

Project Management and Reproducibility in RStudio: Part 1

Doug Joubert

2022-09-07

Table of contents

Configuration	2
Resources	2
Introduction	3
Scientific Reproducibility	3
Nature Survey on Reproducibility	3
Five Compelling Reasons to Practice These Principles	4
Helps to avoid data loss and disaster	5
Makes it easier to write papers	6
Helps reviewers see it your way	6
Enables continuity of your work	6
Helps to build your reputation	6
Using Version Control	6
Git Versus GitHub	6
Why Should I Use Version Control for My Project?	7
About Authentication to GitHub	7
Authenticating in your browser	7
Personal access token for HTTPS	8
Creating your PAT and Connecting it to Git	8
Creating a New Repository in GitHub	12
RStudio	14
Advantages of Using RStudio	14
Setting up RStudio	14
RStudio Projects	15
Creating a New Project in RStudio	16
Good Practices for Managing Projects in RStudio	18
Project Organization Issues	18
File/folder disorganization	18

Storage and sharing issues.....	18
Losing track of project status.....	19
Discussion Points	19
Practice Good File Organization	19
Practices for Naming Files.....	19
Why This is important.....	20
Machine-readable Files.....	20
Human-readable	20
Ordering Files	20
Using Version Control in RStudio	21
Workflows in Git.....	22
Commits and diffs	23
Tips for working with Git.....	25
Pushing our Edits to Git.....	25
Licenses	28
References.....	28

Configuration

This class features exercises that will help you learn by doing. To prepare, please install the following on your machine:

- If you haven't already, please [sign-up](#) for a free GitHub account.
- Install the latest version of [R](#), the current version is 4.2.0
- Install latest version of [RStudio](#), v2022.07.0-548 or later

Resources

Introduction to Reproducible Publications with RStudio: <https://carpentries-incubator.github.io/Reproducible-Publications-with-RStudio/>

R for Reproducible Scientific Analysis: <https://swcarpentry.github.io/r-novice-gapminder/>

Reproducible Research Data and Project Management in R:
<https://annakrystalli.me/rrresearchACCE20/>

Using Projects: <https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects>

Introduction

This one-hour class focuses on data and project management using R and RStudio. RStudio makes it possible to work on a complete research project in a more efficient, integrated, and organized manner. RStudio also connects with Git and GitHub and learners will have a chance to experiment with this integration and understand its advantages for collaboration and version control. This intermediate-level course is designed to be relevant to students from different disciplines. Some familiarity or experience in R and RStudio is recommended but not required. Students are encouraged to install [R](#) and [RStudio](#) before the webinar so that they can follow along with the instructor. Attendees will need to download the class data before the webinar.

Upon completion of this class students should be able to:

- Define scientific reproducibility
- Discuss best practices for organizing data in an RStudio project
- Discuss the importance of using a data dictionary and read me files
- Ensure that their data is machine readable
- Create a new R project using a GitHub repository
- Distinguish between pulling and pushing data from a repository

Scientific Reproducibility

According to the [U.S. National Science Foundation \(NSF\)](#) subcommittee on replicability in science (Bollen et al., 2015):

Science should routinely evaluate the reproducibility of findings that enjoy a prominent role in the published literature. To make reproduction possible, efficient, and informative, researchers should sufficiently document the details of the procedures used to collect data, to convert observations into analyzable data, and to perform data analysis.

Reproducibility refers to the ability of a researcher to duplicate the results of a prior study using the same materials as were used by the original investigator. Reproducibility is a minimum necessary condition for a finding to be considered rigorous, believable, and informative.

Nature Survey on Reproducibility

A 2016 survey in Nature revealed that not being able to reproduce experiments is a problem across all domains of science [Figure 1]:

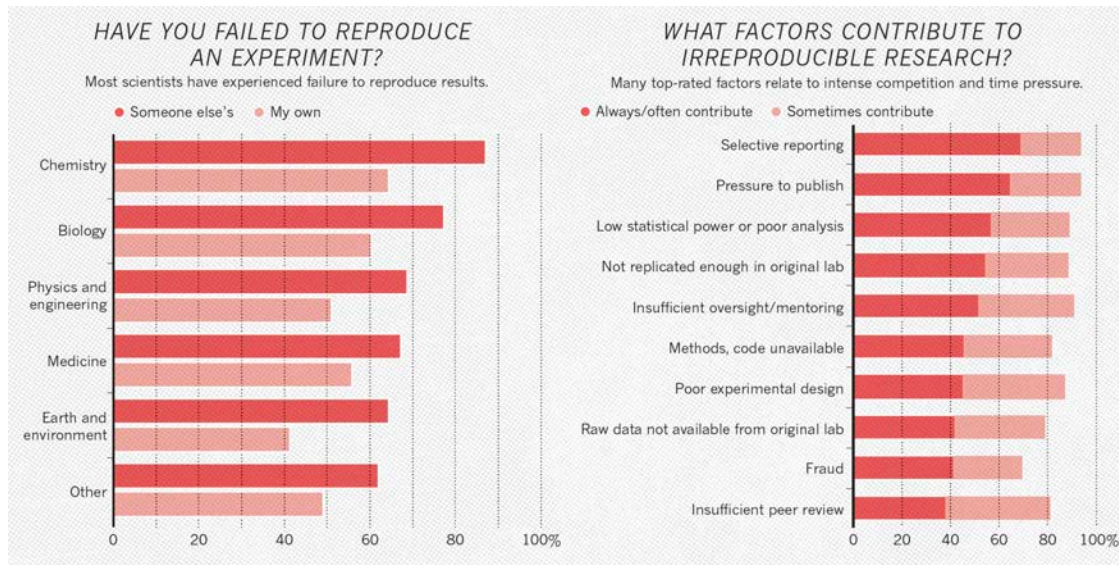


Figure 1: Reproducibility by discipline and problem identified (Baker, 2016).

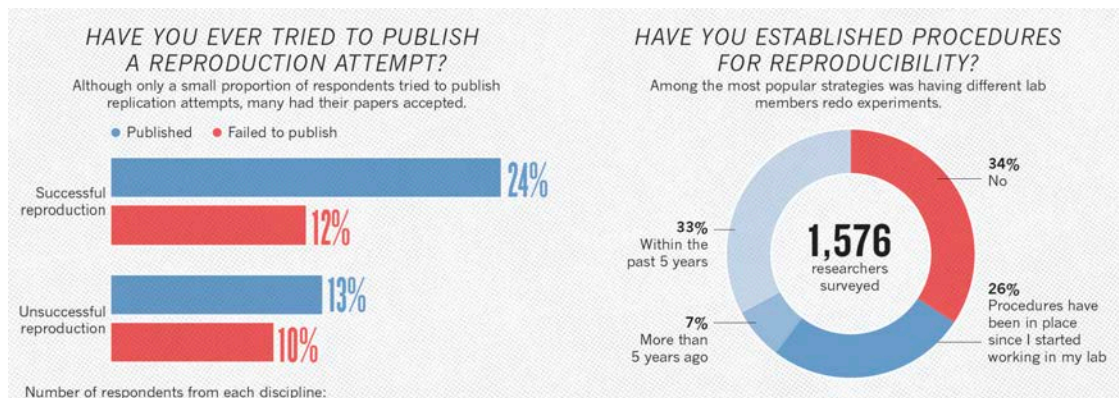
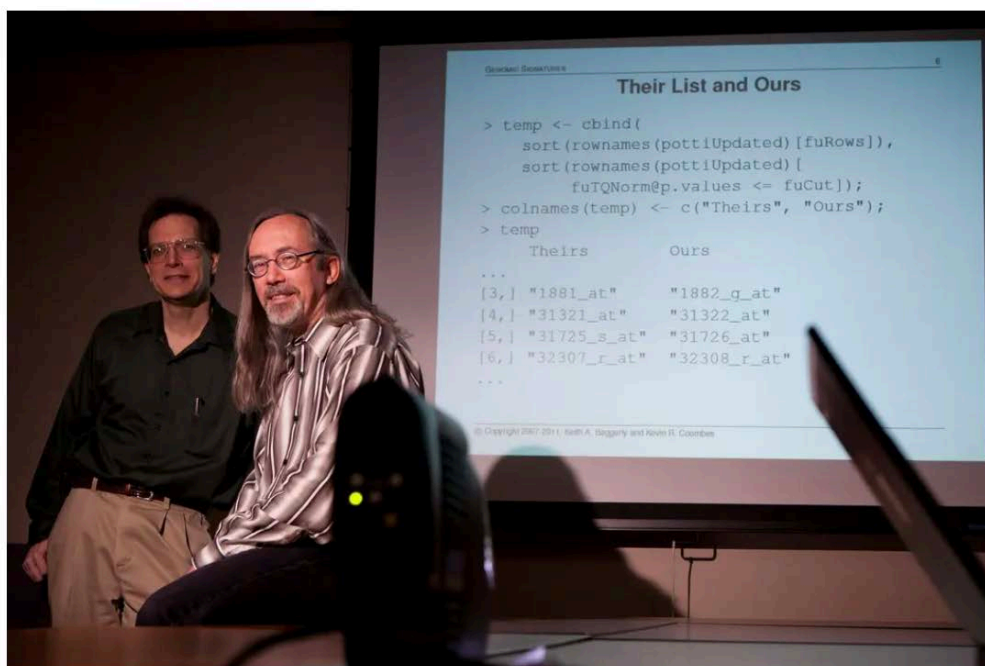
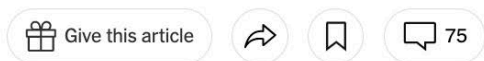


Figure 2: Attempts to publish reproducible research (Baker, 2016).

