Dan Wilder

UMSL SP15-CMPSCI2750-001

Project #4 : Minimal Shell

2015-04-26

Introduction

This project involved creating a minimal shell called **mysh** written entirely in C. A shell is a program which executes commands. More precisely, a shell interprets user input as a command which it passes to the operating system to perform. Bash and csh are two common shell programs. Shells like these are advanced in the functionality that they offer such as pipes, I/O redirection, and many built-in commands. While useful, these capabilities are beyond the scope of this project. Mysh is intended only to execute the specified command.

What is a command?

In most cases, a command is an executable file. Built-in commands are an exception. The resulting execution of a built-in command is hard-coded within the shell program itself. Mysh contains only one built-in command – *exit*. With this exception, mysh interprets user input as a reference to an executable file.

Startup

Mysh is executed simply by specifying its executable file, mysh, from the command line. In other words, mysh is executed from another shell. Before mysh enters its main loop, a few things take place. Although mysh doesn't have the concept of environmental variables, since it is executed from a shell, it can retrieve some of the calling shell's environmental variables with the use of the getenv() function. Mysh does exactly that. Specifically, mysh extracts MYPATH and PS1 from the calling shell's environment. Defaults exist if the variable is not set which are the value of the PATH environment variable and “>>> ” respectively. Mysh has an internal variable, envPath, which is set to the value of MYPATH/PATH. It is a colon (:) delimited string which indicates which directories for the search for

when looking for a command. Likewise, mysh has an internal variable set to the value PS1 or the default. This variable determines mysh's prompt. The prompt is printed right before the shell is ready to retrieve user input. Thus, its presence indicates to the user that the shell is waiting on user input.

User input

Mysh interprets user input as a reference to an executable file or a built-in command. Though not specified as a requirement, I changed it so that it is actually only the first word of user input that is interpreted as such a reference. Subsequent words, are arguments to the given command. This subtle change, significantly improves the utility of mysh. For example, without this addition “ls -l” is seen as a single command whereas with the addition, ls is seen as command and -l as an argument to that command. This allows the called program's logic to handle the arguments. In this example, “ls -l” will perform as it would if run in bash or csh assuming that ls is an executable that exists in one of the directories given by envPath. If Ctrl-D is entered following mysh's prompt, the shell prints an exiting message and terminates.

Searching for a command

It would be tedious if the full path of a command had to be given each time the user wished to call it. Instead, only the program name is given. A loop exists that will continue until all possible search directories have been ruled out or the command has been found. For each search directory, a full path is constructed as: searchDirectoryPath/commandName. Calling the stat() function with this full path and checking if it returned -1 was used to check if the command existed within that directory. If not, then another full path would be constructed using the next search directory and then checked.

Main Loop

Mysh is intended to run until the user indicates that she wants it to close. In this sense, it makes sense that most of the shell's logic occurs within a loop. The following provides an overview of what happens within this loop.

1) Mysh's prompt printed

2) User input is retrieved

3) Each word of the user input is stored in an index of an array

4) A Command data type is constructed with knowledge of command name and envPath

5) Shift execution based on the results of “probing” that command

6) Free memory associated with Command

The Command data type

For the scope of this project, this was an unnecessary step, but I chose to so to gain some experience with function pointers and I wanted to emulate object-oriented programming using C syntax. Additionally, I believe that encapsulating data in this manner would have made it easier to expand mysh's functionality if I decide to later. *Command* is actually a typedef of a pointer to structure called Command_Struct which has as attributes: name (char *), path (char *), stats (struct stat *). It also has two function pointers: probe, and destruct. A constructor function was defined outside the structure. It takes as parameters a name and list of paths (colon delimited), and returns a Command. Searching for a command was implemented within this constructor. If a command isn't found, command->path will be NULL as will command->stats.

