

Git branches and pull requests

Adapted from Casey O'Hara's git activities for EDS 211 [\[1\]\[2\]](#).

Branches are a great GitHub feature that allows you to work on a project independently while maintaining a functional `main` branch. For example, your Shiny app is working fine, and you want to keep it up and running, but you'd like to safely try a modification without the risk of breaking the whole app.

In this lesson we will also explore **pull requests**, which encourage users to review changes that happened in a branch before incorporating them into the main branch. Some other Git vocabulary we'll use:

- **upstream** is where you cloned for, this is the **origin** repository in GitHub
- **HEAD** is a pointer indicating where in your Git tree you are currently working
- a **local branch** is a branch that exists in your local repository, it may or not be tracked by a **remote-tracking branch** upstream

Activity 1: branching with upstream (no merge conflicts)

Count off within your group to set who is going to do which step. Unless otherwise specified, all the commands given should be run on the terminal.

Step 1 (team member 1): Set up a test repo

1. Log in to GitHub and go to your Capstone organization.
2. Create a new repo called "branches-test" with a README
3. Using RStudio, clone the repository to create a local version controlled R Project.
4. Create a new RMarkdown document called test.rmd, attach the *tidyverse* package.
5. If needed, delete everything else below the setup chunk, and then create a new chunk for each person in the team.
6. Inside each chunk, write a comment along the lines of "code goes here".

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

```{r}
code team member 1
```
```

```
```{r}
code team member 2
```
```

7. Update default method for pulling into repository by running this line of code in the terminal:

```
Unset
git config pull.rebase false
```

Remember 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.

8. Stage everything (.Rmd, .gitignore, .Rproj), commit, pull, push to main.

Step 2 (everyone except team member 1): Set up repository

1. Clone the branches-test repository. Do not modify the Rmarkdown file.
2. Run `git config pull.rebase false` in the terminal to update the default pull method.

Step 3 (everyone): View current branches

1. We can list the *local* branches within our repository using `git branch`:

```
Unset
git branch
-----
* main
```

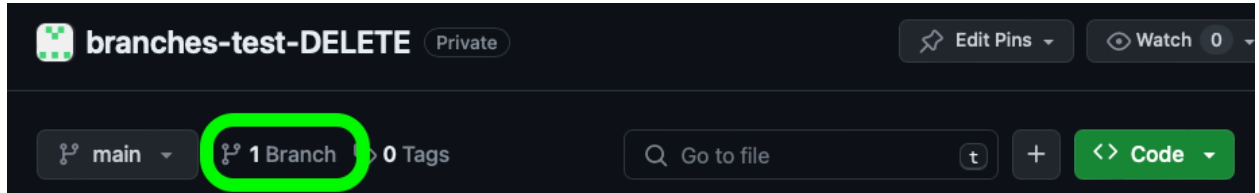
An asterisk to the left of the branch name indicates this is the branch we're currently on.

2. We can list both the *local and remote-tracking branches* in the repository using `git branch -a`:

```
Unset
git branch -a
-----
* main
remotes/origin/HEAD -> origin/main
remotes/origin/main
```

The line `remotes/origin/HEAD -> origin/main` means that the default branch for the remote `origin` repository is the branch `origin/main`.

3. If you go back to your GitHub repository, you'll see there is only one branch, namely the remote branch that is tracking your local `main` branch.



Step 4 (everyone): Create and checkout a new branch

1. Type the following to create a new branch called name `<your-initials>-branch`. For example, I'll call mine `cgg-branch`. To verify we created a new branch we can list all the git branches again:

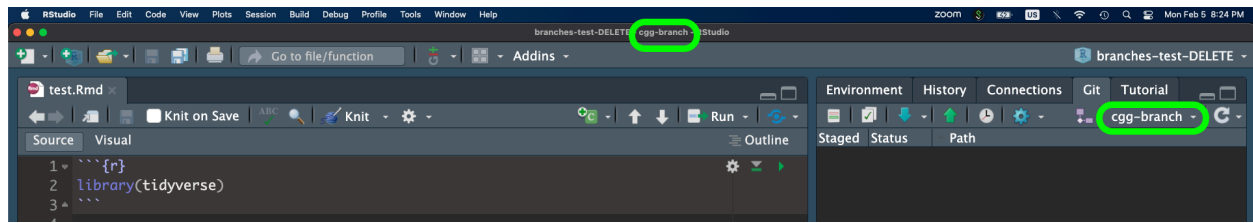
```
Unset
git branch cgg-branch
git branch -a
-----
    cgg-branch
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
```

Notice that although we created the branch, we are still on the `main` branch.

2. Move to your new branch using the `git checkout` command. Notice the asterisk moved from `main` branch to your branch, that's how we know we are in the `cgg-branch` branch:

```
Unset
git checkout cgg-branch
git branch -a
-----
* cgg-branch
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
```

On RStudio, the branch you're now in shows up in the top bar of RStudio, as well as in the top right of the Git tab. Sometimes it takes RStudio a while to update this - **always verify which branch you're in using the terminal.**



Step 5 (everyone): Connect local branch to remote

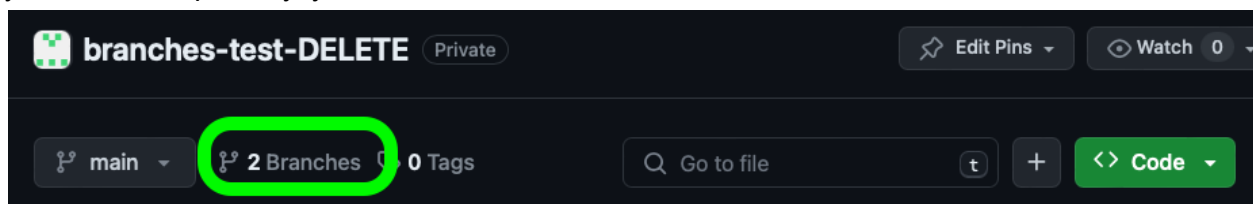
1. Let's connect the branch you just created to a remote branch (upstream branch):

```
Unset
git push -u origin cgg-branch

git branch -a

-----
* cgg-branch
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/cgg-branch
  remotes/origin/main
```

Notice we now have a new upstream branch `remotes/origin/cgg-branch`. If you go back to your GitHub repository, you will see the new branch there:



Step 6 (everyone): Update code and push to your remote branch

For this step **make sure you are working on your branch.**

1. **In your code chunk**, update the test.Rmd file by making a visualization of the *starwars* dataset included in tidyverse. For example, create a scatter plot of character mass versus height:

Unset

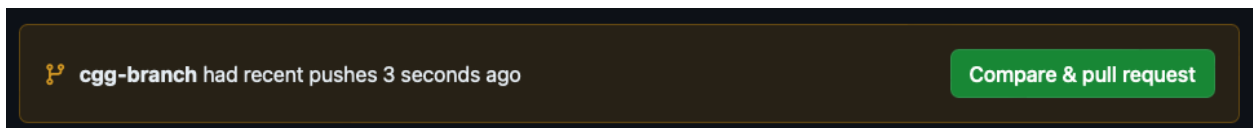
```
starwars %>% filter(species == 'Human') %>%  
  