Five Compelling Reasons to Practice These Principles

If contributing to science and other researchers seems not to be compelling enough, here are 5 selfish reasons to follow reproducibility best practices (Markowetz, 2015).

How Bright Promise in Cancer Testing Fell Apart



Keith Baggerly, left, and Kevin Coombes, statisticians at M. D. Anderson Cancer Center, found flaws in research on tumors. Michael Stravato for The New York Times

Figure 3: Two statisticians from MD Anderson found with research from Duke University.

The issues discovered by Baggerly and Coombes could have easily been spotted by any co-author before submitting the paper:

- Data sets are not huge and can easily be spot-checked on a standard laptop
- You do not have to be a statistics wizard to realize that patient numbers differ, labels got swapped or samples appear multiple times with conflicting annotations in the same data set.

Why did no one notice these issues before it was too late? Because the data and analysis were not transparent and required forensic bioinformatics to untangle.

Makes it easier to write papers

Transparency in your analysis makes writing papers much easier. For example, in a dynamic document (R-markdown) so that the results automatically update when the data are changed.

Helps reviewers see it your way

What do you think is a major complaint of authors when going through peer review and addressing comments? “Reviewers didn’t even read the paper and had no idea what we were really doing. So, practice making the data and well-documented code easily accessible to the reviewers.

Enables continuity of your work

Can you think of any examples where “not” employing this principle can lead to problems?

- “I am so busy, I can’t remember all the details of all my projects”
- “I did this analysis 6 months ago. Of course I can’t remember all the details after such a long time”
- “My principal investigator (PI) said I should continue the project of a previous postdoc, but that postdoc is long gone and hasn’t saved any scripts or data”.

Helps to build your reputation

For several papers, we have made our data, code, and analyses available as an Experiment Package on Bioconductor. When I came up for tenure, I cited all these packages as research output of my lab. Generally, making your analyses available in this way will help you to build a reputation for being an honest and careful researcher.

Using Version Control

Using version control has the power to revolutionize how you work in R. Being able to have a record of all the changes you’ve ever made to your code both locally and on a remote website is powerful. But getting everything set up can be a challenge. In this section I will walk through everything you need to do to use Git and GitHub alongside RStudio.

Git Versus GitHub

1. [Git] (<https://en.wikipedia.org/wiki/Git> “Opens in a new window”) is open source software for version control. Using Git, you can do things like see all previous versions of code you’ve ever created in a project.
2. [GitHub](#) is the most popular service for collaborating on code using Git
3. [GitLab](#) and [BitBucket](#) are two other options

It is possible to use Git without using GitHub, though most people combine the two.

Why Should I Use Version Control for My Project?

There are many reasons you would want to adopt a Git/GitHub workflow:

1. **Using Git and GitHub serves as a backup.** Because GitHub has a copy of all the code you have locally, if anything were to happen to your computer, you'd still have access to your code.
2. **Using Git and GitHub allows you to use version control.** Ever had documents called report-final.pdf, report-final-v2.pdf, and report-final-v3.pdf? Of course, you have. Instead of making copies of files over fear of losing work, version control allows you to see what you did in the past, all while keeping single versions of documents.
3. **Using Git and GitHub makes it possible to work on the same project at the same time as collaborators.** For many using Git/GitHub is better than collaborating using Dropbox, Google Drive, or OneDrive. Git and GitHub have built-in tools that enable simultaneous asynchronous work, a major benefit for those working in teams.

You might want to check out this 2016 [talk](#) by Alice Bartlett of the Financial Times, on Git and GitHub. Another great resources (one that I used for this section) can be found on [Happy Git and GitHub for the useR](#)

About Authentication to GitHub

When we interact with a remote Git server, such as GitHub, we must include credentials in the request. This proves we are a specific GitHub user, who's allowed to do whatever we're asking to do.

You can access your resources in GitHub in a variety of ways: in the browser, via GitHub Desktop or another desktop application, with the API, or via the command line. Each way of accessing GitHub supports different modes of authentication.

- Username and password with two-factor authentication
- Personal access token
- HTTPS
- SSH key

Authenticating in your browser

You will authenticate using your GitHub.com username and password. You may also use two-factor authentication and SAML single sign-on, which can be required by organization and enterprise owners. The help [documentation](#) for this topic on GitHub is an excellent resource.

Personal access token for HTTPS

With HTTPS, we will use a **personal access token (PAT)**. Head over to chapter 10 if you really want to set up SSH keys.

Let it be known that the password that you use to login to GitHub's website is NOT an acceptable credential when talking to GitHub as a Git server. This was possible in the past, but those days are over at GitHub. You can learn more in their blog post [Token authentication requirements for Git operations](#).

Here's the error you'll see if you try to do that now:

```
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/OWNER/REPO.git/'
```

Creating your PAT and Connecting it to Git

This is a very minimal account of getting and storing a PAT. This might be all you need when you're first getting yourself set up. You can always come back later and read other parts of this chapter.

1. Go to <https://github.com/settings/tokens> and click "Generate token".
2. Or you can do this from the R Terminal by using the [usethis](#) package. `usethis` is a workflow package: it automates repetitive tasks that arise during project setup and development, both for R packages and non-package projects.

```
# install.packages("usethis")
library(usethis)
create_github_token()

• Call `gitcreds::gitcreds_set()` to register this token in the local Git credential store
  It is also a great idea to store this token in any password-management software that you use
• Open URL 'https://github.com/settings/tokens/new?scopes=repo,user,gist,workflow&description=DESCRIBE THE TOKEN'S USE CASE'
```

3. When you run this section of code your default browser will open and you will be prompted to log into GitHub. After you log in you will automatically go to the create repository page [Figure 4].

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

RStudio Intro Classes

What's this token for?

Expiration *

30 days The token will expire on Thu, Aug 18 2022

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input checked="" type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry

Figure 4: Creating a new PAT on GitHub.

Look over the scopes; I highly recommend selecting “repo”, “user”, and “workflow”. Recommended scopes will be pre-selected if you used `create_github_token()`.

4. Click “Generate token”.
5. Copy the generated PAT to your clipboard. Or leave that browser window open and available for a little while, so you can come back to copy the PAT.
6. Provide this PAT next time a Git operation asks for your password.
7. You could even get out ahead of this and store the PAT explicitly right now. In R, call `gitcreds::gitcreds_set()` to get a prompt where you can paste your PAT. I am going to do this in Console to make it a bit easier.
8. Figure 5 is displaying the `gitcreds::gitcreds_set()` in the Console. Hit Enter/Return to run this command.

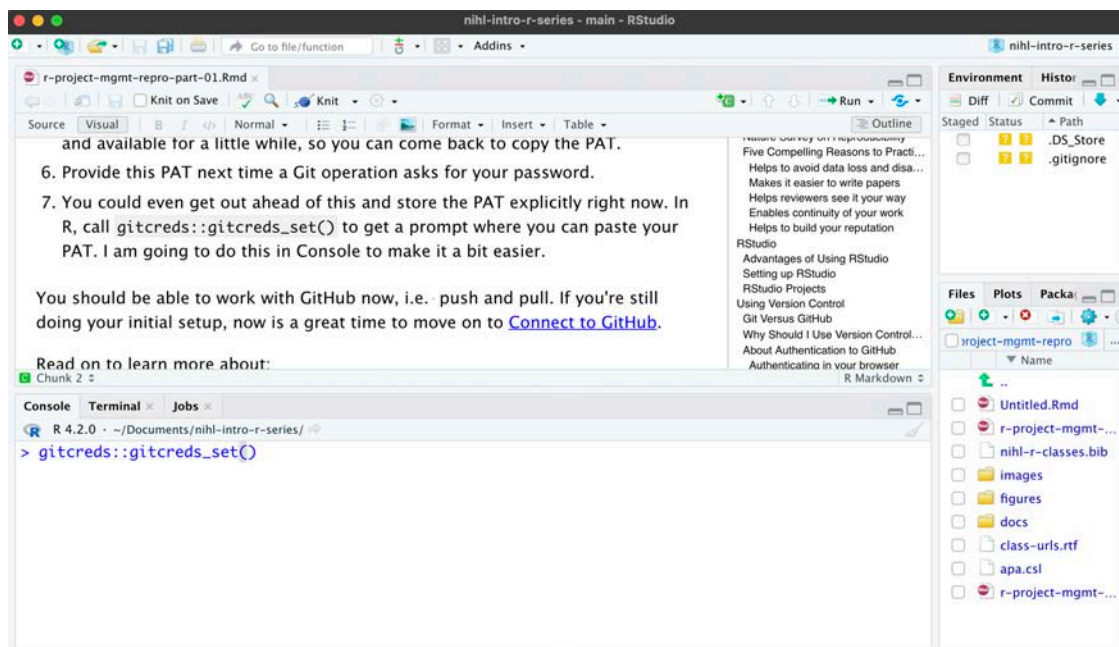


Figure 5: Initiating the `gitcreds_set()` command.

9. Select the most appropriate option. Since we are setting this up for the first time, we would select option 2 [Figure 6].

