Code Management Guidelines

R and GitHub Starter Kit for New Team Members

Florencia D'Andrea

2024-12-23

Table of contents

Part I

Goal

GitHub Organization: www.github.com/StringhiniLab

The goal of this manual is to provide the **minimum necessary guidelines** for new members of Dr. Silvia Stringhini's lab to follow agreed-upon practices in code management.

Introduction

The use of programming languages has become an essential part of data analysis for most researchers today. In this context, a basic skill set in computer science is key to ensuring reliable and reproducible results (Wilson et al. 2017; Hicks 2023; Abdill, Talarico, and Grieneisen 2024). Although a variety of educational materials, tutorials, and recommended practices specifically designed to train researchers are available (The Carpentries; Our Coding Club; The Turing Way Community; CodeRefinery Project; Sherman Center Workshops 2024), there is a trade-off: adopting and practicing these techniques often requires significant effort, taking time away from researchers' primary fields of study (Allen and Mehler 2019; Goldsmith et al. 2021; Hicks 2023).

One consequence of the deficiency in training is the uncertainty researchers may have about how to write code correctly, which negatively impacts their willingness to share their analyses (Gomes et al. 2022). Thus, this results in a decrease in the number of publications with available code, impacting the reproducibility and transparency of scientific research (Gomes et al. 2022; Sharma et al. 2024). This issue is exacerbated by the lack of incentives from the scientific system, leading to a high number of publications where authors do not share their code, despite the benefits of making their code open source (Allen and Mehler 2019; Melvin et al. 2022; Bertram et al. 2023; Tazare et al. 2024; Xu et al. 2025).

Encouraging researchers to actively adopting best practices and seek training in the use of computational tools that facilitate or enhance their work is desirable and should be promoted. However, leaving code management decisions entirely in their hands could have negative consequences for a research group.

Ten reasons to implement code management practices early in a research group

Would the problem be solved if future new members of the lab arrived with better training in data science? No. We believe the research group should still define its priorities when it comes to managing code.

There are several benefits to defining clear minimum guidelines and basic computational skills from the moment new members join the lab:

1. **Set a solid foundation to avoid messy projects.**
   Define the file formats to be used and establish a basic file structure to ensure reproducibility from the project's inception. Additionally, outline how the data will be managed and integrated into the analysis.

2. **Define a consistent set of practices from all the different schools of thought.**
   Educational materials and training tutorials present various management practices, and researchers from different backgrounds may adopt different approaches. Therefore, providing clear guidelines ensures consistency in management practices across the projects.

3. **Focus on domain-specific skills first.**
   Identifying domain-specific computational skills can save time for new researchers. This knowledge is sometimes shared in publications tailored to each discipline but is too specific to be addressed by general training courses and tutorials for scientists.

4. **Early peer review.**
   In this manual, we suggest creating private repositories that are visible only to team members. Sharing analyses within these private repositories allows for valuable feedback. This practice could help researchers gain confidence in making their code publicly accessible once published and benefit from unpublished analyses conducted in the lab.

5. **Standardize documentation practices.**
   For example, there could be a README template that all researchers use, making it easy to understand what can be found in a repository. This saves time, facilitates access to materials for all team members, increases project reproducibility, and makes it easier to identify repositories with older analyses.

6. **Optimize time management.**
   Taking a workshop on a computational tool may occur at an advanced stage of a project. As a result, decisions about code organization, documentation, and file structure could have been made more effectively from the beginning, saving valuable time.

7. **Maintain the group's research history.**
   Centralizing data analyses on a repository hosting organization, such as a GitHub Organization, creates a historical archive of the group's data analyses, ensuring continuity and avoiding dependence on researchers leaving behind their code and data when they move on.

8. **Facilitate exchange of ideas about data and code management among team members.**
   Creating guidelines helps build a body of knowledge that can be improved over time with contributions from students/researchers, allowing for discussions on which practices should be added, prioritized, or removed.

9. **Make informed decisions about what to learn next.**
   A researcher might hear that they need to learn Git but have no idea what this tool is for. A brief introduction to Git and clear guidance on where to begin make it easier to assess

whether learning additional skills will be useful. Supporting new members in adopting basic computational techniques from the beginning lowers the barrier for researchers to explore other tools early.

10. **Adoption of open science practices.**
    If the group embraces open science, adopting these practices early will ensure that a high percentage of the code generated remains open source.

These ten reasons can serve as a starting point for opening a discussion on how to approach these topics within the research group. Leaders do not need to be experts in software development. Guiding principal investigators to select the essential tools and practices maximizes the benefits of making key decisions for the team without requiring large investments in learning.

At the same time, the existence of a research group manual allows younger researchers to share, propose, and contribute improvements on how the code is managed based on their expertise in the research area and the training they will receive. Eventually, the manual should include the criteria for publishing code and how to recognize the need to create a software package that can be used in the lab to facilitate the group's work.

Finally, beyond these ten reasons, there is an additional benefit: demonstrating how software will be maintained throughout the project lifecycle strengthens the case for long-term sustainability. This transparency encourages funding agencies to invest in similar future projects.

> **i How to cite this book?**
>
> D'Andrea, F., & Stringhini, S. Code Management Guidelines: R and GitHub Starter Kit for New Team Members. https://github.com/StringhiniLab/GitHubProceduresLab. Available at: https://stringhinilab.github.io/GitHubProceduresLab/ DOI: *https://doi.org/10.5281/zenodo.14510774*

Acknowledgments

References

What are the lab's basic guidelines for data and code usage?

Who are we, and what do we need?

Dr. Silvia Stringhini is an epidemiologist with an extensive career. She has served as the Head of the Unit of Population Epidemiology at the Geneva University Hospitals. Her main research areas include social inequalities in chronic diseases and aging, the role of health behaviors in the genesis of social health inequalities, the biological consequences of social inequalities, and the role of environmental factors in social health disparities.

Recently, she moved her lab to the School of Population and Public Health at the University of British Columbia in Canada, where she is establishing a new team. She is in the process of welcoming new students and staff, making this an ideal time to outline how her new group will manage data and code in its publications.

