cash — Command and Script Shell

Project Information

Field	Details
Course ID:	CSE323
Course Name:	Operating Systems
Faculty Initials:	SMU1
Student Name:	Adib Ar Rahman Khan
Student ID:	2212708042
Date:	19/4/24

Table of Contents

1. Introduction

cash (*Command And Script Shell*) is a lightweight yet powerful command-line shell, crafted in C as part of the CSE323 Operating Systems course.

The goal?
To create a simple, readable shell that **teaches core OS concepts in action** — process control, file manipulation, IPC, signals, and more.
Instead of mimicking complex shells like Bash, **cash** focuses on clarity, hands-on practice, and real-world OS applications.

2. Core Features

cash offers a surprisingly rich feature set:

- **Command Execution** — Runs external programs (like `ls`, `pwd`, `echo`).
- **Built-in Commands** — `exit`, `cd [dir]`, `clear`, `jobs`, `fg %<id>`, `bg %<id>`.
- **Input/Output Redirection** — Supports `< file`, `> file`.
- **Pipes** — Connects commands via `|`.
- **Background Execution** — Runs jobs asynchronously with `&`.
- **Job Control** — Lists jobs, moves jobs between foreground and background.
- **Signal Handling** — Cleans up terminated processes (prevents zombies).
- **Command History** — Persistent command history via `~/.cash_history`.
- **Line Editing** — Smooth navigation using GNU Readline (arrows, backspace, etc).

3. Operating Systems Concepts Demonstrated

cash helps you understand and apply:

Process Management

- `fork()`, `execvp()`, `waitpid()` — process lifecycle
- Managing PIDs and process groups
- Foreground/background distinction

Inter-Process Communication (IPC)

- Anonymous pipes (`pipe()`, `dup2()`)

File System & File I/O

- Redirection of stdin, stdout, stderr
- Changing directories, environment variables

Memory Management

- Dynamic memory (`malloc()`, `free()`)
- Heap vs Stack allocation

Signal Handling

- Handling `SIGCHLD`
- Sending signals (e.g., `kill(SIGCONT)`)

Concurrency

- Managing asynchronous background jobs
- Handling signals asynchronously

System Calls

- Direct interaction with Linux/Unix kernel

Terminal Control

- Managing terminal foreground process groups (`tcsetpgrp()`, `tcgetpgrp()`)

4. Code Structure Overview

cash is cleanly modular:

Function	Purpose
`main()`	Startup, REPL loop, command reading, history management.
`execute_pipeline()`	Handles commands, pipes, background jobs.
`execute_single_command()`	Executes built-in or external command.
`handle_child_execution()`	Child-side I/O redirection and execution.
`parse_command()`	Parses input string into command arguments and redirection.
Job Management	Track jobs using `add_job()`, `remove_job_by_pgid()`, etc.
Signal Handling	Manage child process termination (`SIGCHLD`).
Global Variables	Manage job list, PGID, terminal file descriptor, shell state.

5. Detailed Feature Implementation & Testing (Expanded)

5.1 Basic Command Execution

Inside cash:

- Forks child → `execvp()` in child → Parent waits.

Test:

```
ca$h> ls -l /etc
ca$h> echo Hello CSE323!
ca$h> pwd
ca$h> non_existent_command
```

Expect:

- Successful program execution or clean error for missing commands.

5.2 Built-in Commands

Inside cash:

- Built-ins like `cd`, `exit`, `jobs` are handled internally.

Test:

```
ca$h> cd /tmp
ca$h> pwd
ca$h> clear
ca$h> exit
```

Expect:

- Directory changes, screen clears, graceful shell exit.

5.3 Input/Output Redirection

Inside cash:

- Detect `<`, `>` → Open file → Replace stdin/stdout → `execvp()`.

Test:

```
ca$h> echo "Line 1" > output.txt
ca$h> cat < output.txt
```

Expect:

- Files created/overwritten; input/output redirected correctly.

5.4 Pipes

Inside cash:

- Detects `|` → Forks twice → Connects stdout→stdin between processes.

Test:

```
ca$h> ls -l | wc -l
```

Expect:

- Correctly piped outputs.

5.5 Background Execution

Inside cash:

- Forks child → Sets process group → Returns immediately without wait.

Test:

```
ca$h> sleep 5 &
```

Expect:

- Prompt returns immediately. Background job listed with `jobs`.

5.6 Job Control (jobs, fg, bg)

Inside cash:

- Maintains job list, controls jobs with process groups and signals.

Test:

```
ca$h> sleep 100 &
ca$h> jobs
ca$h> fg %1
```

Expect:

- Job is correctly moved between background and foreground.

5.7 Signal Handling (SIGCHLD)

Inside cash:

- `SIGCHLD` handler reaps finished child processes (no zombies).

Test:

- Start background jobs → Allow completion → Check no `<defunct>` processes.

Expect:

- Clean removal of finished processes.

5.8 Command History and Editing (Readline)

Inside cash:

- Loads history on startup.
- Saves on exit.
- Supports arrow key editing.

Test:

```
ca$h> echo first
ca$h> echo second
# Use Up/Down Arrows
```

Expect:

- Command recall across sessions.

6. Limitations

cash is intentionally simple:

- No support for quotes (`" "`), globs (`*`), or variables (`$VAR`).
- No redirection appending (`>>`) or here-documents (`<<`).
- Only one `|` pipe handled per command.
- Partial Ctrl+C/Ctrl+Z handling.
- No full environment variable manipulation.
- No script file reading or execution.
- No tab autocompletion.

7. Potential Future Enhancements

Exciting improvements are possible:

- Proper Ctrl+C (`SIGINT`) and Ctrl+Z (`SIGTSTP`) handling.
- Multiple pipelines (`cmd1 | cmd2 | cmd3`).
- Appending output (`>>`).
- Advanced parsing with quotes and variable expansion.
- Tab completion (using Readline features).
- Adding built-ins like `export` , `unset` , `pwd` .
- Scripting support!

8. Building and Running cash

Prerequisites:

- GCC compiler
- GNU Readline library

Install Readline:

```
sudo apt-get install libreadline-dev    # Ubuntu/Debian
sudo dnf install readline-devel         # Fedora
brew install readline                   # macOS (Homebrew)
```

Compile:

```
gcc cash.c -o cash -lreadline -Wall
```

(macOS users may need `-I` and `-L` flags for Homebrew.)

Run:

```
./cash
```

Exit with `exit` command or `Ctrl+D` .

9. Conclusion

cash proves that you don't need thousands of lines of code to implement powerful Operating Systems concepts!
By building a functional shell, this project taught hands-on skills in **process management, IPC, memory management, terminal control, and concurrency**.