

Version Control Systems

What is GitHub?

Version Control through the GitHub website

Git with RStudio

Keeping your local code updated on GitHub

(Optional) Working in a Team

R Markdown

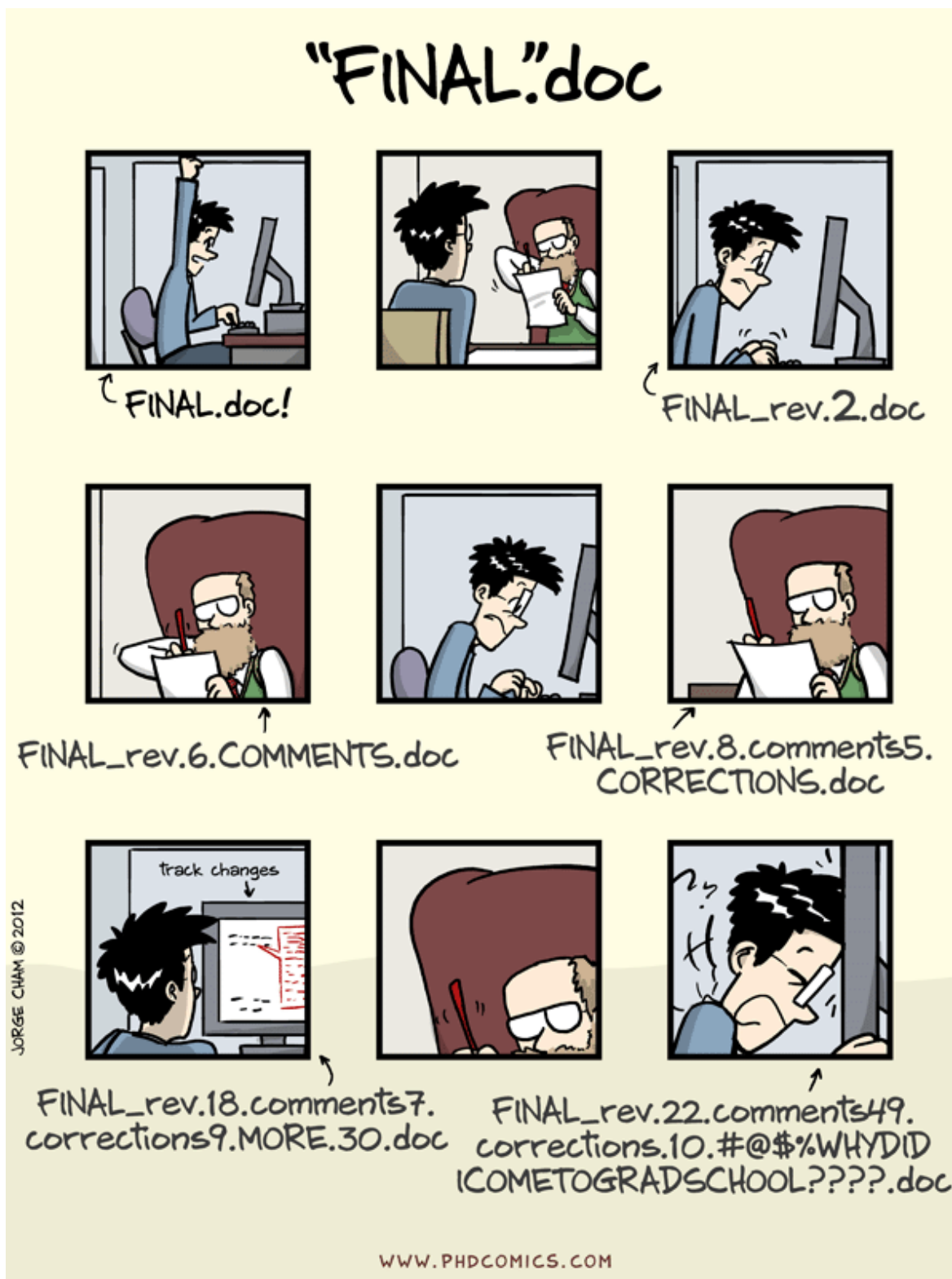
Alexia Cardona

Version Control Systems

A version control system is a system which keeps track of the history/changes/versions of a file making it easier for multiple people to work on the same project at the same time. This gives them the possibility of comparing changes between different versions of the file, move back to an previous version of a file if needed, keep track on who,when and why someone in the team made changes to a file.

