A red text on a black background

AI-generated content may be incorrect.

**PROJECT AND TEAM INFORMATION**

## Project Title

(Try to choose a catchy title. Max 20 words).

|  |
| --- |
| LiteShell: - A lightweight, efficient and user-friendly command line shell. |

## Student/Team Information

|  |  |
| --- | --- |
| Team Name: | Tech Trio |
| Team member 1 (Team Lead)  (Name, Student ID, Email, Picture): | Ishika Sharma – 220111796  [ishikasharma24124@gmail.com](mailto:ishikasharma24124@gmail.com) |
| Team member 2  (Name, Student ID, Email, Picture): | Diksha – 220112703 [danudiksha892@gmail.com](mailto:danudiksha892@gmail.com) |
| Team member 3  (Name, Student ID, Email, Picture): | Mahi Tyagi - 220123195 [tyagimahi716@gmail.com](mailto:tyagimahi716@gmail.com) |
| Team member 4  (Name, Student ID, Email, Picture): | NA |

**PROJECT PROGRESS DESCRIPTION**

## Project Abstract

(Brief restatement of your project’s main goal. Max 300 words).

|  |
| --- |
| The main objective for this project is to learn how an operating system works improve C/C++/Python coding skills and understand Linux commands by creating a simple custom shell. This will help in building a strong resume and also to learn and add features like scripting, multithreading, automation etc. A shell is a program that allows users to interacts with operating system by typing commands. |

## Updated Project Approach and Architecture (Describe your current approach, including system design, communication protocols, libraries used, etc. Max 300 words).

|  |
| --- |
| **1. Requirements & Design**  **Defining core function: -**   * Command Execution * I/O redirection * Pipelining * Built in Commands   **2. Technology Stack**   * Programming Language: C++ (for direct system interaction) * System Calls: fork(), waitpid(). * Version Control: Git & GitHub for collaboration. * Header Files - iostream, string, vector, algorithm, unistd.h, sys/wait.h, cstring, fstream   **3. Development Plan**   * Core Shell Implementation: -  1. Implement Command parsing to split input into commands and requirement. 2. Add basic command execution using fork() and execvp().  * Built-in commands: -  1. Implement cd, exit, echo, etc. 2. Handle core message.  * Implement redirection using dup2(). * Implement piping to perform command chaining. * Background Process Management. * Enhancement & Optimization. * Testing and fixing bugs and edge cases. |
| **System Architecture –**       * **User Input Handler** - accepts & handles user commands from terminal. * **Command parser** - Splits input into command, arguments and special operators. * **Execution engine** - Run system commands using fork() and execvp(). * **Built in commands** - Handles commands like cd, exit, pwd without external process. * **Redirection & Piping** - Manage I/O redirection * **Process Manager** - Handles background processes and manages running processes. * **Error handling and Logging** - Detects invalid commands, report errors and log system issues. |

## 

## Tasks Completed (Describe the main tasks that have been assigned and already completed. Max 250 words).

|  |  |
| --- | --- |
| Task Completed | Team Member |
| User Input Handler  Command Parser  Execution Engine  Built-in commands  Process Manager  Error Handling  I/O Redirection  Pipelining | Diksha  Mahi  Ishika, Mahi  Ishika  Mahi  Diksha  Diksha, Mahi  Ishika |

## Challenges/Roadblocks (Describe the challenges that you have faced or are facing so far and how you plan to solve them. Max 300 words).

|  |
| --- |
| * **Working with Linux** - Used Ubuntu for Linux Commands * **Command Parsing and Tokenization** - Accurately parsing user input, especially when handling quoted strings, escape characters etc. * **Handling Built-in vs External Commands** - Maintain a map of built-in commands and their corresponding handler functions. * **Process Creation and Management** - Use fork() to create a child process and execvp() to execute commands.   Use waitpid() in the parent process to handle cleanup and avoid zombies. |

## Tasks Pending (Describe the main tasks that you still need to complete. Max 250 words).

|  |  |
| --- | --- |
| Task Pending | Team Member (to complete the task) |
|  |  |

## Project Outcome/Deliverables

(Describe what are the key outcomes / deliverables of the project. Max 200 words).

|  |
| --- |
| A lightweight efficient command line shell that executes system commands.  A parser for parsing and tokenizing the input.  Supports foreground, background execution and process control.  Handles I/O redirection and piping.  Detects invalid inputs, prevents crashes and ensures smooth execution. |

# Progress Overview (Summarize how much of the project is done, what's behind schedule, what's ahead of schedule. Max 200 words.)

|  |
| --- |
| **Completed (On Schedule)**   * **User Input Handler** – Implemented and stable. * **Command Parser** – Functional with expected input handling. * **Execution Engine** – Successfully executes commands. * **Built-in Commands** – Core built-ins (e.g., cd, exit) implemented. * **Process Manager** – Handles fork(), exec(), and process cleanup. * **Error Handling** – handling of common runtime and syntax errors. * **I/O Redirection** – Yet to be implemented. This includes support for >, <, >>, etc. * **Piping** – Still pending. Requires handling of inter-process communication using pipe(). |

# Codebase Information (Repository link, branch, and information about important commits.)

|  |
| --- |
| GitHub Repository – https://github.com/Mahi-1905/LiteShell  Main Branch  Commits done –  f3538de (HEAD -> main, origin/main, origin/HEAD) modifying some commands  ed3d48b adding doc files  33099dd adding files  e190882 Initial commit |

## Testing and Validation Status (Provide information about any tests conducted)

|  |  |  |
| --- | --- | --- |
| Test Type | Status (Pass/Fail) | Notes |
| Tested using Ubuntu command line  Tested using text files | Pass  Pass | Executes the built-in commands  Executes commands in files |

# Deliverables Progress (Summarize the current status of all key project deliverables mentioned earlier. Indicate whether each deliverable is completed, in progress, or pending.)

|  |
| --- |
| **Completed (On Schedule)**   * **User Input Handler** – Implemented and stable. * **Command Parser** – Functional with expected input handling. * **Execution Engine** – Successfully executes commands. * **Built-in Commands** – Core built-ins (e.g., cd, exit) implemented. * **Process Manager** – Handles fork(), exec(), and process cleanup. * **Error Handling** – handling of common runtime and syntax errors. * **I/O Redirection** – Yet to be implemented. This includes support for >, <, >>, etc. * **Piping** – Still pending. Requires handling of inter-process communication using pipe(). |