  ggplot(aes(x=height, y=mass)) +  
  
  geom_point()
```

2. Stage, commit, pull, push. When working on a real project you would keep working here for a while until you've implemented a feature, made updates, or fixed a bug.

Stage 7 (one team member at a time): Merge your remote branch with origin/main

This step happens completely on GitHub.

1. Go back to GitHub & refresh your repo, and you will *probably* see a highlighted message that looks something like this:



2. Click on the "Compare & pull request" button. This will take you to a page where you can **open a pull request**. Here, you can compare the differences between your branch and the main branch (scroll down). You should write a title for your pull request and a brief description of what you accomplished in your branch. You can also assign someone to review your code before incorporating it into the main branch.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

base: main ← compare: egg-branch ✓ **Able to merge.** These branches can be automatically merged.

Add a title

Created graph of Star Wars humans height and mass.

Add a description

Write Preview H B I

Created graph of Star Wars humans height and mass. We could still discuss final colors for plot.

Markdown is supported Paste, drop, or click to add files

Reviewers

No reviews

Assignees

No one—[assign yourself](#)

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Use [Closing keywords](#) in the description to automatically close issues

Helpful resources

Create pull request ▾

- Click on the green “Create pull request” button. This will take you to another page where you start a pull request and, after discussion with your team if necessary, you add the changes in your branch to the main branch by merging. You should be able to merge the branch with no conflicts (more on conflicts that later):

✓ **This branch has no conflicts with the base branch**
Merging can be performed automatically.

Merge pull request ▾

Click the “Merge pull request” button. That’s it - we merged the branches!

- Optional: Once the pull request is closed, you can let GitHub delete the branch if you won’t be using it anymore.

Pull request successfully merged and closed

You’re all set—the `egg-branch` branch can be safely deleted.

Delete branch

- Repeat 1-4 until all team members have merged their branch into main.

Step 8 (everyone): Update your local main branch

1. Go back to RStudio. To get the updates from the merged remote main branch we need to move to our local main branch and retrieve the upstream changes:

```
Unset  
git checkout main  
  
git pull origin
```

If you check your test.rmd document (on the main branch) you should see the code from all your teammates. If you want to check how your git tree looks after the merges, run the following in the terminal:

```
Unset  
git log --graph --oneline --all
```

Congratulations! You're now working with Git branches!

Here's a summary of the commands we've used and some extra ones:

| Command | What it does |
|---|--|
| <code>git config pull.rebase false</code> | establishes auto-merge as the default strategy for pulling. You only need to run this once per local repository. |
| <code>git branch</code> | list local branches |
| <code>git branch -a</code> | list all branches (local and remote) |
| <code>git branch new-branch-name</code> | create a new local branch with name new-branch-name |
| <code>git checkout branch-name</code> | move into branch-name branch |

| | |
|--|---|
| <code>git push -u origin branch-name</code> | connect branch-name with a remote branch in upstream repository |
| <code>git pull origin</code> | pull changes from default branch in remote origin into current branch |
| <code>git log</code> | See full commit history. Press q to exit. There's all sorts of options for this git command . |
| <code>git log --pretty="%h - %s"</code> | A nice view of the commit history with only the abbreviated hash and subject |
| <code>git log --graph --oneline --all</code> | A view of the commit history as a git tree. |
| <code>git branch -d branch-name</code> | Delete local branch named branch-name |
| <code>git remote prune origin</code> | Delete tracking of remote branches in origin that have been deleted |

Activity 2: resolving merge conflicts when merging into upstream

Setup

We'll continue working in our branches-test repository. Resolve any lingering Pull Requests related to this repo.

Step 1 (team member 2): Set up a new .rmd

1. **Verify you are working in the main branch of the repository.**
2. Create a new R markdown called something like `pr_merge_test.Rmd`
3. Attach the `tidyverse` and, if needed, delete all below the setup chunk.
4. Write a brief code chunk to perform some analysis and data viz related to the built-in `diamonds` dataframe. For example:

Unset

