GitHub for Public Health

Corinne Riddell and Lauren Wilner

2024-04-19

Contents

Chapter 1

About this training

(insert abstract here)

1.1 Target audience

(insert target audience)

1.2 Questions? Comments?

Contact us @ (insert contact info).

Chapter 2

Welcome to the workshop!

This pre-workshop guide is designed to walk you through the initial setup of git on your computer.

If you have any issues, please reach out to us at wilnerl@uw.edu or c.riddell@berkeley.edu.

2.1 Introduction

Git is a version control system that allows you to track changes in your code.

In order to get setup, we need to install git on your computer, make a GitHub account, and configure git on your computer.

We will get into the details of how to use git in the workshop, but are requesting that you complete the following steps before the workshop so that we can spend all of the workshop time on using Git rather than on setup!

2.2 Step 1. Install R and RStudio if you have not already:

- https://posit.co/download/rstudio-desktop/

2.3 Step 2. Open RStudio and install the following packages:

- **tidyverse**
- **usethis**
- **gitcreds**

- **broom**

To do this, run the following code in the RStudio console: `install.packages('tidyverse')` Do this for each of the packages.

If any of these packages fail to install, please let us know. You may need to update your RStudio but we will try to help you get everything set up with minimal interruption to your other work.

2.4 Step 3. Create a github account

- https://github.com/

2.5 Step 4. Install Git

Follow the instructions on the following slides to install git on your computer. Select the slides that correspond to Windows or Mac depending on what machine you are using.

2.5.1 Windows Instructions

1) Download Git for Windows from here: https://gitforwindows.org/. There are lots of things to click through.

Notes from HappyGitWithR: NOTE: When asked about "Adjusting your PATH environment", make sure to select "Git from the command line and also from 3rd-party software". Otherwise, we believe it is good to accept the defaults. Note that RStudio for Windows prefers for Git to be installed below C:/Program Files and this appears to be the default. This implies, for example, that the Git executable on my Windows system is found at C:/Program Files/Git/bin/git.exe. Unless you have specific reasons to otherwise, follow this convention.

2) Start menu > Git > Git Bash. Confirm that you have access to Git Bash.

3) RStudio next RStudio should automatically detect the presence of Git Bash. You can inspect and influence this directly via Tools > Global Options > Terminal. Unless you have good reason to do otherwise, you want to see "Git Bash" in the "New terminals open with …" dropdown menu.

4) The next set of tasks are done in RStudio. The code to run these commands are found in the file `code/00_Setup-instructions-for-Windows.R`. After running all those commands you are ready to use GitHub from the Shell. (Currently, these instructions do not include setting up a Git client.)

2.5.2 Mac Instructions

1) Type git –version in your terminal to check if git is installed. If it is, you will see a version number. If not, type: `git config` and then `enter`. You will be prompted to install Git and follow the prompts!

2.6 Configure Git using an HTTPS token

Load `usethis` & `gitcreds`:

- `library(usethis)`
- `library(gitcreds)`

Pick a user name - it does not need to be your GitHub user name. This is the email address linked to your GitHub account:
`use_git_config(user.name = "Corinne Riddell on Dell", user.email = "corinne.riddell@gmail.com")`

After you have installed the packages, you will need to create a personal access token. This is a way to authenticate yourself with GitHub. You will need to do this in order to push and pull from your repository. Run the following:

```
usethis::create_github_token()
```

This will bring you to a browser page. Put in a description for your token and then select an expiration date from the drop down - please select `No expiration`. Scroll down and click the `Generate token` button. Copy the token that is generated and paste it somewhere where you will be able to access it.

Go back to R and run the following:

```
gitcreds::gitcreds_set()
```

When prompted, paste in the token you copied. This will add your credentials to your cache. The following will print out to the RStudio console:

```
? Enter password or token: ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
-> Adding new credentials...
-> Removing credentials from cache...
-> Done.
```

2.7 Resources

You should now be set up to use Git and Github! If you had any issues, here are a few links you can look at for help. If you are still having trouble, please reach out to us before the workshop!

- https://happygitwithr.com/

- https://git-scm.com/book/en/v2/Getting-Started-About-Version-ControlLinks

We look forward to seeing you at the workshop!

Chapter 3

Why Git and GitHub

3.1 What is version control, Git, and Github?

- **Version control** is the practice of tracking and managing changes to (statistical) code and other files.
- **Git** is a version control system. It tracks what is changed in a file, when and by whom and synchronizes the changes to a central server so that multiple contributors can manage changes to the same set of files (Wilson et al., 2017).
- **GitHub** is a hosting service on the web for Git repositories. Reference: Wilson G, et al. Good enough practices in scientific computing. PLoS Comp Bio. 2017.

3.2 The case for version control

3.2.1 Error reducing

- Version control **eliminates** the need to send code or outputs (graphics, reports) via email or share folders between collaborators. With a version control system, everyone has access to the most recent set of files.

3.2.2 Version control facilitates reproducible analyses

- Have you ever tried to reproduce an analysis you did 3 years ago?
- Have you ever tried to reproduce someone else's analysis?
  - Their links don't work on your computer
  - You may or may not have access to the data

3.2.3 Version control facilitates reproducible analyses

- Because everyone has access to the same files, a project's workflow can be set up to ensures that the analyses are reproducible for everyone.
- In R, this can be as simple as hitting the "knit" button to run the analyses on anyone's computer – no need to update the file pathways, no need to download new versions of the code or data.

3.2.4 Makes supervision and collaboration easier

- With Github, you can easily view changes made to statistical code. So if you are working together, it is easy to tell what lines of code were changed, alongside downstream changes to reports or data visualizations as a result of the change to the analysis.

3.2.5 Rollback capabilities

- You can use Git to roll back to a previous version of a file at any point. To do this, you would search for the `commit ID` of the version you want to revert to and then use the `git checkout {commit ID}` command to revert to that version. This is useful if your team decides that a change made to the code was not beneficial or wants to revert back to a different strategy that was used previously.

3.2.6 Github supports expeditious sharing of scientific approaches and findings

- Anything posted on GitHub can be shared widely to your organization or with the public.

Chapter 4

Git Jargon (Alice Bartlett)

Write a blurb about Alice's slides, link her slides, show screenshot of first slide.

