundation for more advanced features such as persistent history, scripting support, environment variable handling, and customizable configurations in future versions.

## TABLE OF CONTENTS

# INTRODUCTION

## 1.1 Prologue

In modern computing, the command-line interface (CLI) remains a foundational tool for interacting with an operating system. Despite the growing dominance of graphical user interfaces (GUIs), command-line environments are still widely used in programming, server management, automation, and debugging due to their speed, flexibility, and power.

This project, titled CustomShell, aims to replicate and extend the functionality of a traditional command-line shell by developing a fully functional shell application from scratch for the Windows platform using C++. The project encapsulates core operating system concepts such as process creation, command execution, inter-process communication, I/O redirection, piping, and it culminates in the development of a GUI using the Win32 API.

CustomShell is not just a utility; it is an educational tool that bridges the gap between theoretical operating system design and hands-on practical implementation. This project was undertaken as part of the academic curriculum with the objective of enhancing understanding and mastery of low-level system programming in a real-world scenario.

## 1.2 Background and Motivations

The motivation behind this project stems from the growing need for system-level understanding among software developers. While many students and professionals are familiar with using shells like CMD, PowerShell, or Bash, few understand how these tools function internally.

Modern shells are complex, and their implementation details are hidden. This opacity leaves a gap in knowledge that can only be filled through building such a system from the ground up. By designing CustomShell, we aim to:

- Gain an in-depth understanding of how shells parse and execute user commands.
- Learn how operating systems handle process creation, background execution, and input/output management.
- Explore system calls and how user-level programs interact with the OS kernel.
- Enhance debugging and low-level development skills.

Moreover, the project serves as a strong foundation for students interested in systems programming, cybersecurity, and OS development.

### 1.3 Problem Statement

While most modern shells are well-documented and widely used, they are often treated as black boxes in educational settings. Students use them but rarely understand how they work. This creates a significant disconnect between theoretical knowledge of operating system principles and the practical skills required to implement or modify such systems.

The core problem addressed by this project is:

**"How can we build a fully functional custom shell on Windows using C++ that supports system commands, custom built-ins, process control, I/O redirection, and GUI integration, while reinforcing core OS principles?"**

The solution must be robust, modular, and extensible, allowing for future enhancements like environment configuration and user-defined command scripts.

### 1.4 Objectives and Research Methodology

Objectives:

The primary objectives of the CustomShell project are as follows:

- To implement a functional shell from scratch using C++ on the Windows platform.
- To support:
    - Execution of system commands via CreateProcess().
    - Built-in commands like mycd, mypwd, myecho, and myexit.
    - Process control features like background execution, job listing, and foreground/background switching.
    - I/O redirection (>, <) and command piping (|).
    - Command history with navigation via arrow keys.
    - Tab-based autocomplete functionality for file and folder names.
- To reinforce the academic understanding of system programming and OS concepts through hands-on experience.

Research Methodology

The project follows a structured and modular research and development approach:

- **Literature Survey**: Study of existing shell architectures (Bash, CMD, PowerShell) to understand their functionalities and limitations.
- **Tool and API Analysis**: Researching Windows system calls like CreateProcess(),

WaitForSingleObject().

- **Module-Based Development**: Each functionality (e.g., built-in commands, piping, ) is treated as a separate module and developed/tested independently.
- **Iterative Refinement**: Constant testing, debugging, and enhancement of features during each development phase.
- **Version Control**: Git was used to track code changes and manage features in separate branches.

This approach ensures clarity, better debugging, scalability, and maintainability throughout the development lifecycle.

### 1.5 Project Organization

The CustomShell project is organized into seven development phases, each corresponding to a specific set of features. The project adopts a modular architecture, where each component is implemented in a separate file for better clarity and maintainability.

