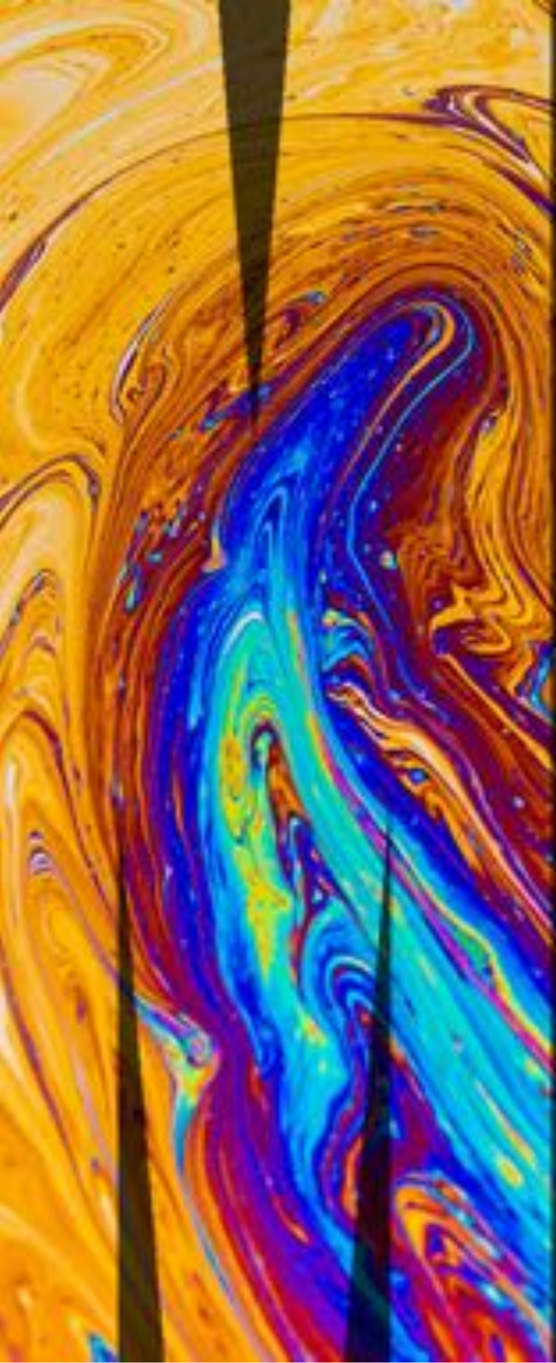


An Intro to Git and GitHub

A crash course

Dr Patricia Menéndez

Department of Econometrics and Business Statistics
patricia.menendez@monash.edu
@PM_maths

What is a version control system?

"Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later."

It also allow us to collaborate and share our projects with others!

Book on Git here - fantastic resource.

Version control

- Version control systems are a category of software tools that help store and manage changes to source code (projects) over time.
- Version control software keeps track of every modification to the source code in a special kind of database.
- If a mistake is made, you can turn back to previous versions and compare the code to fix the problem while minimizing disruption.
- It is easy to manage multiple versions of a project
- It is a very useful (actually essential!) tool for collaborating and for sharing open source resources.

Different Version control Systems

Distributed version control systems

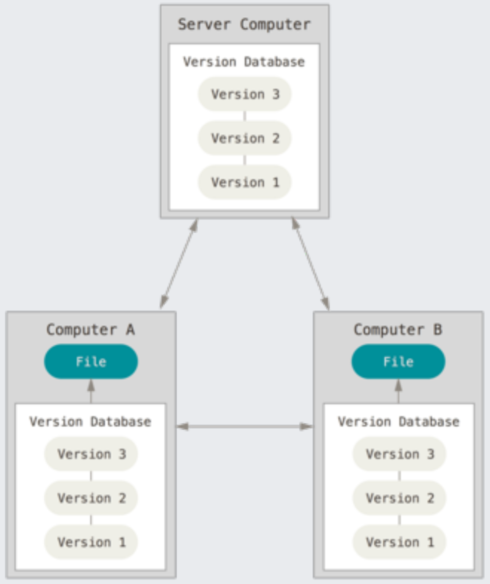


Figure source & more info

- These systems --> fully mirror the repository, including its full history in various servers/locations
- If any server malfunction, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.
- Every clone is really a full backup of all the data.

Distributed Version Control: Git

We are going to use a distributed version control called Git

Git for us

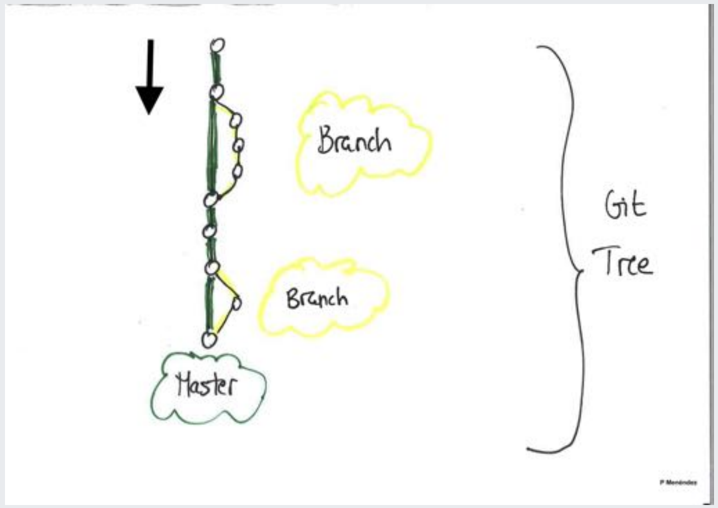
- A system for controlling our project versions
  - A disaster recovery system
  - A synchronization service
  - A platform for disseminating our work
  - A tool for collaboration
  - ...

More on Git here

Git overview in a nutshell

Let's think of the connections between the different versions of an R project as a tree (Git tree).

- Git tree example.
- White circles represent each version of the project.
- We have what we call master/main (default branch).
- We have branches that appeared and then merged with the master branch.

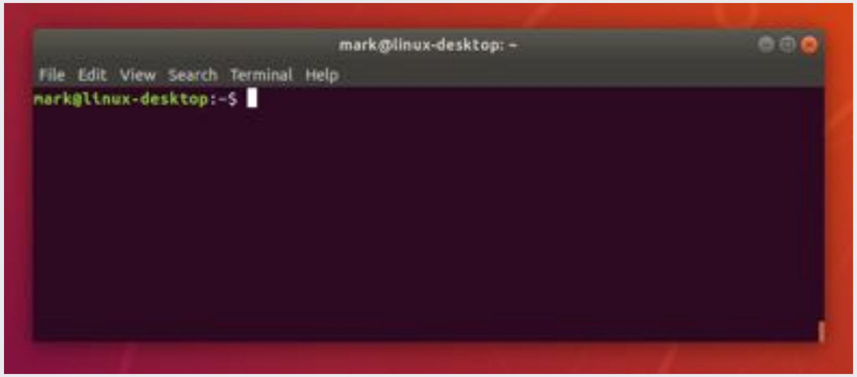


We need to learn

- How Git operates --> shell/command line
- How to connect our R projects to a Git repo
- How to connect our local Git repo to a Git Cloud repository (GitHub).

Command Line Interface (cli)

- In most cases (non-linux users) use a Graphical User Interface (GUI) to interact with their programs and operating systems $\longrightarrow$ for example Windows, Mac OX
- However, at the beginning of the computing times most people would use the command line interface to interact with their computer



```
mark@linux-desktop:~$ 
```

Git and Command Line

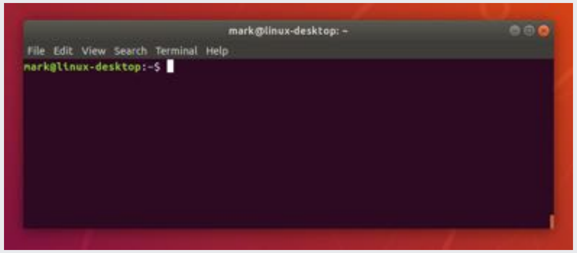
Learn how to use the shell/command line interface

Why?

- We will use the command line interface to interact with Git and with Github
- The shell or command line interface is an interface where the user types commands.
- This interface allow us to control our computer using commands entered via our keyboard.
- That means that instead of using a graphical user interface (GUI) + our mouse to open and close programs, create folders and moving files --> we are going to type commands.

