CIS*3110 (Operating Systems)

Assignment 1: Writing a Shell

Due Date: Friday, February 12, 2016 at 23:59.

Requirements and Specifications

Objective

In this assignment you are to implement a simple UNIX shell program. A shell is simply a program that conveniently allows you to run other programs. Read up on your the *bash* shell to see what it does.

Specification

Your shell must support the functions described below. The functions are grouped into 3 sections. Successful completion of section 1 functions will result in a C+ grade. The additional successful implementation of the functions in section 2 will raise the grade to B+ and adding section 3 functions will reward the student with an A+ grade.

Notes

You must check and correctly handle all return values. This means that you need to read the **man** pages for each function to figure out what the possible return values are, what errors they indicate, and what you must do when you get that error.

There are submissions instructions on the CourseLink site. With your fully commented code you must submit a **README.txt** file that explains what functions you have implemented, how they work (any assumptions or limitations) and how they were tested. You must also submit a **Make** file.

This is an individual assignment - no code sharing is allowed. Code will be checked for integrity.

References

http://www.tldp.org/LDP/Bash-Beginners-Guide/html/Bash-Beginners-Guide.html#sect_01_01

http://tldp.org/LDP/abs/html/internalvariables.html

Section 1 Functions

1. **The internal shell command "exit" which terminates the shell.**
   - *Concepts*: shell commands, exiting the shell
   - *System calls*: exit(), kill()

2. **A command with no arguments.**
   - *Example*: ls
   - *Details*: Your shell must block until the command completes and, if the return code is abnormal, print out a message to that effect.
   - *Concepts*: Forking a child process, waiting for it to complete, synchronous execution
   - *System calls*: fork(), execvp(), exit(), wait(), waitpid()

3. **A command with arguments.**
   - *Example*: ls -l
   - *Details*: Argument 0 is the name of the command
   - *Concepts*: Command-line parameters

4. **A command, with or without arguments, executed in the background using &.**
   - *Notes*: For simplicity, assume that if present the & is always the last thing on the line.
   - *Example*: xemacs &
   - *Details*: In this case, your shell must execute the command and return immediately, not blocking until the command finishes.
   - *Concepts*: Background execution, signals, signal handlers, processes, asynchronous execution
   - *System calls*: sigset()

Notes

a) When the shell exits it should print out the following:

> exit
logout

[Process completed]
$

where the red lines are in your shell program and the blue are back in your login shell.

b) When your shell exits it should explicitly kill any child processes that are still active (*i.e.* those that are in the background).

Section 2 Functions

3. **A command, with or without arguments, whose output is redirected to a file.**
    - *Example*: ls -l > foo
    - *Details*: This takes the output of the command and put it in the named file.
    - *Concepts*: File operations, output redirection
    - *System calls*: freopen()

4. **A command, with or without arguments, whose input is redirected from a file.**
    - *Example*: sort < testfile
    - *Details*: This takes the named file as input to the command.
    - *Concepts*: Input redirection, file operations
    - *System calls*: freopen()

5. **A command, with or without arguments, whose output is piped to the input of another command.**
    - *Example*: ls -l | more
    - *Details*: This takes the output of the first command and makes it the input to the second command. You are required to implement one level of pipe.
    - *Concepts*: Pipes, synchronous operation
    - *System calls*: pipe()

Section 3 Functions

Advanced shell functionality including the following:

1. Limited shell environment variables: PATH, HISTFILE, HOME.
    - The shell allows for the definition and storage of variables and there are a number of default ones. You need to only provide 3 environment variables:
    - $PATH: contains the list of directories to be searched when commands are executed. For this exercise the default will be /bin to facilitate testing. This is not the normal default. To find the default on a UNIX bash shell, execute the following command: echo $(getconf PATH). *Hint*: execvP().
    - $HISTFILE: contains the name of the file that contains a list of all inputs to the shell. The default name of this file is ~/.bash_history. For the purposes of this assignment, the default will be the file .CIS3110_history in the directory in which the shell is initialized, *i.e.* the current working directory when the shell program is executed.
    - $HOME: contains the home directory for the user. For the purposes of this assignment, the default home directory will be the directory in which the shell is initialized, *i.e.* the current working directory when the shell program is executed.

2. Reading in the profile file on initialization of the shell and executing any commands inside.
    - By default in the bash shell, the profile file is call .bash_profile and is in the user's home directory. Please see the notes on $HOME about the directory that will be considered HOME for this assignment. Also the profile file will be named .CIS3110_profile for the purposes of this assignment.
    - Commands in the profile file typically involve the setting of environment variables.
    - Using the $PATH variable to set the path for commands and changing the PATH using the export command.
    - Using your own $PATH variable will mean that you cannot use the one designated by the shell from which you are executing your code (*i.e.* the bash shell). Therefore you must not use an exec() command that uses the bash shell $PATH.

3. Implementing the builtin functions: export, history and cd:
    - The **export** builtin command
        - The POSIX standard defines export as follows:
            The shell shall give the **export** attribute to the variables corresponding to the specified names, which shall cause them to be in the environment of subsequently executed commands. If the name of a variable is followed by *= word*, then the value of that variable shall be set to *word*.
        - You may restrict the use of export to just the environment variables PATH, HOME and HISTFILE.
        - The format of the export command is illustrated by the following examples:

export PATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:$HOME/bin
export PATH=/usr/local/sbin:$PATH

- The **history** builtin command
  - history prints out all of the input to the shell. The format of its output is illustrated in the following example:

```
1  echo $PATH
2  gcc assign1a.c -o assign1a
3  ./assign1a
4  ls /bin
5  ls /usr/bin
6  vi assign1a.c
7  gcc assign1a.c -o assign1a
8  ./assign1a
9  ls -la ~ | more
```

  - *(space) number (2 spaces) command line*
  - The following two parameters must be recognized by your shell:
    - *history -c* which clears the history file
    - *history n* which outputs only the last *n* command lines
- The **cd** builtin command
  - The cd or "*change directory*" changes the notion of which directory the command is being issued from.
  - *Hint*: chdir()

Notes

If you implement the cd command (and thus your shell is aware of the current working directory's full path name) then replace the "> " prompt in your shell with "*cwd* > " where *cwd* is the full path name of the current working directory. For example:

```
$ ./CIS3110sh
/Users/dastacey/Teaching/CIS3110> cd ..
/Users/dastacey/Teaching> cd ..
/Users/dastacey> exit
$
```