CSE 303, Spring 2007, Assignment 2
Due: Monday 16 April, 9:00AM

Last updated: April 4

You will debug a bash script and write a bash script. Problem 2 is more difficult than problem 1.

1. Get `datedwordcount_buggy` from the course website. It is a buggy solution to problem 6 from homework 1, with the "extra-credit" feature that it prints a total line even if there is only one file's words counted. Make `datedwordcount` by modifying `datedwordcount_buggy` to work correctly.

- Do *not* make large changes to the file; change as little as possible.
- There are about 10 bugs (depending how you count). Fixing each bug requires *at most* a few keystrokes.

2. Write a bash script, described below, for building an HTML file with a table summarizing some information from www.foodnetwork.com.

Motivation: www.foodnetwork.com lets you enter phrases and it returns a page with the number of recipes containing the phrases and rating information for the first 20 recipes. For example, if you type exactly:

```
"ice cream" and "chocolate"
```

in the search box on the left, you get a page reporting there are 374 recipes and showing that of the first 20, 6 have a 5-star rating. It might be more convenient to have this sort of information for a bunch of pairs of ingredients in one easy-to-read table. You will write a script that automatically takes ingredient-pairs, runs a search for each of them, and produces a web-page with a simple text table as described below.

Approximate Size: The sample solution is 46 lines, not counting comments, but counting 9 blank lines and a number of lines that are similar or identical to each other. This problem description is *much* longer than the solution and includes a fairly detailed description of the algorithm for completing the assignment. However, you will have to learn some basic HTML and a couple new bash features and Linux programs.

Specification: Write a script called `foodtable` that does the following:

- Take 1 argument, the name of a file where each line contains one or more words, then the word "and", then one or more words. Each line has exactly one "and" and otherwise contains only letters and spaces. Your script should check that it gets one argument and that the argument is a regular file, but you may *assume* the format of this file is correct. An example input-file is on the course website.
- For each line in the input file, extract information from the web page corresponding to searching for those two ingredients. For example if the line is

  ```
  chicken and ice cream
  ```

  the web page to get information from is (all on one line with no spaces)

  ```
  http://web.foodnetwork.com/food/web/searchResults?
  searchString=%22chicken%22+and+%22ice+cream%22&searchType=Recipe&styleId=search
  ```

  The first part (up through the first `=`) and last part (starting with the first `&`) are always the same. The middle part is the contents of the line with `%22` before and after each non-and phrase and all spaces replaced with `+`. This form of URL lets you search "directly" rather than using the search box.
- The information to extract is:
  - The total number of recipes. Assume there is exactly one line in the page with the phrase "found for", this line has exactly one number before the "found for", and that is the number you want.
  - Of the first up-to-twenty recipes, the number of not-rated, 1-star, 2-star, 3-star, 4-star, and 5-star recipes each. Assume the number of not-rated recipes is the number of lines containing the phrase "No Rating", the number of 1-star recipes is the number of lines containing the image `1_stars.gif`, and similarly for 2–5 stars.

- Print *to standard-out* an entire HTML file containing a table with one row of column titles ("Ingredients", "Total", "Top 20 Unrated", "Top 20 1 star", "Top 20 2 stars", ..., "Top 20 5 stars") and one row for each line in the input file. An example HTML file is on the course website.
- Assuming the input argument is correct, your script should not print anything to standard-out or standard-error except the HTML file. (When debugging, it may be very useful to violate this rule.)
- The script should delete any temporary files it makes. (When debugging, it may be very useful to violate this rule too.)

So if your input was in `example_input`, running `foodtable example_input > result.html` should generate no output and produce `result.html` that when opened in a web-browser (e.g., program `firefox` on `attu`) shows an appropriate table.

Advice and Hints: Here is a detailed description of the sample solution, which uses some tricks. It is probably not advisable to deviate significantly from this approach.

- First process the command-line arguments as usual, using appropriate file-test operators.
- Then create two temporary files (using `mktemp`; store the file names in variables so you can delete them later), one for the URLs you will download and one for the files you download.
- Then create a URL file with one line for each web page you eventually need to download. Use `sed`; the sample solution actually uses it twice, piping the output of one use to the other, just because it seemed simpler.
- Then output all the stuff for your HTML output that comes before the rows with actual data. It is easiest to have a separate file containing this text and just `cat` it. (The alternative is lots of `echo` commands, which will make your script longer and more difficult to read.)
- Using a while-loop, the `read` command, input redirection using your URL file, and a variable keeping track of how many lines you have read, do the following for each URL:
  - Print the beginning of the table-row and the first table item by printing the n^{th} line of the input file on the n^{th} iteration of the while loop.
  - Use `wget` to get the web-page (command-line options can help you produce no output and save the file where you want).
  - Print the total number of recipes by printing a modified version of the "found for" line that contains only the number.
  - Print the total number for each rating by printing how many lines contain the appropriate phrase or image.
  - Print the end of the line.
  - Increment the counter.
- Then output all the stuff for your HTML output that comes after the rows with actual data by printing a separate file.
- Then delete any temporary files you created.

Be Careful: As programmers, you have the ability to write buggy programs that harm others. For example, if you write an infinite loop that constantly downloads files from the web, you (and possibly your instructor) could get in trouble. Please be careful!

3. **Extra Credit**

   1. (relatively easy) Change `foodtable` so that for table entries *not including* the total number of recipes, it does not include a 0 (just putting a space instead).
   2. Change `foodtable` so that it reports the ratings for all the recipes, not just the first 20.
   3. Change `foodtable` so that it sorts the recipes from highest to lowest rated. Include in the generated HTML file your rules for deciding what makes a recipe higher rated. These rules must be reasonable.

Assessment: Your solutions should be:

- Correct scripts, etc. that run on `attu.cs.washington.edu`
- In good style, including indentation and line breaks
- Of reasonable size

Turn-in: Use the `turnin` command (man turnin) for course cse303 and project hw2. In particular, type:

```
turnin -ccse303 -phw2 datedwordcount foodtable ...
```

from a directory containing your solution, where `...` should have any other files your script needs to run. If you use one late-day (see the syllabus) use the project hw2late1 instead of hw2 and similarly hw2late2 for two late days. If you do the extra credit, turn in additional files with appropriate names.