This system is popular in the software development industry as it allows teams to have a common location or **repository** where all the source code/files in the project are kept and thus are able to work on an updated version of the project.

Version control systems are also popular as they prevent the situation of loosing all your work if your computer breaks down. Most probably you have already been using a kind of version control without realising by renaming your files.



A version control system will be keeping versions for each file in the repository behind the scenes, so you do not need to worry about renaming files, etc...

Version control systems are often categorised as either **centralised** version control systems or **distributed** version control systems.

Centralised Version Control Systems

A centralised version control system has only one copy of the project held in a central place (server) which also keeps track of the files' history (versions). The people working on the project would be able to **commit** or make changes to the central server. Normally multiple people are not able to work on the same

file simultaneously. Examples of centralised version control systems are CVS and Subversion (SVN). Distributed version control systems are becoming increasingly more popular as they overcome some of the limitations imposed by centralised version control systems.

Centralized version control

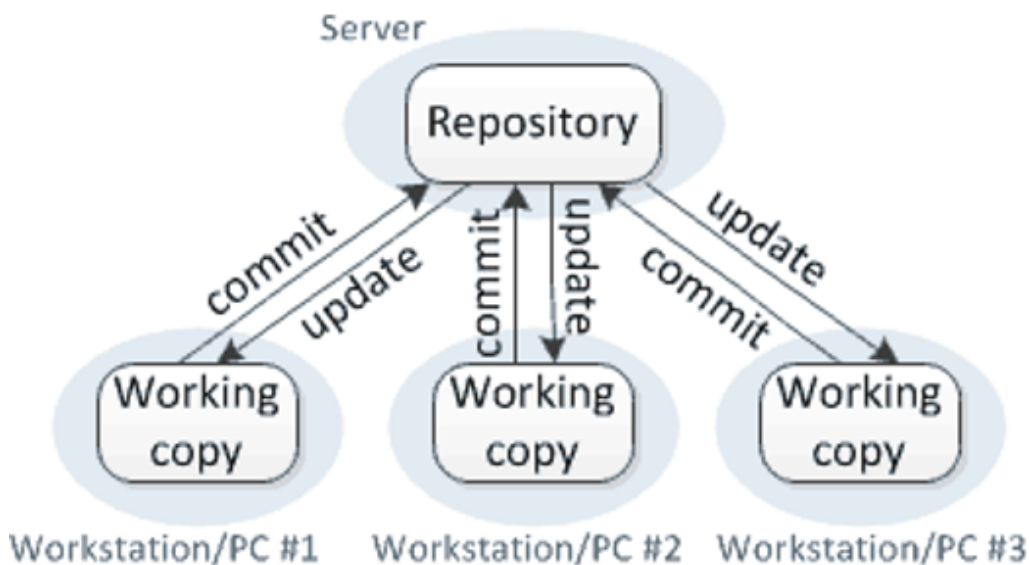


Image from <https://homes.cs.washington.edu/~mernst/advice/version-control.html>
(<https://homes.cs.washington.edu/~mernst/advice/version-control.html>)

Distributed Version Control Systems

In a distributed version control system each person in a team working on the project makes a copy or **clones** the repository on their own computer which also includes the full history of the different files inside the repository. Thus if a server dies, it would be able to be restored back from the copies of the repositories held by the different people in the team.

In this course we will be learning about **git** which is an example of a distributed version control system.