Chapter 5

Setup of Solo Workflow

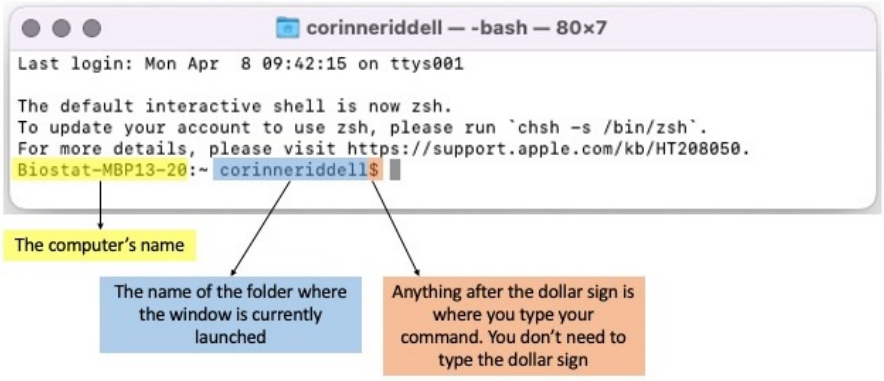
5.1 Outline for this section

We have discussed why Git and GitHub are important, now we will set up a repository and work through an example. During this first section, you will be working in a repository alone. We will:

- Make a new repository on GitHub
- Clone the repository to your local machine
- Write code in your repository locally
- Push the code to your repository
- Merge your branch into main

5.2 Working in Terminal (Mac) and Bash (Windows)

- We will use Terminal and Bash applications to interact with Git on our laptops.
- Below is a Mac Terminal window. It looks very similar to a Windows Bash window.



- In this session, we will supply you with Git code for you to manually input into your Terminal/Bash windows. Note that all of this code will be written after the dollar sign in the figure above.

5.3 Working in Terminal (Mac) and Bash (Windows)

- Here is an example of the command `git branch`, followed by the output printed to screen:

5.4 Folder names and file names tips

- Recall when you name a variable in SAS or R, the variable name cannot contain spaces or unusual characters.
- It is best practice to not use spaces or unusual characters in folder or file names, even though spaces are permissible and commonly used by Windows and Mac Users.

5.5 Good folder names...

- Use dashes in place of spaces
- Use capitalization instead of spaces

5.5.1 Good folder names examples

- life-expectancy
- lifeExpectancy
- LifeExpectancy

5.6 What is the problem with spaces, anyway?

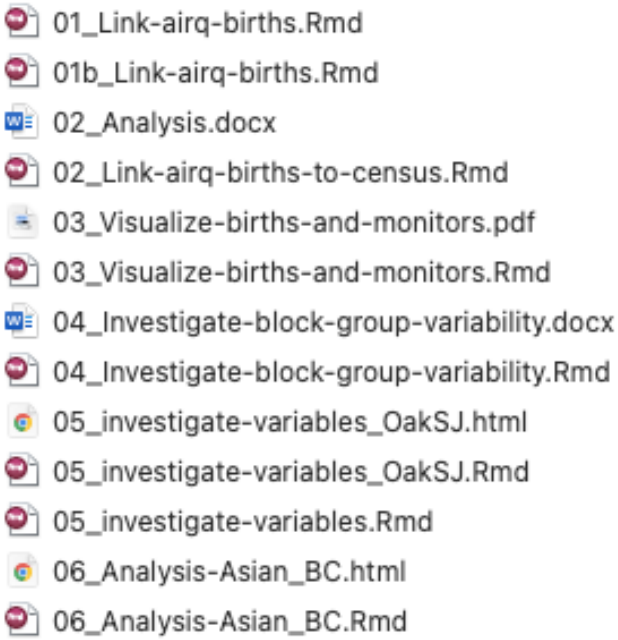
- While spaces are human-readable they aren't machine-friendly.
- When you refer to a folder or file using Git in Terminal or Bash, a name without spaces is much easier to type (otherwise you have to insert a backslash before the space)
- Spaces also break the auto-complete function that Git users love. This is a very frustrating experience.

5.7 Good file names are...

- machine readable
- human readable
- play well with default ordering

See the file "how-to-name-files.pdf" for a more deeper dive into file naming.

5.7.1 Good file names examples



- Filenames start with a number (padded by 0) to order the files according to the order performed in the analysis
- This is followed by a short (human and machine readable) descriptor of what the file does
- Uses underscore "_" to delimit field, and dashes "-" to separate words within field

5.7.2 Bad file name example (and associated Git pain)

- Can see that there is an R markdown (Rmd) file named "Data Visualization Evaluation Report" that has been modified.
- The pain arises when I go to `git add` the file. Before each space, I need to include a backslash (which looks ugly). Even worse, the space breaks the auto-complete that happens when I press "tab" to auto-complete the file name. Auto-complete will become your friend when you use Git, and not being able to use it is very sad/infuriating when you have grown to love it.

```
Biostat-MBP14-02:course-evaluations corinneriddell$ git status
On branch cr-apr8
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   reports/Data Visualization Evaluation Report.Rmd
        modified:   reports/Data-Visualization-Evaluation-Report.pdf

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .DS_Store
        .gitignore
        code/ReadLinkedData.html
        data/.DS_Store

no changes added to commit (use "git add" and/or "git commit -a")
Biostat-MBP14-02:course-evaluations corinneriddell$ git add reports/Data\ Visualization\ Evaluation\ Report.Rmd
```

5.8 Set up a project that you want to track with git and GitHub

Let's suppose that you have a project that you want to start tracking using Git and Github. For this project, you are already working on with some code, data, and visualizations that have already been saved. We have made this project for you. To download it, run the following two commands in the RStudio console:

```
install.packages("usethis")
usethis::use_course("corinne-riddell/existing-project")
```

- R will asked you if you want this folder copied onto the Desktop. Select Yes.
- R will display messages showing you that the folder has been downloaded and unzipped. Tell R whether to delete the file.
- RStudio will then open. Click the code folder in the file viewer. Then, click the filename "01_Analyze-life-expectancy.R" to open this file in RStudio.
- Run all the code in the .R file you just downloaded. Note that it created a figure and saved it into the images sub-folder.

