Collaborative Coding

Version Control with Rstudio/GitHub

Compscibio2023

Welcome!

Module expectations & conduct

- Welcoming and supportive space for everyone regardless of background or identity: please use inclusive language
- Show respect and appreciation for participants and facilitators
- Accept constructive criticism with grace
- Everyone has different levels of experience and everyone, including facilitators, has space to learn
- Encourage questions
- “No drop” module: speak up if you fall behind so we can help

Course Outline

I. Introductions

II. Concepts of version control and collaborative coding

III. Sign up for a GitHub account

IV. Walk through and help installing/upgrading software

V. Rstudio/GitHub integration

VI. Workflows for collaborative coding with version control

VII. Practice working collaboratively on a (non-coding) project

VIII. Bonus: project management boards, issues, milestones

Systems Supported:

Systems that are supported:

- Windows (honestly don’t know - a modern one?)
- Mac OS 11 or higher

Systems that are not supported:

- Older Windows systems (How old? Dunno)
- Mac OS <11
- Linux :(

Rob Baker

Data Scientist/National Park Service

- Formal education entirely biology (plant evolution, development, genetics, etc)
- First introduced to R in 1999. But really… came kicking and screaming to R during my Ph.D.
- GitHub user since 2013 . . . but similar trajectory: took a long time to really use it
- Ex-Academic and currently work for the National Park Service in Colorado
- Mainly develop in R, build tools for data management, metadata construction, etc (“NPSdataverse”)
- Even as a “power user” of R and GitHub I only scratch the surface the capability/potential

Co-facilitators

Emma Girolami

- Recent Miami grad - Spring of 2023!
- Biology major
- Currently works at Hueston Woods State Park

Co-Facilitators

Thad Deiss

- Recently completed post-doctoral work at Uppsala University

- Interests in the evolution of vertebrate adaptive immune systems (especially cartilaginous fish and bovids)

Introduce yourself

~30 seconds each

- Name
- Affiliation/what you do
- Why you chose this module
- What is your most frequently used emoji? (🙄)

Course Outline

I. Introductions

II. **Concepts of version control and collaborative coding**

III. Sign up for a GitHub account

IV. Walk through and help installing/upgrading software

V. Rstudio/GitHub integration

VI. Workflows for collaborative coding with version control

VII. Practice working collaboratively on a (non-coding) project

VIII. Bonus: project management boards, issues, milestones

What is Version Control?

Version Control is the practice of tracking and managing changes (to software, code, documentation, etc)

You're probably already using version control!

- If you've ever used track changes in a word document or google doc, that's a type of version control
- If you've ever made changes and then saved your file so that you have files such as "final_draft1.docx", "final_draft2.docx" that is also version control

So why use GitHub for version control?

Cons:

- It isn’t easy to set up
- It’s a very different way of thinking about how you work
- Will require re-engineering how you save your code
- Substantial learning curve: can be painful!

So why use GitHub for version control?

Pros:

- Very easy to share code and data with collaborators
- Essentially provides a cloud backup and support for working from different computers
- Easy to work collaboratively with colleagues on coding (or other!) projects
- Code and data can be publicly available for publication purposes
- It’s easy to get wrong - but it’s hard to do something unrecoverable
- Project management tools can be built in

What is a Repository?
“Repo”

Repository - a collection of files that are managed in a structured way

- These can be static - think of something like data archives or a collection of previously published journal articles
  - JSTOR
  - Dryad
- Or they can be dynamic - these files are subject to change, updates, edits, and versions. We will primarily be dealing with dynamic repositories. These are kind of like DropBox or GoogleDrive or OneDrive but much better!
  - GitHub
  - BitBucket
  - GitLab

Why GitHub?

- Free!
- Public or private repos
- Advanced functionality (Issues, pull requests, automation, handles merging)
- Great for R development (particularly if you want to develop packages!)
- Fairly ubiquitous across academic, government, and industry institutions

(Yes you can and should put your GitHub account on your CV!)

Course Outline

I. Introductions

II. Concepts of version control and collaborative coding

III. Sign up for a GitHub account

IV. Walk through and help installing/upgrading software

V. Rstudio/GitHub integration

VI. Workflows for collaborative coding with version control

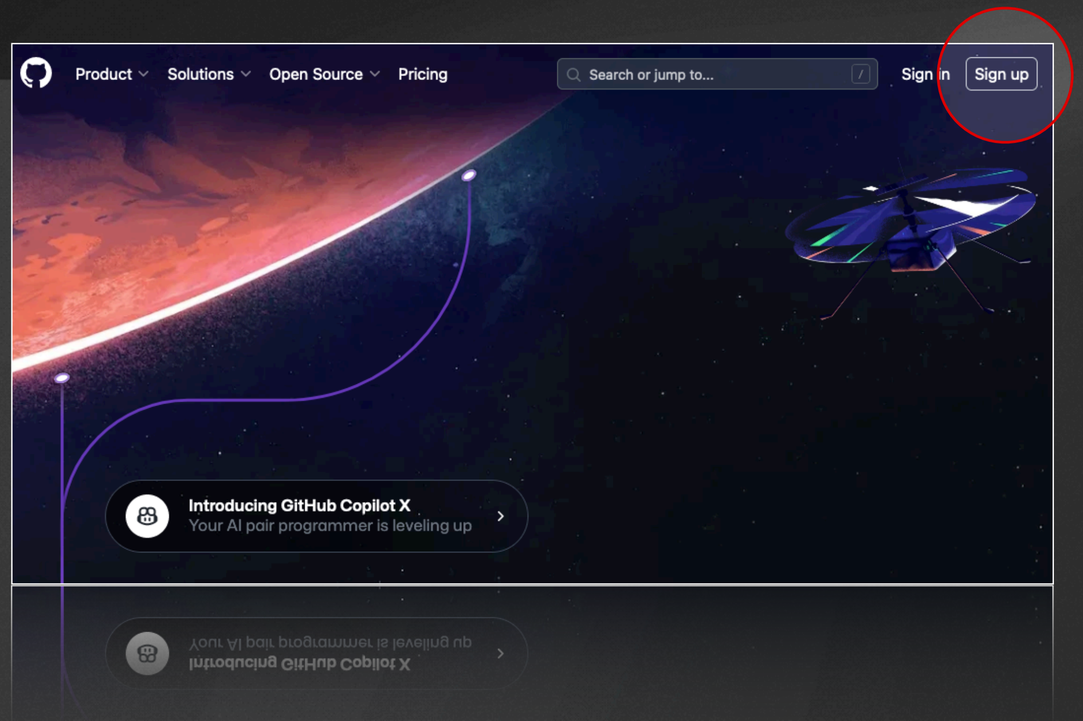
VII. Practice working collaboratively on a (non-coding) project

VIII. Bonus: project management boards, issues, milestones

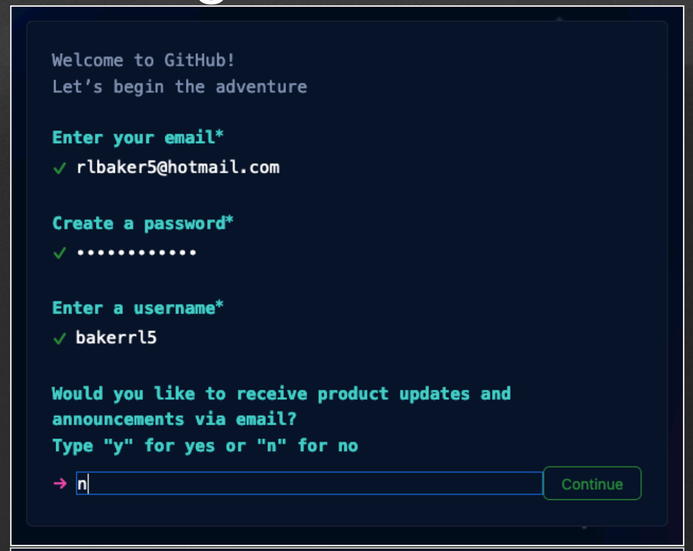
Set up:

Get a GitHub account (Step 1 of 4)

- Go to https://github.com
- Click “sign up” in the top right corner



- Supply Email & password
- Username can be changed later

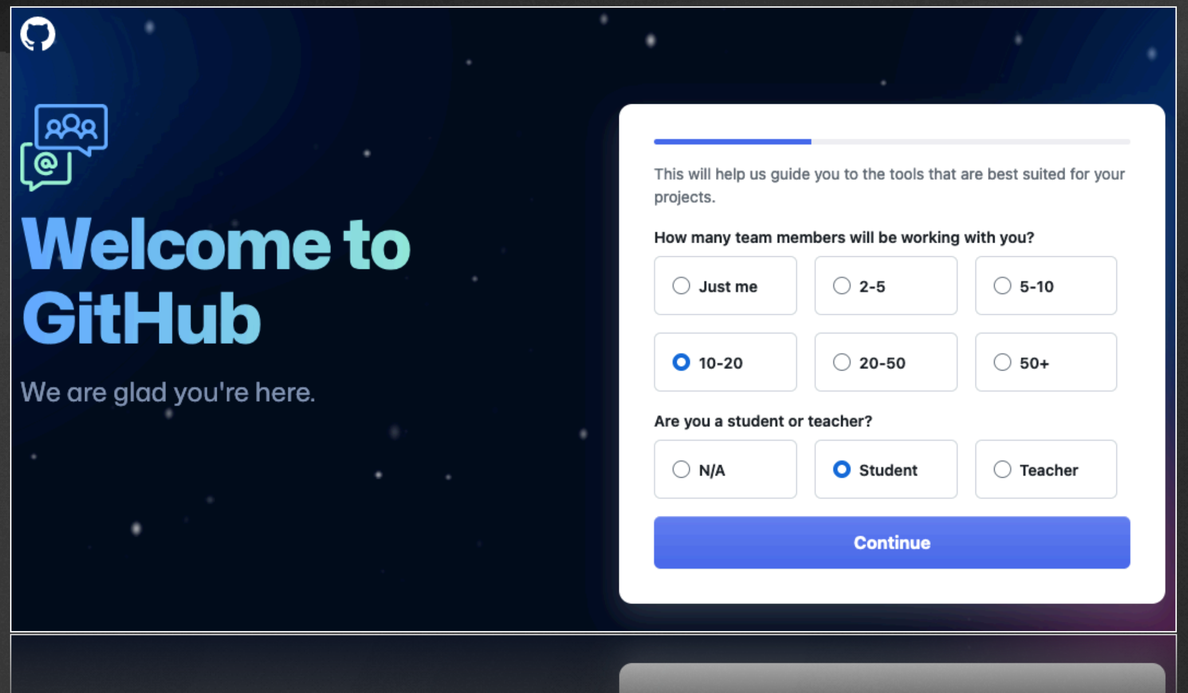


- Click Create Account
- Enter code sent to your email

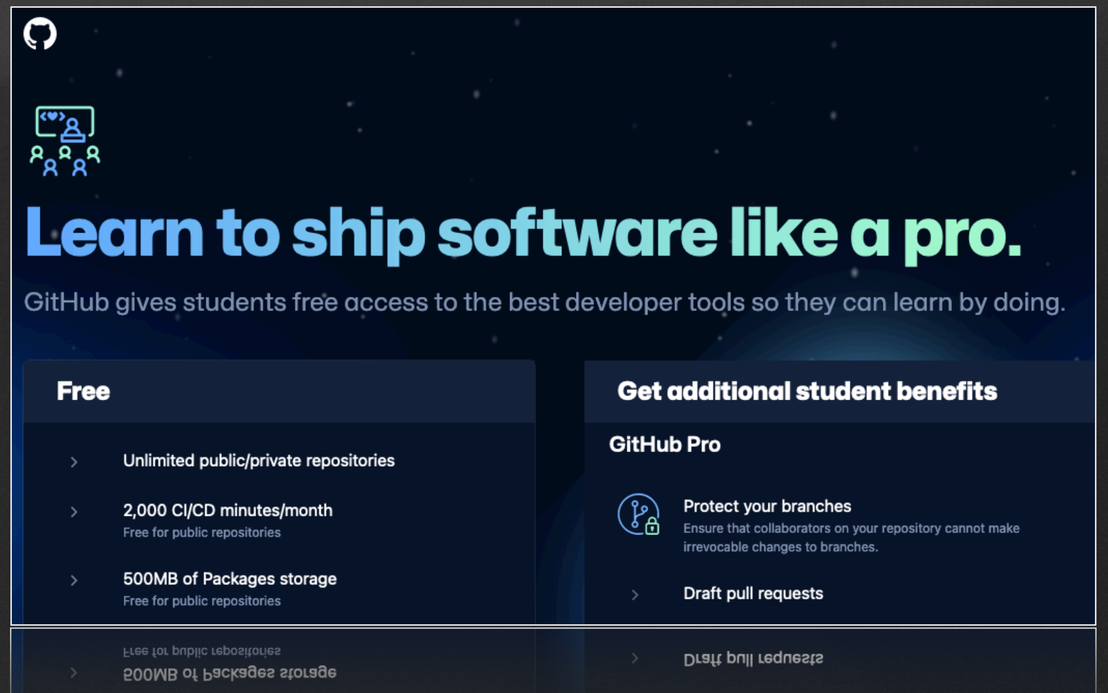
Set up

Get a GitHub account (Step 1 of 4)

- Fill out the Welcome questions. Not sure this really matters that much.



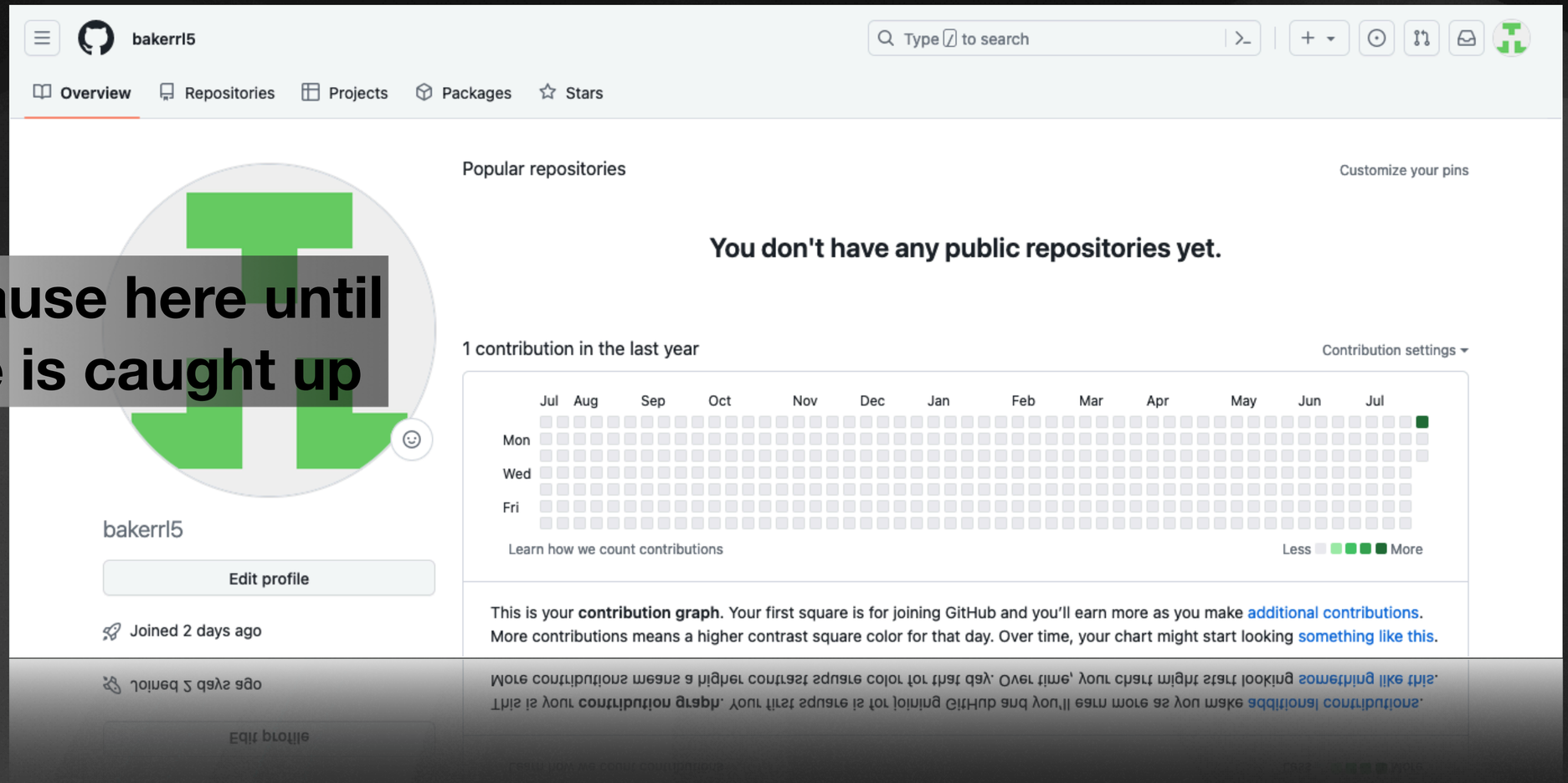
- Scroll down and select the **Free** version!
(unless you don't like your money)



Set up

Get a GitHub account (Step 1 of 4)

We will pause here until everyone is caught up



Course Outline

I. Introductions

II. Concepts of version control and collaborative coding

III. Sign up for a GitHub account

IV. Walk through and help installing/upgrading software

V. Rstudio/GitHub integration

VI. Workflows for collaborative coding with version control

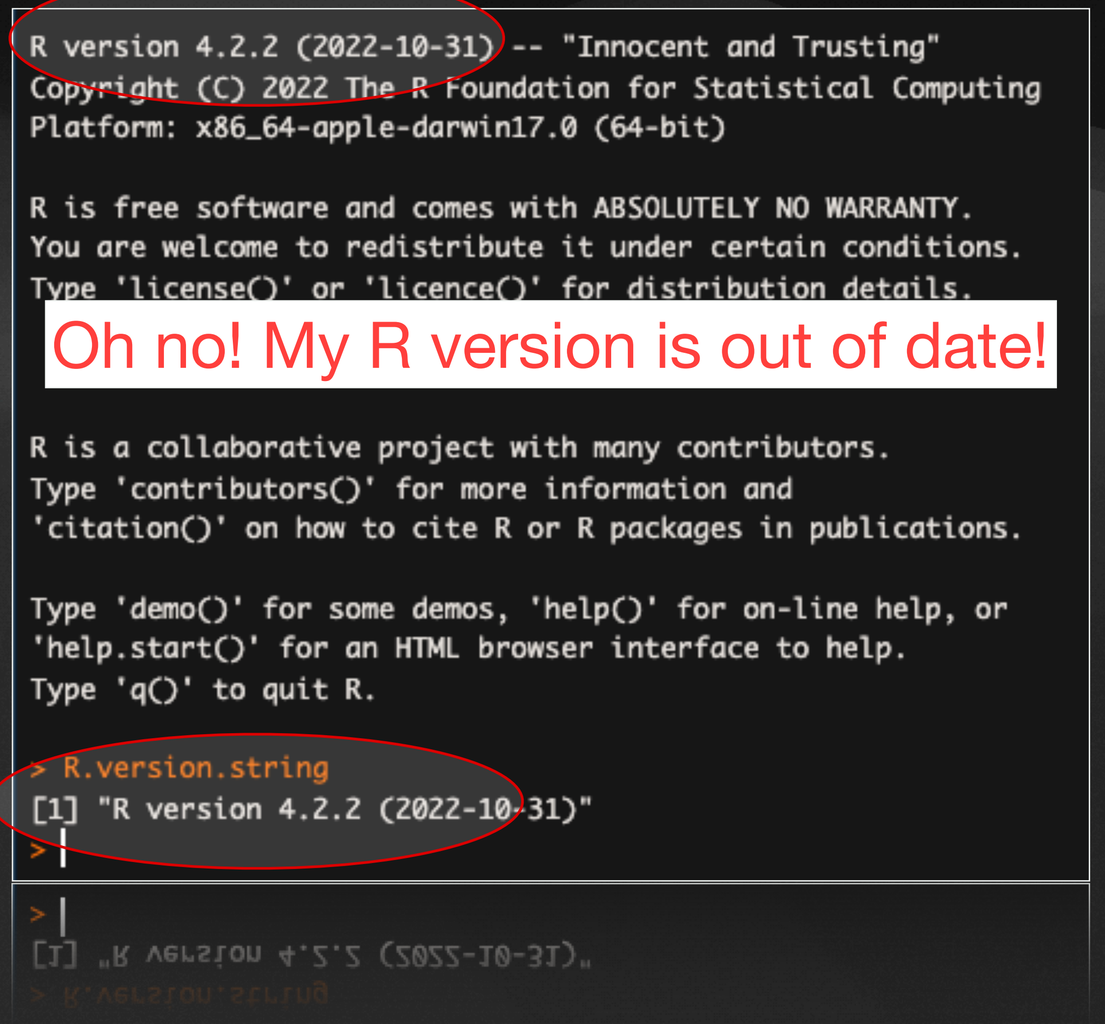
VII. Practice working collaboratively on a (non-coding) project

VIII. Bonus: project management boards, issues, milestones

Set up

Get (or upgrade) R (step 2 of 4)

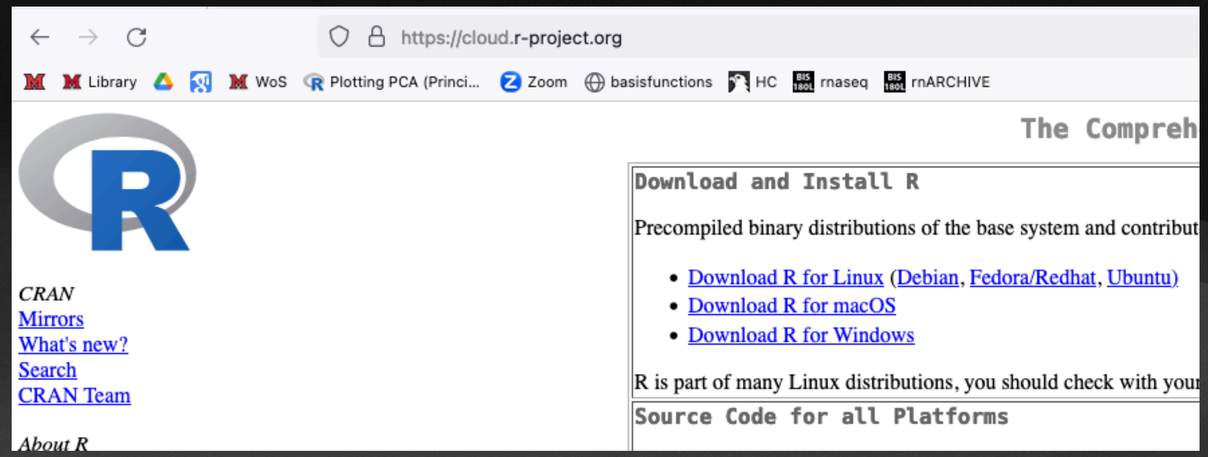
- If you already have R:
- Open R and see what version it is
- Or type:
  >R.version.string
  to retrieve the version
- If you have **v4.3.1** (“Beagle Scouts”) you are ready to go
- If not, follow along to upgrade