Distributed version control

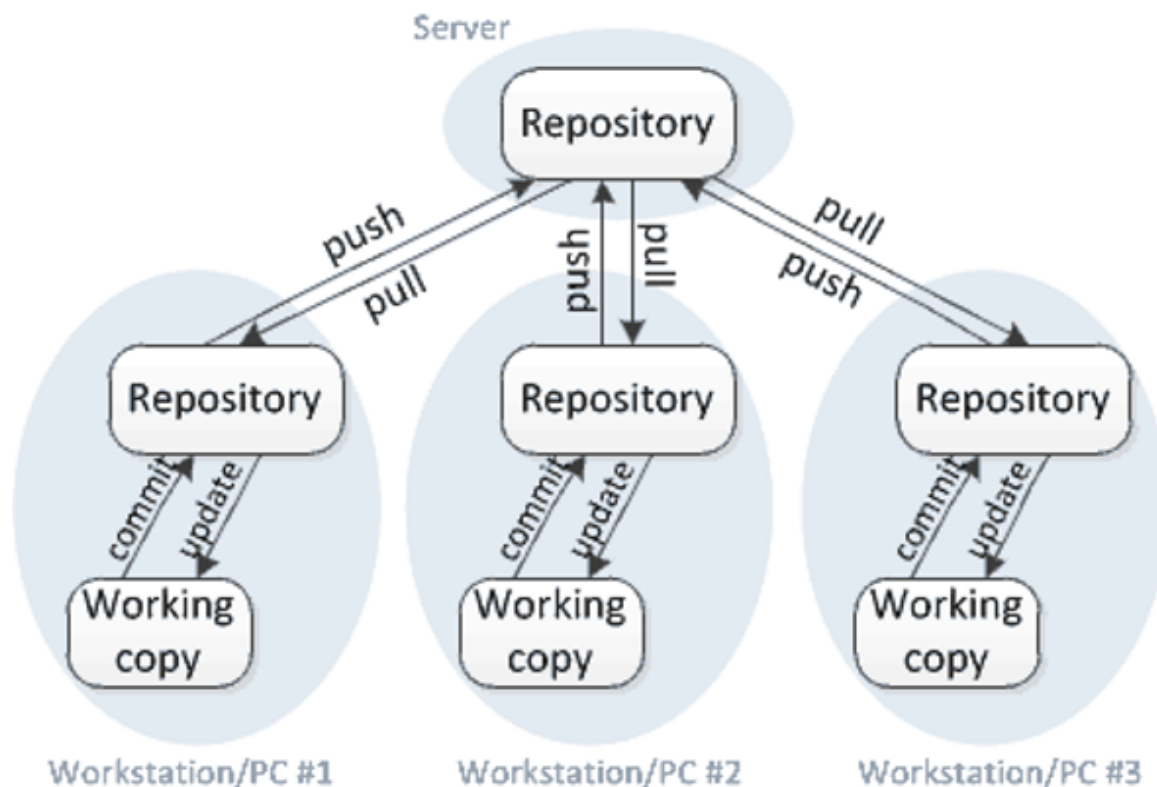


Image from <https://homes.cs.washington.edu/~mernst/advice/version-control.html>
 (<https://homes.cs.washington.edu/~mernst/advice/version-control.html>)

What is GitHub?

It is important to distinguish between git and GitHub. git is the version control system - the tool needed to keep track of your files/repositories history. On the other hand, GitHub is a hosting platform where you can keep your files/repositories in.

The best way to understand this clearly is by seeing this in action and working through an example together. There are several ways you can keep versions of your files through git and GitHub and we will be looking at two popular ways in this course; using the GitHub website and using Rstudio.

Version Control through the GitHub website

In this section we will use GitHub and manage our project online without using any coding.

The first step is to create a repository. These slides provide step-by-step instructions on how to do this (https://docs.google.com/presentation/d/e/2PACX-1vQ0VHWripOuP0wNye8q-HpIHfLGAQSi_QTlrcEAiET8kSx8g3XDY83201-OVg4muCdZlr4-CcQ21iFr/pub?start=false&loop=false&delayms=3000).

After this example we have now a repository in place where anyone can see the files.

Issues

In GitHub, Issues are created to suggest an idea, improvements or keep track of bugs, in other words, it can be viewed as a task list for the project. The steps involved to create an Issue on GitHub are here (http://docs.google.com/presentation/d/e/2PACX-1vRAAAzpAWQTFhcSxq4Vtz0Z_XrXam5ZykWXA56BaucHwCjrJeESPv37yjhD1SGDR1gcw9ac7rA2SUi/embed?start=false&loop=false&delayms=3000).

The GitHub flow

Making changes to a repository that only yourself is contributing to is one thing. However, when working in teams especially one which involves a large number of people where everyone is making changes to the same repository you want to have a stable **branch** in place.

A **branch** means a copy of your repository where you can make changes to without getting worried that you will do something wrong that will effect the remaining of the team. Therefore you are free to experiment on your own branch. There are several different workflows, e.g., git-flow, GitHub flow and GitLab flow.

It is not the scope of this course to get into details about the different software development versioning models. However, we will look briefly at GitHub flow which is one of the simple workflows available, with the aim to understand the basic constructs of software development versioning models so that we can adopt some good practices when sharing code or working in teams.

