C-Shell: Documentation

Saiyam Jain

Introduction

This document provides a comprehensive guide to the custom Unix-like shell implemented in C, known as C-Shell. Designed to mimic core functionalities of popular shells such as bash and zsh, it introduces a controlled, educational environment to understand internal mechanisms like process management, signal handling, command parsing, I/O redirection, and piping. This document explains each command in detail, how it mimics the behavior of standard shells, and practical scenarios where each feature proves useful.

Building and Running C-Shell

Use the following commands to compile and run the shell:

```
make
./main
```

Detailed Command Descriptions

hop

Changes the current working directory, similar to 'cd' in bash.
• Supports relative (., ..) and absolute paths.
• Special tokens (~ for home, - for previous directory).
• Sequential path change if multiple paths are supplied.
Use case: Navigate directories interactively within the shell.

reveal

Mimics 'ls' command functionality.
• Lists files and directories alphabetically.
• -l displays detailed info (permissions, size, modification time).
• -a includes hidden files (starting with .).
• Color-codes output: Executables (green), Directories (blue), Files (white).
Use case: Directory content exploration.

log

Maintains a history of up to 15 commands (excluding consecutive duplicates and log commands).
• 'log' displays history.
• 'log purge' clears history.

• 'log execute <index>' re-executes command at a specified index.
Use case: Quick access to frequently/recently used commands.

proclore

Displays process information mimicking 'ps' and 'top' utilities.
• Process ID, status, group, memory usage, and executable path.
• Defaults to current shell process if no PID is given.
Use case: Process monitoring and debugging.

seek

Searches for files/directories under a target directory.
• -d restricts search to directories.
• -f restricts search to files.
• -e executes action if a unique match is found (open or change directory).
Use case: Locating files and directories efficiently.

activities

Lists background processes spawned by the shell.
• Shows PID, command, and state (Running/Stopped).
Use case: Process management.

ping

Sends a Unix signal to a process.
• Requires PID and signal number.
Use case: Process control (e.g., terminate, pause, resume processes).

fg / bg

'fg' moves a background process to foreground.
'bg' resumes a stopped background process.
Mimics built-in job control in bash.
Use case: Interactive process management.

neonate

Continuously monitors and prints the latest system PID at intervals.
• User-defined polling interval.
• Stops on 'x' keypress.
Use case: System monitoring and learning process creation patterns.

iMan

Fetches manual pages over the internet using sockets (default: man.he.net).
• Mimics 'man' command with external retrieval.
Use case: On-demand remote manual access in restricted environments.

Core Features and Implementation Notes

• Command Parsing: Tokenizes input using spaces, handles piping and I/O redirection operators.
• Signal Handling: Responds to Ctrl-C, Ctrl-Z, and Ctrl-D as bash would.
• Background Processes: Detach child processes to run in background using '&'.
• Aliases and Functions: Loaded from .myshrc file, similar to bashrc.
• I/O Redirection: Supports <, >, and >> operators for file I/O control.
• Pipes: Implements Unix pipe (|) to chain command outputs as inputs.

Assumptions and Limitations

• Commands assume no whitespace within file and directory names.
• Single redirection each for input and output per command.
• No support for command chaining using semicolon or '&&'.
• Neonate command suppresses keyboard signals until 'x' pressed.
• Limited signal handling within piped commands.
• Aliases and functions don't support I/O redirection.
• No support for multiline command input.

Practical Use Cases

• Educational tool for understanding Unix process and signal control.
• Safe sandbox for testing command parsing and I/O operations.
• Customizable local environment for running batch jobs and background processes.
• Demonstration platform for advanced C programming concepts like system calls, fork, exec, wait, and pipe.