

# Technical Report: Mini UNIX Shell Implementation

Student Name: Jaskaran Singh

Date: 9 Dec 2025

## 1. Introduction

This project involves the design and implementation of a custom UNIX-style command-line shell written in C. The shell serves as an interface between the user and the operating system kernel, capable of parsing commands, managing processes, and handling I/O redirection.

## 2. System Architecture

The shell follows the standard **REPL (Read-Evaluate-Print Loop)** architecture. It operates as an infinite loop that performs three distinct stages:

1. **Read:** Accepts user input.
2. **Parse:** Tokenizes the input string into arguments.
3. **Execute:** Determines if the command is a built-in function or an external program and executes it accordingly.

### 2.1 Process Management Model

The core execution logic relies on the `fork()` and `execvp()` system calls.

- **The Parent (Shell):** Duplicates itself using `fork()`. It then typically waits for the child to terminate using `waitpid()`.
- **The Child:** Replaces its memory image with the requested program using `execvp()`.
- **Separation of Concerns:** This architecture ensures that if an external program crashes, the shell process remains unaffected.

## 3. Implementation Details

### 3.1 Advanced Argument Parsing

Initially, the standard `strtok` library function was used for tokenization. However, this failed to handle quoted strings (e.g., `echo "Hello World"`). A custom state-machine parser was then implemented.

- **Logic:** The parser iterates through the raw input string pointer by pointer.
- **Result:** This allows the shell to treat space-separated words within quotes as a single argument while stripping the quotes before execution.

## 3.2 I/O Redirection

Input (<) and Output (>) redirection were implemented by manipulating file descriptors (FDs) inside the child process *before* calling `execvp`.

- **Mechanism:** The shell opens the target file using `open()` with specific flags (`O_CREAT` | `O_TRUNC` | `O_WRONLY` for output).
- **FD Swapping:** `dup2()` is used to overwrite `STDIN_FILENO` or `STDOUT_FILENO` with the new file descriptor.
- **Safety:** Because this occurs only in the child process, the parent shell's I/O streams remain untouched.

## 3.3 Pipeline Support

Implementing pipelines (`cmd1 | cmd2`) required creating a unidirectional data channel using `pipe()`.

- **Dual-Fork Strategy:** The shell forks two children simultaneously.
  - **Child 1 (Left):** Connects `STDOUT` to the *write-end* of the pipe.
  - **Child 2 (Right):** Connects `STDIN` to the *read-end* of the pipe.
- **Process Synchronization:** The parent process closes both ends of the pipe immediately after forking to prevent deadlocks.

## 3.4 Signal Handling & Zombie Cleanup

To satisfy the non-functional requirements, two signal handlers were registered:

1. **SIGINT (Ctrl-C):** Caught by the shell to prevent termination. The handler prints a new line and reprints the prompt. The child process (which inherits default signal behavior) correctly terminates.
2. **SIGCHLD (Zombie Cleanup):** Caught whenever a child process terminates. The handler calls `waitpid(-1, NULL, WNOHANG)` to reap background processes immediately, preventing zombie accumulation.

## 4. Development Workflow

Per project requirements, a structured Git workflow was utilized to ensure incremental development. Features were developed on dedicated branches and merged into main only after verification.

### Branching Strategy:

- `feature/parser`: Implementation of the tokenizer.
- `feature/execute`: Core fork/exec logic.
- `feature/redirection`: I/O redirection support.

- feature/pipeline: Pipe logic.
- feature/quotes: Upgrading the parser for quoted strings.
- fix/wait-race: Patches for process race conditions.

## Commit History Visualization:

The screenshot shows the GitHub commit history for the repository 'Cornerstone-Project-col7001' by user 'mrsingh3131'. The interface is in dark mode. At the top, there's a navigation bar with links to Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below this, the 'Commits' section is active, showing a list of commits on the 'main' branch. The commits are grouped by date: 'Commits on Dec 9, 2025' and 'Commits on Dec 8, 2025'. Each commit entry includes a title, a description, the commit hash, and a link to the commit details. The commits on Dec 9, 2025, include: 'added Lab -1 pdf', 'Updated message', 'removed old parse\_input function', 'added makefile', 'added a proper message when ctrl+c is hit', 'Use waitpid to avoid race condition with signal handler', 'Prevented shell termination on Ctrl-C (SIGINT)', 'Improved parser to handle quoted strings', 'Implemented pipe support with dual fork strategy', 'Implemented background execution and zombie cleanup', 'Implemented input and output redirection support', and 'Implemented fork/exec/wait pattern and cd built-in'. The commits on Dec 8, 2025, include: 'Implement tokenizer using strtok', 'Initial commit: Basic REPL (Read-Eval-Print-Loop) skeleton', and 'Initial commit'.