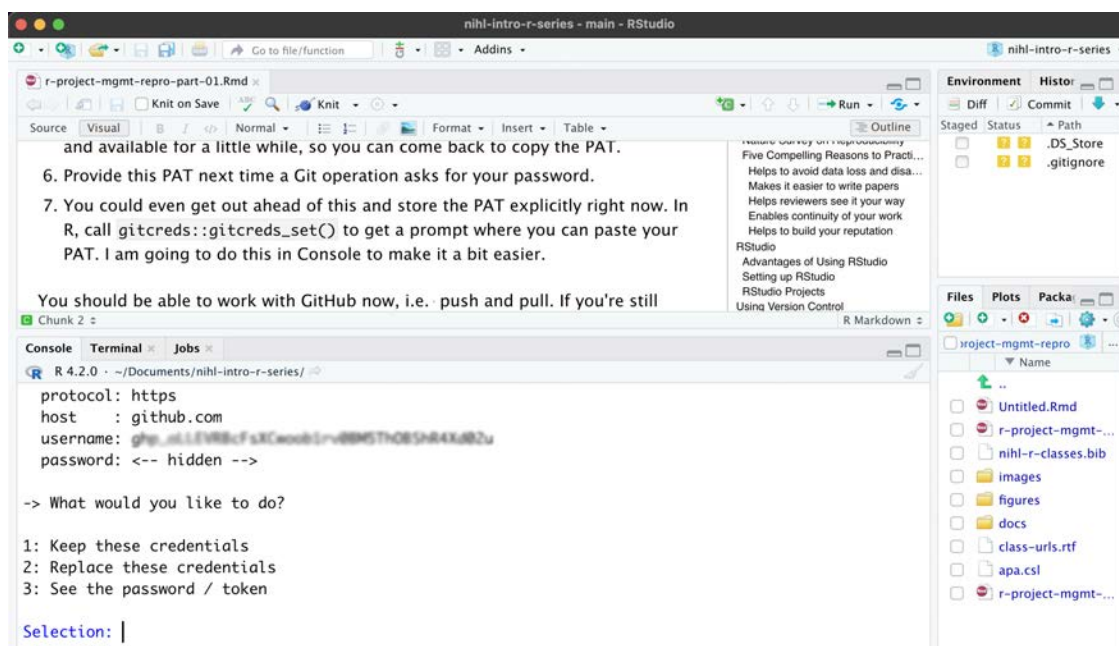


Figure 6: Replacing the current PAT.

10. Follow the prompts and enter the PAT that you just created.

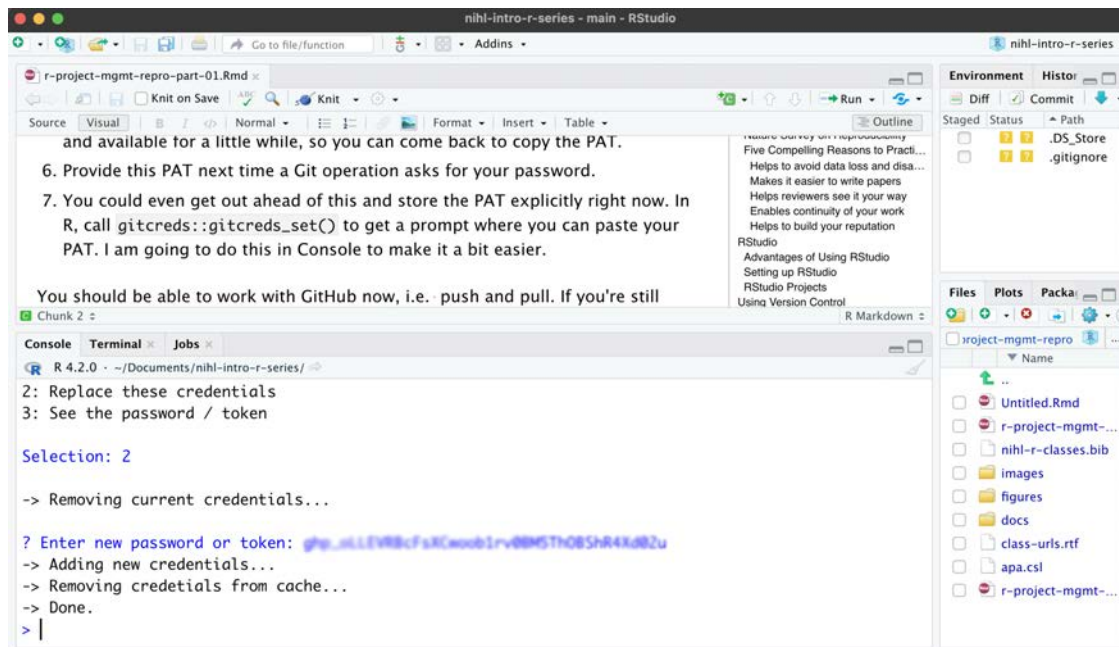


Figure 7: Final output from storing the PAT.

11. If you go back to GitHub you should see a list of all of the PATs that you have created and when they expire [Figure 8].

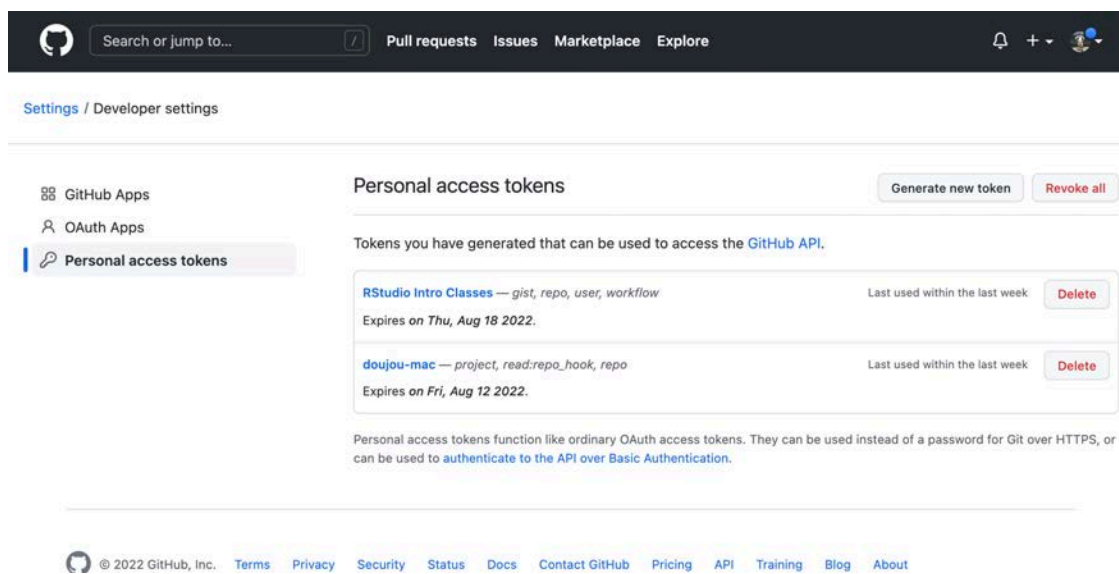


Figure 8: List of GitHub PATs.

You will also receive a message from GitHub (default email address) letting you know that a PAT has been created [Figure 9].

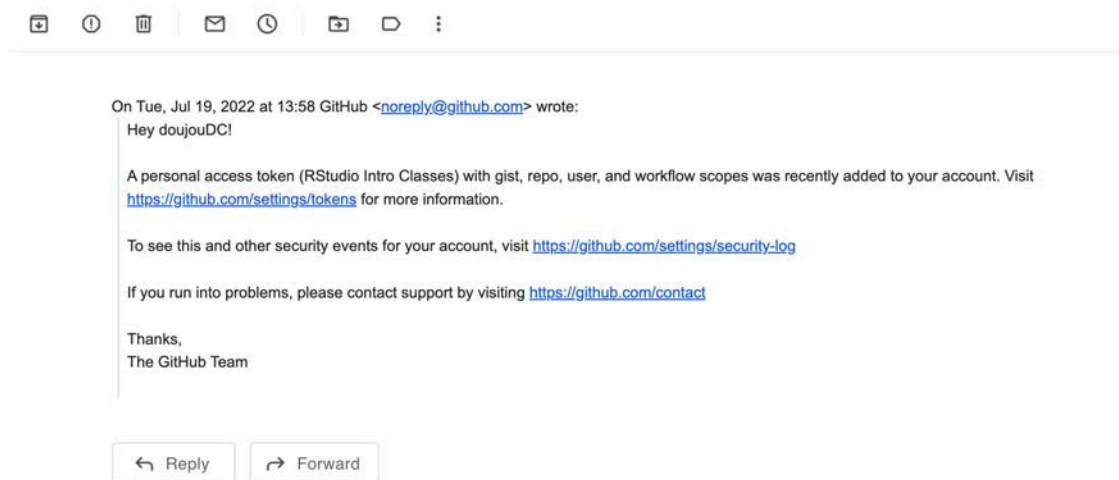


Figure 9: PAT alert email from GitHub.

You should be able to work with GitHub now, i.e. push and pull.

Remember, if you are already syncing on another computer, you will have to enter this new PAT before you push or pull.

Read on to learn more about:

- [How to decide between the HTTPS and SSH protocols](#)
- [PAT scopes, names, and expiration](#)
- [PAT storage](#)
- [Troubleshooting](#)

Creating a New Repository in GitHub

GitHub's collaborative approach to development depends on publishing commits from your local repository to GitHub for other people to view, fetch, and update.

[At this link](#), you may sign into your GitHub account or create one if you have not already. Figure 10 is displaying the repositories in my GitHub account.

