INFO1112 Assignment 2018

Due Week 11, Monday 15th October at 12PM (Noon)

This assignment is worth 20% of your overall grade for the course.

Assessment

The assignment will be marked with a testing system that will run your program against a number of scripts written in the language. An automatic mark will be given based on percentage of tests passed (16%) and a manual mark will be given for overall style, quality, readability, etc (4%).

Marks will allocated for:

- basic command loop
- reading line continuations
- builtins
- using variables
- i/o redirection
- exec-ing external commands
- exec-ing with correct i/o redirection
- pipes
- code layout, readability, overall quality, etc

Other behaviour will not be assessed.

This assignment involves writing your own command interpreter or shell using the Python language. The specification for the language of your shell is given below.

Specification

You are to implement the following features.

Invoking the shell

Your shell program is to be written in Python3 and called `mysh.py`. The program will be invoked in the following way.

```
python3 mysh.py [filename]
```

Where `[filename]` specifies an optional filename.

When invoked **without** the filename argument, `mysh` is an interactive shell that prints a prompt, then reads a line (and any continuations, described below) and interprets it. When invoked **with** a file name argument, `mysh` reads lines from that file and interprets them. No prompts are printed if invoked with a file name.

`mysh` should terminate if it reads an end-of-file (EOF) indicating no more lines to read from a file, or Ctrl-D has been typed at the keyboard.

Reading commands

A command will consist of a series of "words" separated by whitespace (spaces+tabs). Words are defined as being any of the following.

- An identifier which starts with an alphabetic character (upper or lower case) followed by a sequence of either alphabetic or numeric characters or underscore
- A file path consisting of alphabetic or numeric or forward slash characters
- A dollar sign followed by an identifier
- A number
- One of the characters: < > |

If the input line ends in a backslash and newline, then the next line is read and joined to the previous before interpretation. This may be repeated until a line is read that ends in newline only, without a backslash.

In the following example the "**$** " part is the prompt, printed by mysh.

```
$ say cheese
cheese
$ say \
> cheese
cheese
$ say \
> blue \
> cheese
blue cheese
$ exit
Goodbye!
```

Commands

The first word of the line is interpreted as either a builtin command or a pathname of a file somewhere in the namespace.

For builtin commands, the corresponding function is called with the rest of the words on the line passed in a list.

If the command is not a builtin, then treat the command as a path to an executable file. The file is executed with the arguments set by the rest of the words on the line. This file should execute in a separate process. Any executable file should be executable in this manner (i.e. it does not need to be a mysh file).

```
$ /bin/echo hello
hello
```

If the external file does not exist or cannot be interpreted, then an error message is displayed.

```
$ /invalid/file
Unable to execute /invalid/file
```

Builtins

You are to implement the following built-in commands for mysh.

exit

Exit mysh with message: “Goodbye!”.

```
$ exit
Goodbye!
```

say [arg ...]

Echoes the argument to standard output. If no arguments are specified, a blank line is printed.

```
$ say

$ say cheese
cheese
$ say blue cheese
blue cheese
```

changedir [directoryname]

Change directory to directoryname, if specified, or to the directory given in the $HOME variable. If the HOME variable does not exist in mysh, do nothing. Before changing directory, save the current directory path to the front of a history list (see `historylist`).

showdir

Print the current working directory.

```
$ showdir
/home/alice
```

historylist

List the directories previously visited by mysh as a numbered list starting at 0. The entry numbered 0 is the current working directory.

```
$ showdir
/home/alice
$ changedir uni
$ changedir info1112
$ historylist
0: /home/alice/uni/info1112
1: /home/alice/uni
2: /home/alice
```

cdn [n]

Change directory to the numbered directory as shown in `historylist` and remove all the directories earlier in the list, leave current directory name at the beginning of the list. If n is not given, change directory to the first directory in `historylist`.

show [filename ...]

Reads from each filename in turn, writing the data to standard output. If no file name is given, it reads from standard input and writes the data to standard output until an end-of-file is read.

set [variable [value...]]

If no arguments are given, print all variables which are currently defined, in lexicographical order. If one argument is given, set the variable to an empty string. If more than one argument is given, set the variable to space-separated values. Use a python dictionary to hold variables and their values.

unset variable

Remove variable from mysh variables.

```
$ set
PS=$
$ set subject info1112
$ set name
$ set
name=
PS=$
subject=info1112
$ unset name
$ set
PS=$
subject=info1112
```

sleep N

Sleep for N seconds.

```
$ sleep 2
$ exit
Goodbye!
```

Shell variables

If a word begins with a dollar sign (`$`), then the word is replaced by a value from a dictionary of variable values maintained by `mysh`. If the variable does not exist, then it is replaced by an empty string.

Some variables may be predefined. For example, the variable "`PS`" defines the initial prompt string to be used in interactive mode. Set the value to "`$`" initially.

```
$ set
PS=$
$ set foo bar
$ say $foo
bar
$ set PS mysh>
mysh> exit
Goodbye!
```

Redirection

The characters < and > are used to indicate standard input and output, respectively. A filename must follow. For example:

```
justdoit arg1 arg 2 < myinput > myoutput
```

This will be involve executing the file "`justdoit`" with arguments "`arg1`" and "`arg2`" with standard input coming from the file "`myinput`" and standard output going to the file "`myoutput`".

```
$ say hello > sample
$ show sample
hello
$ exit
Goodbye!
```

Piping

The pipe character (`|`) is used to redirect standard output of a command to the standard input of another command.

```
justdoit arg1 arg2 | show
```

This will be involve executing the file "`justdoit`" with arguments "`arg1`" and "`arg2`". The standard output of this command is then used as standard input into the program “`show`”.

```
$ say hello | show
hello
$ exit
Goodbye!
```

Restrictions

You are limited to using the following Python modules.

- os (excluding the os.system and os.popen functions)
- sys
- time

You must not use Python features such as the "`os.system()`" function or the "`subprocess`" module to directly execute commands within the system shell. The purpose of this assignment is to emulate how the shell works, rather than write a Python interface for the command line (i.e. builtin commands should be implemented in Python; not by launching a bash subprocess).

If you have any questions regarding the scope of these restrictions, please ask on Ed.

Please note, further clarifications to this specification will be posted on Ed.

Submitting your code

An Ed assessment workspace will be available soon for you to submit your code.

Public and hidden test cases will be rolled out over the next few weeks leading to the deadline, up until the 8th October (one week before the due date). Additionally, there will be a set of unreleased test cases which will be run against your code after the due date.

Any attempt to deceive the automatic marking system will be subject to academic dishonesty proceedings.