We discussed with Dr. Stringhini the minimal requirements regarding code management that someone joining the lab should learn. A summary of the agreements reached and how they influenced the creation of this manual can be found in Table 1. Many of these requirements focus on creating a GitHub Organization for the lab.

> **ℹ Definitions: Git, GitHub and GitHub Organization**
>
> **Git** is a tool that helps you track changes to your files, like a digital history of your work.
>
> **GitHub** is an online platform that uses Git to store files and manage changes, collaborate with others, and manage your code in a centralized location. It's like having a shared folder with built-in tools to see what's been changed and by whom. Each project can be stored in its own **repository**.
>
> A **GitHub organization** is a group on GitHub that allows teams to collaborate and manage repositories together.

Table 1: **Book content overview.** This table presents the benefits and selected topics included in the book to guide the team on each prioritized action.

Action	Benefit	What does this book cover?
Centralize the data analysis of the group in a GitHub organization	Preserve copies of the group's data analyses	- Steps to create a GitHub account and be added to the lab's organization - How to create a GitHub repository
Avoid sending confidential data to GitHub	Protection of sensitive data	- Use of `.gitignore` - Project structure recommendations including how to organize the data folder
Select R as the primary programming language and RStudio as the IDE	Standardize the software used in the lab	- Installation instructions - Recommendations on learning resources and good practices
Share a copy of each lab member's analyses in a private GitHub repository	- Create an initial version of the project that is organized and minimally documented so another lab member can understand it - Foster the habit of performing code backups - Receive feedback from colleagues early in the project development - Have access to analyses from other lab members that have not been published	- How to create a private GitHub repository and what information to include - Basic information to include in the `README` - Creating a GitHub team and defining procedures to manage access to private repositories - Develop a basic workflow for everyday use of Git and GitHub.
Store the code associated with scientific publications publicly in a GitHub repository	The benefits of leaving code open source Bertram et al. (2023).	This section will be developed at the time of the first paper's publication

Additionally, considering the lab's long-term evolution, onboarding and offboarding procedures were defined.

Justification

We recognize that creating the code for a scientific publication takes time and involves numerous attempts before deciding what figures and results effectively will be published. Keeping this in mind, it was decided that each student would generate a **private GitHub repository** by project to maintain a backup of daily data analyses conducted in the lab. This private repository could then serve as the foundation for a public version for the final GitHub repository with the scientific article's code. Publicly sharing the code and data management process can be more challenging for early-career researchers (Gomes et al. 2022; Tazare et al. 2024).

Maintaining this initial private repository has other benefits in addition to functioning as a backup: it allows sharing the code with other lab members (as part of a GitHub lab team), makes available analyses that may not be included in the final paper but could be relevant for another publication, helps keeping a clearer project structure from the beginning and **improves the overall documentation of the project**.

Characteristics specific to the research area were discussed, including handling **sensitive data** (Mathur and Fox 2023; The Turing Way Community). As a result, practices like using `.gitignore` and and creating a `data/` folder with `raw/` and `processed/` sub folders were suggested to prevent private data from being pushed to GitHub and to maintain an orderly system for storing such information within the project. Also, we created a `README` template designed to be accessible to non-programmers to be sure that the relevant information, as the database version in use and computational environment, is captured.

> **ℹ What is a `.gitignore` file**
>
> A `.gitignore` file is a special file used in Git repositories to specify which files and directories should be ignored, meaning they won't be tracked or included in version control. This is useful for excluding temporary files, sensitive data, or files that shouldn't be shared with others.

One of the more challenging aspects to adopt is using Git, as it has multiple utilities and a considerable learning curve. Considering this, it was decided that, in this initial stage, **Git and GitHub's** primary use would be to create an online and centralized backup of the projects, share repositories among team members, and manage version control instead of focusing in collaborative tools.

Since R is the most widely used programming language in the discipline, the team decided to leverage the Git integration provided by **RStudio's Git tab** for integrating students' local work into the GitHub repositories.

> **i** **What is the difference between R and RStudio?**
>
> **R** is a programming language and software environment used for statistical computing and data analysis.
> **RStudio** is an integrated development environment (IDE) for R, providing a user-friendly interface with tools for writing, editing, and running R code, as well as visualizing data and managing projects.

About lab manuals

In our case, we focused this manual on code management, but lab manuals vary in scope: some include code management, while others do not. Public resources discuss how and/or why to create a lab manual (Aly 2018; Tendler et al. 2023; The Turing Way Community). If you're looking for a practical example that incorporates code management, check the Computer-Oriented Geoscience Lab Manual (2025).

References

Defining roles

GitHub Organization Manager

The Owner of the organization can add new members. Ensure that new lab members join the GitHub Organization. Follow the instructions on how to do it in Inviting users to join your organization.

> ⚠ You will need to wait for the new member to create a GitHub account before you can add them. They should share their username with you.

Responsibilities

- Add new lab members to the GitHub Organization.

GitHub Team Maintainer

At least one person in the lab will be assigned as the Team Maintainer of the GitHub organization. This role will allow them to add and remove members from the team, granting or revoking access to private repositories.

For this, it's important to assign the Lab Member in charge the **Team Maintainer** role for the Lab Team.
Follow the instructions in Assigning the team maintainer role to a team member

If properly assigned the label `maintainer` should appear next to their name in this repository: https://github.com/orgs/StringhiniLab/teams/lab-team.

Responsibilities

- Add new lab members to the Lab Team.
- Remove team members who are no longer part of the lab.
- Assign other members as Team Maintainers if necessary.

Part II

Practical Steps

1 Onboarding

Checklist

☐ Create your personal GitHub account.

☐ Request access to the GitHub Organization.

☐ Request access to the Lab Team.

☐ Install Git, R and RStudio.

1.1 Create Your Personal GitHub Account

First, you need to create an account on GitHub by following these steps: Creating an account on GitHub.

After creating your GitHub account, you'll notice that your profile is associated with a specific URL, structured as follows:

- `https://github.com/<username>`

This page allows you to access your account settings and all repositories you create.

