Assignment 4

Adding Redirection to your Shell

Author: Mike Izbicki

This project must be done in a group of two

Github Setup

The first part of assignment to is to update your github 'Projects'. You will add a new project, call it 'Assn 3 -- Test Commands' with the same 3 columns, 'Backlog', 'In Progress', and 'To Deploy'. Now, after reading the assignment specifications, create manageable sized **user stories**. Each of these user stories should be logged as an **issue** on github, labeled as an **enhancement.** They should start in the 'Backlog' column. You then want to assign the first 2 or 3 user stories to each of the group members. *Note: you can assign issues to contributors*. The first issue each member decides to work on should be moved to the 'In Progress' column. Each user will want to track their work through the 'Projects' Kanban board and resolve the issue when they complete it. After completing all their assigned user stories they can assign themselves new ones and begin working on them.

Coding Requirements

Extend your `rshell` program so that it properly handles input redirection `<`, output redirection `>` and `>>`, and piping `|`. This will require using the Unix functions `dup` and `pipe`. You can find help on how to use these functions in the man pages.

As an example, after this assignment, your program should be able to successfully handle the following command:

```
$ cat < existingInputFile | tr A-Z a-z | tee newOutputFile1 | tr a-z
A-Z > newOutputFile2
```

IMPORTANT: This is a necessary but not sufficient test case. You must come up with others on your own.

Bash has an extensive syntax for redirection, and you are not required to implement all of it. But if you're curious, see the linux documentation project's bash io-redirection tutorial for details.

Submission Instructions

Create a branch called inputRedirection off your main project repo. Do all of your work under this branch. When finished, merge the exec branch into the master branch, and create a tag called assn4

NOTE: git push will not automatically push tags to your repository. Use git push origin hw3 to update your repository to include the assn4 tag.

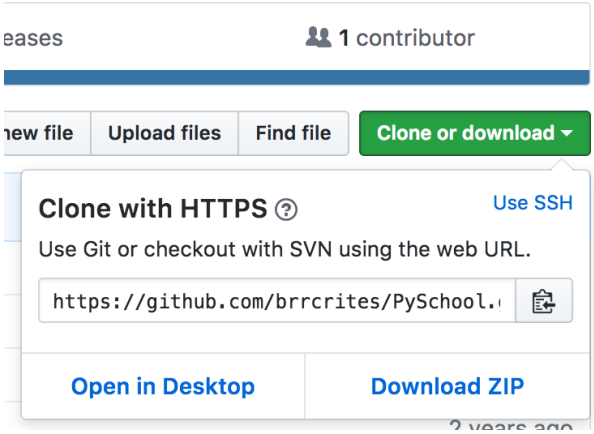
To download and grade your homework, the TA will run the following commands from the hammer server:

```
$ git clone  https://github.com/yourusername/rshell.git
$ cd rshell
$ git checkout assn4
$ make
$ bin/rshell
```

IMPORTANT: You should verify these commands work for your repository on hammer.cs.ucr.edu 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 JSON file with the following information: you and your partner's name, email, and id as well as the github url of your assignment's repository. Save the file as **assn4.json** and submit it to iLearn's to Assignment 4 submission link.



Make sure you are using the HTTPS GitHub link (shown above), not the SSH link when creating your JSON submission file. This link is accessible from the front page of your GitHub repository.

Here's an example file:

```
{
      "github_url": "https://www.github.com/busrac/rshell.git",
      "partners": [
            {
                  "name": "Busra Celikkaya",
                  "email": "bceli001@ucr.edu",
                  "id": 861000000
            },
            {
                  "name": "Lam Le",
                  "email": "lle018@ucr.edu",
                  "id": 861000000
            }
      ]
}
```

You **MUST** validate that your JSON is correct using a JSON linter (like this one) before you submit it. Invalid JSON will result in a zero for this assignment. 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):

```
input_redirect_test.sh                  #tests for <
output_redirect_test.sh                 #tests for >
append_redirect_test.sh                 #tests for >>
pipe_test.sh                            #tests for |
```

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

Again, the tests you choose will be the most important part of your grade.

You should carefully consider: which redirections can be legally combined together, and which cannot? Does order matter? Also make sure to test that you are parsing the command correctly.

IMPORTANT: 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

Points	Description
5	Code comments
5	Following style guidelines
5	Github management
20	Sufficient test cases
20	Input redirection (<)
20	Output redirection (> and >>)
25	Piping
100	**Total**