Command Line Interface

Also known as the Shell, command line interface (cli) or terminal is an interface for **typing** commands to interact directly with a computer's operating system.



```
mark@linux-desktop:~$
```

Examples of things that we can do from the shell or terminal:

- Navigating through folders and files
- Create/delete folders
- Run and install programs (i.e interact with Git)
- And much more :-)!

Terminal in action

Typically when you open your terminal, it will welcome you with a prompt that looks like this:

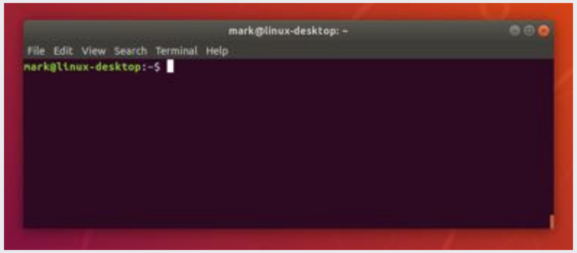
```
patricia@computerid-macbook:~$
```

or with the new Catalina Mac OX

```
patricia@computerid ~ %
```

On Windows it will contain the same elements but look like this:

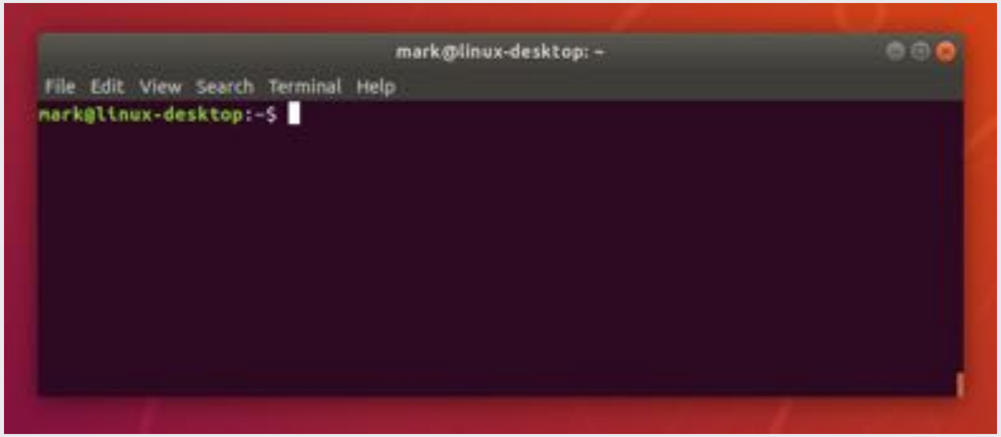
```
patricia@computerid-pc MINGW64 ~$
```



```
mark@linux-desktop:~$
```

Terminal in practice

We will start writing commands after ~$ or ~% depending on the terminal version that you are using



```
mark@linux-desktop:~$ 
```

- The commands that we are going to use are the same regardless the terminal version you have.

Let's start learning the commands to navigate our computer and to interact with Git via the command line interface

What is the path to my current computer location?

- **pwd** --> print working directory/present working directory

```
patricia@computerid ~ % pwd

/Users/patricia/Documents/ETC5513
```

Understanding the output of pwd command:

```
/Users/patricia/ETC5513/GitHub
```

- / --> represents the root directory
- Users --> is the Users directory
- patricia --> refers to my directory or folder within the users directory
- ...

Contents within the location --> directory

- Listing the contents of the directory or folder where I am:

ls --> list files inside my directory

```
patricia@computerid Documents~ % ls
```

```
Courses Research Teaching file.pdf example.txt
```

- Documents is an argument to the **ls** command.
- **ls** $\rightarrow$ gives you a list of all the elements in a directory
- **ls -a** $\rightarrow$ list of all the files including hidden ones.

Each Linux command (pwd,ls ...) have lots of options (flags) that can be added.

Command Line Basics: Navigating between directories

cd --> change directory

- First we need to make sure where we are **pwd**. We can also use **pwd** to figure how we can get where we want to go!

- The **cd** command syntax is very simple --> we just need to specify the directory that we want to navigate to

- At any moment of your navigation, you can use the **pwd** command to confirm your current location
- A path that starts with / is assumed to be absolute.

cd in practice!

My current location is

Documents (relative location --> Documents/Research/COVID):

- **cd Research** $\rightarrow$ means that we move into Research
- **cd COVID** $\rightarrow$ means that we move into COVID
- **cd .** $\rightarrow$ means the current directory COVID
- **cd ..** $\rightarrow$ means (parent directory) that we move back into Documents
- The **~** symbol is a shorthand for the user's home directory and we can use it to form paths:
  - If you are in your Downloads directory (/Users/John/Downloads) typing **cd ~** $\rightarrow$ will bring you to your Home directory /Users/John!

More commands practice!

My current location is

COVID (inside we have --> Documents/Research/COVID):

- **..** is shorthand for "the parent of the current working directory".
- **cd ..** $\rightarrow$ means that we move into Research (1 directory up). That is from COVID back to Research
- **mkdir Project1 Project2** means "make two new directories (folders)"called Project1 and Project2.

More commands practice!

- **mv** move files or folders $\rightarrow$ takes two arguments, the first being files or folders to move and the second being the path to move to.
- **cp** $\rightarrow$ this command is used to copy files or group of files or directories. When copy files we need to use **cp -r** to copy all the directory contents.
- **rm** $\rightarrow$ remove files and folders
- To remove entire folders **rm -r** $\rightarrow$ It requires the -r (recursive) flag.
- We can create empty files with **touch** example.Rmd

Cheat sheet for command line

Excellent summary about the commands that we will be using can be found here

You don't need to learn all the commands only those that we are going to use!

Let's get some practice using the terminal

Demo

Files in a Git repository

The states in a Git repository are: the working directory, the staging area and the git directory:

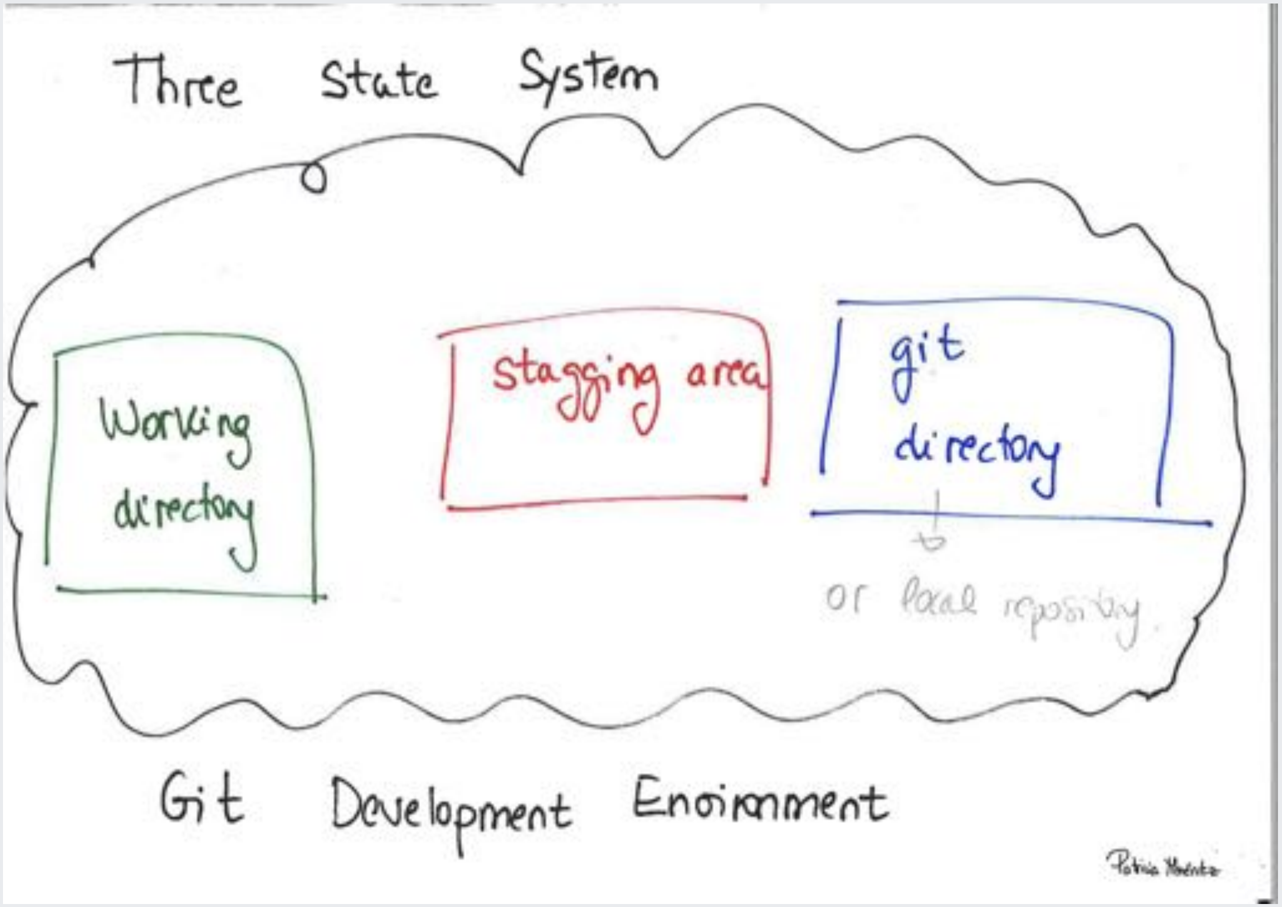
- The working directory is the current snapshot that you are working on.
- The staging area is where modified files are marked in their current version ready to be stored in the database $\longrightarrow$ index of changes
- The git directory is the database where the history is stored