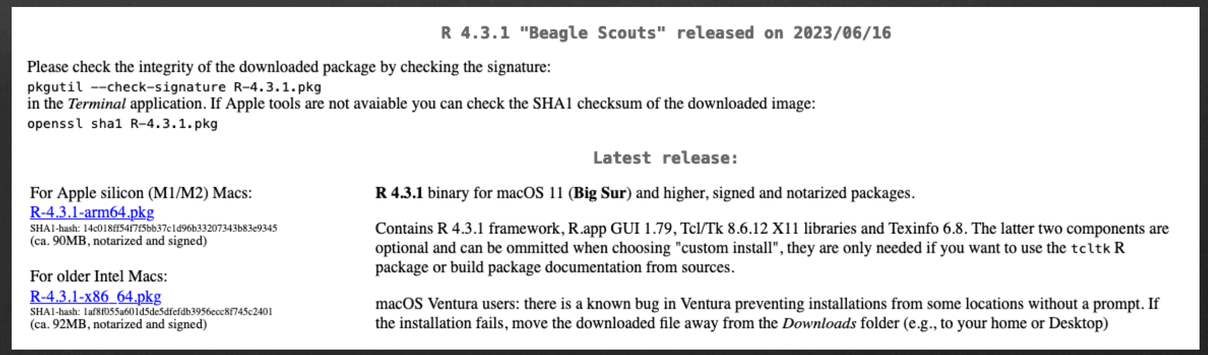


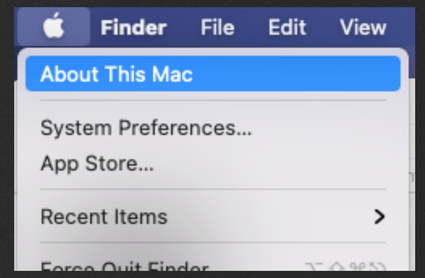
Set up

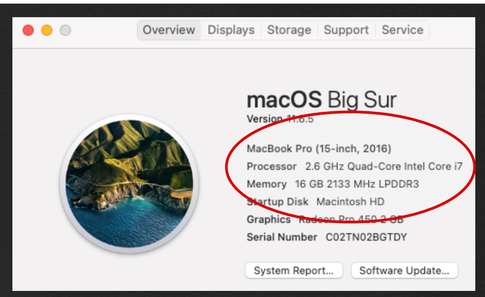
Get (or upgrade) R (step 2 of 4)

- To install or upgrade go to: https://cloud.r-project.org
- Click on the correct Download button for your operating system
- **Mac only:** you will need to choose the installation for your processor
- **Mac only:** To determine your processor, click on the Apple icon in the top left of your screen and choose “About this Mac”
- Download R and install using all the defaults







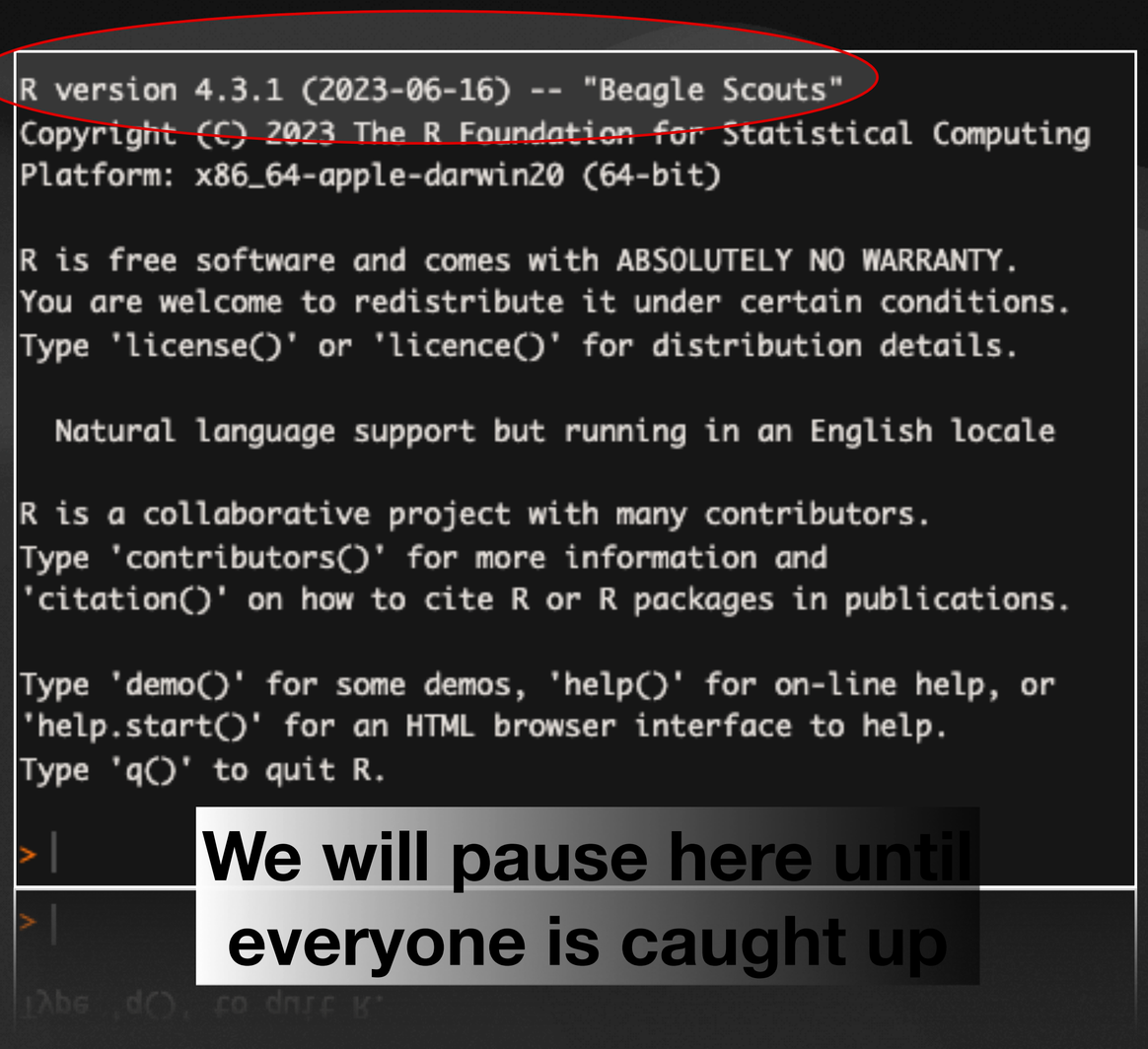


Set up

Get (or upgrade) R (step 2 of 4)

- Check to make sure the install went well
- Open R
- Make sure it says:

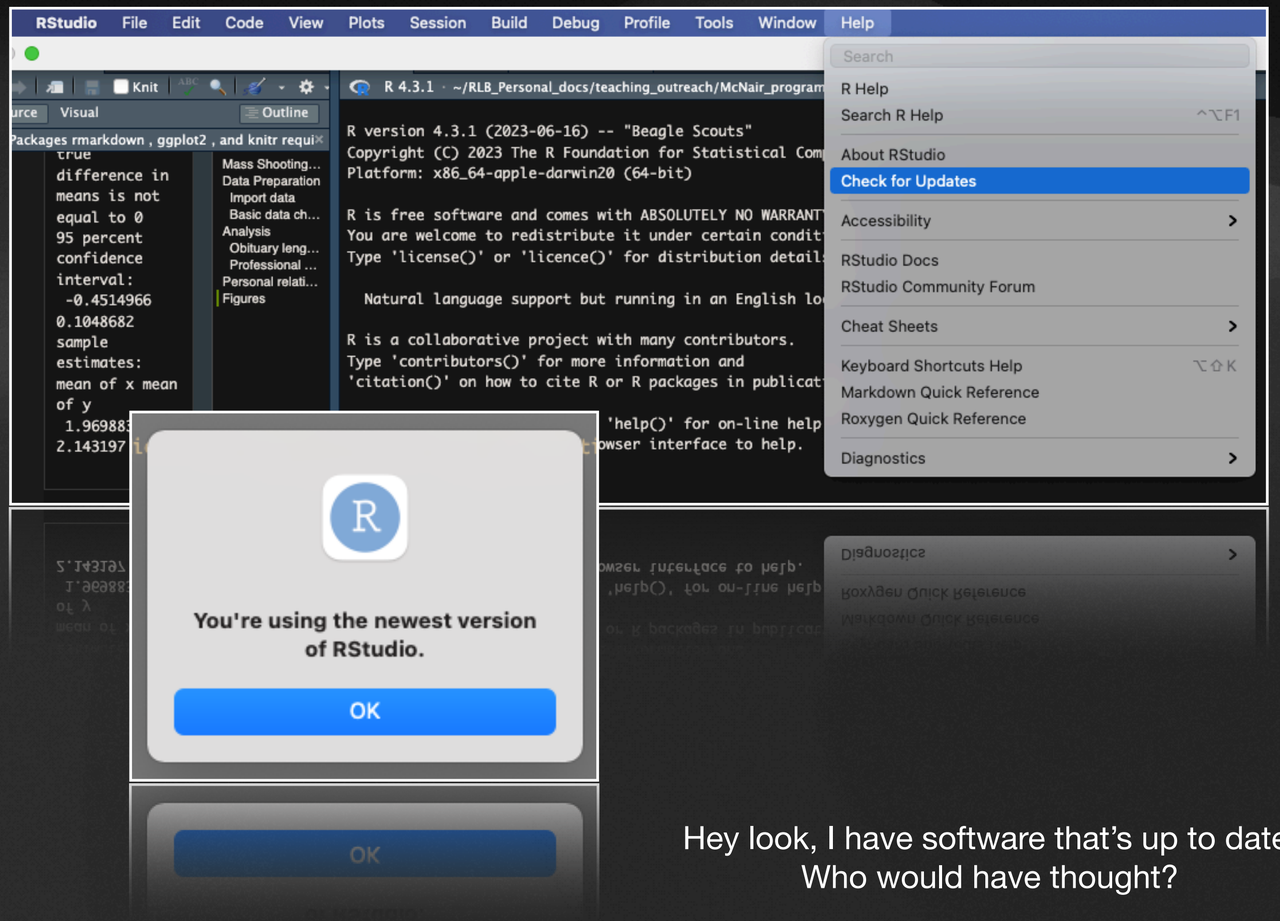
“R version 4.3.1 (2023-06-16) — “Beagle Scouts”



Set up

Get (or upgrade) Rstudio (step 3 of 4)

- If you already have R studio, make sure you have the latest version.
- Open Rstudio, go to the “Help” drop down menu at the top of the window
- Select “Check for Updates”
- If you have the latest version, you’re good to go!
- If you see “Updated Available” follow along to update Rstudio.

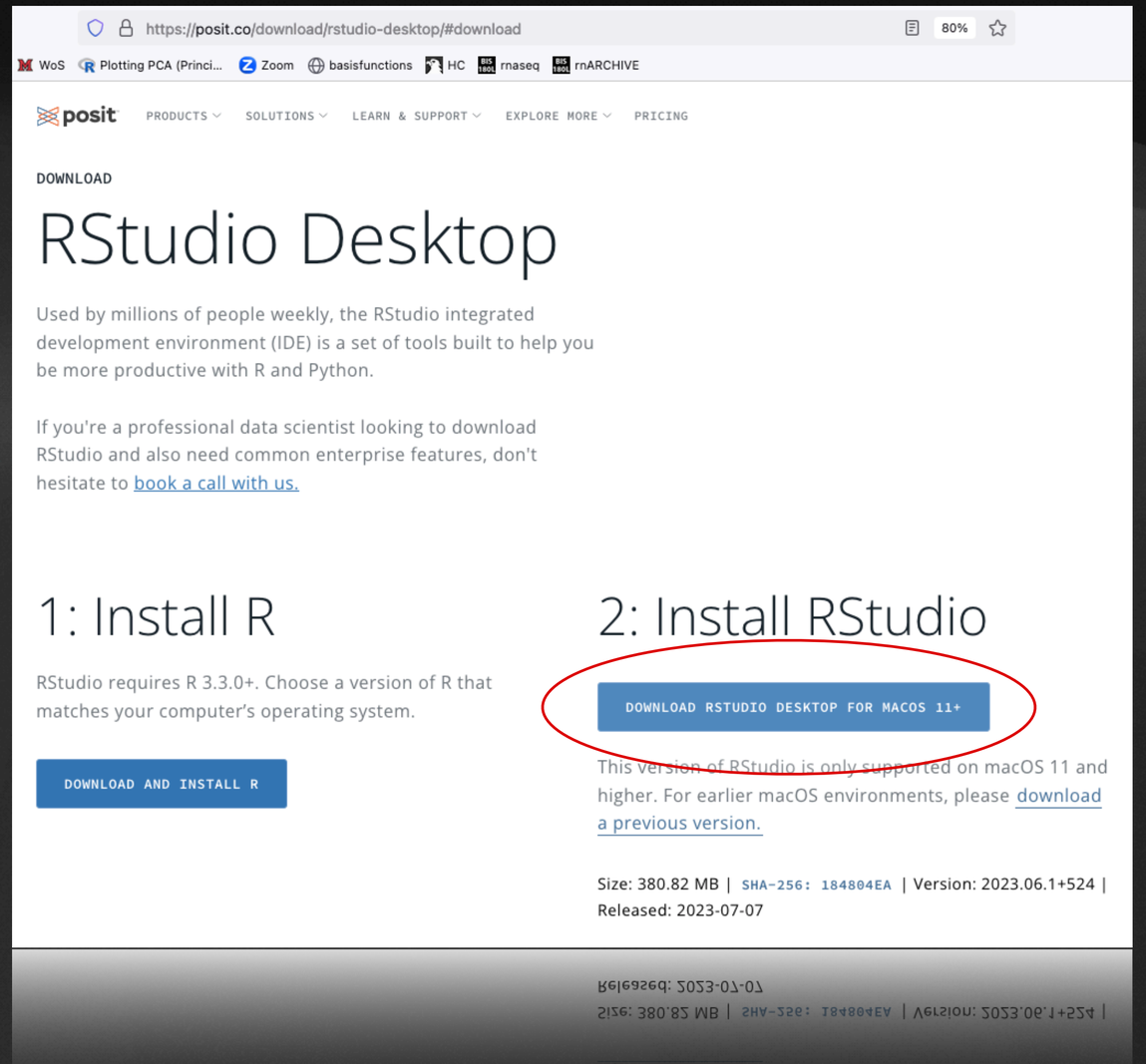


Hey look, I have software that’s up to date! Who would have thought?

Set up

Get (or upgrade) Rstudio (step 3 of 4)

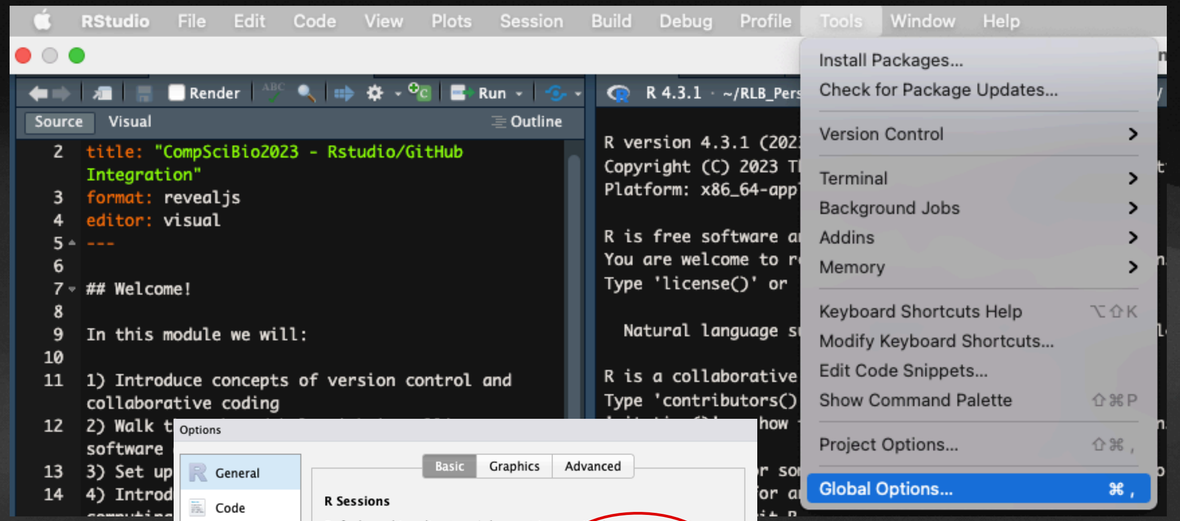
- Go to: https://posit.co/download/rstudio-desktop/#download
- Click “Download R Studio” (it will automatically detect your OS)
- Install Rstudio using all the defaults.



Set up

Get (or upgrade) Rstudio (step 3 of 4)

- Open Rstudio
- Make sure it is accessing the correct version of R (4.3.1)
- If it is not, select “Global Options”
- Click “Browse” and navigate to your newly installed version of R (4.3.1)
- Click “Apply” (you may need to restart Rstudio too)

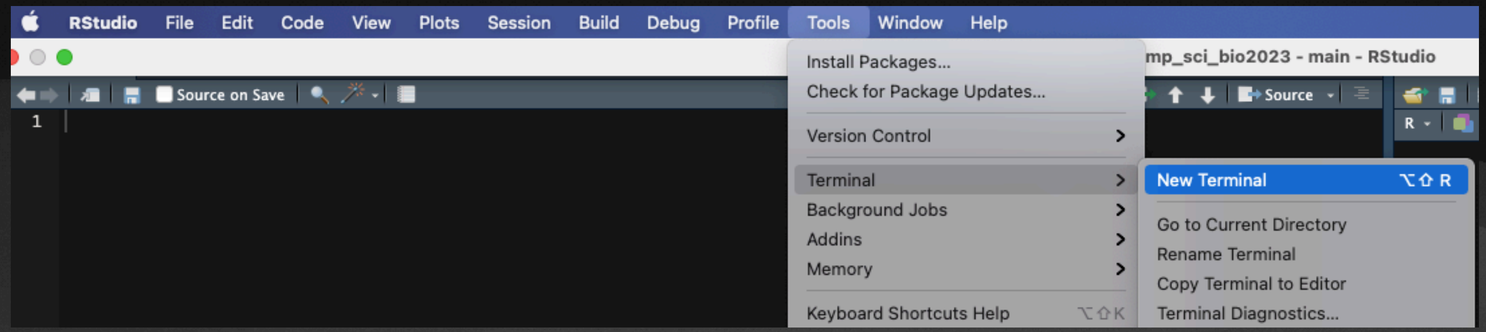


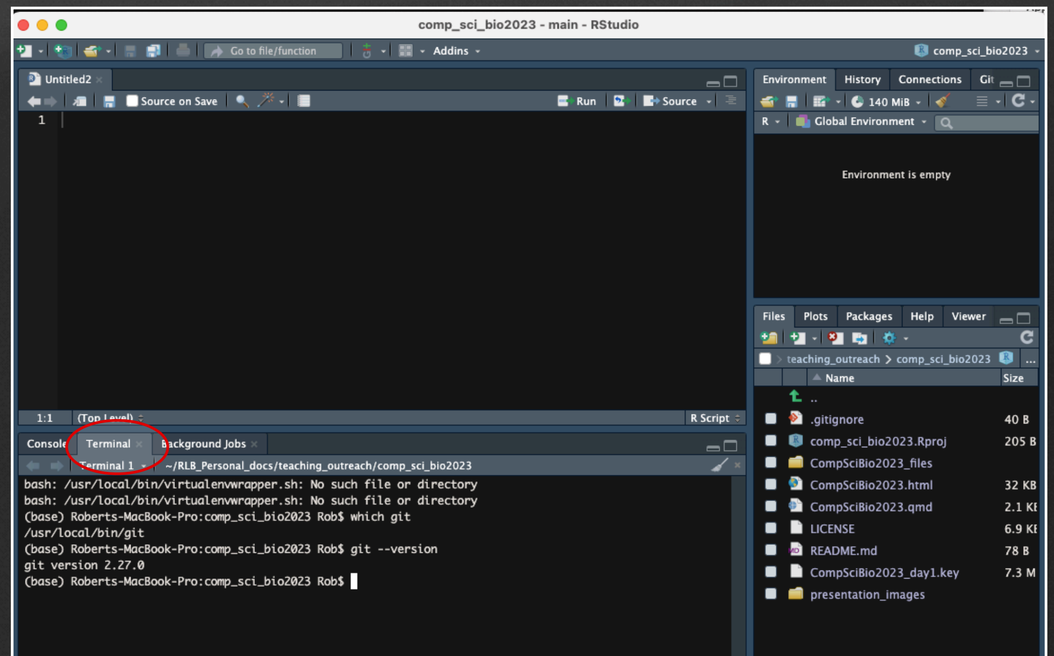
We will pause here until everyone is caught up

Set up

Get (or upgrade) Git (step 4 of 4)

- Open Rstudio
- Find the “Terminal” Window
- Type:
  $ which git
  response: “/usr/bin/git”
- Type:
  $ git --version
  response: “git version 2.41.0”
- If you don’t have Git (response: “git: command not found”) or it is not version 2.41.0, follow along to install/ upgrade





Set up

Get (or upgrade) Git (step 4 of 4)

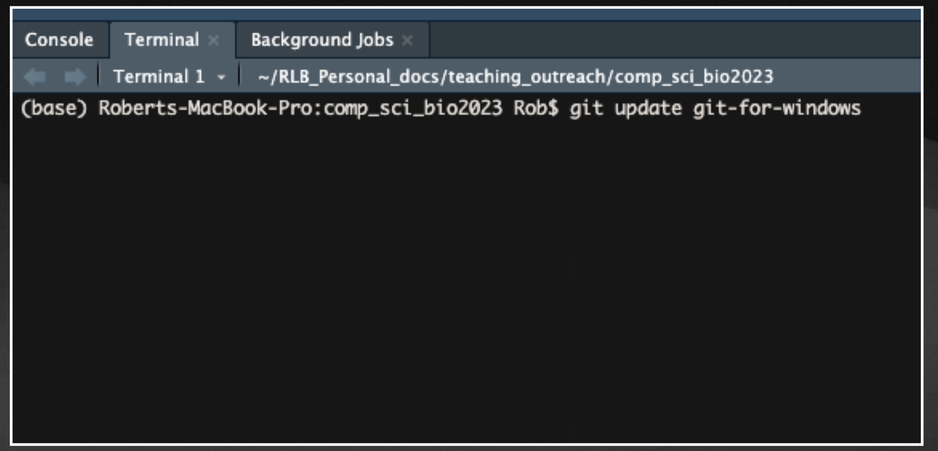
WINDOWS: Upgrading Git

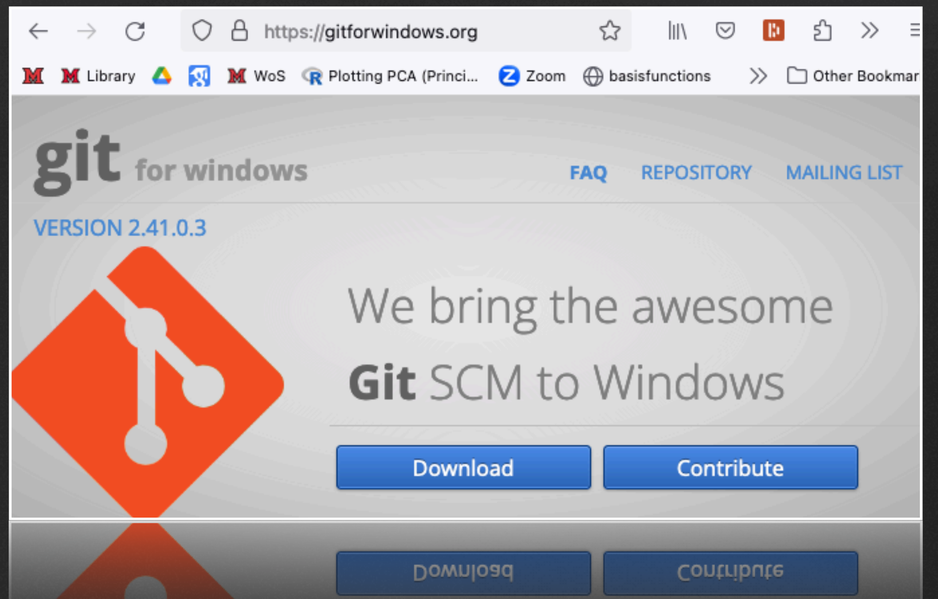
- In the terminal, type:

  $ git update git-for-windows

WINDOWS: Git Initial Install

- Go to https://gitforwindows.org
- Click Download & Install:
  - When asked to “adjust your PATH environment”, **select** “Git from the command line and also from 3rd-party software”
  - Install in the default location





Set up

Get (or upgrade) Git (step 4 of 4)

MAC: Upgrade or Install Git

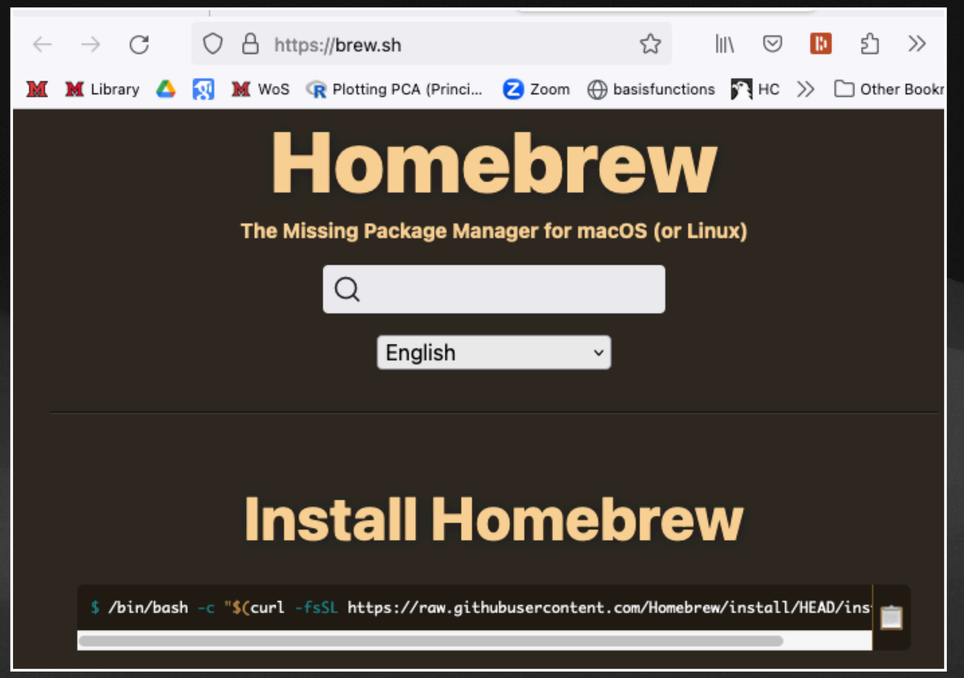
1. Install home-brew
    - Got to https://brew.sh
    - In the terminal, type:

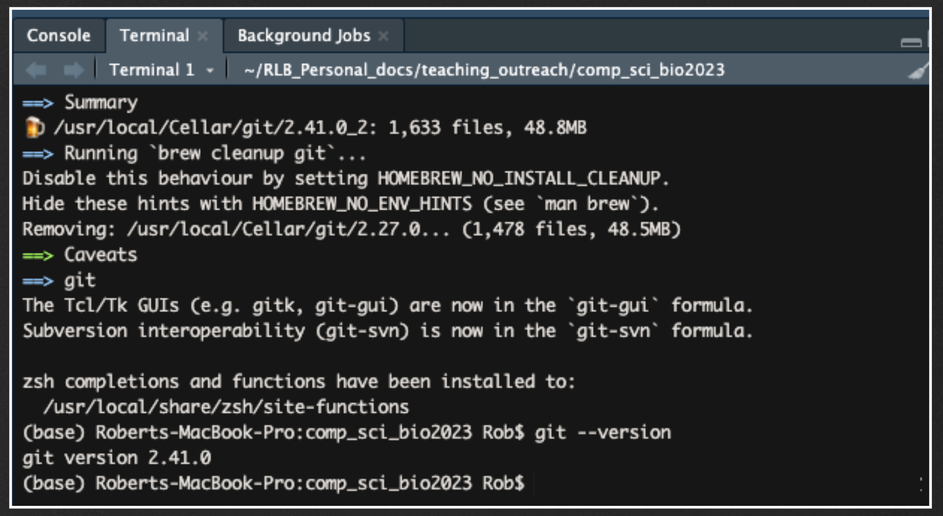
      $ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)" (you can copy this from the brew.sh site)

    - Enter your password when prompted
2. Install git
    - In the Terminal type:

      $ brew install git
3. Check that you have the updated version

      $ git --version





We will pause here until everyone is caught up

Course Outline

I. Introductions

II. Concepts of version control and collaborative coding

III. Sign up for a GitHub account

IV. Walk through and help installing/upgrading software

V. **Rstudio/GitHub integration**

VI. Version Control and Collaborative Coding Workflows

VII. Practice working collaboratively on a (non-coding) project

VIII. Bonus: project management boards, issues, milestones

Rstudio/GitHub integration

Configure Git

From the Terminal pane in Rstudio run these four lines:

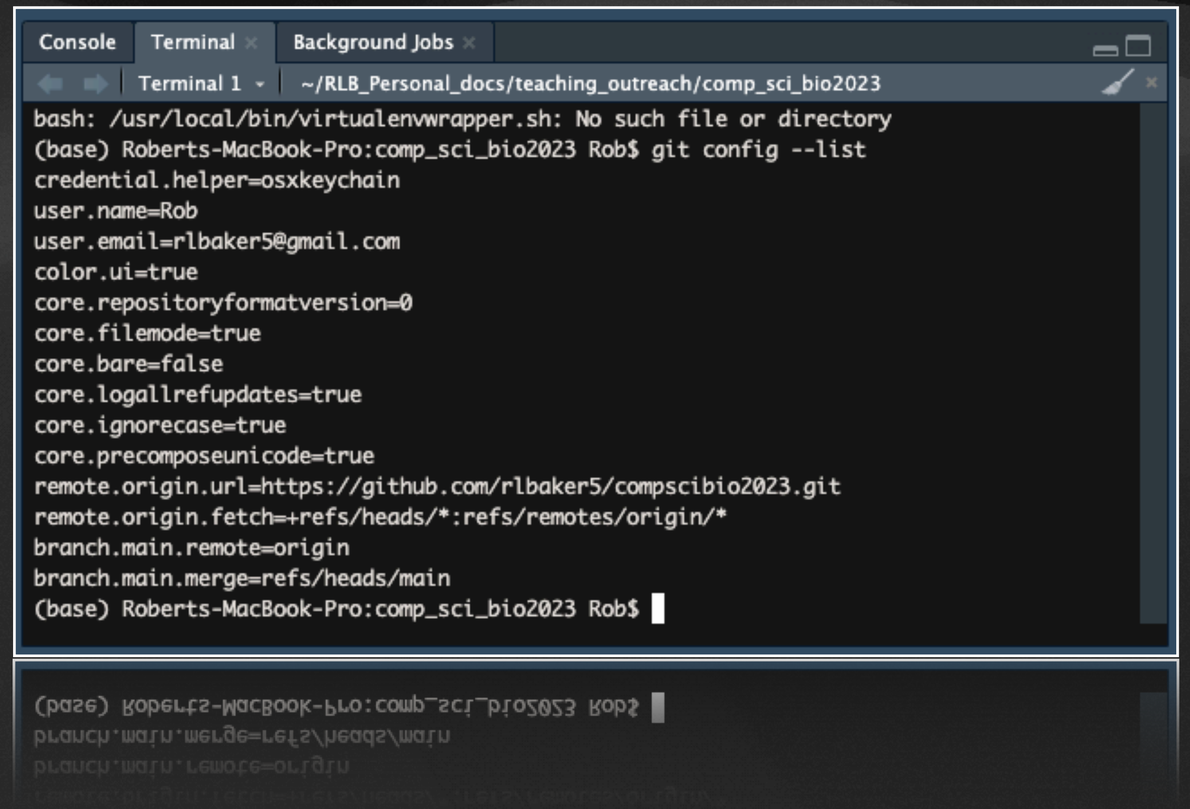
(For user.name and user.email use your GitHub username and email associated with your GitHub account)

git config --global user.name ‘rlbaker5’

git config --global user.email ‘rlbaker5@gmail.com'

git config --global --list

git config --global init.defaultBranch main



Rstudio/GitHub Integration

Get access to Git

- You can use SSL or HTTPS
- We will use HTTPS because it’s just easier
- You can switch back and forth at any time if you choose

Rstudio/GitHub Integration

Get access to Git

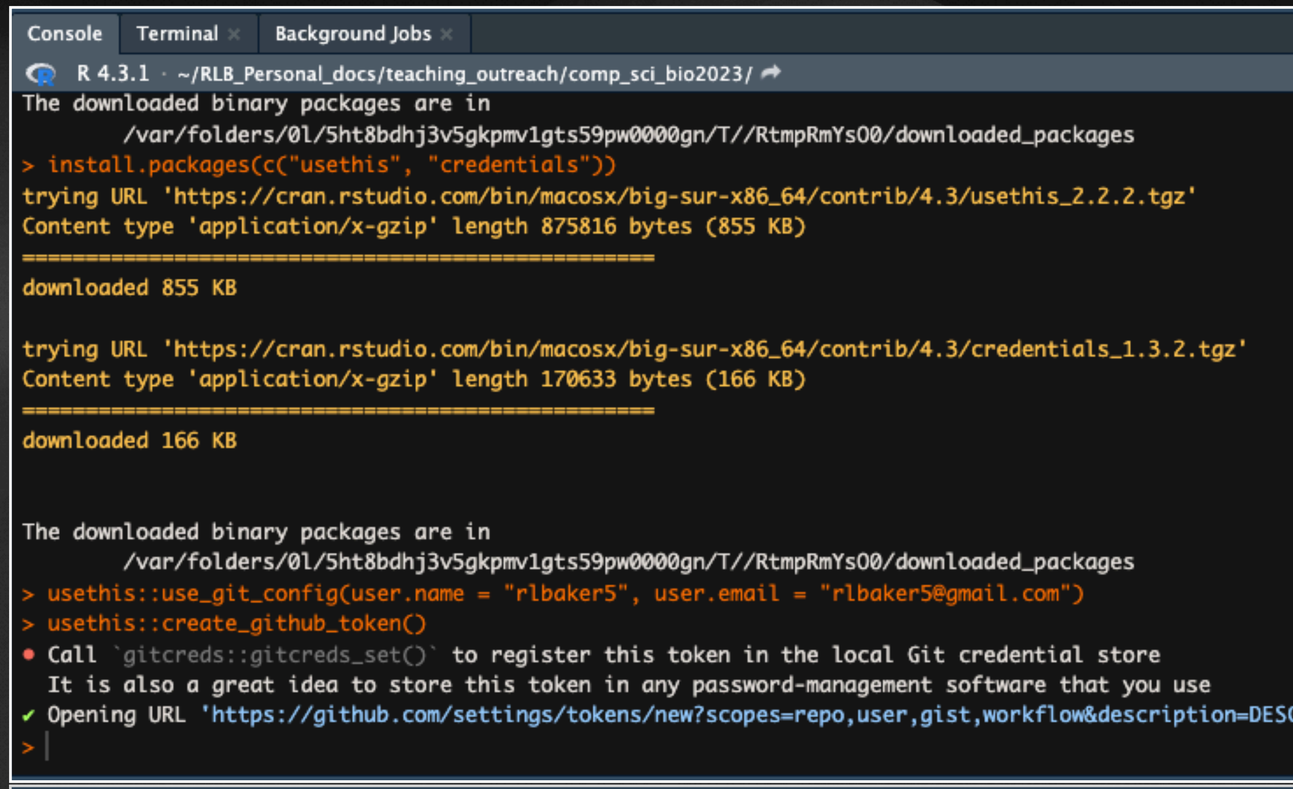
From the Console in Rstudio run the following 3 lines of code:

>install.packages(c("usethis", "credentials"))

>usethis::use_git_config(user.name = "YourName", user.email = "your@mail.com
(Use your GitHub username and email address)

>usethis::create_github_token()

- This should open a browser window for GitHub.com

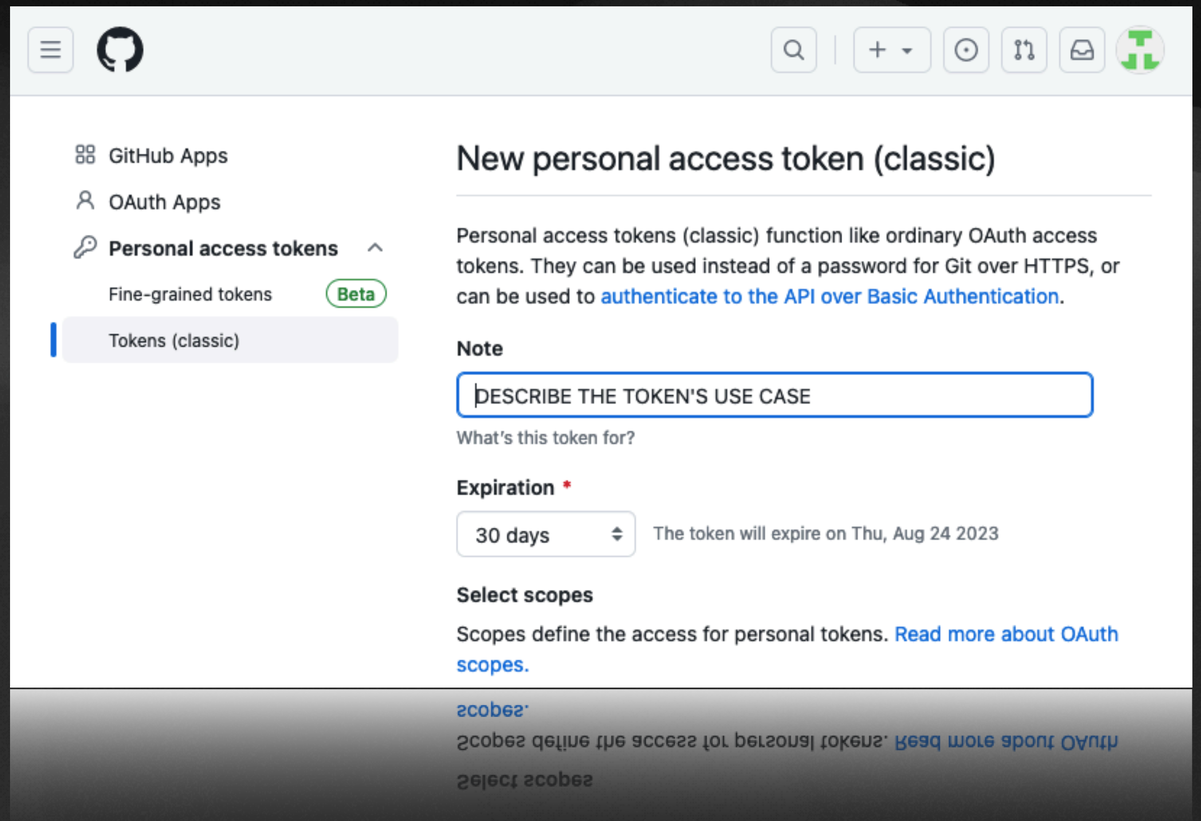


Rstudio/GitHub Integration

Get access to Git

After signing in, you will be brought to a “New personal access token (classic)” page

- Describe the token (“compscibio2023”)
- Set an expiration date
- Use the default “Scopes”
- Scroll down and click “Generate Token”

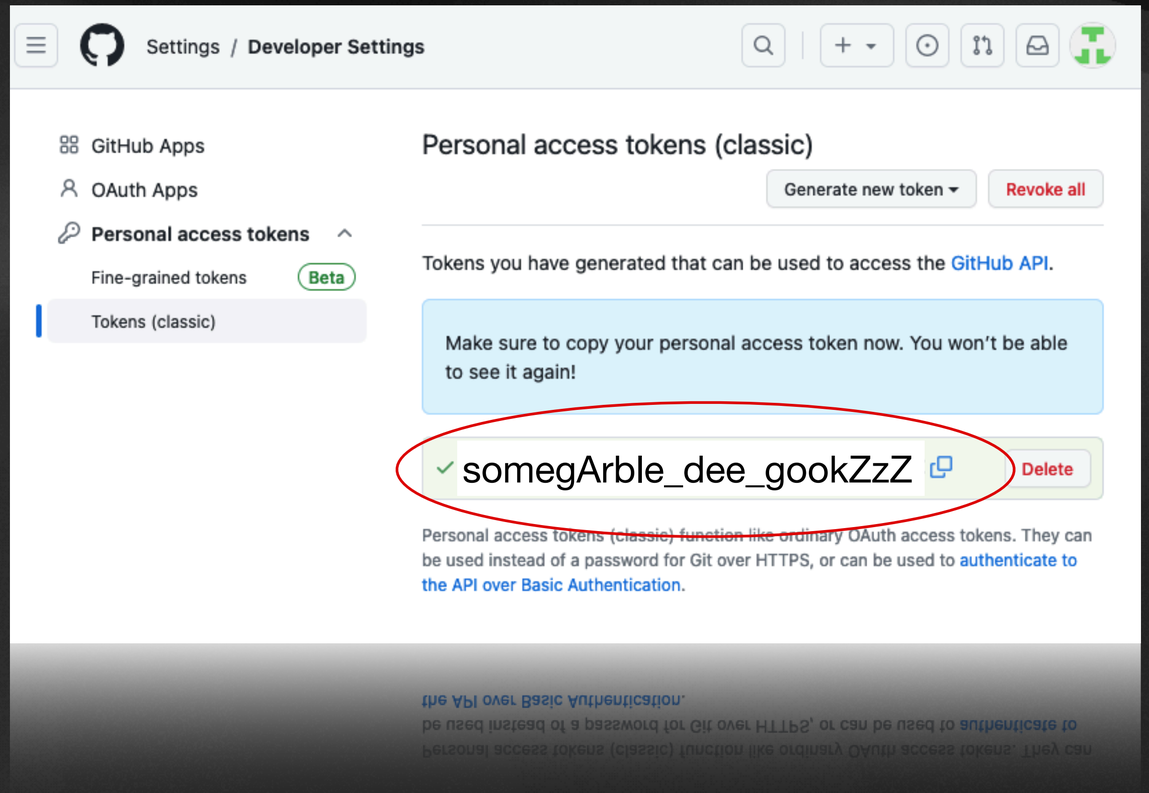


Rstudio/GitHub Integration

Get access to Git

Copy your personal access token (PAT) to your clipboard!

(Click the two linked squares next to the token)



Rstudio/GitHub Integration

Get access to Git

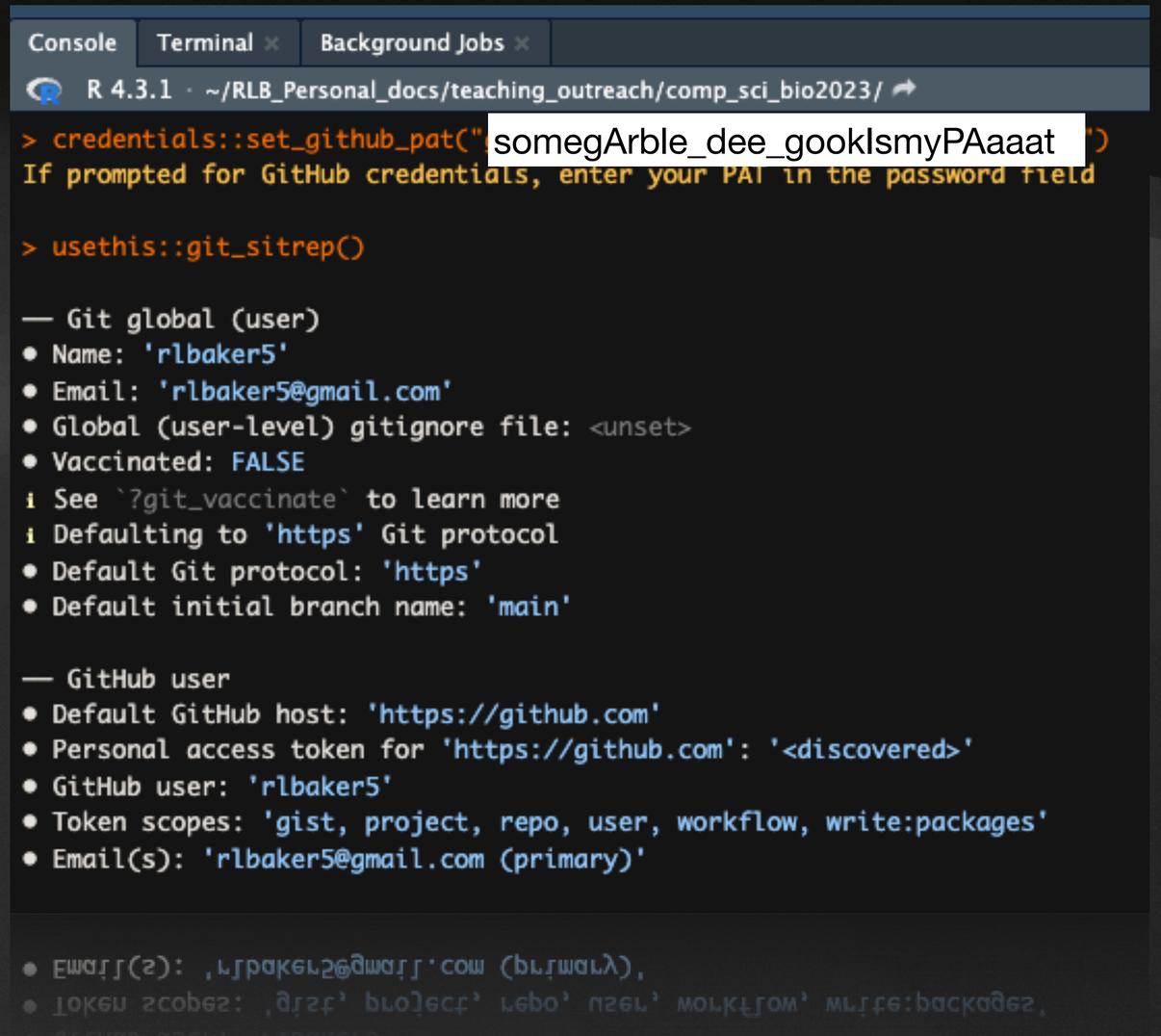
Back in the Rstudio console….

>credentials::set_github_pat(“<paste_your_pat>”)

#don’t forget the “” around your PAT!

(If prompted, enter your password)

>usethis::git_sitrep()



Rstudio/GitHub Integration

Get access to Git

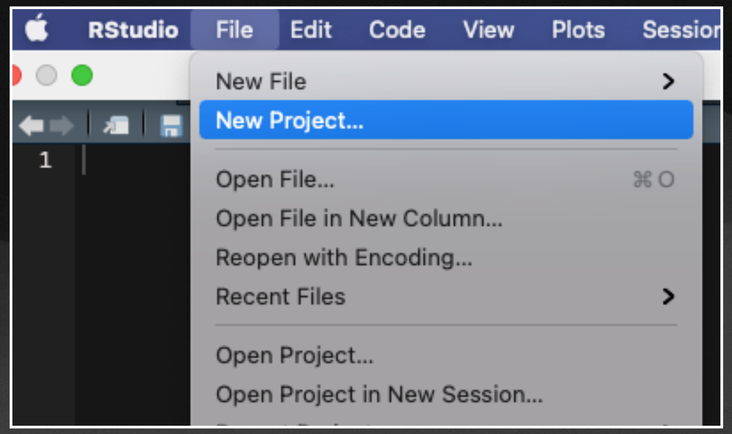
Did it work?