```
ggplot(diamonds, aes(x = carat, y = price)) +  
  geom_point(aes(color = clarity))
```

5. Save, commit, pull, push.

Step 2 (everyone except team member 2): Pull changes

1. **Verify you are working in the main branch of the repository.**
2. Pull changes from upstream. You should see the new pr_merge_test.Rmd file. **Do not modify anything in it.**

Step 3 (everyone): Create a branch and update the code


1. Create a new branch named <initials>-update and move into it. **Confirm you are in your new branch.**
2. Connect your new branch with a remote branch in the upstream repository. **Confirm your branch is being tracked.**
3. Within your new branch, open the pr_merge_test.Rmd file.
4. Modify the code inside the existing chunk in some fashion to update or override the current code. Don't tell others what you are doing - pretend you are a poorly functioning team that doesn't communicate well - we're trying to break things, remember?

Reflect: Will you create a merge conflict when you commit-pull-push your changes? Why or why not?


5. Save your work, commit, pull, push to GitHub.

Step 4 (everyone): Return to GitHub and examine pull requests

If all goes according to plan, you should see several pull requests, one from each team. Don't click on anything yet!

 **cgg-update** had recent pushes 14 minutes ago

[Compare & pull request](#)

 **other-update** had recent pushes 4 seconds ago

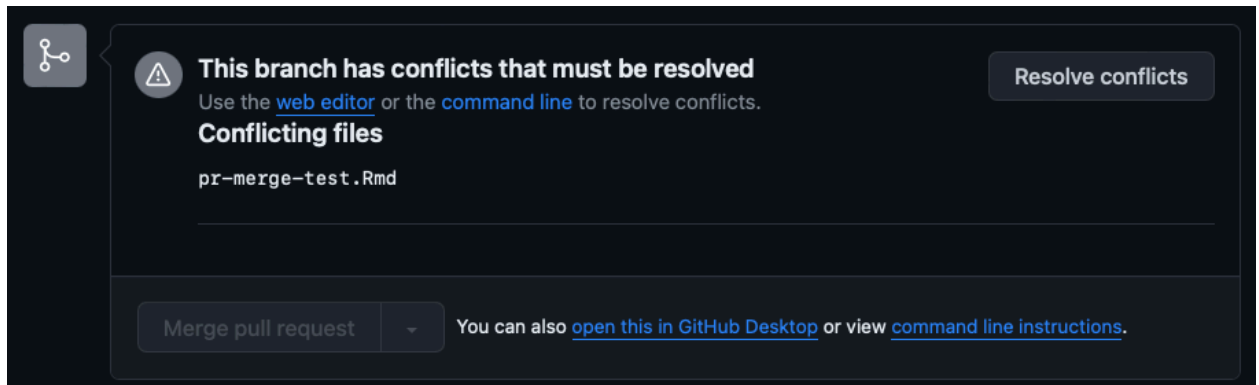
[Compare & pull request](#)

Step 5 (team member 3): Resolve Pull Request

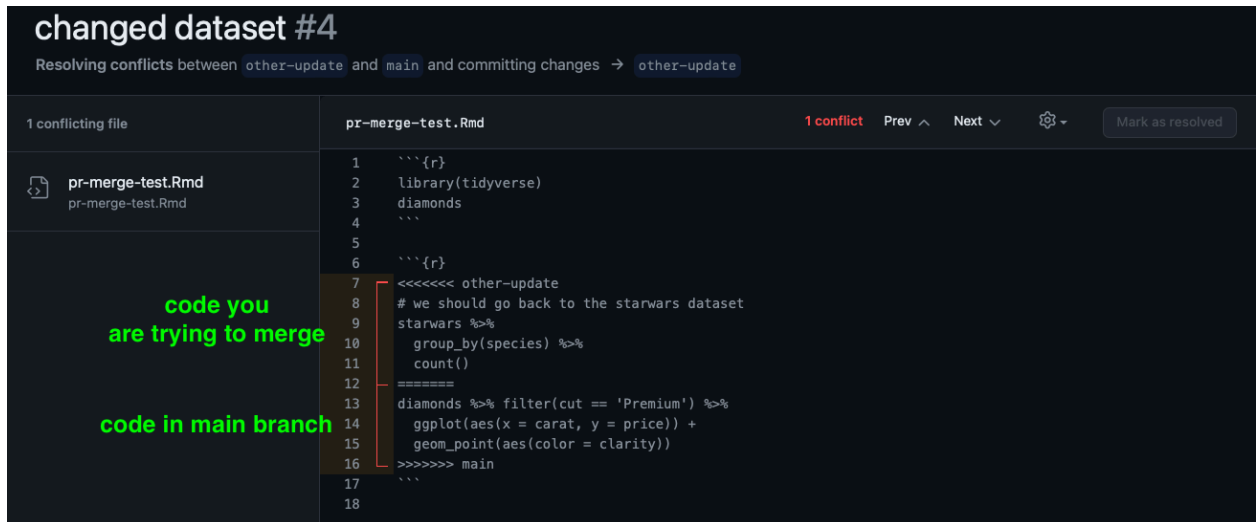
