**Week 3: The wilder side of git collaboration: co-editing code & dealing with merge conflicts when working in branches**

In the previous two discussions, you learned to work in branches on your own (pushing to a branch & submitting a PR to yourself) and in a carefully curated example with a collaborator (pushing to a branch & submitting a PR that your partner reviews & merges in).   
  
But what might co-working with R and git look like in the wild?

Often, that means you have a primary document (e.g. RMarkdown file), and different people will be working on different features in their own branch, with a plan to eventually merge in their changes to the main branch. But it seems tedious to specify exactly *which line* each person is working in to avoid merge conflicts when multiple branches are getting merged in. In this discussion, we’ll learn how git deals with merge conflicts when we have multiple (and not carefully curated) branches getting merged into main.

**START: ALLISON & CASEY DEMO**

Follow along with Allison & Casey, who will demo the steps below. Then, you’ll work with a partner to practice the same thing in your own collaborative project.

**STEP 1: ASSIGN PARTNERS**

Allison & Casey will assign partners in breakout rooms, designating “Partner 1” and “Partner 2.” Partner 2: share your GitHub username with your partner so they can find & add you as a collaborator.

**STEP 2: PARTNER 1 - PROJECT SET-UP**

* Create a repo in GitHub, with a ReadMe, named **wild-git-collab**
* Clone to create a version controlled R Project in RStudio
* Drop the file ‘git\_wild.Rmd’ into the project root
* Stage, commit, pull, then push to main
* Add your partner as a collaborator to the project on GitHub (Settings > Manage Access)

**STEP 3: PARTNER 2 - ACCEPT THE INVITE & CLONE**

* Accept the invitation to be a collaborator on the project
* Clone to get the project up-and-running locally in RStudio

**STEP 4: BOTH PARTNERS - MAKE & SWITCH OVER TO A BRANCH**

* Make a branch named **yourinitials-branch** (e.g. Allison’s is ah-branch). **Try setting it up in the Terminal** with the following commands, instead of with the RStudio button (replace branch-name below with what your branch name is):

git checkout -b branch-name

git push -u origin branch-name

**STEP 5: BOTH PARTNERS - UPDATE SEPARATE CODE CHUNKS, PUSH, PR & MERGE**

* In the RMarkdown document, find the code chunk with **your partner number** as the code chunk name. Within that code chunk, update the code to complete the commented instructions, which will make changes to your branch from what is in *main*.
* Stage, commit, pull, then push changes, and submit a PR.
* **Merge in your own changes**. Check the updated git\_wild.Rmd file in remote main, and see that your merged updates exist there.
* Note that git just *deals* with these changes - we do not get a merge conflict every time there is a difference from main getting merged in (that would make it pretty tedious to do anything with git, right?)

**STEP 6: BOTH PARTNERS - MERGE UPDATES INTO YOUR LOCAL BRANCH**

Following Step 5, there are now changes that exist in *remote main* that do not exist in your *local main,* or your *local branch*. Get changes from remote into your local branch by running the following two commands in the Terminal:

git pull

git merge origin

Now you should see all updates to remote main appearing in your local branch.

**STEP 7: BOTH PARTNERS - DO SOME UNSUPERVISED EDITING in break-it-1**

In your branch, find the code chunk named **break-it-1**, which contains existing code. **Delete the existing code in that chunk.** Write some random code & comments in the break-it chunk to make a totally different subset or graph (with a different object name) starting from the diamonds dataset. Do not tell your partner what you're doing there.

Once you’ve updated the code, stage, commit, pull, push.

**STEP 8: PARTNER 1 - Submit a PR (there shouldn’t be a conflict here)**

Partner 1, submit a PR, merge it in. There shouldn’t be any issues.

**STEP 9: PARTNER 2 - Submit a PR (there WILL be a conflict)**

Create the PR anyway (you can still submit a PR if there is a conflict), then use the **web editor on GitHub** to **resolve** conflict. This highlights the lines where there is a merge conflict (and surrounds them with alligator mouths). You can choose which one, or keep both! Delete everything but the code you want to keep.

Once you’ve resolved, commit merge.

**STEP 10: BOTH PARTNERS - MERGE UPDATES INTO YOUR LOCAL BRANCH**

Following Step 9, there are now changes that exist in *remote main* that do not exist in your *local branch*. Get changes from remote into your local branch by running the following two commands in the Terminal:

git pull

git merge origin

Now you should see all updates to remote main appearing in your local branch.

**STEP 11: BOTH PARTNERS, DO SOME UNSUPERVISED EDITING in break-it-2**

In your branch, find the empty code chunk named **break-it-2**. Write some random code & comments in the break-it-2 chunk to make a subset or graph starting from the diamonds dataset. Do not tell your partner what you're doing there. Once you’ve updated the code, stage, commit, pull, push.

**STEP 12: PARTNER 2 - Submit a PR & merge in**

There should be no conflicts. Submit your own PR & merge in. Let you Partner 1 know when you’ve done that.

**STEP 13: PARTNER 1 - Submit a PR, resolve conflicts, merge in**

Partner 1, since Partner 2 has updated main \*since\* you last synced it, you will get a conflict. Submit a PR anyway, you will need to resolve it (through GitHub), then commit & merge after resolving.

**STEP 14: BOTH PARTNERS - MERGE UPDATES INTO YOUR LOCAL BRANCH**

Following Step 13, there are now changes that exist in *remote main* that do not exist in your *local main,* or your *local branch*. Get changes from remote into your local branch by running the following two commands in the Terminal:

git pull

git merge origin

Now you should see all updates to remote main appearing in your local branch. YAY!

**DONE WITH DISCUSSION 3**