Lecture 2: R Markdown, Version Control with Git(Hub), and Other Productivity Tools

James Sears*
AFRE 891/991 FS 25
Michigan State University

*Parts of these slides are adapted from **"Advanced Data Analytics"** by Nick Hagerty and **"Data Science for Economists"** by Grant McDermott.

Table of Contents

Prologue

Prologue

Before we dive in, let's double check that we all have

☑ Installed **R**.

☑ Installed **RStudio**.

☑ Signed up for an account on **Github**

☑ Installed **Git** and **Github Desktop**

☑ Log into your Github account on Github Desktop

R Markdown

R Markdown

Before we dive into version control, let's chat about **R Markdown**.

R Markdown is a document type that allows for integration of R code and output into a Markdown document.

Resources:

- Website: **rmarkdown.rstudio.com**
- **R Markdown Cheatsheet**
- Book: **R Markdown: The Definitive Guide** (Yihui Xie, JJ Allaire, and Garrett Grolemund)

R Markdown

Before we dive into version control, let's chat about **R Markdown**.

R Markdown is a document type that allows for integration of R code and output into a Markdown document.

Other points:

- We'll be completing assignments using R Markdown.
- FWIW, my lecture slides and notes are all written in R Markdown too. (E.g. This slide deck is built using the **xaringan** package with the metropolis theme.)

R Markdown: Getting Started

☑ Installed **R**.

☑ Installed **RStudio**.

☐ Add the `rmarkdown` package

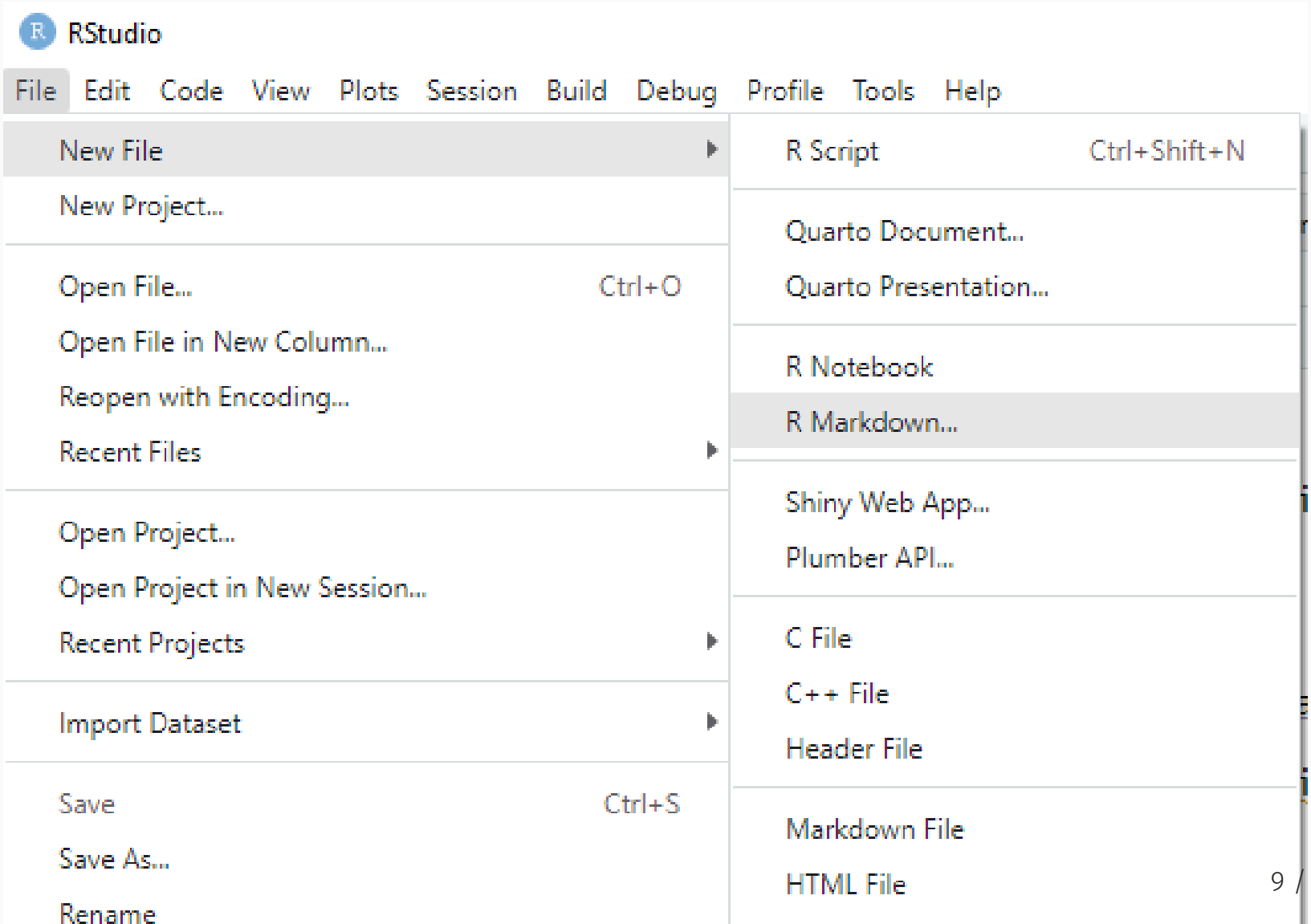
```
install.packages("rmarkdown")
```

☐ Install LaTeX

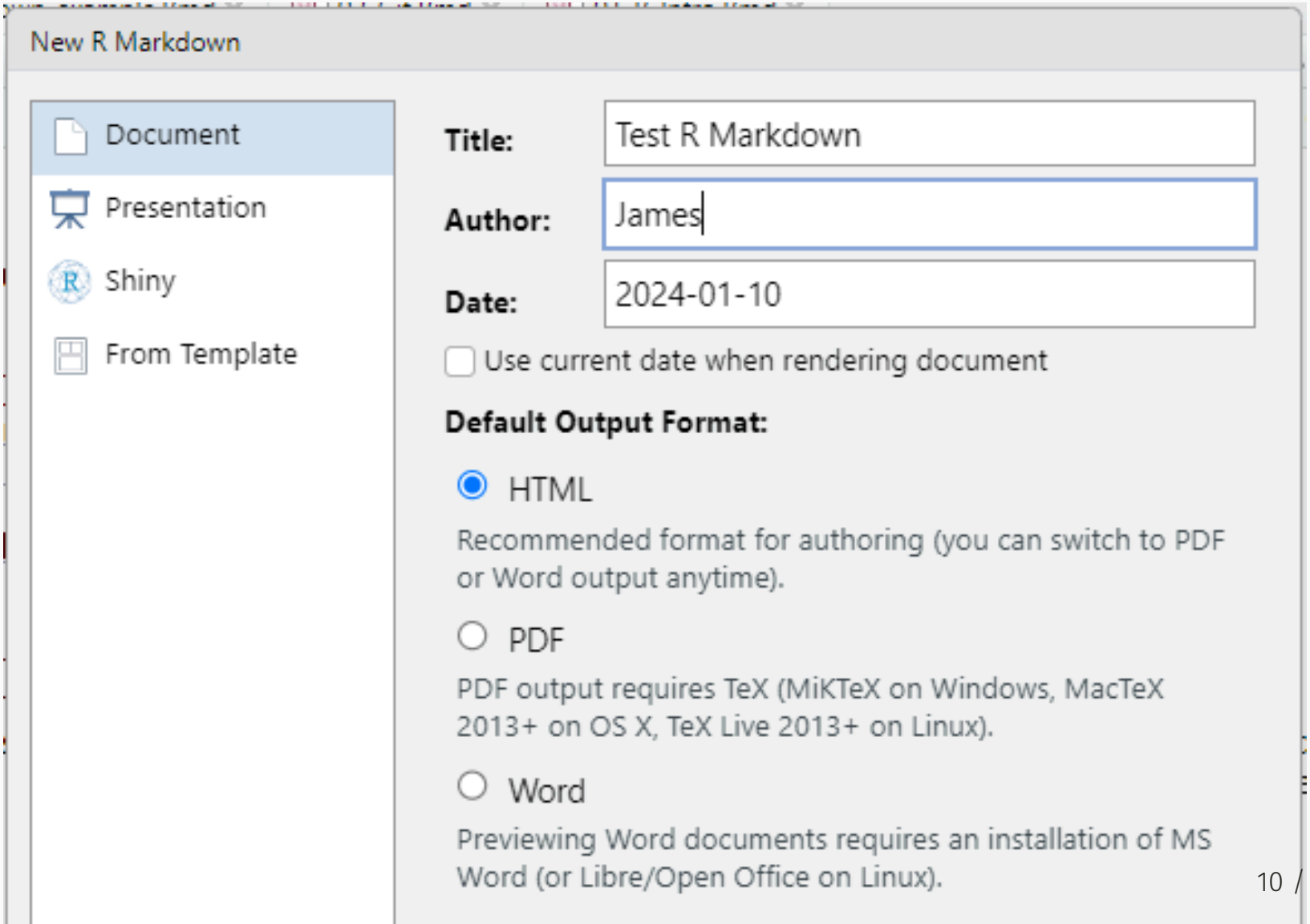
- If just for this, can use **TinyTex**

```
# Install only if you don't have LaTeX already
install.packages("tinytex")
tinytex::install_tinytex()
```

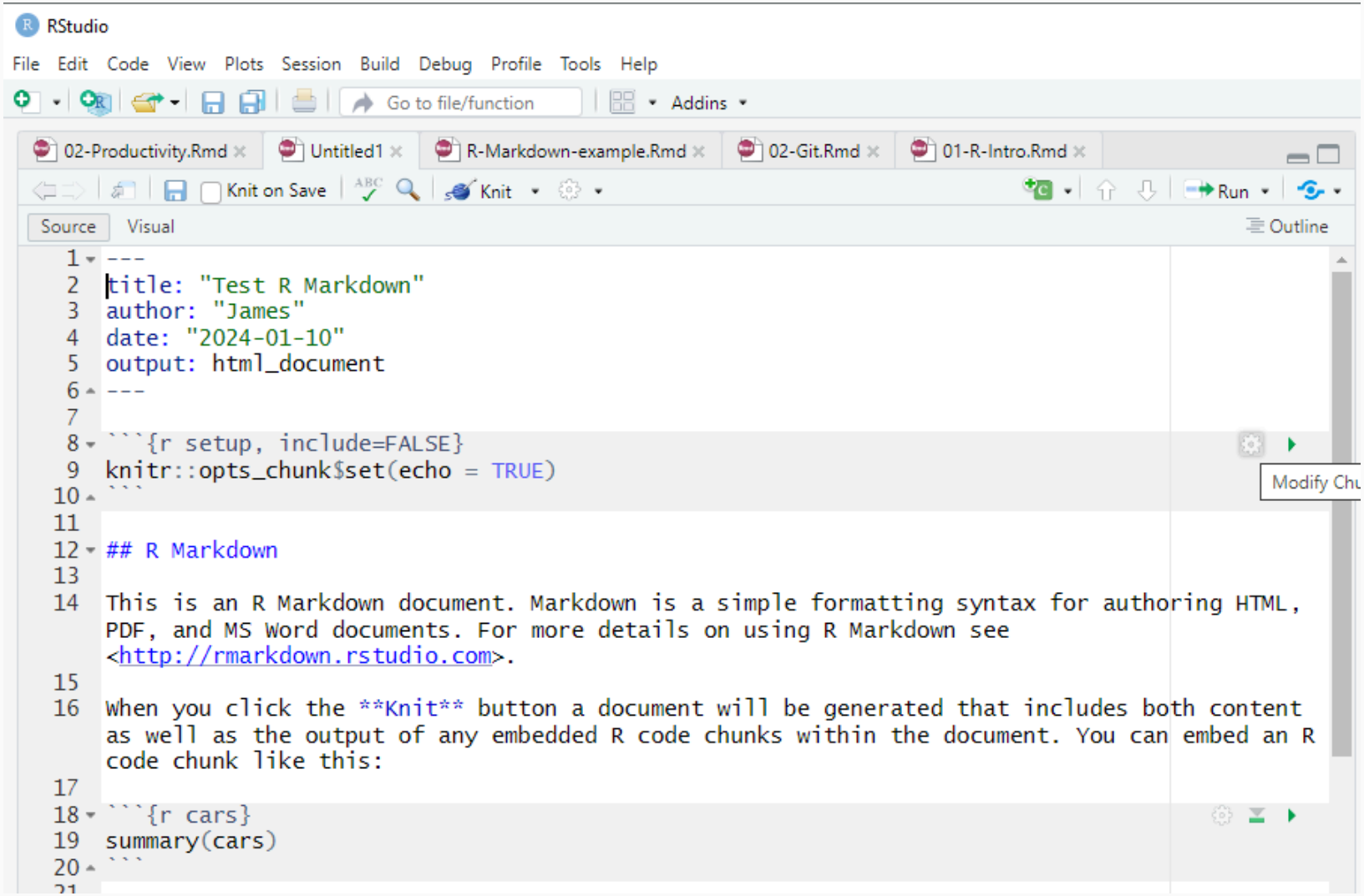
R Markdown: Creating a New .Rmd File



R Markdown: Creating a New .Rmd File



R Markdown: Creating a New .Rmd File



R Markdown Components

R Markdown combines

1. **Markdown:** lightweight markup language
2. **LaTeX:** typesetting for math
3. **R:** include code and generate output

Let's do some practice: **open a new .Rmd file** and try adding content as we go

Markdown

Markdown allows for formatting text in a lightweight way

I highly recommend the handy **Markdown Guide** for more details

Markdown: Heading

Headings emphasize text and add chunks to your script

Largest headingwith one leading # (slide title above)

Second Largest (##)

Third Largest (###)

Getting Smaller... (####)

Normal Text for comparison

Markdown: Text Format

Bold text with **your text**

Italicize with *single asterisks*

Add `code text` with grave accents (the back tick symbol)

- `
- The other output of the tilde key `~` on keyboard

End a line with two spaces to start a new paragraph

- or leave a line space between sentences

Can also start a new line with backslash (\)

Markdown: Text Format

Add superscripts^{2} with ^carets^

Add ~~strikethroughs~~ with ~~double tildes~~

Add a line break (horizontal rule)

---

with ***

Markdown: Text Format

Draw **tables** using | and -

```
| Col A | Col B | Col C|
|-------|-------|------|
| This  | is    | a    |
| Table |       | wow  |
```

Markdown: Text Format

Draw **tables** using | and -

Col A	Col B	Col C
This	is	a
Table		wow

Markdown: Text Format

You can adjust the **alignment** of table text by adding `:`'s in the second row:

- `:----` for left-aligned
- `:---:` for center-aligned
- `----:` for left-aligned

```
| Column A | Column B | Column C|
|:---    |:---:|            ---:|
| Col A | is  | left-aligned   |
| Col B | is  | center-aligned |
| Col C | is  | right-aligned  |
```

Markdown: Text Format

You can adjust the **alignment** of table text by adding `:`'s in the second row:

- `:----` for left-aligned
- `:---:` for center-aligned
- `----:` for left-aligned

Column A	Column B	Column C
Col A	is	left-aligned
Col B	is	center-aligned
Col C	is	right-aligned

Markdown: Lists

Add an **ordered list** with **1.**

1. First Item
2. Second Item
3. No need to change the number - keep using 1. It will automatically update.

Add an **unordered list** with *** or -**

- A thing
- Another related thing
    - Indent to nest
        1. Can mix ordered and unordered

Markdown: Inputs

Add a **link** with []()

- [text label](URL)
- Add direct link with <link> **https://www.markdownguide.org**

Add an image with

- ![alt text](URL)

practice by adding `images/smile.png`:

Markdown: LaTeX

Another advantage of Markdown is that it integrates LaTeX functionality for typesetting math.

Add an **inline equation** with TeX

$$Var(X) = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n} \qquad Y_{it} = \beta_0 + \beta_1 X_{it} + \epsilon_{it}$$

Add multiple rows of LaTeX with

$$

LaTeX lines here

$$

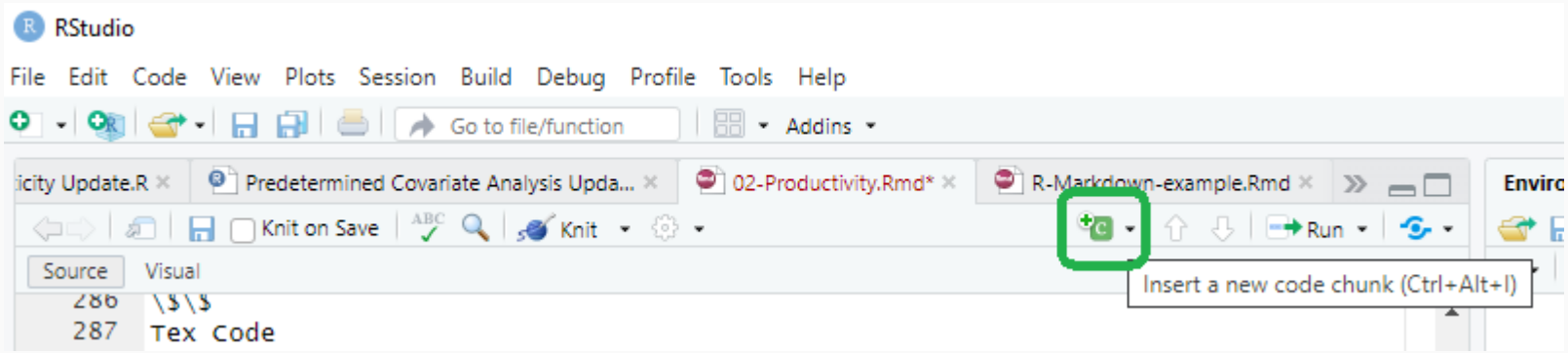
Use the **standard LaTeX commands** for symbols/characters

Rmd: R Code

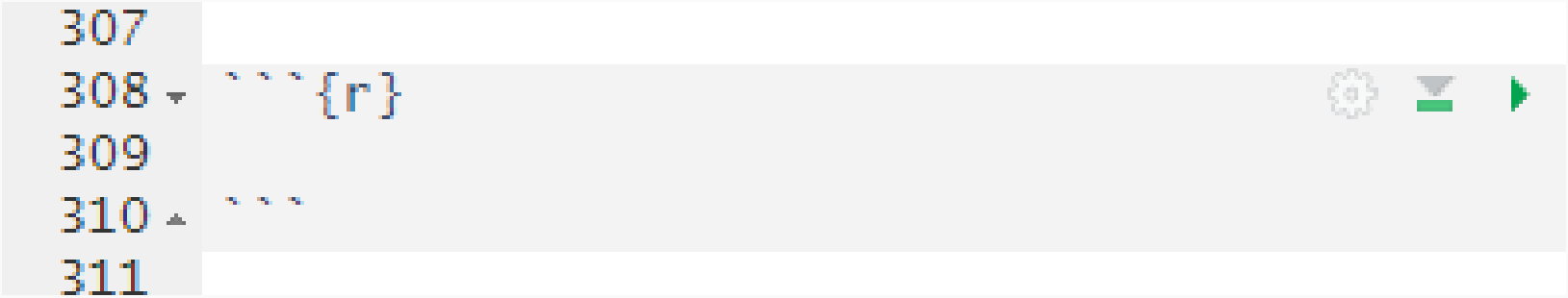
R code is primarily executed with **code chunks**

Add a chunk with

- `Cmd + Option + I (Ctrl + Alt + I on PC)`
- The `Insert` button in the UI
- Manually type



Rmd: Code Chunks



Code chunks allow us to add as many lines of code as we want

- Output will appear underneath after executing the full chunk
- Can customize whether it runs, how output is displayed
- Can run manually
  - Line by line with `Cmd/Ctrl + Enter`
  - Entire chunk with `Run Entire Chunk` button

Code Chunk Options

You can **add chunk options** in brackets after `r` and separated by commas.

Some commonly-used options include:

- **Chunk label** (`ex_chunk`)
- `include = FALSE` will run the chunk but hide it from the final document
- `eval = FALSE` will display code without evaluating it
- `results = 'hide'` runs code but hides output from the final document

````
52
53 ```{r sum, echo = FALSE, warning = FALSE}
54 2+2
55
56 ```
````

Code Chunk Options

You can also **change the color** of code chunks and/or output in the rendered document through code chunk options[1]

- `class.source` to change the **code chunk**
- `output.source` to change the **output**

default

```
1 + 2 * 3 ^ 4
```

```
## [1] 163
```

`bg-primary`

```
1 + 2 * 3 ^ 4
```

```
## [1] 163
```

[1] We'll chat more about this when we get to web scraping, but what's actually happening here is that we're harnessing some built-in CSS classes to change the backgrounds. This also means that, using CSS, you can define custom classes and format things however you'd like.

Code Chunk Options

You can also **change the color** of code chunks and/or output in the rendered document through code chunk options[1]

- `class.source` to change the **code chunk**
- `output.source` to change the **output**

`bg-success`

```
1 + 2 * 3 ^ 4
```

```
## [1] 163
```

`bg-info`

```
1 + 2 * 3 ^ 4
```

```
## [1] 163
```

[1] We'll chat more about this when we get to web scraping, but what's actually happening here is that we're harnessing some built-in CSS classes to change the backgrounds. This also means that, using CSS, you can define custom classes and format things however you'd like.

Code Chunk Options

You can also **change the color** of code chunks and/or output in the rendered document through code chunk options[1]

- `class.source` to change the **code chunk**
- `output.source` to change the **output**

`bg-warning`

```
1 + 2 * 3 ^ 4
```

```
## [1] 163
```

`bg-danger`

```
1 + 2 * 3 ^ 4
```

```
## [1] 163
```

[1] We'll chat more about this when we get to web scraping, but what's actually happening here is that we're harnessing some built-in CSS classes to change the backgrounds. This also means that, using CSS, you can define custom classes and format things however you'd like.

Rmd: Inline Code

You can call R objects from earlier chunks **inline** with

`` `r` ``

```
four = 2+2
```

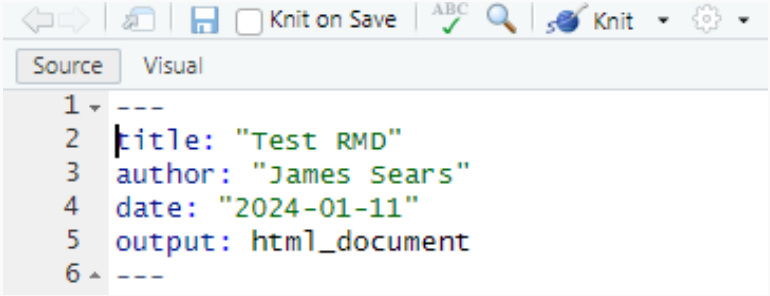
This can output in line with text: 2 + 2 = 4

R Markdown File Organization

1. Header

RStudio automatically builds the R Markdown file from a template, which begins with a **header**

- Title
- Author
- Date
- Output Format
  - Main options[1]: HTML (`html_document`), PDF (`pdf_document`), LaTeX (`latex_document`), or Word (`word_document`)[2]



1: See **CH 3 of "R Markdown: The Definitive Guide" for more on how to customize output formats**

2: For **better formatted Word output** with greater customisability, use the **officedown** package's `rdox_document` format.

