Assignment 3

Writing a Basic Command Shell

Author: Brian Crites

This project must be done in a group of two

Coding Requirements

The Test Command

For this assignment you will add the test command to your rshell, as well as its symbolic equivalent [ ]. The square brackets [ ] are actually set up as the test command in bash, you can read about it here and try it out on your own. This command returns 0 (TRUE) if the test succeeds and 1 (FALSE) if the test fails. This command is very useful for writing conditions that can be combined with && and || to write more complex bash command structures.

IMPORTANT: You cannot use execvp to run the test command as you did in assignment 1. You must write the functionality yourself.

Your subset of the test command should allow the user to run the command using the keyword test

```
$ test -e test/file/path
```

Additionally, your rshell should allow the user to use the symbolic version of the command

```
$ [ -e test/file/path ]
```

You command should also allow the user to run tests using the following flags

`-e` checks if the file/directory exists
`-f` checks if the file/directory exists and is a regular file
`-d` checks if the file/directory exists and is a directory

If a user does not specify a flag, then the -e functionality will be used by default.

You will also add an extra feature that the test command currently does not have. Your test command will print out to the terminal if it evaluated to True or False

If the command test -e /test/file/path evaluates to True, then print display the following

```
(True)
```

And likewise, if the above command evaluates to False, then print False in the same manner

```
(False)
```

Additionally, your test command should work with the connectors && and || that you have written previously in assignment 2, as well as all other functionality from assignment 2.

```
$ test -e /test/file/path && echo "path exists"
```

- or -

```
$ [ -e /test/file/path ] && echo "path exists"
```

This will check if path exists at /test/file/path, and if path does exist will print "path exists".

When your input requires you to have multiple outputs, simply print the (True) or (False) labels as they are evaluated, so the above command (assuming the path exists) would print

```
(True)
path exists
```

Your test function should be designed to work with both full directory paths and relative directory paths. In order to create this command, you will need to use the stat() command, which you can read about here. You will need to use the stat() function along with the S_ISDIR and S_ISREG macros in order to implement all the flags required for this program.

The Precedence Operators

Additionally, you will implement parentheses ( ) as precedence operators in your rshell. The parentheses ( ) operators are used to change the precedence of the returns of commands, connectors, and chains of connectors. For example

```
$ echo A && echo B || echo C && echo D
```

Would print the following

```
A
B
D
```

However, we can add parentheses to change the precedence of the connectors

```
$ (echo A && echo B) || (echo C && echo D)
```

which would print

```
A
B
```

This is because the parentheses change the precedence, so that when (echo A && echo B) returns true (since it execute correctly) to the || operator then it will not run the entire group (echo C && echo D) since the parentheses have now grouped these commands together. Note that if for some reason echo A failed, echo B would not execute, and (echo C && echo D) would be executed.

If you are unsure about how the parentheses will change how a set of commands runs, try out some examples in your terminal.

Submission Instructions

Add this new code to your rshell project from homework 1. Create a branch called test. Do all of your work under this branch. When finished, merge the test branch into the master branch, and create a tag called hw3. Remember that tags and branches in git are case sensitive!

NOTE: git push will not automatically push tags to your repository. Use git push origin hw3 to update your repository to include the hw3 tag.

To download and grade your homework, the TA will run the following commands from the hammer server:

```
$ git clone  https://github.com/yourusername/rshell.git
$ cd rshell
$ git checkout hw3
$ make
$ bin/rshell
```

You should ssh into hammer.cs.ucr.edu and run the above commands to verify that you've submitted your code successfully. If you forget how to use git, two students from previous cs100 courses (Rashid Goshtasbi and Kyler Rynear) made video tutorials on the git commands needed to submit your assignments via Github.

Do not wait to upload your assignment to Github until the project due date. You should be committing and uploading your assignment continuously. If you wait until the last day and can't figure out how to use git properly, then you will get a zero on the assignment. NO EXCEPTIONS.

You will also need to create a file for submitting your partner and github information. Create a text file using vim with the following information: you and your partner's name, you and your partner's net IDs, and the github url of your assignment's repository.

Follow this format EXACTLY:

```
name1=
ucrnetid1=
name2=
ucrnetid2=
repourl=
```

Save the file as hw3 (**WITH NO EXTENSION**) and submit it to iLearn's to Assignment 2 submission link.

Here's an example file:

```
name1=Busra Celikkaya
ucrnetid1=bceli001
name2=Amirali Darvishzadeh
ucrnetid2=adar001
repourl=https://www.github.com/busrac/rshell.git
```

Your repository should be public, however if you prefer to have a private repository please email the instructor for additional information on adding collaborators.

Project Structure

You must have a directory called src which contains all the source code files for the project.

You must have a Makefile in the root directory. In the Makefile you will have two targets. The first target is called `all` and the second target is called `rshell`. Both of these targets will compile your program using `g++` with the flags: `-Wall -Werror -ansi -pedantic`.

You must NOT have a directory called bin in the project; however, when the project is built, this directory must be created and all executable files placed here.

You may also optionally have a .gitignore file to automatically stop generated files from being uploaded to your remote repository.

You must have a LICENSE file in your project. You may select any open source license. I recommend either GPL or BSD3.

You must have a README.md file. This file should briefly summarize your project. In particular, it must include a list of known bugs. If you do not have any known bugs, then you probably have not sufficiently tested your code! Read this short intro to writing README files to help you. You must use the Markdown formatting language when writing your README.

You must have a directory called `tests`. The directory should contain a bash script that fully tests each segment of the program mentioned in the rubric. This means that for a completed project, you should have the following files (with these exact names):

```
test_test.sh              #tests for the test command
precedence_test.sh        #tests for precedence operators
commented_command.sh      #tests commands with comments
exit.sh                   #tests exit and commands with exit
```

Each of these files should contain multiple commands in order to fully test each functionality. Proper testing is the most important part of developing software. It is not enough to simply show that your program works in some cases. You must show that it works in every possible edge case

Coding Conventions

Your code must not generate any warnings on compilation.

You must follow the CalTech coding guidelines, as stated in the syllabus.

Your final executable must have no memory leaks.

Testing

Proper testing is the most important part of developing software. It therefore will have a very large impact on your grade.

It is not enough to simply show that your program works in some cases. You must show that it works in every possible edge case. **That means you must try to write test cases that will break your program!**

If you are unsure if your test cases are sufficient, ask one of the instructors to review them before the deadline.

Collaboration Policy

You MAY NOT look at the source code of any other student.

You MAY discuss with other students in general terms how to use the unix functions.

You are ENCOURAGED to talk with other students about test cases. You are allowed to freely share ideas in this regard.

You are ENCOURAGED to look at bash's source code for inspiration.

Grading

Rubric

5 points for well commented code
5 points for following style guidelines
20 points for sufficient test cases
40 points for test operator (including integration with precedence and past requirements)
30 points for precedence operators (including integration with test and past requirements)

IMPORTANT: Your project structure is not explicitly listed in the grading schedule above, however not following the correct structure will result in a 20 point deduction.

IMPORTANT: Projects that do not correctly compile as described above will receive no points.