Now you are set up with some existing code and things that you might want to start tracking on GitHub. The next thing to do is make a folder on GitHub that will store this project.

5.9 Make a new repository on GitHub

- Go to github.com and log in. Click the green "New" button to make a GitHub repository. Type "life-expectancy" in the repository name.
- Write whatever you want in the description. For example, type "An analysis of life expectancy in the US"
- Select either to make this a public or private repository.
- Check the box next to `Add a README file`. This tells Git to create a file that can describe your project. For now, you can write a sentence about this being a practice repository for this workshop.

- Choose `.gitignore` template: `R`. This tells Git to use defaults that work well for R users.

- Choose `MIT License`. This relates to the licensing for your code, which is not relevant for this workshop.

- Click "Create repository". Github will then bring you to the repository's main page.

5.10 Clone the repository to your local machine

- From the main page of your repository and click on the green `Code` button.

- You'll see a URL that starts with https://. Push the icon with two overlapping squares to copy the URL to your clipboard. [Lauren, you had written this differently and had them use the SSH one… is there a reason to use one over the other?]
- Open terminal (Mac) or bash (Windows) program. Navigate to where you want to place this repository using the `cd {folder_name}` command. Then write `git clone {paste the url you copied here}` and then press the return/enter button. The following will display in Terminal/Bash if this was successful:

```
Biostat-MBP13-20:repos corinneriddell$ git clone https://github.com/corinne-riddell/li

Cloning into 'life-expectancy'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
```

You now are ready to begin tracking changes to this folder using Git and GitHub.

5.11 Get oriented to the new directory

To get yourself oriented, do the following in the Terminal/Bash window:

- Navigate into your repository by typing `cd life-expectancy/`.
- Type `git status`. The results shows you that no changes have been made yet:

```
Biostat-MBP13-20:life-expectancy corinneriddell$git status
On branch main
Your branch is up to date with 'origin/main'.
```

```
nothing to commit, working tree clean
```

- Type `git branch`. This shows you that you are currently on the main branch.

```
Biostat-MBP13-20:life-expectancy corinneriddell$git branch
* main
```

5.12 Make your first branch

Set yourself up in a new branch off of main. In terminal/bash:

- Type `git checkout -b may3-XY`, replacing XY with your initials. (If today is not May 3, replace "may3" with today's date.)

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git checkout -b may3-cr
Switched to a new branch 'may3-cr'
```

- Type `git branch`, to confirm to yourself that you have indeed switched to the new branch.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git branch
  main
* may3-cr
```

5.13 Make some changes to your code

Okay, you are now set up to track changes. Let's do the following:

- Copy the code/ data/ and images/ sub-folders from your "existing-project" folder into the "life-expectancy" folder.

5.14 Commit the changes that you made and push them to GitHub

- Go back over to terminal or bash. Type `git status`. The output will tell you what has been changed. It tells us that there are untracked files:

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch may3-cr
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store
    code/
    data/
    images/
    life-expectancy.Rproj
```

We want to track the code/, data/ and images/ subfolders we just copied over, as well as the life-expectancy.Rproj file.

Use `git add` to add the newly-added files to be tracked. Then use `git status` to confirm you have added everything you want to track:

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git add code/
Biostat-MBP13-20:life-expectancy corinneriddell$ git add data/
Biostat-MBP13-20:life-expectancy corinneriddell$ git add images/
Biostat-MBP13-20:life-expectancy corinneriddell$ git add life-expectancy.Rproj
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch may3-cr
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   code/01_Analyze-life-expectancy.R
    new file:   data/Life-expectancy-by-state-long.csv
    new file:   images/ca-black-women-LE.png
    new file:   images/placeholder.md
    new file:   life-expectancy.Rproj

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store
```

Note: Computers create files that we don't want to track. For example, Macs create .DS_Store files. Another example is that Windows creates temporary files when a Word doc or Excel spreadsheet is open. You will see these weird files listed under the Untracked files list. You don't need to worry about them because we don't want to track changes to any of those files.

- Commit these changes locally: `git commit -m 'your commit message'`, replace 'your commit message' with a short message about what you've done (keep the quotes around the message). For example, your message could be something like `git commit -m "added first set of files"`.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git commit -m "added first set of file
[may3-cr 58fcc58] added first set of files
5 files changed, 7253 insertions(+)
create mode 100644 code/01_Analyze-life-expectancy.R
create mode 100644 data/Life-expectancy-by-state-long.csv
create mode 100644 images/ca-black-women-LE.png
create mode 100644 images/placeholder.md
create mode 100644 life-expectancy.Rproj
```

- Push these changes to GitHub: `git push origin your-branch-name`, replacing your-branch-name with the name of your branch. If you don't remember your branch's name, type `git branch` to print it to the screen and then the `git push` command.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git push origin may3-cr
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (10/10), 136.50 KiB | 10.50 MiB/s, done.
Total 10 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'may3-cr' on GitHub by visiting:
remote:      https://github.com/corinne-riddell/life-expectancy/pull/new/may3-cr
remote:
To https://github.com/corinne-riddell/life-expectancy.git
 * [new branch]      may3-cr -> may3-cr
```

You have successfully pushed your changes to GitHub!

5.15 Merge the changes from your branch into main

- Navigate to GitHub.com to your repository's URL. There should be a pale yellow banner informing you about the changes you just pushed. Click the button "Compare & pull request". Notice that the title is your commit message from the previous step. Scroll down. Look at the files that have been added.
  - The code is all shown in green, indicating that every line of code is new.
  - The csv data file has been added but is not rendered because it is large
  - The png file is displayed.
- Click on the green "Create pull request" button. Github will check that it is able to merge your branch with main without problems. Note the message "This branch has no conflicts with the base branch". This means you are good to go!
- Click on the green "Merge pull request" button.

- Click on the green "Confirm merge" button.
- Click the "Delete branch" button.

5.16 Summary

- You setup a folder on your laptop so that Git is used to track changes made *locally on your laptop*.
- You linked that folder to GitHub.com so that the same changes can be tracked externally on GitHub.

- You compared changes you made locally on your branch to the main branch on GitHub and pulled your changes into main. This means that the main branch has been updated with your changes *on GitHub*.