2. R Setup

By default, RStudio adds a **setup** code chunk next.

````
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
````

- Can set global options
- Useful as your preamble
- For **R Notebooks**, this will automatically be run and is the only place where you can change your working directory

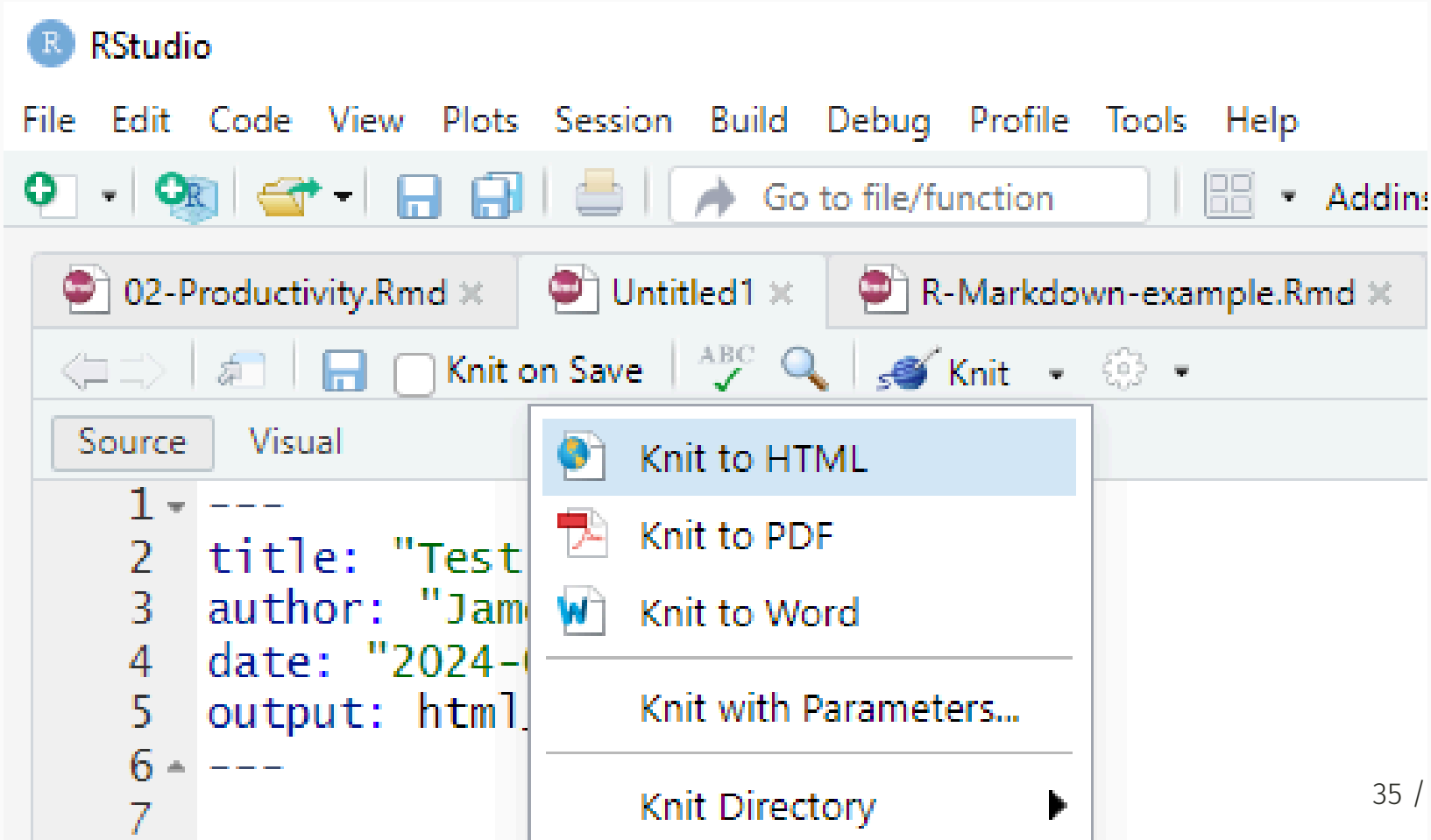
3. Contents

From here on you can build the report/notebook as needed for the task.

- Add any writing and outside graphics or **bibTeX citations**
- Add code chunks to carry out desired analysis
- Employ sections and formatting to structure the document as desired

Compiling/Knitting

When you are ready to compile your final document, use the `Knit` button or `Ctrl/Cmd + Shift + K`



R Markdown: Knit to Compile Output

Markdown Practice!

Markdown Practice

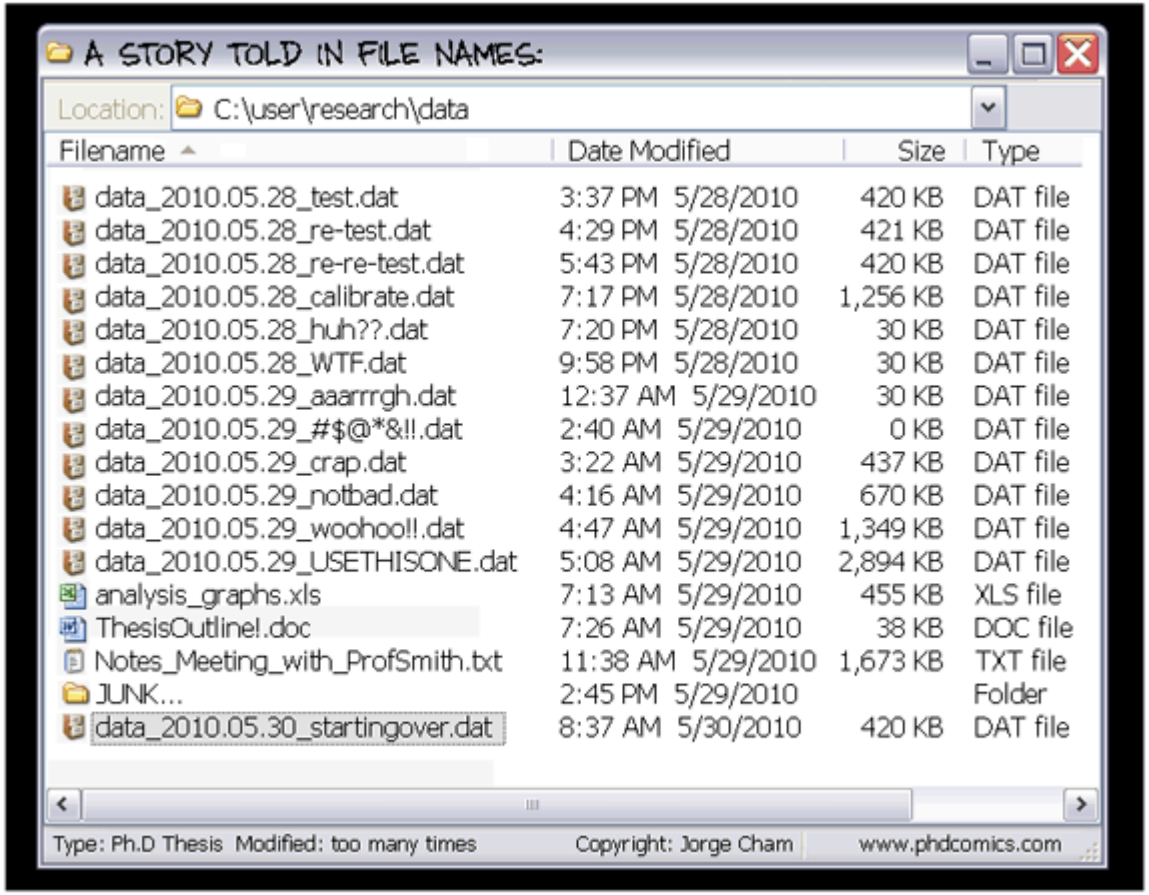
1. Create a new R Markdown file named "R-Markdown-Ex.Rmd"
2. In the setup chunk, load the **dslabs** and **tidyverse** packages
   - Use the `data()` function to read in the `divorce_margarine` dataset
3. Add a header labeled "Correlation vs. Causation" and a text explanation below for why we often want to differentiate between the two
4. Add a code chunk with the label `plot`
   - Type the following code:

```
ggplot(divorce_margarine) +
geom_point(aes(x = margarine_consumption_per_capita,
               y = divorce_rate_maine)) +
labs(title = "Relationship between Margarine Consumption and
              Divorce Rates in Maine",
 subtitle = "2000-2009",
 x = "Margarine Consumption per Capita",
y = "Divorce Rate")
```

5. Knit and save a PDF/HTML copy of the file to the "output" folder

Version Control

Why Use Version Control



Goals of Version Control

While building project folders with the above naming conventions is *fun*, a good **version control system** can solve this problem.

- Save each set of changes sequentially
- Keep track of different versions of a file
- Merge changes from multiple versions/sources

Git(Hub) Solves this Problem

Git

- **Git** is a **distributed version control system**
  - Each team member has a **local copy** of files on their computer
- Imagine if Dropbox and the "Track changes" feature in MS Word had a baby. Git would be that baby.
- In fact, it's even better than that because Git is optimised for the things that economists and data scientists spend a lot of time working on (e.g. code).
- There is a learning curve, but I promise you it's worth it.

Git(Hub) Solves this Problem

GitHub

- It's important to realise that **Git** and **GitHub** are distinct things.
- **GitHub** is an **online hosting platform** that provides an array of services built on top of the **Git** system. (Similar platforms include Bitbucket and GitLab.)
- Just like we don't *need* **Rstudio** to run **R** code, we don't *need* **GitHub** to use **Git**... but it will make our lives so much easier.

Git(Hub) for Scientific Research

From software development...

- **Git** and **GitHub**'s role in global software development is not in question.
- There's a high probability that your favorite app, program or package is built using Git-based tools. (RStudio is a case in point.)

... to scientific research

- Benefits of VC and collaboration tools aside, Git(Hub) helps to operationalise the ideals of open science and reproducibility.[2]
- Journals have increasingly strict requirements regarding reproducibility and data access. GH makes this easy (DOI integration, off-the-shelf licenses, etc.)
- I host **teaching materials** on GH. I even use it to host and maintain my **website** for free.

2: **Democratic databases: Science on GitHub (Nature)** (Perkel, 2016).

Using GitHub

There are a couple of different main ways that we could use GitHub:

1. Through **github.com** Only

- **Pros:** doesn't require any software/local repo copies
- **Cons:** much more time-intensive and not automated (the whole point of this thing!)

2. Through the command line (**GitHub CLI**)

- **Pros:** fully programmatic, requires no additional software
- **Cons:** fully programmatic!

Using GitHub

There are a couple of different main ways that we could use GitHub:

3. Integrated with RStudio

- **Pros:** fully integrates RStudio projects
- **Cons:** limited to R projects, doesn't play as nicely with GitHub Classroom

4. Through the GitHub Desktop App

- **Pros:** Intuitive GUI, sync any kinds of files/projects
- **Cons:** requires an extra piece of software, **GitHub Desktop**

We're going to focus primarily on **4**, but the back of the deck will contain slides working through **3** if you want to experiment with that too.

GitHub Desktop

Version Control with GitHub Desktop

Although GitHub integration with RStudio has lots of functionality, there are times where we want to keep track of files and projects **outside of RStudio**.

- For example, when you want version control of projects that **don't only use R** (or don't use it at all)

This is where **GitHub Desktop** comes in.

Version Control with GitHub Desktop

This next section is about learning the basic Git(Hub) commands and the recipe for successful version control with GitHub Desktop.

I also want to bookmark a general point that we'll revisit many times during this course:

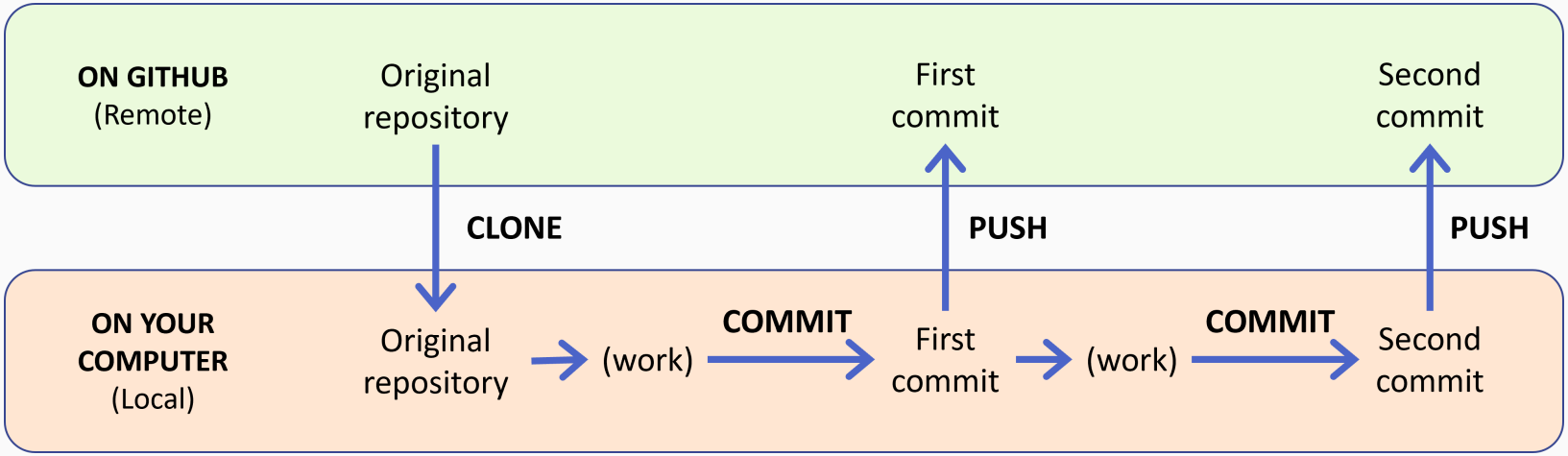
- The tools that we're using all form part of a coherent **data science ecosystem**.
- Greatly reduces the cognitive overhead ("aggregation") associated with traditional workflows, where you have to juggle multiple programs and languages at the same time.

Github Desktop Workflow

With GitHub, if we were working

1. On our own
2. From a single computer

we could just follow the below workflow:



Since we're collaborating with others/potentially across machines, we'll also add in a few more actions.

Main Git operations

The first Git operation in the workflow is **Cloning**

(Git) Cloning

No, not *that* kind of cloning.

Cloning: making a local copy of a **GitHub Repository** (repo for short).

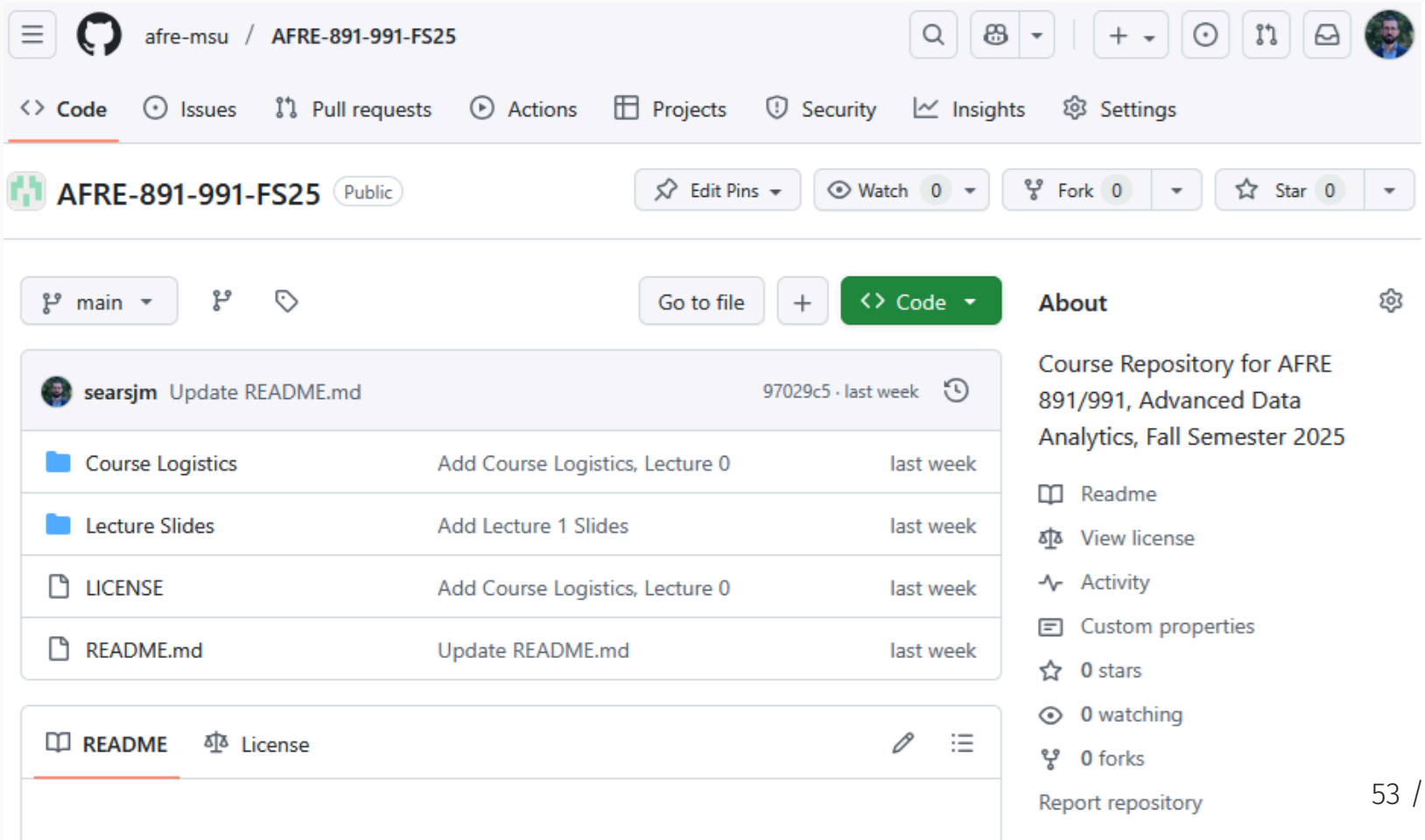
In order to clone a repo, we first need a repo to clone. Let's start by cloning our **Course repo!**

We can do this

1. From the repo page on github.com (direct link or SSH)
2. From GitHub Desktop directly

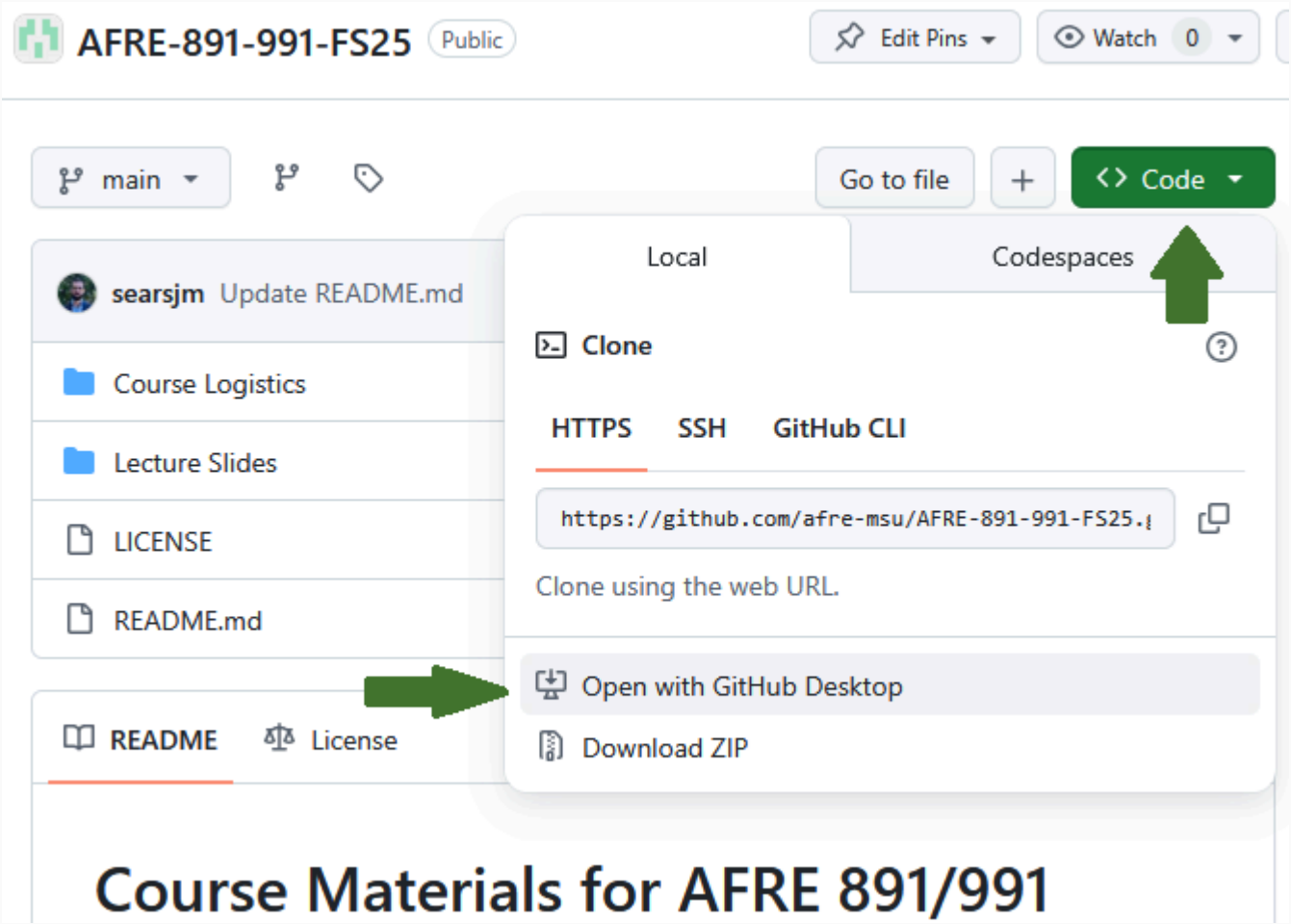
Course Repo Cloning (github.com)

Let's start with the first approach. With GitHub Desktop installed/open, navigate to our **class web page repo**.