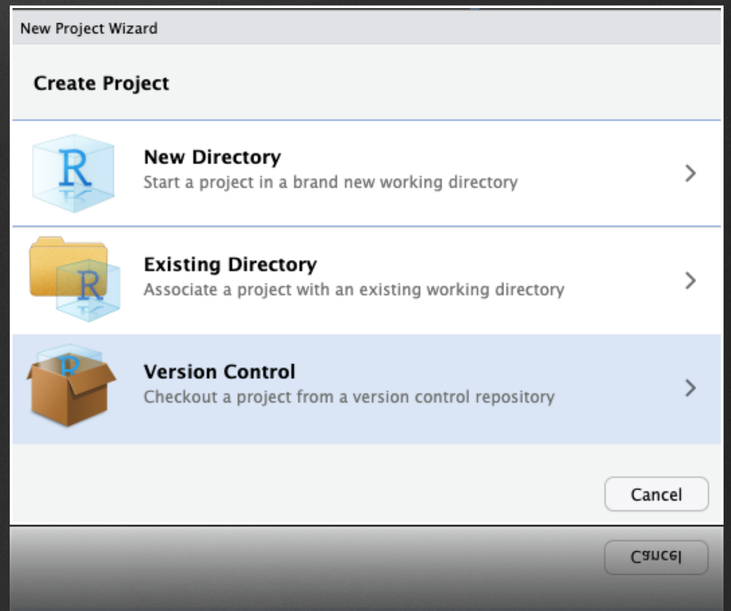
- In Rstudio go to File > New Project

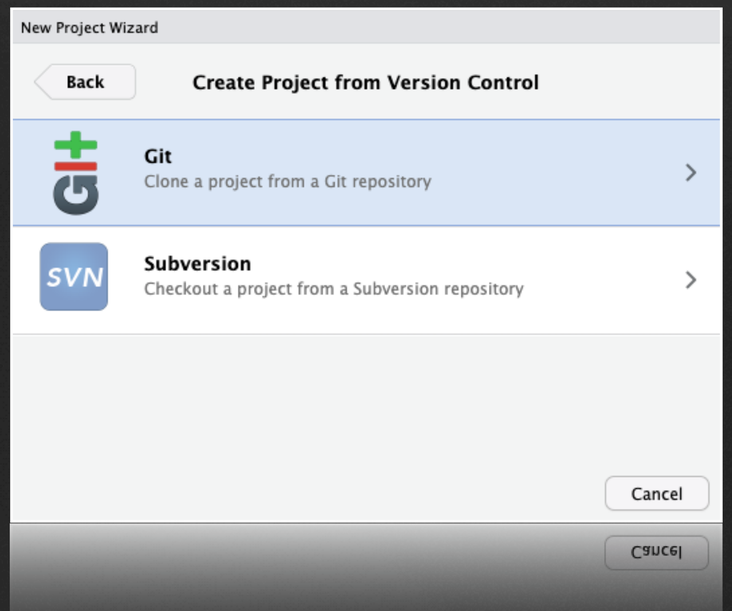
  If you see an option for Version Control, that is good!

- Select “Version Control”

  If you see an option to “Clone a project from a Git repository” that is good!







Rstudio/GitHub Integration

Get access to Git

Ruh-roh. It didn't work . . . .

1. Quit and **restart Rstudio**. . . seriously, just do it. If that didn't fix it. . .

2. Rstudio needs to be able to find git. Usually this is automatic, but you can do it manually. Find your git executable:

    - From the Terminal in Rstudio:

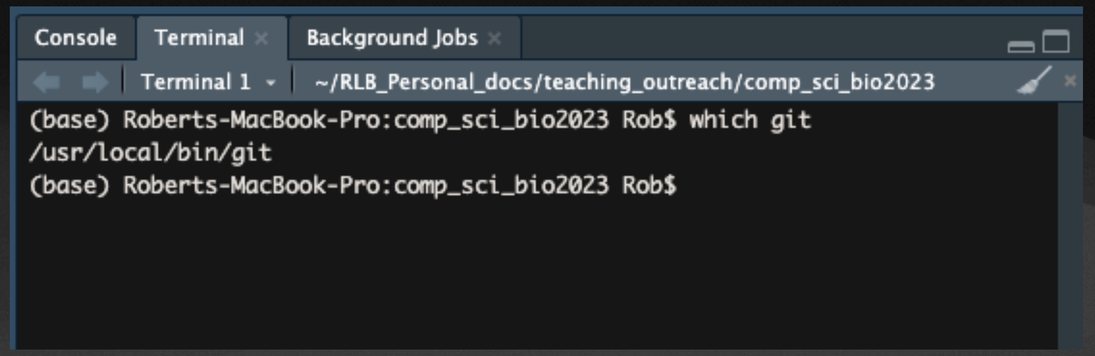
        Windows: $ where git

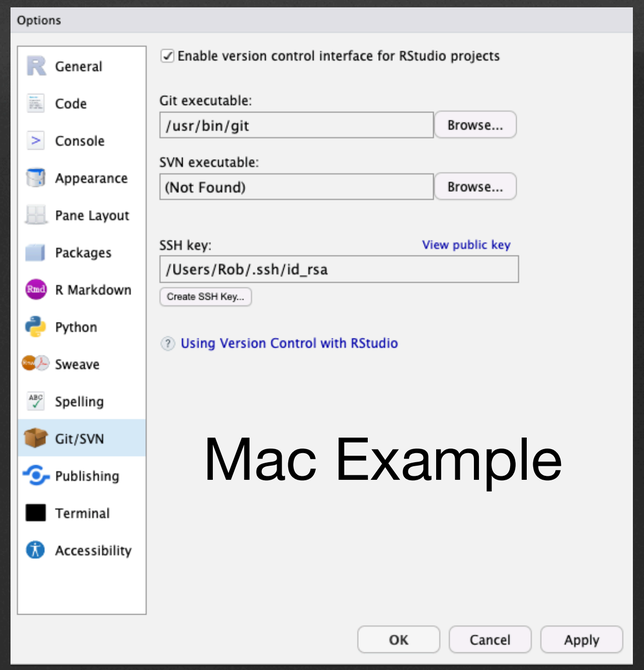
        Mac: $ which git

3. In Rstudio: Tools > Global Options > Git/SVN

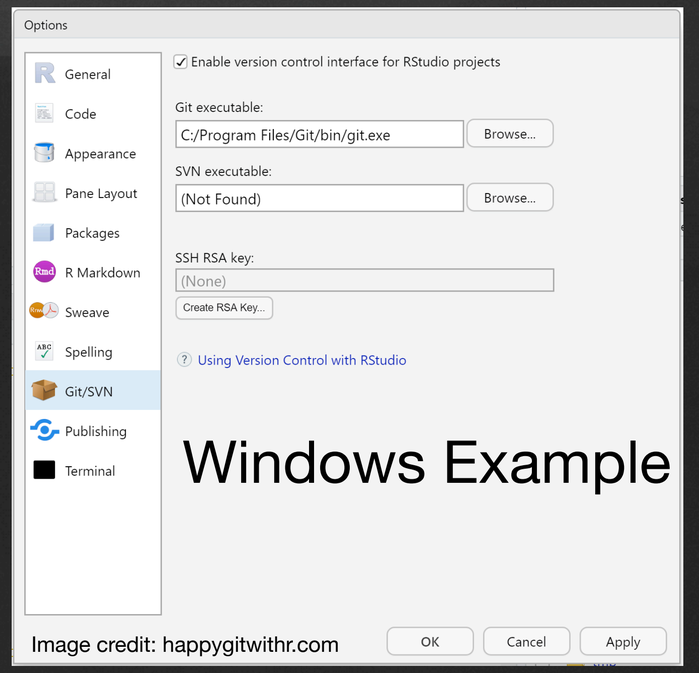
    - Make sure the Git executable path points to your git executable file

4. **Re-restart Rstudio** if you made changes





Mac Example



Windows Example

Image credit: happygitwithr.com

Course Outline

I. Introductions

II. Concepts of version control and collaborative coding

III. Sign up for a GitHub account

IV. Walk through and help installing/upgrading software

V. Rstudio/GitHub integration

VI. **Version Control and Collaborative Coding Workflows**

VII. Practice working collaboratively on a (non-coding) project

VIII. Bonus: project management boards, issues, milestones

Version Control

Working on your own project:

- Start with a repository (typically remote)

Version Control

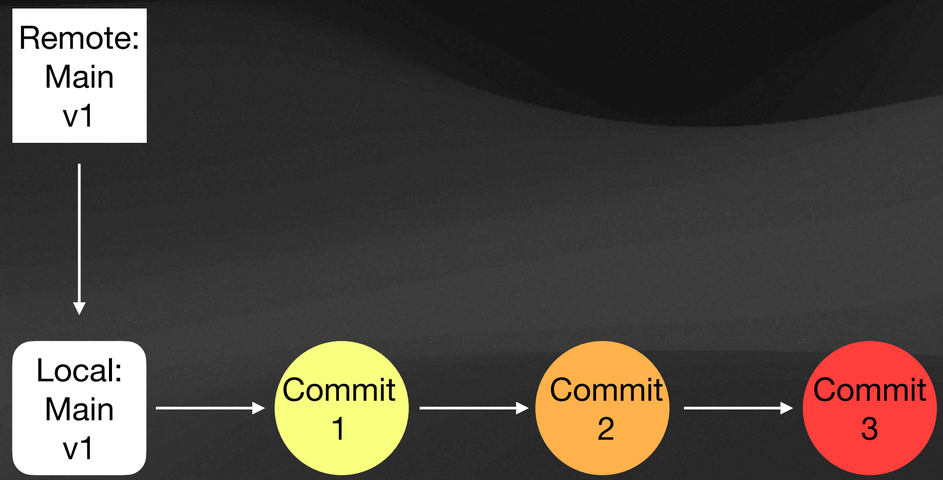
Working on your own project:

- Start with a repository (typically remote)
- Clone (or “pull”) the repository to local machine

Version Control

Working on your own project:

- Start with a repository (typically remote)
- Clone (or “pull”) the repository to local machine

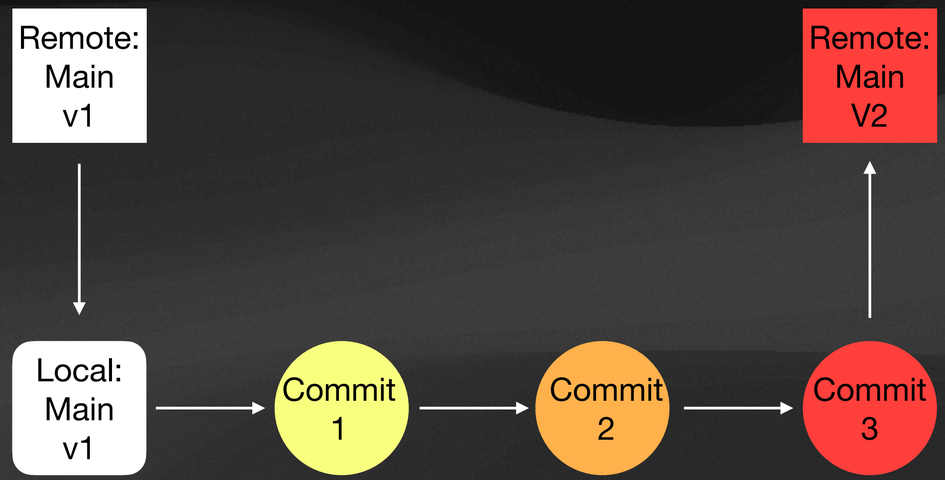


- Work locally, making snapshots as you go (“commits”)

Version Control

Working on your own project:

- Start with a repository (typically remote)
- Clone (or “pull”) the repository to local machine

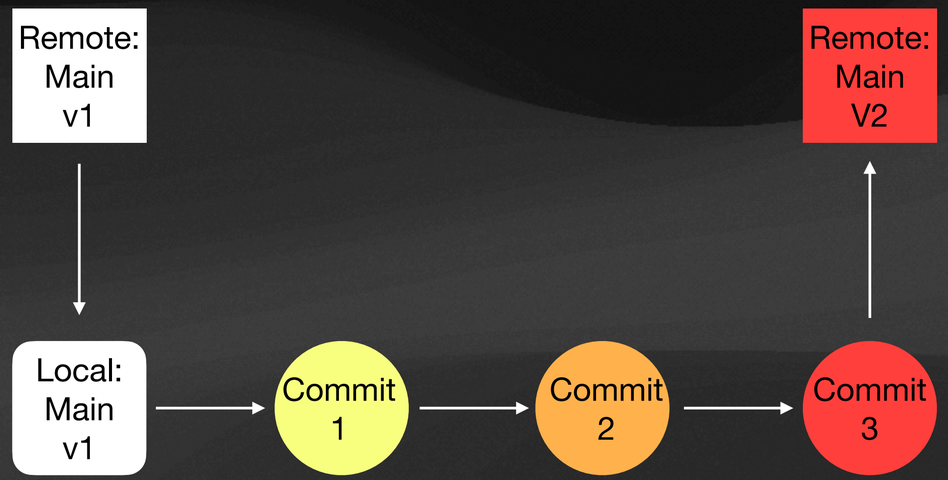


- “Push” changes to your remote repository
- Work locally, making snapshots as you go (“commits”)

Version Control

Working on your own project:

- Start with a repository (typically remote)
- Clone (or “pull”) the repository to local machine

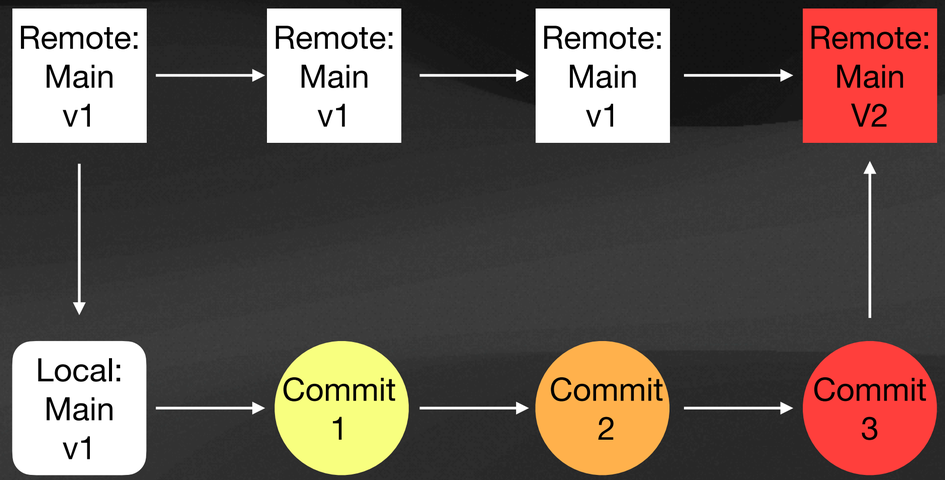


- Work locally, making snapshots as you go (“commits”)
- “Push” changes to your remote repository
- GitHub now reflects most recent snapshot & all commits

Version Control

Working on your own project:

- Start with a repository (typically remote)
- Clone (or “pull”) the repository to local machine

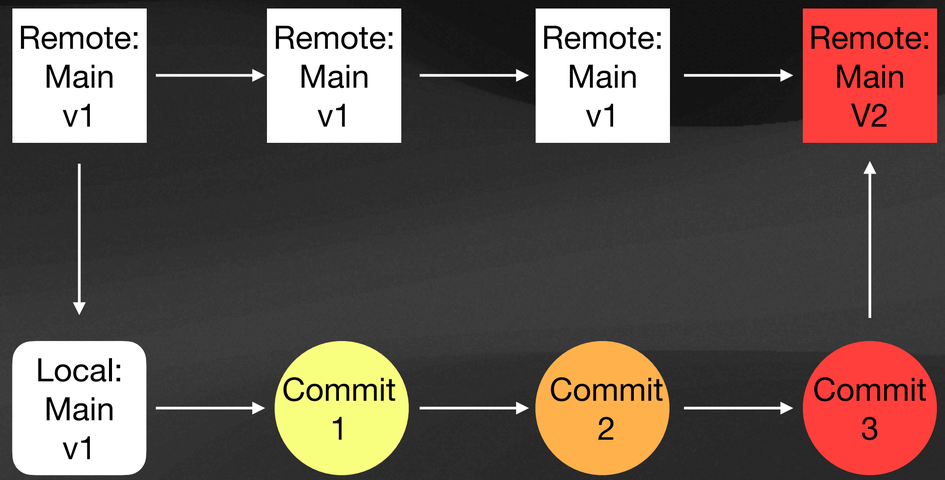


- Work locally, making snapshots as you go (“commits”)
- “Push” changes to your remote repository
- GitHub now reflects most recent snapshot & all commits
- Remote repository available throughout development

Version Control

Working on your own project:

- Start with a repository (typically remote)
- Clone (or “pull”) the repository to local machine



- Work locally, making snapshots as you go (“commits”)
- “Push” changes to your remote repository
- GitHub now reflects most recent snapshot & all commits
- Remote repository available throughout development
- Version Control: commit history maintained; changes to be rolled back

Collaborative Coding

Forking Workflow

- Lightweight, relatively easy
- Highlight your own work on your own GitHub account
- Safe! You aren’t changing anything on the main repo until you’re really ready; you can test it out after upload
- Can request code reviews, etc before your changes get incorporated into the main repo

Collaborative Coding

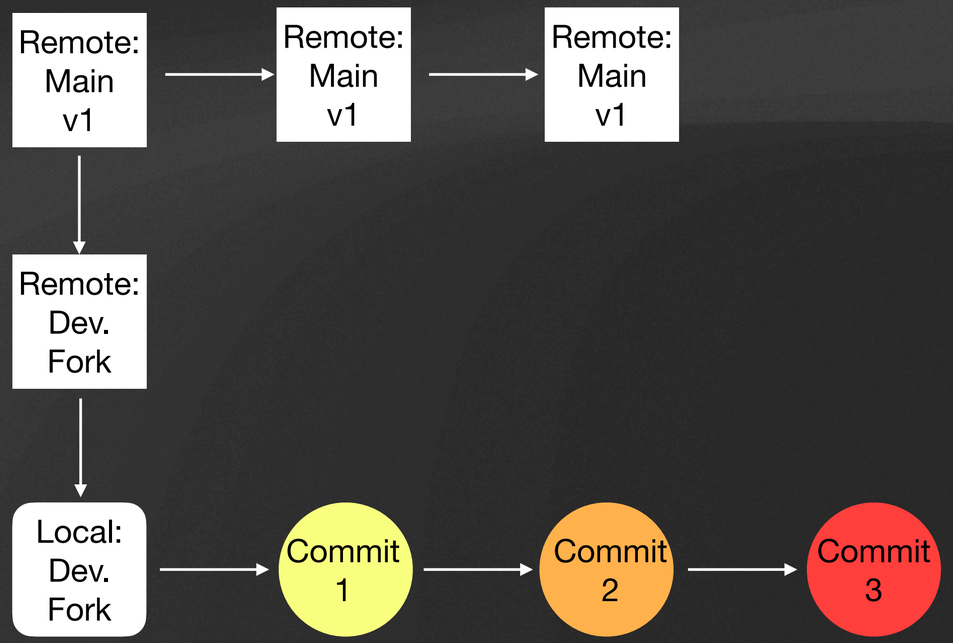
Forking Workflow

- Remote Main v1 - might not be your repo (maybe a lab repo?)
- “Fork” the lab repo to a repo to create a copy on your own GitHub account
- Clone (or pull) your development fork to your local machine

Collaborative Coding

Forking Workflow

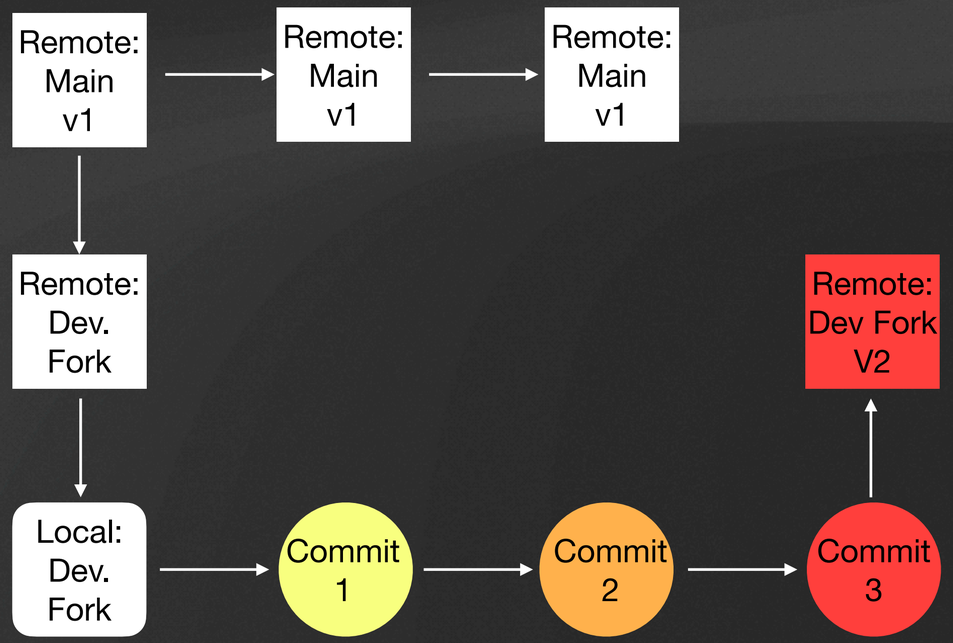
- Remote Main v1 - might not be your repo (maybe a lab repo?)
- “Fork” the lab repo to a repo to create a copy on your own GitHub account
- Clone (or pull) your development fork to your local machine



Collaborative Coding

Forking Workflow

- Remote Main v1 - might not be your repo (maybe a lab repo?)
- "**Fork**" the lab repo to a repo to create a copy on your own GitHub account
- "**Clone**" (or "**Pull**") your development fork to your local machine

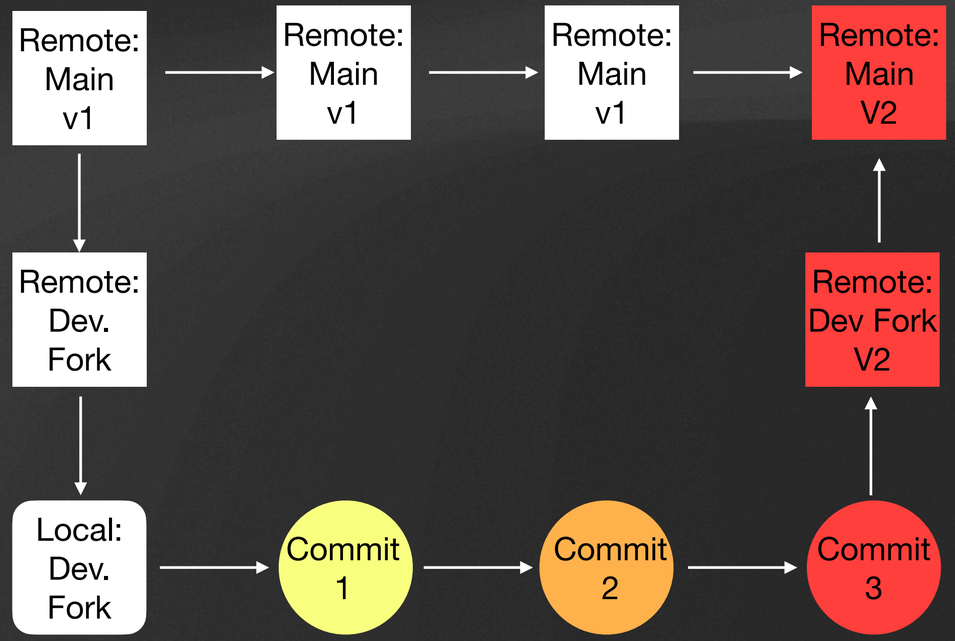


- Initiate a "**Pull request**": ask that someone incorporate your changes to the lab account (NOT a PULL)
- "**Push**" your changes to the Forked copy of the repo on your own GitHub account
- Make your edits ("**commits**") locally as before

Collaborative Coding

Forking Workflow

- Remote Main v1 - might not be your repo (maybe a lab repo?)
- "**Fork**" the lab repo to a repo to create a copy on your own GitHub account
- "**Clone**" (or "**Pull**") your development fork to your local machine