5.17 Tracking changes

Now suppose a few days have gone by and you are ready to work on your analysis project. In particular, you want to update some code that will affect some of the results and "outputs", where outputs are results saved in any form. In this section, we outline the process to follow when you want to implement some set of tracked changes.

5.18 Get setup for a new day of work

The first thing we need to do is make sure we are in a good place with git and GitHub:

- Open up bash or terminal and navigate to the life-expectancy folder using the `cd` command.
- Check which branch you are currently on using `git branch`. All local branch names are displayed. The asterisk is next to the branch we are currently on.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git branch
  main
* may3-cr
```

- Check if you forgot to save anything from last time using `git status`. Ideally, you have saved all your changes and there is nothing to add/track/commit. Here is my status:

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch may3-cr
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store

nothing added to commit but untracked files present (use "git add" to track)
```

There is one untracked file: .DS_Store. This is okay since it is an internal file used by Mac OS that does not need to be tracked. We just want to ensure no code files or outputs we intended to track have been forgotton.

- You are likely still on your branch from the last day. In that case, navigate back to main using `git checkout main`.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git checkout main
Switched to branch 'main'
```

```
Your branch is up to date with 'origin/main'.
```

If you want to double check, type `git branch` to confirm you are on main.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git branch
* main
  may3-cr
```

You can also delete the "may3-XY" branch since you no longer need to track it locally:

```
Biostat-MBP13-20:github-training corinneriddell$ git branch -d may3-cr
Deleted branch may3-cr (was 761bb97).
```

CORINNE TODO: ON HOME COMPUTER DELETE THE MAY3 BRANCH FROM THE LE REPO SO CAN GRAB THE CORRECT CODE AFTER "WAS" AND REPLACE IN THE ABOVE.

5.19 Pull down the changes from main

This is your **local** copy of main. It needs to pull down the changes to made that you made on GitHub in an earlier step. To do that, type `git pull origin main`. A graphic will be drawn that summarizes which files have been updated and by how much.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git pull origin main
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From https://github.com/corinne-riddell/life-expectancy
 * branch            main       -> FETCH_HEAD
   884028e..5f81e34  main       -> origin/main
Updating 884028e..5f81e34
Fast-forward
 code/01_Analyze-life-expectancy.R      |   38 +
 data/Life-expectancy-by-state-long.csv | 7201 ++++++++++++++++++++++++++++++++
 images/ca-black-women-LE.png           |  Bin 0 -> 74550 bytes
 images/placeholder.md                  |    1 +
 life-expectancy.Rproj                  |   13 +
 5 files changed, 7253 insertions(+)
 create mode 100644 code/01_Analyze-life-expectancy.R
 create mode 100644 data/Life-expectancy-by-state-long.csv
 create mode 100644 images/ca-black-women-LE.png
 create mode 100644 images/placeholder.md
 create mode 100644 life-expectancy.Rproj
```

5.20 Start a new branch

- Like the last day, start a new branch to track today's changes. Let's pretend it is now May 7. Type `git checkout -b may7-XY`, where XY is replaced with your initials. Type `git branch` to confirm you have changed branches.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git checkout -b may7-cr
Switched to a new branch 'may7-cr'
Biostat-MBP13-20:life-expectancy corinneriddell$ git branch
  main
* may7-cr
```

5.21 You are now ready to make changes!

Think about what you would like to do in advance. In particular, suppose you want to:

- make a table that summarizes the mean life expectancy by race and gender for each state and,
- save the above table as a CSV file into the data folder.
- Re-launch RStudio by double-clicking the .Rproj file in your file viewer window. Navigate to the code file 01_Analyze-life-expectancy.R and insert the following R code (if working in R) or SAS code (if working in SAS) to make and save this table:

```
# R Code

# Calculate the LE for each state, separately by race and gender:
le_averages <- le_data %>%
  group_by(state, race, sex) %>%
  summarise(mean_LE = mean(LE))

# print the first 10 rows to the screen. By default, R rounds the numeric
# information in the display to make it more compact
le_averages
# alternatively, type View(le_averages) in the Console to open up a Viewer
# window, or click the table icon beside the le_averages objects in the
# Environment pane (upper right hand panel of RStudio).

#save this table as a CSV file in the data sub-folder
write_csv(le_averages, "./data/le_averages.csv")
```

```
/*SAS Code*/

/*Calculate the LE for each state, separately by race and gender:*/
```

```
proc sort data=le_data; by state race sex; run;
proc means data=le_data; by state race sex; var le; output out=le_averages mean=mean_le ; run;

/*Print the first 10 rows to the screen*/
proc print data=le_averages (obs=10); run;
/*Or you could just open the dataset to browse it.*/

/*Export this file to a .csv file
(if you use the following code, don't forget to replace YourFilePathHere with the appropriate fil
PROC EXPORT DATA= WORK.LE_AVERAGES
            OUTFILE= "YourFilePathHere\data\le_averages.csv"
            DBMS=CSV REPLACE;
     PUTNAMES=YES;
RUN;
```

5.22 Track your changes using Git

- Re-run your previous R code (highlight all the previous code and hit command + Return [Mac] or control + Enter [Windows]). Then run the newly-added code line by line to see what it is doing. Save the updated .R file by pushing the save icon.
- Track the changes using Git. Go to the bash/terminal window. Type "git status". Which files have been modified? Which files are new and untracked?