Commit Title	Commit Hash	Author	Time Ago
added Lab -1 pdf	fbefcd23	Jaskaran Singh	29 minutes ago
Updated message	8765dc4	Jaskaran Singh	30 minutes ago
removed old parse_input function	36d970b	Jaskaran Singh	1 hour ago
added makefile	dc4eb1f	Jaskaran Singh	1 hour ago
added a proper message when ctrl+c is hit	e48cb56	Jaskaran Singh	1 hour ago
Use waitpid to avoid race condition with signal handler	0314e41	Jaskaran Singh	1 hour ago
Prevented shell termination on Ctrl-C (SIGINT)	92f0bfd	Jaskaran Singh	1 hour ago
Improved parser to handle quoted strings	64dcdd7	Jaskaran Singh	1 hour ago
Implemented pipe support with dual fork strategy	6dd6d80	Jaskaran Singh	1 hour ago
Implemented background execution and zombie cleanup	1bd8934	Jaskaran Singh	7 hours ago
Implemented input and output redirection support	ef4ac77	Jaskaran Singh	8 hours ago
Implemented fork/exec/wait pattern and cd built-in	3de0392	Jaskaran Singh	8 hours ago
Implement tokenizer using strtok	5d9dae4	Jaskaran Singh	12 hours ago
Initial commit: Basic REPL (Read-Eval-Print-Loop) skeleton	7074e0a	Jaskaran Singh	13 hours ago
Initial commit	c3eda4b	mrsingh3131	14 hours ago

## 5. Challenges and Solutions

### 5.1 The ps Race Condition

Problem: Initially, running the ps command would cause the shell to hang or behave erratically.

Analysis: The global SIGCHLD handler was "reaping" the ps process immediately upon its completion. When the main execution loop subsequently called wait(NULL), it blocked indefinitely because the intended child was already gone.

Solution: The logic was refactored to use waitpid(pid, ...) in the main execution loop. This ensures the shell waits only for the specific foreground process it just spawned, ignoring

background processes handled by the signal handler.

## 5.2 Async-Signal Safety

Problem: Using `printf` inside the `SIGINT` handler is unsafe and can lead to deadlocks.

Solution: `printf` was replaced with `write()`, which is an async-signal-safe system call, ensuring stability during interrupts.

## 6. Testing Strategy

The shell was verified against a suite of 11 test cases covering all functional requirements. The tests were executed in a macOS (UNIX-based) environment to ensure POSIX compliance.

1. **Basic Command Execution:** Verified using `ls` to confirm the shell correctly performs PATH lookup and executes external programs.
2. **Argument Parsing:** Verified using `ls -la /tmp` to confirm the parser correctly handles flags and arguments.
3. **Built-in Commands:** Verified using `cd ..` followed by `pwd` to ensure directory changes affect the shell's process state directly rather than a child process.
4. **Quoted String Support:** Verified using `mkdir "my folder"` to confirm the custom state-machine parser treats quoted strings with spaces as single arguments.
5. **Output Redirection:** Verified using `echo ... > test_log.txt` to confirm standard output is correctly redirected to files using `dup2`.
6. **Input Redirection:** Verified using `wc -w < test_log.txt` to confirm standard input is correctly read from files.
7. **Pipeline Support:** Verified using `ls -la | grep shell` to confirm the `pipe()` system call correctly connects the standard output of the first command to the standard input of the second.
8. **Background Execution:** Verified using `sleep 10000 &` followed by `ps` to confirm the prompt returns immediately and the shell can handle concurrent processes without hanging.
9. **Foreground Signal Handling:** Verified by interrupting a foreground `sleep` command with `Ctrl-C` to confirm the shell catches `SIGINT` and remains active while the child process terminates.
10. **Background Signal Handling:** Verified by using `Ctrl-C` while a background process runs, confirming that the signal affects the process group and effectively terminates the background job.
11. **Robustness & Error Handling:** Verified using invalid commands to confirm the shell handles `execvp` failures gracefully (printing error messages) without crashing.