After creating a variable of this type, the probe method returns results. The return value is a sum

of macro values where each macro value represents the presence of some tested condition. The macro values were defined in a separate header file. Each is a power of 2 which allows client code to use a statement such as 'if (results & macro_value)' where results is the return value of probe. If I were to extend mysh, one approach would be to add macro values for new testing conditions and alter the probe method to reflect these changes. Client code operation would remain the same: results & newMacro. For this project, two such macro values were defined which can be used to check if a user has permission to execute the file and if the file is a regular file.

The destruct function pointer is the C equivalent to the destructor found in object-oriented programming.

Executing the command

Assuming a valid command was found, mysh calls fork(). The child process executes the command with the following statement: execv(command->path, myargv). Here, command is a pointer to a Command_Struct. Thus, command->path references the full path of a command. Myargv is the variable name which stores each word of user input at an index. This statement executes the specified command and passes the user input as arguments to it. The parent process which is mysh, waits until the command is finished before resuming execution. Mysh can also detect Ctrl-C which sends a termination signal to the child process.

Limitations

As stated earlier, mysh lacks more advanced features found in modern shells such as pipes, I/O redirection, wildcard expansion, etc. Additionally:

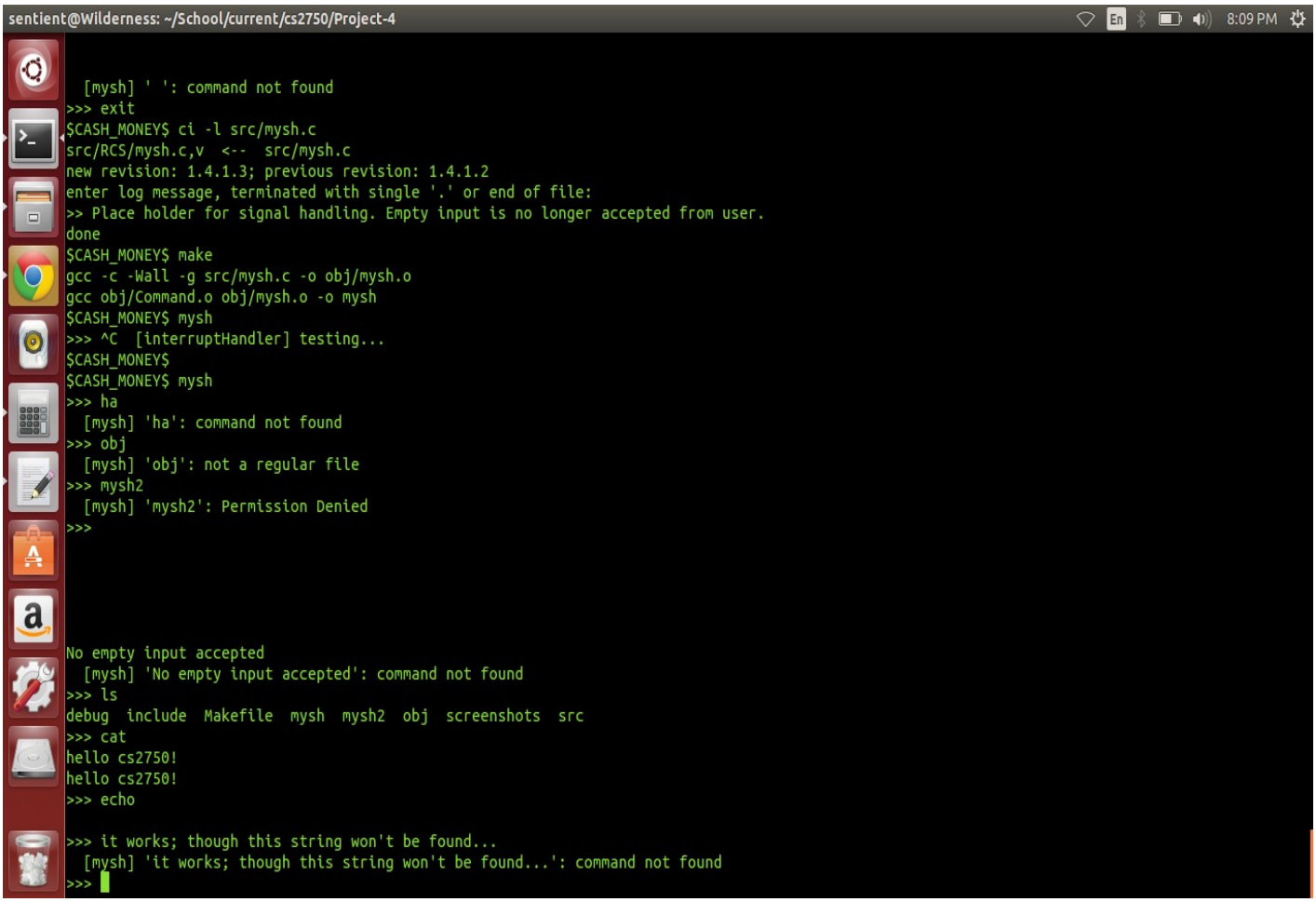
- Non-built-in commands entered are always interpreted as being files within a directory; no absolute pathnames are interpreted.