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch may7-cr
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   code/01_Analyze-life-expectancy.R

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store
    data/le_averages.csv
```

5.23 Add new and modified files

- Use `git add` to add the specific files that have been modified or created. Add them one by one. Use `git status` again to check that all the changed files are being tracked. When you are satisfied, commit these changes locally.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git add code/01_Analyze-life-expectancy.R
```

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git add data/le_averages.csv
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch may7-cr
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   code/01_Analyze-life-expectancy.R
    new file:   data/le_averages.csv

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store
```

5.24 Commit the changes locally

Helpful hint: Terminal/Bash plays well with autocomplete. For example, if you are typing the pathway for the .R file as "code/…" you can push the tab button as you are typing the name and it will autocomplete. This makes selecting the specific files to commit much easier.

- "git commit -m 'your message' ". Replace 'your message' with a short message describing the changes. Remember to keep the quotes around the message!

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git commit -m 'calc LE averages'
[may7-cr a7435b8] calc LE averages
 2 files changed, 177 insertions(+), 1 deletion(-)
 create mode 100644 data/le_averages.csv
```

5.25 Push the changes up to GitHub

- You are now ready to push these changes up to GitHub onto GitHub's version of your local branch. First, remind yourself of your branch's name using "git branch". Then push: `git push origin {YOUR-BRANCH-NAME}`, replacing {YOUR-BRANCH-NAME} with the name of your branch.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git branch
  main
* may7-cr
Biostat-MBP13-20:life-expectancy corinneriddell$ git push origin may7-cr
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 2.84 KiB | 2.84 MiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
```

```
remote:
remote: Create a pull request for 'may7-cr' on GitHub by visiting:
remote:      https://github.com/corinne-riddell/life-expectancy/pull/new/may7-cr
remote:
To https://github.com/corinne-riddell/life-expectancy.git
 * [new branch]      may7-cr -> may7-cr
```

5.26 Create a pull request

- Navigate to GitHub.com and go through the steps described previously to create a pull request to pull these changes into main.

5.27 Changes after receiving an updated dataset

Another week goes by. It is now May 14. You received an email that there was an error in the data file that you used to conduct the analysis. A new data file was securely transferred to you by the data holder. You need to rerun the analysis using the new dataset. The new data file is the one called "LEbsyrx.csv" in the data folder.

5.28 Get ready for the day

- Set yourself up to work with git and GitHub for the day:
- `git branch`: see which branch you are on.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git branch
  main
* may7-cr
```

- `git status`: confirm you committed everything you wanted to commit.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch may7-cr
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store

nothing added to commit but untracked files present (use "git add" to track)
```

- `git checkout main`: switch to the main branch
- `git branch -d may7-cr`: delete the old branch (change "cr" to your initials)

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

- `git pull origin main`: pull GitHub's copy of the main branch to update your local version. Examine the figure made by git about the changes.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git pull origin main
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From https://github.com/corinne-riddell/life-expectancy
 * branch            main       -> FETCH_HEAD
   5f81e34..e15298a  main       -> origin/main
Updating 5f81e34..e15298a
Fast-forward
 code/01_Analyze-life-expectancy.R |  17 ++++-
 data/le_averages.csv              | 161 ++++++++++++++++++++++++++++++++++++++++
 2 files changed, 177 insertions(+), 1 deletion(-)
 create mode 100644 data/le_averages.csv
```

5.29 Checkout a new branch

- `git checkout -b may14-xy`, replacing xy with your initials.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git checkout -b may14-cr
Switched to a new branch 'may14-cr'
```

5.30 Update the dataset

- Update the CSV file "Life-expectancy-by-state-long.csv" with the new dataset. First, decide if you want to archive this older version of the dataset for any reason. If you do, then decide where you would store the archived version and move it there. The archived version could stay on GitHub or be moved off of GitHub – this is up to you and your file organization system.

- Move the LEbsyrx.csv into the data folder and rename it to have the name of the file it is replacing ("Life-expectancy-by-state-long.csv").

5.31 Re-run the analysis

- Re-run all your code that uses this file. First open the .Rproj file to launch RStudio. Then, you can highlight all the code and hit the "Run" button.

5.32 Track the changes using Git

Use "git status", "git add…", and "git commit…" to track these changes locally. Use "git push" to push these changes to GitHub.

Using `git status`, I can see that the data files have all been modified, as has the image file of the plot. There is a new untracked folder called data/archive/ where I moved the archived dataset.

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch may14-cr
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   data/Life-expectancy-by-state-long.csv
    modified:   data/le_averages.csv
    modified:   images/ca-black-women-LE.png

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store
    data/.DS_Store
    data/archive/
```

I use `git add` to specify all the new things I want to track:

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git add data/Life-expectancy-by-state-long.csv
Biostat-MBP13-20:life-expectancy corinneriddell$ git add data/le_averages.csv
Biostat-MBP13-20:life-expectancy corinneriddell$ git add images/ca-black-women-LE.png
Biostat-MBP13-20:life-expectancy corinneriddell$ git add data/archive/
```

I then use `git status` to confirm everything is being tracked:

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch may14-cr
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   data/Life-expectancy-by-state-long.csv
    new file:   data/archive/Life-expectancy-by-state-long_old.csv
    modified:   data/le_averages.csv
    modified:   images/ca-black-women-LE.png

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .DS_Store
    data/.DS_Store
```

5.33 Commit the changes locally

Then I commit my changes:

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git commit -m 'data update and downsti
[may14-cr c4b02db] data update and downstream changes
 4 files changed, 7570 insertions(+), 369 deletions(-)
 create mode 100644 data/archive/Life-expectancy-by-state-long_old.csv
 rewrite images/ca-black-women-LE.png (98%)
```

5.34 Push the changes to GitHub

Finally I push these changes to GitHub:

```
git push origin may14-cr
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 68.08 KiB | 9.73 MiB/s, done.
Total 8 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/corinne-riddell/life-expectancy.git
   e15298a..c4b02db  may14-cr -> may14-cr
Biostat-MBP13-20:life-expectancy corinneriddell$
```

5.35 Submit a pull request

- Go through the process to start a pull request. On the pull request page, scroll down to see the "diffs" in the data and image. Pay close attention to the files that were changed:
- use the "2-up" "swiper" and "onion skin" tools to see the changes to the saved figure. How did the data changed? Which tool do you prefer?
- can you see which rows of data were affected by the change?
- can you tell from the data or images how the change affected the analytic findings?
- anything else you noticed?

5.36 To be worked in

(vi) Open your .gitignore file in a text editor that you are comfortable with (e.g. nano or notepad). Add the following two lines to it: **LBW COMMENT: CORINNE, DO WE WANT THEM TO ADD ANYTHING TO THEIR GIT IGNORE?**

Chapter 6

Do's and Don't's of using GitHub

6.1 Different approaches to using Github for data analysis

6.2 Different approaches to using Github for data analysis

Approach 1: Some people use Github to track changes to their statistical code only.

- Here, they are only tracking their code files in R (.R or .Rmd), or SAS (.SAS), or the corresponding files for whichever language they use.
- They add all other types of files to the git ignore file so that git won't track them or accidentally push them to GitHub. This would include: data files, word documents, pdfs, image files, etc – anything else you are putting in the github folder that you don't want to track.
- This approach most closely corresponds to how GitHub is used by software developers/ other code writers, but is also an option chosen by researchers and research teams depending on their objectives.

Approach 2: There is another approach to using Git and GitHub in which you might choose to track a larger set of files. These other files might include:

- Data
- Images
- Documents

6.3 Approach 2: Tracking data files

- **Don't track restricted data**
- **Dont' track large datasets.** Git will warn you if your file exceeds 50 MB and block you from tracking files 100 MB or larger.
- To ensure these files aren't tracked, you can store them elsewhere (outside of the tracked Git folder), or store them in the tracked folder while also listing them or their file type to your .gitignore file.
- It doesn't often make sense to track a large "raw" data file – it is too big and not useful to track any changes to this file.
- It may be helpful to track "intermediate data products", if these files are not restricted.

- Intermediate data products might include aggregated datasets that are either reported directly or used in analysis.
- For example, you may have access to a restricted dataset, but the data may become non-restricted if you aggregate at the level of the census tract. The benefit of tracking this smaller dataset is that if the raw data is updated, you can easily see how those updates affect these intermediate data products if you track them.
- In this case you need to ensure you are not reporting any private/restricted data (eg no cell counts below 10 is a restricted often imposed on aggregated tables, or not reporting any identifying features such as protected health information or anything else that would allow anyone with access to identify individuals.
- What type of data files should you track?
- Anything that is plain text like csv files and txt files (e.g., files you can open in a text editor).
- These are best for tracking because they render nicely on GitHub, so you can easily view the differences to these files when you submit a pull request.
- You could also track things like Excel files but you can't easily view them on GitHub, so some of the benefits of using GitHub do not apply to these files.

6.4 Approach 2: Tracking image files

- Image files (e.g., png, jpeg), such as plots/other figures you create for a report, may also be tracked.

- The benefit of also tracking figures are the use of the image comparison tools in GitHub to see an image pre/post a change in the analysis.
- This can be super helpful when you have modified something in the analysis after having already written up some results.
- If you track the image, you can easily see how it changed (it being the point estimates and confidence intervals, of the slope of the regression line, or the shading of a colored map).
- This has the direct pay-off of making it much easier to revise the written results as you can more easily see the changes using these tools.
- One thing to be careful about is not uploading very large images like the ones that are generated by some GIS analyses (large maps).
- If you aren't sure if you should track your image file, take a look at the file size. (I had a look at all of my repositories for epidemiologic analyses – most images were < 1000 kb. Some were between 1 MB and 4 MB – these were some maps and some images saved at higher resolutions.)

6.5 Approach 2: Tracking documents

- You can also track reports and manuscripts using GitHub.
- If these reports are written in a plaintext language (e.g., R markdown, LaTeX) then they will render nicely on GitHub and, and permit you to see the "diffs" made to the document during a pull request.
- Tracking pdfs is permissible but you can't easily see the "diffs" when they are updated. However, pdfs are easily viewable on GitHub.
- Tracking docx files is also permissible, but you can't easily see the "diffs" and they are not easily viewable on GitHub (since they require MS Word to render them).

6.6 Things you definitely do not want to track

- Anything that is private or restricted or that you would never want to inadvertently share.
- This includes passwords, or API keys that you might use to extract data.
- For example, I have an API key to access Census data. I do not include this API key in my GitHub repository. There are methods for writing the code to still access the key on my personal computer without writing it out explicitly in the code.

Chapter 7

Group Workflow

7.1 Outline for this section

Now, we're going to learn about working collaboratively using Git and GitHub.

- Setting up a shared repository
- First Commit
- Collaborative Routes
  - Independent Work
  - Code Review
  - Direct Edits
- Big picture: Tracking Changes outside of Git
- Practice

7.2 Set up

- **Find a partner:** One of you will be the *lead researcher*, and the other will be your *colleague*.
- **Decide roles:** The lead researcher will create the repository, and the colleague will contribute to it.

7.3 Create your repository: Person 1

One of you will follow these instructions and one will follow instructions on the next slide.

7.3.1 Lead researcher (Person 1): Create a new repository on GitHub.

- Go to GitHub, click on the "+" icon in the top-right corner, and select `New repository`.
- Follow the same instructions as we did in the solo exercise to create a new repository (add a README, choose a license, etc.).
- Add your colleague as a collaborator by going to `Settings` > `Collaborators` > `Add people`. Ask your colleague for their GitHub username and add them to this repository.

7.4 Join the repository: Person 2

If you did not follow the person 1 instructions, follow these!

7.4.1 Person 2 (Colleague): Accept the invitation to collaborate on the repository.

- Check your email for an invitation to collaborate on the repository.
- Accept the invitation by clicking on the link in the email.

7.5 Clone the repository

7.5.1 Both: Clone the repository

- Go to the main page of your repository clicking on the green `Code` button and copying the URL. Then, navigate to the location where you will clone your repository and run the following command in your terminal:

```
git clone {repository URL}
```

7.6 Check the status of your repository

Now that you both have the repository cloned, we will set ourselves up following the same steps as we did this morning. Let's redo what we did this morning:

Type `git status`. The results shows you that no changes have been made yet:

```
Biostat-MBP13-20:life-expectancy corinneriddell$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

7.7 Determine the branch you are on

Type `git branch`. This shows you that you are currently on the main branch.

```
Biostat-MBP13-20:life-expectancy corinneriddell$git branch
* main
```

7.8 Create a new branch

Now that we know that our repository is cloned, up to date, and on main, we can create a new branch to work on.

- **Both:** Create a new branch with the following command:

```
git checkout -b {new-branch-name}
git branch
```

Typing `git branch` will show you that you are now on the new branch, though only the first line was *necessary* to create this branch.

7.9 Collaborative Routes

Now we will go through three different ways of working collaboratively on a project:

1. **Independent Work:** You each work on separate files and merge your own changes.
2. **Code Review:** Submit pull requests for code reviews without direct edits.
3. **Direct Edits:** Submit pull requests and make direct changes to each other's code.