Course Repo Cloning

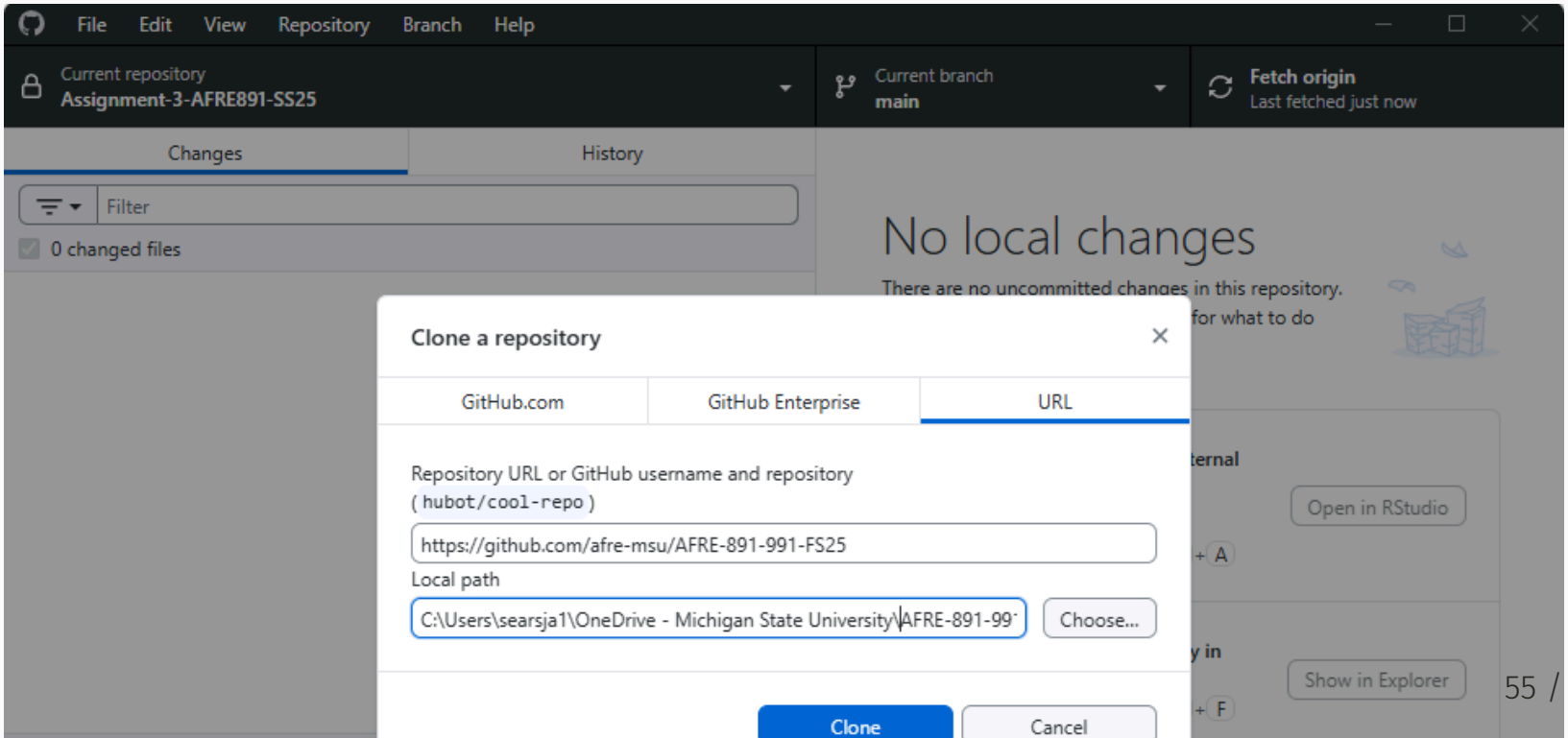
Click the green **Code** button and then "Open with GitHub Desktop"



Course Repo Cloning

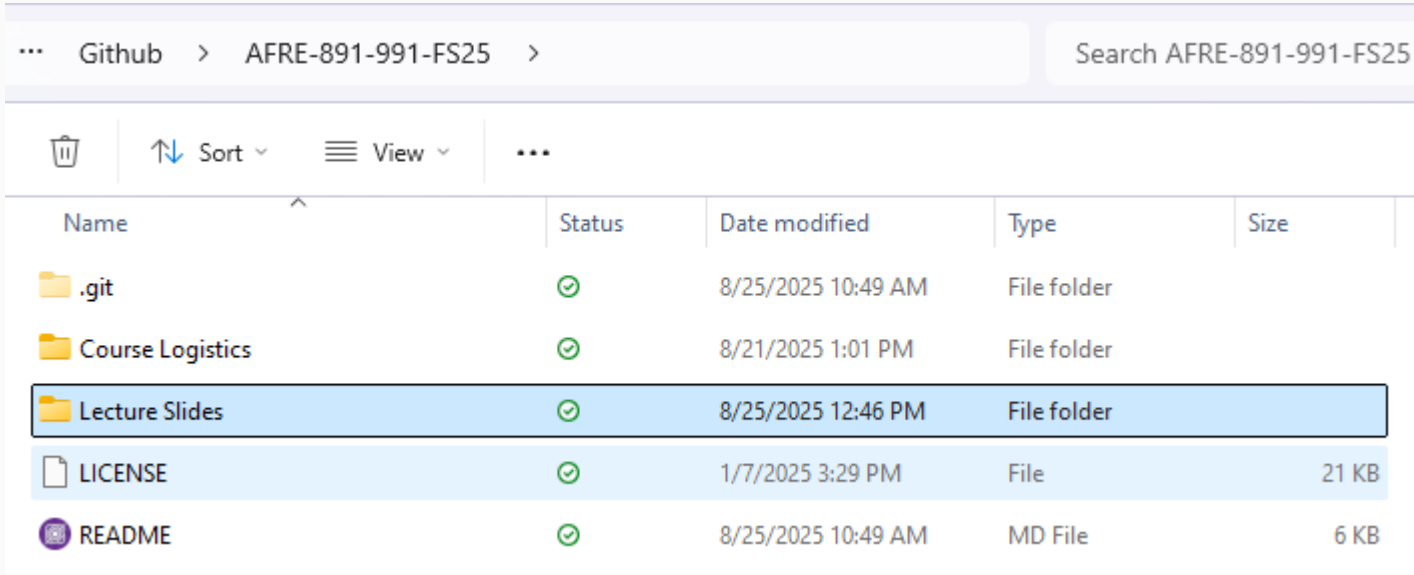
Allow your web browser to open the link. This will redirect you to GitHub Desktop, automatically adding in the repo URL.

Next, choose where you want the local copy of the repo file's saved and hit **Clone**



Course Repo Cloning

Wait a little bit, and navigate to the local path you gave it. Voila!



Creating a Repo

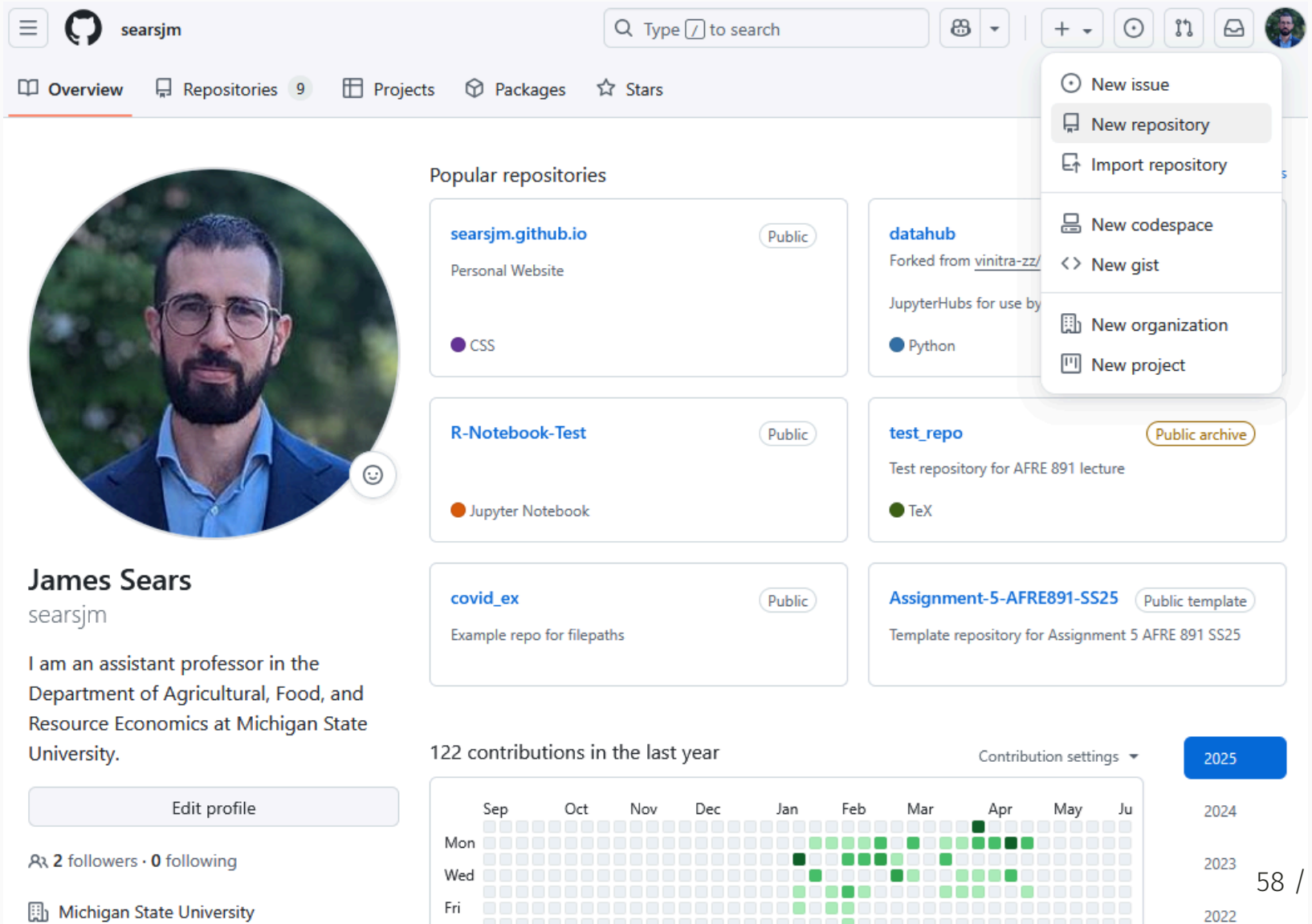
While there are *tons* of awesome repositories on GitHub that we might want to clone and use as our starting point, other times we want to **create our own repo!**

Let's see how to do this from github.com.

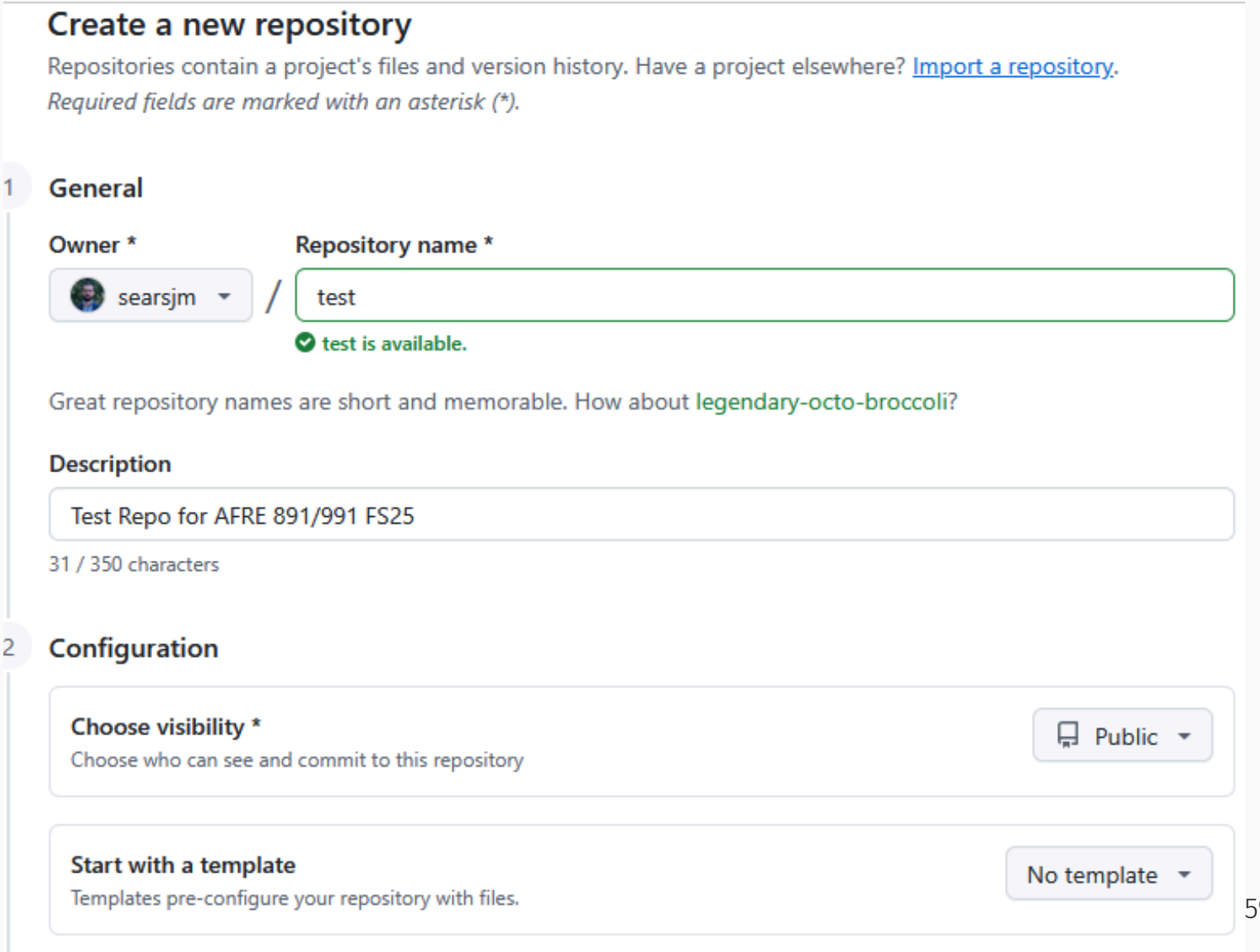
Start by creating your own repo on GitHub — call it "test" — and clone it, this time using the HTTPS link.[3]

[3] It's easiest to start with HTTPS, but **SSH** is advised for more advanced users.

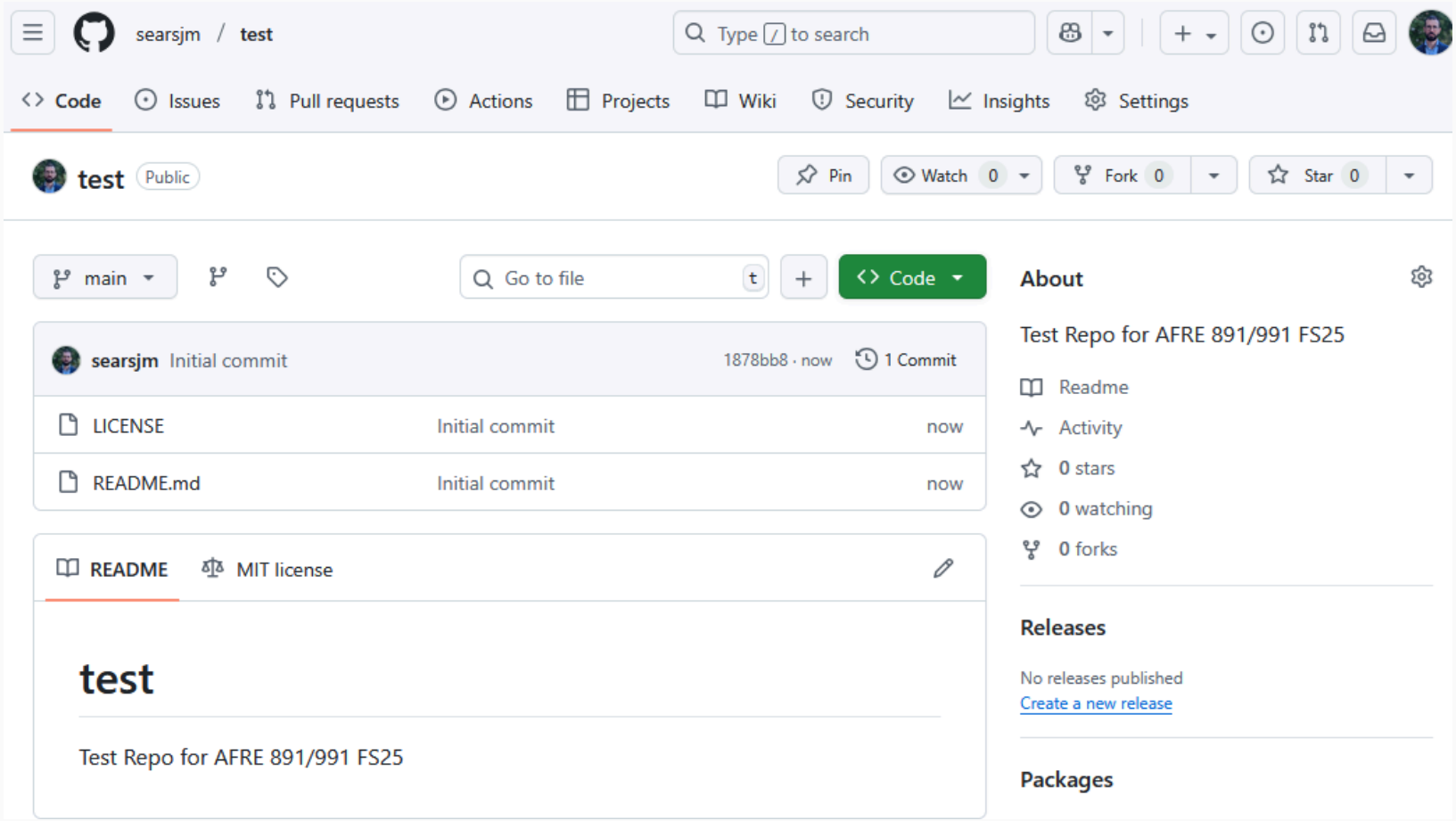
Create a Repository on GitHub (Repo)



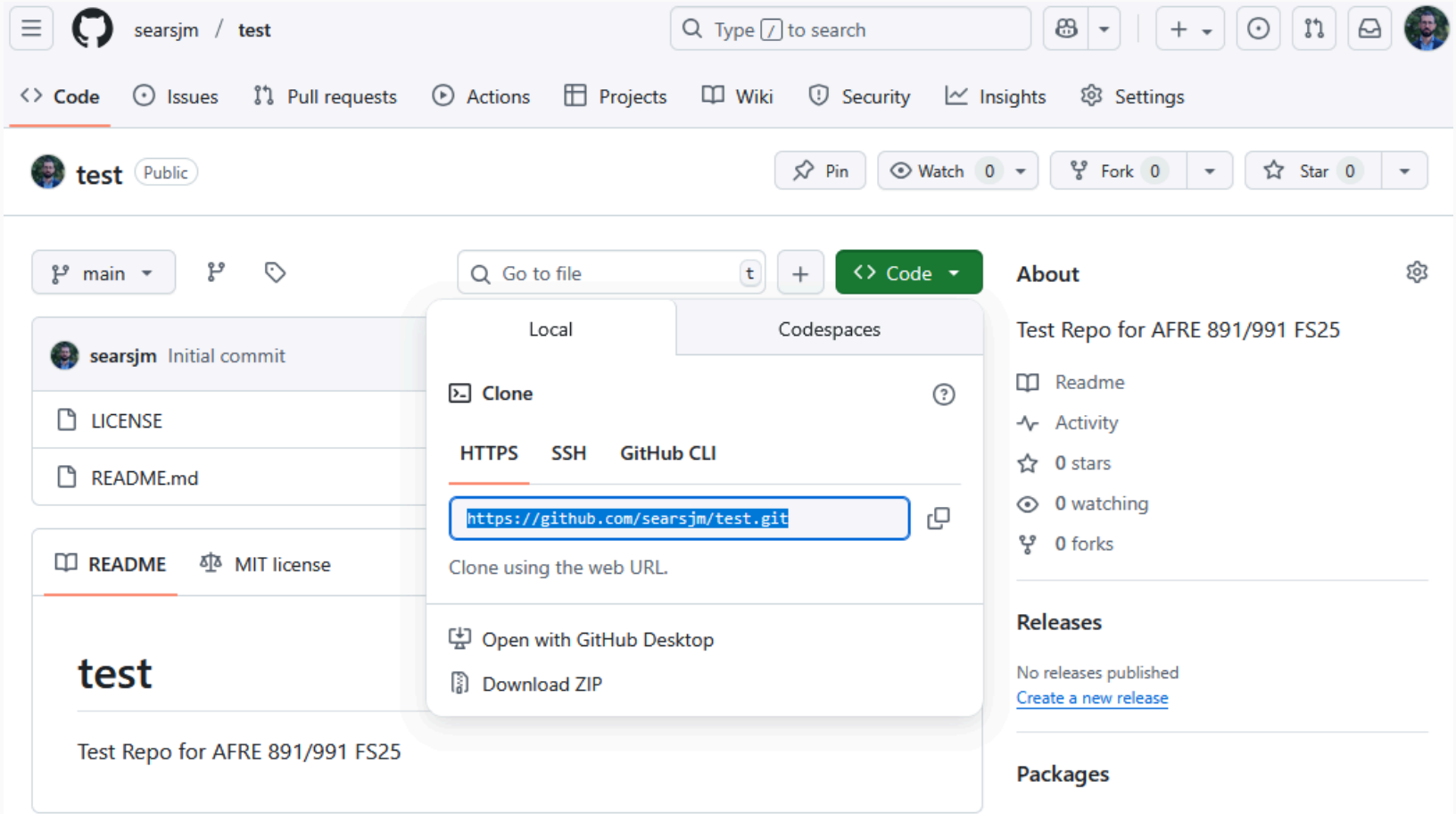
Create a Repository on GitHub (Repo)



