SIENAcollege
Computer Science

Computer Science 330
Operating Systems
Siena College
Fall 2022

Programming Project 4: Roger Bacon Shell with Pipes and Redirection

Due: 4:00 PM, Wednesday, December 7, 2022

Parsing Complete by: 4:00 PM, Wednesday, November 30, 2022

In this programming project, you will build on your basic shell project to include pipes and input and output redirection.

Getting Set Up

We will use the same repository as you did for the basic shell, but will create Git branches for your work for the previous project and for this one.

When your repository was initially created, it consisted of a single branch named `main`. Most likely, this is still the only branch in your repository. You can see this for sure by issuing the command

```
git branch
```

in a clone of your repository or by looking in the branch dropdown menu on the main GitHub page for your repository, That dropdown is just under the "Code" tab on the left side of the window.

The following steps will create a branch called `basic` where you will keep a your final version of the basic shell functionality, and a branch called `pipesio` where you will develop your code for this programming project's functionality.

- In a clone of your repository, create your `basic` branch with the command

  ```
  git branch basic
  ```

- Switch to be using ("check out") the new branch with the command

  ```
  git checkout basic
  ```

  You should be able to see that your clone is using the `basic` branch by issuing the command

  ```
  git branch
  ```

and noticing that the `basic` branch is selected.

- Edit your `rbsh.c` file to include a note in the banner comment at the top, indicating that this is the final version for the basic functionality. Commit this change.
- To get the branch to exist also on GitHub, you will need to do a `git push`, but this will not work because your branch does not exist on GitHub. Fortunately, when you try a `git push`, git will give you the command you need to get the branch tracked at the origin repository on GitHub. Issue that command, which will set the "upstream" for this branch to be the repository on GitHub. Then verify that your `basic` branch is now shown in the branch dropdown menu on GitHub. Also make sure you can see the change you made in the previous step in `rbsh.c` when you select the `basic` branch on GitHub.
- Back in your clone, switch your working copy back to the `main` branch:

  ```
  git checkout main
  ```

  Notice that the `rbsh.c` file in your clone no longer has the comment you added in the `basic` branch. You could see that version again by using `git checkout basic`. If you try that, be sure to switch back to your `main` branch before continuing.
- Now, create your branch called `pipesio` for this programming project's functionality, check out that branch, add a comment in `rbsh.c` that this is the version where you will be implementing this programming project's functionality, commit and push (you will need to do the special `push` command to set the upstream for this new branch).

Do your work for this programming project in the `pipesio` branch.

Pipes and I/O Redirection

For this version of the Roger Bacon Shell, you should add the following functionality.

- *Input and output redirection* should be implemented.

  For example,

  ```
  shell# cat < cat.c > myfile.c
  ```

  should cause the `cat` program to read from `cat.c` and write to the file `myfile.c`.

  ```
  shell# ls -l >> dirlistings.txt
  ```

  should cause the output of `ls -l` to be appended to the end of the existing file `dirlistings.txt`.

  An individual command may only redirect input once and output once, but those redirections may be specified in any order.

```
shell# > alines grep -i < Makefile a
```

should be interpreted the same as the more usual

```
shell# grep -i a < Makefile > alines
```

- *Pipes* should be implemented.
  - For example,

    ```
    shell# cat cat.c | wc > count.txt
    ```

    should cause the output of `cat cat.c` to be the input of `wc > count.txt`.
  - You should allow a sequence of pipes to be specified:

    ```
    shell# ls -l *.c | grep "Oct 31" | wc -l
    ```

    The program should be able to handle a sequence of pipes of any length.
  - Only the first command in a pipeline may have input redirection. Only the last command in a pipeline may have output redirection. Redirection of other commands should be reported as an ambiguous command line.
- Typing <ctrl-c> should abort a command being run, but not cause `rbsh` to terminate.
- A list of commands separated by semicolons should be executed in sequence. Each `;`-separated part of your command line can itself be a pipeline and can have I/O redirection.

As was the case with the basic shell version, a significant challenge here is to parse the command line into appropriate data structures so that it is convenient to create the child process(es) for each pipeline and perform I/O redirection when appropriate.

The definitions of the structures used by the reference solution are available linked from the HTML version of this document. You may wish to use something similar.

Once you have your command line parsed and you are ready to launch the processes in a command line, it is most straightforward to create the processes from the last in the pipeline to the first. For each process created by `fork` to run a program in a pipeline, the I/O redirection and pipe connections would follow this kind of pattern:

```
if input redirect:
   open input file
   dup2 to replace stdin with opened fd

if output redirect:
   open output file
   dup2 to replace stdout with opened fd
```

```
if not first in pipeline:
   dup2 to replace stdin with read end of incoming pipe
   close write end of incoming pipe

if not last in pipeline:
   dup2 to replace stdout with write end of outgoing pipe
   close read end of outgoing pipe

execvp!

Parent: close both ends of all pipes, waitpid for all children
```

Be sure that any "end" of any pipe that was not connected to the input or output of some process is closed in all processes that have access to it!

Executables, both standard and with debugging outputs turned on, for the reference solution are in `/home/cs330/pipeshell` on noreaster. (Note: some of the debugging output could be confusing, referring to foreground and background processes, and a process table – these are because this version also has some of the support for background processes still in the code.)

Submission

Commit and push!

For this programming project, an in-person demonstration and code review is required for each group following submission. This can occur during office hours or by appointment.

Grading

This assignment will be graded out of 80 points.

Feature	Value	Score
Git branch setup	5	
Documentation	10	
Verbose mode output	5	
Running a command with input/output redirection	15	
Launching a sequence of jobs separated by semicolons	10	
Correct trapping of Ctrl-C keystroke	5	
Pipeline of 2 processes	15	
Arbitrarily long pipeline of processes	10	
In-person demo and code review	5	
Total	80	