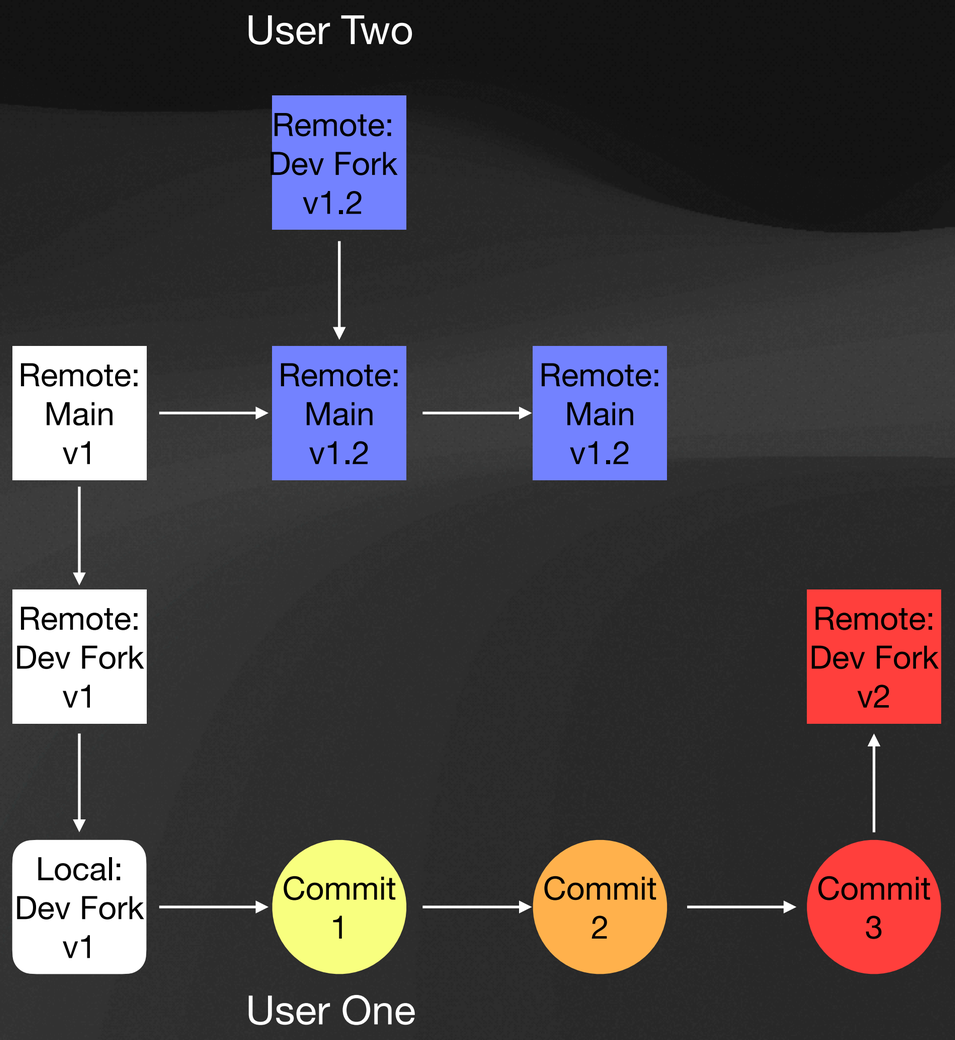


- Initiate a "**Pull request**": ask that someone incorporate your changes to the lab account (NOT a PULL)
- "**Push**" your changes to the Forked copy of the repo on your own GitHub account
- Make your edits ("**commits**") locally as before

Collaborative Coding

Forking Workflow

- Remote Main v1 - might not be your repo (maybe a lab repo?)
- “**Fork**” the lab repo to a repo to create a copy on your own GitHub account
- “**Clone**” (or “**Pull**”) your development fork to your local machine

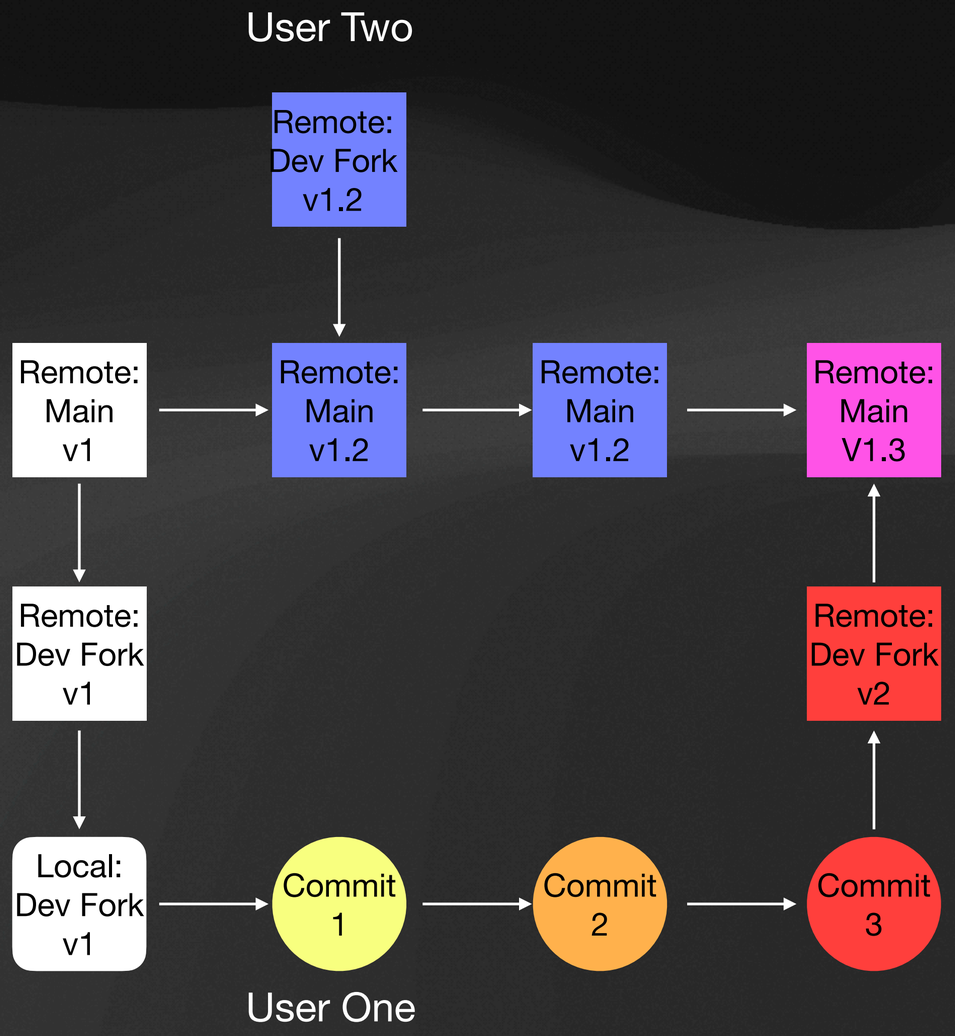


- “**Push**” your changes to the Forked copy of the repo on your own GitHub account
- Make your edits (“**commits**”) locally as before

Collaborative Coding

Forking Workflow

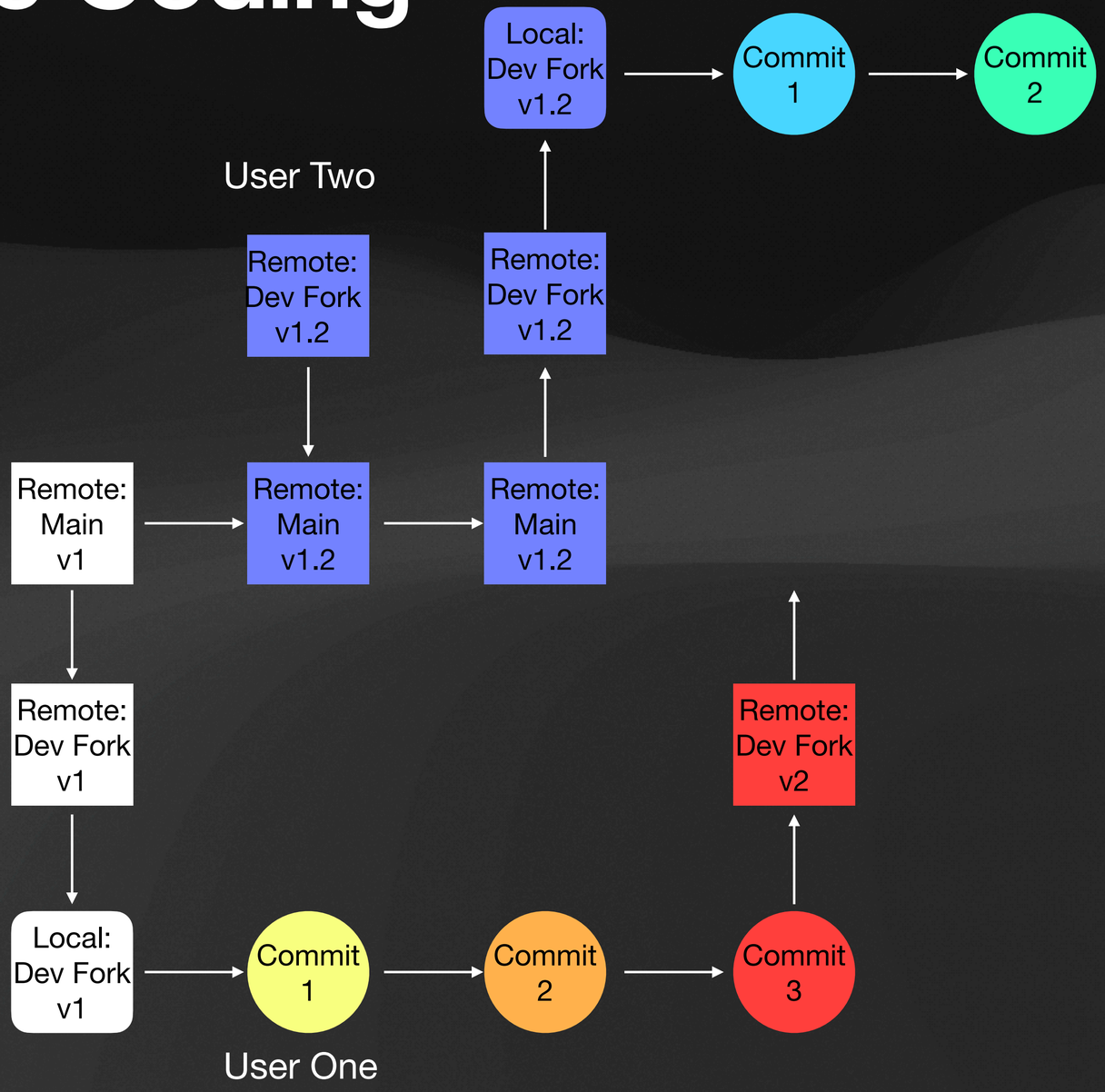
- Remote Main v1 - might not be your repo (maybe a lab repo?)
- "**Fork**" the lab repo to a repo to create a copy on your own GitHub account
- "**Clone**" (or "**Pull**") your development fork to your local machine



- Initiate a "**Pull request**": ask that someone incorporate your changes to the lab account (NOT a PULL)
- "**Push**" your changes to the Forked copy of the repo on your own GitHub account
- Make your edits ("**commits**") locally as before

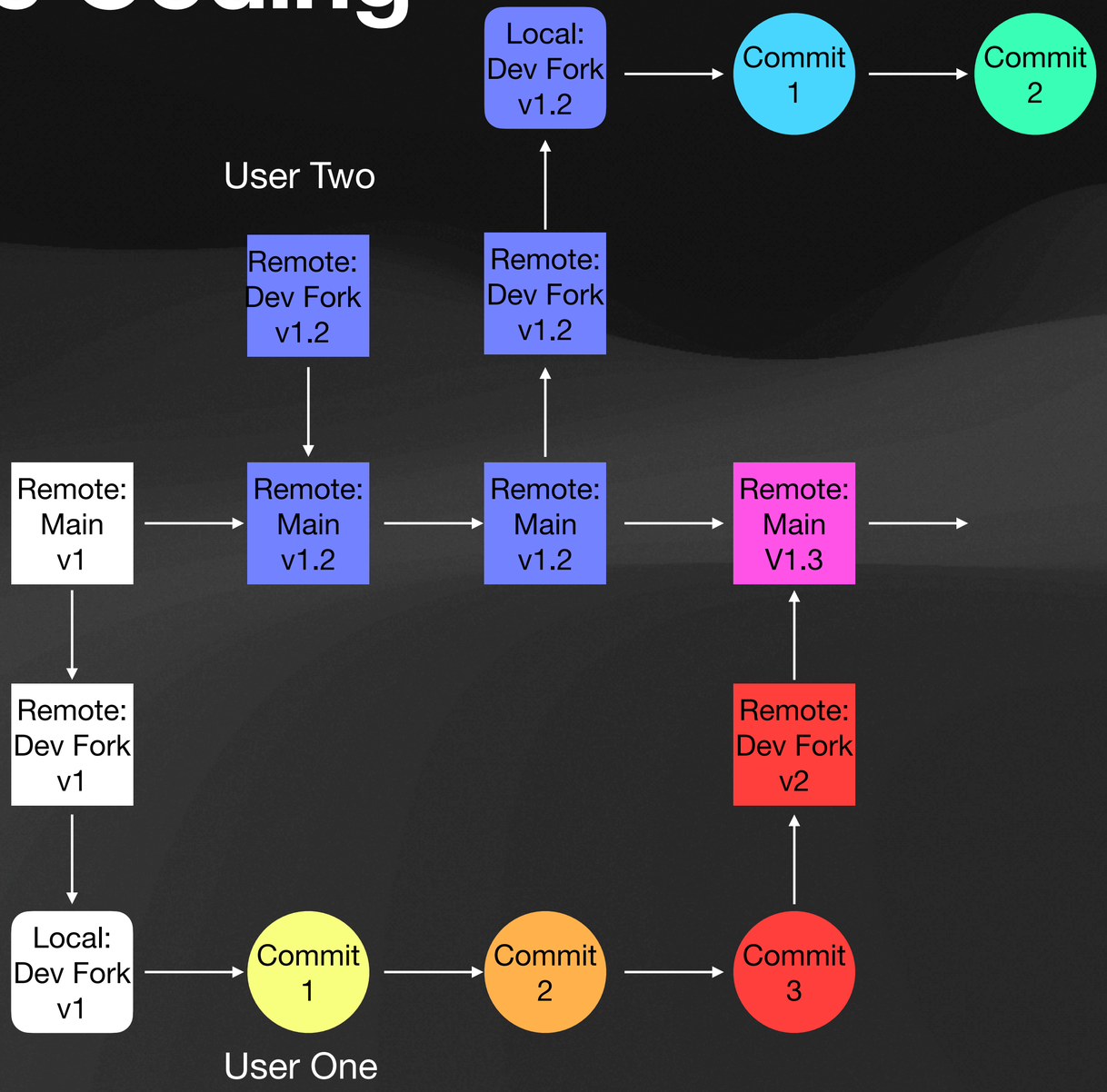
Collaborative Coding

Forking Workflow



Collaborative Coding

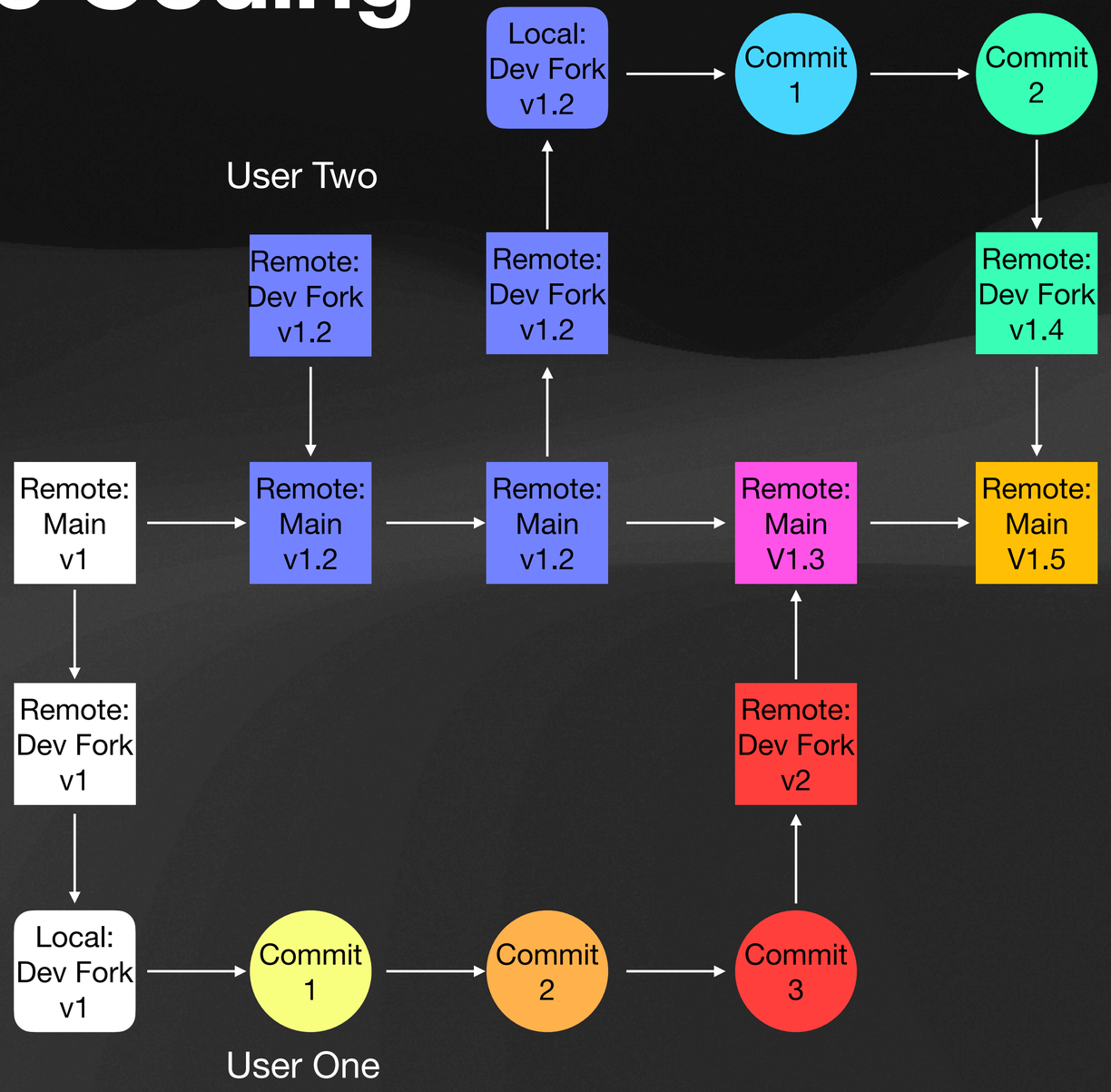
Forking Workflow



Collaborative Coding

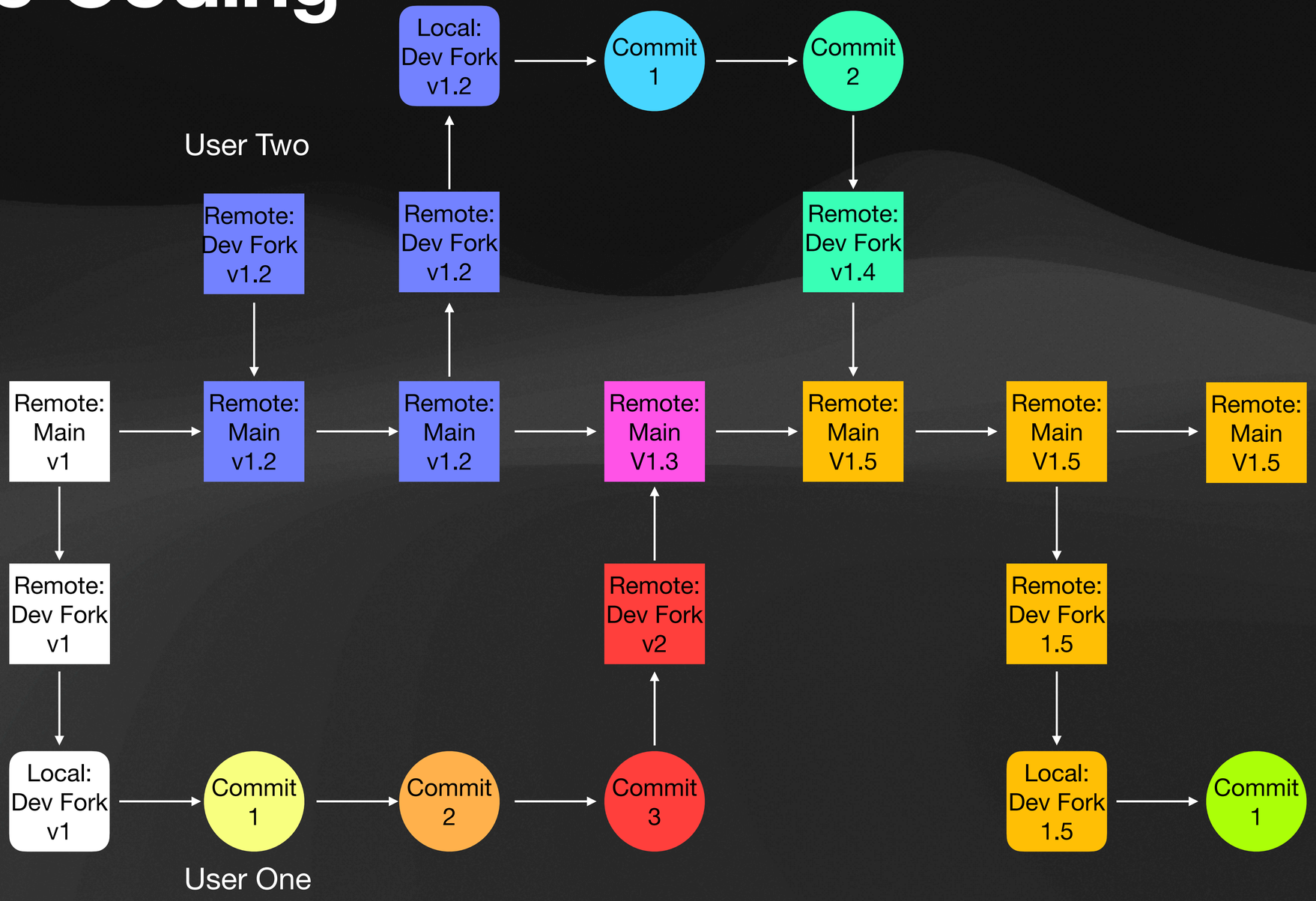
Forking Workflow

- Remote Main v1 - might not be your repo (maybe a lab repo?)
- “**Fork**” the lab repo to a repo to create a copy on your own GitHub account
- “**Clone**” (or “**Pull**”) your development fork to your local machine