**Phases of Development:**

1. **Basic Shell Structure**
    - Reads user input, tokenizes it, and uses CreateProcess() to execute commands.
    - Handles command parsing and error feedback.
2. **Built-in Custom Commands**
    - Implements internal shell commands like mycd, mypwd, myecho, and myexit.
    - These commands are processed without invoking external executables.
3. **Process Management**
    - Adds support for background execution using &.
    - Introduces commands like jobs, fg, and bg.
    - Handles signals like Ctrl+C using Windows-specific APIs.
4. **I/O Redirection and Piping**
    - Implements input (<), output (>), and append (>>) redirection.
    - Adds support for pipelines using the | operator between commands.
5. **Command History**

- Stores previously entered commands in memory.
- Supports navigation using arrow keys.

6. **Auto-Completion & Prompt Customization**
    - Enables tab-completion for filenames and directories.
    - Allows users to customize the shell prompt.
7. **Configuration and Customization**
    - Loads shell settings from a .myshellrc file.
    - Supports environment variables like $PATH and $USER.

Each phase is developed incrementally and integrated into the core shell loop, maintaining backward compatibility and extensibility for future enhancements.

# HARDWARE AND SOFTWARE REQUIREMENTS

## 2.1 Hardware Requirement

The development and testing of the CustomShell project required a system with the following minimum hardware specifications to ensure smooth compilation, execution, and GUI rendering using the Windows API:

| Component | Specification (Minimum) |
|---|---|
| Processor | Intel Core i3 6th Gen or equivalent (x64) |
| RAM | 4 GB |
| Storage | 500 MB of free disk space |
| Display | 1024 x 768 resolution or higher |
| Input Devices | Keyboard and Mouse |
| Architecture | 64-bit (x64) processor architecture |

## 2.2 Software Requirement

The software tools and platforms used for the development of the **CustomShell** project are as follows:

- **Operating System**: Windows 10 or later (64-bit)
- **Programming Language**: C++
- **Compiler**: MinGW or Microsoft Visual C++ Compiler
- **Build System**: Visual Studio IDE
- **Editor/IDE**: Visual Studio Code / Visual Studio 2022
- **Terminal/Console**: Windows Command Prompt (for testing CLI)
- **GUI Library**: Win32 API (for GUI Integration phase)
- **Version Control**: Git

These tools enabled efficient development, debugging, version tracking, and GUI rendering support for the CustomShell project.

