CS2043 Homework 3

Due: Friday, February 14, 2014 at 5:59PM EST **soft deadline** (so that you can enjoy your Valentines Day and your February break!)

or Sunday, February 16, 2014 at 11:59PM EST **hard deadline** (the TAs will not be available during the February break, i.e., Saturday, the 15th, and Sunday, the 16th)

Important Note: The configuration of the CSUG Lab machines is our standard environment. That means that you are responsible for making sure that your scripts execute as intended in one of the CSUG lab machines or in a clone thereof installed on your computer as a Virtual Machine.

General Instrictions:

- You can form groups of **two** students to complete this assignment. Please form a group on CMS and submit only one solution per group.
- You must complete this assignment using only the Unix tools that were discussed in class between (and including) Lectures 2 and 9, plus the tools mentioned in the hints for each problem. You are allowed to use any of their options (as long as they work on the CSUG Lab environment). You are not allowed to use **awk**, **loops** or **if statements** either in bash or in any other programming language.
- Make sure that you strictly follow the specifications for each exercise. Points will be deducted if your submission fails to comply with the specifications.
- For each problem, you will write a script (not the output thereof), and save it to a file named with the problem label, e.g., **problem1.sh**. Assume that the input is in the same directory as the script. Remember that a bash script is a text file whose first line contains:

```
#!  /bin/bash
```

- **New!**: Unless you have been asked to write files to a specific location, your scripts should not write files anywhere (e.g., temporary or output files). To be safe, assume that the environment in which we are going execute your script does not give your script permission to write files on the current directory or anywhere else in the filesystem.
- Once you complete the assignment, make a compressed tarball using gzip named `submission.tgz`, containing all the scripts you have produced. Submit your compressed tarball to CMS (`http://cms.csuglab.cornell.edu`). Your tarball should contain no directories.

– **Important**: You are not allowed to disclose parts of your solution on piazza. You can ask for clarifications without revealing your attempts to solve the problem.

Everyday Tasks

– **problem1.sh** : Suppose you receive your paycheck on the 15th day of each month. Write a script that sends you an e-mail with subject "Payday yet?" and whose body contains the number 1 if the day of the month in which you execute the script is on or after your payday, or 0 otherwise.

– **problem2.sh** : Create a file called `weekly.txt` that describes your weekly activities using the following format:

```
Mon :  List of activities for Monday
Tue :  List of activities for Tuesday
Wed :  List of activities for Wednesday
Thu :  List of activities for Thursday
Fri :  List of activities for Friday
```

(replace "List of activities for..." with your weekly tasks for that day).

Write a script that sends you an e-mail whose subject is "Today's tasks" and whose body contains your tasks for the **the day in which you run the script** using `weekly.txt` as input, i.e., on Mondays, it will send you the task for Monday, on Tuesdays, the tasks for Tuesday, and so on. Please do not include `weekly.txt` in your submission.

– **problem3.txt** : We want to program the computer to remind us of our everyday tasks by e-mail. Write a crontab line (not a shell script) in file `problem3.txt` that is going make the system execute `problem2.sh` automatically every weekday (Mon-Fri) at 6:00 AM.

Hint1: you can send the output of a command via e-mail by redirecting its output to

```
mailx [-s "subject"] <e-mail_address>
```

Remark: `mailx` is not easy to configure, but it is already configured and ready to use on the CSUG Lab machines and on the VM images (you need to be connected to a Cornell network for `mailx` to work if you are using a VM).

Hint2: the command `date` prints today's date. We can use its user-defined formatting options to extract what we want from its output (see `man date`) or parse its output using other tools.

Remix

A remix is a song that has been edited or completely recreated to sound different from the original version. One common form of remix is to combine two songs together.

- **problem4.sh** : Assume that the input consists of two files, one for each song, whose names are `song1.txt` and `song2.txt`. Write a script to create remix lyrics that alternate between the lyrics of two different songs, i.e., first line of the remix is the first line of `song1.txt`, the second, the first line of `song2.txt`, the third, the second line of `song1.txt`, and so on. (If one song is shorter than the other, the longer should continue through its end, alternating with empty lines).

  We have made available a tarball with the lyrics of the top 100 songs on the radio today so that you can have fun with your script. Once you find two songs you want to remix, rename their files to match the input filenames and run your script on them. The lyrics tarball can be found at `http://www.cs.cornell.edu/Courses/cs2043/2014sp/top100.tgz`. **Remark**: The lyrics files were automatically extracted from the web and may contain errors, such as missing new lines. Feel free to modify them so that they will look better after your remix.

Straight A

We recorded the NetId of the students who took CS 2043 in the past 4 years in the following way. If a student whose NetId is abc123 has taken CS2043 in the spring of 2012, then she will have a text file in the following path[1]:

```
cs/cs2043/2012sp/abc123
```

However, in 2010, CS2043 used to be two different courses, namely CS2042 and CS2044. Thus, the students who took the course in 2010 have their files either in

```
cs/cs2043/2010sp/CS2042/
```

or

```
cs/cs2043/2010sp/CS2044/
```

[1]The data is fake and generated by the instructor.