In your file system you will see the folder of your project

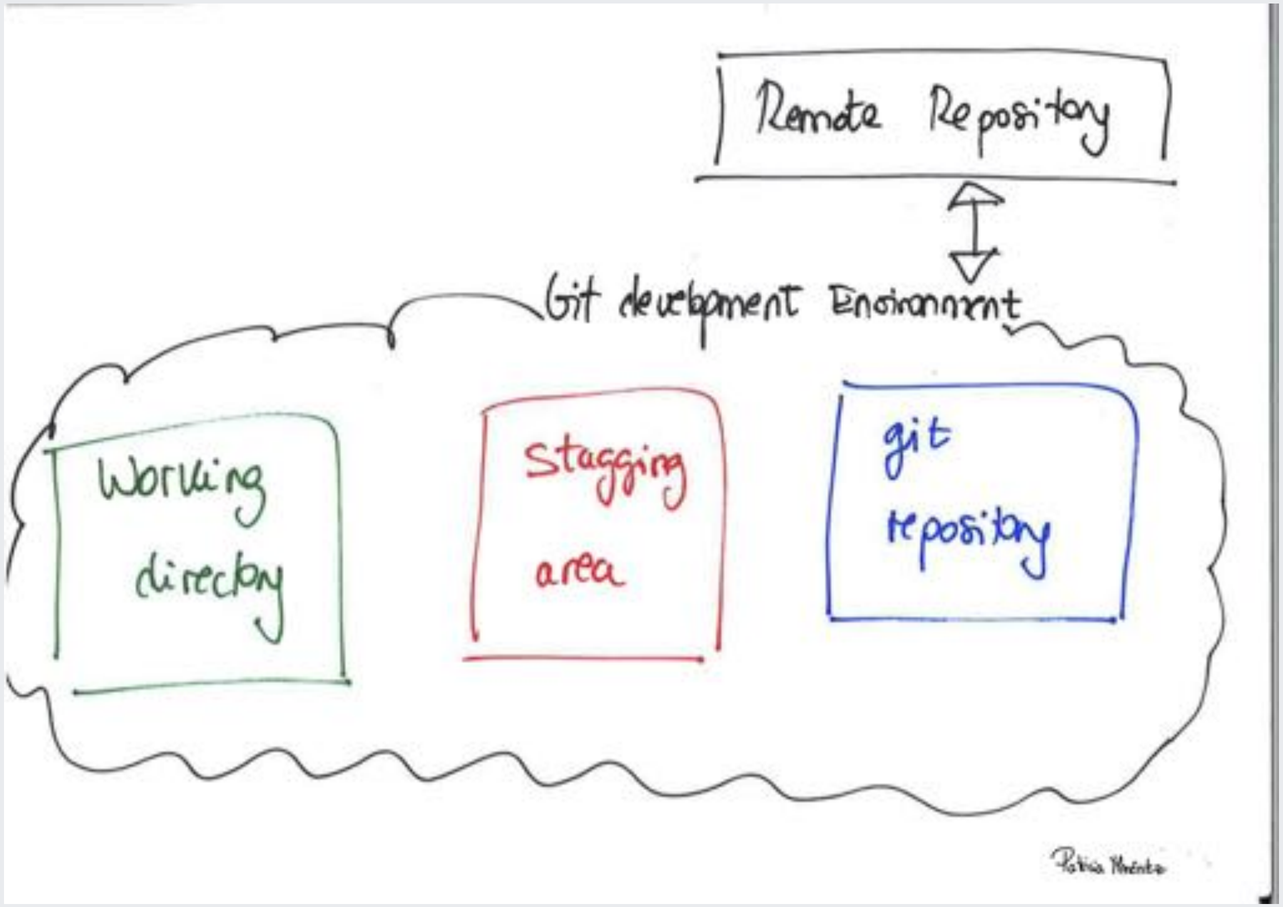
Three state system ~ Git Development Environment

More info here Begining Git and Github

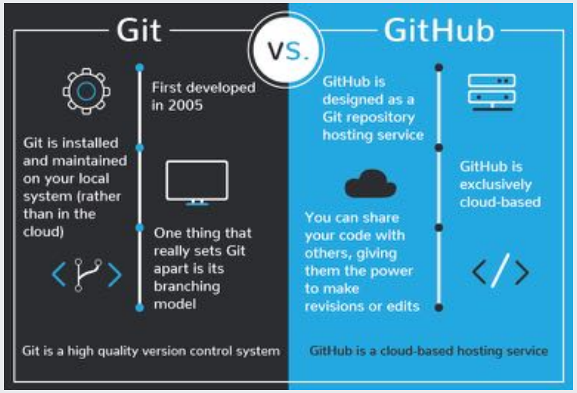
Components of a Git repository: Visually