## CODE:

```
vector<string> tokenize(const string& input) {
    vector<string> tokens;
    string current;
    bool inQuotes = false;

    for (size_t i = 0; i < input.length(); ++i) {
        char c = input[i];

        if (c == '"') {
            inQuotes = !inQuotes;
        } else if (isspace(c) && !inQuotes) {
            if (!current.empty()) {
                tokens.push_back(current);
                current.clear();
            }
        } else {
            current += c;
        }
    }

    if (!current.empty()) {
        tokens.push_back(current);
    }

    return tokens;
}
```

```
void parseRedirection(string& command, string& inputFile, string& outputFile, bool& appendMode) {
    istringstream iss(command);
    string token;
    string cleanedCommand;

    while (iss >> token) {
        if (token == "<") {
            iss >> inputFile;
        } else if (token == ">>") {
            iss >> outputFile;
            appendMode = true;
        } else if (token == ">") {
            iss >> outputFile;
            appendMode = false;
        } else {
            cleanedCommand += token + " ";
        }
    }

    command = trim(cleanedCommand);
}
```

```
void executeCommand(const string& commandLine){
    string line = trim(commandLine);
    if (line.empty()) return;

    string expandedLine = expandEnvironmentVariables(line);
    if (expandedLine.find('<') != string::npos || expandedLine.find('>') != string::npos) {
        executeWithRedirection(expandedLine);
        return;
    }

    vector<string> args = tokenize(expandedLine);
    if (args.empty()) return;

    string command = args[0];

    try{
        if (command == "mycd") {
            if (args.size() > 1) mycd(args[1]);
            else cerr << "Usage: mycd <path>\n";
        }
        else if(command == "mypwd"){
            mypwd();
        }
        else if(command=="myecho"){
            if (args.size() > 1) {
                string text = trim(line.substr(line.find(" ") + 1));
                cout << shellConfig.expandVariables(text) << "\n";
```

```
try{
    else if (command == "mymkdir") {
        if (args.size() > 1) mymkdir(args[1]);
        else cerr << "Usage: mymkdir <dirname>\n";
    }
    else if (command == "myrmdir") {
        if (args.size() > 1) myrmdir(args[1]);
        else cerr << "Usage: myrmdir <dirname>\n";
    }
    else if (command == "mycp") {
        if (args.size() > 2) mycp(args[1], args[2]);
        else cerr << "Usage: mycp <source> <destination>\n";
    }
    else if (command == "mymv") {
        if (args.size() > 2) mymv(args[1], args[2]);
        else cerr << "Usage: mymv <source> <destination>\n";
    }
    else if (command == "mytouch") {
        if (args.size() > 1) mytouch(args[1]);
        else cerr << "Usage: mytouch <filename>\n";
    }
    else if (command == "myrm") {
        if (args.size() > 1) myrm(args[1]);
        else cerr << "Usage: myrm <filename>\n";
    }
    else if (command == "mytime") {
```

```
void shellLoop() {
    string input;
    cout << "Welcome to MyShell! Type 'myexit' to quit.\n";

    loadConfig();

    HANDLE hStdIn = GetStdHandle(STD_INPUT_HANDLE);
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

    while (true) {
        char cwd[MAX_PATH];
        string prompt;

        if (GetCurrentDirectoryA(MAX_PATH, cwd)) {
            string configPrompt = shellConfig.getEnv("PROMPT");
            prompt = configPrompt.empty() ? string(cwd) + "> " : configPrompt;
        } else {
            prompt = shellConfig.getEnv("PROMPT").empty() ? "myshell> " : shellConfig.getEnv("PROMPT");
        }

        cout << prompt << flush;

        input.clear();
        size_t cursorPos = 0;
```

```
while (true) {
    while (true) {

        if (ch == 9) {
            size_t lastSpace = input.find_last_of(" ");
            string prefix = (lastSpace == string::npos) ? input : input.substr(lastSpace + 1);

            vector<string> completions = getCompletions(prefix);
            if (!completions.empty()) {
                string completion = completions[0];
                input = (lastSpace == string::npos ? "" : input.substr(0, lastSpace + 1)) + completion;
                cursorPos = input.length();


                cout << "\r" << prompt << input << " ";
                cout << "\r" << prompt;
                for (size_t i = 0; i < cursorPos; ++i) cout << input[i];
            }
        } else if (ch == 13) {
            cout << endl;
            break;
        } else if (ch == 8) {
            if (cursorPos > 0) {
                input.erase(cursorPos - 1, 1);
```

```
        if (!input.empty()) {
            history.addCommand(input);
        }

        executeCommand(input);
    }

    history.saveHistoryToFile();
}
```

```
void mycd(const string& path) {
    wstring wpath = to_wstring(path);
    if (!SetCurrentDirectoryW(wpath.c_str())) {
        cerr << "Error: Unable to change directory to " << path << "\n";
    }
}

void mypwd() {
    wchar_t cwd[MAX_PATH];
    if (GetCurrentDirectoryW(MAX_PATH, cwd)) {
        wcout << cwd << "\n";
    } else {
        cerr << "Error: Unable to get current directory\n";
    }
}

void myecho(const string& text) {
    cout << text << "\n";
}

void myexit() {
    cout << "Exiting the shell...\n";
    exit(0);
}
```

```
void ProcessManager::addBackgroundProcess(PROCESS_INFORMATION pi) {
    backgroundProcesses.push_back(pi);
}

void ProcessManager::listBackgroundProcesses() {
    if (backgroundProcesses.empty()) {
        std::cout << "No background processes.\n";
        return;
    }

    std::cout << "Background processes:\n";
    for (size_t i = 0; i < backgroundProcesses.size(); ++i) {
        std::cout << "Job ID: " << i + 1 << "\n";
        std::cout << "PID: " << backgroundProcesses[i].dwProcessId << "\n";
    }
}

void ProcessManager::bringToForeground(int jobId) {
    if (jobId <= 0 || jobId > backgroundProcesses.size()) {
        std::cerr << "Invalid job ID.\n";
        return;
    }

    PROCESS_INFORMATION pi = backgroundProcesses[jobId - 1];
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
```