Moreover, the professor annotated the files of the students who got an A+ in the class, by writing the character '#' in their files.

Download the data at
`http://www.cs.cornell.edu/Courses/cs2043/2014sp/cs2043-students.tgz`

- **problem5.sh**: Write a script that outputs a list of NetIds (only), one per line, from all students who got an A+ in the class over all years. Your output should contain a list of NetIds only, one per line, and no other character. In addition, *your script should not display any errors on the* `stdout`. Your script should be saved at the same level as that of directory `cs`. To help you debug, your output should contain 50 NetIds.
- **problem6.sh**: Create a directory called `all` at the same level of a script you are going to write for this problem and copy to it a flattened version of the `cs` directory, i.e., place all the student files under `cs` in `all` with no directory structure. **Note**: Submit your scripts only (not the directories).

Big Data Investigation: The Enron dataset

Enron was the worlds leading energy company; it declared bankruptcy in December 2001, which was followed by numerous investigations. During the investigations, the managers' mailboxes were publicly released by the Federal Energy Regulatory Commission (FERC)]. It contains data from about 150 users, mostly senior management of Enron, organized into folders. The corpus contains a total of about 0.5M messages.

If you're working on the CSUG Lab machines, make a link in the directory you are going to write the script for this problem to `\home\bda23\pub\enron_mail_20110402`. Please don't copy this dataset to your home directory in the CSUG Lab. The file is huge and it is going to make the filesystem full and usable to others.

If you are working on your own computer, you can download and extract the Enron dataset from `https://www.cs.cmu.edu/~enron/enron_mail_20110402.tgz` and place it at the same level as the script you are going to write for this problem.

- **problem7.sh**: Write a script to extract a list of all the e-mail addresses from which e-mails have been sent. The list shouldn't contain duplicates. Remember that the e-mail address of the sender of an e-mail can be found in the e-mail files in a line that starts with any of the strings: "From:", "from:", "FROM:", followed by whitespace characters and an e-mail address. As this dataset is huge, your script may take several hours to run. Accordingly, *the commands you will include in your scripts should run in the background.*

  Here is an example of an e-mail header which is included in every e-mail of the dataset. This is an e-mail stored in file `enron_mail_20110402/maildir/allen-p/inbox/1.`.

```
Message-ID: <16159836.1075855377439.JavaMail.evans@thyme>
Date:  Fri, 7 Dec 2001 10:06:42 -0800 (PST)
From:  heather.dunton@enron.com
To:  k..allen@enron.com
Subject:  RE: West Position
Mime-Version:  1.0
```

Hint: restrict your search to parts of the dataset to test your script so that it can run fast. For example, examine one manager's mailbox or one of their mail folders at a time. Once you are confident about your script and curious to know a list of all senders, you can run it for a few hours on the entire dataset.

Spell Cheker

In this exercise you will build a rudimentary spell checker for formal English using `grep`.

Here is our strategy. We are going to tell `grep` to read the list of English words from the file at `http://www.cs.cornell.edu/Courses/cs2043/2014sp/english.txt.gz`. We will tell `grep` to ignore case, and to invert the match so that it will only print lines that do not match the English words (that is, we delete the words that are correctly spelled and print the misspelled ones). We will also tell `grep` to match full words (and not substrings, such that "futon" does not count as a match for the preposition "on").

Consult the man pages for `grep` and find out how to accomplish these tasks.

- **problem8.sh**: Write a script that takes in a text in formal English (i.e., without contractions, such as don't, we're, they're, and so on) and prints a list of the misspelled words. **Warning**: Your spell checker will not be perfect as there are many missing words in the English words file. We will be looking for a correct script for this task, not for a perfect spell checker.

Revisiting Restaurants

In the previous assignment we analyzed the restaurant files (which can be found at `http://www.cs.cornell.edu/courses/cs2043/2014sp/restaurants.txt.gz`), which lists the restaurant going events, one per line, where each line lists the names in the party and the associated restaurant. Each line has the following format:

```
name,name,...,name;restaurant
```

In this problem, we will give you some more details on these groups. We kept track of the person who paid the bill at the restaurant. He or she is the second to last person in the list of people in each party (from left to right).

– **problem9.sh**: Write a script that uses regular expression to extract the name of the person who paid the bill in each party and print them, one per line.

Phone numbers

We asked our friends to fill out a form with two fields, their names and their phone numbers. We collected 25 responses, but each person entered their numbers using a different format. For example, some of them wrote their number without area code, such as 9232-0092, others wrote them with separators, such as (607) 434-2323, and others without them, such as 905993232. A phone number contains between 7 and 11 digits.

Download a small address book at
`http://www.cs.cornell.edu/courses/cs2043/2014sp/phone-data.txt.gz)` and learn all the variations in phone number format.

– **problem10.sh**: Write a script that uses regular expression to extract and print all phone numbers from the file, one per line, without having to explicitly list any of them.