Collaborative Coding

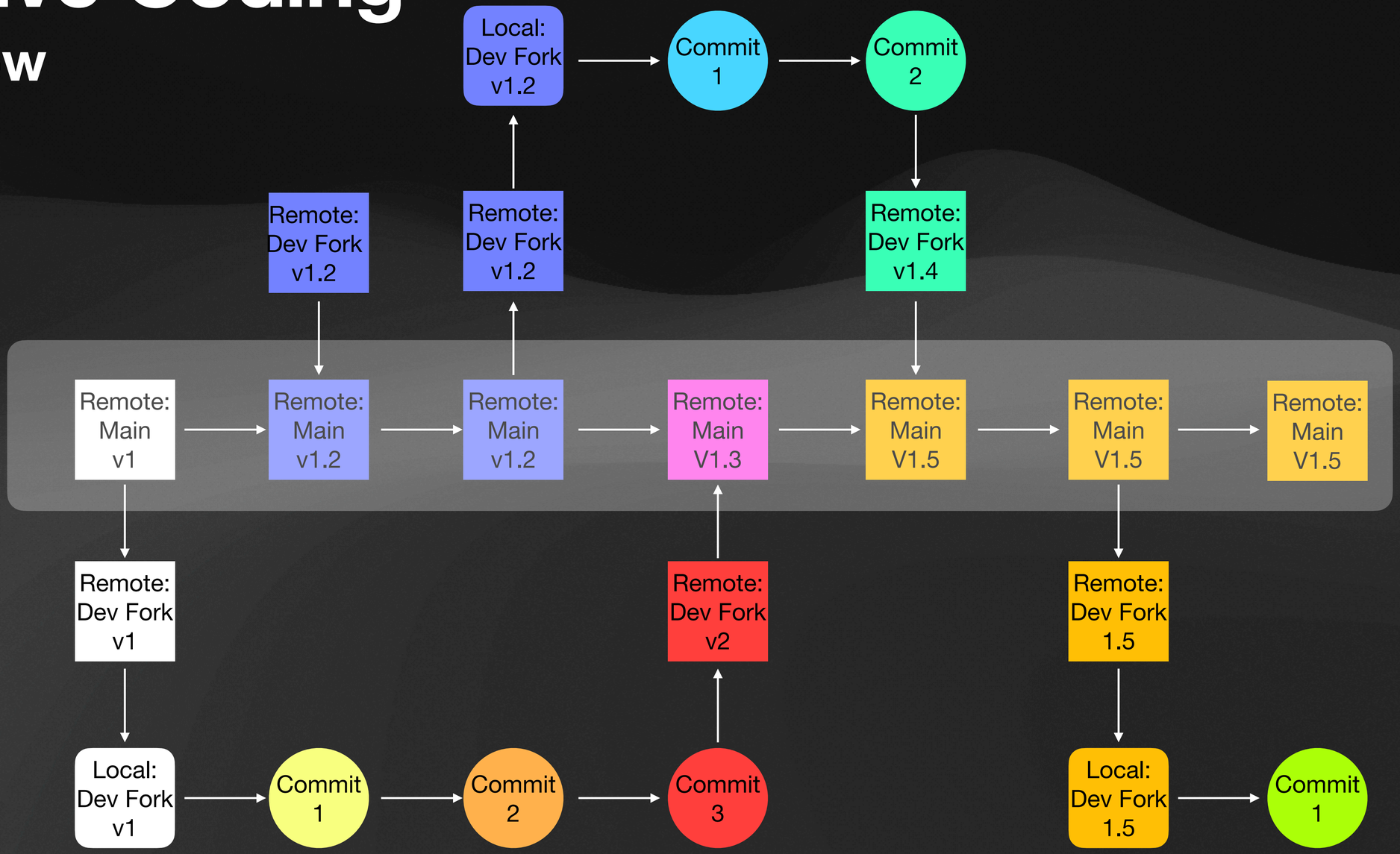
Forking Workflow



Collaborative Coding

Forking Workflow

- Two (or more) users can simultaneously collaborate on the same project
- The main branch is always available to the public
- Multiple different development forks can be worked on simultaneously



Version Control And Collaborative Coding

Terminology

- Repository (repo) - managed collection of files
- Version Control - keeping track of changes in files
- Fork - create a copy of a repo on a different account
- Branch - A new line of development; may be on a different repo or account (result of a Fork) or may be in the same repo.
- Clone - create a copy of a repo on a local machine
- Pull - pull all the changes from a repo to your local machine
- Commit - a versioned change, with a comment
- Push - move all the commits to your branch
- Pull Request - Request that your (modified) Forked branch be incorporated into a the main branch
- Merge - the process of reconciling two branches upon a Push or a Pull Request
- Merge conflict - when the two branches cannot automatically be reconciled and must be manually adjusted

Course Outline

I. Introductions

II. Concepts of version control and collaborative coding

III. Sign up for a GitHub account

IV. Walk through and help installing/upgrading software

V. Rstudio/GitHub integration

VI. Version Control and Collaborative Coding Workflows

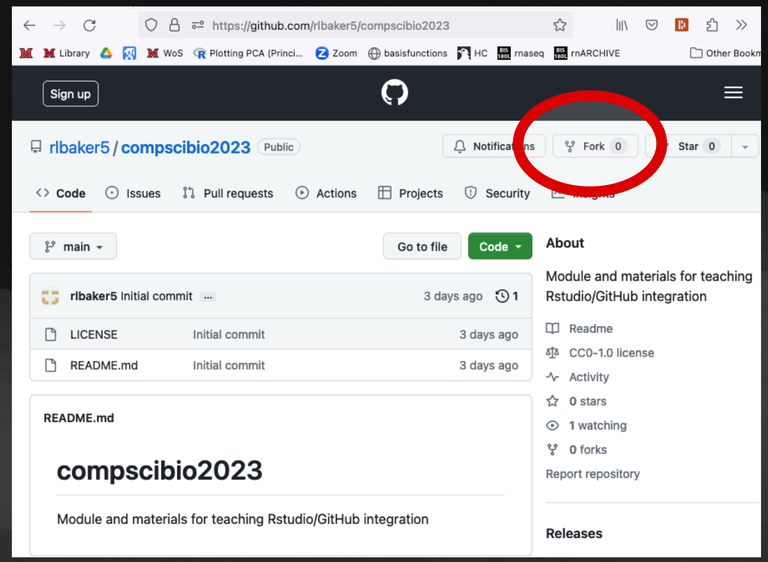
VII. **Practice: collaborative version control**

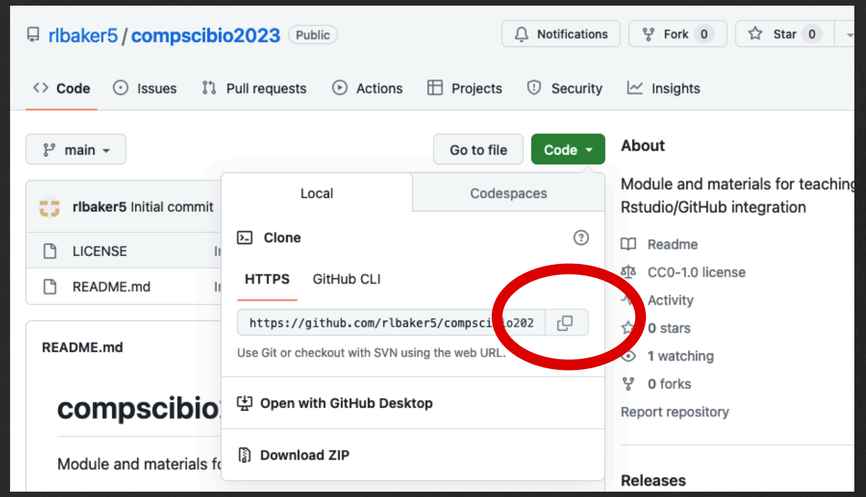
VIII. Bonus: project management boards, issues, milestones

Practice: collaborative version control

Fork a Repo

- Log in to your GitHub.com account
- Navigate to: GitHub.com/rlbaker5/compscibio2023
- Fork the repo to your own account
- Navigate to your own profile and enter the newly forked repo (github.com/<username>/compscibio2023)
- Use the green “Code” drop-down to copy the URL by clicking on the interlocking squares

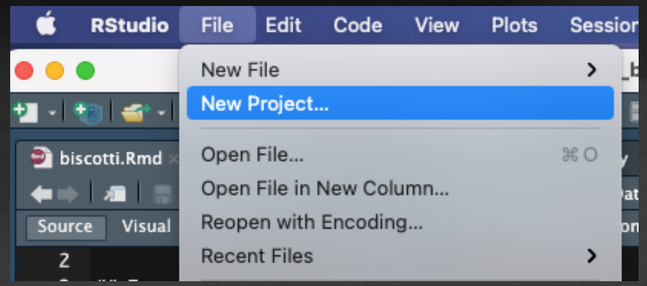


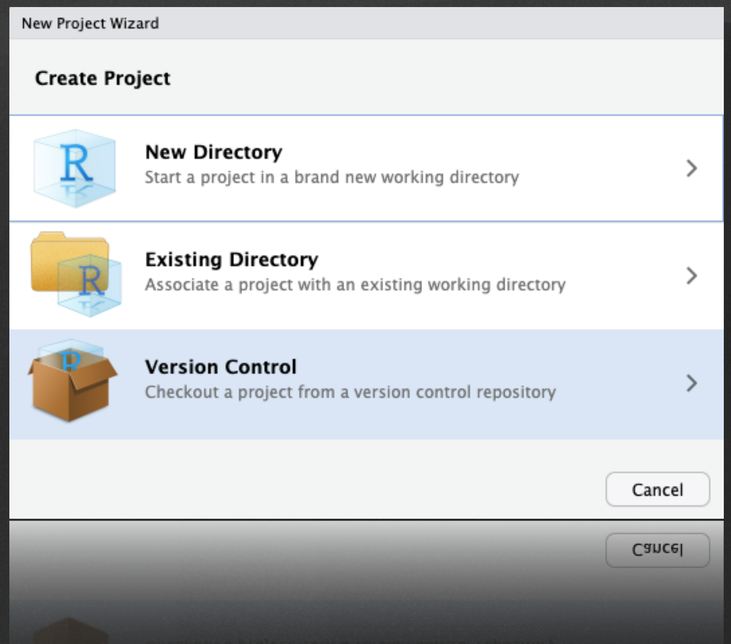


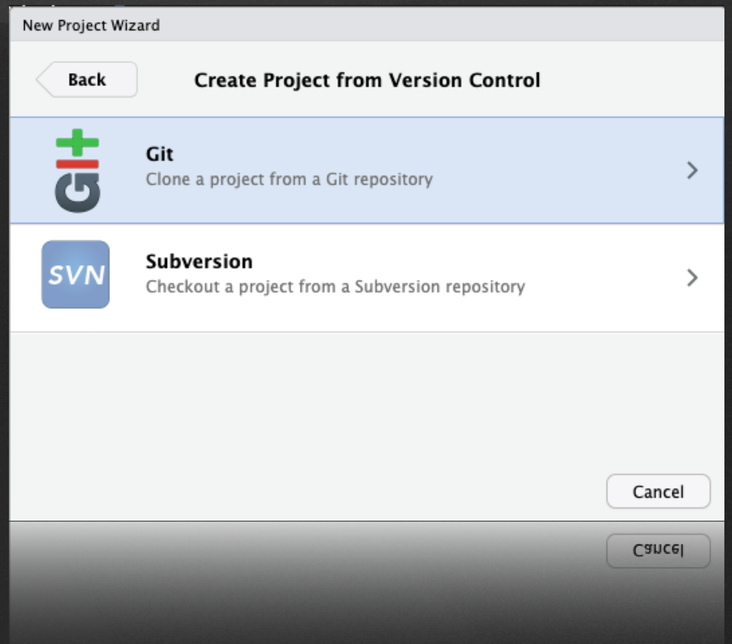
Practice: collaborative version control

Clone a Repo

- Open Rstudio
- Using From the “File” drop-down menu, select “New Project”
- Select “Version Control”
- Then choose “Git (“clone a project from a Git repository”)



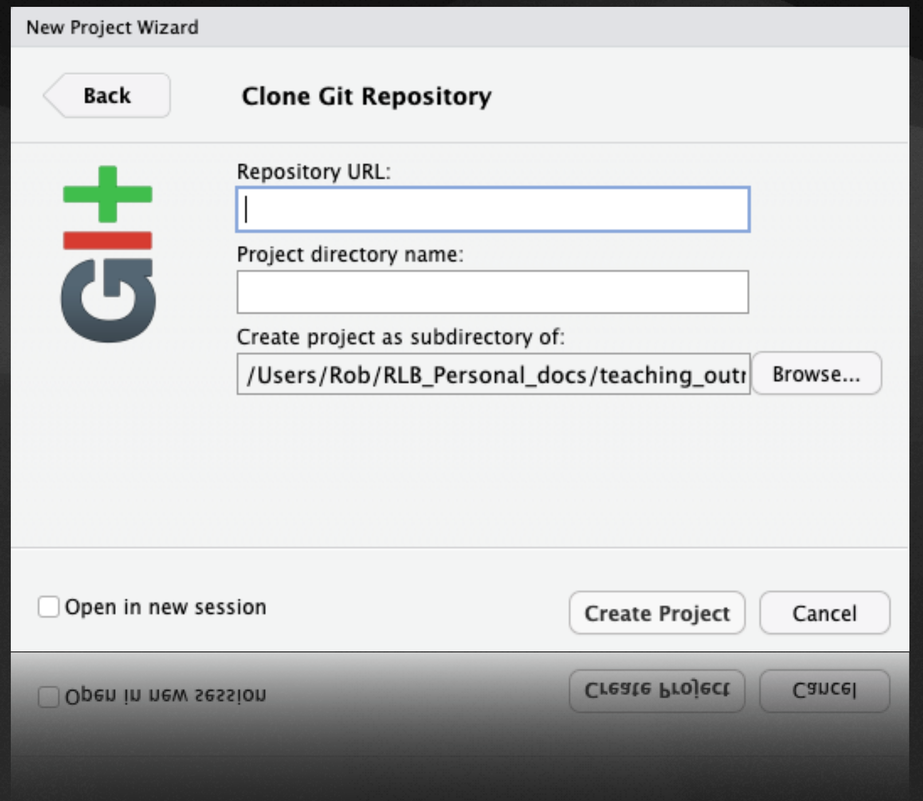




Practice: collaborative version control

Clone a Repo

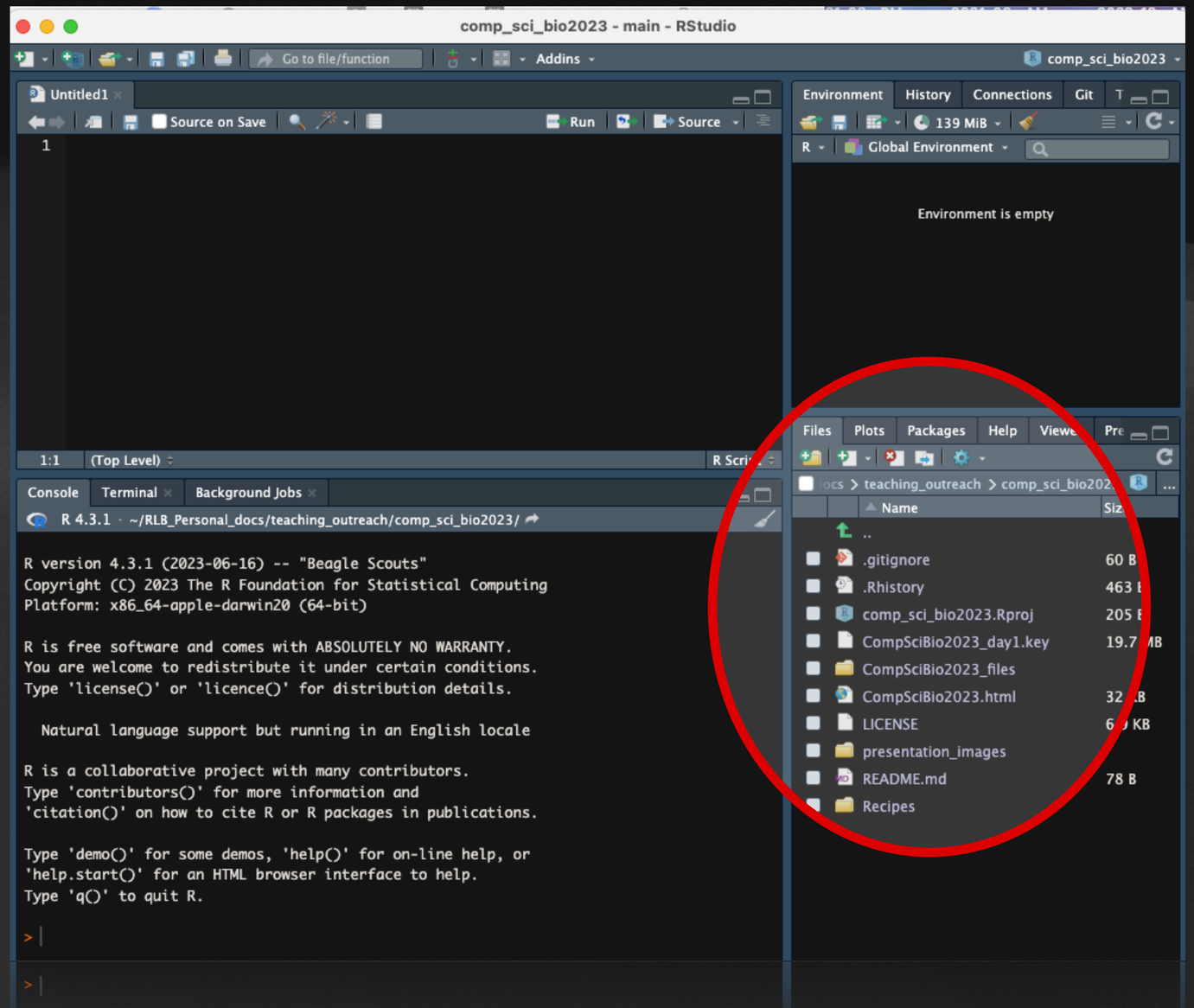
- Paste in the repo URL
- Give your project a name (e.g. “compscibio2023_dev”)
- Select a location where you want the project to live on your hard drive
- Click “Create Project”



Practice: collaborative version control

Clone a Repo

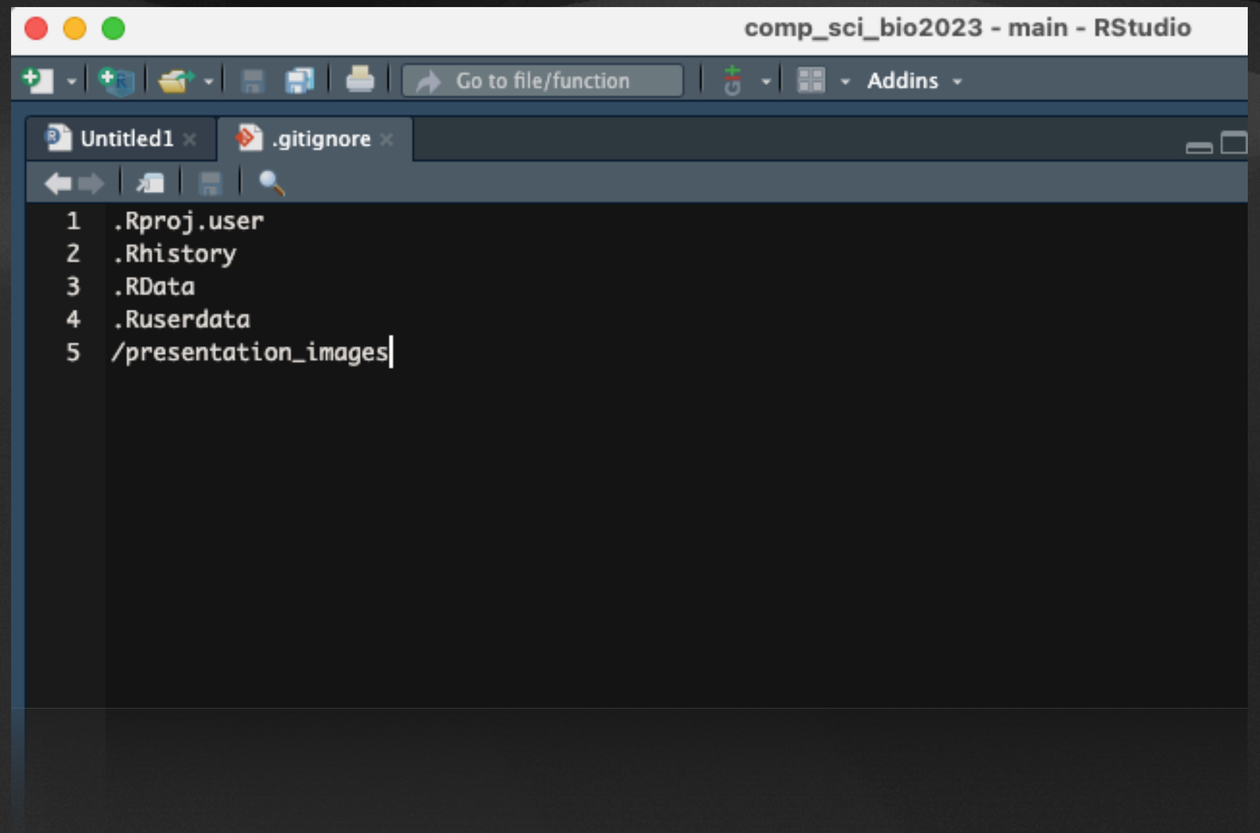
- Your Rstudio should look some thing like this (probably with a different color scheme)
- But notice all the files from the repo are now listed under Files and are now on your local machine!
- (Your files may differ slightly from the ones shown here)



Practice: collaborative version control

.gitignore

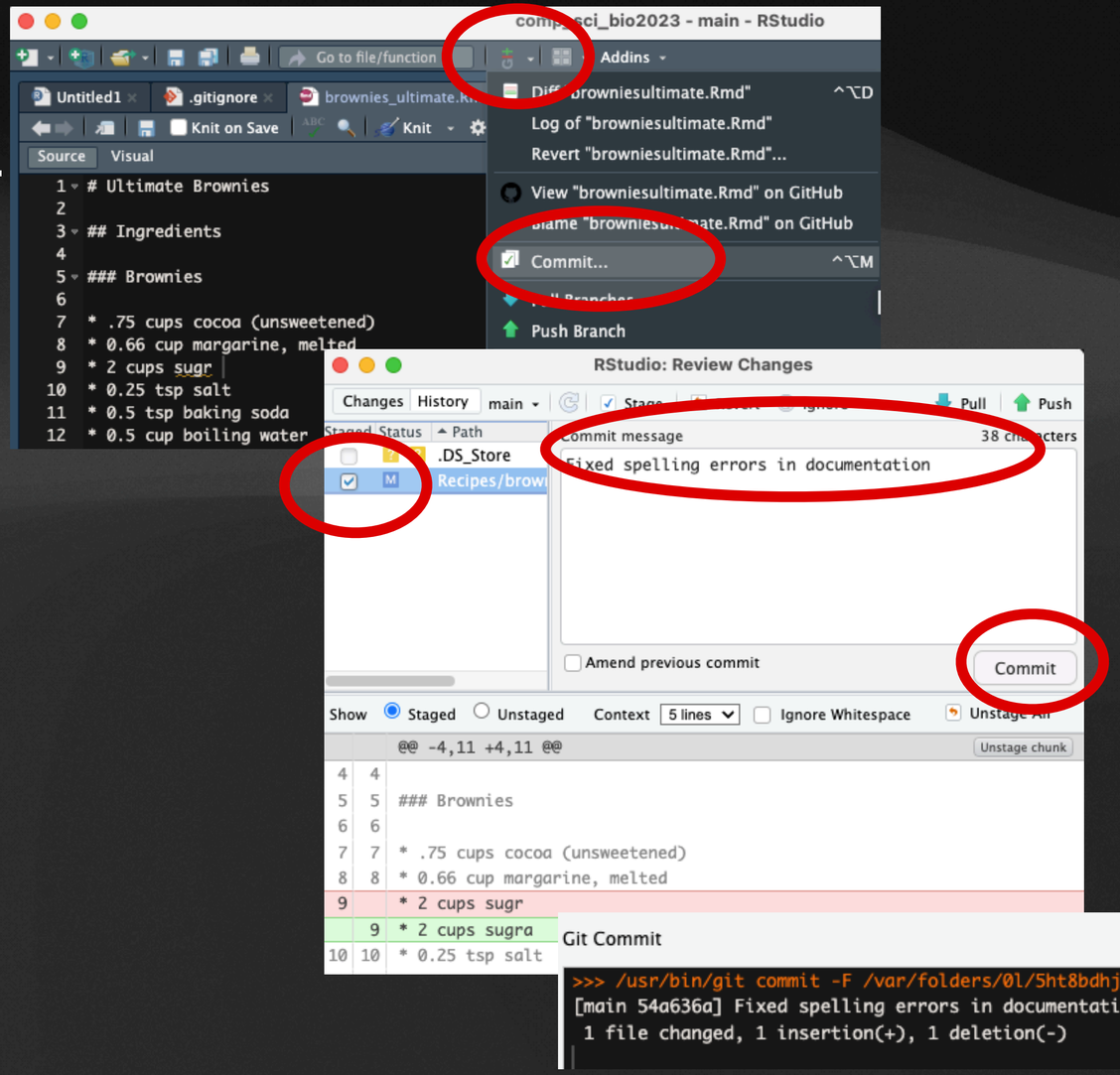
- All the changes you make will be pushed to your (public) repo.
- There are likely files you don’t want to make public (don’t want to track via version control)
- You can tell Git to ignore these files or folders by adding them to “.gitignore”
- Click .gitignore to open it
- Add some files to ignore. Good ones include .Rhistory and .xlsx



Practice: collaborative version control

Commits

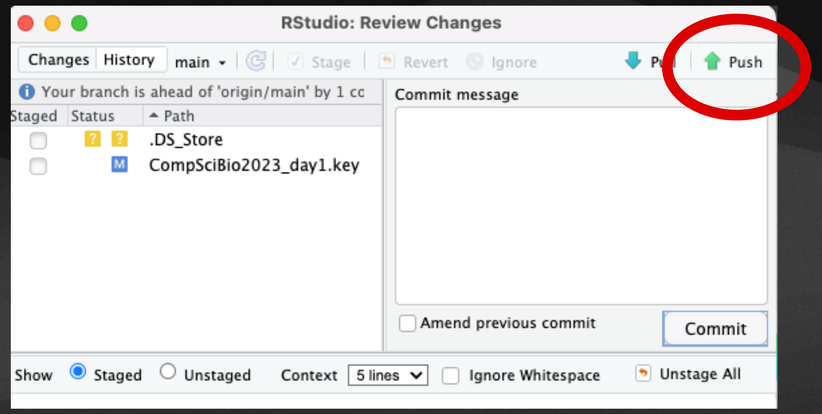
- In Rstudio, click on the “Git” drop down menu.
- Select “Commit” and a new window will open up.
- Select the file you changed and make sure to “stage” it by clicking the check box.
- Enter a comment into the Commit Message pane and then click “Commit”
- A new window will open with the results of your commit.
- Best practices: commit often, document well.