7.10 Collaborative Routes

While these different routes may seem ambiguous right now, you can think of pull requests as akin to "Track Changes" or "Comments" in Microsoft Word or Google Docs.

Collaborative route #1 (Independent work) is like creating two separate documents and having them in a shared folder.

Collaborative route #2 (Code review) is like having a shared document where you can leave comments for each other - think about when you leave comments on a word document for your colleague.

Collaborative route #3 (Direct edits) is like having a shared document where you can make changes directly to your colleague's work - think track changes.

7.11 Collaborative Route #1: Independent work

You each work on separate files and merge your own changes. The repository will have two files, one that each of you made!

7.12 Collaborative Route #1: Add a file

- **Lead researcher:** Save the code file that you brought to this workshop in your repository!
- **Colleague:** Save the code file that you brought to this workshop in your repository!

Your files should have different names but if they do not, please make sure to rename one of them!

7.13 Collaborative Route #1: Commit our new file

- **Both:** Add, commit, and push your file to GitHub as shown.

```
git add {file-name}.R
git commit -m "Initial commit with {file-name}"
git push origin {new-branch-name}
```

7.14 Collaborative Route #1: Create a pull request

We'll do the same thing we did earlier today:

- Navigate to GitHub.com to your repository's URL. There should be a pale yellow banner informing you about the changes you just pushed. Click the button "Compare & pull request". Notice that the title is your commit message from the previous step. Scroll down. Look at the files that have been added.
- The code is all shown in green, indicating that every line of code is new.

7.15 Collaborative Route #1: Merge the changes from your branch into main

Both: Again, this is the same as what you did earlier today:

- Click on the green "Create pull request" button. Github will check that it is able to merge your branch with main without problems. Note the message "This branch has no conflicts with the base branch". This means you are good to go!
- Click on the green "Merge pull request" button.

- Click on the green "Confirm merge" button.
- Click the "Delete branch" button.

7.16 Collaborative Route #1: Let's look at our main branch now

Both:
- Go into the main `Code` page and take a look at the code on `main` (which should be the only existing branch in your repository).
- You should see the files that *both* of you created in the previous steps.

7.17 Collaborative Route #1: Recap

You have now successfully completed the first collaborative route. You each created a file, committed it, and merged it into the main branch. You worked on different files and had no merge conflicts or interaction, but now have access to each other's work!

7.18 Collaborative Route #2: Code Review

In this code review style collaborative approach, the lead researcher will make a new file, make changes to that file and push these to GitHub. The colleague will then review these changes by submitting comments on the pull request.

7.19 Collaborative Route #2: Create a new branch

Lead researcher: Create a new branch with the following command:

```
git checkout -b {new-branch-name}
git branch
```

Typing `git branch` will show you that you are now on the new branch, though only the first line was *necessary* to create this branch.

7.20 Collaborative Route #2: Lead Researcher makes changes to a file

- **Lead Researcher:** Make a change to the file you used above. You can add in a comment or a new line of code - something small. Save your file.

7.21 Collaborative Route #2: Push Your Changes

- **Lead Researcher:** Add, commit, and push your new file or changes to your branch on GitHub.

```
git add {file-name}
git commit -m "{your description of what you changed}"
git push origin {your-branch-name}
```

7.22 Collaborative Route #2: Create a Pull Request

- **Lead Researcher:** Go to your GitHub repository page. You should see a notification about your recent push. Click on "Compare & pull request" next to your branch.
- Fill in the details of the pull request, explaining what you've added or changed and why.
- **This time, add your colleague as a reviewer. We will not immediately merge this in this time!**
- Submit the pull request.

7.23 Collaborative Route #2: Colleague Reviews the Pull Request

- **Colleague:** Once the pull request is submitted, you should receive an email telling you that your lead researcher requested your review on PR #{X}.
- Click `view it on GitHub` and it will bring you to GitHub and there will be a yellow box at the top.
- Click on `Add your review` in the yellow box. Then click `Review changes` in green in the right corner.
- Leave comments on the pull request by clicking the plus signs next to chunks of code. You can leave multiple comments Your feedback can include suggestions, questions, or general feedback about the code. Focus on clarity, efficiency, and any potential errors you might notice.

- When you're finished commenting, click `Finish your review` in green in the top right.

7.24 Collaborative Route #2: Colleague Reviews the Pull Request

- Submit your comments. You can either:
  1. Comment: This submits feedback without explicitly approving the PR.
  2. Approve: This submits feedback and approves the PR for merging into main.
  3. Request Changes: Submit feedback that the other person needs to address before they can merge into main.

7.25 Collaborative Route #2: Pull Request Review Options

1. Comment: This submits feedback without explicitly approving the PR.

- This will send the lead researcher an email with your comments on their code. The lead researcher can click `View it on GitHub` to see your comments. They can reply and/or merge the PR into main at this point.
- This can be used either as an FYI or to start a conversation about the code - e.g. maybe a certain function is new to you so you want to ask about why they chose to use that.
- You may want to use this if you have a workflow where you always want the person who wrote the code to do the merging.

7.26 Collaborative Route #2: Pull Request Review Options

2. Approve: This submits feedback and approves the PR for merging into main.

- This will send the lead researcher an email with your comments that will also indicate that the PR is approved.
- This option can be used as an FYI - e.g. telling them about an alternate function that they *could* use but do not have to.
- Either person can merge the PR into main. You can establish norms around how these work with your colleagues - there is no right answer!

7.27 Collaborative Route #2: Pull Request Review Options

3. Request Changes: Submit feedback that the other person needs to address before they can merge into main.

- This will send the lead researcher an email with your comments that will also indicate that the PR needs changes.
- This option can be used if you see something that needs to change before the PR can be merged - e.g. a function that is not working as intended.
- The lead researcher will need to address your comments before they can merge the PR into main.

NOTE: Technically, the PR could be merged in without the changes, but this is not good practice.

