Documentation Standards

MyShell: Code and Documentation Standards

Tarun Jaswal, Mack Bautista, Jarod Dacocos

Mount Royal University

COMP 3659

Last Modified: Oct 19, 2025

1. Purpose

This document outlines the standards our group will adhere to when developing our own Linux shell program. Its goal is to ensure our codebase is consistent, readable, maintainable, and easy to debug or extend.

2. General Coding Guidelines

1. **Language:** C
2. **Build System:** All code must compile using the make file without any warnings or errors
3. **System Calls Only:** The shell should primarily rely on Linux system calls and standard C wrappers.
4. **Function Length:**
    - Each function should contain no more than 25 lines of executable code (excluding braces and comments)
    - If longer logic is required, refactor into smaller helper functions
5. **Readability:**
    - Use consistent indentation
    - Use descriptive variable and function names
6. **Bracing Style:** Open braces go on the same line as the condition statements

    E.g.

    ```
    if (condition) {
            statement;
    } else {
            statement;
    }
    ```

7. **Modularity:**
    - Organize source files by logical purpose
    - Each .c file must have an accompanying .h file

3. Documentation Standards

3.1 Function Headers

- Each function must be preceded by a comment block that describes its name, purpose, input(s) and output(s).
- Example:

```
/* —
Function Name: foo

Purpose:
  Serves some purpose

Input:
  i - something …

Output:
  returns 0 upon successful execution
— */
```

3.2 Inline Comments

- Use inline comments to explain non-trivial code segments.
- Avoid comments that restate obvious code.
- Example:

```
/* Handle Input Redirection */
if (...)  {
        …
}
```

4. Naming Conventions

Element Type	Convention	Example
Functions	Lowercase with Underscores	example_func()
Variables	Lowercase with Underscores	example_var
Constants	All Caps with Underscores	EXAMPLE_CONST
Structures	PascalCase	ExampleStruct
Files	Lowercase	filename.c

5. Function Design Rules

1. Single Responsibility Principle:
    - Each function performs exactly one logical task.
2. Testability:
    - Functions must be individually testable in isolation.

6. Header Files

Each module must have a .h file containing:

- Function prototypes
- Struct and constant definitions
- Include guards (ex. #ifndef FOO_H, etc.)

Header files must not include unnecessary headers to avoid circular dependencies.

7. Git

1. Use Git for source control
2. Commit with descriptive messages
3. Do not commit compiled binaries or object files

8. Testing and Quality Assurance

Manual Testing:

- Maintain a test log for each feature (interactive input, redirection, background jobs).

Regression Testing:

- After each modification, re-test existing features.

Error Injection:

- Simulate invalid commands, empty input, and permission errors.

9. Documentation Practices

Maintain a separate README.md that explains:

- Build instructions (make)
- How to run mysh
- Supported features and known limitations

Keep this Code and Documentation Standards document up to date as the project evolves.

10. Example of a Fully Standardized Function

```
/* —
Function: foo

Purpose:
  Returns the square of an integer input if it is non-negative.

Input:
  x - integer value to be squared.

Output:
  Returns the square of x.
— */
int foo(int x)
{
   /* Compute the square */
   int result = x * x;

   return result;
}
```