The GitHub flow steps consist of the following:

- **Anything in the master branch is deployable:** The master branch is the main branch which reflects what is deployed. Therefore the master should always be in a ready state and stable.
- **Create a branch:** If you want to make changes to the files present in the master branch create a new branch off of the master branch; we will refer to this as a feature branch. Name this feature branch with a descriptive name that reflects the change that you would like to do e.g. fix-issue1, add-chapter1. By creating a new feature branch off the master branch, you can safely do any changes you want without worrying of breaking the master branch. When you create a branch from your master branch, you're creating a copy of of the master repository where you can add your new features or try out new ideas without effecting the master repository.
- **Make changes to the files and then commit to the feature branch.** Adding commits to the feature branch is important as it keeps track of the updates that you do to your files and also the reason why they were made (make sure you write clear commit messages). The creation of a feature branch allows us to keep track of any changes that are done and compare the difference between the master and the feature branch. In this way you can have multiple people working on different things at the same time. As you have a stable master branch and all the other existant branches are fixes or updates that are work-in-progress to the project. Committing changes to the feature branch is also important as it helps you back up your work in case of a system failure or loss.
- **Open a pull request:** Once you have tested the changes you have applied on your code to make sure that everything is running well and you are happy with the changes/commits done on the feature branch, the next step is to propose these changes to be submitted into the master branch for someone to review your work. This is done by creating a **pull request**. You can use the @ment ion system on GitHub to bring this to attention to specific people.

- **Discussion and review:** The whole idea is that when you create a new pull request, this will enable a process of discussion and feedback about the changes being proposed back and forth between the maintainers of the master branch and yourself (the person that created the pull request). If you need to make additional changes to your feature branch after discussion, you can commit the changes to the feature branch again and then push the change. Pull requests can be written in Markdown so you can easily add images and other markdown formatting.
- **Merge:** Now that the code and files you have committed have passed all your testing, you can merge the changes to the existing master repository so that now they form part of the master repository. The good thing about all this process is that all the changes made during this process are preserved and anyone can track changes made in each merge and why they were made.

Let us do this whole process in action on GitHub! These steps are the steps required when creating a feature branch in the same repository and then finally merging them to the main repository

([http://docs.google.com/presentation/d/e/2PACX-](http://docs.google.com/presentation/d/e/2PACX-1vRfSuD6DfkG8LT130neS7tJQPpPz2TxtYkezlm_L0RUcQVyZRyjpI5yKBokxjC7Lg/embed?start=false&loop=false&delayms=3000)

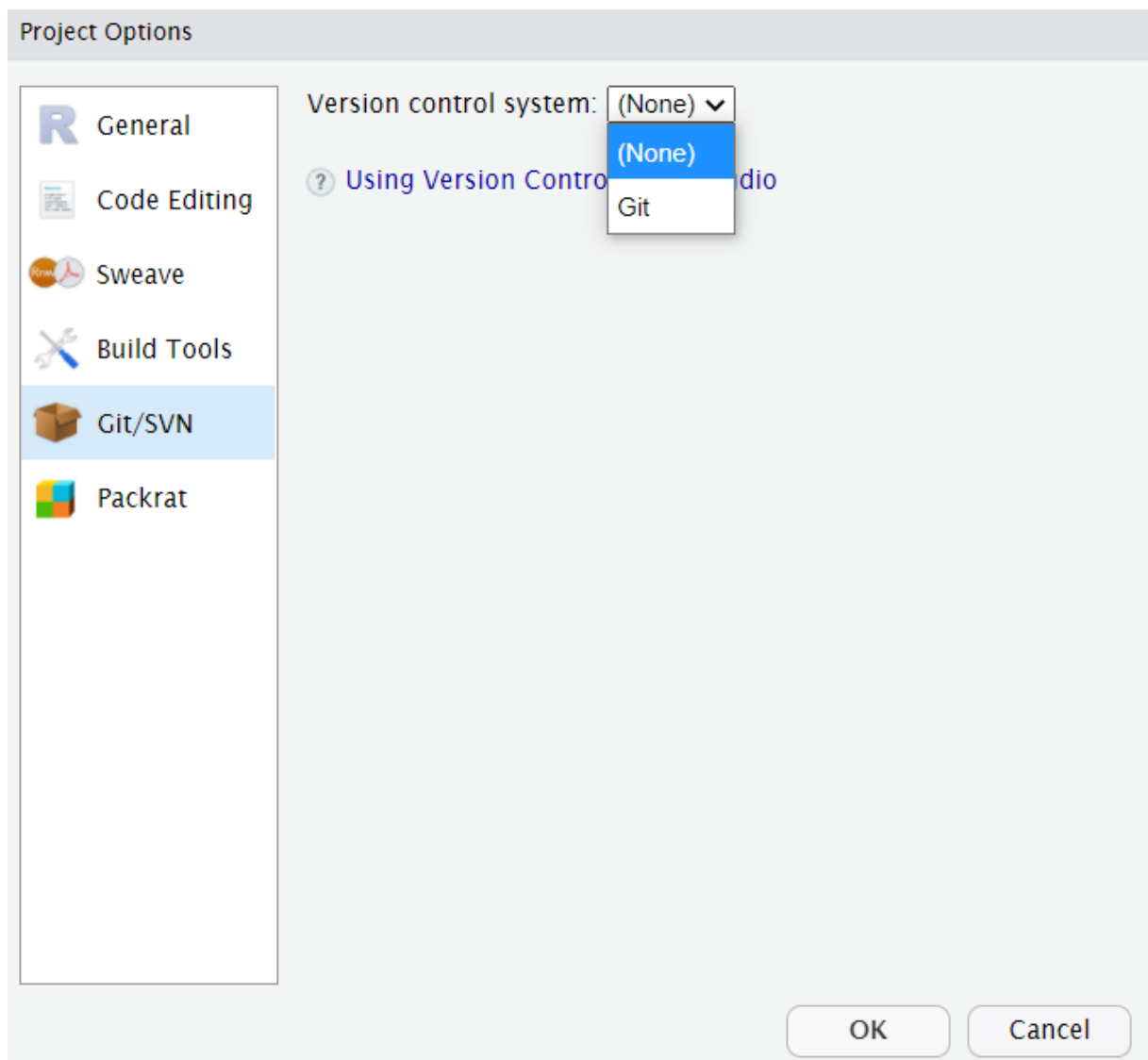
[1vRfSuD6DfkG8LT130neS7tJQPpPz2TxtYkezlm_L0RUcQVyZRyjpI5yKBokxjC7Lg/embed?start=false&loop=false&delayms=3000](http://docs.google.com/presentation/d/e/2PACX-1vRfSuD6DfkG8LT130neS7tJQPpPz2TxtYkezlm_L0RUcQVyZRyjpI5yKBokxjC7Lg/embed?start=false&loop=false&delayms=3000)).