Git repo and remote repository



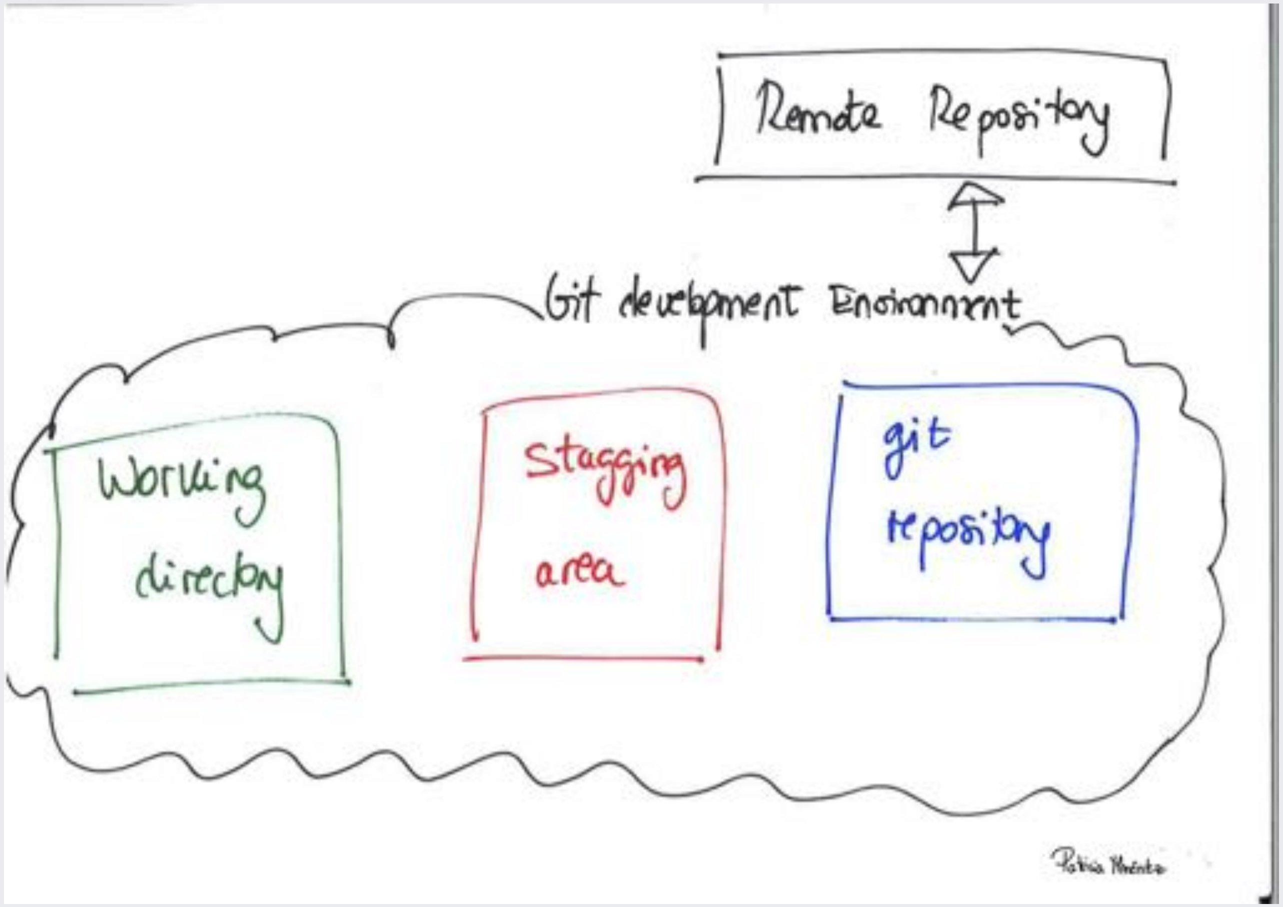
GitHub is our remote repository


Git
VS.
GitHub
First developed in 2005
Git is installed and maintained on your local system (rather than in the cloud)
One thing that really sets Git apart is its branching model
Git is a high quality version control system
GitHub is designed as a Git repository hosting service
GitHub is exclusively cloud-based
You can share your code with others, giving them the power to make revisions or edits
GitHub is a cloud-based hosting service

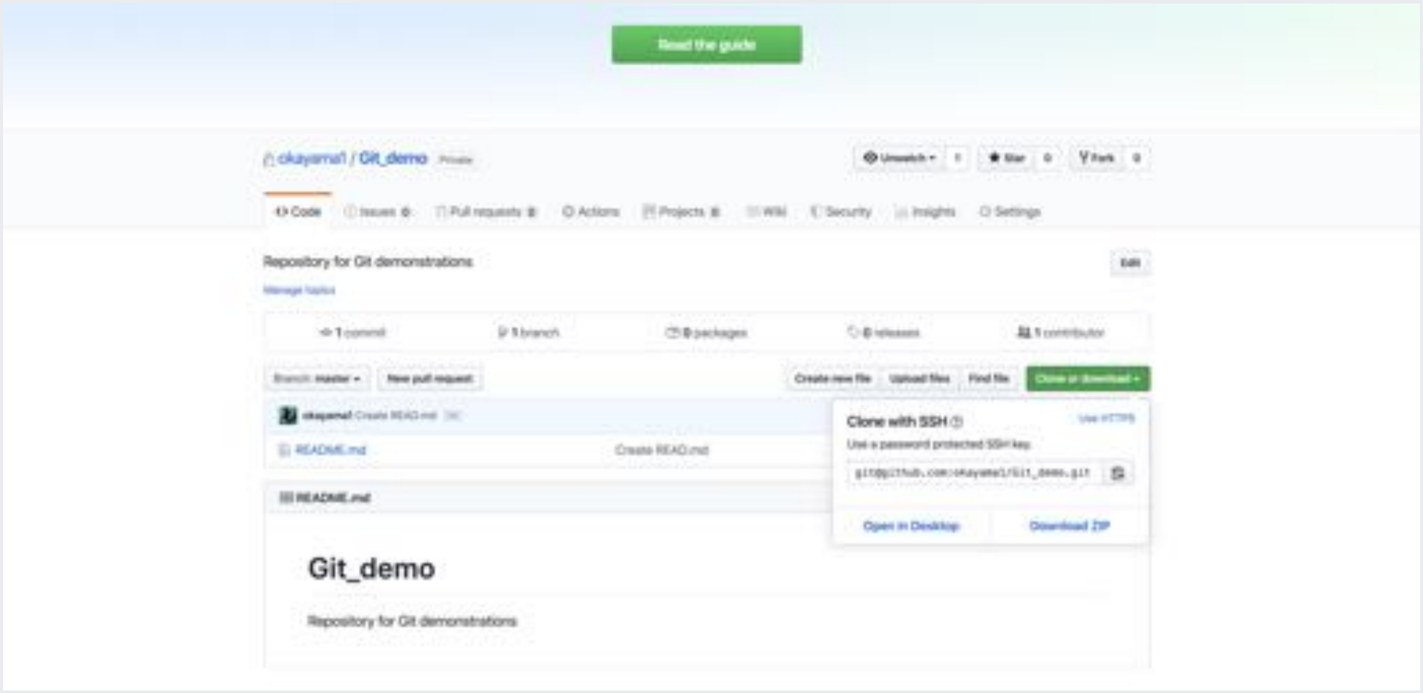
- GitHub is an interface and cloud hosting service built on top of the Git version control system.
- Git does the version control and Github stores the data remotely.
- GitHub makes your projects accessible on a fully feature project website

More info about Github here

Git repo and a remote repository



GitHub is our remote repository



Repository for Git demonstrations

okayama1 Create README.md

README.md Create README.md

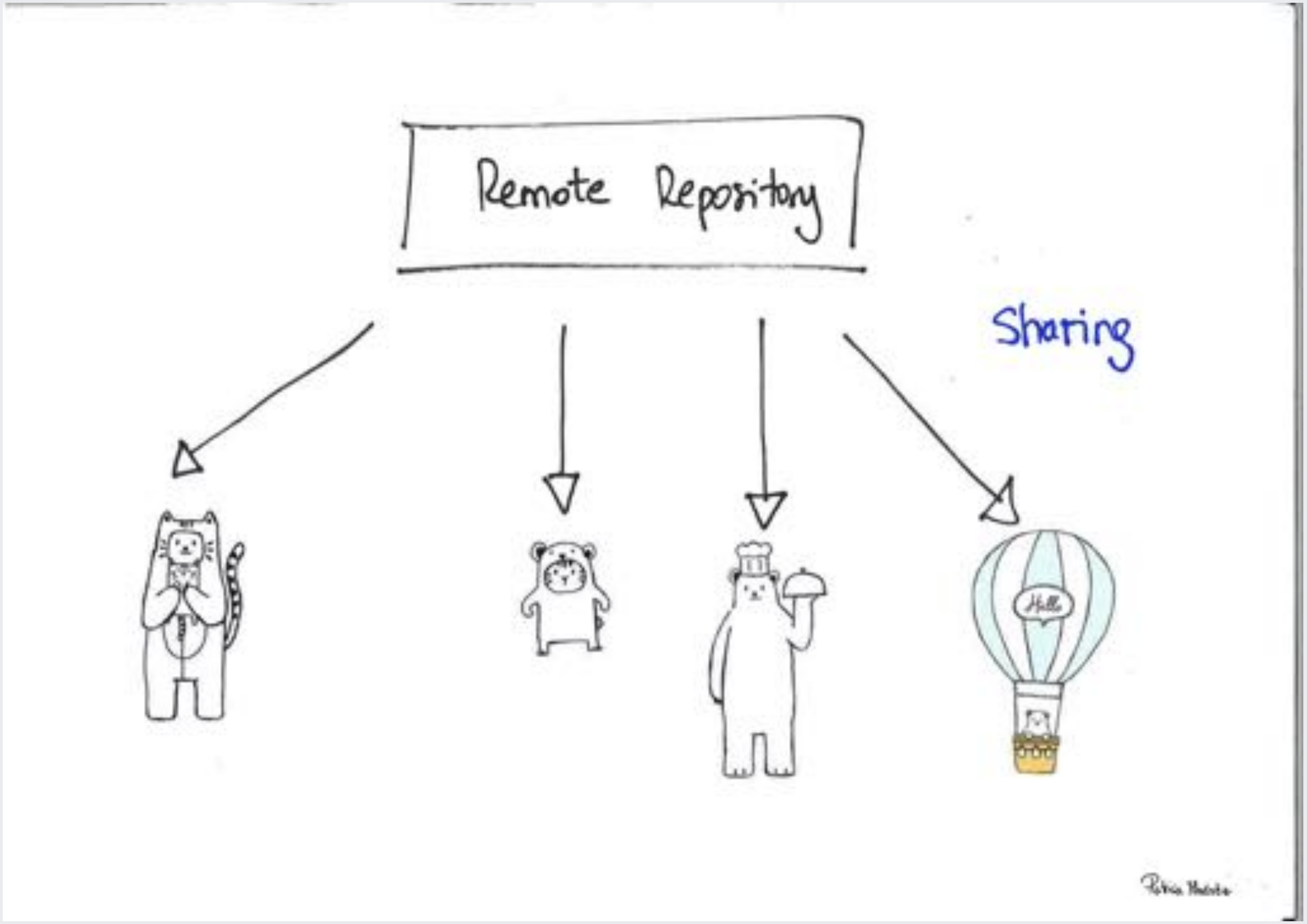
Use a password protected SSH key.

```
git@github.com:okayama1/Git_demo.git
```