1.2 Request Access to the GitHub Organization

Our lab has a [GitHub organization](#) that centralizes the repositories for everything produced in the lab.
Notice that the organization's URL is different from your profile's:

- https://github.com/StringhiniLab

To gain access, you need to provide your GitHub username to the person managing the organization. Having access to the organization will enable you to create repositories within it.

> **i** Note that you have your personal GitHub url, and also there is the url of the GitHub organization.

1.3 Request Access to the Lab Team

Not all lab repositories are public. To access repositories owned by other lab members, you must also be added to the Lab Team.
Dr. Stringhini, as the owner of the organization, always has access to all repositories.

- [Adding organization members to a team](#)
- [Managing team access to an organization repository](#)

1.4 Installation Instructions

To work in the lab, you will need to have Git, R, and RStudio installed.
Please note that the installation process may vary depending on whether you have a computer with Windows, Linux, Mac (Intel chip), or Mac M1, M2 and M3 (Apple Silicon chip) operating systems.

- **Git**: [Download Git](#)
  You can read more about options on how to [install Git](#) in the book Happy Git and GitHub for the useR (Bryan and Hester 2024)
- **R**: [Download R](#)

- **RStudio**: [Download RStudio](#)
  If you want to read some more details about basic use of R and RStudio you can read the [preqrequisites](#) section of the book R for Data Science (Wickham and Grolemund 2024)

> ⚠️ **Installation issues**
>
> It could happen that you find issues during the installation process. Remember that we are a team and that you can use Slack to share the problem and look for support

References

2 Starting a New Project

These steps only need to be completed once, at the beginning of a project.

Checklist

☐ Create a private repository by new project.

☐ Add the Lab Team to the repository.
☐ Clone the repository and associate it with an RStudio Project.
☐ Add your Data folder to `.gitignore`.
☐ Complete the `README` file using the template.

> **i** The most comprehensive book on using Git and GitHub for R users is Happy Git and GitHub for the useR (Bryan and Hester 2024)

2.1 Create a private repository by new project

When starting to work on a new project, your first step is to create a **private** repository in the lab's GitHub organization: StringhiniLab GitHub.

1. Click the green **New** button to open a window like this:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *   **Repository name ***

StringhiniLab ▾   /   chronic-diseases_private

✅ chronic-diseases_private is available.

Great repository names are short and memorable. Need inspiration? How about **silver-fiesta** ?

Description (optional)

Code for analyzing and modeling chronic disease patterns using data from the Canadian Longitudinal Stu

○ 🖥️ **Public**
Anyone on the internet can see this repository. You choose who can commit.

🔘 🔒 **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☑️ Add a README file
This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. Learn more about licenses.

This will set 🔀 main as the default branch.

ⓘ You are creating a private repository in the StringhiniLab organization.

Create repository

2. Complete/select using the following criteria:

☐ **Owner**
Select `StringhiniLab` as the owner, not your personal GitHub account.

☐ **Repository Name**
Choose a name that represents your project. Since this repository will be private, append `_private` to the name.
For example, if the repository name is `chronic-diseases`, name it `chronic-diseases_private`.

☐ **Description**

18

Provide a more detailed description of the project here. This helps identify the repository's content in the organization.

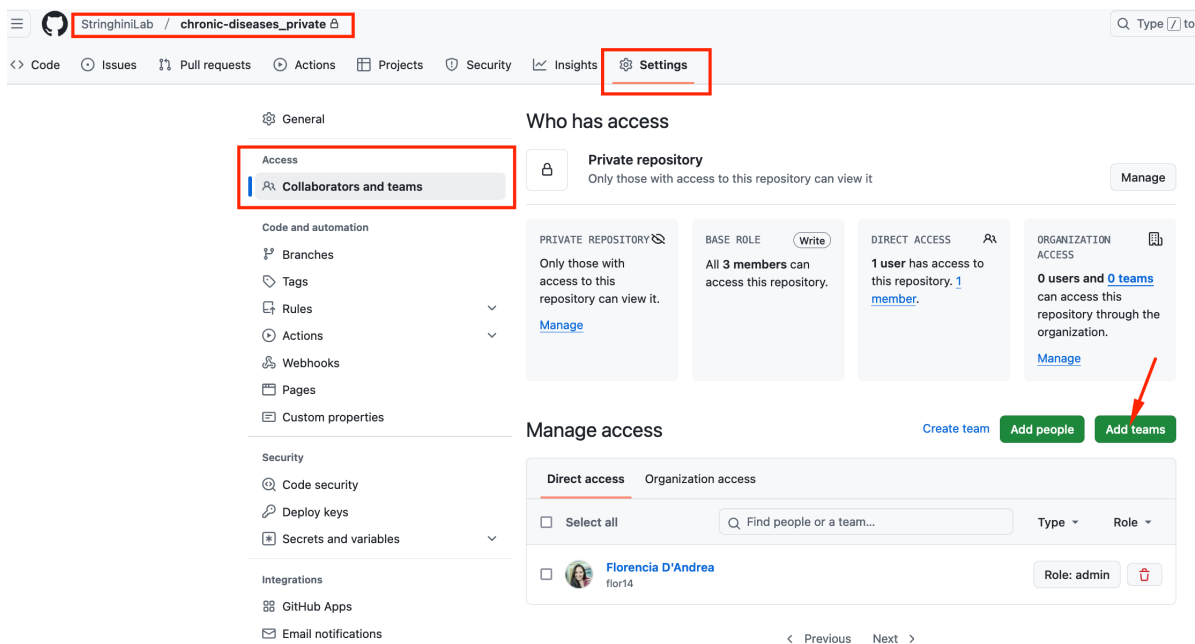☐ **Public or Private?**
Ensure the repository is set to `Private`.

☐ **Initialize Repository With**
Add a `README.md`.

Ignore the other options for now.
If everything works, you'll see your repository within the GitHub organization labeled as `Private`.

2.2 Add the Lab Team to the repository

1. **Navigate to the repository's `Settings` tab.**
In the left-hand sidebar, find and click on `Collaborators and teams`.

