HW 2: Data Plot

In this assignment you will write three different Bash scripts that combined will compare the sizes of a set of webpages.

You will write three separate Bash scripts:

- `getcourses` will produce a list of URLs found in a given html file
- `perform-measurement` will assess a given set of URLs for the size of file they link to
- `run-analysis` will take a set of URLs and compare the size of files they point to by plotting them on a graph

Getting Started

Download the starter code by running these commands from Klaatu or the VM. Copy it without the `$`

```
$ wget
https://courses.cs.washington.edu/courses/cse374/20au/homeworks/hw2.tar.gz
$ tar -xzf hw2.tar.gz
```

You will now see a directory called hw2 with some helper files in it:

- `courses-index.html`
  - The data we will be working with for this assignment
- `popular-small.txt`
  - A small file of example URLs to use to test your `perform-measurement` script
- `scatterplot.R`
  - A script we wrote to output a chart to illustrate your measurements

You will turn in `getcourses`, `perform-measurement`, and `run-analysis`

You will need to download your files from klaatu to submit from your own computer. You can use a program like Filezilla or WinSCP to do this, or you can run this script from your laptop's terminal (Command Prompt for Windows, Terminal for MacOS)

```
scp <username>@klaatu.cs.washington.edu:<path-to-your-files> .
```

- Your submission will be automatically graded on Gradescope- the score you receive will be your final score minus any extra credit or late penalty deduction
- You may resubmit to Gradescope as many times as you like until you get the score you like, the Gradescope output will include details on what the test cases are looking for
- Your scripts can always create temporary files, but they must remove them after they're done with them

- Error messages should be printed to stdout and can be any text you find appropriate

Part 1: getcourses - extract URLs

getcourses

Arguments

1. name of the output file for results
2. name of input html file

Validation

- If the user provides fewer than 2 arguments, the script should print an error message and exit with a return code of 1
- If the file provided for input does not exist, the script should print an appropriate error message and exit with a return code of 1

Functionality

- Searches through input html file for URLs that contain `courses.cs.washington.edu/courses/cse`
    - These will by default link to the hub of this course's pages across multiple quarters. We specifically want the webpages for the most recent quarter- to get these simply append "20su/" to the end of each matching line
    - Add each 20su link to the output file on its own line
    - Not every course is offered this quarter, so some of these URLs will not be valid- that is expected
    - We would also like you to only consider courses that follow the pattern `cseXXX` where X is a number. This means that you will have to exclude courses like csem584 and cse599b
- If the text file provided as an argument (for the output) exists, the script should simply overwrite it without any warning
- If the script does not report any errors, it should exit with a return code of 0
- Don't worry if there are duplicate URLs in the output of `getcourses`, this is expected
- The core idea is that you would have exactly one URL per "block" (title and paragraph below it) in `courses-index.html` to be copied to `courselist`, minus any whose URLs need to be filtered (i.e. have letter section extensions or are CSEM/CSEP).

Example

```
./getcourses courselist courses-index.html
```

Should write content similar to the following into `courselist`:

```
http://courses.cs.washington.edu/courses/cse120/20su/
http://courses.cs.washington.edu/courses/cse131/20su/
http://courses.cs.washington.edu/courses/cse142/20su/
...
```

Hints

- There are several ways to do this, and different utilities can help. The following hints can help but you don't need to use all these hints:

    a. grep can use string patterns to find the URLs within the file

    b. sed - stream editor - can do insertion, deletion, search and replace(substitution) of a file

    c. read - can let you move through a file one line at a time

- Reminder you may create temporary files, but you must remove them before finishing your process

- `courses-index.html` is taken from https://cs.washington.edu/education/courses/. You can view this page in a web browser to see what is there, and if you right-click and 'inspect' you can see which html code is associated with which part of the page.

- Search for all the lines that contain the string `http` or `https`.

- Output only the URL part of that line. There are several ways to do this.

    a. `grep -o` may be useful. Remember that you can pipe grep commands together like `grep 'pattern1' file | grep 'pattern2'`, which will search for pattern2 on the lines which matched pattern1

Part 2: perform-measurement - page size

perform-measurement

Arguments
- Takes one argument, a URL

Validations
- If the user does not provide any arguments, the script should print an appropriate error message and exit with status code of 1.
- If the user provides an erroneous argument or if downloading the requested page fails for any other reason, the script should simply print the number "0" (zero). In this case, or if the page is downloaded successfully, the script should exit with a return code of 0 after printing the number to standard output.