Git_demo

Repository for Git demonstrations

Collaborative



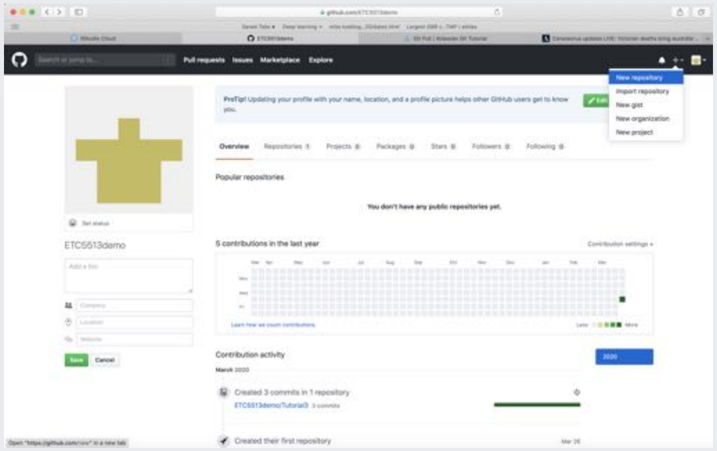
From GitHub <--> to our computer

- Create a repository (repo) on https://www.github.com
- Clone this GitHub repository into our computer $\rightarrow$ making a 'local copy'.
- Work on our local copy of the repo
- Stage and Commit changes to local repository
- Push those changes into the remote repo in GitHub.

Let's look at all these in more detail!

Creating a repo on GitHub

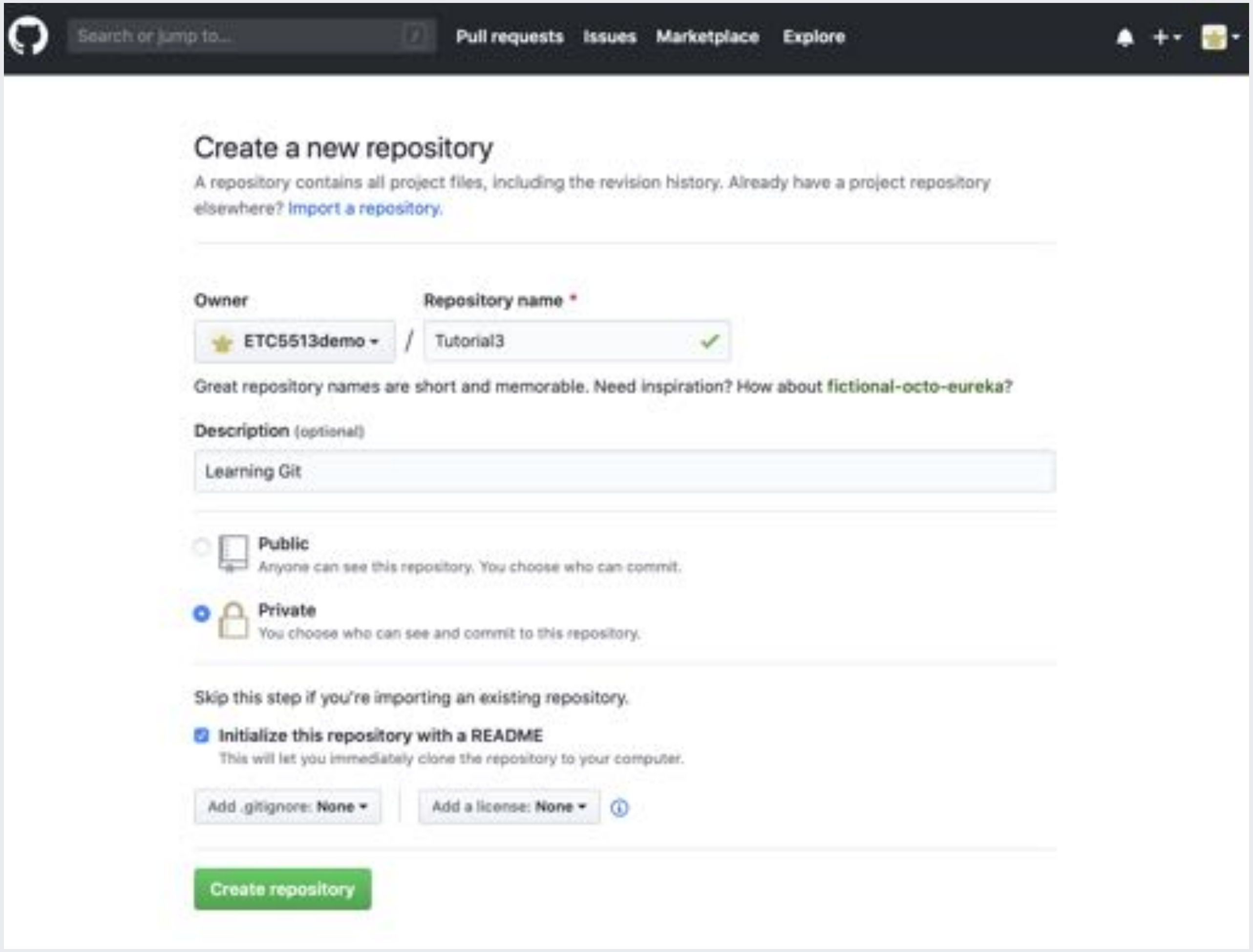
1. Login into GitHub
2. Click the ‘+’ icon on the top right on the menu bar and select ‘New Repository’.



Important:

- Repo name
- Repo --> public or private
- Make sure it is initialized with REAME.md: It is important to have a README.md file for every repository. GitHub will use this file as the "presentation" of the repository and should briefly describe what the repo is about.

Creating a repo on 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: ETC5513demo / Repository name *: Tutorial3

Great repository names are short and memorable. Need inspiration? How about fictional-octo-eureka?

Description (optional): Learning Git

Public
Anyone can see this repository. You choose who can commit.