7.28 Collaborative Route #2: Your turn [CR/LBW: THIS SLIDE NEEDS TO BE REFINED]

- **Colleague:** Review the changes made by your lead researcher in the pull request. Leave comments on the code as needed. Choose whichever option you think is best.
- **Lead Researcher:** Review the comments made by your colleague. Make any necessary changes to your code based on the feedback.
- Add, commit, and push these changes to the same branch. This will automatically update the pull request.
- Colleague will receive an email saying that the lead researcher pushed one commit. At this point, the colleague can click `View it on GitHub` and verify that the changes are sufficient.
- Now you can restart the process and either comment, comment and approve, or comment and request changes again!

7.29 Collaborative Route #2: Finalizing the Pull Request [CR/LBW: THIS SLIDE NEEDS TO BE REFINED]

- **Colleague:** Review the changes made by the lead researcher in response to your comments. If you're satisfied with the updates, approve the pull request.
- **Lead Researcher:** Once the pull request is approved, merge your changes into the main branch.

7.30 Collaborative Route #2: Recap

We have now practiced using Git to do a code review! You used Git to review/check/comment on our colleague's work without emailing or copying/pasting code back and forth. The lead researcher made changes to their code based on your feedback, and you both worked together to finalize the pull request.

7.31 Collaborative Route #3: Direct Edits

CR/LBW: DO WE WANT TO COVER THIS?

In this scenario, after your colleague pushes their code, they will submit a pull request to you, and you will directly edit their code if needed. This may be useful if one person is stuck on a certain piece of the code, or if you are working very closely together on one script.

7.32 Collaborative Route #3: Create a new branch

- **Colleague:** Create a new branch with the following command:

```
git checkout -b {new-branch-name}
git branch
```

Typing `git branch` will show you that you are now on the new branch, though only the first line was *necessary* to create this branch.

7.33 Collaborative Route #3: Colleague Creates a File

- **Colleague:** Make a change to the file you used above. You can add in a comment or a new line of code - something small. Save your file.

7.34 Collaborative Route #3: Push Your Changes

- **Colleague:** Add, commit, and push your new file or changes to your branch on GitHub.

```
git add {file-name}
git commit -m "{your description of what you changed}"
git push origin {your-branch-name}
```

7.35 Collaborative Route #3: Create a Pull Request

- **Colleague:** Go to your GitHub repository page. You should see a notification about your recent push. Click on "Compare & pull request" next to your branch.
- Fill in the details of the pull request, explaining what you've added or changed and why.
- **This time, add your lead researcher as a reviewer. We will not immediately merge this in this time!**
- Submit the pull request.

7.36 Collaborative Route #3: Lead researcher makes direct edits

- **Lead Researcher:** Review the code in the pull request. Look at the section with a comment asking for your help, make direct edits to the code in the pull request.

7.37 Collaborative Route #3: Commit the Changes

CR/LBW: THIS SLIDE SHOULD BE REFINED ONCE WE PRACTICE THIS TOGETHER

- After making direct edits, push your changes back to the branch.
- **Both:** Review the final changes together, discuss any further modifications, and then merge the pull requests into the main branch.

7.38 Collaborative Route #3: Recap

This route had you directly edited your colleague's code. This can be very useful when one person knows how to do something that another person is stuck on, or when the code is a collaboration. This may not be very common in your work, but is a valuable tool if you are helping someone with code!

7.39 Merge conflicts

7.40 Practicing a merge conflict

This occurs when XXX.

7.41 Practicing a merge conflict

Both: - Both partipiants should be on `main` now. If you are not, please do `git checkout main` and `git pull origin main` to make sure you are up to date. - Both participants make and checkout a new branch: `git checkout -b {new-branch-name}` - Both participants make a change to the same line of code in the same file. You can choose whichever file you want to use. Edit the last line of the file to make this easier to use for practice. - **Lead Researcher**: Add, commit, and push your changes to your branch. Create a pull request and merge your changes into main. - **Colleague**: Add, commit, and push your changes to your branch. Create a pull request and you will not be able to merge into main.

7.42 Practicing a merge conflict

Colleague: You will see the following.

LBW INSERT FIRST SCREENSHOT HERE

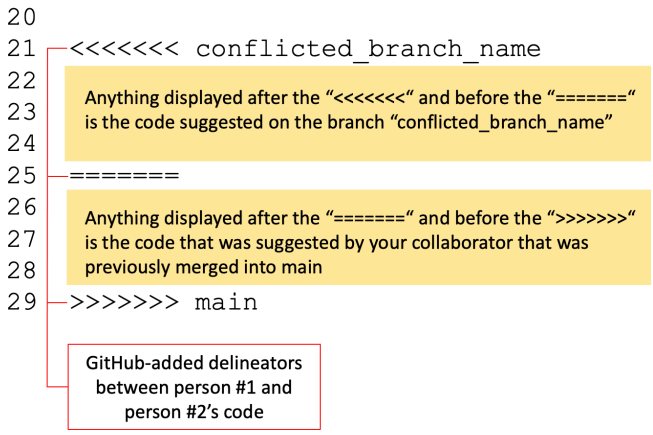
7.43 Practicing a merge conflict

Colleague: Click on `View Pull Request` to begin resolving the merge conflict. When you click on `View Pull Request`, you will see the following.

LBW INSERT SECOND SCREENSHOT HERE

7.44 Practicing a merge conflict

Colleague: Click on `Resolve conflicts` to begin resolving the merge conflict. When you click on `Resolve conflicts`, you will see the following:



7.45 Practicing a merge conflict

Both: Together, look at the code in the file. Decide which edits you will keep. For this exercise, it is abritrary which edits you keep. However, keep in mind that in real life you will make this decision in an informed way.

When you choose what to keep, take out all of the `>>>` and `===` and `<<<` lines, as well as the lines from the person whose code you are not keeping. When you are finished, click `Mark as resolved` in the top right corner.

Note: You are still doing all of this on GitHub, not in your terminal.

7.46 Practicing a merge conflict

Now that you have clicked `Mark as resolved`, you will see a green checkmark next to the file name, and it will say `Resolved` in the top right corner. There will also be a green button in the top right corner that says `Commit merge`. Click on this button.

This brings you back to the page you are familiar with, where you can merge into main. Click `Merge pull request` and then `Confirm merge`. You can delete the branch as you usually do.

You have just resolved your first merge conflict!

CR/LBW: add conclusion slides make sure to mention that if all else fails, they should blow up their repo.