CS 618 LAB 2 Indvidual assignment, Deadline: 14th August 11.59 PM

This lab assignment will help you to get familiar with git and bash.

Begin by creating a git repository.

a. Create a directory in which to practice via `mkdir ~/your_roll_no_CS618`,
b. Change to the new directory via `cd ~/your_roll_no_CS618`

Configure git as follows.

```
a. git config --global user.name "Your Name"
b. git config --global user.email "Your email address"
c. git config --global core.editor vim
d. git config –list
```

Create an empty git repository in your directory: `git init`

Create another directory lab2 in ~/your_roll_CS618

Create bash script files (.sh files) containing the solutions for the questions given below

Stage the new file for tracking: `git add .sh files`

Check the status of your repository: `git status`

`Save a snapshot of your file; that is, commit your changes with a descriptive message, For example:`

```
git commit -m " added solution file for Q1 school.sh."
```

Now, put your repository on GitHub.

a. Visit https://github.com
b. Click “Sign up” with your iitdh email address
c. Follow the instructions to complete setting up your account

On GitHub, create a repository called “`your_roll_no_CS618`”. The new account dialog gives an opportunity, or you can use the “+ > New repository” menu in the upper right corner.

i. Choose “Private”, not “Public”

ii. Leave “Add a README” unchecked.

iii. Leave “Add .gitignore” unchecked.

iv. Leave “Choose a license” unchecked.

v. Click "Create repository. (This will create the repository and the branch will be named as main)

In `your_roll_no_CS618` directory, to tie your local repository to GitHub, run

```
git remote add origin git@github.com:GitID/your_roll_no_CS618.git
```

(after changing GitID to your GitHub ID).

Run `git remote -v` to confirm that your remote has been set up correctly

Set up SSH (Secure Shell) if you want to avoid typing your username and password every time you push or pull

To setup SSH authentication for git, open a new terminal (Ctrl + Alt + t), then follow the steps mentioned in this link https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

(You can ignore the part related to "Generating a new SSH key for a hardware security key").

After generating the ssh key using the steps given in the above link,

Copy the key to the clipboard: gedit ~/.ssh/"your_key".pub followed by ctrl+ c

At GitHub.com, click the top right (profile picture) menu and choose "Settings,"

Click "SSH and GPG keys" in the left-hand menu.

Click "New SSH key," paste your key, and click "Add SSH key."

(You may be prompted for your Git password, in which case you should enter it)

Finally, push your local repository to GitHub: `git push origin master`

(You may get an error message about the authenticity of the host github.com cannot be verified, asking if you want to continue connecting. Say "yes".)

Q1.

Download the Property_Tax_Roll.csv file from https://drive.google.com/file/d/1GjKXoHRrnTS9DtxzjA5d5Yl2TxfZNXpI/view?usp=sharing

Write a bash script, school.sh, that finds the average

TotalAssessedValue for properties in the "MADISON SCHOOLS" district, and echos

that average, and nothing else, onto the command line.

Your script should operate as a single pipeline. Solutions that require writing partial results, for example, by creating “in-between” or “intermediate” files, will not receive full credit.

Hint: Write a pipeline with these stages

• Use cat to write Property_Tax_Roll.csv to stdout. (Or, to work with small input while debugging, use head to write only the first few lines.)

• Use grep to select only those lines containing MADISON SCHOOLS.

• Use cut to pick out the TotalAssessedValue (7th) column.

• Pipe this as the input to a brace-enclosed “group command” that works in a pipe. This one finds a sum:

```
{ my_sum=0; while read line; do
    my_sum=$(( $my_sum + $line ));
done; echo $my_sum; }
```

The `while read x` pattern creates a while loop which, in each iteration, reads the next line from the while-loop’s “stdin” and stores it in a variable $x. Revise the group command above so that it finds a mean instead of a sum. Note that the variable being updated inside the while loop must be declared inside of the curly braces.

Once the while-loop terminates, you’ll have a sum and a number of lines. We need to divide one by the other to compute the average. The division operator / in bash math evaluation performs integer division. That means that, for example `$(( $my_sum / $num_lines ))`, returns an integer (i.e., throws away the remainder), even if `num_lines` does not divide `my_sum`. it’s okay to return the answer rounded down in this case.

Q2. Write a script, digits.sh, to count how many of the numbers between 1000 and 2000 (inclusive) that contain 2 in one or more of their digits, and print the number to standard out.

