Part 1: Git workflow, resolving merge conflicts

Adapted from Casey O'Hara's git activity for EDS 211 with information from NCEAS Learning Hub 10 Collaborating using Git and GitHub & Merge Conflicts

Git tracks changes by noting which lines of code have changed - you can only add, delete, or change (delete then add) a line at a time. When collaborating with others on coding tasks, occasionally two people try to change the same line at the same time. This results in a merge conflict!

Here we will collaborate within our Capstone groups to create and then resolve a merge conflict. Count off in your group.

Setup (team member 1 only)

1. Log in to GitHub and go to your Capstone organization.
2. Create a new repo called "merge_test", with a ReadMe
3. Open RStudio to create a local version-controlled R Project.
4. Create a new RMarkdown document, attach the *tidyverse* package. Delete everything else below the setup chunk, and then create a new chunk. Inside this chunk, write a comment along the lines of "code goes here".

   Could look like this:

   ~~~
   Testing merge conflicts

   ```{r}
   library(tidyverse)
   ```


   ```{r}
   # code goes here
   ```
   ~~~

5. Update default method for integrating merges into repository by running this line of code in the terminal: `git config pull.rebase false` This establishes that the default strategy for pulling is that git will first try to auto-merge the files. If auto-merging is not possible, it will then indicate a merge conflict.
6. Stage everything (.Rmd, .gitignore, .Rproj), commit, pull, push to main. You can do this using the RStudio Git tab button or through the terminal.

Step 1 (all!): Make a change to the base document

Team member 1 can skip steps 1-3.

1. Log in to Github and go to your Capstone organization.
2. Find the new practice repo, and clone to your local computer.
3. Run `git config pull.rebase false` in the terminal.
4. Open up the R markdown.
    a. Erase the comment left by the first person.
    b. In its place, add some code to the chunk, using the dataset **`diamonds`** built into the Tidyverse - a summary stat, a plot, something.
    c. Don't tell your teammates what you're doing!
5. Save, and try to pull (using the RStudio Git tab button or through the terminal). What does it tell you? Consider why!

Step 2 (team member 2): commit a legit change (no conflict)

1. Commit, pull, push your changes. Congrats, you've got the easy part!
2. Do your changes go through? Why or why not?
3. If you want to work through a merge conflict, rewrite your code in the code chunk, and we'll come back to it later.

Step 3 (team member 3): commit and create a merge conflict!

1. Save your work, and try to pull (using the R Studio Git tab button, or through console). What does it tell you? Consider why!
2. Commit your work, then try to pull again. What does it tell you? Read the message carefully!
3. Congrats, you've created a merge conflict!
4. Don't panic!
5. Look at the R Markdown and identify where the conflict has occurred.

~~~
```{r}
<<<<<<< HEAD
### the stuff Team Member 3 tried to do
=======
### the stuff Team Member 2 already did!
>>>>>>> a4b95dc743f0244b11754e9cb8c6385d427eaa9b
```
~~~

7. Decide what to do with the code inside the `<<<<<<<` and `>>>>>>>`. Four options:
    a. Keep your new code! You know what you're doing.
    b. Keep Team Member 2's code! They are pretty smart, so you like what they did.
    c. Combine the best parts of code from each! Communicate about it. You're working great as a team!
    d. Delete it all and write something completely different! You see a way to improve on both members' contributions!

8. Delete the boundary lines around the merge conflict - the lines containing `<<<<<<<`, `=======`, and `>>>>>>>`.
9. Now, resolve it! Save your work, commit (explaining how you resolved it), then pull and push.
10. **NOTE: sometimes, multiple conflicts may have occurred in different places in the same script, or in separate scripts.** You must resolve them all before you can wrap up the merge conflict.
11. Celebrate!

Steps 4, 5, 6 (team members 4, 1, 2): commit and create a merge conflict!

Repeat step 3 in sequence until everyone has had a chance to create and then resolve a merge conflict.

Reflect - how can you avoid or prevent merge conflicts?

Merge conflicts are scary at first, but not so bad once you know how to handle them. However, they are still annoying! As a group, discuss some ideas on how you can avoid or prevent merge conflicts.

===

insert new code chunk:
option + command + i

===

FILTER

~~~
```{r setup, include=False}
library(tidyverse)
```


```{r}
head(diamonds)

ideal <- filter(diamonds, cut == 'Ideal')
head(ideal)
```
~~~

===

GRAPH

```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) +
        geom_point(alpha=0.5, size =0.5)
```