

Assignment 1: Building a Basic Shell

Professor: Dalu Jacob

TA: Anshika Prajapati, Priyal Jain, Saket Kandoi

Deadline: 17th August 11:59 p.m

Introduction

In this assignment, you will create a basic shell similar to the bash shell in Linux. The shell will interpret and execute user input, manage communication between commands through piping, and offer core built-in features for basic command execution. As an enhancement, you will also implement additional functionalities such as input/output redirection, support for command chaining using separators, and extend the shell to handle wildcard expressions and command line completion.

Basic Shell

- Develop a shell that captures user input and executes commands using the `exec` family of system calls.
- The shell should display the prompt `MTL458 >` and remain active, continuously accepting and processing commands until the user terminates it.
- You must not use the `readline` library for input handling.
- Ensure that any errors encountered during command execution are properly handled and clearly displayed as error messages.

Assumptions

- Input will not exceed 2048 characters in length.
- Each command will contain no more than 100 arguments.
- Assume the following folder structure in your home directory:

```
/home/user/  
  file1.txt  
  file2.txt  
  directory1/  
    file3.txt  
  script.sh
```

Contents of each file (Each file ends on a newline):

- `file1.txt`: `hello world`
- `file2.txt`: `foo bar`
- `file3.txt`: `foo foo foo`
- `script.sh`: `echo "This is a script"`

Additional Features

Command Execution

The shell must support the execution of standard Linux commands such as `ls`, `cat`, `echo`, and `sleep`.

Input:

```
echo "hello"
```

Expected Output:

```
hello
```

Pipes

The shell must support basic piping functionality, where the standard output of one command is redirected as the standard input of another. Only single pipe operations are required; support for multiple consecutive pipes is not expected. The `cd` and `history` command will not be used as part of any piped command sequence.

Input:

```
ls | grep file
```

Expected Output:

```
file1.txt  
file2.txt
```

I/O Redirection

Extend the shell to support redirection of input and output using the symbols `<`, `>` and `>>`. Assume single redirection operator only. `cd` and `history` commands won't be a part of any redirection expression. Commands with redirection will not include pipes.

Input:

```
echo sample > output.txt  
cat output.txt
```

Expected Output:

```
sample
```

Command Separators

Implement support for multiple commands on a single line using `;` and `&&` separators. Commands separated by `;` run sequentially, while those joined by `&&` run only if the previous command succeeds.

Input:

```
echo one; echo two
```

Expected Output:

```
one  
two
```

Wildcard Expansion

Implement support for wildcard patterns using `*` in command arguments. For example, a command like `cat file*.txt` should display contents of all files in the current directory that match the pattern.

Input:

```
cat file*.txt
```

Expected Output:

```
hello world  
foo bar
```

Built-in Commands

In addition to executing external commands, your shell should support some built-in commands. Implement the following:

- **cd directoryname**: Change the current working directory of the shell. You can assume the folder names do not contain spaces.
- **history** and **history n**: Display the history of commands entered by the user, oldest to latest. Only command name should be displayed on each line. Maximum history of 2048 commands.

Input:

```
cd directory2  
pwd  
history 2
```

Expected Output:

```
/home/user/directory2  
pwd  
history 2
```

Command Line Completion

Implement support for filename autofilling when the Tab key is pressed. If a single matching file or directory exists for the partially typed word, it should be automatically completed.

User Interrupt

Ensure that the shell program terminates after receiving the command `exit`.

Error Handling

Ensure that your shell handles incorrect arguments or command formats gracefully. Display error messages without crashing the shell and continue to prompt for the next command. The error message for all types of errors should be: `Invalid Command`.

Input:

```
lss
```

Expected Output:

```
Invalid Command
```

Submission

- Make sure you have been added to the course Gradescope. The submission should be in C, and the file should be named `entrynumber_a1.c`.
- Ensure your code is well-commented and follows best practices for readability and maintainability.
- All submissions will be strictly checked for plagiarism, including comparisons with submissions from previous years as well as AI-generated content. Any instance of plagiarism will result in a score of zero for the assignment.

Good luck, and happy coding!