- Limit exist on length of user input read at a time (256 characters)

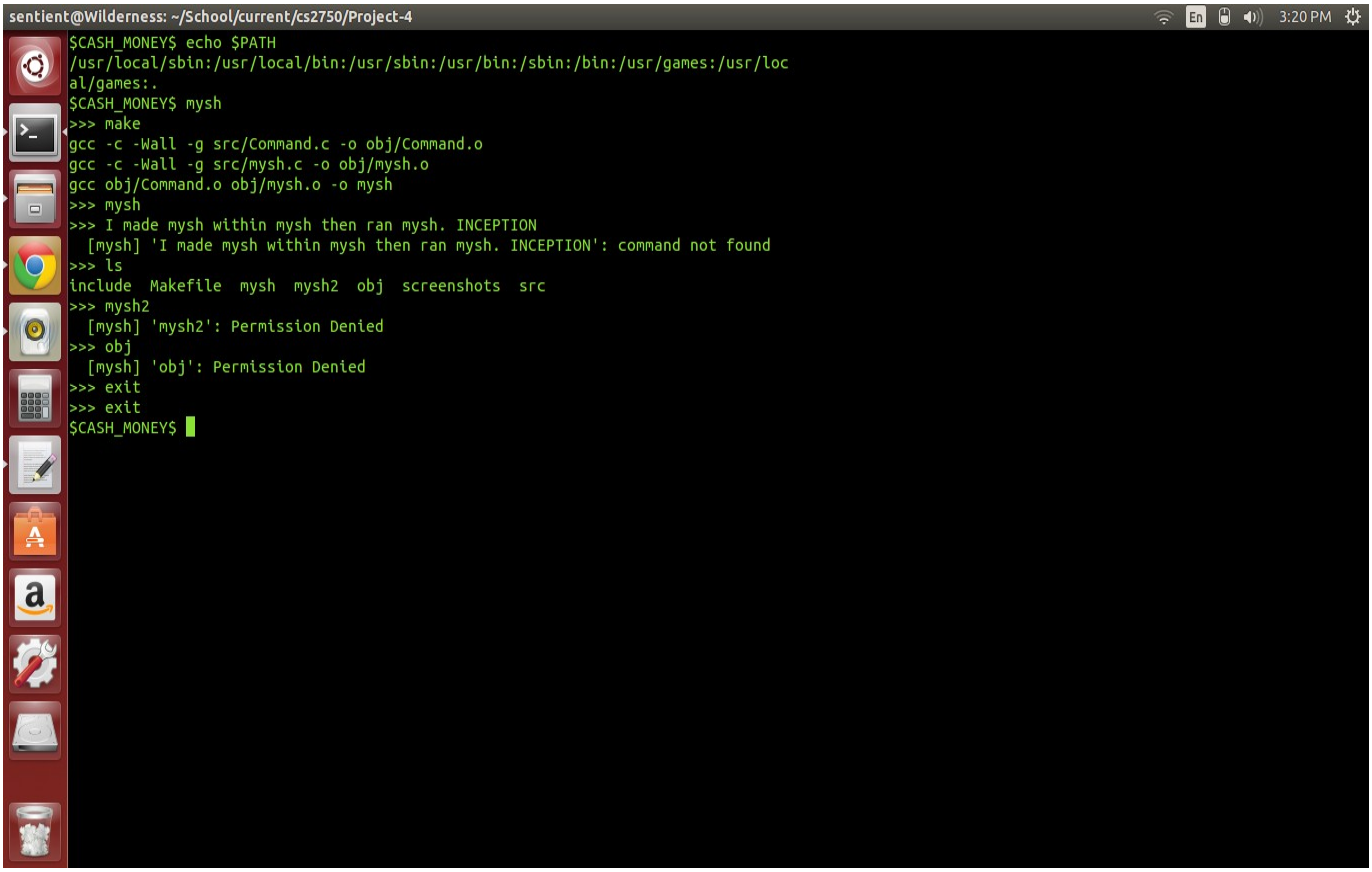
- Limit on number of words that mysh parses (16 words)