2. Click **Add teams** and add `Lab Team`.
By default, you will select the Read role for the team. The idea is that other Lab Members can view the repository but will not be able to edit it by mistake.

This allows all current Lab Members to view (but not modify) your project.

If you don't want to share an analysis with other Lab Members, you can create a repository in your personal GitHub account instead. However, **always ensure sensitive data is not pushed to GitHub** for confidentiality reasons.

All repositories in `StringhiniLab` should be accessible to the Lab Team, which is why these repositories are hosted in the organization instead of personal accounts. Remember that individuals who are Owners of the organization can view all repositories even if there are not part of the Team.

2.3 Clone the repository and associate it with an RStudio Project

> **ℹ Before start using RStudio:**
>
> We recommend changing the default option `Restore .RData into workspace at startup` as explained in this section. (Posit 2024)
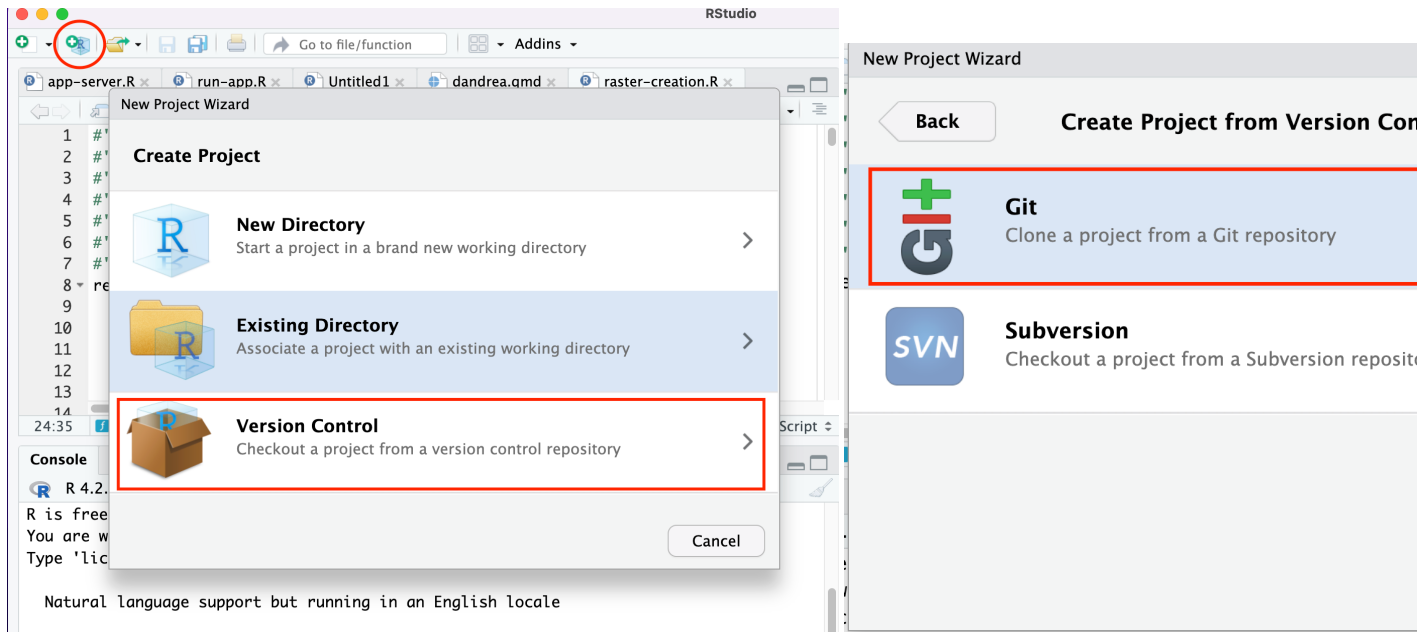
1. **Open RStudio**
   If RStudio is not installed, complete first the installation instructions in the Onboarding section.
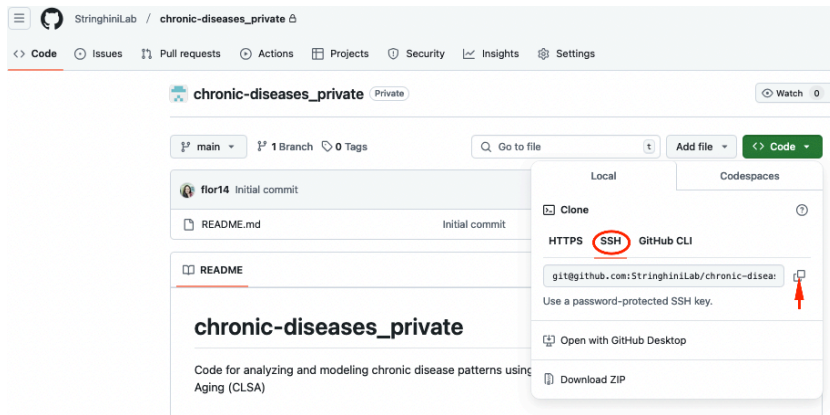
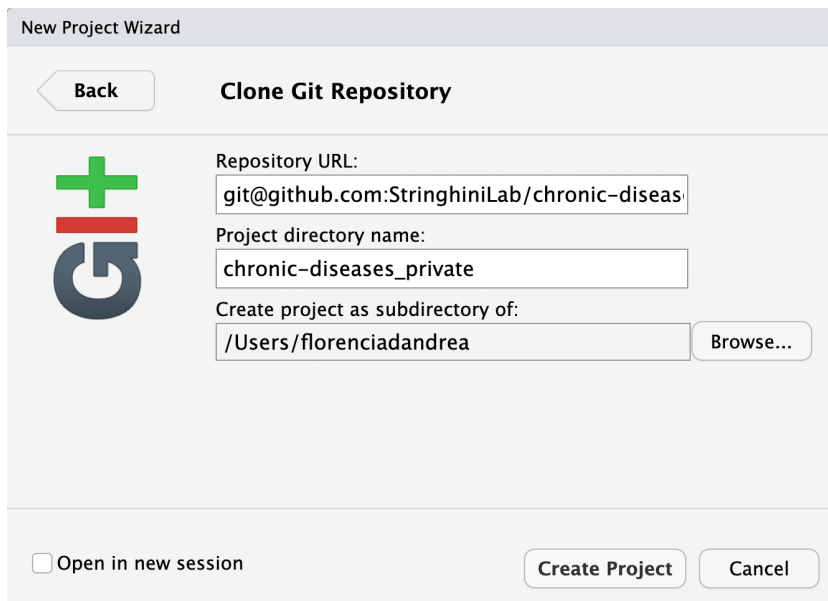2. **Clone the Repository**
   In RStudio:
   `File > New Project > Version Control > Clone a Project from a Git Repository`.