## SNAPSHOTS

```
Welcome to MyShell! Type 'myexit' to quit.
C:\vscode\PBL\PBL_CustomShell> ping google.com

Pinging google.com [2404:6800:4002:812::200e] with 32 bytes of data:
Reply from 2404:6800:4002:812::200e: time=37ms
Reply from 2404:6800:4002:812::200e: time=36ms
Reply from 2404:6800:4002:812::200e: time=97ms
Reply from 2404:6800:4002:812::200e: time=98ms

Ping statistics for 2404:6800:4002:812::200e:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 36ms, Maximum = 98ms, Average = 67ms
C:\vscode\PBL\PBL_CustomShell> mypwd
C:\vscode\PBL\PBL_CustomShell
```

```
C:\vscode\PBL\PBL_CustomShell> myhelp
List of supported commands:
  mycd <path>      - Change directory
  mypwd            - Print current directory
  myecho <text>    - Print text to the console
  myexit           - Exit the shell
  myclear          - Clear the screen
  myls             - List files in the current directory
  mycat <file>     - Display contents of a file
  mydate           - Show current date and time
  mymkdir <dir>    - Create a new directory
  myrmdir <dir>    - Remove an empty directory
  mymv <src> <dst> - Rename or move a file
  mycp <src> <dst> - Copy a file
  mytime           - Show the current system time
  mytouch <file>   - Create a new file (like touch in Linux)
  myhelp           - Show this help message
  myrm <file>      - Remove a file
C:\vscode\PBL\PBL_CustomShell> myls
.git
builds
include
src
C:\vscode\PBL\PBL_CustomShell> myecho HELLO EVERYONE
HELLO EVERYONE
C:\vscode\PBL\PBL_CustomShell> mydate
Sun May 25 14:40:40 2025
```

```
C:\vscode\PBL\PBL_CustomShell> mysystemtasks
PID             Process Name
---------------------------------
5304            ETDCtrl.exe
6772            ETDTouch.exe
7460            sihost.exe
7488            svchost.exe
7540            svchost.exe
7596            svchost.exe
7800            taskhostw.exe
8096            igfxEM.exe
7196            Explorer.EXE
8896            svchost.exe
8176            LocationNotificationWindows.exe
9244            SearchHost.exe
9332            StartMenuExperienceHost.exe
9580            RuntimeBroker.exe
9708            svchost.exe
9732            Widgets.exe
10436           DllHost.exe
11256           TextInputHost.exe
10180           DSATray.exe
12208           steam.exe
12248           chrome.exe
11692           WebexHost.exe
5672            chrome.exe
1604            chrome.exe
2864            chrome.exe
10916           chrome.exe
```

```
C:\vscode\PBL\PBL_CustomShell> killtask WhatsApp.exe