- Limit on full path length when searching for command (256) → limit of command name length (32)

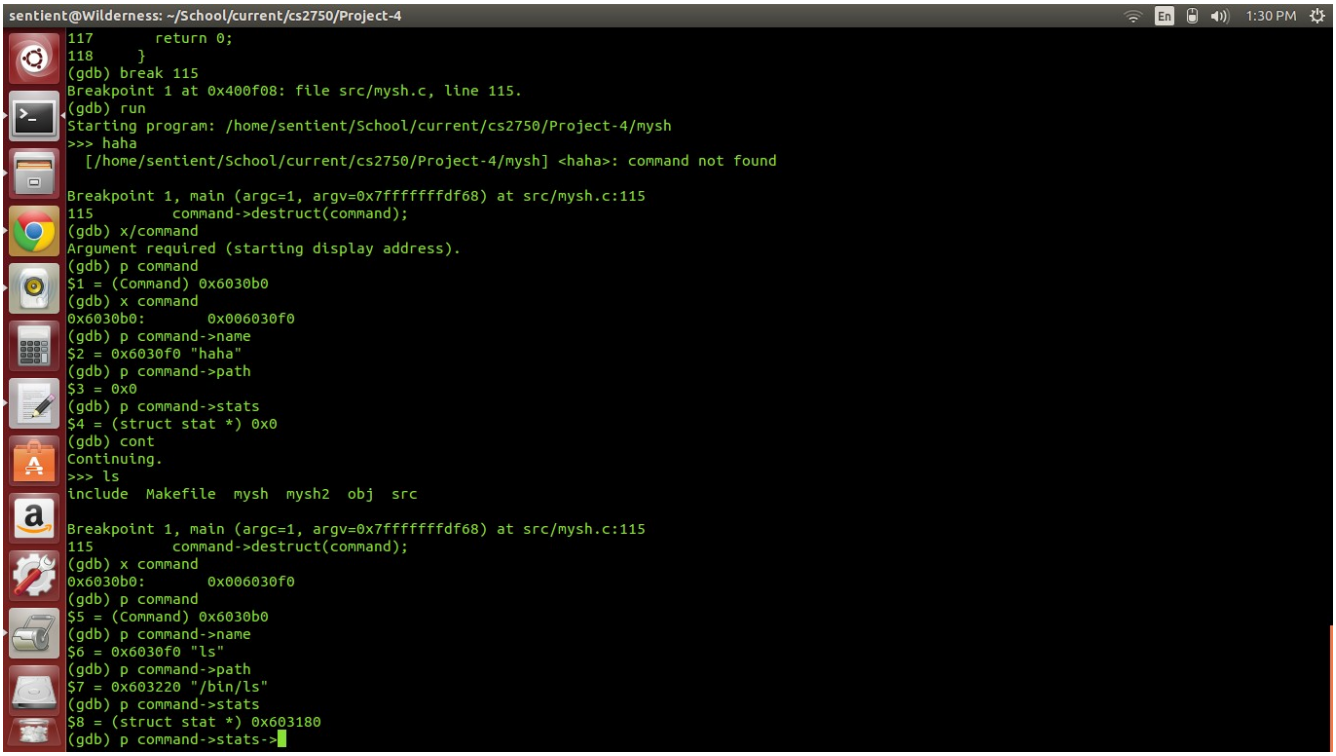
Screenshots



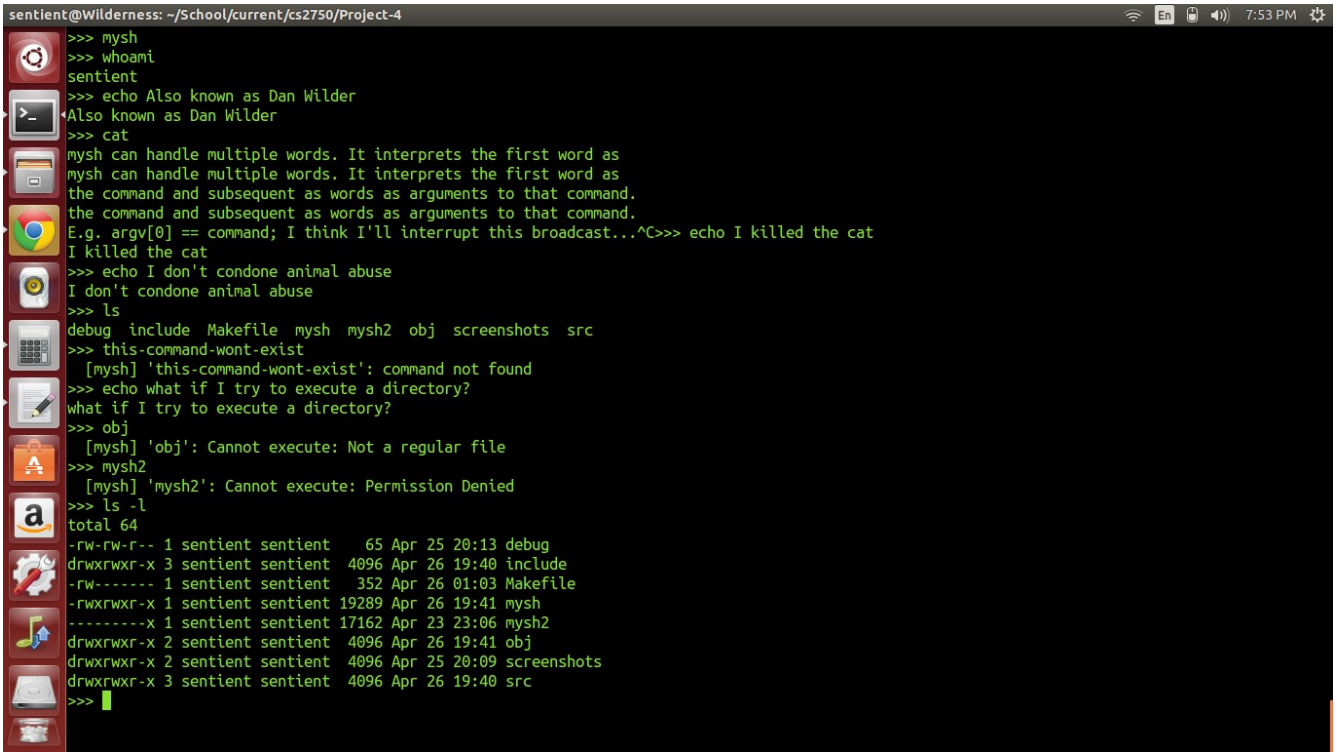
Here is an older revision of mysh which interprets an entire string of user input as a single command. Even at this stage of development, mysh is able to detect if a command exists, if it is a regular file, and if the user has permission to execute it.



