CS 45 // Winter 2023 Assignment #2
Akshay Srivatsan, Ayelet Drazen, & Jonathan Kula January 23, 2023

Assignment #2—Bourne to Be Wild

Due: January 30, 2023 at 11:59pm

This assignment consists of two different components:

1. **You'll write a simple shell script that does some data analysis**
2. **You'll get some practice using vim**

Before starting the assignment, you will want to ensure you've installed all the software required for lectures 1 through 5.

We expect this assignment to take 1-3 hours depending on your proficiency level with the tools. If you find yourself unproductively stuck or unproductively struggling (spinning in circles), ask on Ed and/or go to office hours!

As a reminder, you can use `man` pages to learn more about commands you don't yet know (especially what flags they support). The website cheat.sh also has many examples of how to use different commands.

Part I: Write A Shell Script (2 points)

For this part of the assignment, you will write a shell script which does some data wrangling. You'll be analyzing an input file containing information about Stanford CS Faculty. To get that file, use the command below:

Unset

```
curl -Lo cs_faculty.csv
https://cs45.stanford.edu/res/assign2/cs_faculty.csv
```

This file contains faculty members' names, titles, and research groups. For example, here's a line from the file:

Unset

```
Alex Aiken, Professor, Computer Systems + Programming Systems and Verification
```

As you can see, the different columns are separated by commas. This is what's called a "comma-separated values" file, or a CSV file for short. The last column contains the professor's research groups, each separated by a "+" (plus sign). For the purposes of this assignment, you can treat this as an ordinary text file.

Your task is to write a shell script that analyzes this file. As a reminder, a shell script is a text file containing shell commands, which might look like this:

Unset

```
#!/usr/bin/env bash
name="world"
echo "hello $name"
echo "you can run other shell commands here"
```

Each line of the shell script is run as if you typed it at the shell, except for comments (lines beginning with "#").

Create a shell script called `analyze.sh`, and download `cs_faculty.csv` into the same directory. You can run your script by running `sh analyze.sh`. In the shell script, add commands which do the following:

1. Make a new subdirectory called `analysis`. You might run the shell script multiple times, which means the directory may already exist; make sure your script works whether or not the directory already exists (hint: you might need to pass a flag to `mkdir`).
2. Delete any files inside `analysis`, if there are any. Once again, make sure your script works whether or not there are files inside the subdirectory.
3. For each of the following research groups, create a file in `analysis` with the information of every professor who researches that topic. Make sure the information is in the same format as the original file. (In other words, if you find a faculty member who researches a topic, you should include the *entire* line with all of that faculty member's information in the research group specific file). Make sure each file has the following specified name, and is within the `analysis` subdirectory.
    a. "Artificial Intelligence" => `ai.csv`
    b. "Computational Biology" => `bio.csv`
    c. "Computer Graphics" => `graphics.csv`
    d. "Computer Security" => `security.csv`
    e. "Computer Systems" => `systems.csv`
    f. "Human-Computer Interaction (HCI)" => `hci.csv`
    g. "Theory" => `theory.csv`

    One hint here is that it actually does not matter that the data is in CSV format, and we strongly recommend that you do not parse the data into separate columns. Instead, we recommend treating each line of data as a single unit and searching for the research group of interest on that line.

4. Create a file named `big_groups.txt` in the `analysis` directory with the names of every research group with more than 10 members. You might want to edit the function you defined in step 3 to make this easier. Make sure to output the name of the *group* (e.g., "Artificial Intelligence"), not the name of the *file*.

5. Create a file `analysis/systems-ai-join.csv` with the information of every professor who researches both "Artificial Intelligence" and "Computer Systems". Remember that they may also research other things.
    a. There's a few different ways to do this, but here's a hint for one of the ways: the command `uniq` (which normally removes duplicate lines from a file) can be used to print *only* duplicate lines with a certain flag.
    b. Note that `uniq` requires its input to be sorted.
6. Create a new file `analysis/names.txt`, with the names of every faculty member sorted alphabetically. Make sure this file only contains their names, not their title or research groups. You may want to use the `cut` command, although there's a few others that would work too.

You're free to output whatever you want (with `echo`) to help you debug. You might want to add the command `set -x` at the beginning of the script; this makes the script print out each command as it runs it.

Make sure your shell script has the name specified above (`analyze.sh`), and creates files with the correct names.

Part II: Try vim out! (1 point)

For this part of the assignment, you'll try `vim` out by editing a file on a remote computer (ooo!) – this is the same one that we checked out some logs from during the Data Wrangling lecture. You'll be fixing up a Python file with a few errors—but don't worry if you're rusty with Python, the comments will instruct you what to do and what to write.

Configuring vim

Let's make your vim a little easier to use, more helpful, and fun :)

The first step is to create a `.vimrc` file if you don't already have an existing one. To check if you have a `.vimrc` file (you may have configured one in the past!), you should run the following: `cat ~/.vimrc`. If you get an error message that reads `No such file or directory`, then you don't have a `.vimrc` file and will need to create one.