Hint: read the man page for the command `seq` to see a simple way to generate the numbers 1000 through 2000. Pipe the output of `seq` into `grep`, then to `wc`.

Q3. Write a script ten_dirs.sh that does these tasks:

• make a directory ten

• make ten subdirectories ten/dir01, ten/dir02,... through ten/dir10. Note the formatting dir01, dir02, etc., rather than dir1, dir2, etc. Read the seq man page to see how to do this easily.

• in each subdirectory, make four files, file1.txt through file4.txt, such that file1.txt has one line consisting of the digit 1, file2.txt has two lines, each consisting of the digit 2, file3.txt has two lines, each containing the digit 3, and file4.txt has four lines, each containing the digit 4.

Note that hard-coded solutions (e.g., writing mkdir dir01; mkdir dir02; ...), will not receive any credit. You should use a loop of some kind or another.

Q3. Write a script rm_n.sh whose usage statement is

usage: `rm_n.sh  <dir> <n>` that removes all files in directory dir larger than bytes.

Note: By convention for usage statements, the "`<...>`" delimiters around "`<dir>`" indicate a required argument. Square brackets, "`[...]`", indicate an optional argument.

So `rm_n.sh` takes two required arguments, a directory name and a number. Your script should print a usage message to stderr. You may assume that the second argument, n, is a number, but your script should check that the first argument is a directory, and print an error message to stderr if not.

Something like "ERROR: input is not a directory." is sufficient. Your script should also print an error message to standard error in the event that more or fewer than two arguments are supplied. In the event of either of these errors, your script should exit with exit status 1 (see man exit). You can try out your script on your ten directory from the previous problem by running `rm_n.sh ten 3`

Hint: use the command find.

A couple of other points:

• Make sure that you use the script-name variable $0 in your usage statement, rather than hard-coding the name “rm_n.sh”. This way, the usage statement will be correct even if you change the script name later.

• Because the usage statement is for humans to read, and not for further programs in a pipeline, we write it to stderr. One way to do this is via echo. Normally, echo writes to stdout. Redirect its stdout to go to stderr via “1>&2” as in echo "hello" 1>&2.

Q4. Write a script, `mean.sh`, with usage statement `usage: mean.sh [file.csv]` that reads the columns specified by (a positive number) from the comma-separated-values file (with header) specified by `[file.csv]` (or from stdin if no [file.csv] is specified) and writes its mean . You may assume that the selected column contains numerical data. Here are three example runs:

mean.sh prints the usage statement to standard error and exits with status 1 (because it did not receive the required argument).

mean.sh 3 mtcars.csv finds the mean of the third column of mtcars.csv (File mtcars.csv can be obtained from https://drive.google.com/file/d/1W7t9Wl3vOpRJiiSPZV4PR5hGVE2Q3qY5/view?usp=sharing

cat mtcars.csv | mean.sh 3 also finds the mean of the third column of mtcars.csv. (Here mean.sh 3, with no file specified, reads from stdin.)

mean.sh 3 mtcars.csv foo.txt prints the usage statement to standard error and exits with status 1 (because it received too many arguments).

Hint: One approach processes command-line arguments and then uses a pipeline:

• Use cut to select the required column

• Use tail to start on the second line (to skip the header)

• Use a pattern like that suggested for school.sh above to accumulate a sum and line count. Once the loop terminates, use the sum and count to find the mean, and echo it. Once again, since bash’s division is integer division, you will actually get the mean rounded down, but that is okay.

To handle reading from file.csv or from stdin, one approach is to set a variable file to either the file specified on the command line or to /dev/stdin in the case that the user did not provide file.csv on the command line. Then you can read from the file variable in either case.

Create a new branch called Lab 1 inside the “`your_roll_no_CS618” repo and push all the script files (Q1 to Q5) of the lab 1`

Adding instructor as collaborator

On GitHub, click on `your_roll_no_CS618` repository, then click on “Settings” on the menu under the repository name, and “Manage Access” in the left margin.

- Scroll down and click the green “Invite a collaborator” button.
- In the “Search by username, full name, or email” box, enter my github ID (ga-ananth)
- Click “Add collaborator.

The local assignment working directory should be pushed to your github repo on or before the deadline and should be shared with the instructor before the deadline.

Evaluation Details:

During assignment evaluation, the student should pull the assignment repo from their github repository for evaluation of their soultions and commit time of the assignment in repo will be used for calculating the late submission penalty .