Git with RStudio

So far we have learned how to do the main GitHub operations directly on GitHub. We will now look at how we can perform the same operations in RStudio. Why do you want to do this in RStudio? If you are writing your code in R, it will make more sense to use the inbuilt RStudio integration of git, in this way you do not need to switch back and forth to GitHub.

Let us use the R Markdown project we produced before. We would like to, firstly, make a copy of it on GitHub in a new repository, and secondly, we would like to use the git integration in RStudio so that any additional changes we make to it will be easily versioned in GitHub. To link the existing RStudio project with GitHub do the following steps:

1. Make sure you have the project you would like to upload on GitHub open in RStudio.
2. Go to Tools -> Version Control -> Project Setup and select Git from the Version control system dropdown menu.



3. If asked whether you want to initialise a new Git repository for this project, click Yes.
4. If asked whether RStudio needs to be restarted, click Yes.
5. Next we need to configure Git. Go to Tools > Shell. This should take you to the Terminal window in RStudio. Make sure you are first in the working directory of your project (use `getwd()` to check that). Then set the Git username and email using the following two commands:

```
git config --global user.name "yourGitHubUsername"  
git config --global user.email "name@provider.com"
```

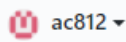
You can check if these are set up properly with `git config --global --list`

6. Next create a repository in GitHub.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *



Repository name *

/ first-notebook



Great repository names are short and memorable. Need inspiration? How about [urban-tribble?](#)

Description (optional)

My First Notebook Repository

☒ **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:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

