Project 3: The Shell

This project was written by instructors of CS354 at Purdue University, and is adapted from their web site.

Third part: Everything else

This third part will allow you to test your programming mettle. The third part is divided into three sections: vegetables, cake, and icing. I expect you to implement all items in the vegetable section. Some more difficult items are located in the cake section, and you can do these for additional credit as you desire, but only after you have taken care of the vegetables. The hardest items are located in the icing section, and again can be attempted for additional credit. You should complete the cake before trying the icing.

Vegetables

1. Your shell has to ignore ctrl-c. When ctrl-c is typed, a signal SIGINT is generated that kills the program. There is an example program in ctrl-c.cc that tells you how to ignore SIGINT. Check the man pages for *sigset*.
2. You will also have to implement also an internal command called *exit* that will exit the shell when you type it. Remember that the *exit* command has to be executed by the shell itself without forking another process.


```
myshell> exit
Good bye!!
csh>
```


3. You will notice that in your shell the processes that are created in the background become *zombie* processes. That is, they no longer run but wait for the parent to acknowledge them that they have finished. Try doing in your shell:

```
ls &
ls &
ls &
ls &
/bin/ps -u <your-login> | grep defunct
```

   The zombie processes appear as "defunct" in the output of the "ps -u <your-login>" command.

   To cleanup these processes you will have to setup a signal handler, like the one you used for ctrl-c, to catch the SIGCHLD signals that are sent to the parent when a child process exits. The signal handler will then call *wait3()* to cleanup the zombie child. Check the man pages for *wait3* and *sigset*. The shell should print the process ID of the child when a process in the background exits in the form "[PID] exited."

4. Implement the *cd [dir]* command. This command changes the current directory to *dir*. When *dir* is not specified, the current directory is changed to the home directory. Check "man 2 chdir".
5. When your shell uses a file as standard input your shell should not print a prompt. This is important because your shell will be graded by redirecting small scripts into your shell and comparing the output. Use the function *isatty()* to find out if the input comes from a file or from a terminal.

Cake

1. You will implement an internal command *printenv* to print the environment variables of the shell. The environment variables of a process are stored in the variable *environ*. *Environ* is a null terminated array of strings.

   ```
   char **environ;
   ```

   Check the man pages of environ.

2. Implement the builtin command *setenv A B* this command sets the environment variable *A* to be *B*. Check man pages for function setenv().
3. Implement the builtin command *unsetenv A*. This command removes environment variable *A* from the environment.
4. Extend lex to support any character in the arguments that is not a special character such as "&", ">", "<", "|" etc. Also, your shell should allow no spaces between "|", ">" etc. For example, "ls| grep a" without spaces after "ls" and before "grep" should work.
5. Allow the escape character. Any character can be part of an argument if it comes immediately after \. E.g.
   echo \"Hello between quotes\"
   "Hello between quotes"
   echo this is an ampersand \&
   this is an ampersand &
6. Allow quotes in your shell. It should be possible to pass arguments with spaces if they are sorrounded between quotes. E.g.

   ```
   ls "command.cc Makefile"

   command.cc Makefile not found
   "command.cc Makefile" is only one argument.
   ```

   Remove the quotes before inserting argument. No wild card expansion is expected inside quotes, if you get to the icing.

Icing

1. Implement wildcarding. The wildcarding will work in the same way that it works in shells like csh. The "*" character matches 0 or more nonspace characters. The "." character matches one nonspace character. The shell will expand the wildcards to the file names that match the wildcard where each matched file name will be an argument.

   ```
   echo * // Prints all the files in the current directory

   echo *.cc // Prints all the files in the current


   // director that end with cc

   echo c*.cc

   echo M*f*

   echo /tmp/* // Prints all the files in the tmp directory

   echo /*t*/*
   ```

```
echo /dev/*
```

You can try this wildcards in csh to see the results. The way you will implement wild carding is the following. First do the wild carding only in the current directory. Before you insert a new argument in the current simple command, check if the argument has wild card (* or ?). If it does, then insert the file names that match the wildcard including their absolute paths. Use *opendir* and *readdir* to get all the entries of the current directory (check the man pages). Use the functions *compile* and *advance* to find the entries that match the wildcard. Check the example file provided in regular.cc to see how to do this. Notice that the wildcards and the regular expressions used in the library are different and you will have to convert from the wildcard to the regular expression. The "*" wildcard matches 0 or more non-blank characters, except "." if it is the first character in the file name. The "?" wildcar matches one non-blank character, except "." if it is the first character in the file name. Once that wildcarding works for the current directory, make it work for absolute paths.

2. Implement environment variable expansion. When a string of the form *${var}* appears in an argument, it will be expanded to the value that corresponds to the variable *var* in the environment table. E.g.

```
setenv A hello
setenv B world
echo ${A} {$B}
Hello World
setenv C ap
setenv D le
echo I like ${C}p${D}
I like apple
```


3. Tilde expansion: When the character "~" appears by itself or before "/" it will be expanded to the home directory. If "~" appears before a word, the characters after the "~" up to the first "/" will be expanded to the home directory of the user with that login. For example:


```
ls ~           -- List the current directory
 ls ~george     -- List george's current directory
 ls ~george/dir -- List subdirectory "dir" in george's
directory
```

4. You will implement subshells of the form **`*subshell*`** (Notice that the character "`" used here is a **back-tick.** This character is usually in the same key as "~".). A subshell may appear anywhere in a command and when executed the output of the subshell will become input of the shell itself. For example, the following command will copy all files that end with "a" to the directory "mydir".

```
ls *a > file-list
 mkdir mydir
 cp `cat file-list` mydir
```

Deadline

The deadline of this part of the project is Wednesday, April 23rd, before class,

Add a README file to the lab3-src/ directory with the following:

1. Features specified in parts 1, 2 and 3 (vegetables) that work
2. Features specified in parts 1, 2 and 3 (vegetables) that do not work

3. Extra features implemented from the cake section of part 3
4. Extra features implemented from the icing section of part 3

Make sure that your shell can be built by typing "make".
Follow these instructions to turnin your project

1. Login to CSSUN.

2. cd to lab3-src and type "make clean"

3. Type "make" to make sure that your shell is build correctly.

4. Type "make clean" again.

5. cd one directory above lab3-src

6. Create a tar file named <user_name>.tar, where <user_name> is your CSSUN login, by typing

   tar -cf <user_name>.tar lab3-src

7. Gzip the tar file by typing

   gzip <user_name>.tar

8. Since this timestamp will be used to verify whether the work was completed on time or not, you should set the permissions on the file you submitted to make sure that the file timestamp is not changed. So this by typing:

   chmod a-w <user_name>.tar.gz

9. Mail the gzipped tar file to clay at cs dot georgetown dot edu as an attachment.