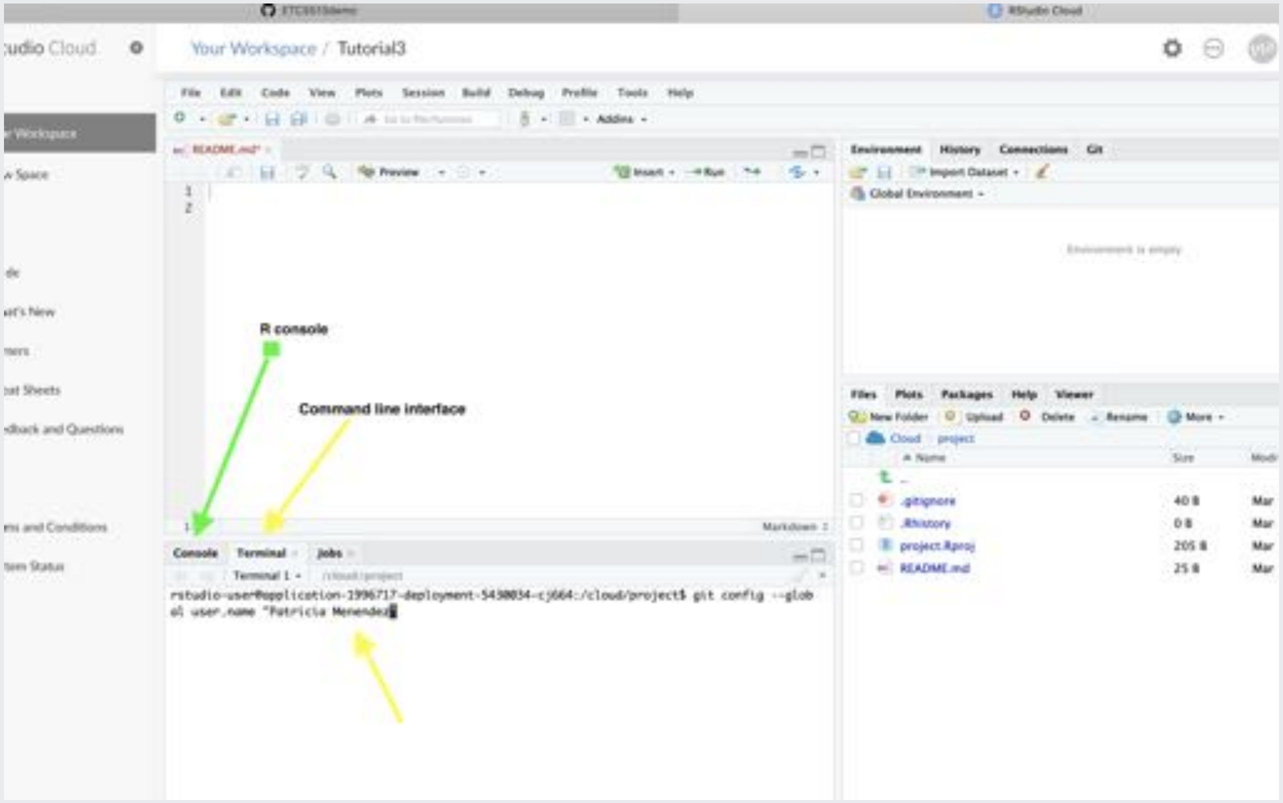
[✓ Private]
You choose who can see and commit to this repository.

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

[✓] Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None | Add a license: None

Configuring Git in your Rstudio project using the terminal



```
rstudio-user@application-1996717-deployment-5430034-cj664:/cloud/project$ git config --glob
al user.name "Patricia Menendez
```

Configuring Git in your Rstudio Cloud project

First of all we need to get your Git configured in Rstudio Cloud (the same follows for your own computer):

- **Open the command line interface/terminal interface (CLI)** and type:

- git config --global user.email "your.email@example.com"

- git config --global user.name "Your_Firstname Lastname"

Make sure you use the same email address for this and for setting up your GitHub account.

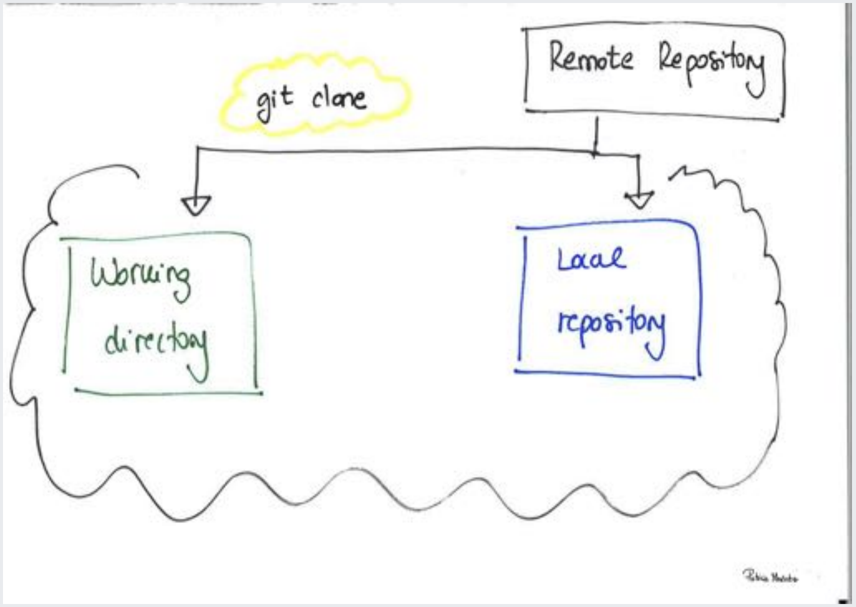
- To check that everything is set up correctly, **type the following in the CLI**: git config --global user.email

Configuring Git in your Rstudio Cloud project using the R console

Alternative you could type the following in your **R console" inside Rstudio Cloud**:

```
usethis::use_git_config(user.name = "Your Name Surname",
user.email = "Your monash email")
```

Getting a remote repository

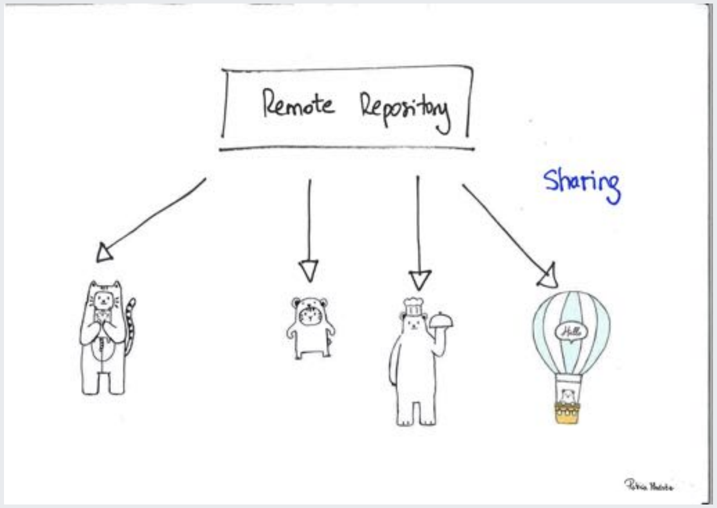


- Grabs remote repository from a server/cloud (i.e github)
- Creates a new folder (copy of the remote repository) in our computer

Clonning a github repo

When you create a repository on GitHub, it exists as a remote repository.

Users can clone your repository to create a local copy on their own computer and sync between the two locations.



More detailed info here

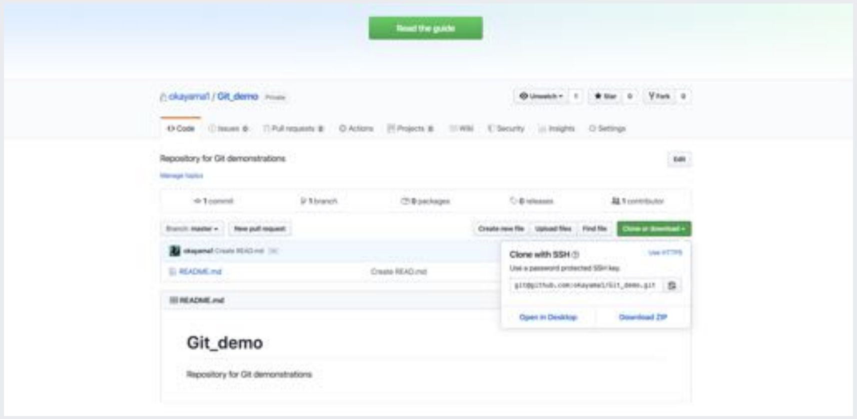
Adding an SHH key into your GitHub

It is very useful to add an SSH key into your workflow:

- When working with a Git repository you will be required to identify yourself to GitHub using your username and password *each time to do a commit!*.
- An SSH key is an alternative way to identify yourself that **does not require you to enter you username and password every time.**
- SSH keys come in pairs, a public key that gets shared with services like GitHub, and a private key that is stored only on your computer or in your Rstudio Cloud project.
- If the keys match, you're granted access!

