CSE 337: Scripting Languages (Fall 2019)

Minor Homework #3

Assignment Due: Saturday, November 23, 2019, by 11:59 PM

Important Assignment Notes

- When writing programs, you **MUST** use the techniques that are described in the lectures (or the lecture slides). You may **NOT** use methods, modules, packages that were not covered in this course.
- Your Perl scripts must use regular expressions to do perform matching and substitution against strings.
- Please put the pragma statements **`use strict;`** and **`use warnings;`** at the start of all your Perl programs.
- The questions for this assignment assume a Unix environment. Your code **MUST** run on the `allv` Linux machines.

Getting Started

This assignment requires you to write several Perl scripts to perform various tasks. Each problem provides a specific name for its script. **Each of your scripts MUST use the filename specified by its problem.** The automated grading system will be looking for scripts with those exact names; if you change one or more of those names, the grading program not be able to locate and grade those script(s), and you will receive a zero for that problem.

Directions

Solve each of the following problems to the best of your ability. This assignment contains X problems, and is worth a total of 30 points. **Note that not every problem may be worth the same number of points!**

⚠ Every file you submit **MUST** begin with an appropriate "shebang" line, followed by a three-line comment block that lists (in order) your name, ID number, and NetID. **If you leave this information out, you may not receive credit for your work!**

⚠ Programs that crash will likely receive a failing grade, so test your code thoroughly (on the `allv` Linux machines) before submitting it.

⚠ Blackboard will provide information on how to submit this assignment. You **MUST** submit your final work according to those instructions by the indicated due date and time. We will not accept (or grade) any work that is submitted after the due date and time, or via different means (without prior written authorization from the instructor).

⚠ **ALL** of the solution code that you submit **MUST** be your own work! You may not receive code from or share code with anyone else, except for the instructor and the (current) course TAs.

Part I: Path Problems (10 points)

(Place your answer to this problem inside a file named “directories.pl”)

Write a Perl script that processes the output of the `echo $PATH` shell command. This command displays the content of the shell environment variable, `$PATH`. This variable contains a list of directory names. When you try to execute a command at the command-line, the shell will look for the program file corresponding to that command in each of the directories listed in the `$PATH` variable. Your script should do the following:

1. Execute the `echo $PATH` command. To execute a shell command in your Perl script, see the example for the `qx` quote operator on the “Quoting Operations” slides.
2. Use a regex to extract the name of each directory in the output of the `echo $PATH` command.
3. Print each directory name to `STDOUT`. Print one directory per line.

Note: You must use the `m//g` regular expression construct to extract all the directory names and store them in an array first. Here is an example to extract all integers in a string and store them in an array:

```
$s = "We had 3 clubs with 10 people each in 4 months.";
@arr = $s =~ /\d+/g;
```

Part II: Replacing HTML Tags (10 points)

(Place your answer to this problem inside a file named “retag.pl”)

Write a Perl script to process an HTML file. Your script should do the following:

1. Accept a filename as a command-line argument.
2. If you encounter a line that is enclosed in paragraph (`<p>` and `</p>`) tags, remove those tags and add a pair of line-break tags (`<br><br>`) at the end.
3. Remove every occurrence of the `<span>` and `</span>` tags.
4. Print the changed content of the file to `STDOUT`.

For example, we have provided you with the sample input file `"tagged.html"`,[1] which has the following contents:

```
I wanna be a rock star and travel really far
<p>And buy me a big expensive car</p>
<p>And make lots of money and find me a honey</p>
And <span>live in</span> a nice big house where it’s sunny
With a pool, I’ll be cool
<p>I’ll <span>always <span>have a gig ’cause</span></span> I’ll be big</p>
<p>I’ll have <span>parties</span> and friends and places to go</p>
<p>The only problem is, I play the banjo</p>
<p></p>
```

Given this file as input, your script should produce the following output:

```
I wanna be a rock star and travel really far
And buy me a big expensive car<br><br>
And make lots of money and find me a honey<br><br>
And live in a nice big house where it’s sunny
With a pool, I’ll be cool
I’ll always have a gig ’cause I’ll be big<br><br>
I’ll have parties and friends and places to go<br><br>
The only problem is, I play the banjo<br><br>
<br><br>
```

[1]Song credit: “Banjo Boy” by Ryan Shupe & the Rubberband.

Part III: Categorizing Files (10 points)

(Place your answer to this problem inside a file named "categorize.pl")

Write a Perl program that takes the name of a single input file as a command line argument. This input file contains the names of several other files, one per line. Your script should read these filenames from the input file and inspect each of those files.[2] It should then do the following:

1. It **must** create and write to five files:
   - `efiles.txt` — This file contains the names of files that exist, one per line
   - `rfiles.txt` — This file contains the names of the files that are readable, one per line
   - `wfiles.txt` — This file contains the names of the files that are writable, one per line
   - `xfiles.txt` — This file contains the names of the files that are executable, one per line
   - `tfiles.txt` — This file contains the names of the files that are text files, one per line
2. Your script **must** also print a summary of the results to `STDOUT`.

For example, if the input file "names.txt" contains the following list of files, all of which exist:

```
/etc/shells (readable and plain-text)
/bin/pwd (readable and executable)
/vmlinuz (neither readable, writable, executable, nor plain-text)
/usr/bin/xxd (readable and executable)
./names.txt (readable, writable, and plain-text)
```

the summary output should look something like this:

```
5 existing files: /etc/shells /bin/pwd /vmlinuz /usr/bin/xxd ./names.txt
4 readable files: /etc/shells /bin/pwd /usr/bin/xxd ./names.txt
1 writable files: ./names.txt
2 executable files: /bin/pwd /usr/bin/xxd
2 plain text files: /etc/shells ./names.txt
```

[2]**Hint:** the Perl slide named "File Checks" is extremely useful here.