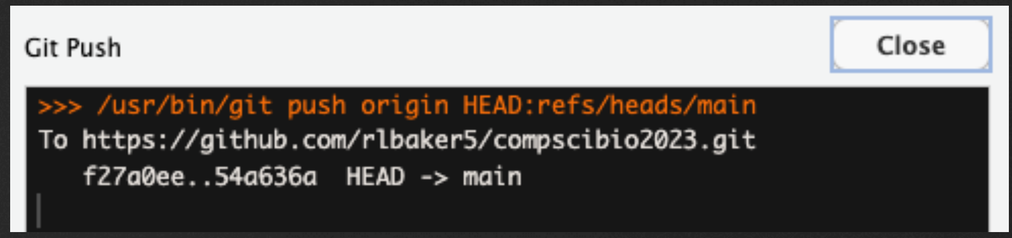


Practice: collaborative version control

Pushing to your GitHub repo

- Close the “Git Commit” Window.
- Notice that in the “Review Changes” Window your edited file is no longer listed (in my case I made changes to this keynote presentation and so it is listed)
- Click the “Push” button to push your commits to your GitHub repo.
- A new window pops up with the results of your Push.
- Best practices: push occasionally (when you are done with your current work, or at the end of the day)

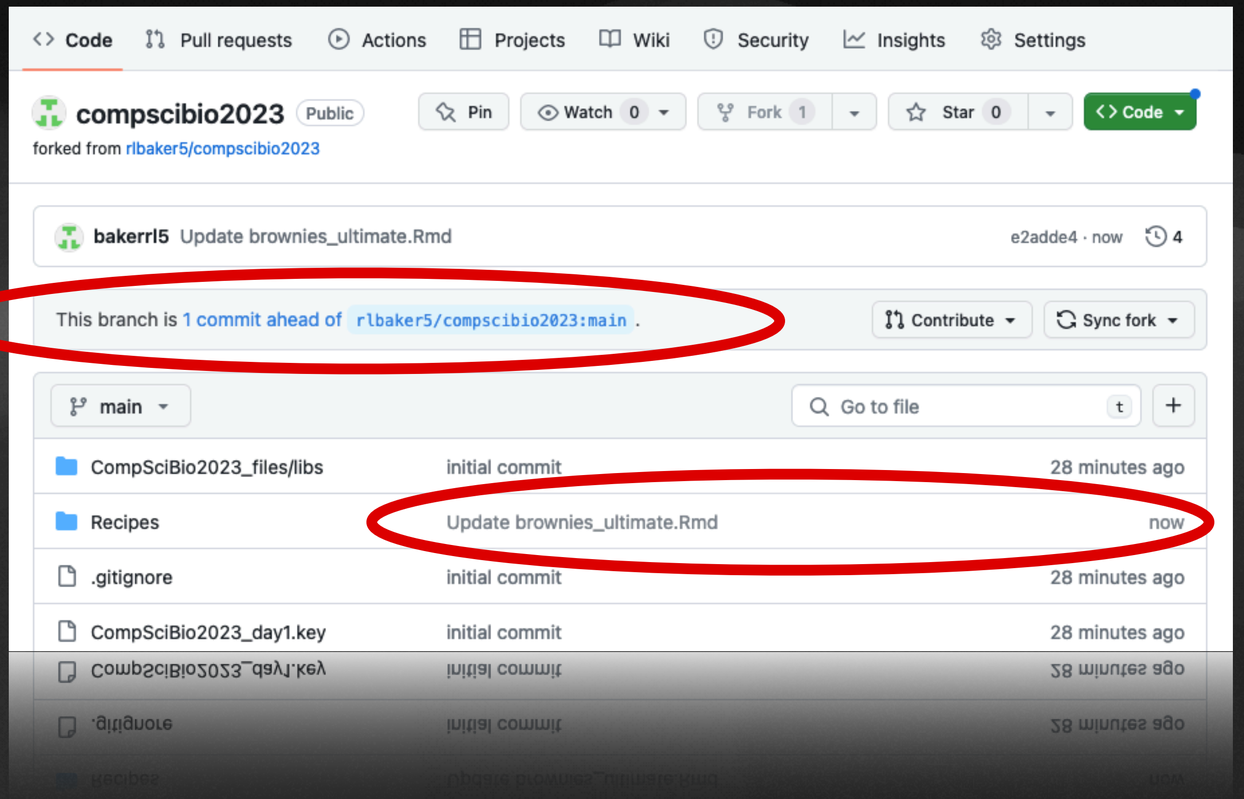




Practice: collaborative version control

Inspect your GitHub Repo

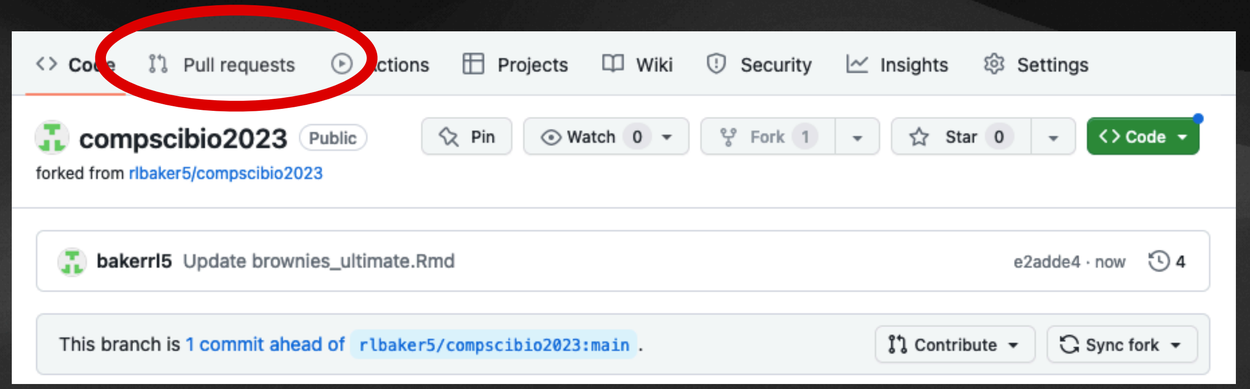
- Navigate to your GitHub account and the repo you are working on
- URL should look something like: GitHub.com/username/reponame
- You should be able to see your time-stamped commit with your comment
- You can also see that your “branch” is ahead of the main branch that it was forked from.
- This is a great time to identify any mistakes, test out code, see if web pages and documentation render correctly, etc.

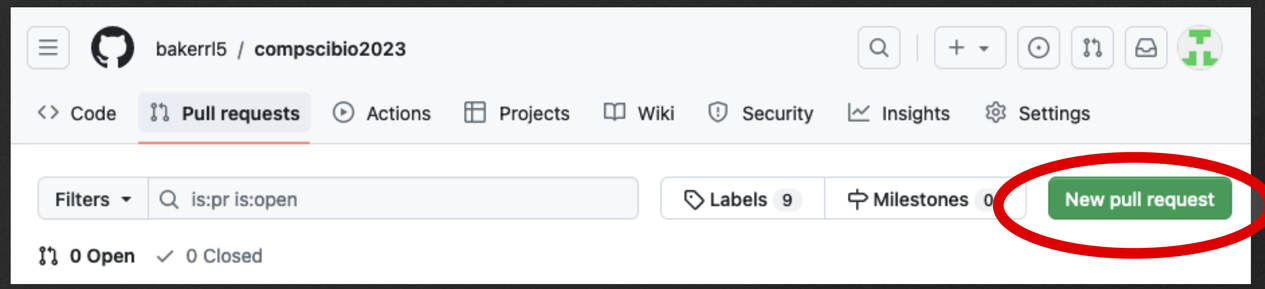


Practice: collaborative version control

Initiate a Pull Request

- Now it’s time to (ask to) add your changes to the main repository (perhaps run by a lab manager?)
- Click on “Pull Requests”, which will take you to a new page.
- Click on “New Pull Request” to open yet another page.
- It’s worth inspecting this page closely to make sure you’re dong what you meant to do….

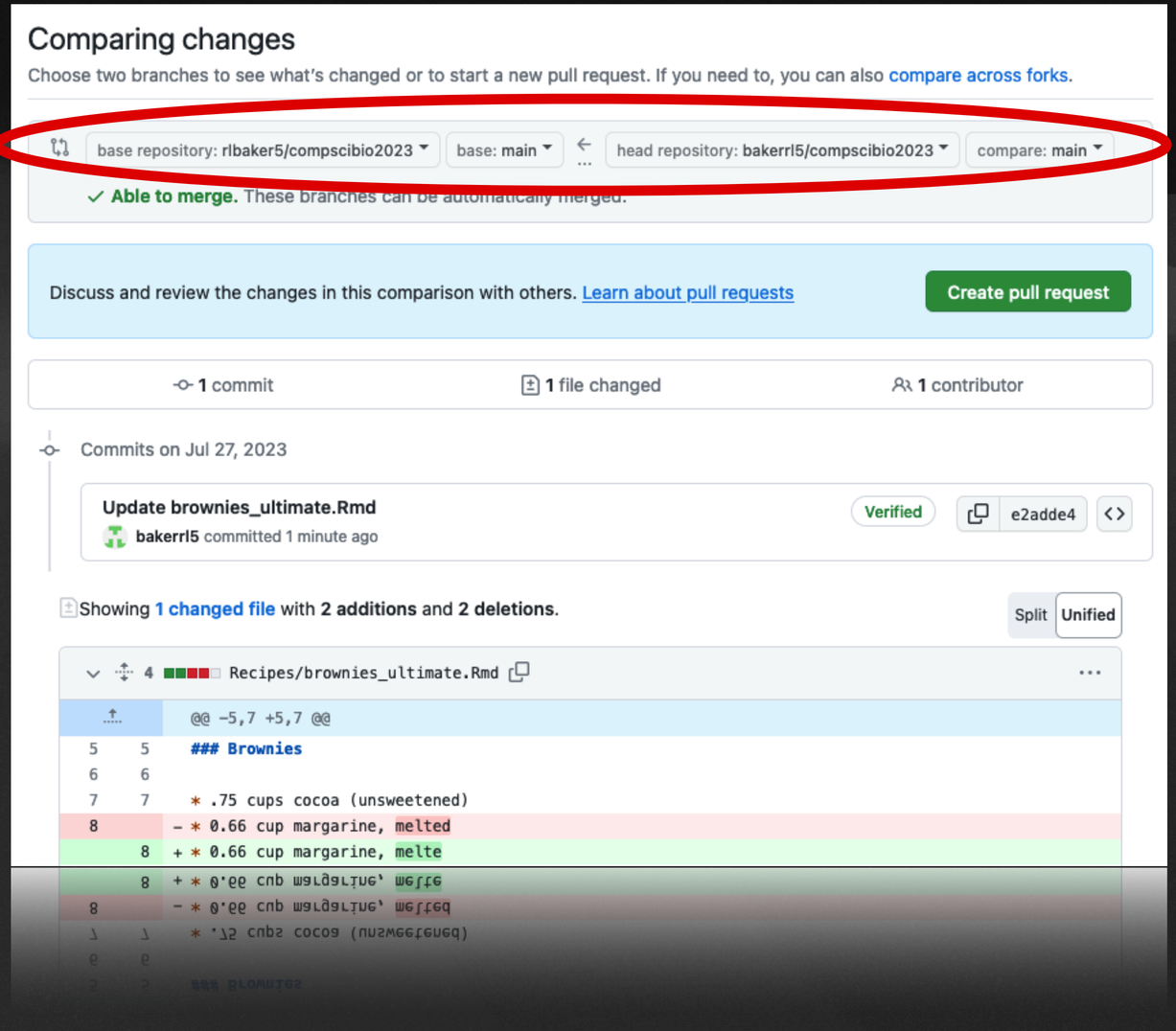




Practice: collaborative version control

Initiate a Pull Request

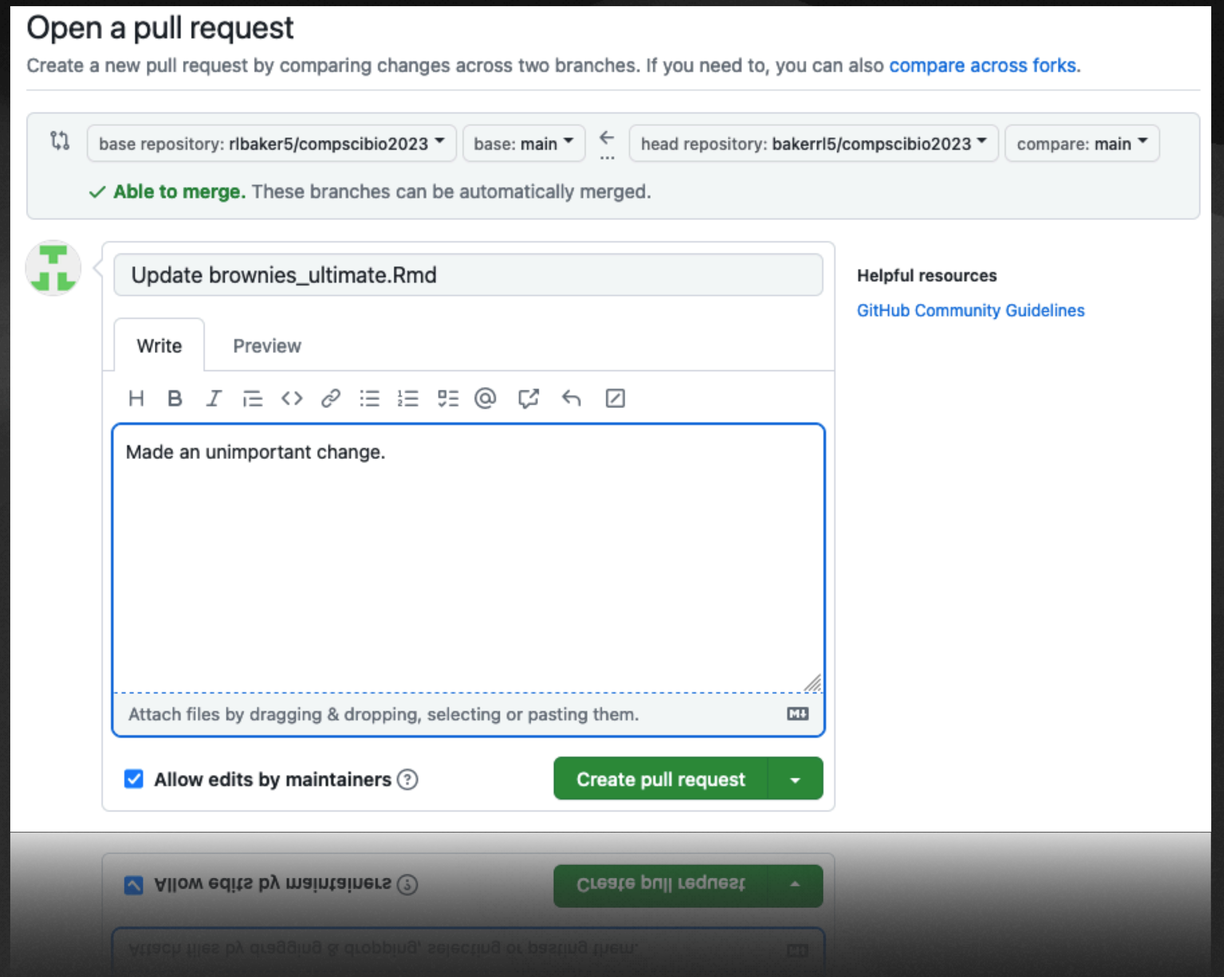
- Check to make sure you are moving files FROM the right place and TO the right place
- Hopefully GitHub tells you that the files can be merged automatically… (but if not, don't worry)
- Check that the files you wanted to change are the ones being changed
- You can even inspect the specific changes (again)
- When you're happy, click "Create pull request"



Practice: collaborative version control

Initiate a Pull Request

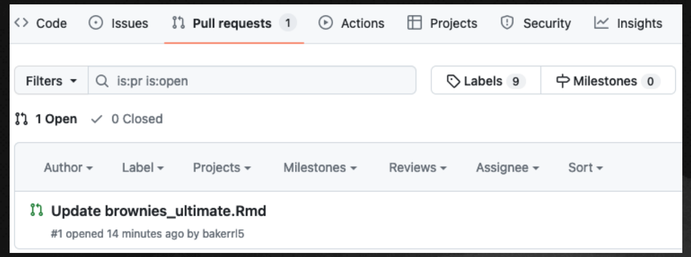
- GitHub may have tried to give your Pull Request a title, which you can edit.
- You can also add some helpful comments so that whoever is reviewing your Pull Request knows what you did (or tried to do)
- Once you are happy, click “Create Pull Request”.
- Whoever controls the main repository will be notified that you want to merge in changes.
- You can’t delete the Pull Request! However, you can still add more comments or you can close it.
- Best practices: open Pull Requests infrequently and only when your are done with whatever task you were working on (or if you need to hot-fix a bug!).
- Best practices: Pull Requests can also be a great way of asking for a code review, even if you’re not quite ready to merge into main.

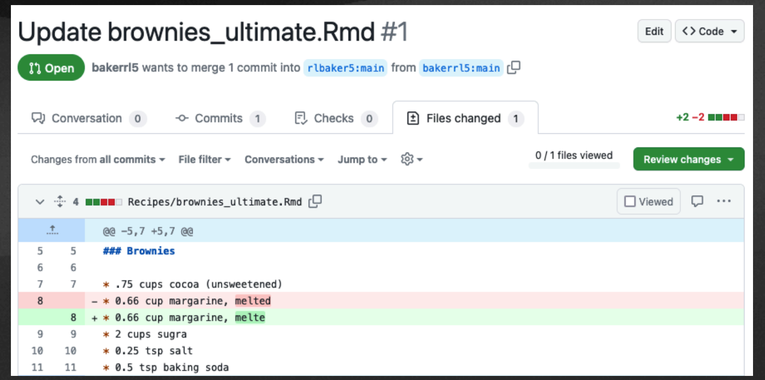


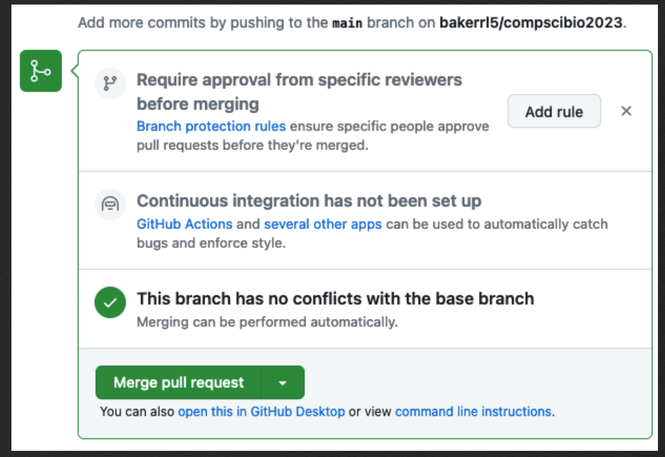
Practice: collaborative version control

What happens to your Pull Request?

- Whoever has access to the repo you requested your changes be pulled into gets notified that there is an open Pull Request.
- They can examine your PR, look at specific changes you made, and can comment on it to request clarification or approve the changes.
- They can ignore it, close it, or merge it it in (at which point it automatically closes)



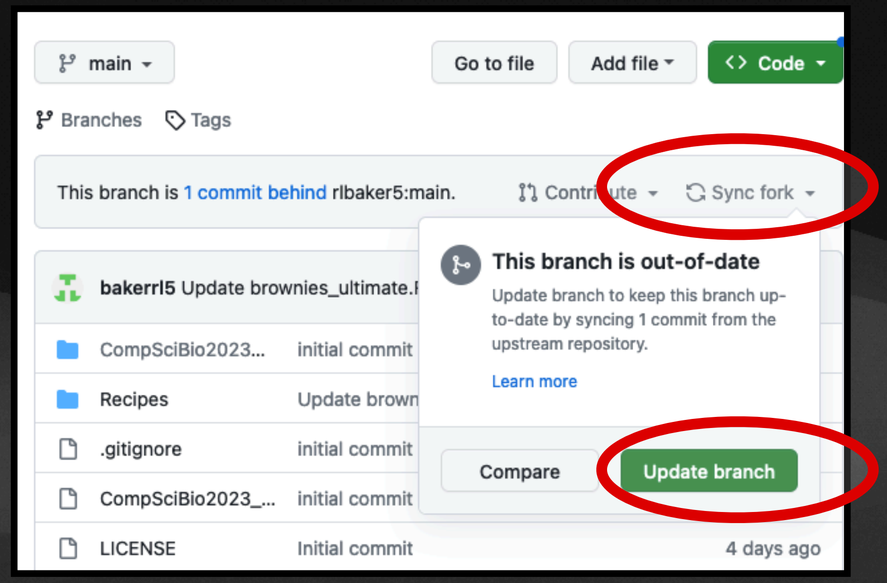


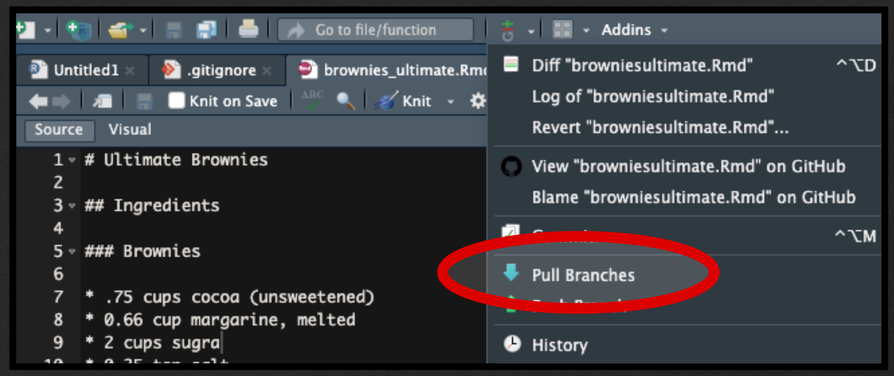


Practice: collaborative version control

What Next?