More info here and here

From GitHub to our computer



Repository for Git demonstrations

okayama1 Create README.md

README.md Create README.md

Use a password protected SSH key.

```
git@github.com:okayama1/Git_demo.git
```

From our shell or command line:

1. Navigate to the computer location where we want to download the GitHub repo.
2. **git clone** git@github.com:okayama1/Git_demo.git.
3. This will create a folder in your computer with the GitHub repository files and folders.

Three Git States

Git has three main states that your files can reside in: modified, staged, and committed:

- **Modified** --> you have changed the file but have not committed it to your repository database yet.
- **Staged** --> you have marked a modified file in its current version to go into your next commit snapshot.
- **Committed** --> the data is safely stored in your local database.

This leads us to the three main sections of a Git project: the working tree, the staging area, and the Git directory.

Three Git States Visually

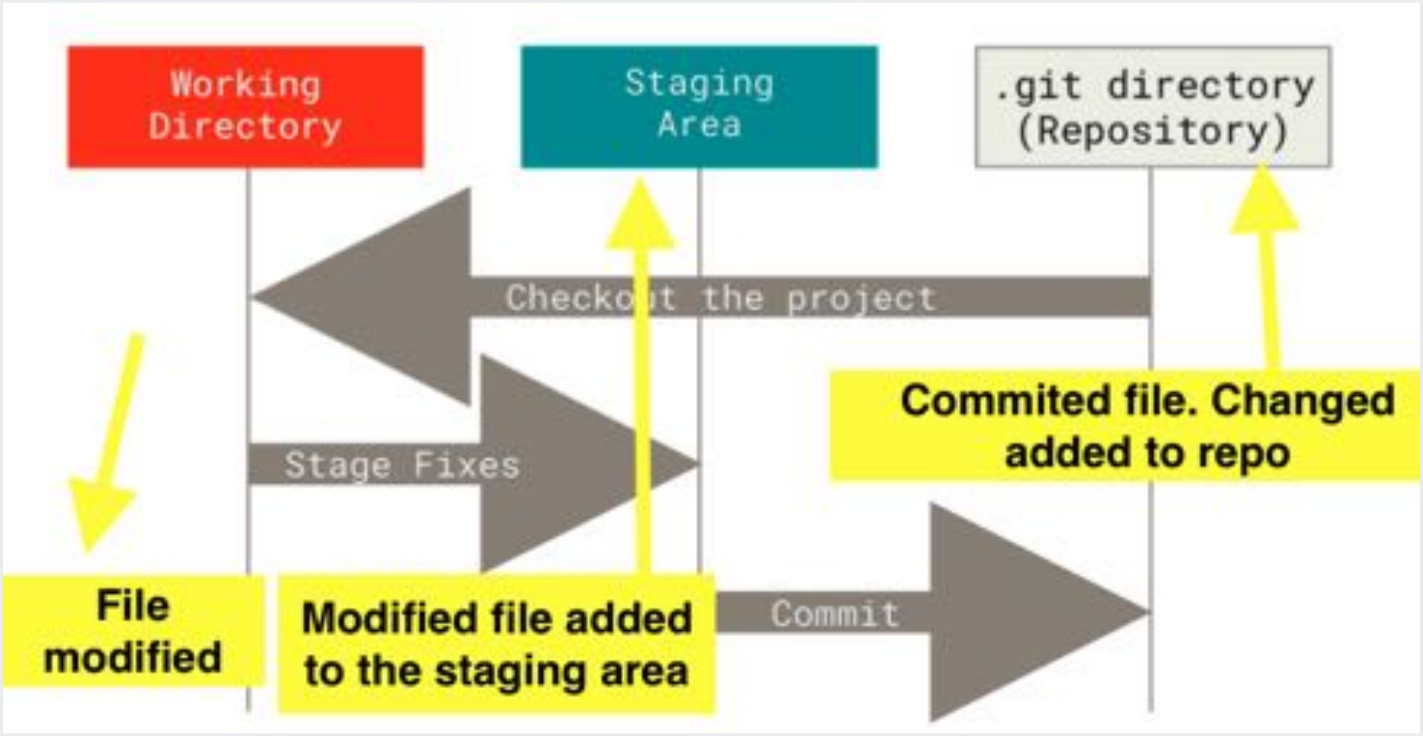
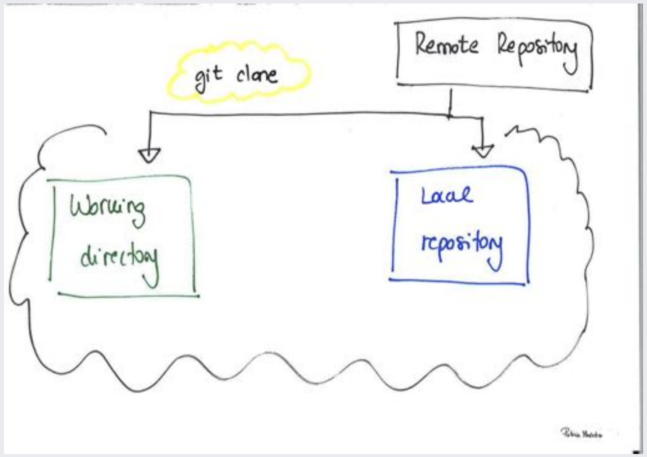
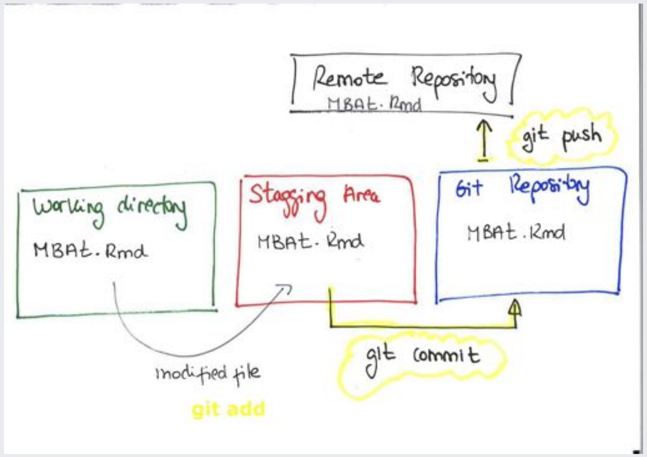

Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit
File modified
Modified file added to the staging area
Commited file. Changed added to repo

Figure source

GitHub Workflow: Visual example



clone info --> GitHub repo



Working in your computer and updating remote repo in GitHub

Tracked and untracked files

- In a git repository tracked files are those which are part of the git repository
- However, we can also have untracked files for which their history is not tracked

- Tracked files are files that were in the last snapshot; they can be unmodified, modified, or staged. In short, tracked files are files that Git knows about.
- Untracked files are everything else — any files in your working directory that were not in your last snapshot and are not in your staging area.

More info here.