Killed process: WhatsApp.exe [PID: 13924]
C:\vscode\PBL\PBL_CustomShell>
```

```
Welcome to MyShell! Type 'myexit' to quit.
C:\vscode\PBL\CustomShell> mycat < test.txt
Hello
World
C:\vscode\PBL\CustomShell> myecho Everyone >> test.txt
C:\vscode\PBL\CustomShell> mycat < test.txt
C:\vscode\PBL\CustomShell> mycat test.txt
Hello
World
Everyone
C:\vscode\PBL\CustomShell>
```

```
C:\vscode\PBL\CustomShell> myexport var_1=HeLLo
C:\vscode\PBL\CustomShell> myecho ${var_1}
HeLLo
C:\vscode\PBL\CustomShell> unset var_1
C:\vscode\PBL\CustomShell> myecho ${var_1}
Usage: myecho <text>
C:\vscode\PBL\CustomShell>
```

```
C:\vscode\PBL\CustomShell> env
PATH=C:/vs code/CustomShell/bin:$PATH
VSCODE_INJECTION=1
PROCESSOR_LEVEL=6
RegionCode=APJ
platformcode=KV
GIT_ASKPASS=c:\Users\Dinesh Joshi\AppData\Local\Programs\Microsoft VS Code\resources\app\extensions\git\dist\askpass.sh
Path=C:\mingw64\bin;C:\ffmpeg\bin;C:\Program Files\Common Files\Oracle\Java\javapath;C:\Python312\Scripts\;C:\Python312\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files\nodejs\;C:\ProgramData\chocolatey\bin;C:\Program Files\Git\cmd;C:\Program Files\Git\mingw64\bin;C:\Program Files\Git\usr\bin;C:\Program Files\HP\HP One Agent;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files\dotnet\;C:\Program Files\MySQL\MySQL Server 8.0\bin;C:\Users\Dinesh Joshi\anaconda3;C:\Users\Dinesh Joshi\anaconda3\Library\mingw-w64\bin;C:\Users\Dinesh Joshi\anaconda3\Library\usr\bin;C:\Users\Dinesh Joshi\anaconda3\Library\bin;C:\Users\Dinesh Joshi\anaconda3\Scripts;C:\Program Files\MySQL\MySQL Shell 8.0\bin\;C:\Users\Dinesh Joshi\AppData\Local\Programs\Python\Python310\Scripts\;C:\Users\Dinesh Joshi\AppData\Local\Programs\Python\Python310\;C:\Users\Dinesh Joshi\App;C:\Program Files\MongoDB\Server\8.0\bin;C:\Program Files (x86)\Dev-Cpp\MinGW64\bin;C:\Program Files (x86)\GnuWin32\bin;C:\Users\Dinesh Joshi\anaconda3;C:\Users\Dinesh Joshi\anaconda3\Library\mingw-w64\bin;C:\Users\Dinesh Joshi\anaconda3\Library\usr\bin;C:\Users\Dinesh Joshi\anaconda3\Library\bin;C:\Users\Dinesh Joshi\anaconda3\Scripts;C:\Program Files\MySQL\MySQL Shell 8.0\bin\;C:\Users\Dinesh Joshi\AppData\Local\Programs\Python\Python310\Scripts\;C:\Users\Dinesh Joshi\AppData\Local\Programs\Python\Python310\;C:\Users\Dinesh Joshi\AppData\Local\Microsoft\WindowsApps;C:\Users\Dinesh Joshi\AppData\Local\Programs\Microsoft VS Code\bin;C:\MinGW\bin;C:\Users\Dinesh Joshi\AppData\Roaming\npm;C:\Program Files\mongosh\
DriverData=C:\Windows\System32\Drivers\DriverData
OS=Windows_NT
```