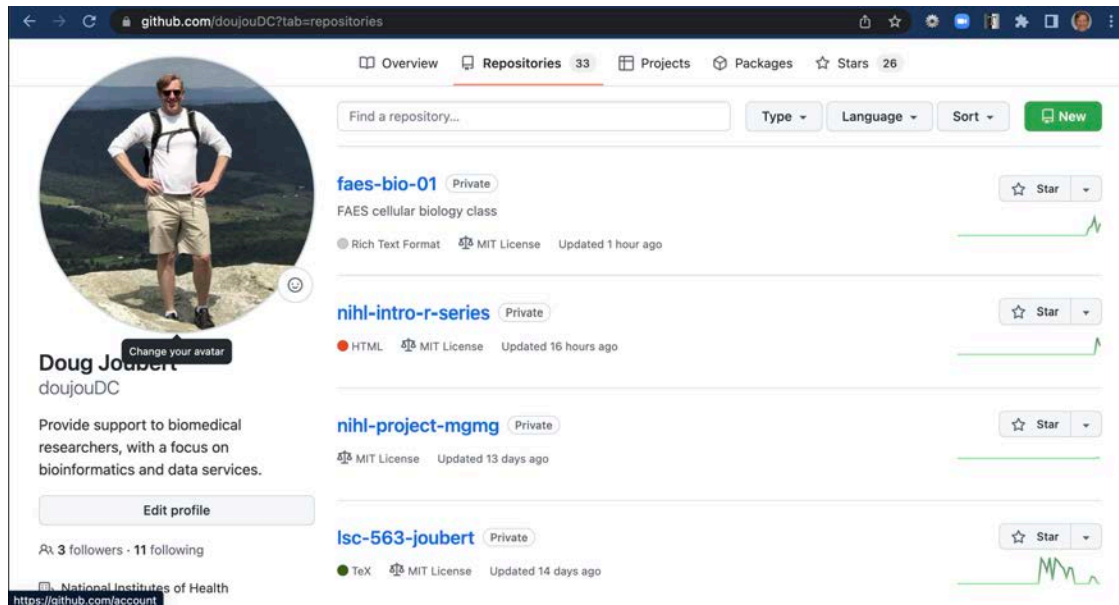


Figure 10: My repository homepage in GitHub.

I have already created a [repository](#) for this class. However, Figure 11 is displaying the Create a New Repository screen. [Figure 11].

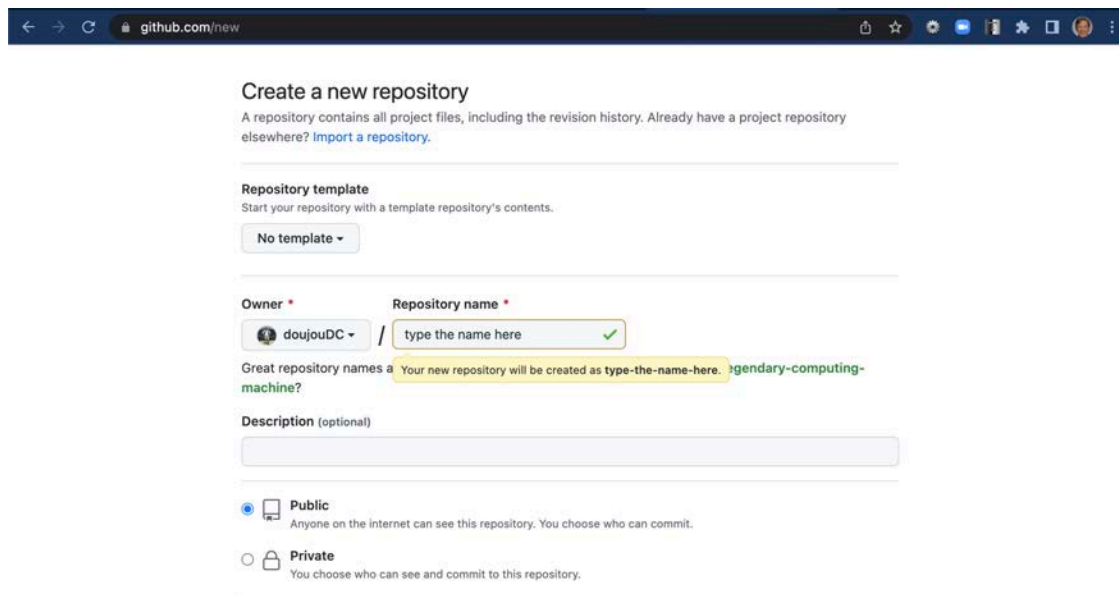


Figure 11: Creating a new repository on GitHub.

Once your repository is created, you will need to copy the repository URL before you create the project in RStudio. Figure 11 shows the 3 different methods for creating a URL: (1) https, (2) SSH, or (3) GitHub CLI. We have already created our PAT to use with the https option [Figure 12].

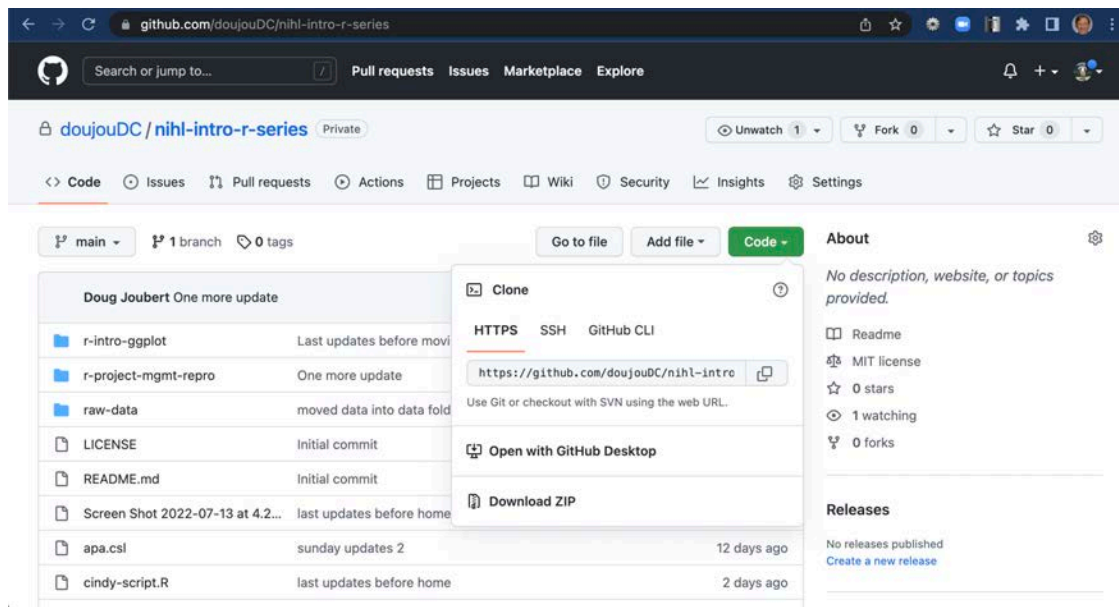


Figure 12: Creating a link to a GitHub repository.

RStudio

RStudio is an integrated development environment (IDE) for R and Python. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging, collaboration, and workspace management. It is a powerful tool which supports research by weaving the principles of reproducibility throughout the entire research lifecycle, from data gathering to the statistical analysis, presentation, and publication of results.

Advantages of Using RStudio

- It is free and open-source
- It is designed to make it easy to write and reuse code
- Makes it convenient to view and interact with the objects stored in your environment
- Makes it easy to set your working directory and access files on your computer
- Integrates with Collaboration and Publishing Tools
- Creates documents using R Markdown

Setting up RStudio

There are a number of RStudio settings that are recommended. These settings will force RStudio to start “fresh” each time you open RStudio.

This means that RStudio will not remember the code that you ran in your previous session. This forces you to document your work in your code and not in the RStudio workspace [which is temporary].

Figure 13 list the preferred RStudio settings. To change these options, go to **Tools>Global Options>General**.

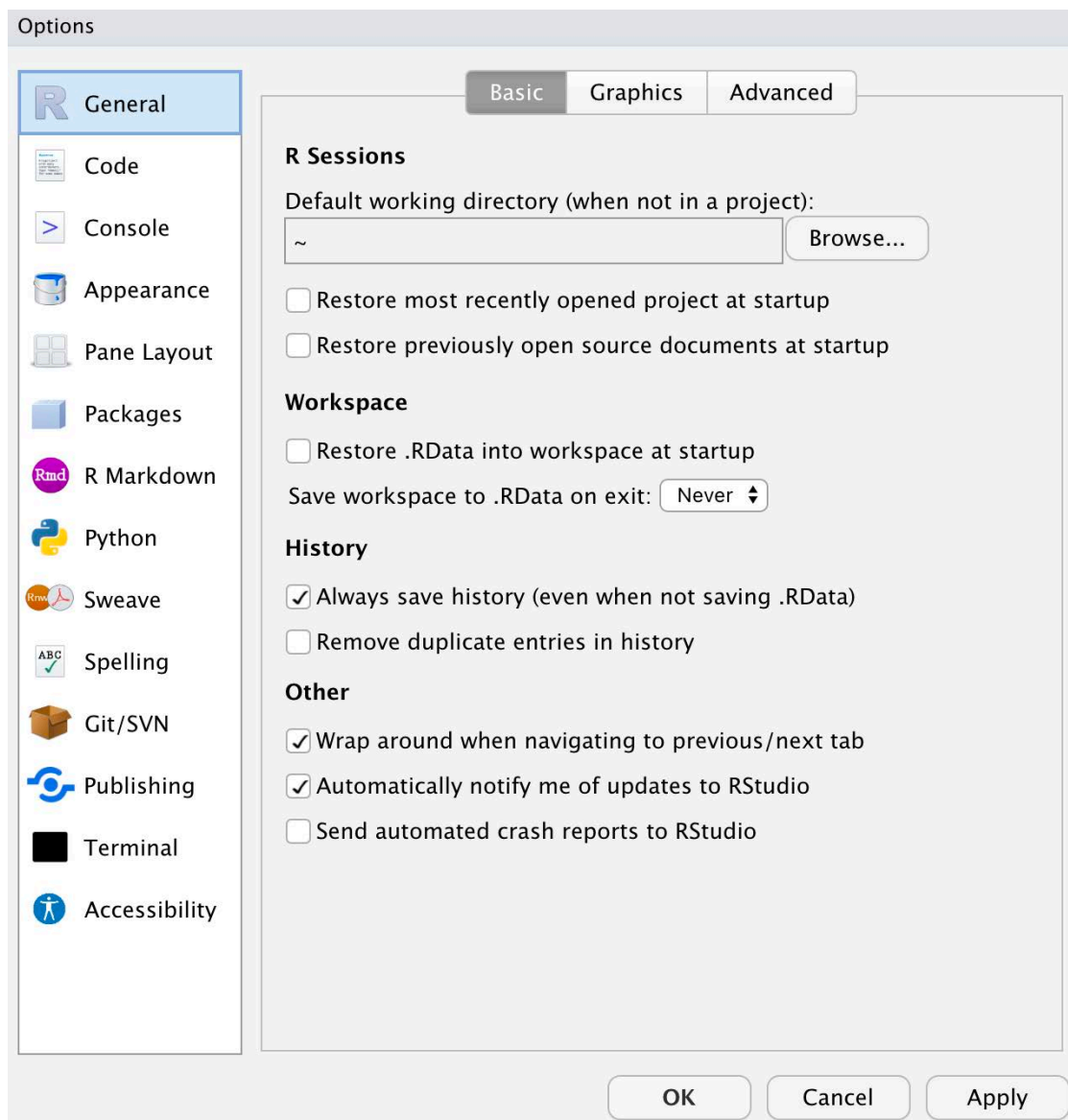


Figure 13: Preferred RStudio options.