Go back to the repository and copy the repository's URL.

And paste it in the correct field:

If successful, you'll see a folder containing your project, including the `README` file that we created on GitHub, in the `Files` tab at the bottom-right of RStudio.

Note that an `.Rproj` icon has appeared. Clicking on this icon outside of RStudio will open RStudio directly within the project.

Additionally, the project name now appears in the top-right corner. By opening that menu, you can easily switch between projects without leaving the RStudio IDE.

You'll also notice a tab named Git in the top-right panel.
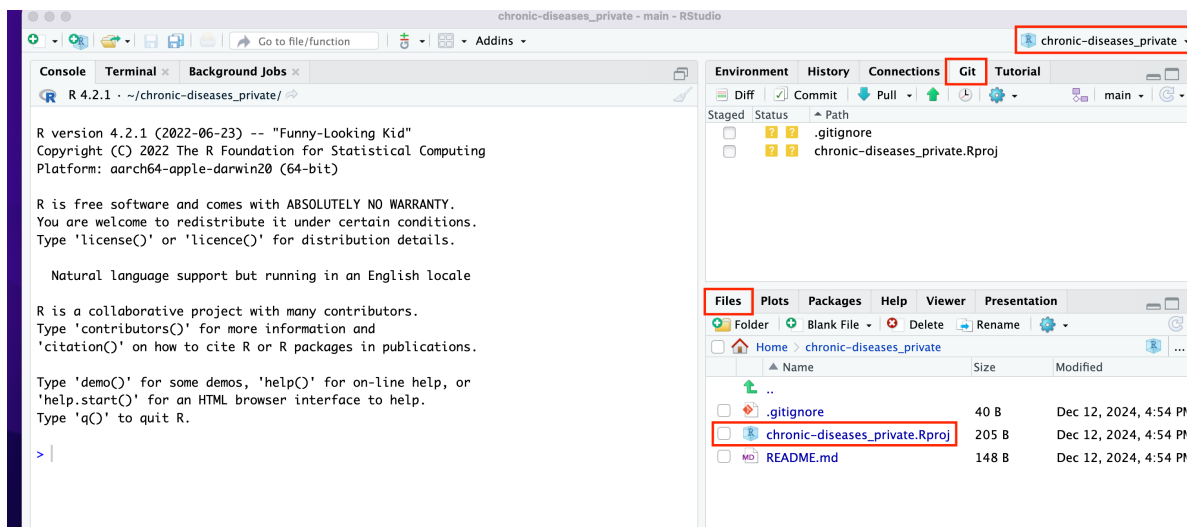
> **ℹ Do you want to learn more about the use of RStudio and how to create projects?**
>
> - Check the RStudio IDE User Guide (Posit 2024)
>
> - Read the Projects section in Chapter 6 of R for Data Science (Wickham and Grolemund 2024)

2.3.1 Add your Data folder to `.gitignore`

We work with sensitive data. If working locally, create a `data` folder (e.g., click the `+ Folder` icon in the `Files` tab). Move your data files into this folder.

Open the `.gitignore` file and add the line `data/`. This tells Git to ignore the contents of the `data` folder, preventing accidental data pushes.

We recommend creating at least two sub-folders within `data/`:

- `raw/`: Use this folder to store the original datasets.
- `processed/`: Use this folder to save any datasets generated as preliminary or final results from your analyses.

If `data/` is listed in your `.gitignore` file, both subfolders will automatically be ignored by Git since they are located within the `data/` folder.

If the folder is not in the project root or has a different name, adjust the `.gitignore` settings accordingly.

Your project structure should look like this:

```
project-folder/
   .gitignore # Specifies files and folders to ignore in version control
   README.md # Documentation about the project
   data/ # Folder to store datasets
      raw/ # Original datasets (never modified directly)
      processed/ # Cleaned and processed datasets
```

> **i What type of document should you use for your data analysis?**
>
> There are many files you can use run your code. You can use a basic R script (Wickham and Grolemund 2024), an RMarkdown document or its more current version Quarto. If you're unsure where to start, we recommend using a Quarto Document. A Quarto document, just like RMarkdown, allows you to include code chunks throughout a text and save it as a Word, PDF, or HTML report. Combining text where you explain your reasoning and the details of the analysis with the code chunks makes it much easier to understand each section of code, both for yourself and for other readers.
>
> - To use Quarto, you must first install it
> - Tutorial to learn how to create a document with Quarto (Quarto Project 2024)
>
> It's important to note that there is a learning curve for using tools like Quarto. Creating a simple R script might be a good enough option if you're looking to start your analysis more straightforwardly.
> **We encourage you to discuss this on the lab Slack to hear the experiences of other researchers. If any consensus emerges, feel free to open a New Issue here to improve this guide.**

and you `.gitignore` file should look like this:

```
.Rproj.user
.Rhistory
.RData
.Ruserdata
data/
```

2.4 Complete the `README` using the template

Before starting work, fill out the `README.md` file with the following information:

```
# Title

## Author
**Name:** [Your Name]
**Email:** [Your Name]

## Start Date
```

```
[YYYY-MM-DD]

## Objective
The objective of this project is to ...

## Database Used and Version
**Database Name:** [Name]
**Data Version:** [Specify version or date accessed]

-[] Sensitive data is stored locally and excluded from version control using `.gitignore.`
-[] All analyses comply with the data use agreements.

## Project Structure

chronic-diseases/
  data/ # Folder for datasets
    raw/ # Original datasets (never modified directly)
    processed/ # Cleaned and processed datasets
  scripts/ # R scripts for analysis
  outputs/ # Figures, tables, and other results
  README.md # Project overview and documentation
  .Rproj # RStudio project file

## Reproducibility
Run `sessionInfo()` in the console and paste the output between the two lines with backticks

```

```
```

There is more material available on organizing project structures (Eugene Barsky 2024), using
.gitignore (The Turing Way Community) or other resources listed here, or creating good
README files (The Turing Way Community) and name conventions (Eugene Barsky 2024)
if you want to explore further.

