Assignment 2

Writing a Basic Command Shell

Author: Mike Izbicki

This project must be done in a group of two

Coding Requirements

Write a command shell called rshell in C++. Your shell will perform the following steps:

1. Print a command prompt (e.g. `$`)
2. Read in a command on one line. Commands will have the form:

cmd = executable [argumentList] [connector cmd]
connector = || or && or ;

where executable is an executable program in the PATH and argumentList is a list of zero or more arguments separated by spaces. The connector is an optional way you can run multiple commands at once. If a command is followed by ;, then the next command is always executed; if a command is followed by &&, then the next command is executed only if the first one succeeds; if a command is followed by ||, then the next command is executed only if the first one fails. For example:

```
$ ls -a
$ echo hello
$ mkdir test
```

is equivalent to:

```
$ ls -a; echo hello; mkdir test
```

There should be no limit to the number of commands that can be chained together using these operators, and your program must be able to handle any combination of operators. For example, you should be able to handle the command:

```
$ ls -a; echo hello && mkdir test || echo world; git status
```

1. Execute the command. This will require using the syscalls fork, execvp, and waitpid. Previous cs100 students created two video tutorials (a fun cartoon tutorial ; more serious explanation). You should also refer to the man pages for detailed instructions.
2. You must have a special built-in command of exit which exits your shell.

3. Anything that appears after a # character should be considered a comment. For example, in the command ls -lR /, you would execute the program /bin/ls passing into it the parameters -lR and /. But in the command ls # -lR /, you would execute /bin/ls, but you would not pass any parameters because they appear in the comment section. You should also note that the # may or may not be followed by a space before the comment begins

IMPORTANT: Most bash commands are actually executables located in /bin, /usr/bin/ (e.g. ls). But some commands are built-in to bash (e.g. cd). So while the ls command should "just work" in your shell, the cd command won't.

HINT: Pay careful attention to how you parse the command string the user enters. There are many ways to mess this up and introduce bugs into your program. You will be adding more parsing features in future assignments, so it will make your life much easier if you do it right the first time! I recommend using either the strtok function from the C standard libraries or the Tokenizer class provided in the boost library. Students often don't do this section of the assignment well and end up having to redo all of this assignment in order to complete the future assignments.

Submission Instructions

Create a new project on Github called rshell. Create a branch called exec. Do all of your work under this branch. When finished, merge the exec branch into the master branch, and create a tag called hw2. 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 hw2 to update your repository to include the hw2 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 hw2
$ make
$ bin/rshell
```

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 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 hw2 (**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 files for the project, additionally you may either have a folder `header` which contains your header files or you may keep them in the `src` directory.

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 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):

```
single_command.sh          #tests single commands
multi_command.sh                 #tests commands with ;, &&, or
||
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

IMPORTANT: The file/directory names above are a standard convention. You must use the exact same names in your project, including capitalization.

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.

Every time you run a syscall, you must check for an error condition. If an error occurs, then call perror. For every syscall you use that is not error checked, you will receive an automatic -5 points. For examples on when, how, and why to use perror, see this video tutorial.

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
20 points for executing single commands
20 points for executing commands with ;, &&, and ||
20 points for executing commands with comments
10 points for exit commands

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.

Extra Credit

Many shells display extra information in the prompt besides just a simple `$`. For example, it is common to display the currently logged in user, and the hostname of the machine the user is logged into. If your username is `jdoe001`, and if you're logged into the machine `alpha023`, then your prompt would look like:

```
jdoe001@alpha023$
```

You can get up to 10 points of extra credit if your prompt prints this extra information. You will need to lookup the man pages for the functions getlogin and gethostname. You must not hard code the username or hostname!