CS214 Homework 1

Due: February 27th, 6pm. Submit solutions to CMS, `http://cms.csuglab.cornell.edu` .
Problem 1: Word Frequencies. 20 points. Turn in `index.sh` and `update.sh` .
Problem 2: Interview Question. 15 points. Turn in `interview.sh` .

Introduction

Welcome to CS214! The goal of this assignment is to get you comfortable with commonly used UNIX tools. After the assignment, you should be familiar with the following concepts.

- Input/output redirection.
- Use of pipes and pipelining to simplify your scripts.
- Use of special characters in the shell.
- Regular expressions.
- Variables, both shell variables and how to use your own.
- Passing arguments to a script, and using them inside the script.

Of course, it is not expected that you become an expert on these topics. I am still learning various big tricks even after using UNIX for over 10 years. Rather the purpose is to have you gain enough confidence that you can make use of these tools for the rest of the course, and will know where to look when you are in need of more information.

Guidelines

It is helpful to write scripts *modularly*, which means to break the original problem into subproblems and write a separate script for each subproblem. The initial task can then be solved by pipelining the small scripts.

Please comment your code. Not only is it helpful for others, but chances are that down the road you may need to rerun your old scripts and your own comments become indescribably valuable. It can make the difference between just tweaking some parameters, and having to tackle and code the whole task again. I suggest using the following header for all your scripts.

```
#!/bin/bash
# Created on [date] by [name]
# Purpose of script: [description]
# Usage: [how to run the script, e.g. script -option <datafile>]
```

Hint: In this assignment, it will be helpful to study the **man** pages for **sort**, **uniq**, **wc**, **sed**, **tr** and **grep**.

Problem 1: Word Frequencies

What happens when we enter, say "Quixotry", into a search engine? How can the computers rank the relevance of the contents on the pages of the Internet to this particular query?

We are probably all aware of the current capabilities of Internet search engines, which have become vital tools of the information age. The more modern, sophisticated search engines use algorithms that determine rank by boosting relevance of the web pages that are more likely to appear when one randomly surfs the web.[1]

Let's look at how the early search engines accomplished this task. The basic idea was simple.

- *Crawl* the Internet, and store a local copy of all pages found.
- Find all pages that contain the word "Quixotry".
- Determine the frequency of the word on each page.
- Sort the pages by frequency, and return the first 10 pages.
- (But of course first list the website owners who pay us money.)

Of course, we can't do all of this when we receive the request since that would be impossibly slow. Instead, search engines even today crawl the web every so often and determine all different words that appear on the Internet. They then create an *index* for each unique word which is a list of web pages that contain it and the frequency of that word on each page.

In this assignment, you will write a script that parses a given web page, determines candidate index words regardless of their capitalization, and gives the frequency of each of them. This information could then be used to update the large index. For simplicity, we will discard the web page information and only look at an index consisting of words and their frequency.

How should you proceed? It will be instructive to follow these steps. (**Hint:** Take a look at **tr**, **sed** and **sort** to solve these subproblems). We will assume that HTML tags have already been removed from all data files.

- Write a command that reads from a data file and removes punctuation characters.
- Write a command that reads from a data file and changes all lower-case characters to upper case characters.
- Write a command that reads from a data file and removes all tabbing (`\t`).
- Write a command that deletes all empty lines in a data file.
- Write a command that takes as input a file containing a word on each line, sorts the file alphabetically, and removes duplicates.

Your assignment is to write two scripts, `index.sh` and `update.sh` .

[1] Our very own Jon Kleinberg developed the HITS algorithm based on this intuition that appeared in print slightly before Google's PageRank algorithm.

1) The script `index.sh` will be invoked with a single parameter that specifies the data file to be used.

```
$ ./index.sh data.txt
```

After removing all punctuation and tabbing characters from the data file, and making every upper-case letter lower-case, the script should output all remaining words to `freq.txt` in decreasing order of frequency. Each line should consist of two columns separated by a tab: the number of occurrences, followed by the word itself. For example:

```
24      quixotry
10      scrabble
2       scoring
1       highest
1       word
```

2) When we crawl a new web page, one way to update the index is simply recalculate it for all web pages. This is unnecessarily expensive since it is not hard incrementally update an index. More to the point, imagine that you already generated `freq.txt` file from running `index.sh` on a large document `big.html` . How could you update the index in `freq.txt` for a new document? For instance, if we use the index from the earlier example and `new.html` contains:

```
The highest scoring word in Scrabble is QUIXOTRY.
```

then the updated index should read

```
25 quixotry
11 scrabble
3 scoring
2 highest
2 word
1 in
1 is
1 the
```

by only parsing `freq.txt` and `new.html` .

Write a script `update.sh` that accepts two parameters: the previous index name, followed by a data file name, and prints an updated index to `stdout` . It is fine if your solution is not the most efficient you can think of.

Hint: The output should be the same as if `big.html` and `new.html` were merged, and run as input to your old `index.sh` .

Hint: Consider using **gawk** if you encounter arithmetic...

Problem 2: Interview question

Software engineer candidates were given the following problem during an hour-long phone interview at Amazon [**1**].

> Last year my team had to remove all the phone numbers from 50,000 Amazon web page templates, since many of the numbers were no longer in service, and we also wanted to route all customer contacts through a single page.
>
> Let's say you're on my team, and we have to identify the pages having probable U.S. phone numbers in them. To simplify the problem slightly, assume we have 50,000 HTML files in a Unix directory tree, under a directory called `"/website"` . We have 2 days to get a list of file paths to the editorial staff. You need to give me a list of the `.html` files in this directory tree that appear to contain phone numbers in the following two formats: `(xxx) xxx-xxxx` and `xxx-xxx-xxxx` .

According to an interviewer, about 25-35% of the candidates are unable to solve this problem at all during the interview, even after being given lots of hints.

Write a single bash shell script `interview.sh` that solves a slightly different problem. Instead of only printing file names, for each potential phone number that you encounter, you should print the name of the file followed by the potential phone number separated by a tab (`\t`) character. More specifically, the script should:

- Print each potential phone number in the files in `/website` along with the file name line-by-line in any order.
- Not print any duplicate entries.
- Not print anything else (including debugging information) to `stdout` .
- Have useful and clear comments.
- Be efficient with respect to running time and memory space (for instance, do not enumerate all potential phone numbers).
- Not require any parameters.
- Accomplish the task in at most 10 lines of code, not including header lines and comments.

For example, if `file1.txt` reads

```
You should call my assistant at 607-255-5555 (+16072555555
internationally) or (607)-255-4321 in the case of emergency.
```

and `file2.txt` reads

```
For sales inquiries, please contact (800) 314-1592, extension 653.
```

then the script should output

```
file1.txt       607-255-5555
file1.txt       607-255-4321
file2.txt       (800) 314-1592
```

in any order.

References

1) Jeff Atwood. January 2008. *Coding Horror: Getting the Interview Phone Screen Right*. http://www.codinghorror.com/blog/archives/001042.html