EE 107S: Assignment 2

Due: Thursday, September 21, 2017 at 11:59 PM

You may work on this assignment with up to one partner.

Introduction

In lecture and lab section we went over some of the text processing commands available on Linux. We went through some basic examples, as well as some more advanced examples. However, many of the concepts and commands covered are very expansive. The goal of this assignment is to give you a better grasp of the syntax of these commands.

Background: Bash scripts

So far we have been referring to the command line as the terminal. However, a terminal really just refers to the window in which you enter commands. The terminal window contains a shell inside it, which is really what the command line is. Bash is a shell that launches inside the terminal window by default on Ubuntu 16.04 LTS. Other shells include `zsh` and `tcsh`, and these can be set to run in the terminal window by default instead. The main difference between shells is the syntax for writing shell scripts.

Shell scripts contain a sequence of statements to execute. Shell scripts are similar to programs written in Python, Ruby, Perl, etc. We will discuss some of the programming constructs in Bash scripts (i.e. shell scripts written for the Bash shell) in a later lecture. For this assignment, all you need to know is that you can specify a list of commands in a shell script that will be executed in sequential order.

Given below is a short Bash script that lists out the files and directories in the current directory.

```
1 #!/bin/bash
2
3 # Lines preceded by # are ignored (except for the #! line)
4
5 echo 'Writing file names to files.txt'
6 ls -l | tr -s ' ' | cut -d ' ' -f 9 > files_temp.txt
7 tail -n +2 files_temp.txt > files.txt
8 rm files_temp.txt
```

The following is a more detailed explanation of the Bash script.

- `#!/bin/bash` is a special line that all Bash scripts will start with. It indicates to the shell that the file should be treated like a Bash script.
- Line 3 contains a comment that is completely ignored when the script is run.

- Line 5 echoes a message to the user. `echo` is very useful in Bash scripts to give users information on what the script is doing or if there have been any errors.
- Line 6 performs a directory listing (`ls`) and then extracts the 9th column. The 9th column contains the names of the files and directories. The output of this entire pipeline of commands is written to `files_temp.txt`.
- Line 7 removes the first line from `files_temp.txt`, because it ends up being an empty line, and outputs the rest of the lines to `files.txt`. Note that we haven't seen the `+2` notation before, but it is explained in the `man` pages.
- Line 8 cleans up the temporary file.

This Bash script can be written as a single pipeline of commands, but this form exemplifies a powerful way of using Bash scripts. In a Bash script you can run multiple unrelated commands and output the result of each into a separate file. Then you can run a sequence of commands that assembles all the temporary files into a single final output. Finally, you can clean up all the intermediate files.

Running a Bash script

You may write a Bash script using the visual text editor in Ubuntu 16.04.1, `gedit`. Once you have written the file, you must make it executable. You only need to make a file executable one time. To make the file `shell.sh` executable, you can run the command `chmod +x shell.sh` (we will discuss this in a later lecture).

Now that the file is marked as executable, it is ready to be run. To run the file `shell.sh`, you can type in `./shell.sh` and then press Enter. The `./` means to look for the `shell.sh` in the current directory. The reason you must explicitly state the directory of the script will be explained in the future.

Assignment

Your job is to write a Bash script that extracts the links from a webpage as shown in the following sections.

How to use the starter files

I have included several starter files. You can look at the *.html files to see examples of webpages. The HTML pages are given in increasing order of difficulty, with `1-simple.html` containing two simple links and `5-wiki.html` containing many links, some of which are present on the same line. For each HTML file, there is a corresponding txt file that contains the expected output.

Additionally, there is a small Bash script template provided, `as2.sh`. As given, the Bash script runs `grep` on `$1` to search for the regular expression `href`. `$1` is a special symbol in Bash scripts that says to use the first argument given to the command. As such, you may run the Bash script as `./as2.sh 1-simple.html`, or on any of the other HTML files. The Bash script should be marked

as executable already, but if you are getting an error that says "Permission denied", you should run the command `chmod +x as2.sh`.

Links in an HTML file

HTML is what is known as a markup language. There are a series of "tags", where each line of text can be surrounded by one or more tags. A simple tag is `<b>`, which indicates everything afterward should be in boldface. To boldface the word "Hello!", the HTML would look like `<b>Hello!</b>`. Note that the boldface tag is ended with a similar tag that is preceded by a `/`.

The semantics of HTML are beyond the scope of this assignment, other than your ability to recognize the specific tag for links. Links in HTML are given as follows:
`<a href="http://google.com"> Search engine</a>` The text that the user sees is surrounded by tags. The target of the link is given in the `href` field of the tag. Your job will be to extract the `href` fields from all the links in the HTML. **Note:** The tags can have fields other than `href`. For example, `<a class="link" href="http://google.com">Search engine</a>` is a valid link.

Expected behavior

For each link present in an HTML file, your Bash script should output the target on its own line. Only the contents inside the quotes of the `href` field should be shown in the output. The `1-simple.html` file is reproduced below:

```
<html>
    <head></head>
    <body>
        <a href="http://google.com">The first link!</a>
     <a href="http://www.utexas.edu/" class="big">The second link!</a>
    </body>
</html>
```

The expected output for this file is:

```
http://google.com
http://www.utexas.edu/
```

There is a command called `diff` that can help you ensure your Bash script has the correct output. `diff` shows a line by line difference between two files. You can use `diff` as follows:

```
./as2.sh 1-simple.html > 1-simple-mine.txt
diff 1-simple.txt 1-simple-mine.txt
```

If the files are the same, `diff` will not have any output.

Submission

All you should submit is the single `as2.sh` script to Canvas (hopefully it's up before the due date...) without renaming the file. In `as2.sh`, you should list you and your partner's names. If you are working alone, delete the line that says "Name 2:". If you are working with a partner, only one of you should make the submission. **Failure to follow these directions exactly may result in a 0 on this assignment.**

Important notes

- You can view the files using the GUI of your virtual machine. If you right-click the files in the file browser, there should be an option to Edit with gedit. `gedit` is the default text editor in Ubuntu.
- Your Bash script should output the final result to `stdout`.
- If you choose to use `awk`, you should put the Awk program **in single quotations**. Using double quotations will confuse Bash into treating `$1` as the first argument to the Bash script as opposed to the first column of a line, as `awk` expects. Aren't conflicting conventions just great...
- You should leverage writing temporary files in the Bash script. You should also clean up all temporary files using the `rm` command at the end of the Bash script.
- I was able to come up with a solution that used a single `grep` piped into a single (fairly simple) `awk`. It may be difficult to arrive at the solution without much experience, but don't overthink it too much! Also, you are free to use any number of any command available with the default installation of Ubuntu 16.04.1 LTS. If you want to use a single `awk` command, be my guest.
- You may find the Awk functions `match` and `split` and the Awk variables `RSTART` and `RLENGTH` useful.
- Google and `man` are your friends.