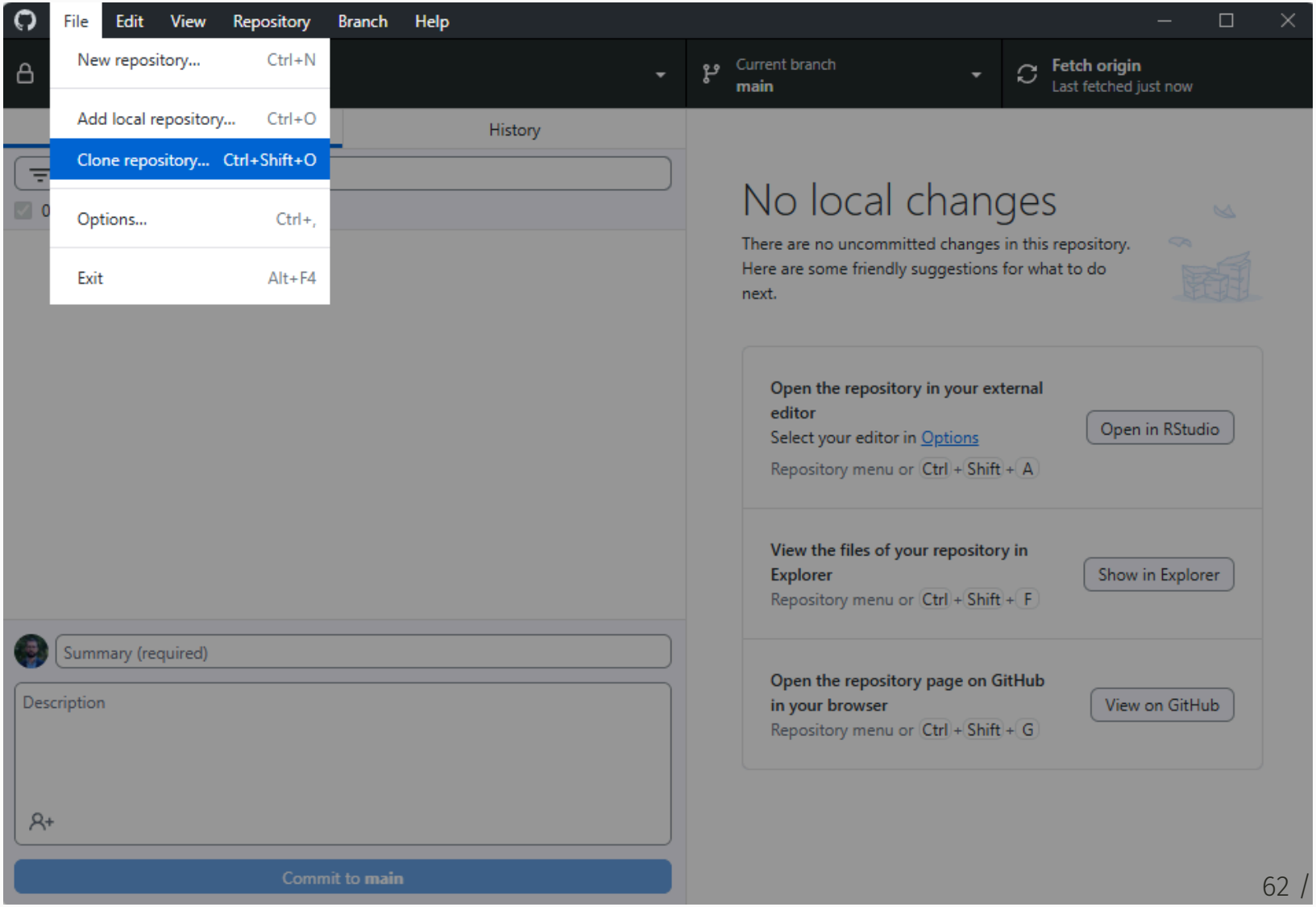
Create a Repository on GitHub (Repo)



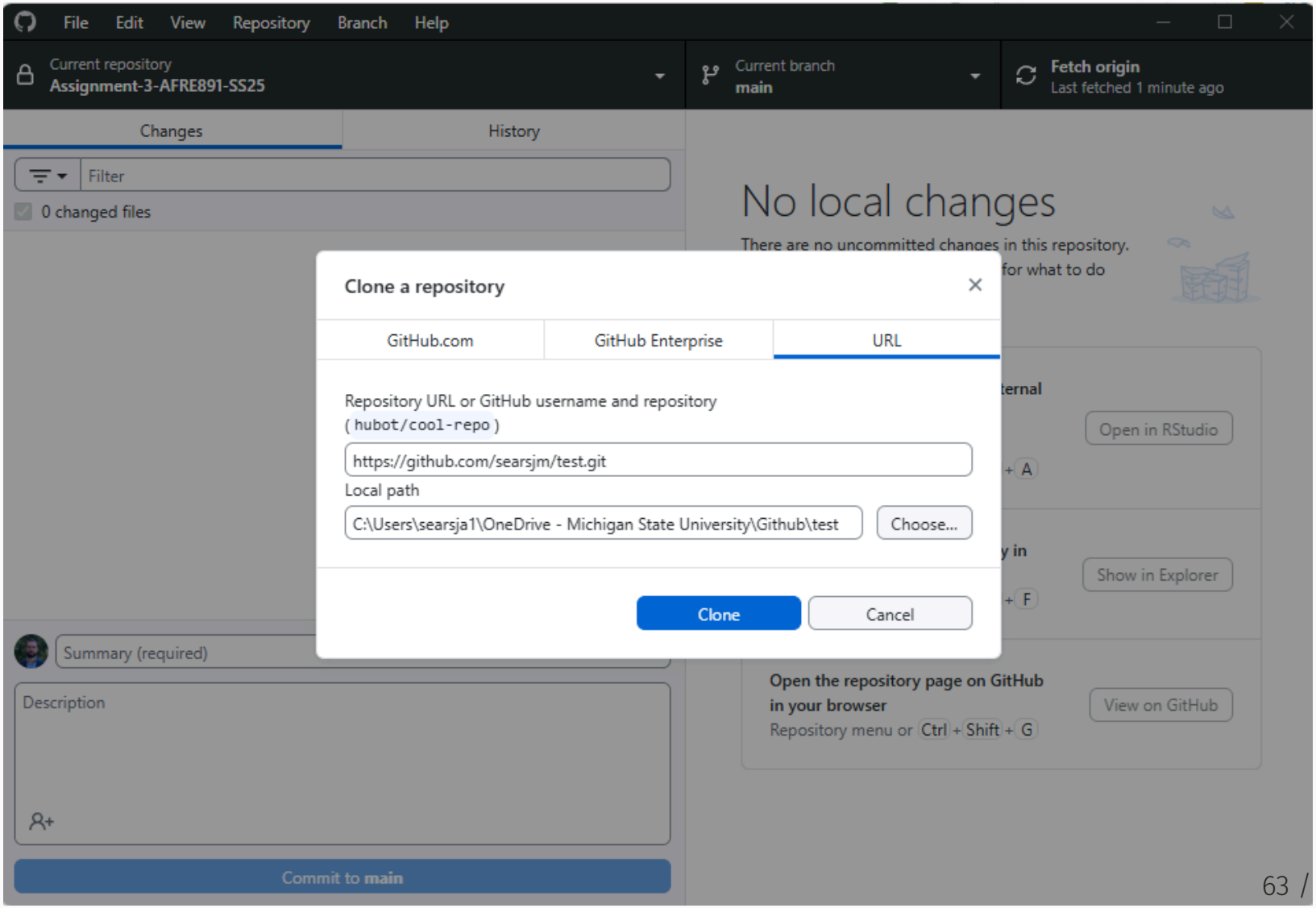
Create a Repository on GitHub (Repo)



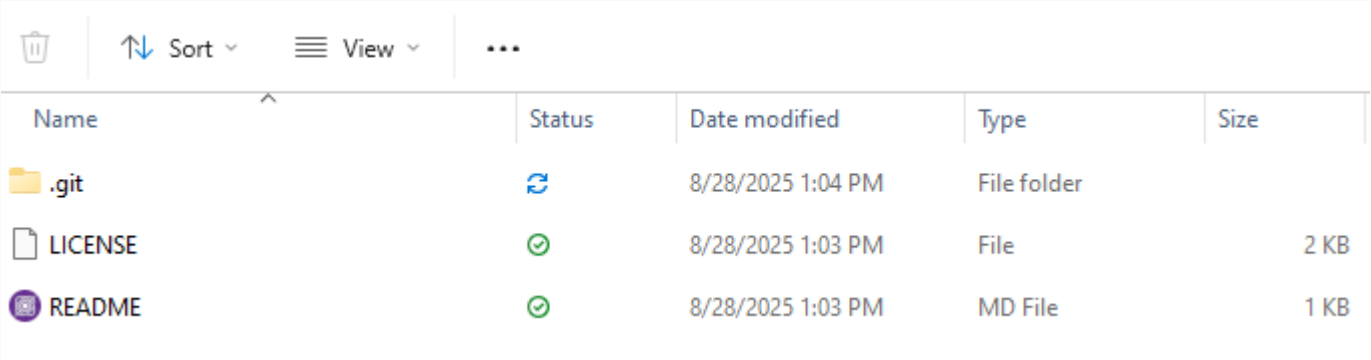
Clone a Repository from GitHub (HTTPS)



Clone a Repository from GitHub (HTTPS)



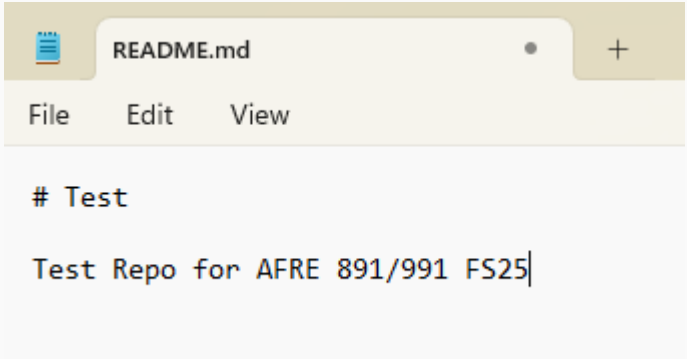
Clone a Repository from GitHub (HTTPS)

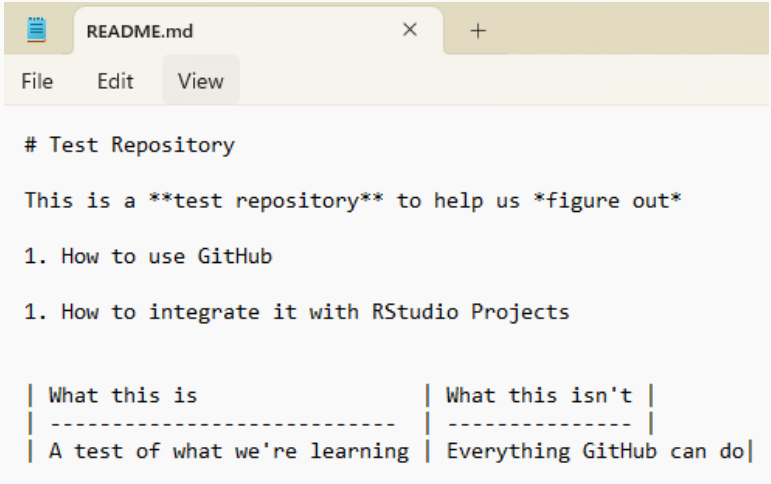


Making Local Changes

Let's start by making some changes to our local copies of the test repo files.

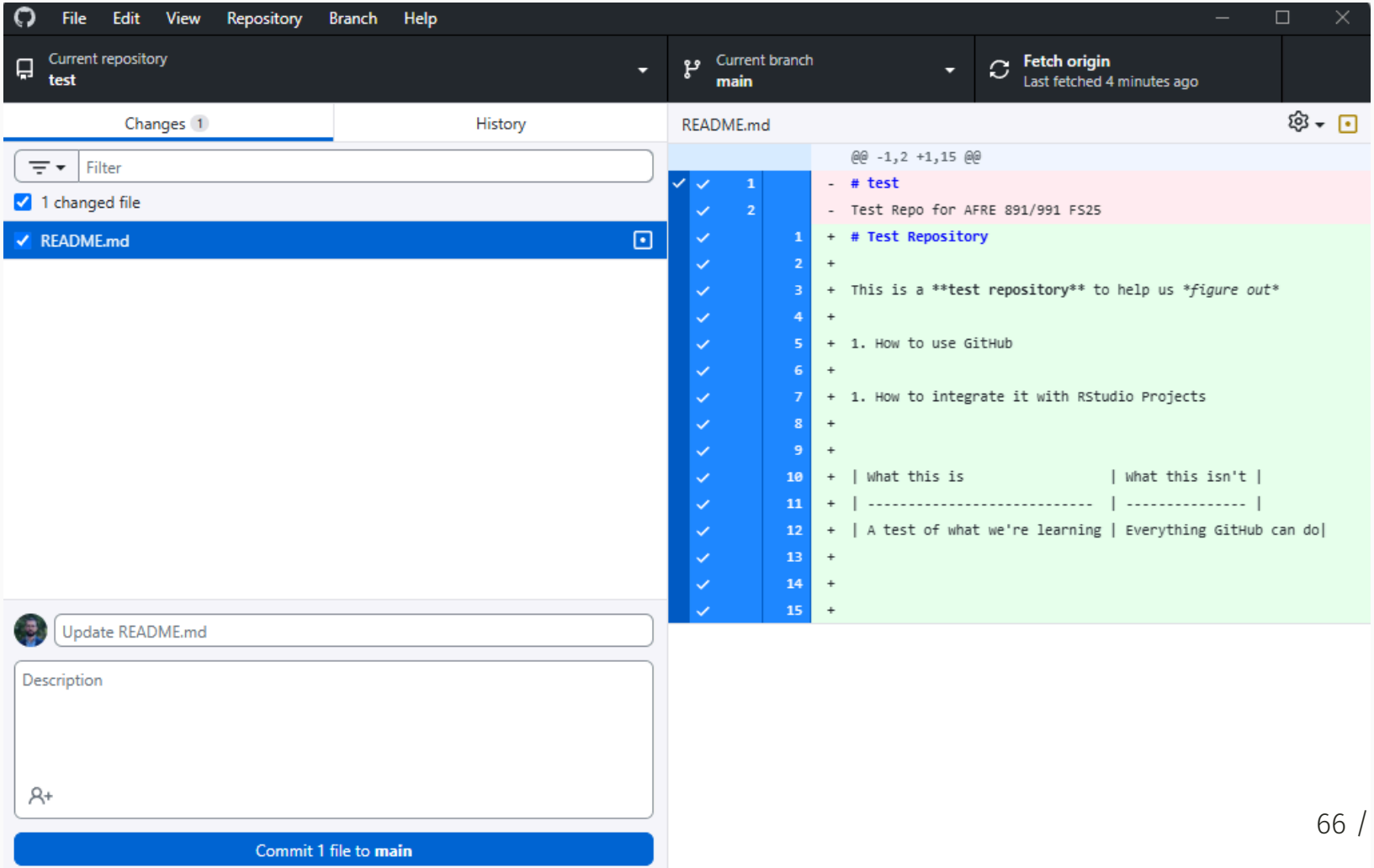
Open up the README file. Add some text and save the README.





Making Local Changes

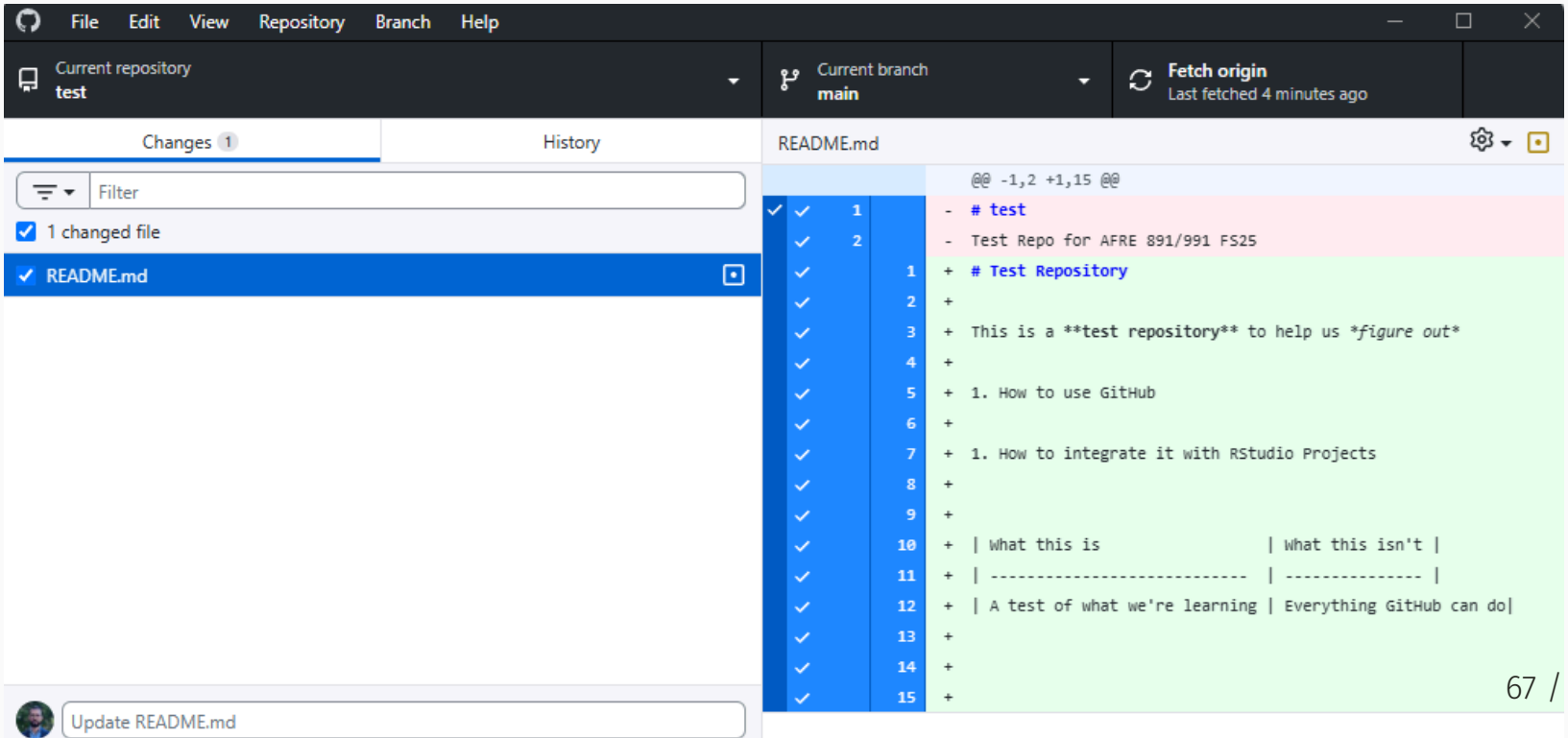
Open up Github Desktop. Do you see any changes?



Stage

Github Desktop detects local changes and automatically **stages** those changes

- Tells Git that you edited the `README.md` file and documents *how* it changed from the upstream repo version



Commit and Push

For now, these changes are **local to our own repo copy**.

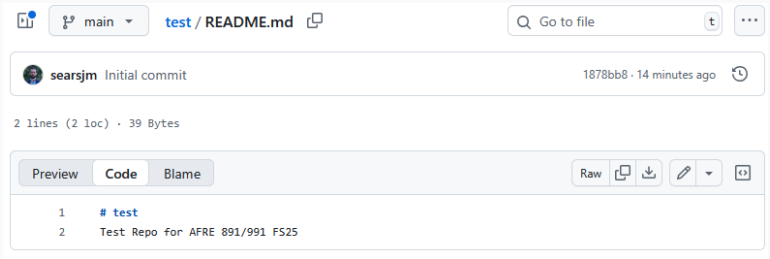
Local System

README.md

File Edit View

```
# Test Repository

This is a **test repository** to help us *figure out*

1. How to use GitHub

1. How to integrate it with RStudio Projects


| What this is                 | What this isn't |
| ---------------------------- | --------------- |
| A test of what we're learning | Everything GitHub can do|
```

GitHub



In order for them to **update the main repo itself**, we need to do two actions: **Commit** and **Push**.

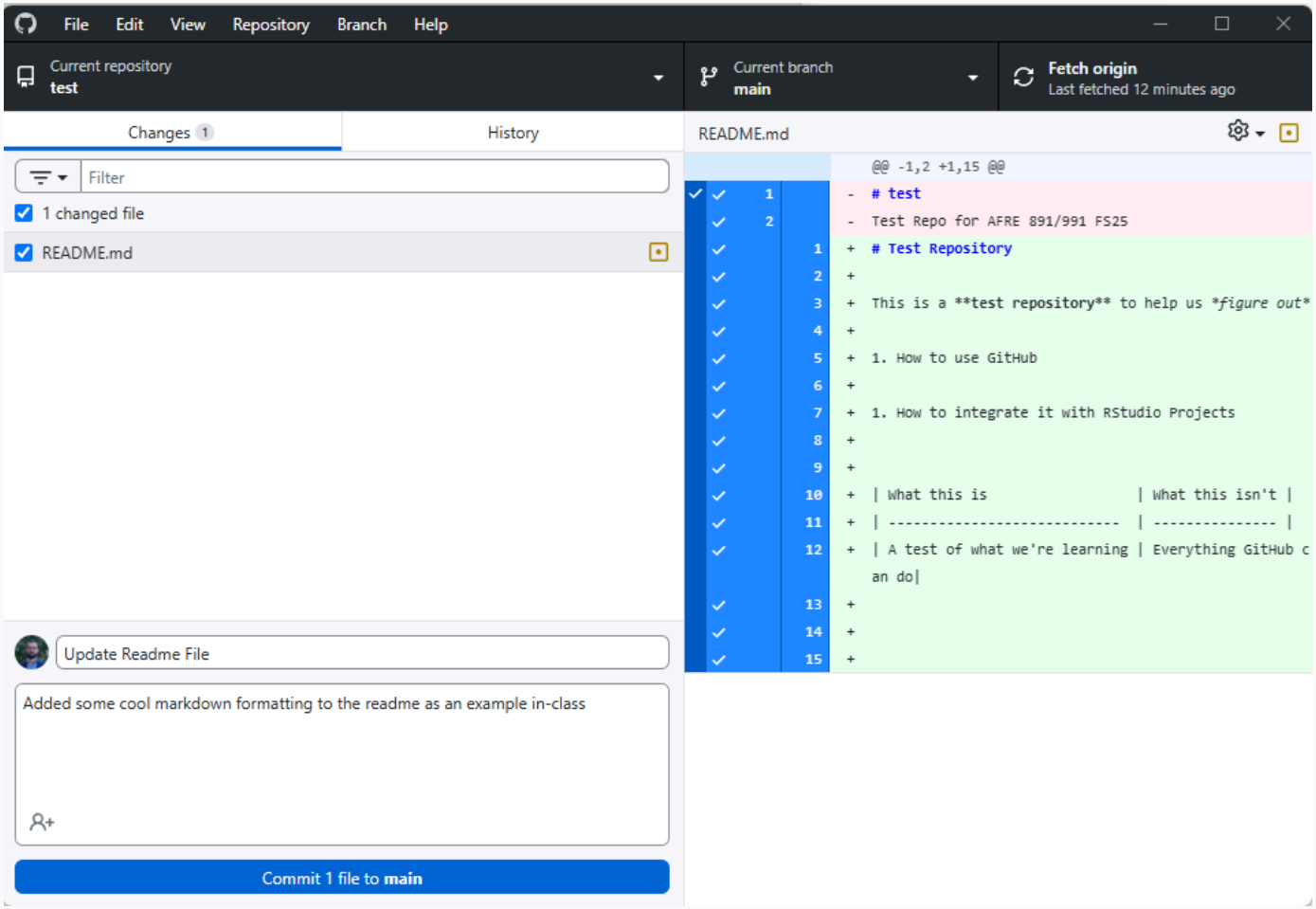
Commit

Github Desktop automatically stages any changes it detects, so that's part of the trouble taken care of already!