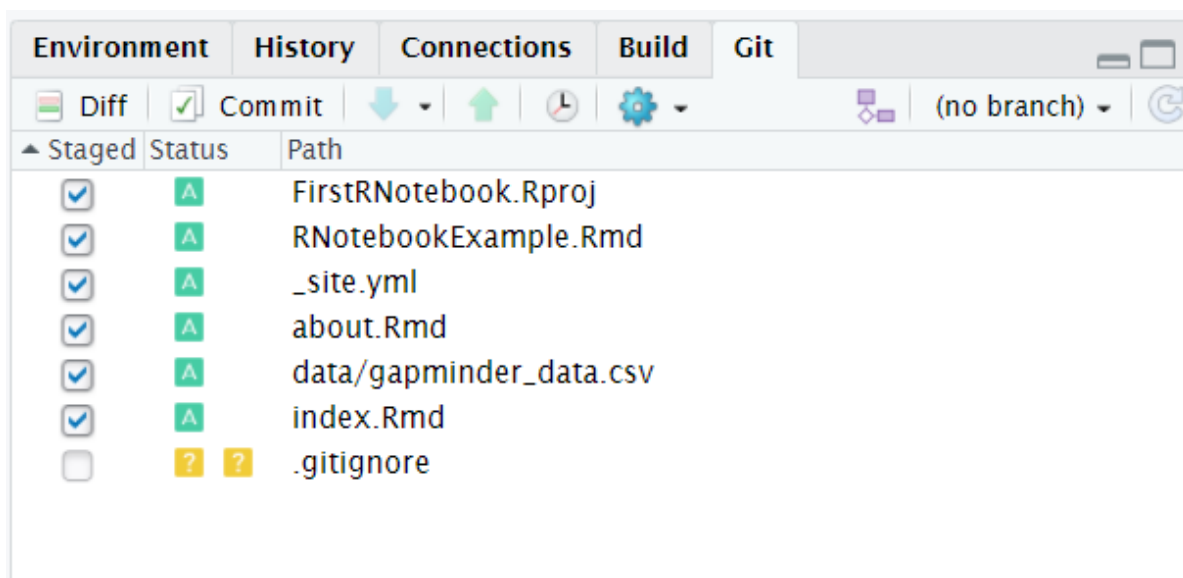
☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

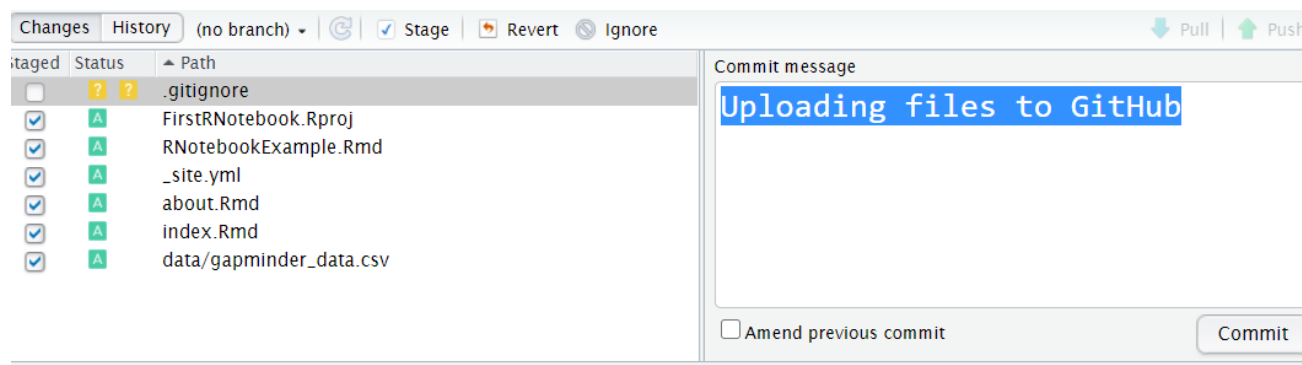
Create repository

7. After you click on the Create repository button, a Quick setup page is displayed which provides code for different scenarios. Copy the first line of the "...or push an existing repository from the command line" section and run it in the RStudio terminal. The line of code should look something like this: `git remote add origin https://github.com/ac812/first-notebook.git`

8. From the Git tab in RStudio, select the files you would like to commit to GitHub.



- Press the Commit button, enter a commit message and then press the Commit button in the new pop-up commit window.