You might want to also choose UTF-8 as the default text encoding. This setting is available from **Tools>Global Options>Code>Saving**.

RStudio Projects

An [RStudio Project](#) provides a set of tools that allow you to organize your scripts, files, and output:

1. Allows you to quickly navigate to your working directory
2. Is a great way to keep all your data organized

3. Can preserve custom settings and open files to make it easier to resume work after a break.

An RStudio project is different from an R session. Meaning that an R session is all the objects and work done in R. Sessions are usually kept in working memory, until you restart R.

There are three options for creating a project in RStudio:

1. New Directory - start a brand new R project (with the option of version control).
2. Existing Directory - add existing work to a R project (with the option of setting up version control).
3. Version Control Continue an existing R project that already uses version control (i.e. download from GitHub).

We will be using option 3, creating a repository on GitHub, and then importing this into RStudio.

Creating a New Project in RStudio

1. Open RStudio
2. Choose New Project from the Tools Menu.
3. There are three options when creating a new project: (1) Creating a project in a new directory, (2) Creating a project in an existing directory, and (3) Creating a project using version control [Figure 14].

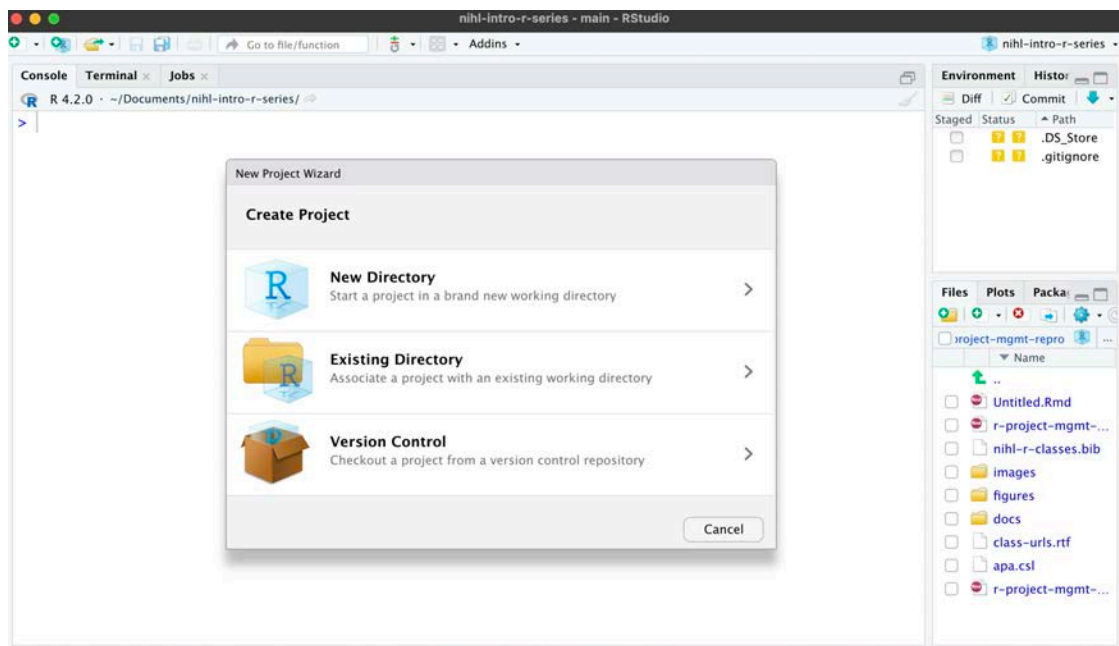


Figure 14: Creating a new project in RStudio.

4. Choose the Version Control option
5. Chose Git as the option, RStudio will automatically open the next screen [Figure 15]

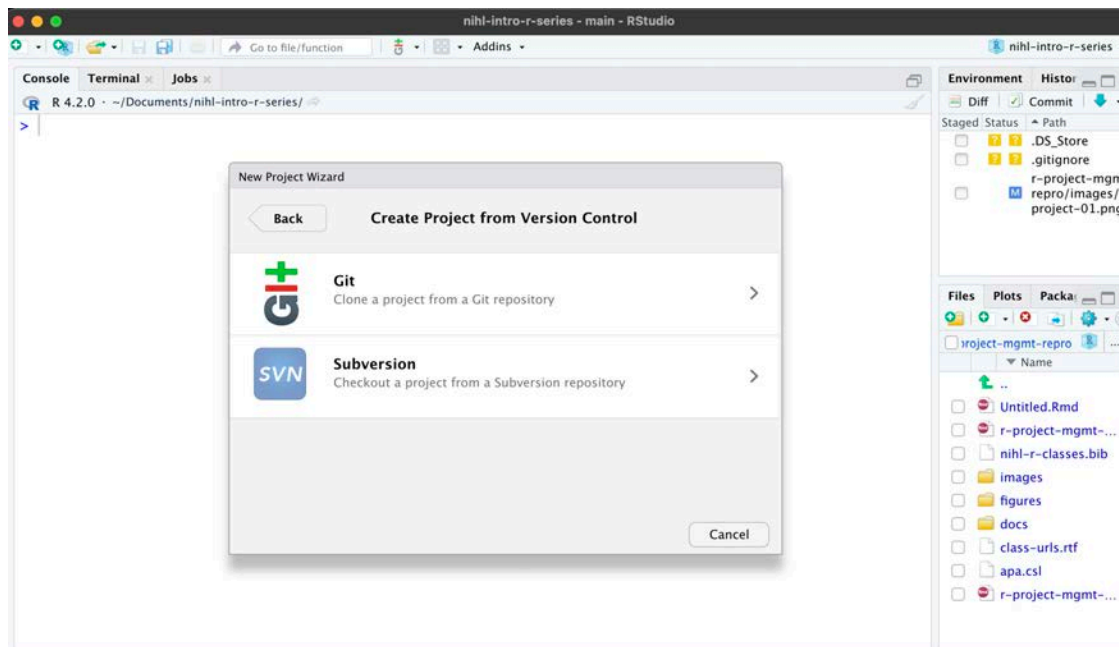


Figure 15: Choosing the version control option in RStudio.

6. This screen is where you enter the repository URL that you copied from GitHub [Figure 16].

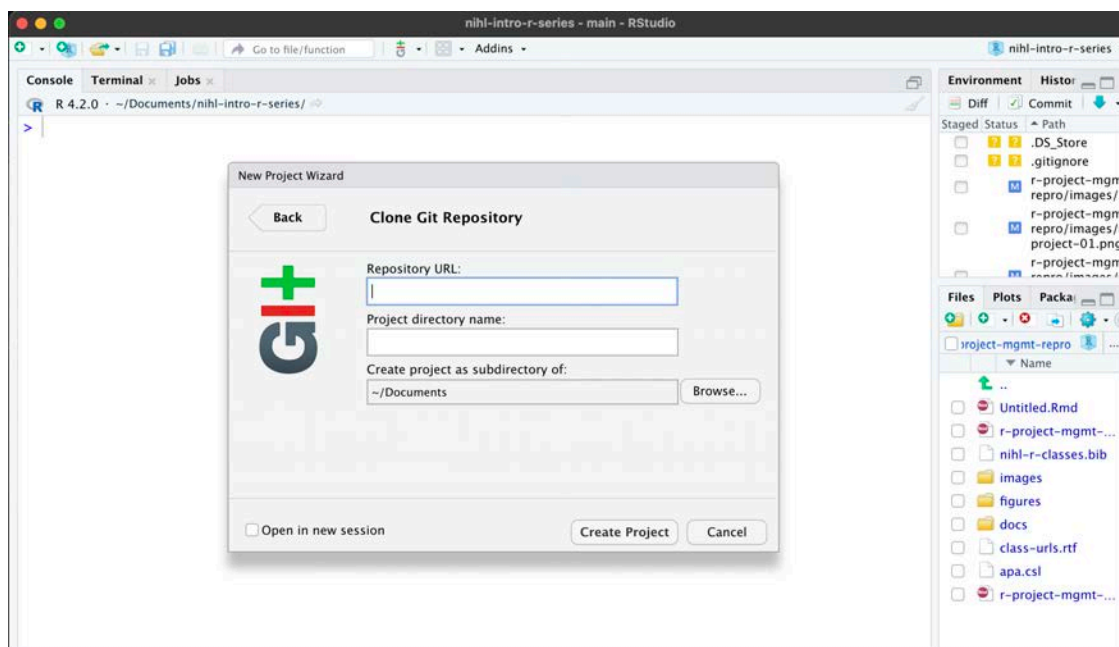


Figure 16: New project wizard - entering the repository URL.