Additionally, UBC Library offers some basic data management courses that might be helpful
to you: Short courses (Eugene Barsky 2024)

References

3 Regular Project Workflow

These steps should be completed every day you work on the project. Although they may seem complex at first, once you get accustomed to them, you won't need to think about it anymore.

Checklist

☐ Add all the files you want to commit to the staging area.

☐ Create a commit message.

☐ Push the changes to the GitHub repository.

3.1 How to use the Git tab in RStudio

> ⚠️ **Why GitHub?**
>
> Git and GitHub were originally created for professional software development. It's important to clarify that lab members are not expected to be expert users of Git and GitHub, but rather to handle basic commands necessary to achieve the use proposed.

Pay attention to the Git tab located in the top-right corner. Git will only display files that have been added, modified, or deleted since the project was initialized or since the last commit (we'll cover what that means shortly).

Keep in mind that when we cloned the project from GitHub, two new files were added:

- The `.Rproj` file, created because we based our RStudio project on the repository.

- The `.gitignore` file, automatically generated as part of the project setup in RStudio.

These files will appear with a yellow question mark, indicating they are *untracked*—in other words, Git is aware of them but has not yet saved them under version control.

The `README.md` file initially did not appear in the Git tab. However, after adding the template and saving the changes, it now shows a blue "M," which indicates that the file has been *modified.*

If you were to remove a file, you would see it next to a red 'D', indicating that it has been *deleted* from the project.
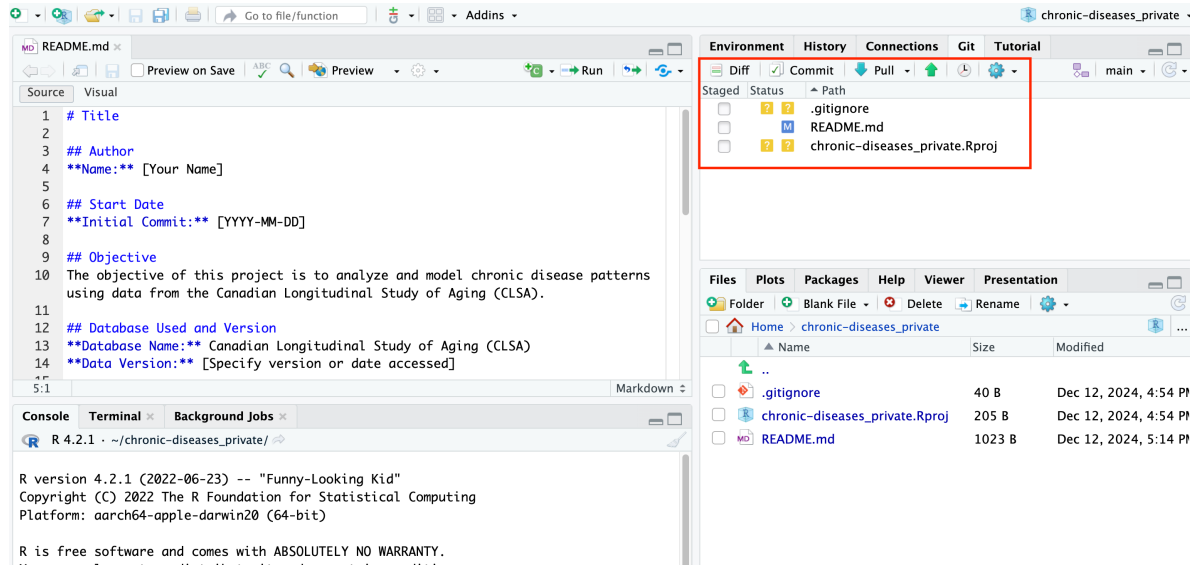

The next step is to save these changes in the project and add a descriptive title. Each time you save a new version of the project, we say you are making a commit, which you label with a title.

### 3.1.1 Add all the files you want to commit to the staging area.

The first thing you need to do is check in the Staged section of the Git tab all the changes you want to save under the same title. You will notice that sometimes a green 'A' appears. You can ignore this. The important part is that you check all the changes you want to save.

In our case, since we are working with non-collaborative repositories and the main purpose of using GitHub is to share data with other coworkers and maintain a backup. If at any point you need to learn more, at the end of this page, there are some recommended materials that might be helpful.
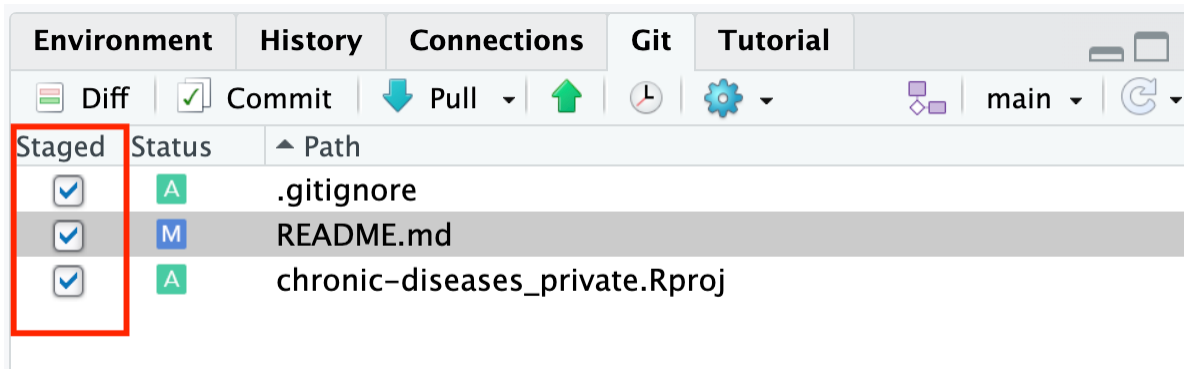
3.1.2 Create a commit message.