Now we need to **Commit:** tell Git that, yes, you are sure these staged changes should be part of the repo history.

Commit

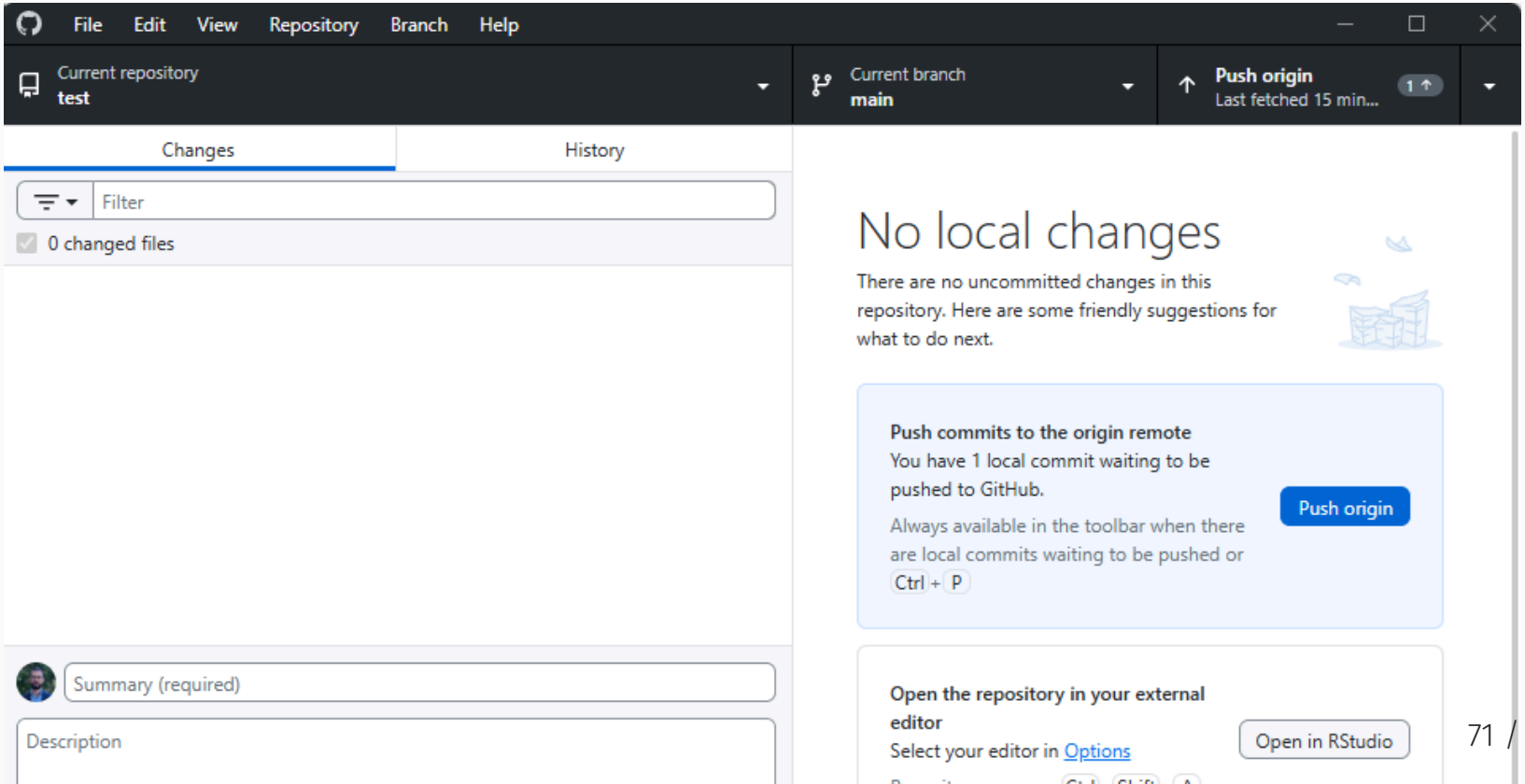
To **commit** in GitHub Desktop, we need to first give our commit a name and a description (bottom-left):



Push

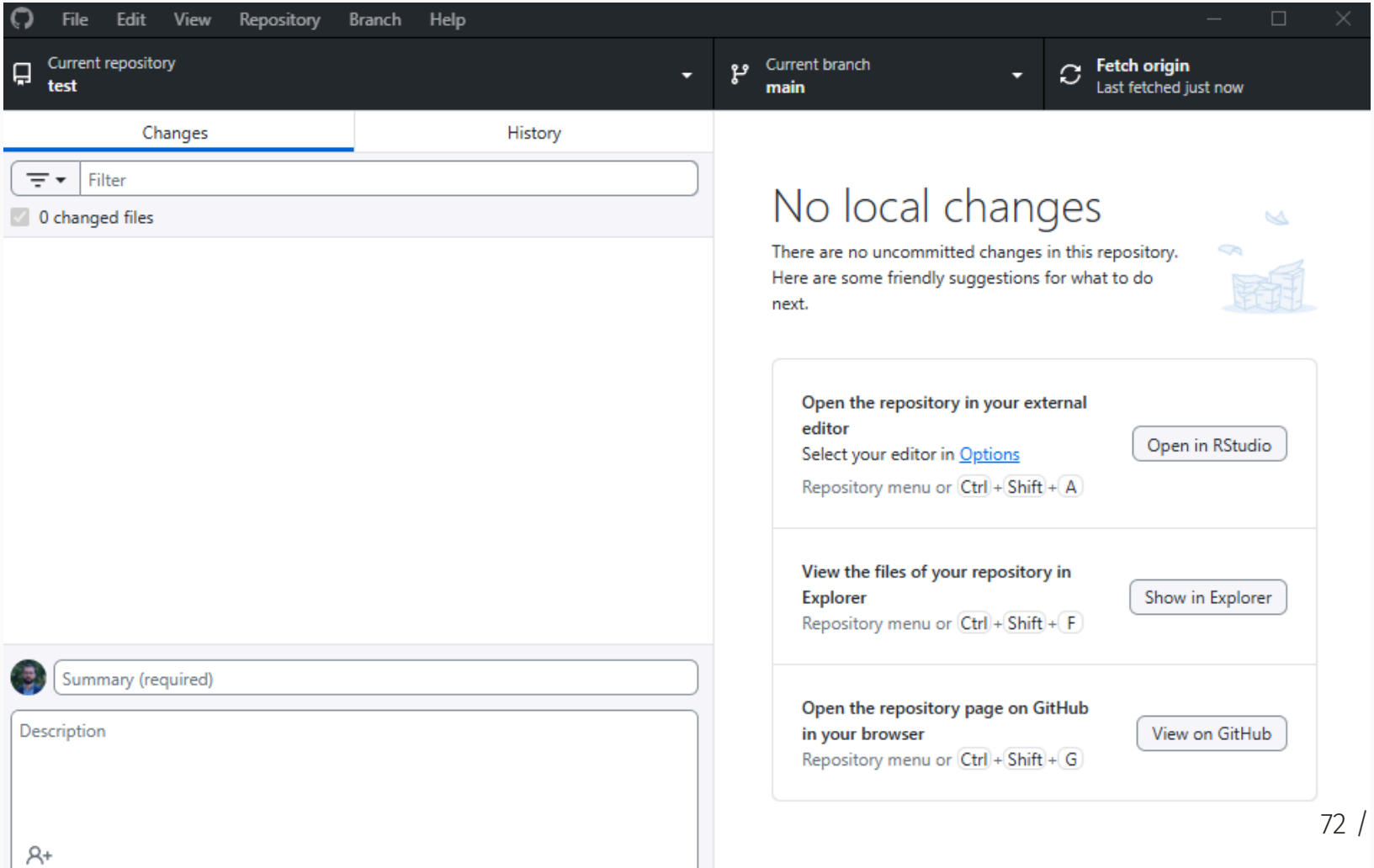
Next we need to **Push:** send the local file changes/additions to update the repo on GitHub.

In the upper-right corner you should now see the option to **Push origin**:

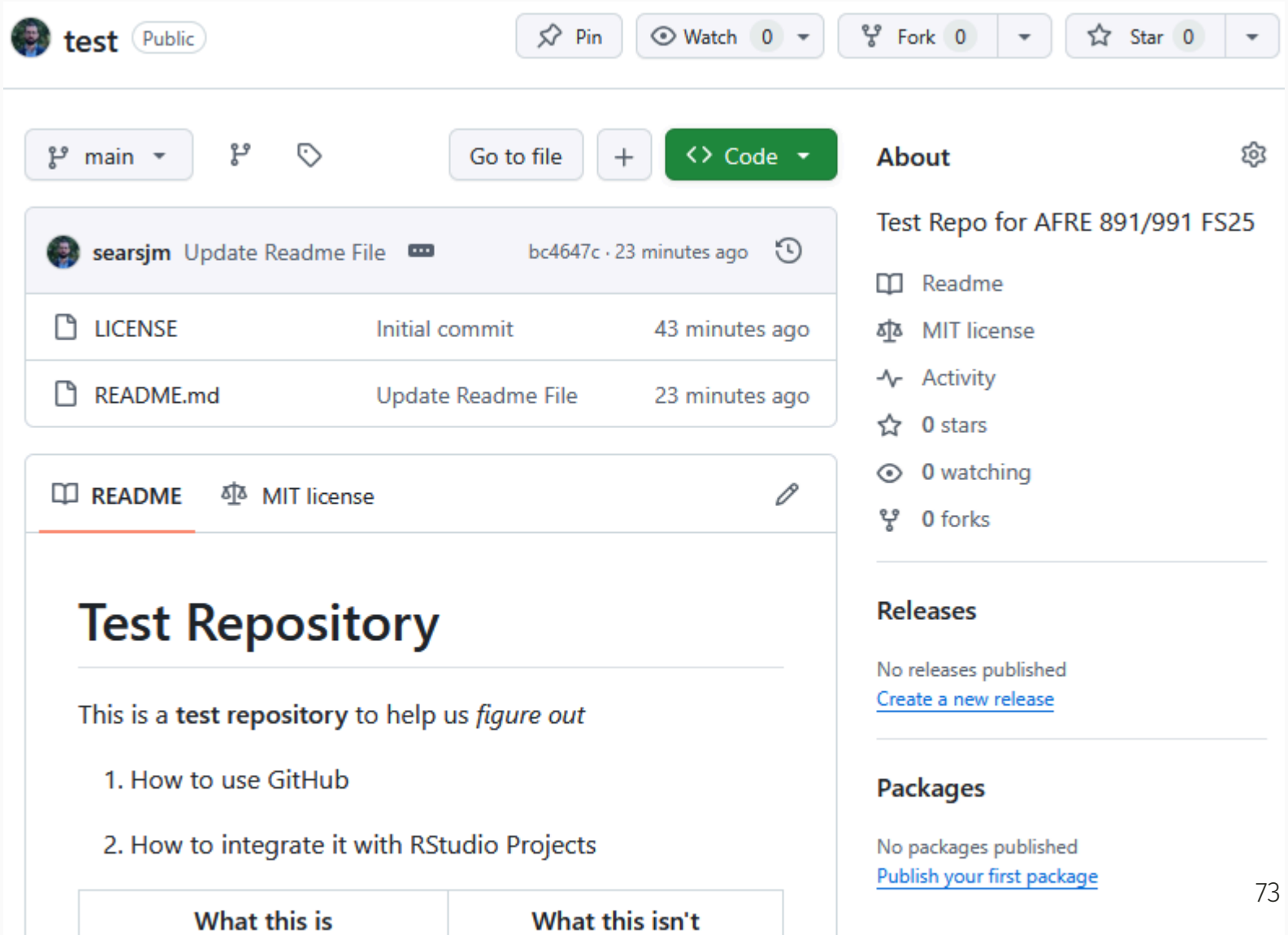


Push

Click it and wait. That's it!



Changes Now Visible on GitHub



Recap

Here's a step-by-step summary of what we just did.

- Cloned a repo from GitHub
- Made same changes to a repo file and saved them locally.
- **Staged** these local changes (okay, Github Desktop did that for us).
- **Committed** these local changes to our Git history with a helpful message.
- **Pushed** our changes to the GitHub repo.

This completes the single collaborator/computer workflow.

But what if we're in a multiple author and/or multiple system situation?

A: we need to bring the last Git Action: **Pull**.

Pull

Just like we **Pushed** our downstream changes *to* the GitHub repo, we can also **Pull** upstream changes *from* the repo

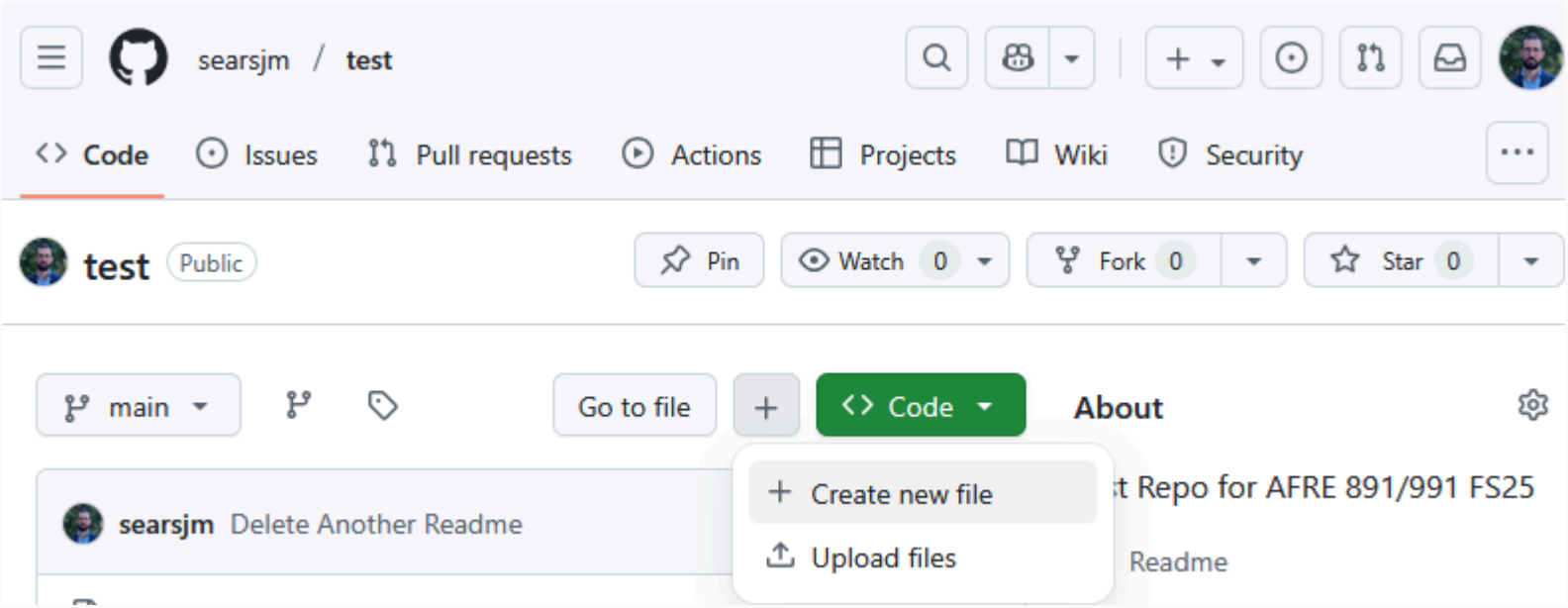
- Overwrites our current version of files with the newer versions from the GitHub repo

When working with others, it's **critical** to **Pull** updates from the GitHub repo just in case anyone else made changes too (not expected here, but good practice).

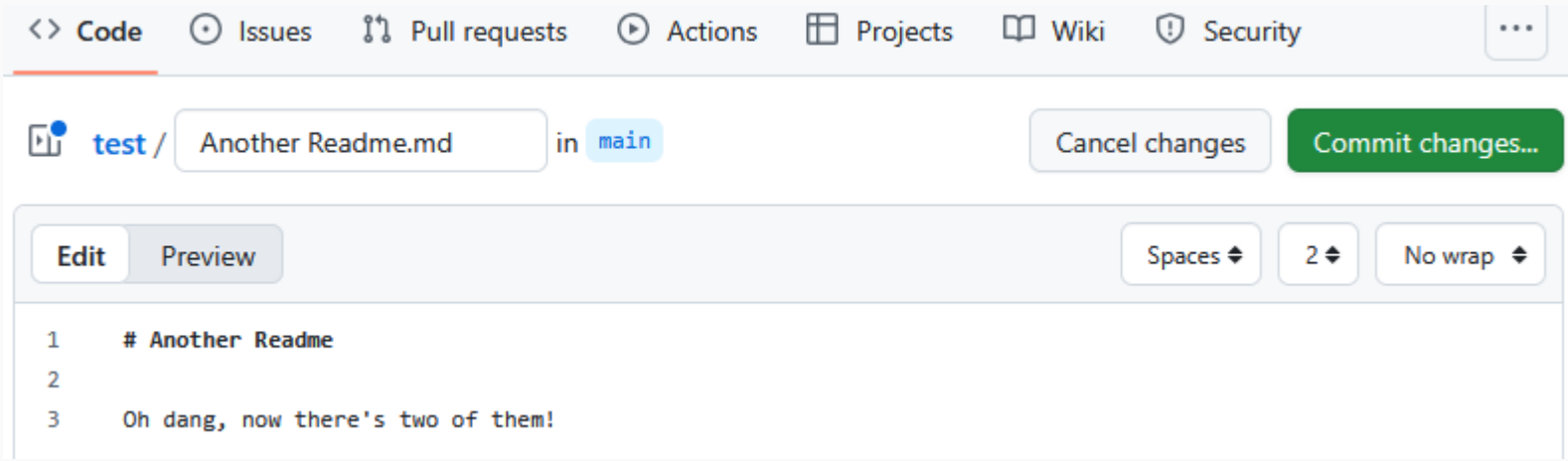
Aside: Always pull from the upstream repo *before* you push any changes. Seriously, do this even on solo projects; making it a habit will save you headaches down the road.

Pull

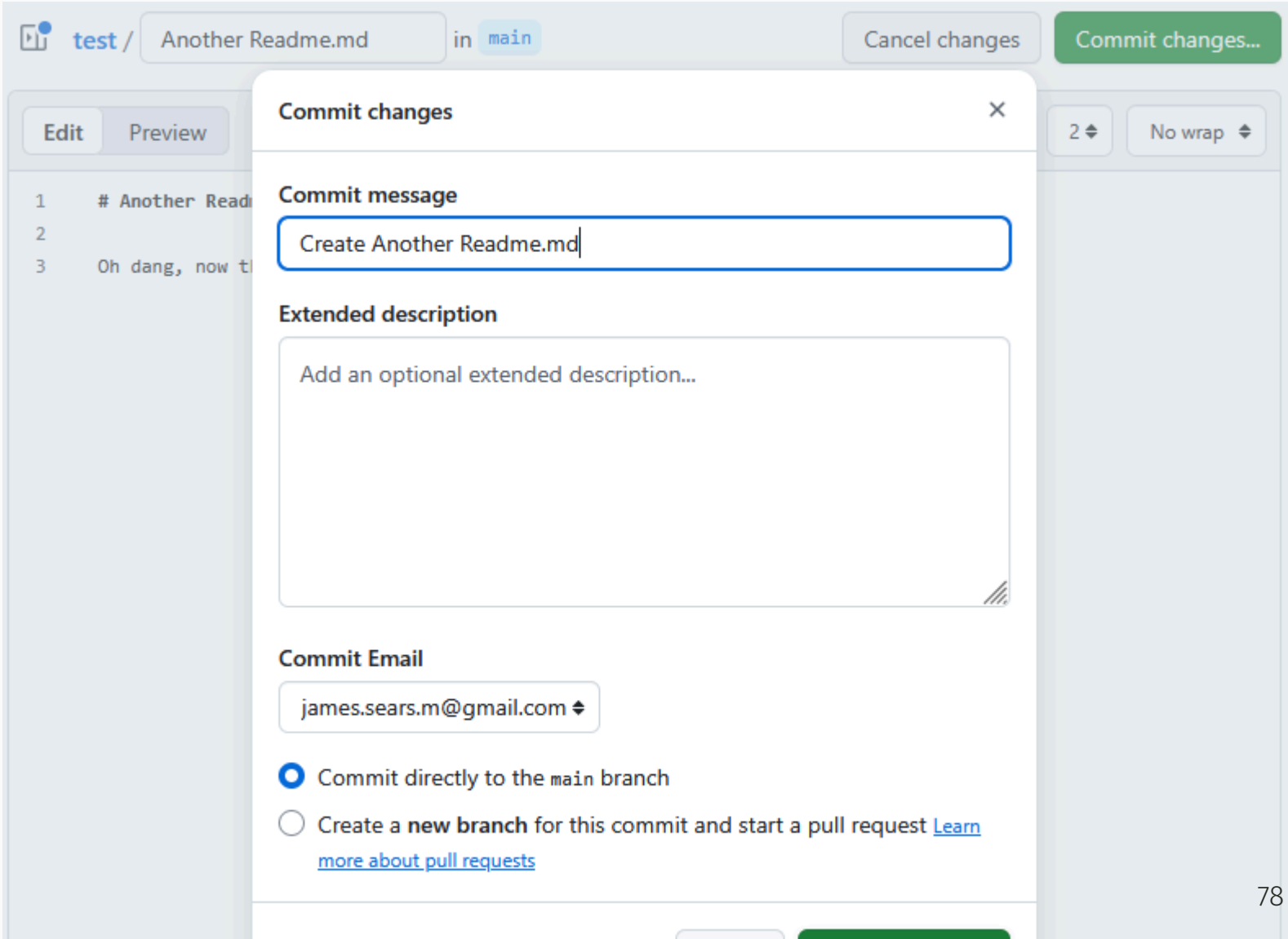
To set up a possible pull, let's make some **upstream changes**. On Github, click on the plus symbol to **add a new file**.