7. Open up the repository that we created in GitHub and copy the https URL [Figure 17].

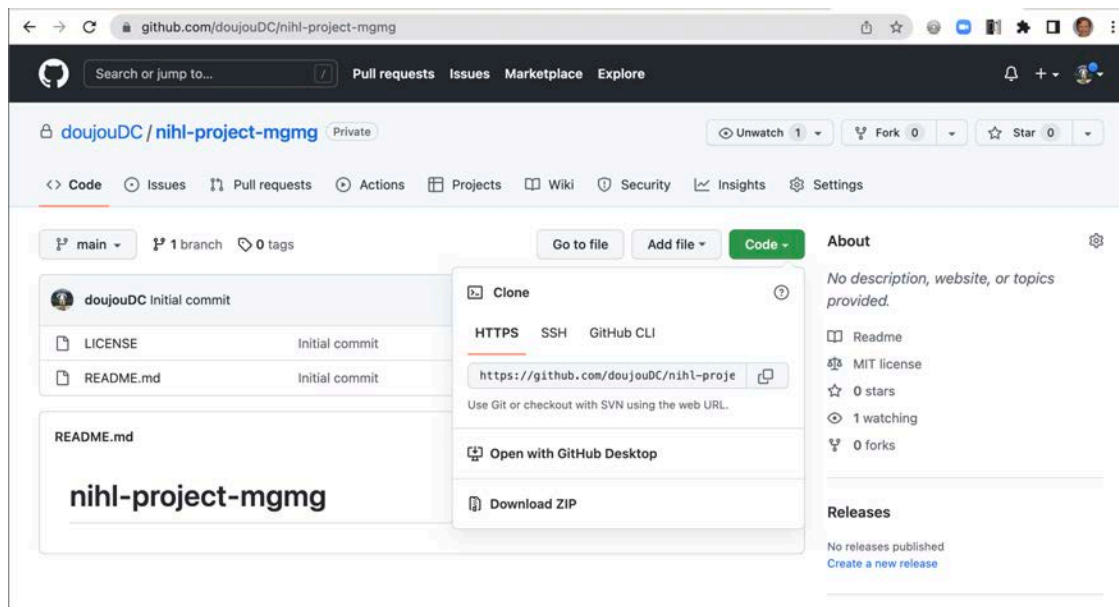


Figure 17: Copying a https repository URL from GitHub.

8. Paste the repository URL into the URL box. Once you do this, you will be prompted to choose a subdirectory for the new project. I usually created mine within my Documents directory.
9. Click **Create Project**. The screen will refresh as RStudio switches over to project view and downloads data from GitHub

Good Practices for Managing Projects in RStudio

Project Organization Issues

We often have stress points in our research that may become breaking points, especially when it comes to working with collaborators, returning to a project after a hiatus, or dealing with data and scripts. Let's discuss three of those common stress points:

File/folder disorganization

- You cannot find your files on your computer (or your cloud storage)
- Multiple versions of files with names such as "finaldraft_4.txt"
- Path issues when trying to run code
- Reviewers or colleagues cannot re-run your code/analyses

Storage and sharing issues

- Files are only saved to your computer and are vulnerable (or have already succumbed to computer/hard drive failure)
- When working with collaborators, they (or you) don't share the files needed
- Files are shared via email attachments

- Difficult to know if you have the latest version of documents

Losing track of project status

- You cannot remember where you are in a project after being away for an extended period (or what you worked on the previous day...no judgement)
- You aren't sure what you should be working on next
- You have various to-do notes spread across your office or home (or never write them down in the first place)

Discussion Points

To what extent do these stress points affect your research projects?

Are there additional issues that you've encountered that slow down or derail your work due to issues with project management?

Practice Good File Organization

Although there is no "best" way to lay out a project, there are some general principles to adhere to that will make project management easier.

[Good Enough Practices for Scientific Computing](#) gives the following recommendations for project organization (Wilson et al., 2017):

1. Each project in its own directory, which is named after the project
2. Text documents associated with the project in the doc directory
3. Raw data and metadata in the data directory (raw-data)
4. Files generated during cleanup and analysis in a results directory
5. Project's scripts and programs in the src directory
6. Programs brought in from elsewhere or compiled locally in the bin directory
7. Name all files to reflect their content or function

Additional files to include:

[README file](#), to communicate important information about your project [LICENSE file](#), so that others are free to use, change, and distribute the software [CITATION.cff file](#), to let others know how you would like them to cite your work

Additional recommendations for projects and creating folders is covered in the Introduction to R and RStudio class. Class handouts are available upon request.

Practices for Naming Files

Three principles for file names 1. Machine readable 2. Human readable 3. Plays well with default ordering

Why This is important

- Globbing: using wildcard characters to request or evaluate sets of files with the same partial names or sets of character
- Regular expression friendly
- “Findability”
- Global name changes

Machine-readable Files

- No spaces, unsupported punctuation, accented characters, or case-sensitive file names
- Deliberate use of delimiters (i.e. for splitting file names)
- Consistently use the same delimiters: `data-analyses-fig1.R` as an example

Human-readable

- Name contains brief description of content `anova-analyses-control.R` `marshall-data-raw-data` `marshall-data-filtered` `marshall-data-load-counts.r`

Ordering Files

With **chronological ordering**, file name starts with date:

`2022-06-26-BRAFWTNEGASSAY-Plasmid-Cellline-100-MutantFraction_HO1.csv`

`2022-06-26-BRAFWTNEGASSAY-Plasmid-Cellline-100-MutantFraction_HO2.csv`

`2022-06-26-BRAFWTNEGASSAY-Plasmid-Cellline-100-MutantFraction_HO3.csv`

`2022-06-26-BRAFWTNEGASSAY-Plasmid-Cellline-100-MutantFraction-platefile.csv`

`2022-02-26-BRAFWTNEGASSAY-FFPEDNA-CRC-1-41_AO1.csv`

`2022-02-26-BRAFWTNEGASSAY-FFPEDNA-CRC-1-41_AO2.csv`

`2022-02-26-BRAFWTNEGASSAY-FFPEDNA-CRC-1-41_AO3.csv`

`2022-02-26-BRAFWTNEGASSAY-FFPEDNA-CRC-1-41_AO4.csv`

Consider using [ISO 8601 date standard](#)

With **logical ordering**, the filename starts with a number or keyword/number combo.

`01-marshall-data.r` *see code directory*

`02-pre-dea-filtering.r` *see code directory*

`03-explore-dea-limma-voom.r`

`04-exploe-dea-results.r` helper

`01-load-counts.rmd`

helper02-load-exp-des.r

helper03-extract-and-tidy.r

As illustrated above, left-pad your numbers to facilitate sorting. If you do not do this, your data sorts like this...which is really sad

01-marshall-data.r *see code directory*

04-exploe-dea-results.r

2-explore-dea-limma-voom.r

3-helper-extract-and-tidy.r

helper01-load-counts.rmd

helper2-load-exp-des.r

pre-dea-filtering-1.r *see code directory*

Adapted from <https://datacarpentry.org/rr-organization1/01-file-naming/index.html>. For more tips on file naming, check: [The Dos and Don'ts of File Naming](#).

Using Version Control in RStudio

There are two places we can interact with Git in the RStudio interface.

