CS2043 Assignment 4

Due: Saturday February 28th 2015 at 11:59 PM on `http://cms.csuglab.cornell.edu`.

You can form groups of two students to complete this assignment. Please form a group on CMS and submit only one solution per group.

General Note: This assignment (and future ones) require you to have access to a Unix-like (Linux, Mac OS X, etc) machine. If you do not have such an operating system installed on your local machine, make sure to get a CSUG Lab account. Different systems have slightly different configurations. The main environment in this class is GNU/Linux.

Assignment Notes:

- You must complete the assignment using tools that were discussed in class.
- This assignment is composed of two parts.
- For each problem, you will write a script (not the output thereof), and save it to a file named with the specified problem label
  e.g. **problem1name.sh** or **problem1name.py**.
  Assume that the input is in the same directory as the script.

  A bash script is a text file with: `#! /bin/bash`
  A python script is a text file with: `#! /usr/bin/python`
  as the first line.
- Please also include a short README file with some feedback on the assignment.
- Remember to start this assignment early.

Email Reminder

Create a file called `myweekly_act.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)

act_reminder.sh: Write a script that sends you an e-mail whose subject is "Reminder: Tasks for today" and whose body contains your **tasks for the day in which you run the script** using `myweekly_act.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 `myweekly_act.txt` in your submission.

remind_daily.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 the file `remind_daily.txt` that is going to make the system execute `act_reminder.sh` automatically every weekday (Mon-Fri) at 6:00AM.

The `date` command 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.

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

```
mailx [-s "subject"] <e-mail address>
```

`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).

My ebook reader

Your objective in this part is to write a python script, `ereader.py` that is a rudimentary "plain-text eBook reader". The idea here is that a user would launch this program, read a little bit of a book and close it. Next time the user launches this program with that book, he continues from where he left off the last time.

The script takes at least one command-line argument, the name of the file to open. The user can also supply an optional paging argument identified by the flag `-n` that dictates how many lines to display per page. If the user did not supply that argument, the script should assume a default paging of 40 lines per page. Finally, for navigation, we will only support three operations: "next page", "previous page", and "quit". The next page is displayed when the user presses the *n* key, the previous page is displayed when the user presses the *p* key, and the application quits when the user presses the *q* key.

To make things simple, here is how we're going to make the script work:

1. When the script starts, it checks to see if the user's home directory contains a ∼`/.reader_rc` file *(the filename starts with a dot)*. We will use this file to store the progress of different files. So if the home directory does not contain a ∼`./reader_rc` file, we will create one.
2. The contents of a `.reader_rc` file will be a two comma-separated columns of `file_content_hash,line_count`. Where the line count is the number of lines the user has read so far. The advantage of storing files by content hash, is that if the file is reopened from a different directory, or from a different file name, or even a different machine, the user will be able to resume reading where he left off. That is, the information your script stores about the file depend only on the content of the file and not its name/location.

   Here are the contents of a sample `.reader` file:

   ```
   3e3ac0b27b0adeb9138206eea2c43bb2,120
   5eb63bbbe01eeed093cb22bb8f5acdc3,0
   98f84fd484f8c2273f28cc645935f650,40
   ```

3. When the program launches, it will open the passed-in file name, compute its content hash using the `MD5` hashing algorithm, read the `.reader_rc` file, and skip the indicated number of lines. Then it will display a page's worth of lines (as dictated by the paging argument) to the user.
4. When the user presses one of the command keys (*n*/p), the script will update the files read lines count, rewrite `.reader` file to disk, and print a page to the terminal.
5. When the user presses *q*, the program quits.

Summary/Notes:

- Python has many built-in hash libraries. Some are deprecated. Do not use the deprecated libraries. Python's online documentation should guide you here. Another alternative is to call the Unix `md5sum` program through python.
- Your script should ideally capture the "command" key presses without the need to press ENTER after each of them. However, it is accepted to require the user to press ENTER before accepting a command key. So, pressing just 'n' or 'n' followed by ENTER are both fine for advancing the page.
- Files beginning with a dot are hidden in Linux, so you won't see them when you do an `ls`. To see all files, do: `ls -a`
- Just to be clear, here is how you call the program:
  - `./ereader.py the_republic.txt` will launch the script with default paging of 40 lines per page.
  - `./ereader.py -n 80 the_republic.txt` will launch the script with paging set to 80 lines per page.
- Since a user might use different paging even with the same file, make sure that the `.reader` file contains number of lines displayed and not the number of pages displayed.
- To download public domain eBooks in plain text file format, check out the Gutenberg project: `http://www.gutenberg.org/browse/scores/top`

Remember your best friends are the Python documentation: `http://www.python.org/doc/`, the `man` tool and your favorite search engine.
If you need to ask questions, use the class discussion board on Piazza.