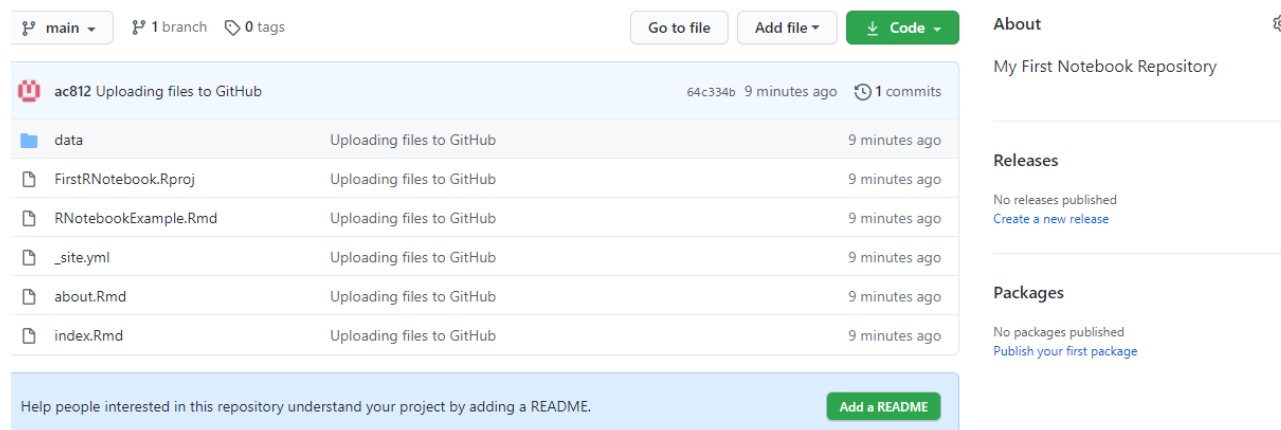


- Once you close the window you should not see the files committed in the Git tab of RStudio anymore.
- In the Terminal tab of RStudio run the two remaining commands displayed in the “...or push an existing repository from the command line” (see step 7). The lines should be similar to the code below. The first line creates a branch called `main` and the second line will push the committed changes to the main branch in the repository on github.

```
git branch -M main
git push -u origin main
```

- After running these two commands, your username and password might be requested, if so provide your GitHub username as the username and the Personal Access Token as the password. The Personal Access Token can be generated here (<https://github.com/settings/tokens>). Select ‘repo’ as scope.

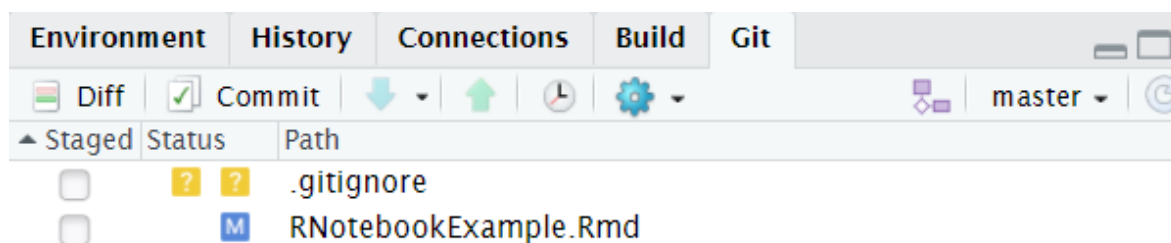
To check if the code has been uploaded successfully go to GitHub and check if the repository contains the files that have been pushed.



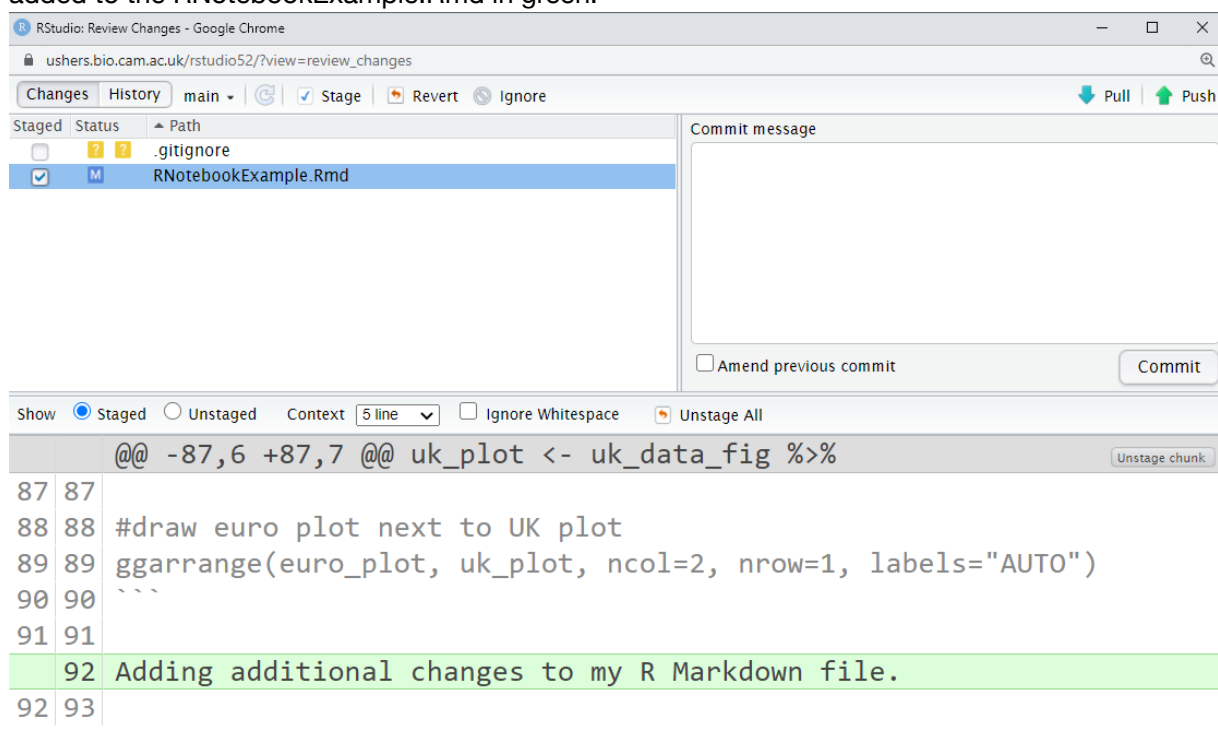
Keeping your local code updated on GitHub