After doing this, you need to click the commit button to make these changes permanent in the project. You will then choose a message for the commit and click the `Commit` button.
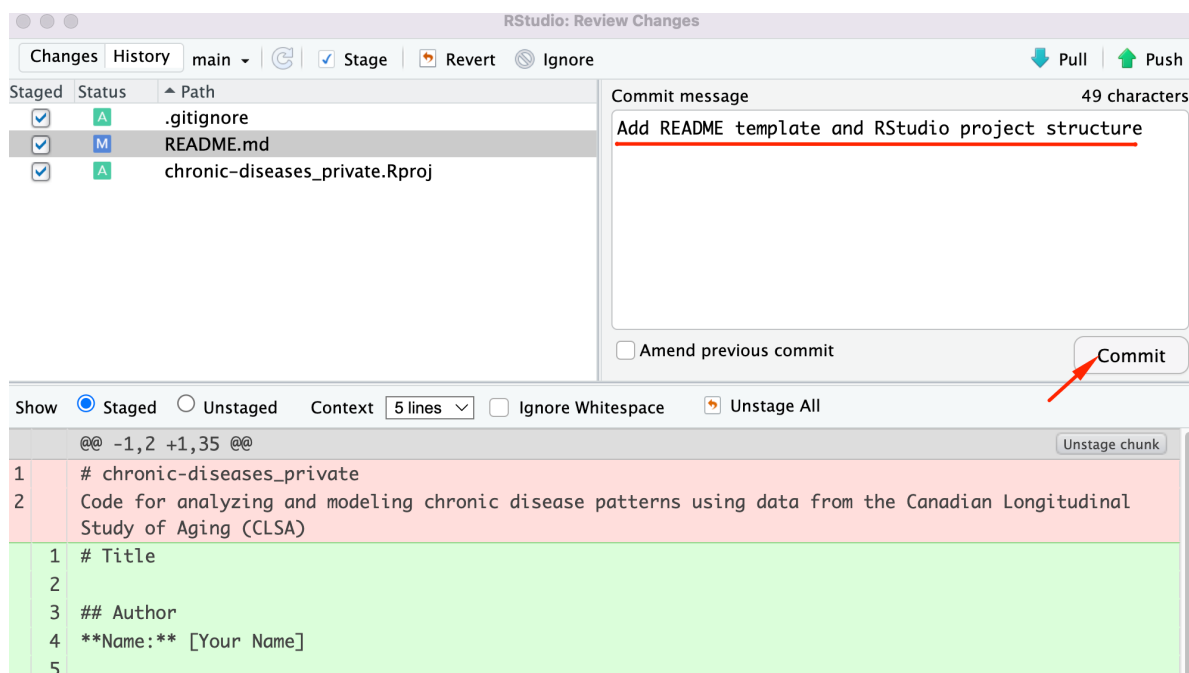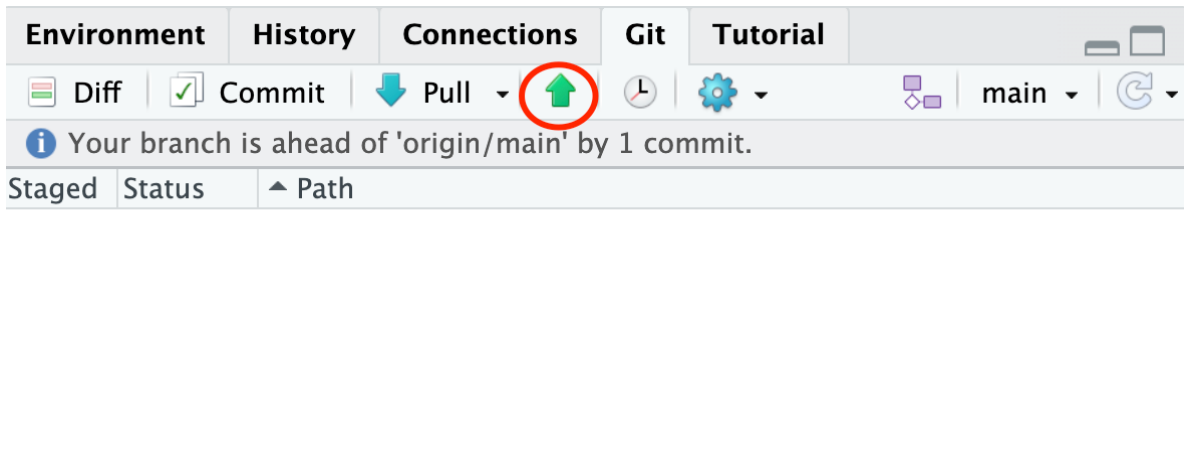

> 💡 **How often should you commit?**
> Think of commits as checkpoints for related changes. If you might want to revert a set of changes later, commit them together.

3.2 Push the changes to the GitHub repository.

Finally, to push the changes to GitHub, click the `Push` button.

You'll notice that the files in the Git tab disappear after you commit. This is expected, as Git only tracks changes between commits. Remember, if you don't click the green arrow representing push, the changes will not take effect on GitHub.

If everything went smoothly, you should navigate to the repository URL and see the changes you made.

> 💡 **How often should you push?**
> Push your changes **at least once a day** after completing your work.

3.3 Moving foward with Git and GitHub

If you want to learn more about Git and GitHub, we recommend the following books and tutorials:

Books

- Happy Git and GitHub for the useR book (Bryan and Hester 2024)
- Chapter 12 - Collaboration with version control from the book Data Science: A First Introduction (Timbers, Campbell, and Lee 2022)
- The Turing Way Handbook: Git for research projects (The Turing Way Community)

Tutorials

- Software Carpentry: Version Control with Git (Gonzalez et al. 2019)

- Setting up a GitHub Repository for Your Lab - Version Control and Code Management with GitHub (Our Coding Club)

- Code Refinery - Introduction to version control with Git (CodeRefinery Project)

References

4 Offboarding

- The Lab Member must be removed from the GitHub Lab Team.