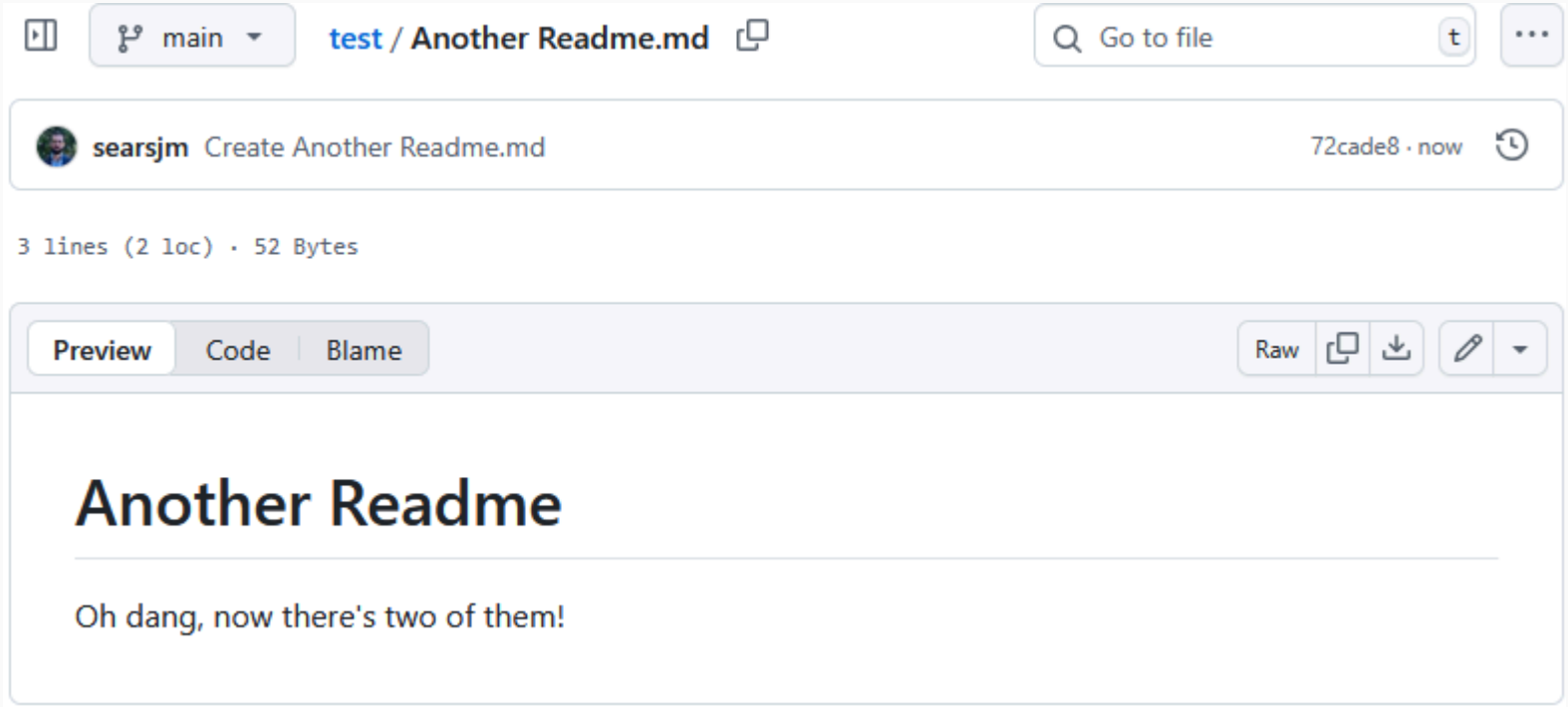
Pull


Code
Issues
Pull requests
Actions
Projects
Wiki
Security
test /
Another Readme.md
in
main
Cancel changes
Commit changes...
Edit
Preview
Spaces
2
No wrap
1 # Another Readme
2
3 Oh dang, now there's two of them!

Pull


test /
Another Readme.md
in
main
Cancel changes
Commit changes...
Edit
Preview
2
No wrap
1 # Another Read
2
3 Oh dang, now t
Commit changes
Commit message
Create Another Readme.md
Extended description
Add an optional extended description...
Commit Email
james.sears.m@gmail.com
Commit directly to the main branch
Create a new branch for this commit and start a pull request Learn more about pull requests

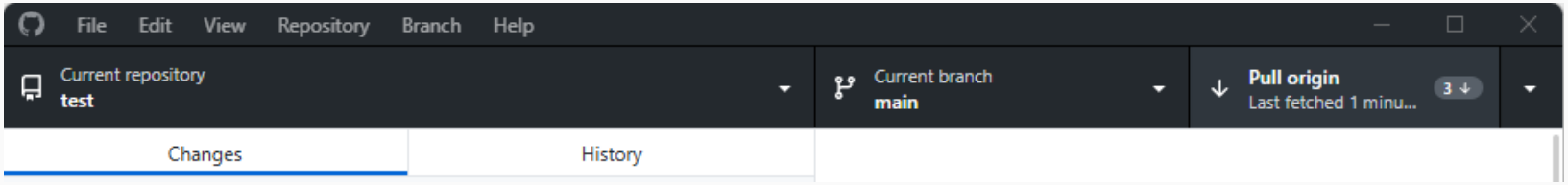
Pull


main
test / Another Readme.md
Go to file
t
searsjm Create Another Readme.md
72cade8 · now
3 lines (2 loc) · 52 Bytes
Preview
Code
Blame
Raw
Another Readme
Oh dang, now there's two of them!

Pull

Flip back over to GitHub Desktop. One of two things may have happened.

1. Your GitHub Desktop already detected the upstream changes

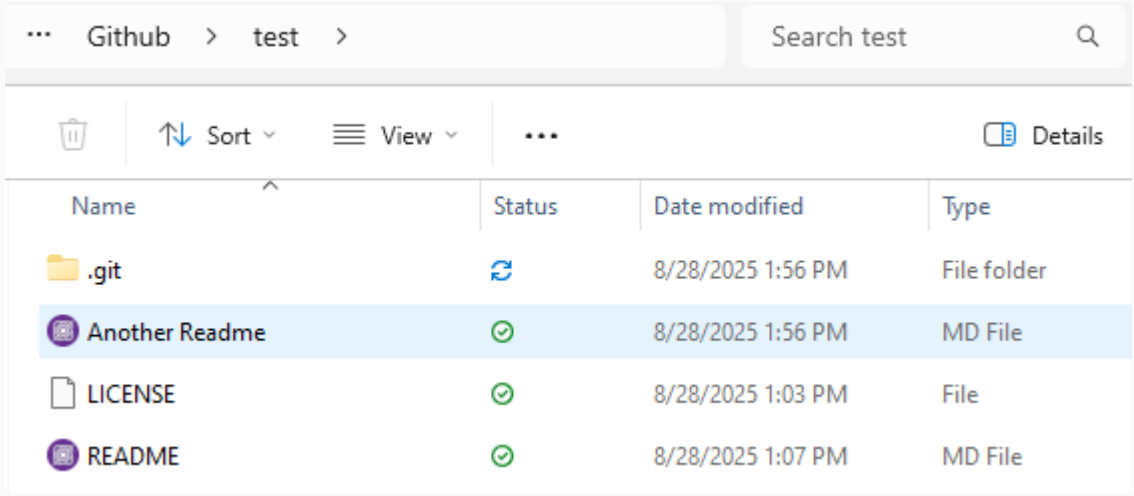


If you just see **fetch origin**, click it to refresh and retrieve the changes.

Pull

Click the **pull origin** button to add the new file to your local clone

- Same thing would have happened if someone had modified existing files



Why this Workflow: GitHub

Creating the repo on GitHub first means that it will **always be "upstream"** of your (and any other) local copies.

- In effect, this allows GitHub to act as the **central node** in the distributed VC network.
- Especially valuable when you are collaborating on a project with others, but also has advantages when you are working alone.
- If you would like to move an existing project to GitHub, my advice is still to **create an empty repo there first**, clone it locally, and then copy all your files across.

Other Tips and Productivity Tools

Productivity Miscellanea

What follows are miscellaneous things that I find **improve my productivity**

- **Synced Cloud Storage** (SpartanDrive or Dropbox/Box)
- **Overleaf** for LaTeX collaboration
- **Connected Papers** for literature networks

Synced Cloud Storage and OneDrive

Synced Cloud Storage is hugely beneficial if you work across **multiple computers** or **with many collaborators**.

- Make sure each computer has the most up-to-date version of all your files
- Renders flash drives almost entirely obselete!

Synced Cloud Storage and OneDrive

One easy way to do this: **SpartanDrive/OneDrive**

All faculty + staff get **5 TB of free storage** on **SpartanDrive (MSU's version of OneDrive)**

Pros

- Free
- Syncable desktop apps
- Part of the MSU Office365 ecosystem

Cons

- Part of the MSU Office365 ecosystem
- Limited storage (5TB, 250gb max filesize)
- Sometimes finicky

Synced Cloud Storage

Alternatives to SpartanDrive:

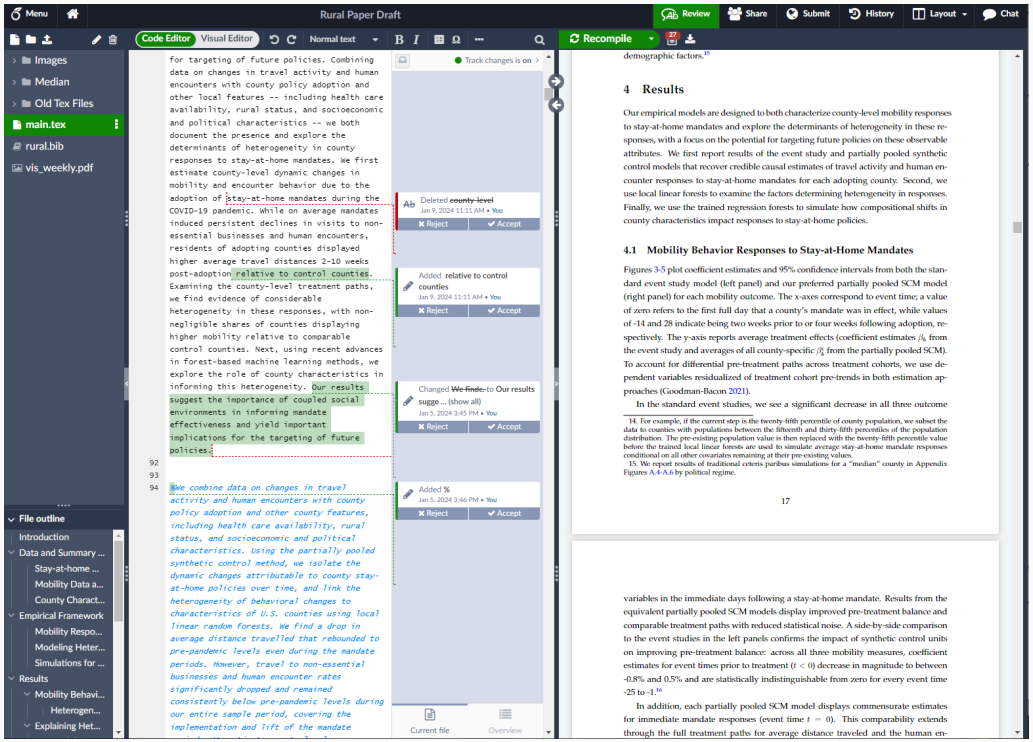
- **Dropbox**: 2GB free ($10/mo for 2TB)
- **Box:** 10GB free ($10/mo for 100GB)

Ultimate choice of platform may depend on coauthors + your current choice, but it's a good idea to **download the desktop app** to keep your files synced + backed up!

- Also gives basic version control

Overleaf

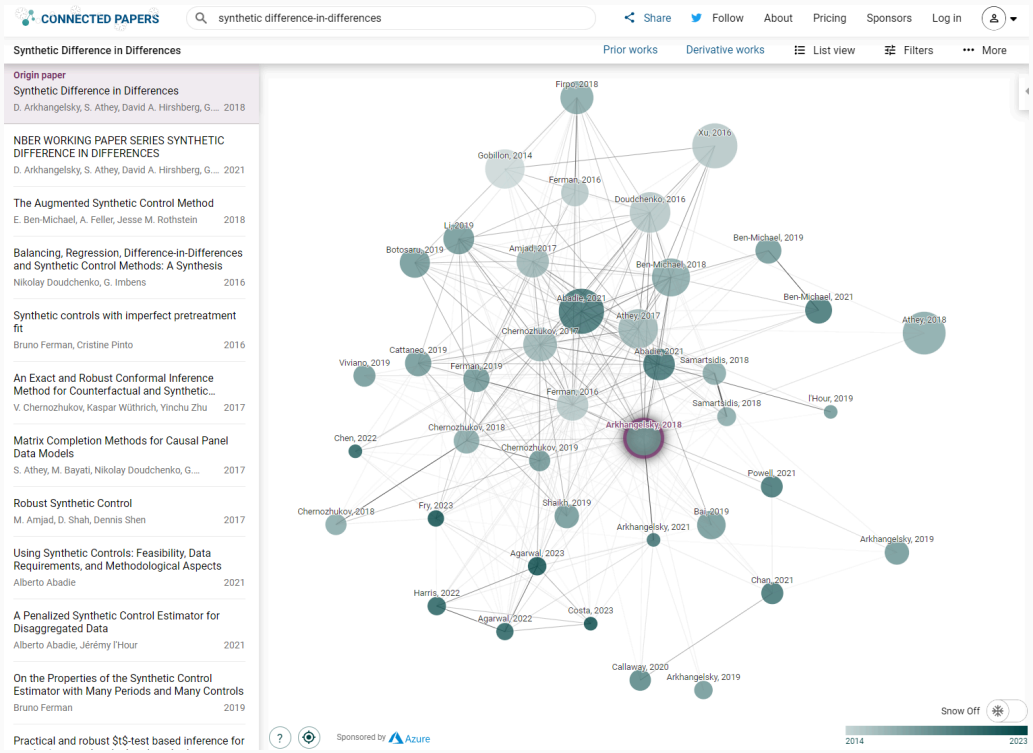
If you typeset using LaTeX, **Overleaf** streamlines access and collaboration



Free Version: remotely host all files, access for you + 1 collaborator

Paid ($7.40/mo): track changes + full document history, Git(hub) + Dropbox integration, up to 6 collaborators

Connected Papers



- Graphical representation of paper networks
  - Visualize literature as directed graph
  - 5 free per month (**unlimited for $6/mo**)

Your Productivity Tips + Tricks?

Git(Hub) and RStudio

Git(Hub) Integration in RStudio

One of the (many) features of RStudio is how well it integrates version control into your everyday workflow.

- Even though Git is a completely separate program to R, they feel like part of the same "thing" in RStudio.
- This next section is about learning the basic Git(Hub) commands and the recipe for successful project integration with RStudio.

I also want to bookmark a general point that we'll revisit many times during this course:

- The tools that we're using all form part of a coherent **data science ecosystem**.
- Greatly reduces the cognitive overhead ("aggregation") associated with traditional workflows, where you have to juggle multiple programs and languages at the same time.

Link a GitHub Repo to an RStudio

The starting point for our workflow is to link a GitHub repository (i.e. "repo") to an RStudio Project. Here are the steps we're going to follow:

1. Create the repo on GitHub and initialize with a README.
2. Copy the HTTPS/SSH link (the green "Clone or Download" button).
3. Open up RStudio.
4. Navigate to **File -> New Project -> Version Control -> Git**.
5. Paste your copied link into the "Repository URL:" box.
6. Choose the project path ("Create project as subdirectory of:") and click **Create Project**.

Let's practice this using our existing test repo. Start by copying the repo link.

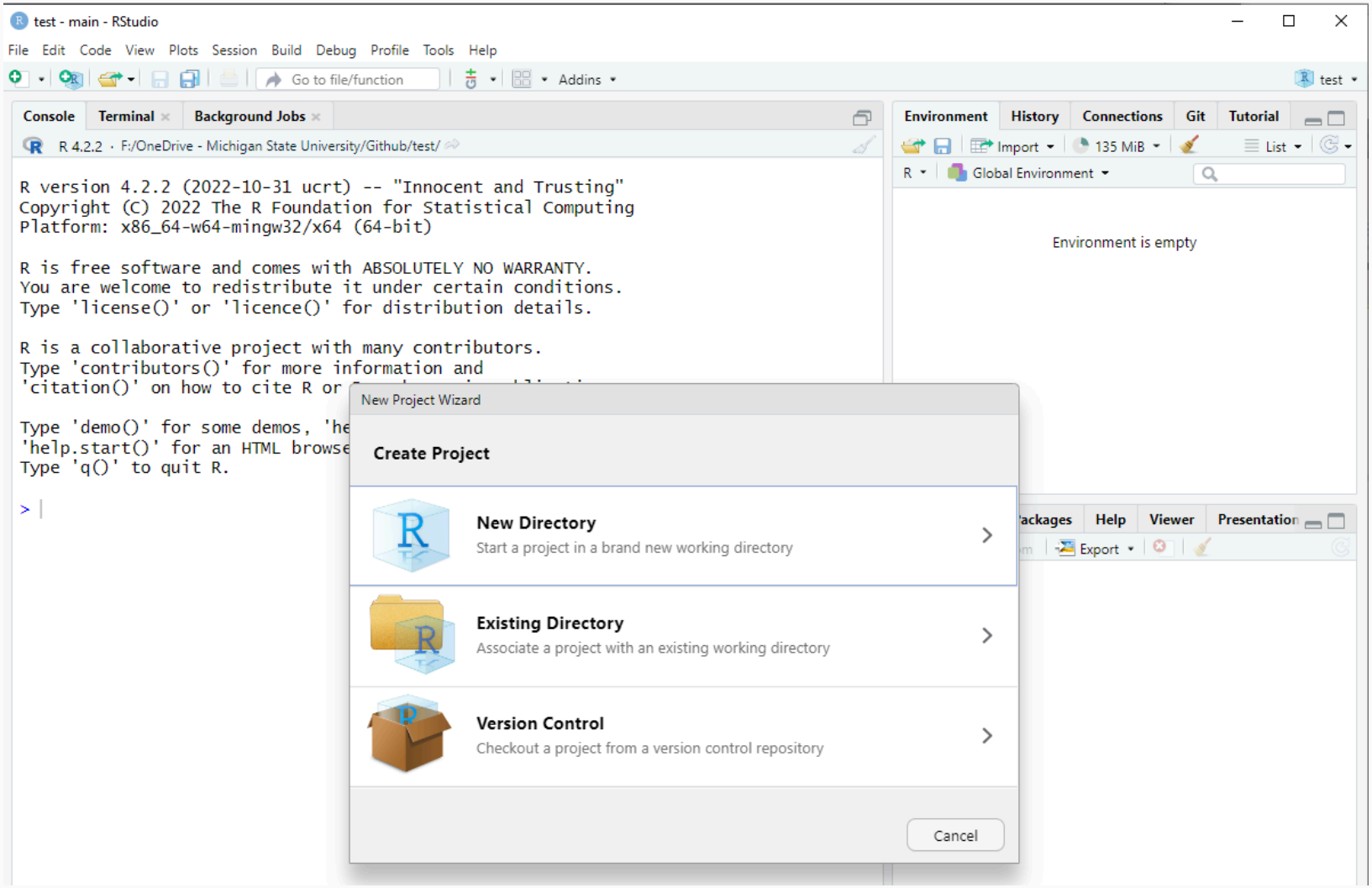
Create new R Project

To start, we need to create an **R Project**

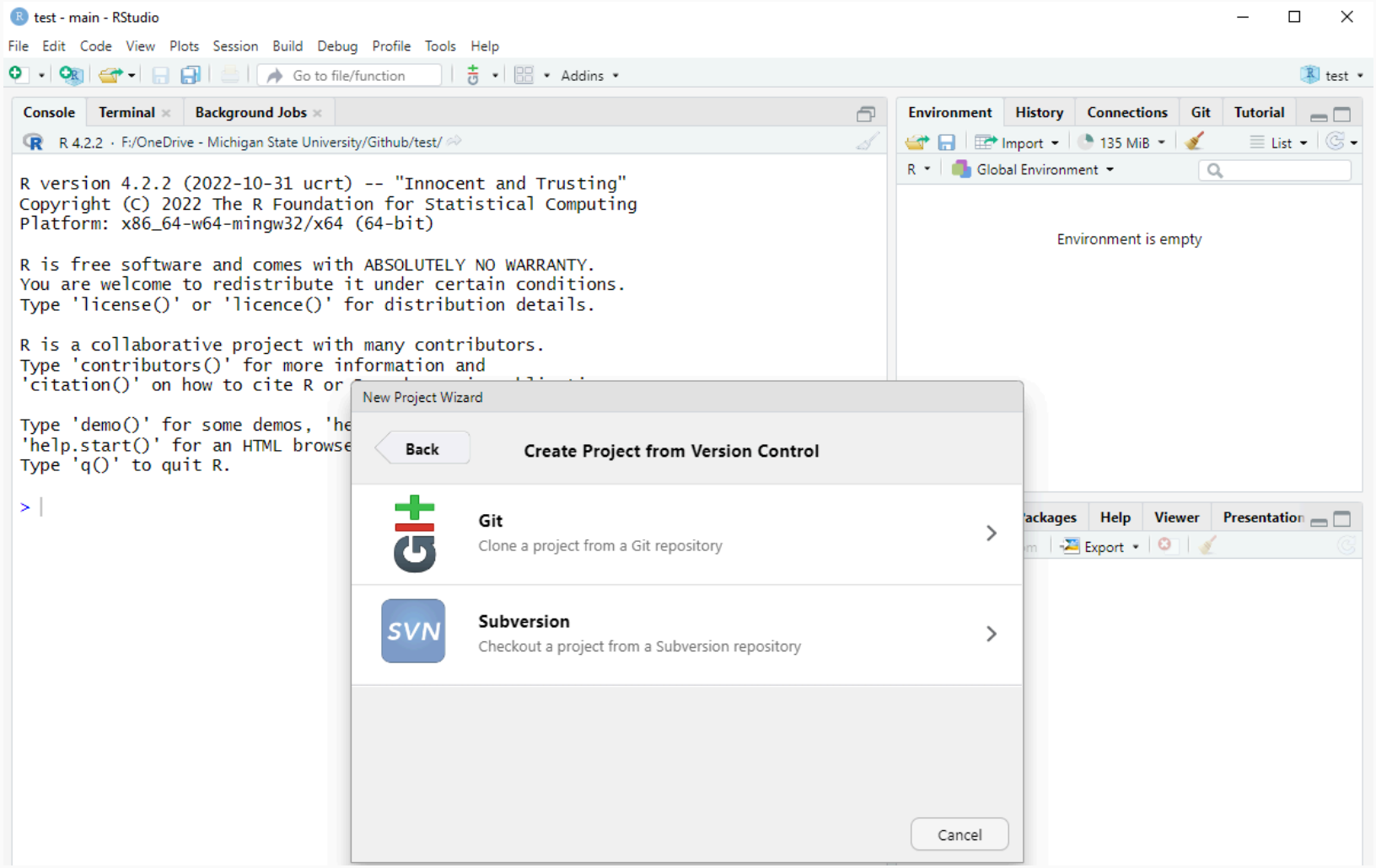
- A project (i.e. folder) specific to a given task