## LIMITATIONS

Despite the robust architecture and comprehensive features integrated into the CustomShell project, there are several limitations that exist due to time constraints, scope boundaries, and platform-specific challenges. Acknowledging these limitations is crucial for evaluating the current state of the shell and identifying areas for future improvement.

**1. Platform Dependency**
CustomShell is built specifically for the Windows operating system using C++ and Windows-specific system calls like CreateProcess(). This makes the shell non-portable to Unix-like systems such as Linux or macOS without significant changes in code structure and API usage.

**2. Lack of GUI Interface**
The shell operates entirely in a command-line environment. There is no graphical user interface (GUI) implemented, which may limit ease of use for users unfamiliar with terminal-based interaction. While this was a conscious design choice to focus on core OS concepts, it affects user accessibility and visual customization.

**3. Limited Error Handling**
Although basic error messages are displayed for invalid inputs and failed system calls, the shell does not include comprehensive error reporting or recovery mechanisms. More granular error descriptions and logging could significantly improve usability and debugging.

**4. No Support for Scripting**
CustomShell is currently intended for interactive use only. It does not support the execution of shell scripts (.bat or .sh files) or user-defined script blocks. This limits its usefulness for automation and complex workflows.

**5. Single-Session Command History**
The command history implemented in the shell is temporary and session-based, meaning once the shell is closed, the history is lost. There is no persistent storage (e.g., history file) to maintain command history across sessions.

**6. Basic Job Control**
The implementation of background processes and job control (jobs, fg, bg) is functional but limited in complexity. Features like signal forwarding, job prioritization, or process suspension via custom signals are not yet implemented.

**7. Incomplete Environment Variable Handling**
While basic command execution is supported, handling of environment variables is limited. The shell does not yet parse or expand variables like $PATH, $USER, or support exporting new variables dynamically during runtime.

## ENHANCEMENTS

While the CustomShell project successfully implements core functionalities such as command execution, built-in commands, job control, I/O redirection, and autocomplete, there remains significant scope for enhancement. These enhancements aim to improve usability, increase feature coverage, and bring the shell closer to the capabilities of professional-grade terminal environments.

### 1. Persistent Command History

Currently, the shell maintains command history only during the active session. An important enhancement would be to implement persistent command history by storing commands in a file (e.g., .myshell_history) and reading it back when the shell restarts. This allows users to reuse previously executed commands across multiple sessions.

### 2. Advanced Job Control

While basic background processing and job listing (jobs, fg, bg) have been implemented, job control can be enhanced by:

- Adding support for suspending and resuming jobs using signals.
- Handling more job states (e.g., stopped, continued).
- Displaying detailed job metadata like CPU usage or runtime.

### 3. Environment Variable Management

A major area of improvement is the dynamic management of environment variables. This includes:

- Supporting commands like export, unset, and env.
- Parsing and substituting variables like $PATH, $HOME, and $USER.
- Allowing runtime modification of the environment.

### 4. Scripting and Batch Execution

The shell can be extended to support user-defined scripts written in a simplified scripting language or even batch-style command files. This would enable automation and make the shell suitable for scripting tasks.

### 5. Tab Completion for Commands and Paths

The current tab-completion can be improved to:

- Suggest available system commands and user-created aliases.
- Dynamically list file and directory names for valid paths during input.
- Integrate predictive suggestions based on recent history.

### 6. Aliases and User Profiles

Adding alias support allows users to define shortcuts for long commands (e.g., alias ll="ls -la"). CustomShell can also support a startup file like .myshellrc to read user preferences, environment variables, and aliases on startup.

# CONCLUSION

The development of CustomShell has been a significant and insightful journey into the world of system-level programming and operating system concepts. This project successfully demonstrates how a shell can be built from the ground up using the C++ language and the Windows API, providing core functionalities similar to traditional command-line interpreters.

Throughout the project, various important aspects of operating systems were explored and implemented, including process creation and management, input/output redirection, piping, signal handling, and command parsing. Each phase of development contributed to building a functional and modular shell that can handle both built-in and external commands while offering features like job control, command history, and autocomplete functionality.

The absence of a GUI was a deliberate design decision to keep the focus strictly on core shell logic and command-line interactions. This helped in reinforcing fundamental programming skills and system call usage, which are often abstracted in high-level development environments.

One of the major strengths of the project lies in its modular architecture, which makes it easy to maintain and expand. Although certain advanced features such as scripting support, persistent command history, and environment variable management were not fully implemented in this version, the foundation has been laid for future enhancements.

In conclusion, CustomShell stands as a practical demonstration of applied operating system principles and serves as an educational tool for understanding how modern shells operate under the hood. It highlights not only the technical knowledge gained but also the problem-solving, debugging, and software design skills developed throughout the course of the project.

## REFERENCES


1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
2. Kerrisk, M. (2010). The Linux Programming Interface: A Linux and UNIX System Programming Handbook. No Starch Press.
3. Microsoft Documentation: CreateProcess Function
4. Robbins, A., & Beebe, N. H. F. (2005). Classic Shell Scripting. O'Reilly Media.
5. Johnson, R., & Lee, K. (2021). A C++ Programming Shell to Simplify GUI Development in a Numerical Methods Course. ASEE
6. Williams, P. (2025). Reinventing PowerShell in C/C++. SCRT Blog