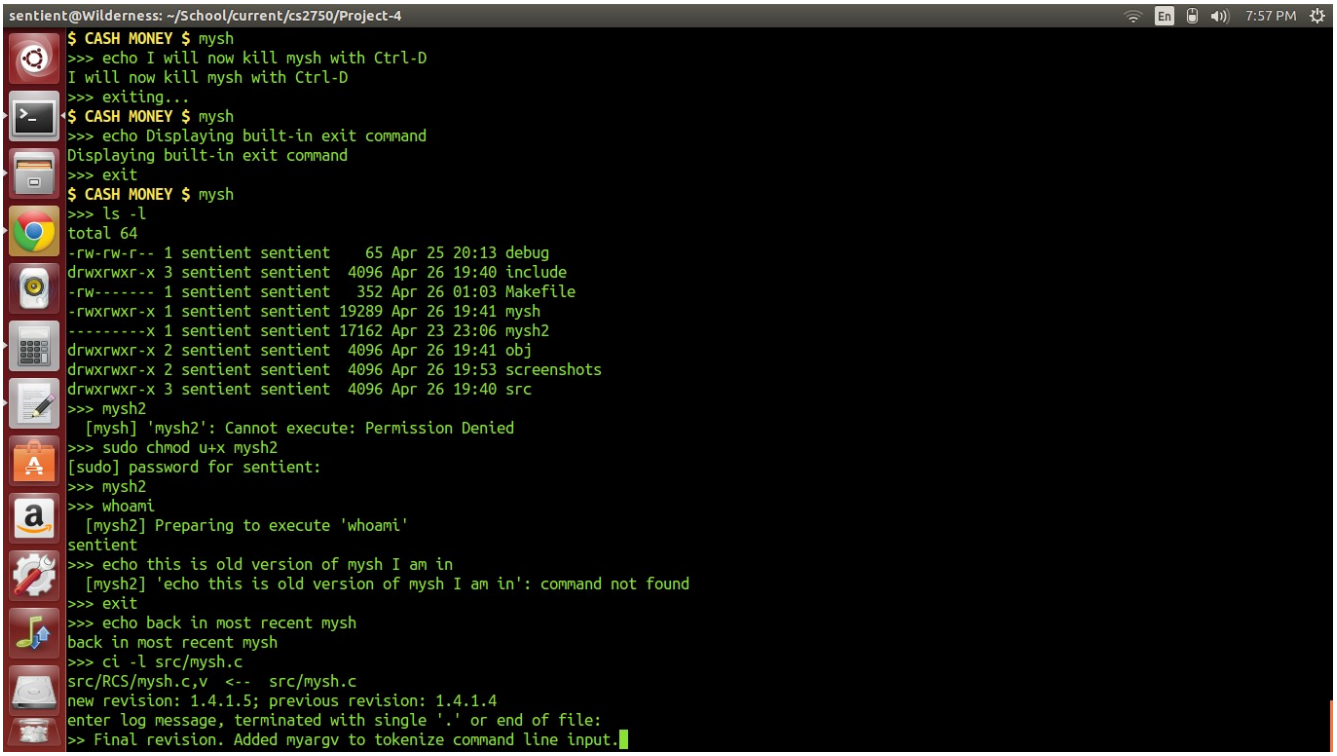
Mysh made itself then called itself!