When leaving the organization, you will lose access to private repositories created by other team members but will retain access to those you created yourself.

Abdill, Richard, Emma Talarico, and Laura Grieneisen. 2024. "A How-to Guide for Code Sharing in Biology." *PLoS Biology* 22 (9): e3002815.

Allen, Christopher, and David MA Mehler. 2019. "Open Science Challenges, Benefits and Tips in Early Career and Beyond." *PLoS Biology* 17 (5): e3000246.

Aly, Mohamed. 2018. "The Key to a Happy Lab Life Is in the Manual." *Nature* 561 (7721): 7–7. https://doi.org/10.1038/d41586-018-06167-w.

Bertram, Michael G, Josefin Sundin, Dominique G Roche, Alfredo Sánchez-Tójar, Eli SJ Thoré, and Tomas Brodin. 2023. "Open Science." *Current Biology* 33 (15): R792–97.

Bryan, Jennifer, and Jim Hester. 2024. *Happy Git and GitHub for the useR.* https://happygitwithr.com/.

CodeRefinery Project. "CodeRefinery Lessons." https://coderefinery.org/lessons/.

Computer-Oriented Geoscience Lab Manual. 2025. "Lab Manual. Computer-Oriented Geoscience Lab. University of São Pablo." https://www.compgeolab.org/manual/.

Eugene Barsky, Paul Lesack, Billie Hu. 2024. "Introduction to Research Data Management." https://github.com/ubc-library-rc/rdm/.

Goldsmith, Jeff, Yifei Sun, Linda Fried, Jeannette Wing, Gary W Miller, and Kiros Berhane. 2021. "The Emergence and Future of Public Health Data Science." *Public Health Reviews* 42: 1604023.

Gomes, Dylan GE, Patrice Pottier, Robert Crystal-Ornelas, Emma J Hudgins, Vivienne Foroughirad, Luna L Sánchez-Reyes, Rachel Turba, et al. 2022. "Why Don't We Share Data and Code? Perceived Barriers and Benefits to Public Archiving Practices." *Proceedings of the Royal Society B* 289 (1987): 20221113.

Gonzalez, Ivan, Daisie Huang, Nima Hejazi, Katherine Koziar, and Madicken Munk. 2019. "Software Carpentry: Version Control with Git." Edited by Ivan Gonzalez, Daisie Huang, Nima Hejazi, Katherine Koziar, and Madicken Munk. https://doi.org/10.5281/zenodo.3264950.

Hicks, Daniel J. 2023. "Open Science, the Replication Crisis, and Environmental Public Health." *Accountability in Research* 30 (1): 34–62. https://doi.org/10.1080/08989621.2023.1962713.

Mathur, Maya B, and Matthew P Fox. 2023. "Toward Open and Reproducible Epidemiology." *American Journal of Epidemiology* 192 (4): 658–64. https://doi.org/10.1093/aje/

kwad007.

Melvin, Ryan L, Steven J Barker, Joe Kiani, and Dan E Berkowitz. 2022. "Pro-Con Debate: Should Code Sharing Be Mandatory for Publication?" *Anesthesia & Analgesia* 135 (2): 241–45.

Our Coding Club. "Setting up a GitHub Repository for Your Lab - Version Control and Code Management with GitHub." https://ourcodingclub.github.io/tutorials/git-for-labs/.

Posit. 2024. "RStudio IDE User Guide." 2024. https://docs.posit.co/ide/user/.

Quarto Project. 2024. "Hello, Quarto: Using Quarto with RStudio." 2024. https://quarto.org/docs/get-started/hello/rstudio.html.

Sharma, Nitesh Kumar, Ram Ayyala, Dhrithi Deshpande, Yesha Patel, Viorel Munteanu, Dumitru Ciorba, Viorel Bostan, et al. 2024. "Analytical Code Sharing Practices in Biomedical Research." *PeerJ Computer Science* 10: e2066.

Sherman Center Workshops. 2024. "Best Practices for Managing Your Code and Scripts You Use to Generate Your Research." https://learn.scds.ca/dr23-24/code-best-practices.html.

Tazare, John, Shirley V Wang, Rosa Gini, Daniel Prieto-Alhambra, Peter Arlett, Daniel R Morales Leaver, Caroline Morton, et al. 2024. "Sharing Is Caring? International Society for Pharmacoepidemiology Review and Recommendations for Sharing Programming Code." *Pharmacoepidemiology and Drug Safety* 33 (9): e5856.

Tendler, Benjamin C, Maddie Welland, Karla L Miller, and The WIN Handbook Team. 2023. "Research Culture: Why Every Lab Needs a Handbook." *eLife* 12 (July): e88853. https://doi.org/10.7554/eLife.88853.

The Carpentries. "WebPage. The Carpentries Teaches Foundational Coding and Data Science Skills to Researchers Worldwide." https://carpentries.org/.

The Turing Way Community. "The Turing Way: A Handbook for Reproducible, Ethical and Collaborative Research." Zenodo. https://doi.org/10.5281/zenodo.7625728.

Timbers, Tiffany, Trevor Campbell, and Melissa Lee. 2022. *Data Science: A First Introduction - Version Control.* Chapman and Hall/CRC. https://datasciencebook.ca/version-control.html.

Wickham, Hadley, and Garrett Grolemund. 2024. *R for Data Science.* https://r4ds.hadley.nz/.

Wilson, Greg, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. 2017. "Good Enough Practices in Scientific Computing." *PLOS Computational Biology* 13 (6): 1–20. https://doi.org/10.1371/journal.pcbi.1005510.

Xu, Edward, Anna Catharina V. Armond, David Moher, and Kelly Cobey. 2025. "Key Challenges in Epidemiology: Embracing Open Science." *Journal of Clinical Epidemiology* 178: 111618. https://doi.org/https://doi.org/10.1016/j.jclinepi.2024.111618.