# Unix/Linux Command Line Interpreter

Mathew Hobson, Nolan Tuttle

College of Engineering and Technology, Grand Canyon University

CST-315: Operating Systems Lecture and Lab

Professor Citro

January 19, 2025

## Project Description

SeaShell is our lightweight command-line interpreter that allows users to execute shell commands interactively or through batch processing. The shell mimics basic functionalities of traditional Unix shells by supporting command execution, batch file processing, and multi-command parsing. It is implemented in C using system calls such as fork(), execv(), and waitpid() to handle process execution.

## Methodology

The approach to implementing our command line interpreter involved the following steps:

1. **Interactive Shell Implementation:** A loop continuously accepts user input, parses commands, and executes them.
2. **Batch File Processing:** When the shell is started with a batch file argument, it reads commands line by line and executes them sequentially.
3. **Command Parsing:** Commands entered interactively or read from a batch file are split based on semicolons to enable multiple command execution.
4. **Process Creation and Execution:** The shell uses fork() to create child processes and execv() to execute commands within them. The parent process waits for child processes to complete before accepting new input.
5. **Exit Mechanism:** The shell recognizes special exit commands (CTRL+X, CTRL+B, quit, exit) to terminate execution.

## Algorithm for Processing Shell Commands

1. **Batch Mode (if applicable):**
   - Open the specified batch file.
   - Read commands line by line.
   - Ignore empty lines and comments (#).
   - Execute each command using system().
2. **Interactive Mode:**
   - Display a prompt (seaShell$).
   - Read user input.
   - Check for exit commands (CTRL+X, CTRL+B, quit, exit).
   - Tokenize input based on semicolons to support multiple commands.
   - For each token:
     - Create a child process (fork()).
     - Execute command using execv().
     - Parent process waits for child process completion.

## Code and Comments

### Handling Batch Files

```
if (argc == 2) // Condition for batch file usage
{
    char *batch_file = argv[1];
    FILE *file = fopen(batch_file, "r");
    char str[100];

    while (fgets(str, sizeof(str), file))
    {
        str[strcspn(str, "\n")] = 0; // Remove newline character
        if (str[0] == '\0' || str[0] == '#') continue; // Skip empty lines/comments
        system(str); // Execute batch command
    }
    fclose(file);
}
```

### Processing User Input

```
while (isRunning)
{
    printf("seaShell$ ");
    fgets(str, sizeof(str), stdin);
```

```
str[strcspn(str, "\n")] = 0;


if (str[0] == 24 || (!strcmp(str, "quit"))) exit(1);
else if (str[0] == 2 || (!strcmp(str, "exit"))) isRunning = 0;
```

**Command Execution**

```
char *command = strtok(str, ";");
 while (command != NULL)
 {
    char *args[] = {"/bin/bash", "-c", command, NULL};
    pid_t pid = fork();

    if (pid == 0) // Child process
    {
        if (execv("/bin/bash", args) == -1)
        {
            perror("exec failed");
            exit(1);
        }
    }
    else if (pid > 0) // Parent process
    {
        int status;
        waitpid(pid, &status, 0);
    }
    command = strtok(NULL, ";");
}
```

## References

GNU. (n.d.). *Bash Reference Manual.* Retrieved from https://www.gnu.org/software/bash/manual/

Stallings, W. (2019). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.

Tuttle, N., Hobson, M., (2025) *CST315* GitHub. https://github.com/nolantuttle/CST315.git