(Bash) Shell Scripts: Exercises

1. Form a group of 3-5 students by moving as you wish and then working with people at your table. (You must work in a group, as collaborating via git/GitHub is impossible by yourself. Today's Group2shell groups are not the same as the Project groups.)

2. Select a group leader (only one per group) who does the following to make a copy of my skeleton repository that belongs to the group:

   (a) Run

   `git clone https://github.com/jgillett-605/linuxExercisesSkeleton ~/linuxExercises`

   to get a repository that contains:

   - five empty shell scripts: `school.sh`, `digits.sh`, `five_dirs.sh`, `rm_n.sh`, `mean.sh`
   - a `group.csv` file that contains only a header line

   (b) Make a new repository on GitHub called `linuxExercises`.

   (c) Invite group members as collaborators.

   (d) Invite our TA, _____, as a collaborator.

   (e) Run these commands (using your GitHub ID for `ID` in the second command):

   ```
   cd ~/linuxExercises
   git remote set-url origin git@github.com:ID/linuxExercises.git
   git push origin main
   ```

   (f) Submit your GitHub URL to Canvas's Group2shell assignment (only one per group). (The URL is `https://github.com/ID/linuxExercises` with your `ID`.)

3. Confirm that your leader submitted the URL to Canvas. Then immediately do the following trivial work on the five scripts and `group.csv`. Each group member must handle at least one file at this stage.

   (a) Make a local copy of your leader's repository by running

   `git clone git@github.com:ID/linuxExercises.git ~/linuxExercises`

   where `ID` is your leader's ID.

   (b) Add the `#!/bin/bash` line (telling the program loader to run `/bin/bash`) to each of the five empty shell scripts.

   (c) Revise `group.csv` so that it includes information on your group members in the line format `NetID,LastName,FirstName`. For example, if Wilma Flintstone (NetID: wflint3) and Charlie Brown (NetID: cbrown71) worked together, their `group.csv` file would be:

   ```
   NetID,LastName,FirstName
   wflint3,Flinstone,Wilma
   cbrown71,Brown,Charlie
   ```

   (d) Use `git add ...`, `git commit -m '...'`, `git pull origin main`, and `git push origin main` to copy these changes to your group leader's GitHub repository.

When all group members are done making these changes, run `git pull origin main` to copy all the changes to your local version. Run this command to quickly inspect all the files:

```
find . -path ./.git -prune -o -print -exec cat {} \;
```

It says "find all files in the current directory, pruning (excluding) the path ./.git, or if not pruned, print the file name and run cat on it." (Or just use emacs or `cat` to inspect the files.)

4. Write (well, revise) `school.sh` to find the average `TotalAssessedValue` for properties in the `"MADISON SCHOOLS"` district.

   (a) Before writing your script, run

   ```
   wget http://pages.stat.wisc.edu/~jgillett/DSCP/linux/Property_Tax_Roll.csv
   ```

   to download that file of 2018 City of Madison property tax data. (I got it from `http://data-cityofmadison.opendata.arcgis.com/datasets/property-tax-roll`.) Do not use git to track this large (138 MB) file.

   If you wish, put the file name `Property_Tax_Roll.csv` in a new file called `.gitignore`. This tells git not to bother you about this untracked file. You could use emacs to make this file; or you could make it with

   ```
   echo Property_Tax_Roll.csv > .gitignore
   ```

   (b) 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 select only the `TotalAssessedValue` (7th) column.
   - Pipe that column to a brace-enclosed compound expression ("group command") that finds the sum and then the average.

   (c) Use `git add ...`, `git commit -m '...'`, `git pull origin main`, and `git push origin main` to copy changes to your group leader's GitHub repository. Do this with each of the rest of your scripts too.

5. Write `digits.sh`, to find the sum of the numbers between 1000 and 2000 (inclusive) having digits only from the set {0, 1}.

   Hint: Use a brace expansion to make the range of numbers, a loop to check each one, and a conditional statement including a regular expression to see whether the digits are in {0, 1}).

   Hint: In emacs, run `M-x sh-mode` to get help with code formatting including indenting.

6. Write `five_dirs.sh` that does these tasks:

   - make a directory `five`
   - make five subdirectories `five/dir1` through `five/dir5`
   - in each subdirectory, make four files, `file1` through `file4`, such that `file1` has one line containing the digit `1`, `file2` has two lines, each containing the digit `2`, ..., and `file4` has four lines, each containing the digit `4`

Use nested loops and elegant code. (That is, do not write a brute-force solution that calls `mkdir` 6 times and has 20 commands to write the 20 files.)

Hint: A convenient way to remove the `five` directory and all its files is `rm -r five` (search the `rm` manual page for `-r` to see what it does), so a convenient way to rerun the scrip several times as you develop it is `rm -r five; ./five_dirs.sh`

7. Write `rm_n.sh` whose usage statement is `usage: ./rm_n.sh <dir> <n>` that removes all files in directory `dir` larger than `<n>` bytes. Try it on your `five` directory via `rm_n.sh five 3`.

   Hint: use `find`. In emacs, do `M-x man Enter find Enter` to check its man page. The page is 1200 lines long—don't read it all. Read about its `-size` argument and search within it for the text "`numeric argument`." Read about its `-type` argument to remove only files. Note:

   - "`rm_n.sh`" in this usage statement should be is specified in your script as `$0`, so that the usage statement will be correct even if you change the script name later.
   - Write the usage statement, which is for humans to read (not for further programs in a pipeline), to `stderr`. One way to do this is via `echo`. Normally it writes to `stdout`. Redirect `stdout` to go to `stderr` via "`1>&2`" as in `echo "hello" 1>&2`.
   - By convention for usage statements, the "`<...>`" delimiters in "`<dir>`" indicate a required argument, and "`[...]`" delimiters indicate an optional argument.

8. Write `mean.sh`, with usage statement `usage: ./mean.sh <column> [file.csv]`, that reads the column specified by `<column>` (a 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. Here are three example runs:

   - `./mean.sh` prints the usage statement to standard error
   - `./mean.sh 3 mtcars.csv` finds the mean of the third column of `mtcars.csv`. (To create the test file `mtcars.csv`, run `Rscript -e 'write.csv(mtcars, "mtcars.csv")'`.)
   - `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`.)

   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 compound expression in braces (`{}`) to initialize a sum and line count, run a `while read` loop to accumulate that sum and line count, find the mean, and `echo` it

   To handle reading from `file.csv` or from `stdin`, I 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 I could read from my `file` variable in either case.

What to turn in (once per group):

Your leader should have already turned in your GitHub URL, above.

To verify your submission, change to a new temporary directory and run `git clone ...` to get a fresh copy of your leader's GitHub repository. Check its `group.csv` file and test its scripts.

All the members of a group will receive the same score based on the state of the leader's GitHub repository at the deadline. Each member is responsible for all the group's work.