Merge Conflict Activity*

Introduction

You will be working in teams for the group project. We will be using GitHub to write code collaboratively. However, when multiple people are working on one document, conflicts can happen!

Sometimes things will go swimmingly, and sometimes you'll run into merge conflicts. So, before we push you off into the deep end of the wonder that is collaborative programming, our first task is to walk you through a merge conflict!

Merges and Merge Conflicts

- Pushing to a repo replaces the code on GitHub with the code you have on your computer.
- If a collaborator has made a change to your repo on GitHub that you haven't incorporated into your local work, GitHub will stop you from pushing to the repo because this could overwrite your collaborator's work!
- So you need to explicitly "merge" your collaborator's work before you can push.
- If you and your collaborator's changes are in different files or in different parts of the same file, git merges the work for you automatically when you *pull*.
- If you both changed the same part of a file, git will produce a **merge conflict** because it doesn't know how which change you want to keep and which change you want to overwrite.

*Adapted in part from Data Science in a Box

Git will put conflict markers in your code that look like:

```
<<<<<<< HEAD

See also: [dplyr documentation](https://dplyr.tidyverse.org/)

=======

See also [ggplot2 documentation](https://ggplot2.tidyverse.org/)

>>>>>>> some1alpha2numeric3string4
```

The `===`s separate *your* changes (top) from *their* changes (bottom).

Note that on top you see the word `HEAD`, which indicates that these are your changes.

And at the bottom you see `some1alpha2numeric3string4` (well, it probably looks more like `28e7b2ceb39972085a0860892062810fb812a08f`).

This is the **hash** (a unique identifier) of the commit your collaborator made with the conflicting change.

Your job is to *reconcile* the changes: edit the file so that it incorporates the best of both versions and delete the `<<<`, `===`, and `>>>` lines. Then you can stage and commit the result.

Merge conflict activity

! Working Together

This activity does not take very long. However, it is very important that you work on this activity during a Zoom meeting, when you can talk each other through the steps. You can work on the Group Contract in a more distributed way, but it is very important that you can talk to each other during this activity.

Setup

- Clone the group project repo (i.e. open an RStudio project from a GitHub repository in Posit Cloud using the url you get when you accept the group-project assignment, at the link in the Group Contract assignment) and open the .qmd file.
- Assign the numbers 1, 2, 3, and 4 to each of the team members. If your team has fewer than 4 people, some people will need to have multiple numbers.

Let’s cause a merge conflict!

Our goal is to see two different types of merges: first we’ll see a type of merge that git can’t figure out on its own how to do on its own (a **merge conflict**) and requires human intervention, then another type of where that git can figure out how to do without human intervention.

Doing this will require some tight choreography, so pay attention!

Take turns in completing the exercise, only one member at a time. **Others should just watch, not doing anything on their own projects (this includes not even pulling changes!)** until they are instructed to. If you feel like you won’t be able to resist the urge to touch your computer when it’s not your turn, we recommend putting your hands in your pockets or sitting on them!

Before starting: everyone should have the repo cloned and know which role number(s) they are.

Role 1:

- Change the team number to your actual team number (i.e. replace [Replace with Group Number] with 7 if you are Group 7).
- Render, commit, push (remember that you will most likely have to enter your personal access token).

> **!** Important
>
> Make sure the previous role has finished before moving on to the next step.

Role 2:

- Change the team name to some other word.
- Render, commit, push (remember that you will most likely have to enter your personal access token). You should get an error.
- Pull. Take a look at the document with the merge conflict.
- Clear the merge conflict by editing the document to choose the correct/preferred change.
- Render.
- Stage your files – make sure to **check the Stage checkbox** for all files in your Git tab. Make sure they all have check marks, not filled-in boxes.
- Commit and push.

> **!** Important
>
> Make sure the previous role has finished before moving on to the next step.

Role 3:

- Change the a label of the code chunk at the beginning of the document.
- Render, commit, push. You should get an error.
- Pull. No merge conflicts should occur, but you should see a message about merging.
- You *may* have to ok a commit message.
- Now push.

> **!** Important
>
> Make sure the previous role has finished before moving on to the next step.

Role 4:

- Change the label of the first code chunk to something other than previous role did.
- Render, commit, push. You should get an error.
- Pull. Take a look at the document with the merge conflict. Clear the merge conflict by choosing the correct/preferred change. Commit, and push.

> **!** Important
>
> Make sure the previous role has finished before moving on to the next step.

Everyone: Pull, and observe the changes in your document.

Tips for collaborating via GitHub

- **Always pull first before you start working.**
- Resolve a merge conflict (commit and push) *before* continuing your work. Never do new work while resolving a merge conflict.
- Render, commit, and push often to minimize merge conflicts and/or to make merge conflicts easier to resolve.
- If you find yourself in a situation that is difficult to resolve, ask questions ASAP. Don't let it linger and get bigger.

Group Contract

After completing this short exercise, work on filling out the Group Contract. Make sure to read the Group Contract instructions.