Open a new RStudio window and click **"File > New Project"**

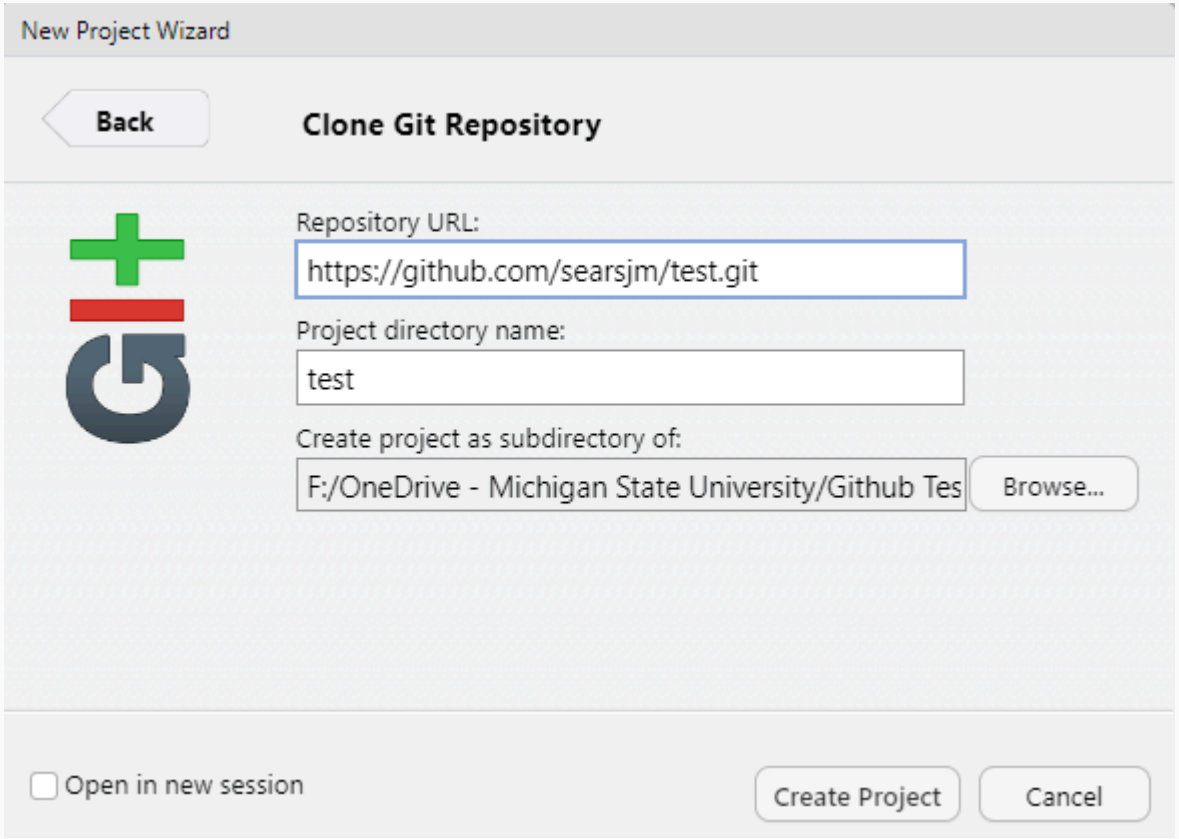
Version Control



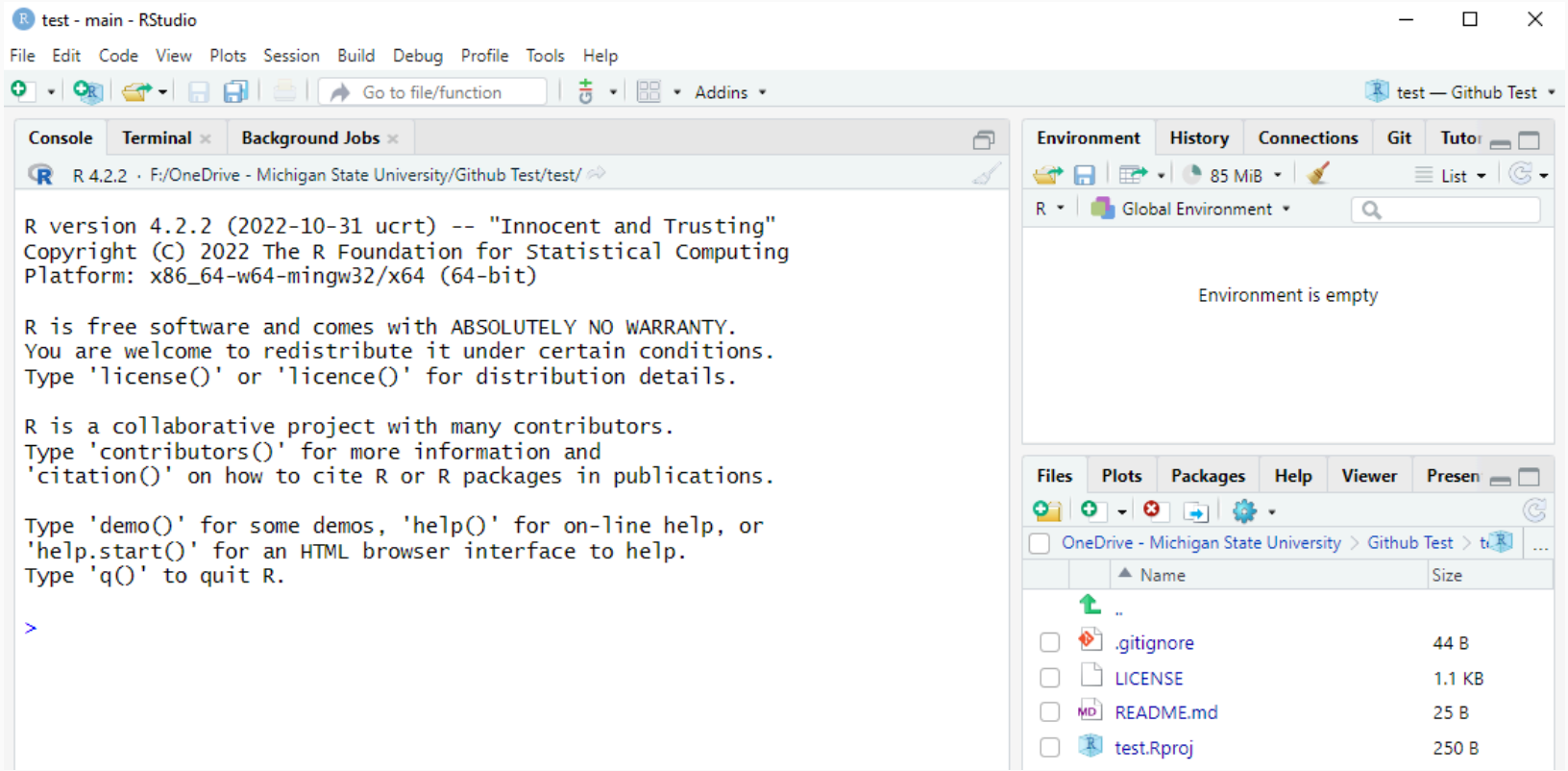
Choose Git


test - main - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
test
Console
Terminal
Background Jobs
R 4.2.2 · F:/OneDrive - Michigan State University/Github/test/
R version 4.2.2 (2022-10-31 ucrt) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or
Type 'demo()' for some demos, 'he
'help.start()' for an HTML browse
Type 'q()' to quit R.
>
Environment
History
Connections
Git
Tutorial
Import
135 MiB
List
R
Global Environment
Environment is empty
Packages
Help
Viewer
Presentation
Export
New Project Wizard
Back
Create Project from Version Control
Git
Clone a project from a Git repository
Subversion
Checkout a project from a Subversion repository
Cancel

Add Repo info



Add Repo into RStudio



Making Local Changes

Look at the **top-right panel** in your RStudio IDE. Do you see the **Git** tab?

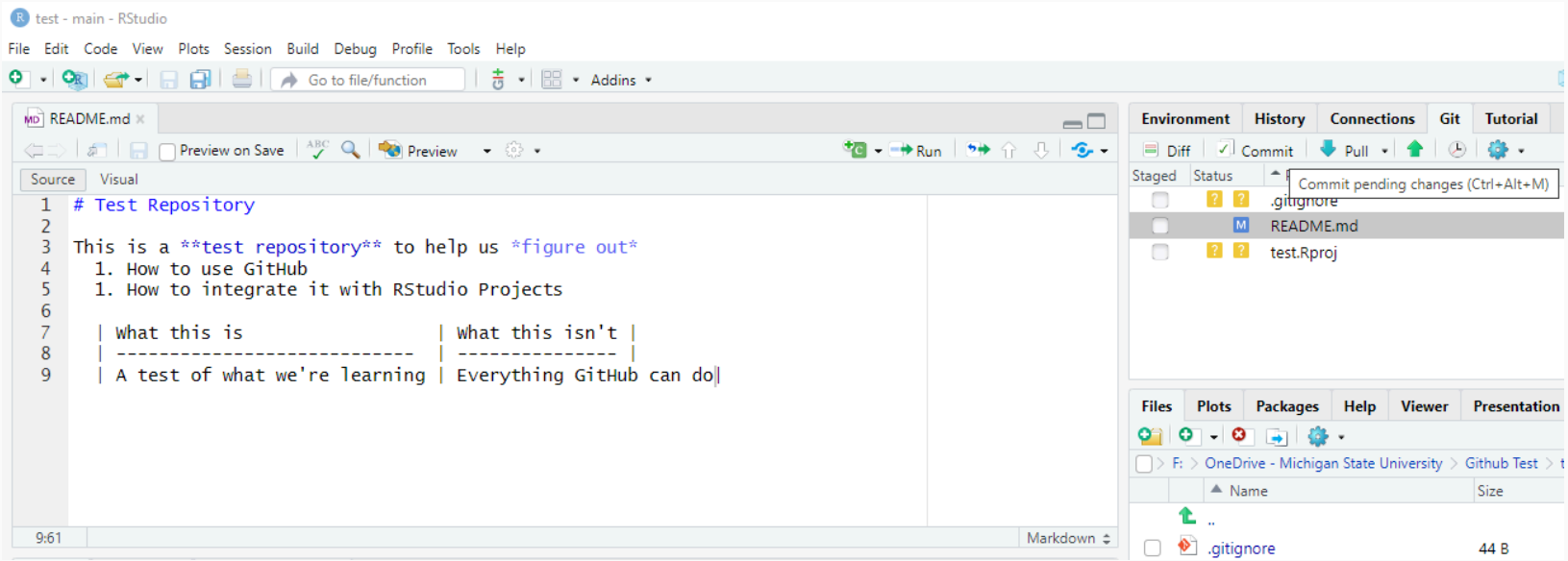
- Click on it.
- There should already be some files in there, which we'll ignore for the moment.[4]

Now open up the README file (see the "Files" tab in the bottom-right panel).

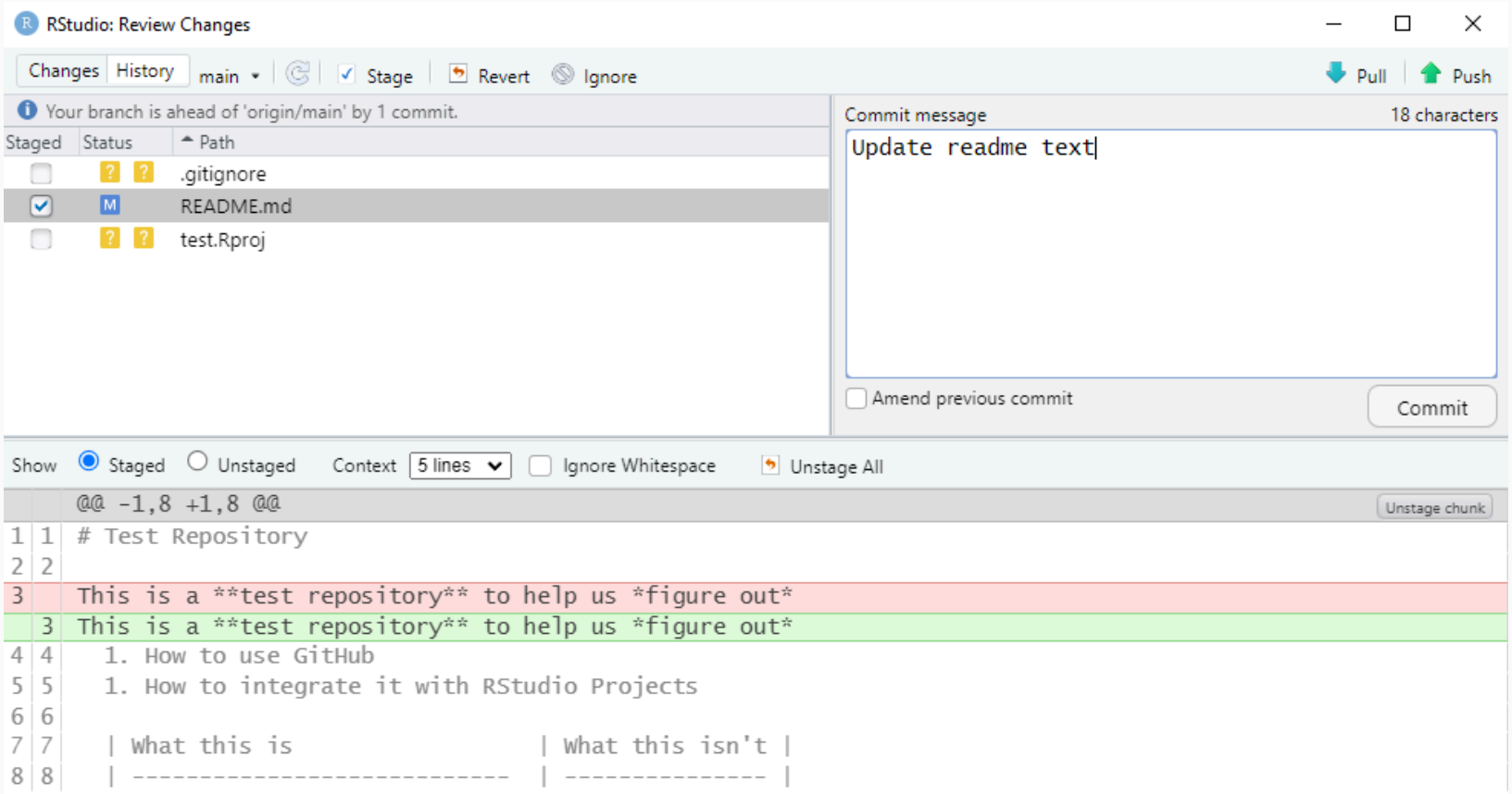
- Add some text and save the README.
- Do you see any changes in the **Git** panel? Good. (Raise your hand if not.)

[4] They're important, but not for the purposes of this section.