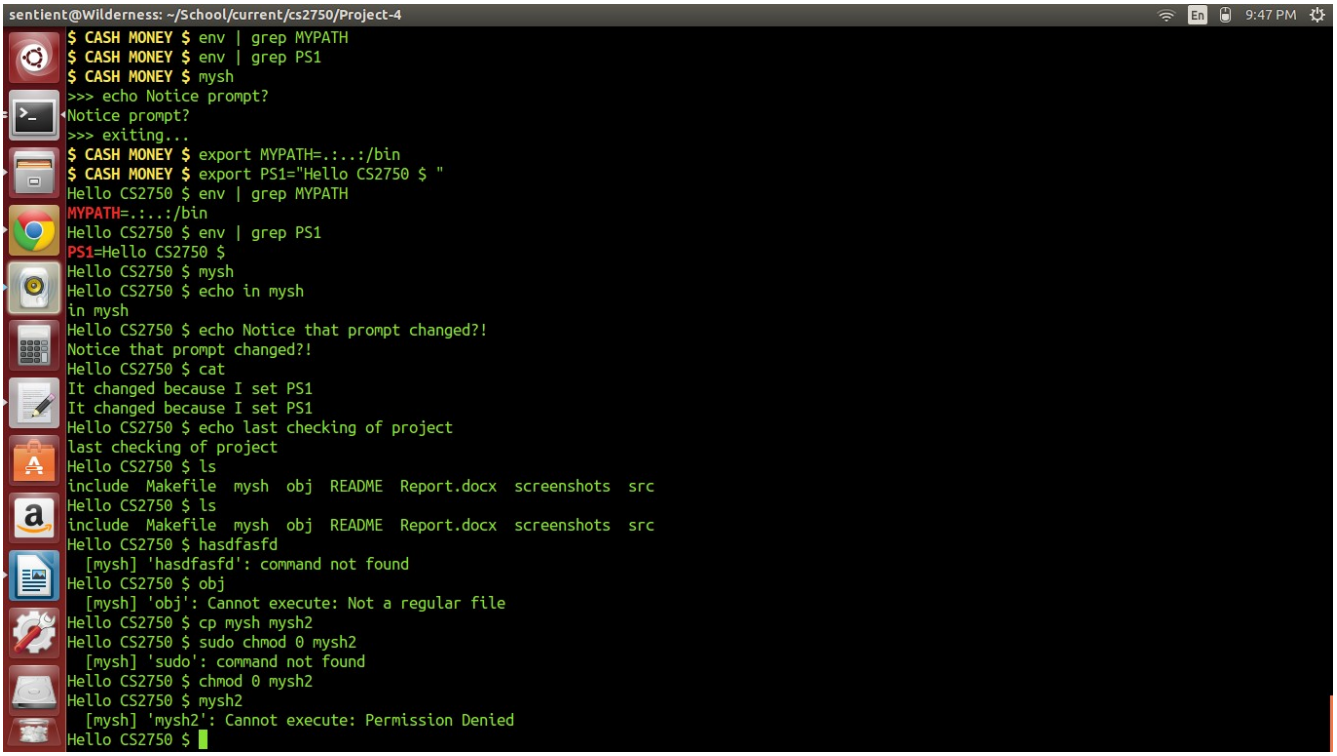
"90% of programming is debugging!"



Running of near final version



Another example



Running with MYPATH and PS1 set

Conclusion

What better way to learn about a shell than programming one? A shell's primary purpose is to execute commands. To do so, it must know what portion of user input is the command. This requires parsing. More advanced interpretation of input requires more parsing. This is a core requirement of any decent shell.

By completing this project, I learned about function pointers and a lot about memory allocation. Most of the errors that I encountered throughout the project involved improper memory usage. I ran into errors such as double free and the occasional segmentation fault. In future projects, I will take greater care in these matters.

Finally, this project encouraged me to develop a project structure before coding and maintain a consistent coding style as these will aid in readability and preemptive bug reduction.