- If your PR gets merged, there is a little maintenance you need to do to on your end:
- Go to your fork of the repo on GitHub and sync it to the main repo (notice the “merge” counts as a commit so your forked repo is already behind the main repo by 1 commit!)
- In Rstudio, go to the Git tab and select “Pull” to pull down any changes and update your local copy.
- Any time you start to work on a project you should sync it and then pull it. Don’t just open Rstudio and start working!

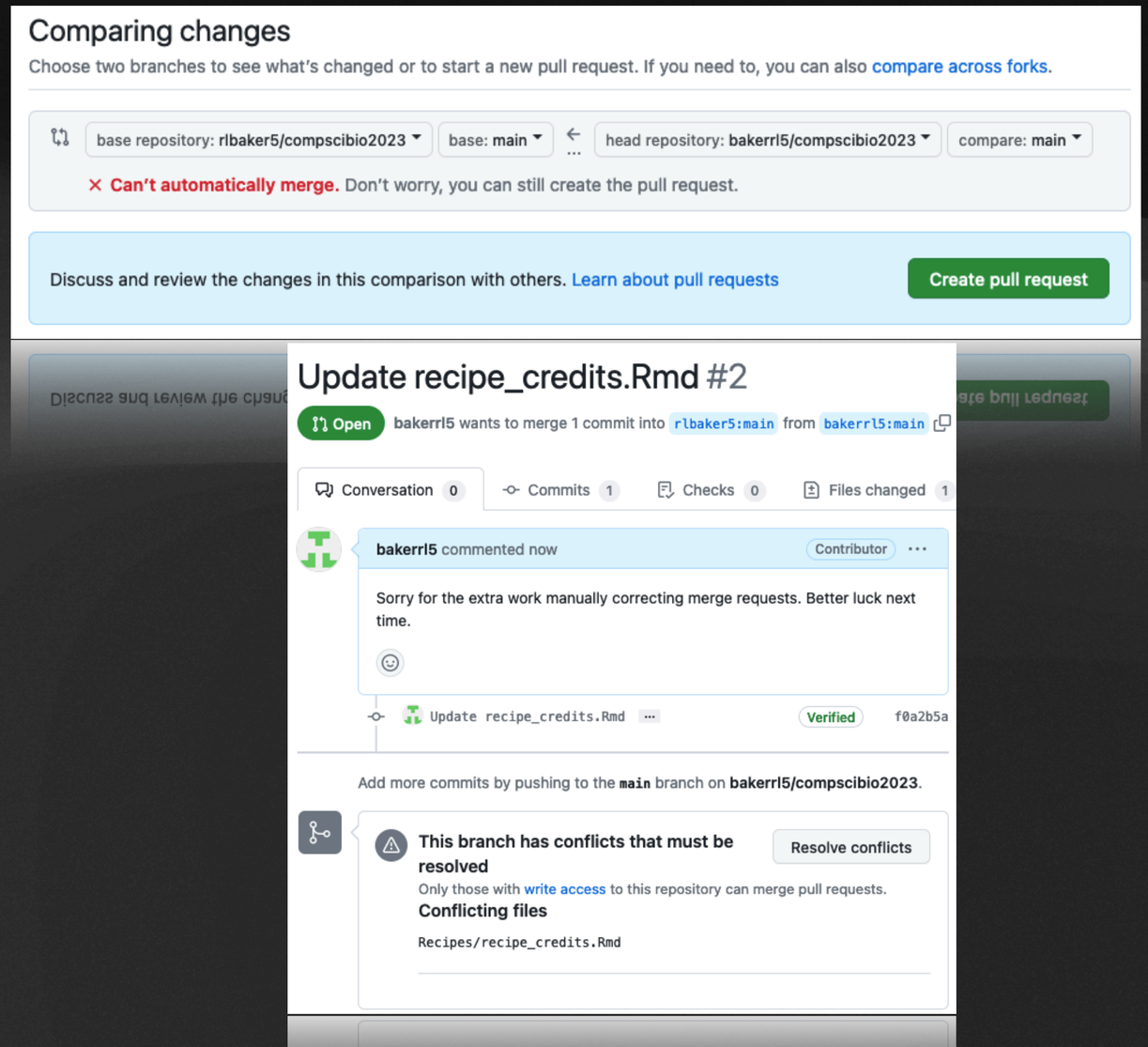




Practice: collaborative version control

Merge Conflicts: here be dragons!

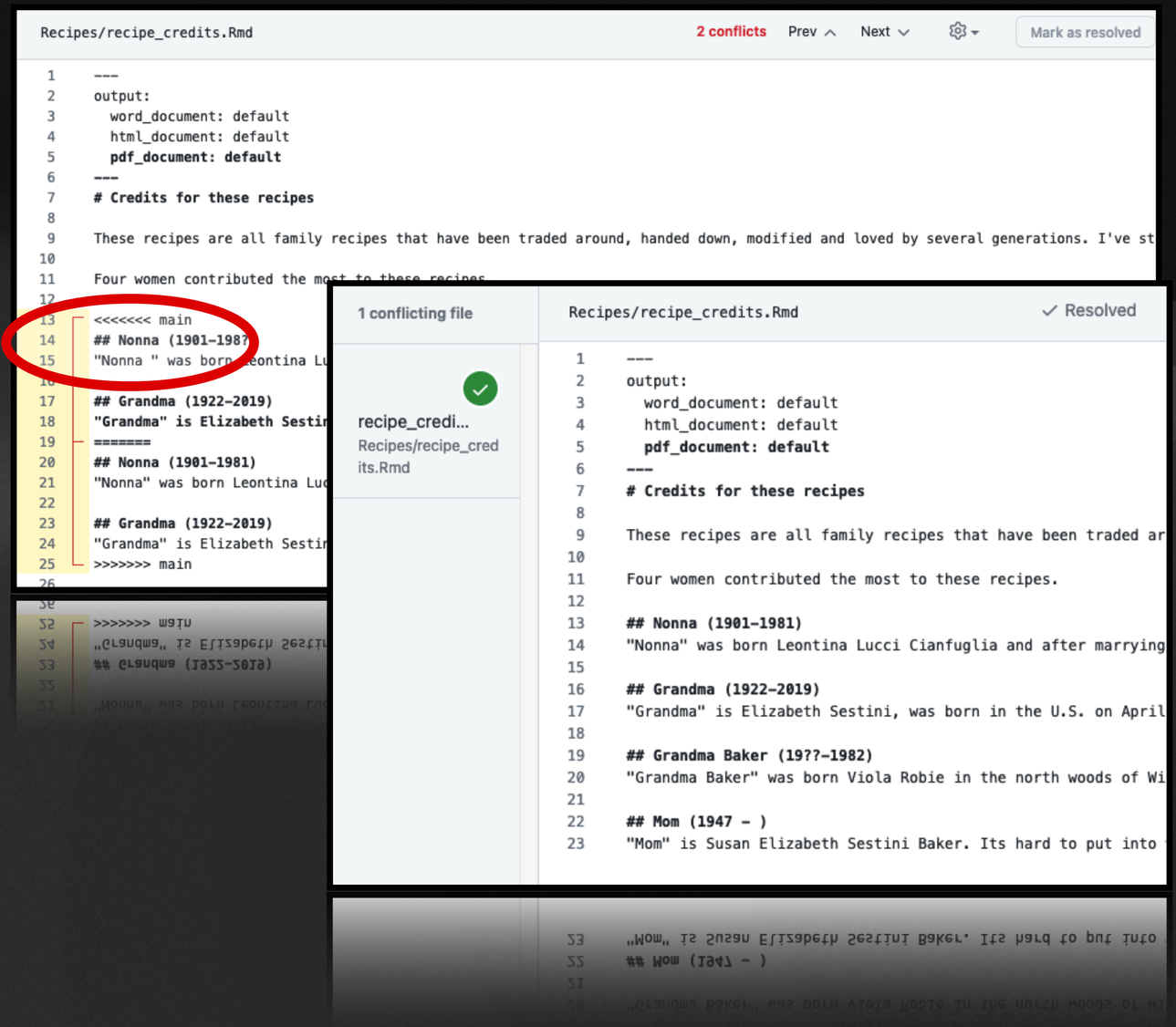
- If multiple people are each working on different files, Git/GitHub can automatically merge the changes
- If multiple people are working on different lines within the same file, Git/GitHub can automatically handle the merge
- Merge Conflict: multiple people work on the same line in the same file generates a Merge Conflict that must be manually resolved.



Practice: collaborative version control

Resolving Merge Conflicts

- Best Case Scenario: merge conflicts are simple and can be resolved via GitHub web page.
- The merge conflicts are identified; delete the rows you want to remove and delete the indicators.
- Click “Mark Resolved”; Commit the merge.
- Merge the Pull Request. Done!



Practice: collaborative version control

Complicated merge conflicts:

- The best way to fix them is avoid them: communicate with your collaborators!
- Require editing off-line, but the concept is the same: delete what you don’t want and keep what you do
- Additional issues can arise if you aren’t careful
- Its hard to screw up so badly it’s not recoverable
- Most mistakes can be fixed after some googling
- Complex problems are beyond the scope of this module

Course Outline

I. Introductions

II. Concepts of version control and collaborative coding

III. Sign up for a GitHub account

IV. Walk through and help installing/upgrading software

V. Rstudio/GitHub integration

VI. Version Control and Collaborative Coding Workflows

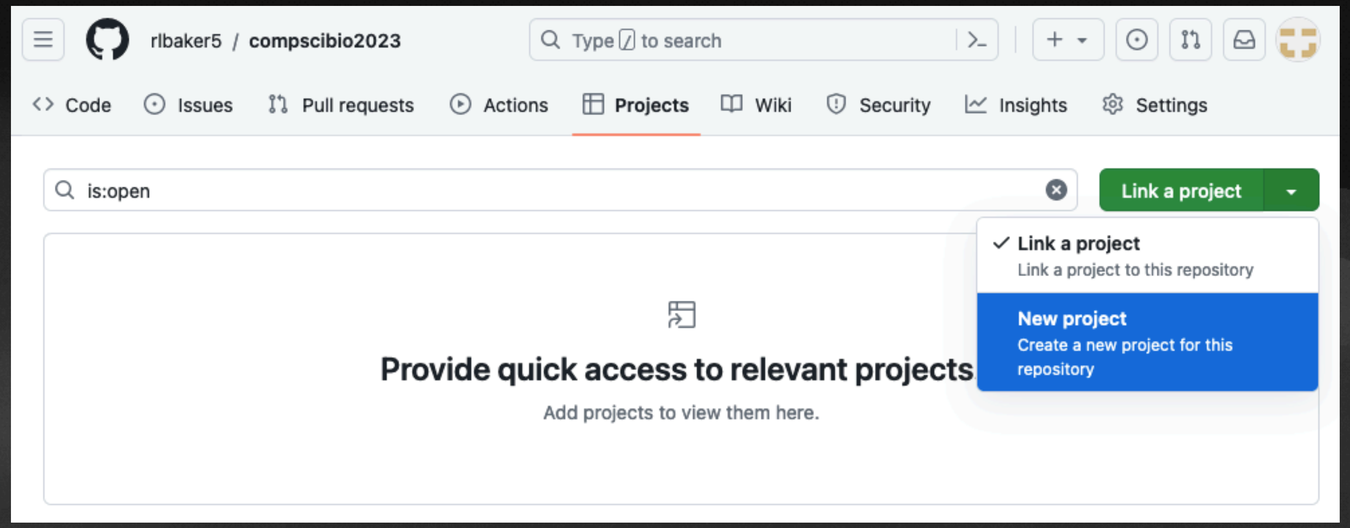
VII. **Practice: collaborative version control**

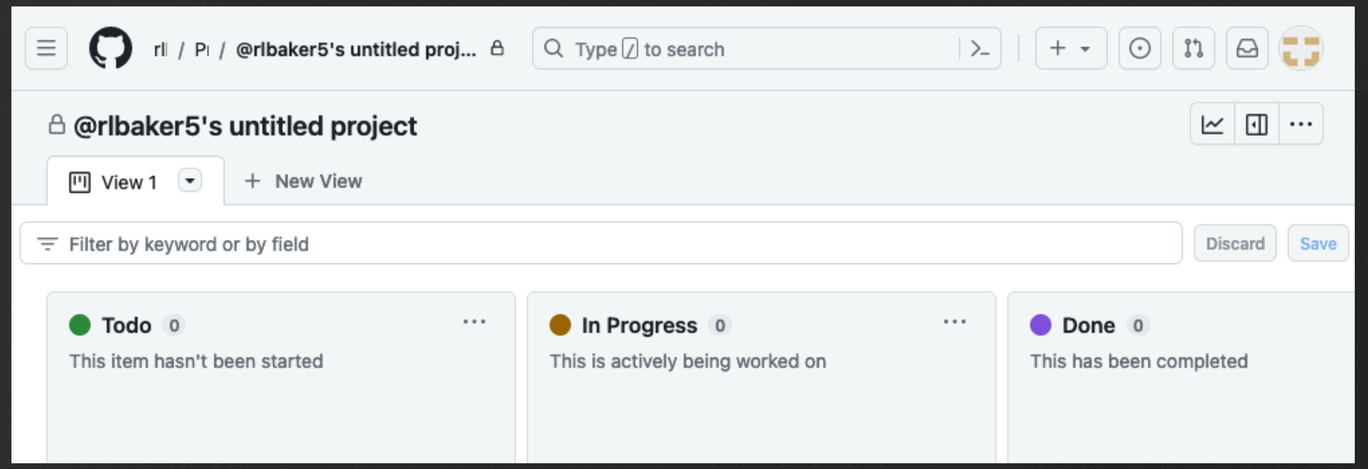
VIII. **Bonus: project management boards, issues, milestones**

GitHub project management

Projects

- Projects are a nice way to keep track of your repo (or even many repos at once): what needs to be done, what is in progress, what has been done.
- I like “Boards” view, but super customizable
- You can customize the columns and add “issues” to each.

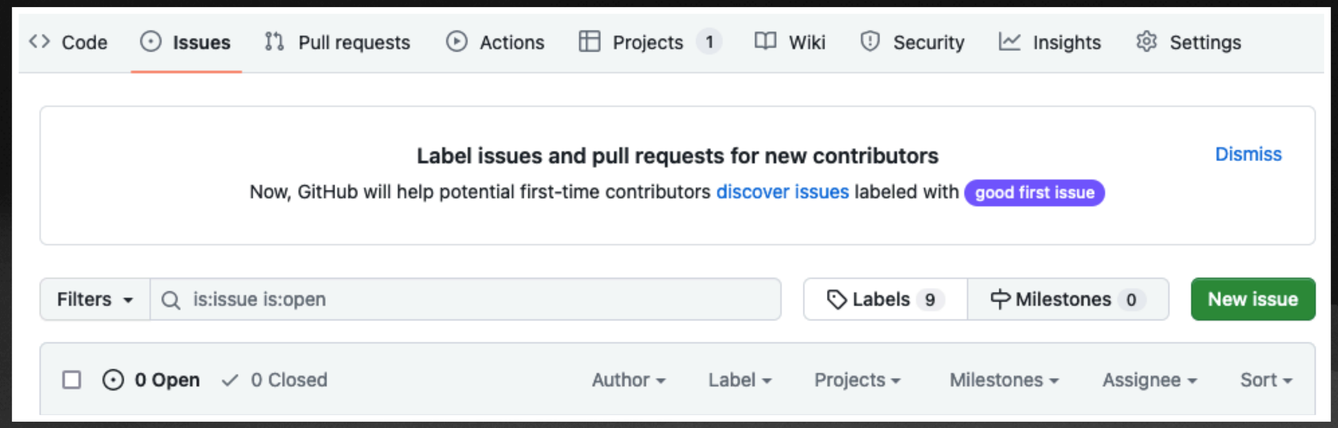


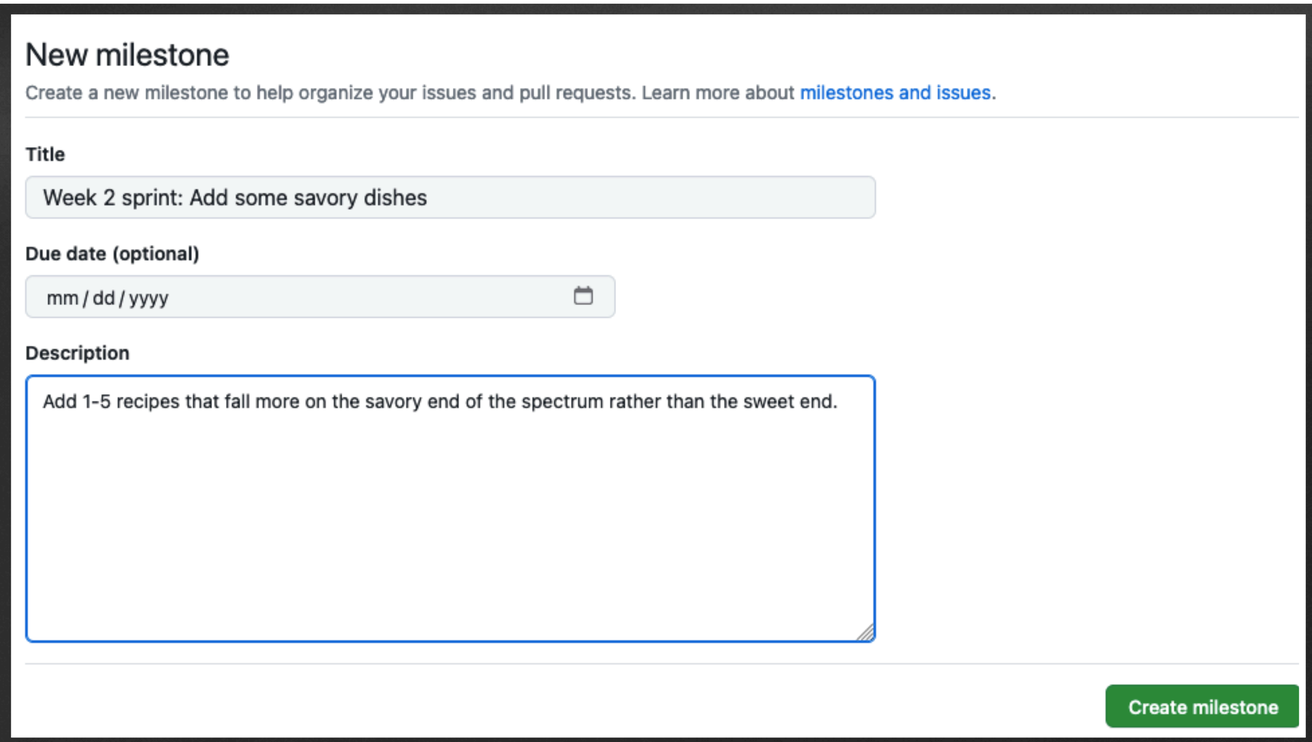


GitHub project management

Milestones

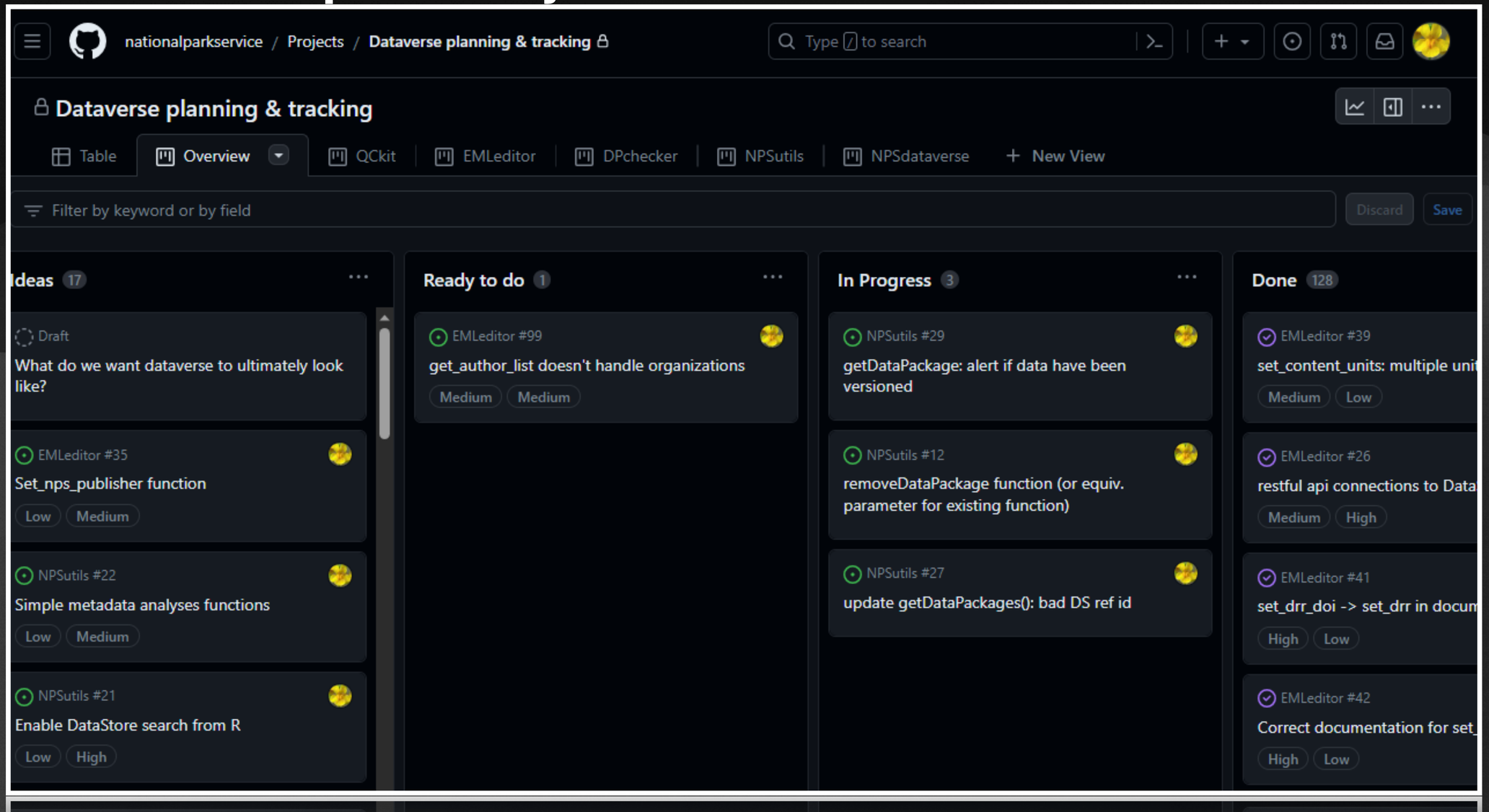
- Milestones are a way of grouping issues
- You can break problems down into multiple issues, then assign them to a milestone
- Milestones help you keep track of progress towards a larger objective





GitHub project management

Advanced examples: Project Boards



GitHub project management

Advanced examples: Github Pages

Credits and additional info

- Happy Git with R: https://happygitwithr.com/ (you may notice we followed this very closely!)
- Posit Version control: https://support.posit.co/hc/en-us/articles/200532077
- Configuring R/GitHub: https://gist.github.com/z3tt/3dab3535007acf10839164976640942 1
- Data Carpentries Workshops: https://datacarpentry.org/
- Various amounts of googling!