1. Menu bar [Figure 18]

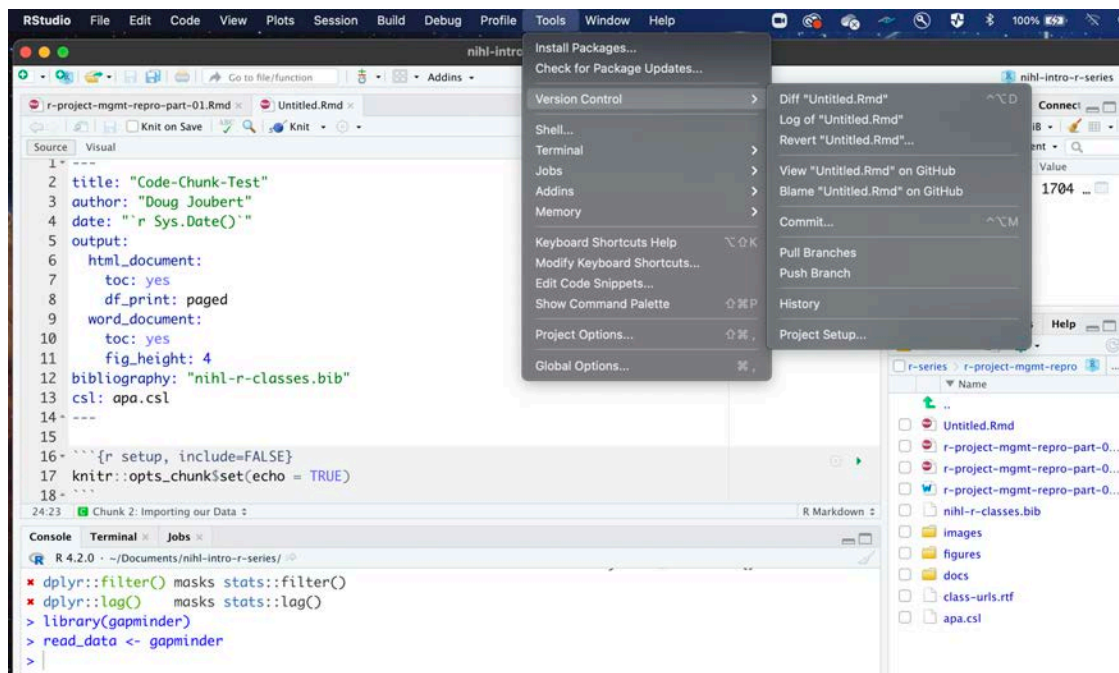


Figure 18: Menu bar options for working with GIT

2. Git Panel [Figure 19]

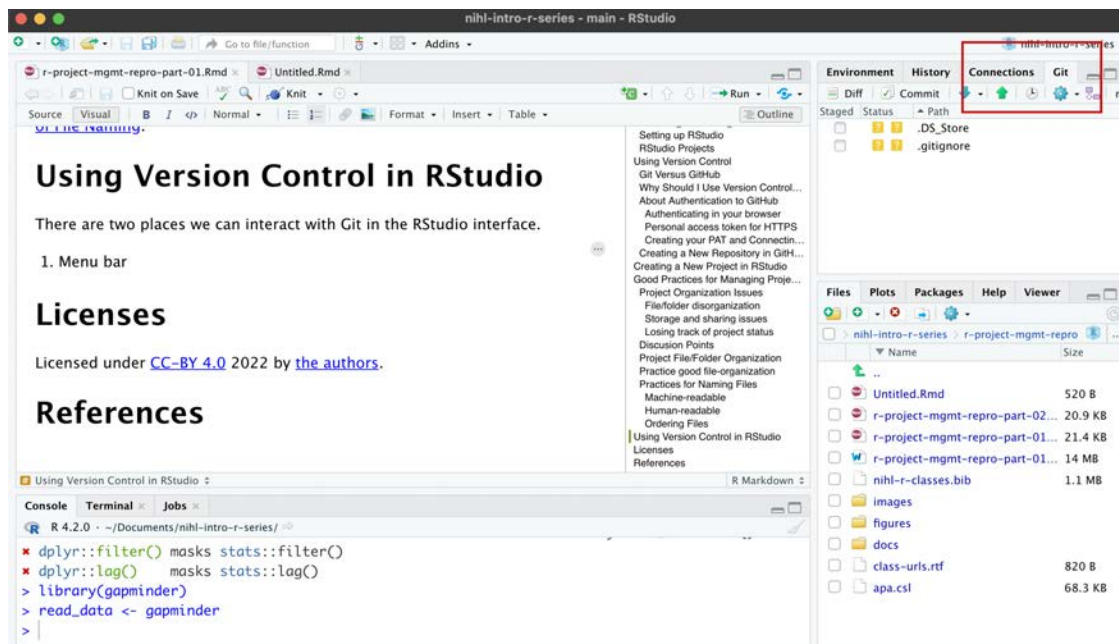


Figure 19: Git panel

It is not possible to cover everything you need to successfully use GIT. There are many excellent [resources](#) for Git.

Workflows in Git

Git workflows are probably not that different from what you do currently. For example, writing R scripts or authoring reports in LaTeX or R Markdown. But instead of only *saving* individual files, with Git you periodically **commit** your work. This means that you take a snapshot of all the files in the entire project. If you have ever versioned a file by adding your initials or the date, you have effectively made a commit, albeit only for a single file.



****footnote:** Thanks to Matt from UCLA for this comic idea!

Figure 20: A humorous look at version control.

This is like sharing a document with colleagues on DropBox or sending it out as an email attachment. By pushing to GitHub, you make your work and all your accumulated progress accessible to others.

Commits and diffs

We now connect the fundamental concepts of Git to the data science workflow:

- repository

- commit
- diff

Recall that a repository or repo is just a directory of files that Git manages holistically. A commit functions like a snapshot of all the files in the repo, at a specific moment. Figure 21 is a fictional analysis of the `iris` data, focusing on the evolution of a script, `iris.R`.

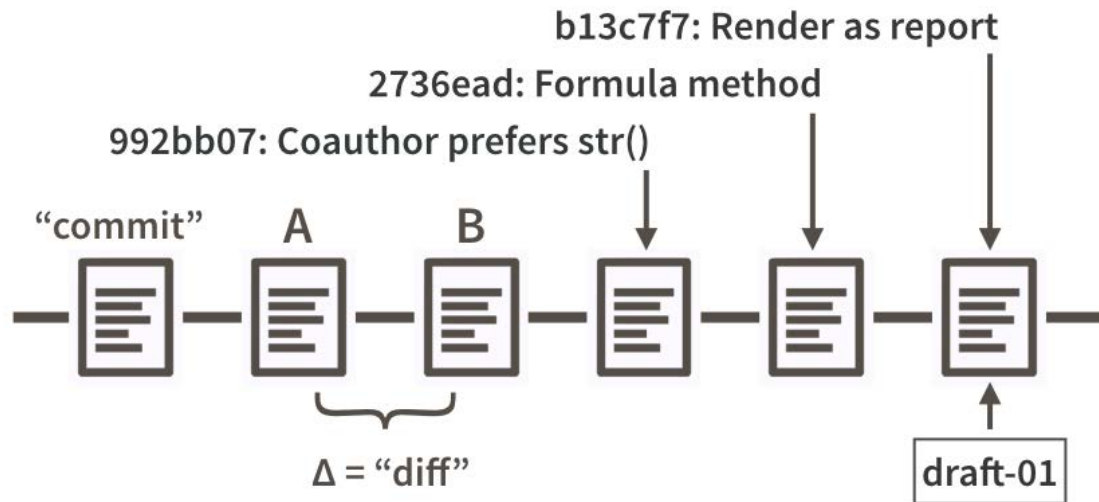


Figure 21: Partial commit history that looks at diffs, commit messages, SHA, and tags.

So what is going on in Figure 21. There are two versions of our document. Version A of this file and a modified version, version B. In this example, let us assume that version A was part of one Git commit and version B was part of the next commit. The set of differences between A and B is called a "diff."

Every time you make a commit you must also write a short **commit message**. Ideally, this conveys the motivation for the change. When you revisit a project after a break or need to digest recent changes made by a colleague, looking at the **history**, by reading commit messages and skimming through diffs, is an extremely efficient way to get up to speed. Figure 21 shows the messages associated with the last three commits.

Every commit needs some sort of nickname, so you can identify it. Git does this automatically, assigning each commit with an SHA, a seemingly random string of 40 letters and numbers (it is not, in fact, random). Though you will be exposed to these, you don't have to handle them directly very often and, when you do, usually the first 7 characters suffice. Figure 17 is showing the history for the training manual that I created for this class. You will notice the SHA column in Figure 22.

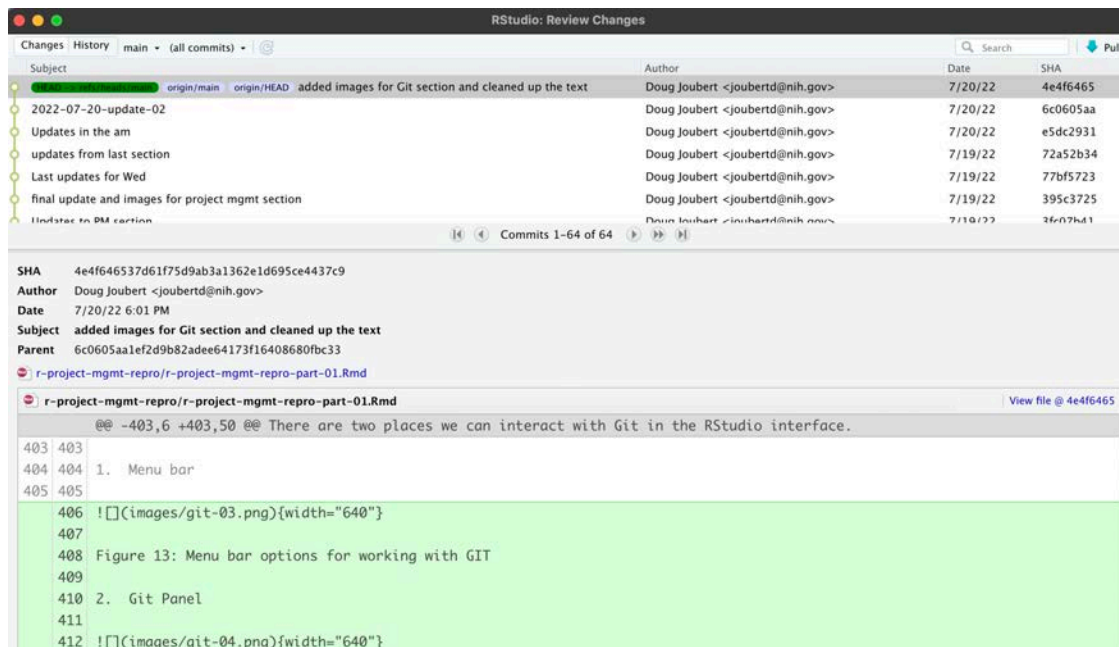


Figure 22: History for training manual.

The Git workflow can be summarized as:

Pull > add > commit > push

Pull: *download* the most recent version of the repository *from* GitHub *to* your local computer.

Push: *upload* the most recent version of the repository *to* GitHub *from* your local computer.

Tips for working with Git

- You should pull each time you start working on your project after a hiatus, or before each edit if you know a team member is working at the same time.
- Commit frequently, each commit should be a distinct set of edits which you can summarize in 50 characters or less. Don't add a bunch of unrelated edits to the same commit, it makes it harder to look back through your "snapshots" and find the right one if you need to.
- At the end of your work session (or more frequently if you are working at the same time as team members), "push" your commits to the remote repository - this is the only way your local changes get added to your team's remote repository.

This pull, add, commit, push routine will become second nature. Pulling at the beginning and pushing at the end of your work session becomes a sort of ritual that marks the beginning and end of your work session.