From now onwards, if you continue making changes in your code in your RStudio project, these will be tracked and you will be able to see the files that had changes in them in the Git tab of RStudio. You will then be able to commit and push the changes to GitHub so that you have a backup of your code there. Let us try to make a few changes in our code and do the process together.

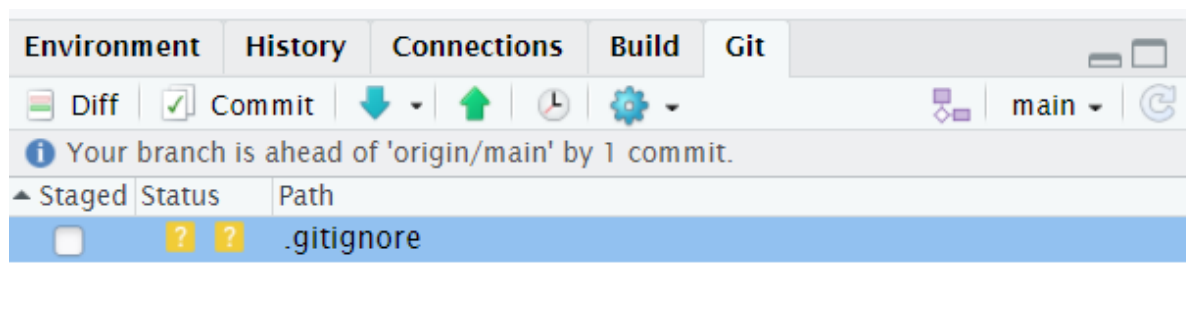
- Add some text at the end of the RNotebookExample.Rmd file and save it. This will make the file appear in the Git tab of RStudio with a status of M (meaning Modified)




- Next click on the Commit button, this will pop up a Commit window. Select the file you would like to commit from the list. This should display the changes that were done to the file. The parts highlighted in green are the parts that were added to file. Text highlighted in red would be text that was removed from the file. In this case since we have only added text, we can see the text that was added to the RNotebookExample.Rmd in green.



- Write a suitable commit message and press commit. Close the commit window. If you look at the Git tab in RStudio you will see that since the file was committed, the file does not appear anymore in the Git tab. However, there is a note saying that "Your branch is ahead of 'origin/main' by 1 commit" - meaning that there are changes committed in your local Git branch that are not present in your remote main branch (origin/main is your remote branch, the one you can see in your GitHub repository).



- To push the local commit to the remote main branch, press the green up arrow in the RStudio Git tab . If RStudio asks for your username or password provide them. Once the operation is complete your code should have been now updated on your GitHub repository (refresh your GitHub repository to see the change). If you click on the file in GitHub you can also see its contents and check that the changes are there.

(Optional) Working in a Team

When you are working on someone else's repository, you would need to first **fork** the repository, *i.e.*, make a copy of someone else's repository into your account so that you now have the same repository present in your list of repositories. You would then need to go over the GitHub flow. Below are the steps on how this can be done. The example shows 2 people working together, one is a Maintainer and one is a Developer. The roles are made on the top right hand side of the slides. Try this example in groups of 2 (https://docs.google.com/presentation/d/e/2PACX-1vQPhU3xOFvFnb-gNhrW2dotoz7BlcrKaBe6fpTMjvBbp3egSxtZUw9J6UcfwuZV21m_JO4Xd5BtQ0Ne/embed?start=true&loop=true&delayms=3000).