1. Compare and create a pull request on your changes. Will it automatically allow you to merge? Why or why not?
2. Write a message in your Pull Request text box, then submit your pull request.
3. Merge your pull request with the main branch.

Step 6 (remaining team members, one at a time): Merge conflict

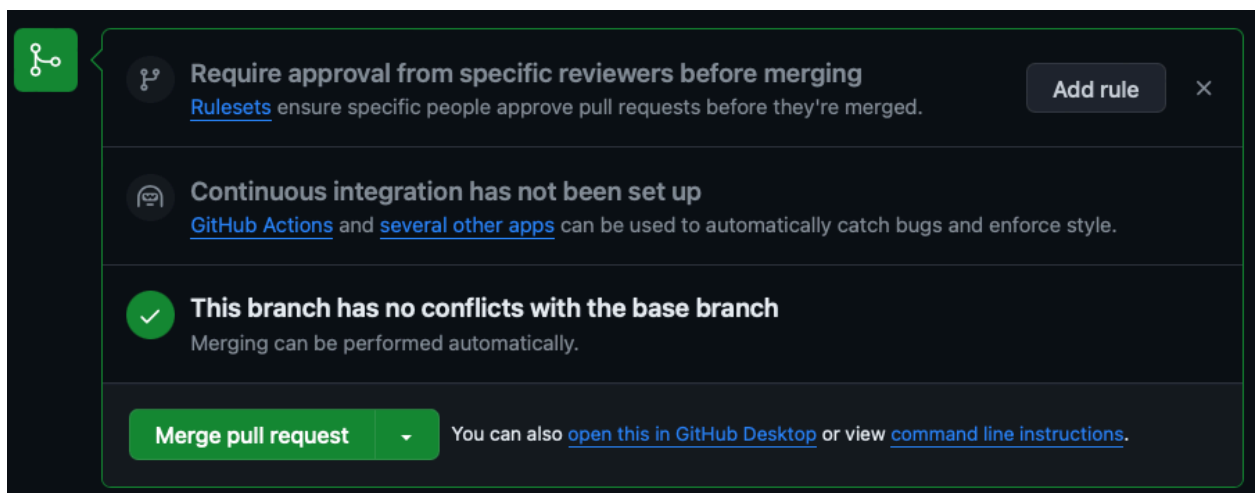
1. Click on “Compare & pull request” for your changes. Will it automatically allow you to merge? Why or why not?
2. Write a message in your Pull Request box describing your changes and why you did them.
3. Submit your Pull Request. You’re not done yet! There appears to be a merge conflict. Click on the “Resolve conflicts” button; this time you will resolve it right in the web browser.



4. You should enter into a text editor. Remember the merge conflict appears in between the <<<<<<<<<<<< and >>>>>>>>>>>> lines. Fix the merge conflict. Remember you should discuss with your team what to keep. Then click the “Mark as resolved” button and “Commit merge.”



- Now you should be back in the pull request page, you should see there is no conflict and you can smoothly merge your branch into the main branch. All done!



- Repeat this step until all team members have resolved a Git conflict when merging their branches.

Step 7 (everyone): Update your local main branch

- Go back to RStudio. To get the updates from the merged remote main branch we need to move to our local main branch and retrieve the upstream changes:

```
Unset
git checkout main
git pull origin
```

If you check your test.rmd document (on the main branch) you should see the updated code you all agreed on.

That's it!