Pushing our Edits to Git

Create a new R-script named "proj-mgmt-01" Add the following lines of code


```
# Importing our Data

# This is an example of a code chunk to read in some data

library(tidyverse)

library(gapminder)

read_data <- gapminder

# Let us add some folders

dir.create("images")

dir.create("docs")

dir.create("figures")

# Do a quick image search for p53

# Download the image and move it into the new images folder that you just
created
```

We are now ready to send these changes to our remote repository

1. Select each file that has been modified from the list in the Git panel except. [Figure 23]

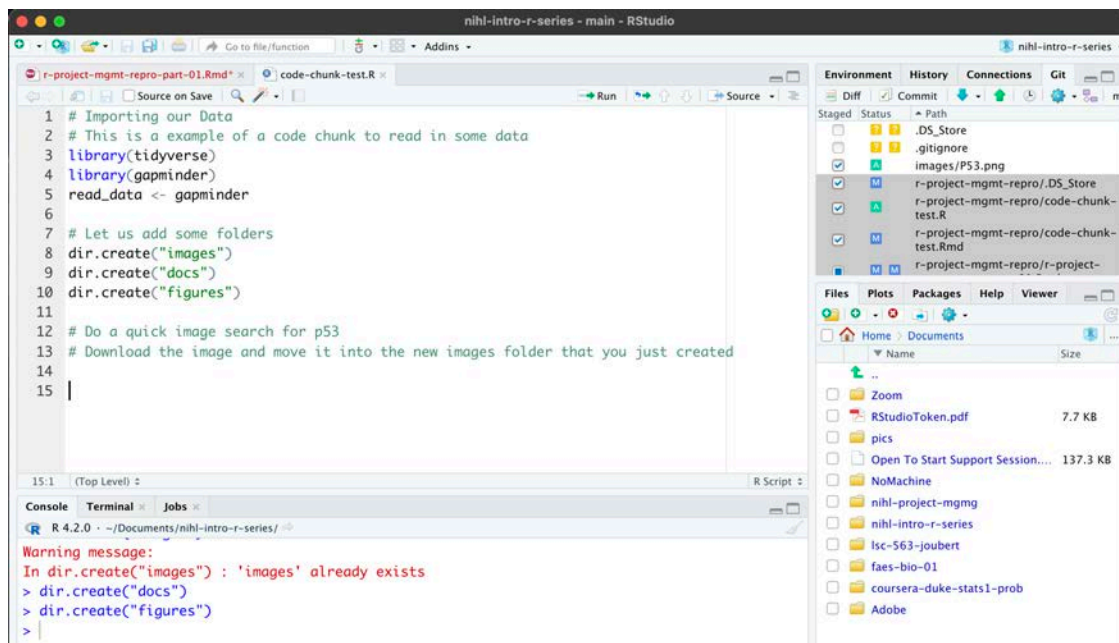


Figure 23: Selecting the files to send to remote repository.

2. Select the Commit icon on the Git Panel. Once you do this, the commit screen will refresh showing the changes [Figure 24].

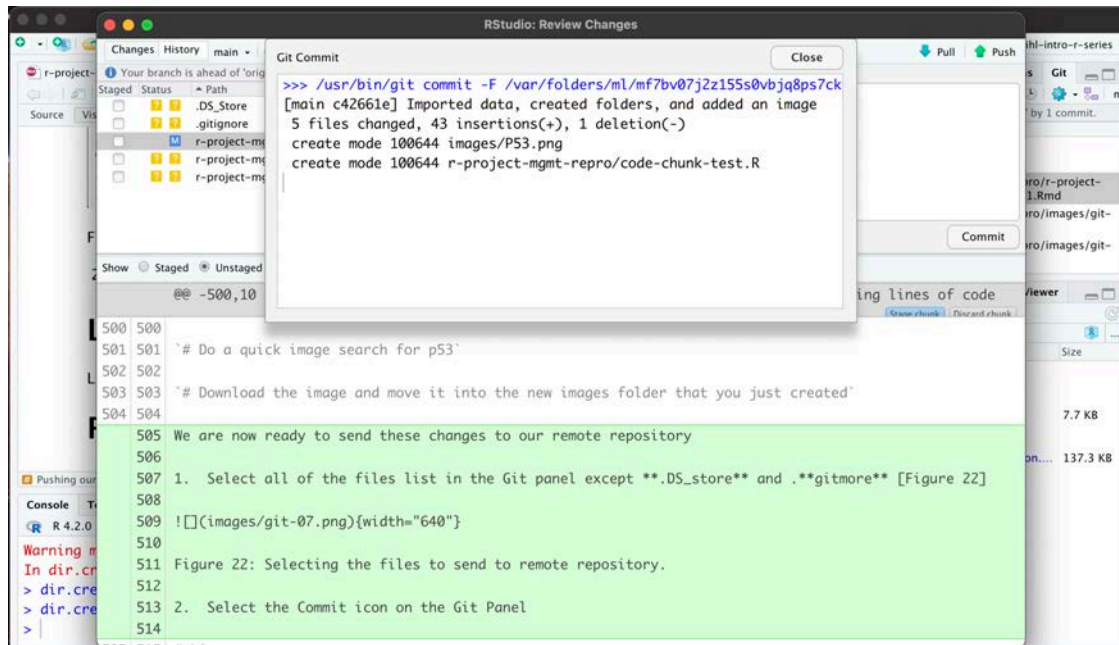


Figure 24: Changes being sent to GitHub.

3. Add a short commit message, and select the commit button [Figure 25].

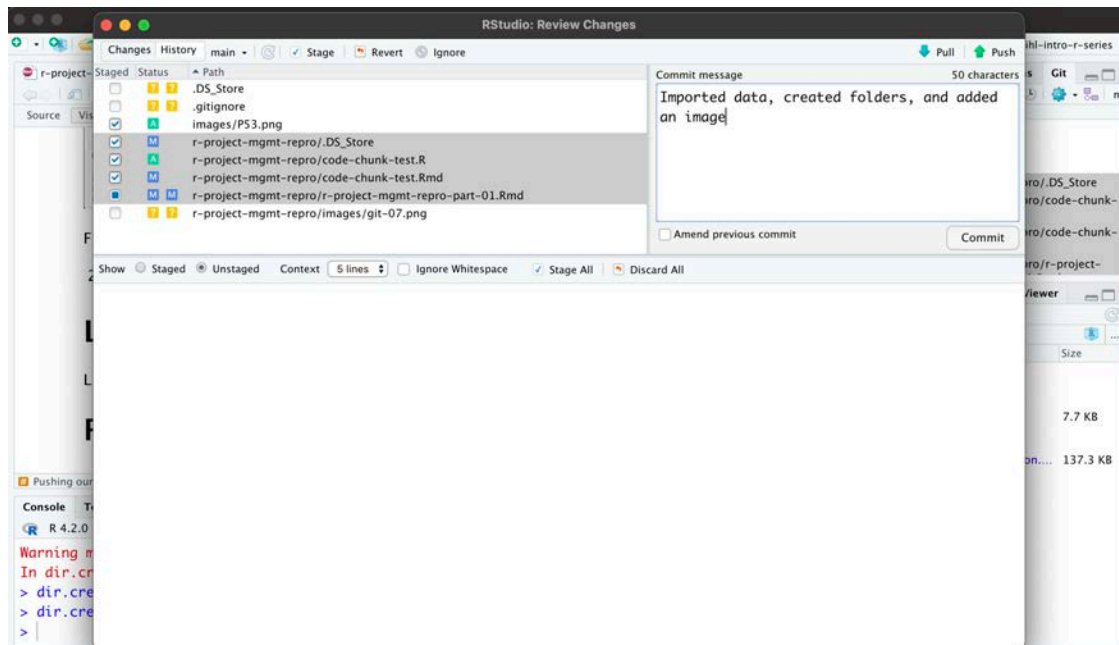


Figure 25: Adding a commit message and committing the changes.

4. Close the Git Commit screen
5. Select the Push button to send the changes to GitHub
6. If we check our repository on GitHub, the changes will be reflected [Figure 26]

github.com/doujouDC/nihl-intro-r-series

doujouDC / nihl-intro-r-series Private

Unwatch 1 Fork 0 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

Doug Joubert Added some more images e5d8171 6 minutes ago 69 commits		
images	Imported data, created folders, and added an image	12 minutes ago
r-intro-ggplot	Last updates before movie	7 days ago
r-project-mgmt-repro	Added some more images	6 minutes ago
raw-data	moved data into data folder	8 days ago
LICENSE	Initial commit	20 days ago
README.md	Initial commit	20 days ago
apa.csl	sunday updates 2	18 days ago
cindy-script.R	last updates before home	8 days ago
nihl-intro-r-series.Rproj	created files and folder for NIHIL r series	20 days ago
nihl-r-classes.bib.txt	sunday updates 2	18 days ago

About

No description, website, or topics provided.

Readme MIT license 0 stars 1 watching 0 forks

Releases

No releases published
[Create a new release](#)

Packages

No packages published

Licenses

Licensed under [CC-BY 4.0](#) 2022 by [the authors](#).

References

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility [Journal Article]. *Nature*, 533(7604), 452–454. <https://doi.org/10.1038/533452a>

Bollen, K., Cacioppo, J. T., Kaplan, R. M., Krosnick, J. A., & Olds, J. L. (2015). *Social, behavioral, and economic sciences perspectives on robust and reliable science* [Report]. Subcommittee on Replicability in Science.

Markowetz, F. (2015). Five selfish reasons to work reproducibly [Journal Article]. *Genome Biology*, 16(1), 274. <https://doi.org/10.1186/s13059-015-0850-7>

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing [Journal Article]. *PLOS Computational Biology*, 13(6), e1005510–e1005510. <https://doi.org/10.1371/journal.pcbi.1005510>