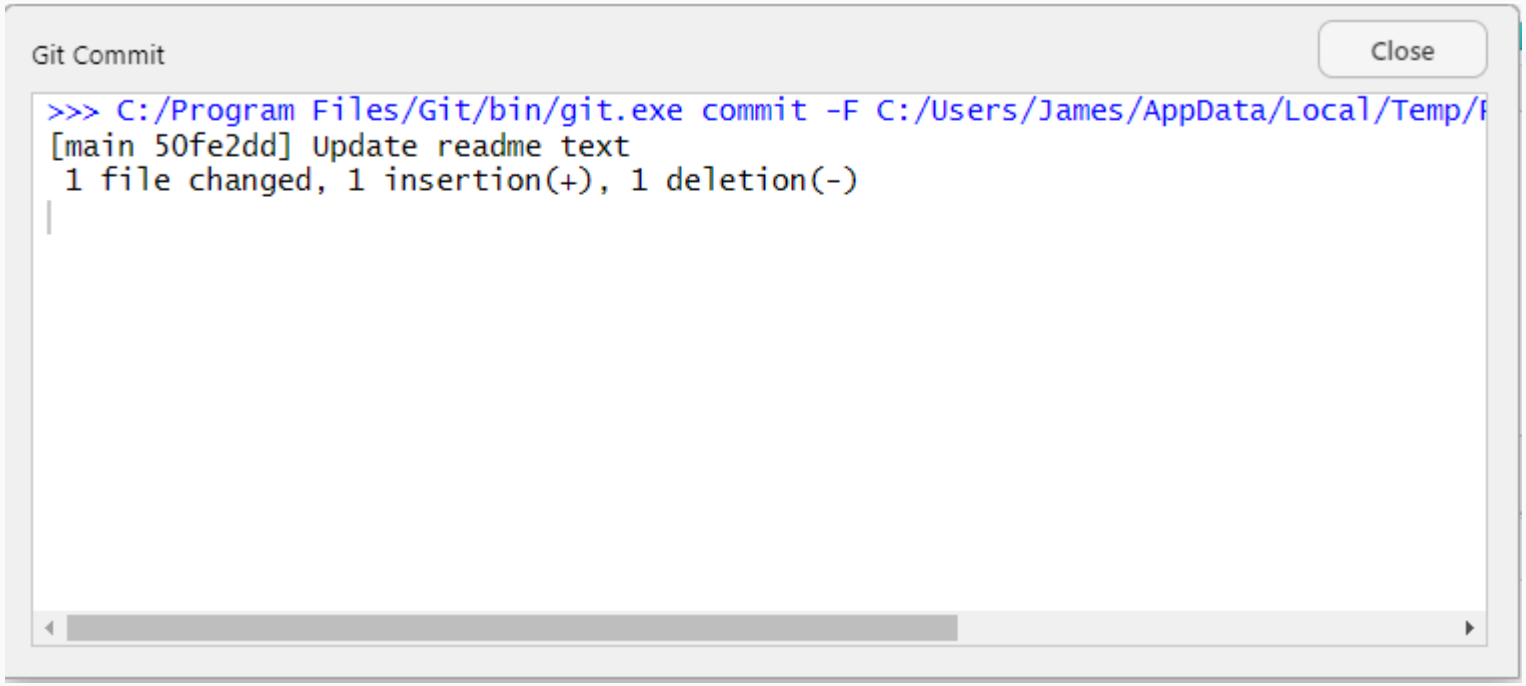
Stage and Commit



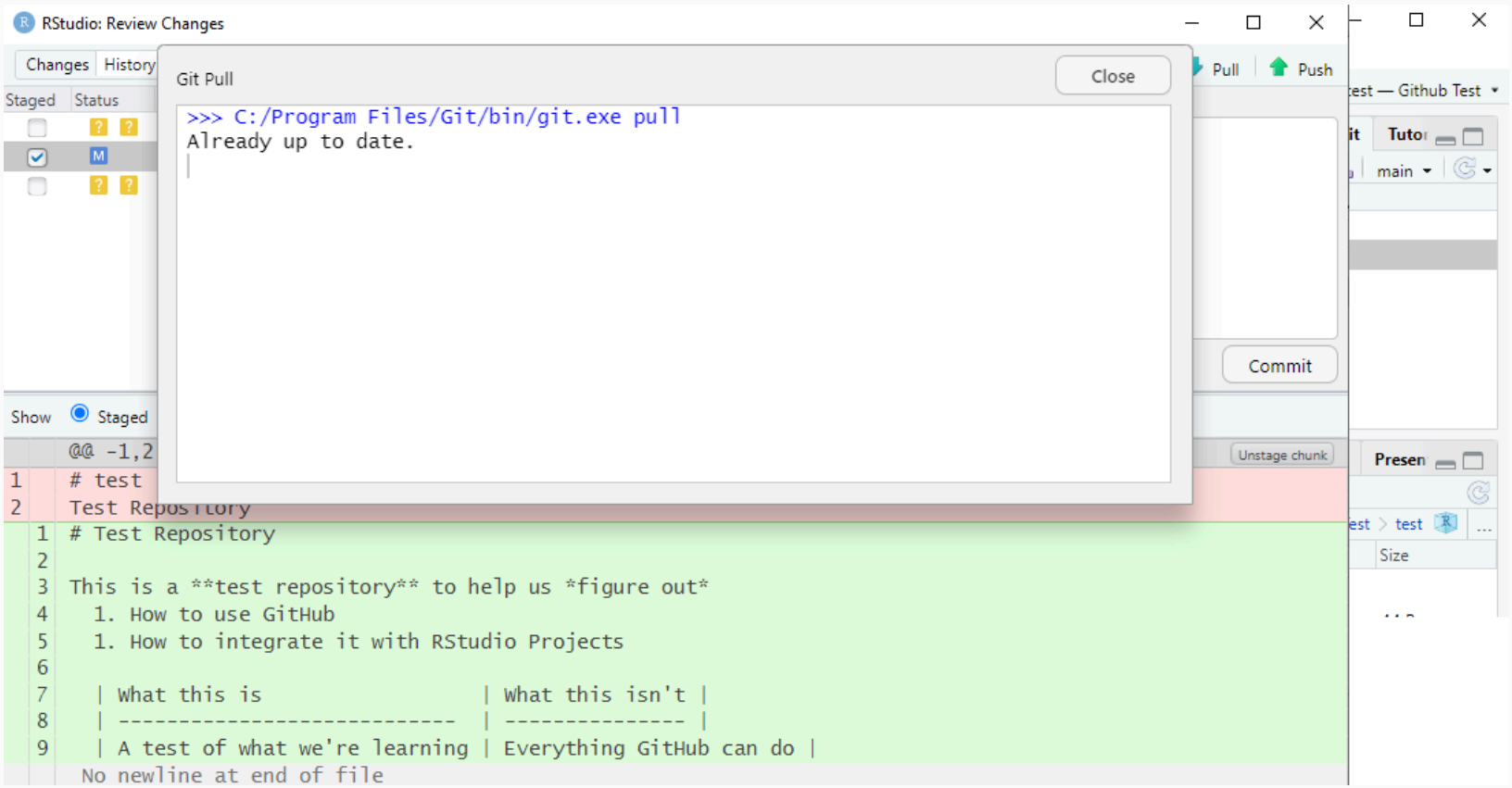
Stage and Commit



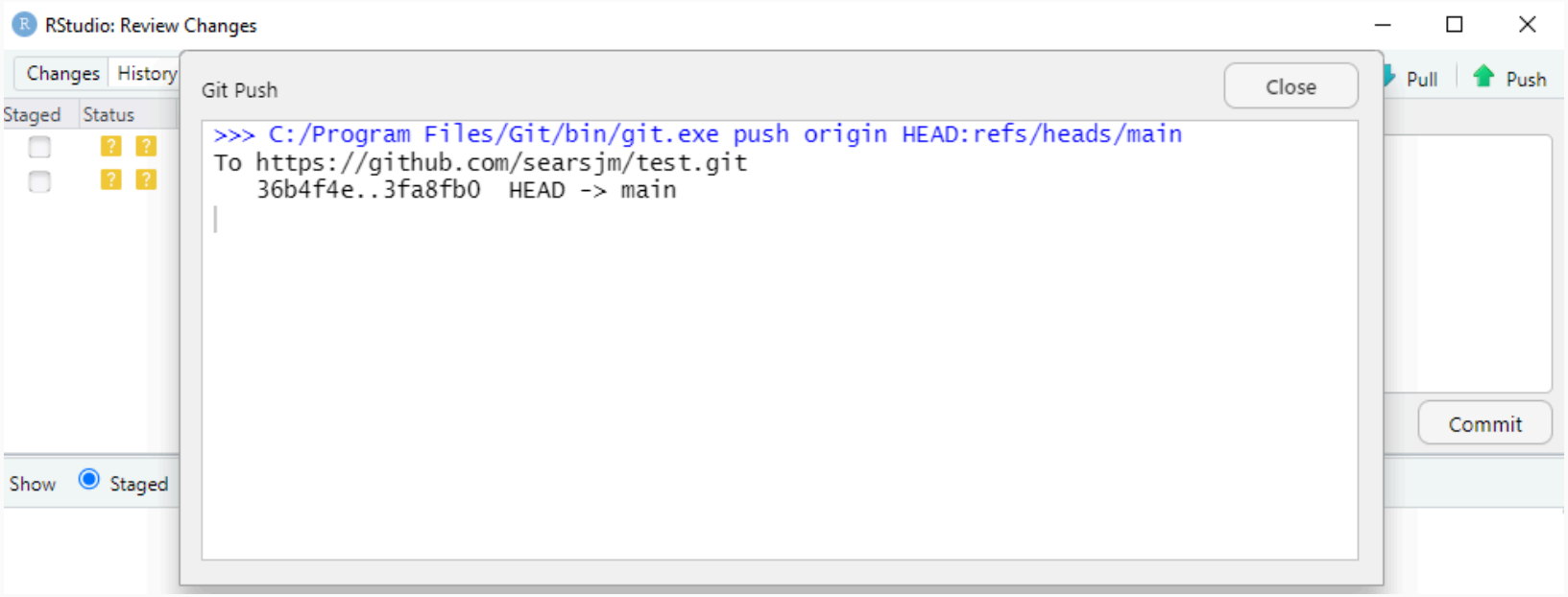
Stage and Commit



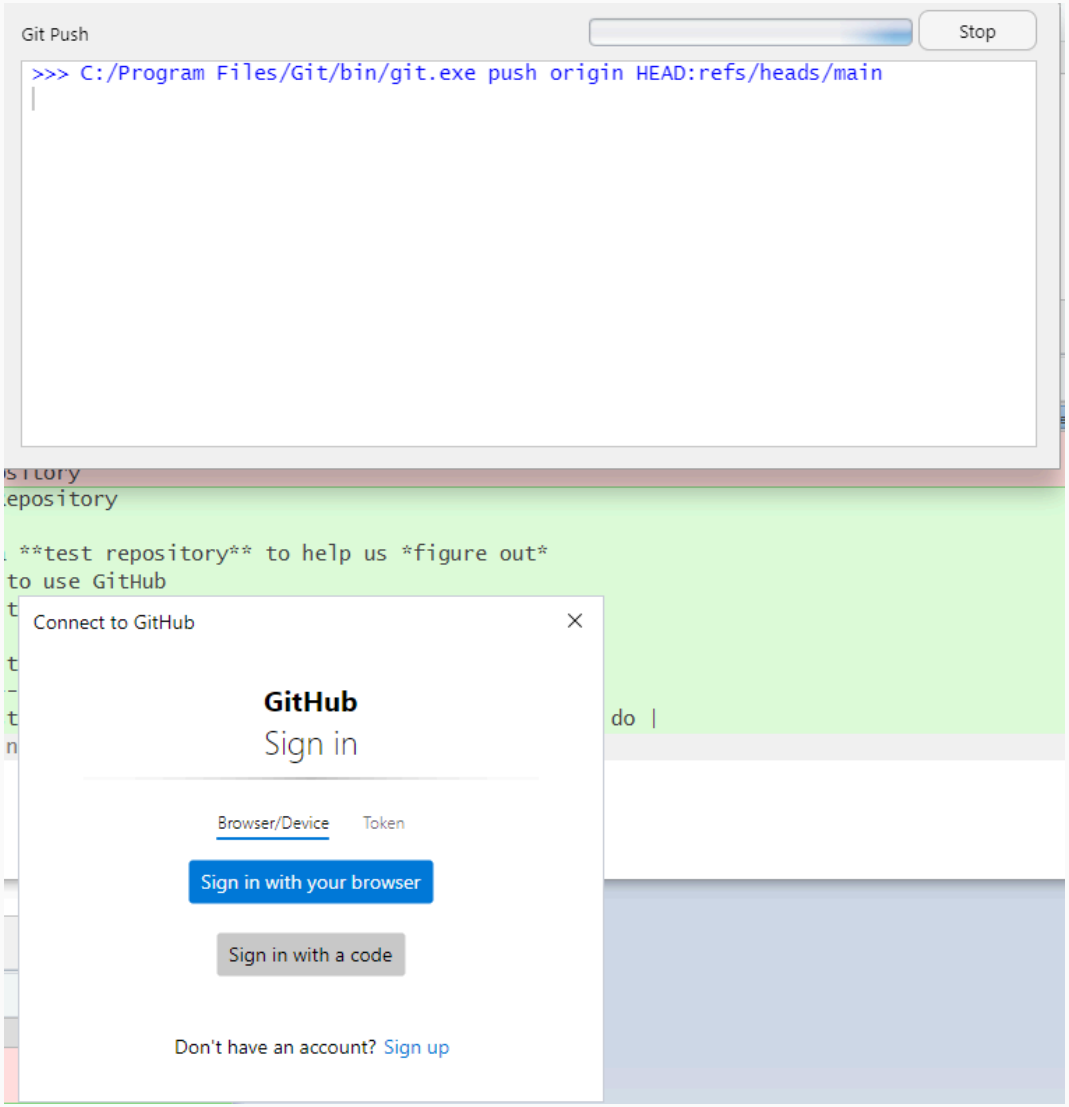
Pull


RStudio: Review Changes
Changes
History
Staged
Status
Git Pull
Close
>>> C:/Program Files/Git/bin/git.exe pull
Already up to date.
Pull
Push
Commit
Show
Staged
Unstage chunk
@@ -1,2
1 # test
2 Test Repository
1 # Test Repository
2
3 This is a **test repository** to help us *figure out*
4 1. How to use GitHub
5 1. How to integrate it with RStudio Projects
6
7 | What this is | What this isn't |
8 | ---------- | ---------- |
9 | A test of what we're learning | Everything GitHub can do |
No newline at end of file
main
Size

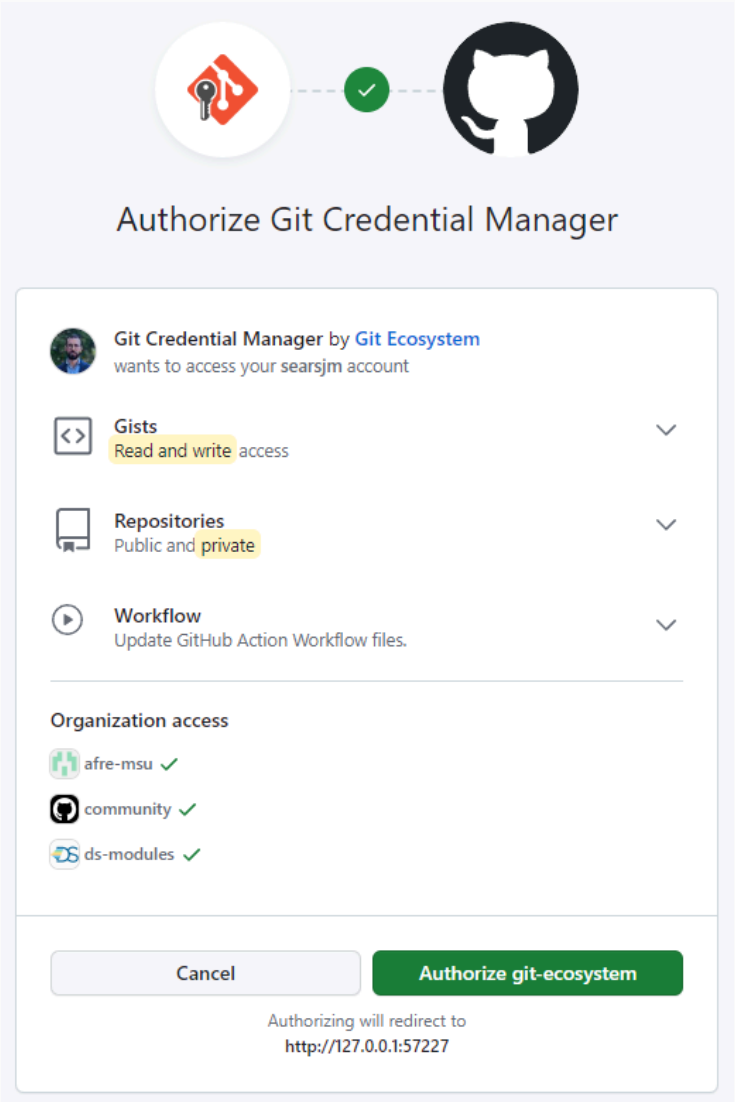
Push


RStudio: Review Changes
Changes
History
Staged
Status
Pull
Push
Commit
Show
Staged
Git Push
Close
>>> C:/Program Files/Git/bin/git.exe push origin HEAD:refs/heads/main
To https://github.com/searsjm/test.git
36b4f4e..3fa8fb0 HEAD -> main

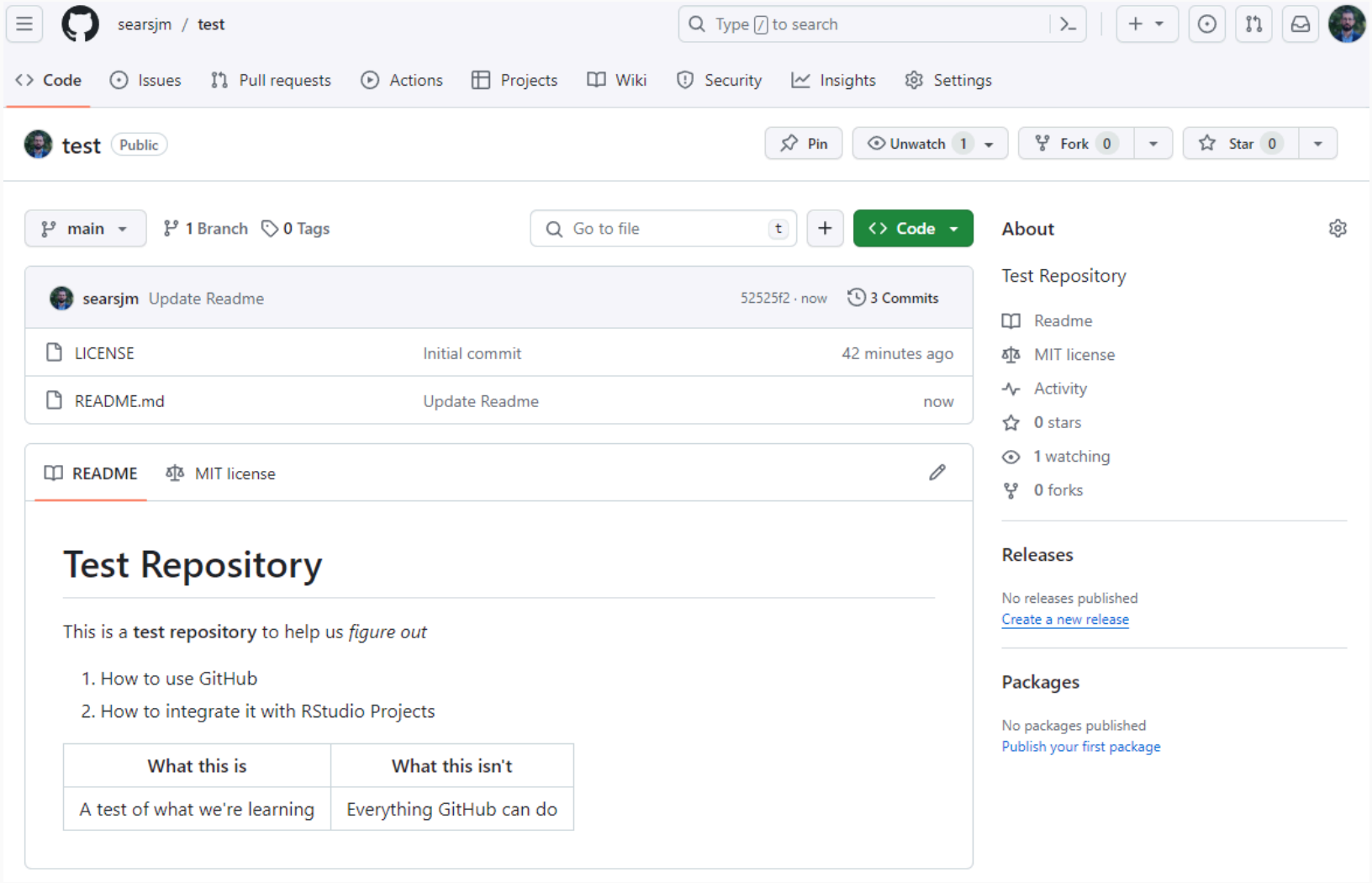
Sign RStudio into Github



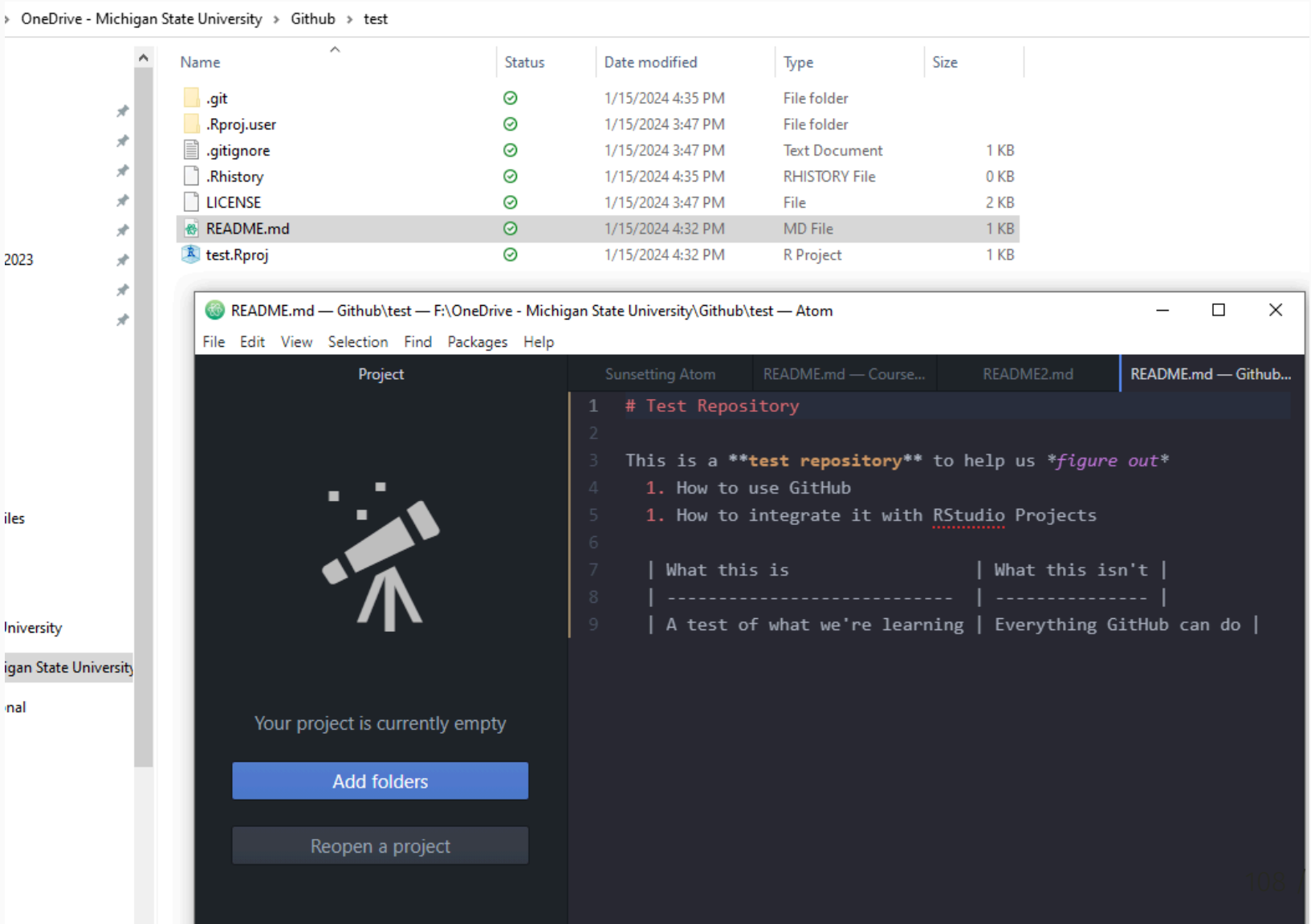
Sign RStudio into Github



Changes Now Visible on Github



Changes Now Visible Locally Too



Recap

Here's a step-by-step summary of what we just did.

- Made same changes to a file and saved them locally.
- **Staged** these local changes.
- **Committed** these local changes to our Git history with a helpful message.
- **Pulled** from the GitHub repo just in case anyone else made changes too (not expected here, but good practice).
- **Pushed** our changes to the GitHub repo.

PS — You were likely challenged for your GitHub credentials at some point. Learn how to cache these **here**.

Why this Workflow: RStudio

RStudio Projects are great.

- They interact seamlessly with Git(Hub), as we've just seen.
- They also solve absolute vs. relative path problems, since the .Rproj file acts as an anchor point for all other files in the repo.[5]

Note however that R Markdown files do the same thing re: paths *and* extend functionality to incorporating text, images, latex, and html.

[5] Calling files from their full `YourComputer/YourName/Documents/Special-Subfolder/etc` paths in your scripts is the enemy of reproducibility!

Troubleshooting Git Credential Issues in RStudio

Troubleshooting Git Credential Issues

One reason I'm not focusing on the GitHub + RStudio integration is **credential errors.**

For instance, do you get a password authentication error when trying to push to GitHub from RStudio?

Troubleshooting Steps

To begin, install the `usethis` package:

```
pacman:p_load(usethis)

# or:
# install.packages("usethis")
# followed by
# library(usethis)
```

Troubleshooting Steps

Next, run the following:

```
library("usethis")
git_sitrep()
```

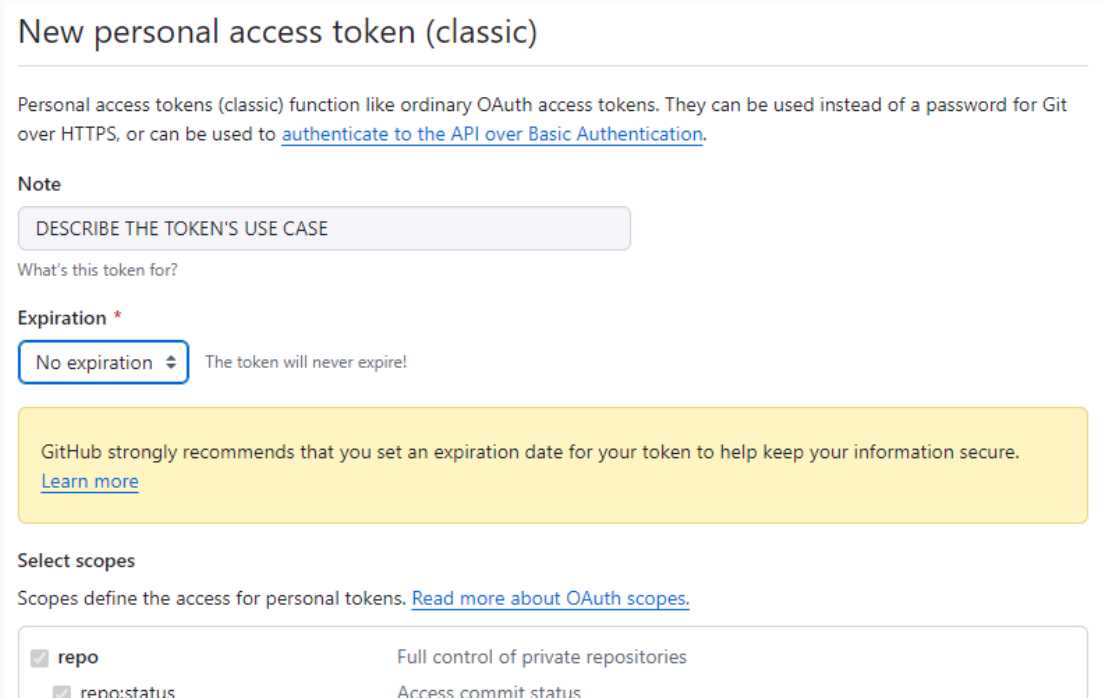
View the output: do you see any Xs or lines saying "lacks recommended scopes", "error", or "can't retrieve"?

Create a Personal Access Token (PAT)

To fix these errors, we'll create a **Personal Access Token (PAT)** on GitHub.

The below code will open a browser window - follow the steps to create a PAT

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



Add a PAT to RStudio

Back in RStudio, run

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

Follow the prompts to add/replace existing credentials, pasting in the PAT.

Check PAT

Once added, run `git_sitrep` one last time to verify.

If all looks good, try pushing again!

```
> git_sitrep()
Git config (global)
• Name: 'James Sears'
• Email: 'james.sears.m@gmail.com'
• Global (user-level) gitignore file: <unset>
• Vaccinated: FALSE
i See `?git_vaccinate` to learn more
• Default Git protocol: 'https'
• Default initial branch name: 'master'
GitHub
• Default GitHub host: 'https://github.com'
• Personal access token for 'https://github.com': '<discovered>'
• GitHub user: 'searsjm'
• Token scopes: 'gist, repo, user, workflow'
• Email(s): 'james.sears.m@gmail.com (primary)', 'searsja1@msu.edu'
Git repo for current project
i No active usethis project
>
```

Table of Contents