If your `.vimrc` file is blank, you might consider using ours! You can read it right in `vim` with:

Unset
```
vim https://web.stanford.edu/class/cs45/res/lec5/.vimrc
```

You can copy it over right to your home directory (note this will overwrite an existing configuration):

Unset

```
curl -Lo ~/.vimrc
https://web.stanford.edu/class/cs45/res/lec5/.vimrc
```

To create or edit a `.vimrc` file, you should run `vim ~/.vimrc`. This will allow you to open your`.vimrc` file with... `vim`!

We recommend the following customizations (which are included in our `.vimrc` file already):

- Enable line numbering
- Enable syntax highlighting
- Enable mouse usage
- Enable "hidden" buffers (opening multiple files without having them all on-screen at the same time; see `:help hidden` for more info)
- Enable filetype detection, plugins, and indentation (see `:help :filetype-overview` for more info)

You are more than welcome to add other customizations to your `.vimrc` file, but we recommend at least using the ones above.

If you submit your `.vimrc` file, our autograder will automatically check it to make sure you enabled the above customizations, but it's not required.

Editing a Python file

To access the file, copy and paste the lines below into your command line- (that's right, you can copy/paste multiple commands at once!). You should change `<SUNET>` to **your SUNet ID,** (which is the part of your Stanford email before @stanford.edu).

Unset

```
export SUNET=<SUNET>
vim sftp://s-cs45-assign2-$SUNET@192.9.152.85/exercise.py
```

For example, if your SUNet ID was `example`, your commands would look like:

```
export SUNET=example
vim sftp://s-cs45-assign2-$SUNET@192.9.152.85/exercise.py
```

You may have to press `ENTER` once when prompted to see the file. If you have any issues accessing the file, let us know in an Ed private post!

Protip: before saving with `:wq`, go into normal mode and enter the command `:retab` to make sure all your whitespace is consistent- Python requires this.

When you submit your assignment, the autograder will automatically download your file and grade it. You don't need to submit it with the rest of your files.

Feedback Survey (0.5 points)

Once you have completed the assignment, you can earn an additional 0.5 points by completing our anonymous feedback survey. Given this is the first offering of the course, we want to collect as much feedback as possible to improve the course in the future. You can complete the survey here. Once you complete the survey, you will receive a completion code which you should place in a text file named `survey.txt`. The survey is anonymous so submitting the completion code is the only way to verify that you completed the survey. *Please do not share this code with anyone, as that would constitute a breach of the honor code.*

Submitting Your Assignment

Once you have finished this assignment, you will need to upload your files to Gradescope. Make sure to upload all files to the Assignment 2 submission page. You should also upload `survey.txt` if you completed the survey.

The only file you need to submit to Gradescope is `analyze.sh` and `survey.txt` if you completed the survey. If you'd like, you can also upload your `.vimrc` to verify that it contains the customizations we recommend, though this is not part of your grade and is not required. You don't need to worry about `exercise.py`, as it will be downloaded by the autograder automatically when you submit.

It may be difficult to upload your `.vimrc` file into Gradescope, as the file is hidden—but it's easy to add all these to a zip file!

Unset

```
# Run this command in your assignment directory:
zip -jv assign2_submission.zip ~/.vimrc ./analyze.sh ./survey.txt
```

Once you have created a zip file, you can upload it to Gradescope.

All files must have the same name as specified above.

Bonus: More on grep & sed (optional, but helpful extra practice!)

We got some feedback from the last assignment that it would be helpful to return to `grep` and `sed` for some additional practice. This exercise will emphasize the difference between these two utilities and will walk you through a couple different examples you can try.

Start by grabbing the files we'll use by copy-pasting the line below into your terminal and pressing enter:

Unset

```
for i in {1..10}; do curl -Lo emails_$i.txt
https://cs45.stanford.edu/res/assign2/emails_$i.txt; done
```

Inside these text files, there are lists of email addresses. There is one email per line, each with a comma afterwards. We want to accomplish two goals: (1) Locate certain email addresses, and (2) replace the commas with semicolons (Outlook only accepts semicolon-delimited email addresses!).

(1) Locating certain email addresses

When your task is locating information, `grep` is your friend. There's a particular email address to locate: `bighearted.piscatorial.americanbulldog@example.com`, and it's in one of those ten email files. Now, you could use `grep` or `sed` on each one of them, searching for a file where a line matches, or you could...

Unset

```
grep -n 'bighearted.piscatorial.americanbulldog@example.com' *.txt
```

The line above uses **globbing** to find all the `.txt` files (`*.txt`), and give them to `grep`. (Note that while this looks like RegEx, the syntax is simplified– e.g. the above glob knows "any files ending in `.txt`." If you were using RegEx for this, you would probably write `^.*\.txt$`).

This will search through *all* the text files at once, and show you what file it found that match in, along with the line number (`-n`). Woah! I use this tactic all the time when searching for something in a bunch of files (for example, finding where I used a certain class or variable name in my code...)

(2) Replacing commas with semicolons

However, if your goal is to modify a file or text– not simply search– `grep` just won't quite cut it. Let's replace all the semicolons in each file:

Unset

```
sed -i .backup 's/,/;/' *.txt
```

This will **replace all the commas with semicolons** in all the `.txt` files it finds in the current directory, and back up the original versions to a `<filename>.backup` file.