Functionality

- Perform-measurement should pull down the webpage given and print to standard out the size of the corresponding page in bytes
- Note we do not want any output from the wget to show up

Example

For example, executing your script with the URL of the Wikipedia page for gay computing pioneer Alan Turing.

```
./perform-measurement https://en.wikipedia.org/wiki/Alan_Turing
```

should output *only* 544474 to standard output. This may change if the Wikipedia page gets edited, but not by a lot.

```
544474
```

Hints:

- The `wget` program downloads files from the web. Use `man wget` to see its options.
- Using `wc` is one way to print the number of bytes in a file. You can use `wc -c < test_file`
- To suppress the output from wget you might want to look at the man pages and try to find a flag that might help you do this
- To suppress the output of a command, try to redirect its output to `/dev/null`. For example try `ls > /dev/null`
- Your script may create *temporary* files if you want.

Part 3: run-analysis

Run the experiment by measuring each web page

run-analysis

Arguments

- The output file where results will be stored
- The input file to analyse. Should be in the format of the output of "getcourses", a list of URLs.

Value

- If the user provides fewer than 2 arguments, the script should print an appropriate error message and exit with status code of 1.

Functionality

- For each URL contained within the input file, `run-analysis` should produce a line in the output file with the course-number and pagesize separated by a space like so:

```
course-number1 page-size
course-number2 page-size
course-number3 page-size
...
```

- The `course-number` is the three-digit course number. You can extract this from the URL you give to perform-analysis using a similar method to the one you used to parse the original course listings. You will want only the three digit numerals - not any letter section extensions. The `page-size` is the result of `perform-measurement`.
- Because it can take a long time for the experiment to finish, your script should provide feedback to the user. The feedback should indicate the progress of the experiment.
    - Before executing perform-measurement on a URL, your script should print the following message: "`Performing byte-size measurement on <URL>`".
    - Once perform-measurement produces a value, if the value is greater than zero, the script should output the following message: "`...successful`". If the value is zero, this means some error has occurred, and the script should output the following message: "`...failure`".
- When `run-analysis` finishes, it should exit with a return code of 0.

Example

To debug your script, instead of trying it directly on `courses-index.html`, we provide you with a smaller file: `popular-small.txt`. Here's what it contains:

```
http://courses.cs.washington.edu/courses/cse332/20su/
http://courses.cs.washington.edu/i.will.return.an.error
http://courses.cs.washington.edu/courses/cse333/20su/
```

Executing your script as follows:

```
./run-analysis dataout popular-small.txt
```

Should produce exactly the following output (note -- there is **no** newline between "on" and the URL, there is only one *after* the URL):

(Note 2: This part of the output should be sent to the terminal and not to the dataout file)

```
Performing byte-size measurement on http://courses.cs.washington.edu/courses/cse332/20su/
...successful
Performing byte-size measurement on http://courses.cs.washington.edu/i.will.return.an.error
...failure
Performing byte-size measurement on http://courses.cs.washington.edu/courses/cse333/20su/
...successful
```

And the content of `dataout` should be similar to the ones below. These numbers may change a bit if these classes change their front page.

```
332 4673
333 17127
```

Heads-up! On Gradescope, the autograder tests `run-analysis` with **fake, random data**. This allows us to verify that your script is parsing input correctly rather than being dependent on the state of the web at the time of writing. Thus, the diffs printed by the autograder **are not consistent** with the results you will get if you run your script locally.

Part 4: Put all the parts together and generate a plot

Note: There is nothing you need to turn in for this part of the assignment, but you should run the provided script anyway to check that the plot generated looks right.

It is hard to understand the results just by looking at a list of numbers, so you would like to produce a graph. More specifically, you would like to produce a scatterplot, where the x-axis will show the course number and the y-axis will show the size of the index page.

We have provided a script to visualize this called `scatterplot.R`

Note that this script expects your experimental results to be stored in a file called `dataout`.

The script should produce a file called `scatterplot_out.jpg`. If you are using VS Code with Remote SSH extension, you can just click on the file. Otherwise, you may need to copy it to your laptop using `scp`

```
scp myusername@klaatu.cs.washington.edu:~/path/to/scatterplot_out.jpg .
```