From git clone to first commit

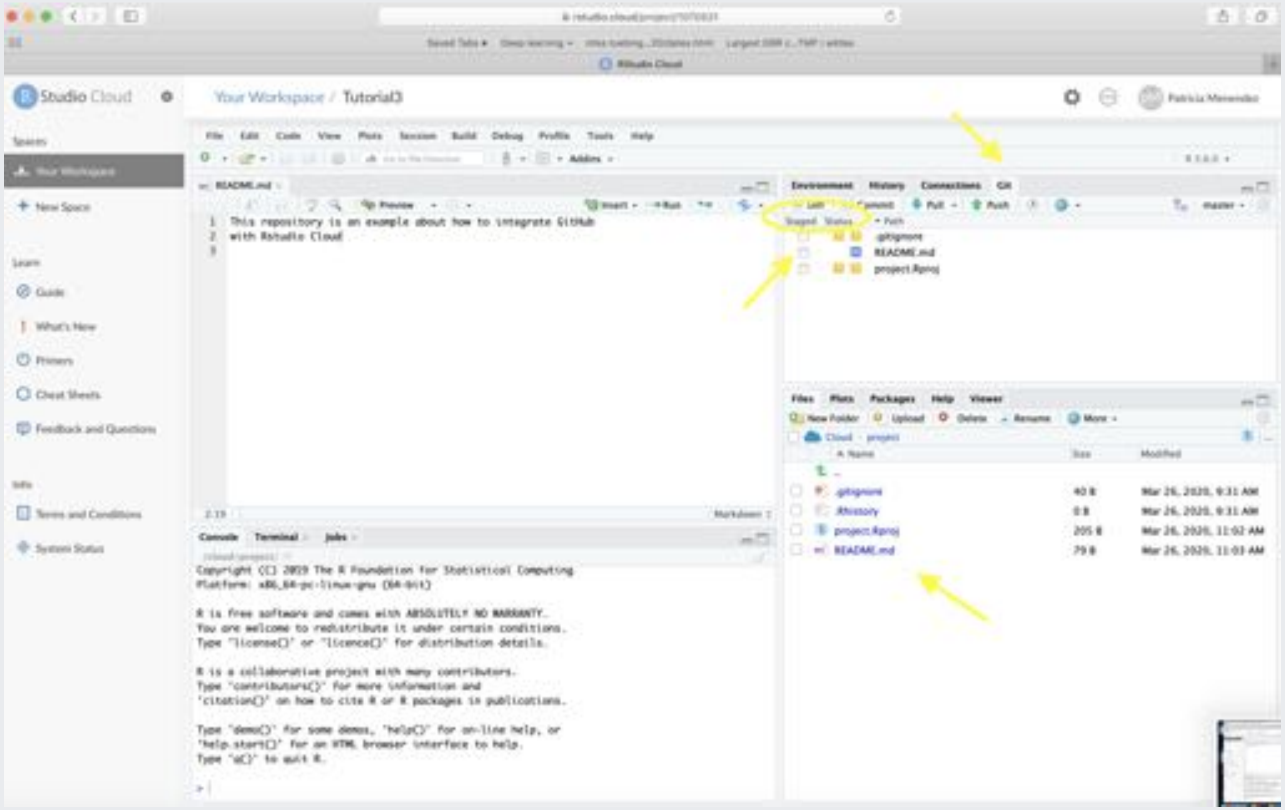
- git clone "remote repo address" $\longrightarrow$ is a Git command used to target an existing repository and create a clone (or copy) of the target repository.
- git add filename $\longrightarrow$ it adds a change in the working directory to the staging area.
- git commit -m "Message" The git commit command captures a snapshot of the project's currently staged changes. (m = message for commit. The git commit is used to create a snapshot of the staged changes along a timeline of a Git projects history.)
- git push origin master (or main) $\longrightarrow$ The git push command is used to upload local repository content to a remote repository, in this case to the master (or main) branch.

General Workflow (via Terminal)

- git clone is used to target an existing repository and create a clone, or copy of the target repository.
- git pull is used to fetch and download content from a remote repository and immediately update the local repository to match that content.
- git status displays the state of the working directory and the staging area
- git add file_name adds a change in the working directory to the staging area
- git commit -m "Message" (m = message for commit. The git commit is used to create a snapshot of the staged changes along a timeline of a Git projects history.)
- git push origin branch name is used to upload local repository content to a remote repository.

Rstudio and GitHub

The status/staging panel in Rstudio



```
This repository is an example about how to integrate GitHub
with Rstudio Cloud
```

```
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

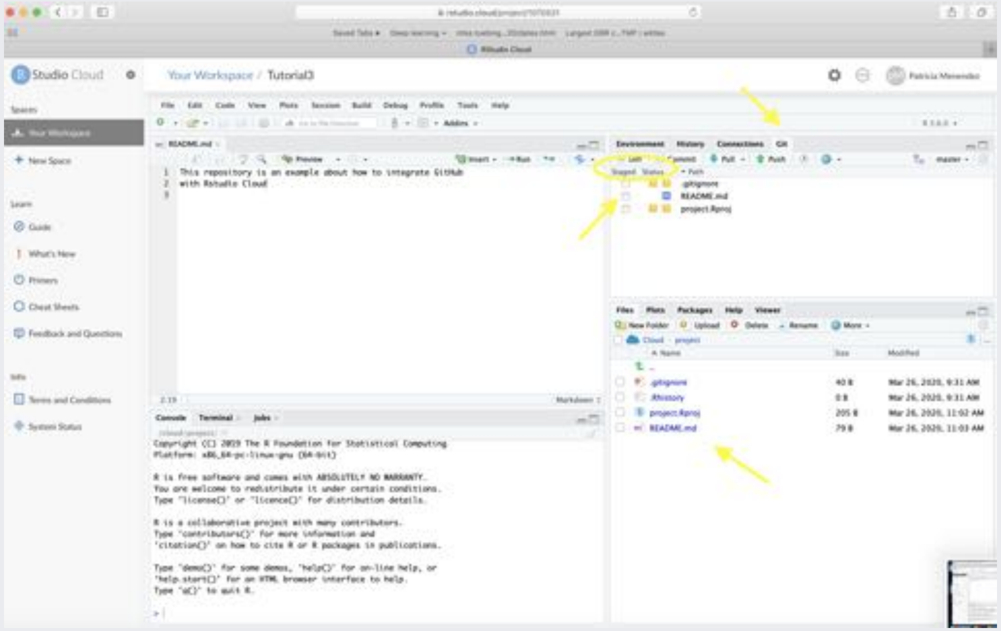
Staged	Status	Path
☐		.gitignore
☐		README.md
☐		project.Rproj

Name	Size	Modified
.gitignore	40 B	Mar 26, 2020, 9:31 AM
.Rhistory	0 B	Mar 26, 2020, 9:31 AM
project.Rproj	205 B	Mar 26, 2020, 11:02 AM
README.md	79 B	Mar 26, 2020, 11:03 AM

Rstudio and GitHub

sStudio keeps Git constantly scanning the project directory to find any files that have changed or which are new.

- By clicking a file's little "check-box" you can stage it.



```
This repository is an example about how to integrate GitHub
with Rstudio Cloud
```

Staged	Status	Path
☐		.gitignore
☐		README.md
☐		project.Rproj

Name	Size	Modified
.gitignore	40 B	Mar 26, 2020, 9:31 AM
.Rhistory	0 B	Mar 26, 2020, 9:31 AM
project.Rproj	205 B	Mar 26, 2020, 11:02 AM
README.md	79 B	Mar 26, 2020, 11:03 AM

```
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Studio Cloud
Your Workspace / Tutorial3
Patricia Menendez
Spaces
Your Workspace
New Space
Learn
Guide
What's New
Primers
Cheat Sheets
Feedback and Questions
Info
Terms and Conditions
System Status
File Edit Code View Plots Session Build Debug Profile Tools Help
README.md
Environment History Connections Git
Diff Commit Pull Push master
Files Plots Packages Help Viewer
New Folder Upload Delete Rename More
Cloud › project
Console Terminal Jobs
Markdown

Rstudio and GitHub

Understanding the symbols in the Rstudio Git pane:

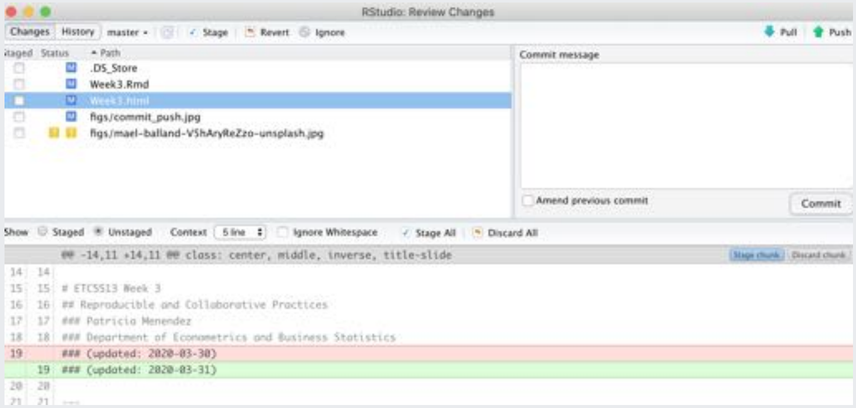
- **Blue-M**: a file that is already under version control that has been modified.
- **Orange-?**: a file that is not under version control (yet...)
- **Green-A**: a file that was not under version control, but which has been staged to be committed.
- **Red-D**: a file under version control has been deleted. To make it really disappear, you have to stage its disappearance and commit.
- **Purple-R**: a file that was renamed. (Note that git in Rstudio seems to be figuring this out on its own.)

Configuration

The **Diff** window

- Shows what has changed between the last committed version of a file and its current state.

- Note: all this output is available from the command line too, but the Rstudio interface is very nice!

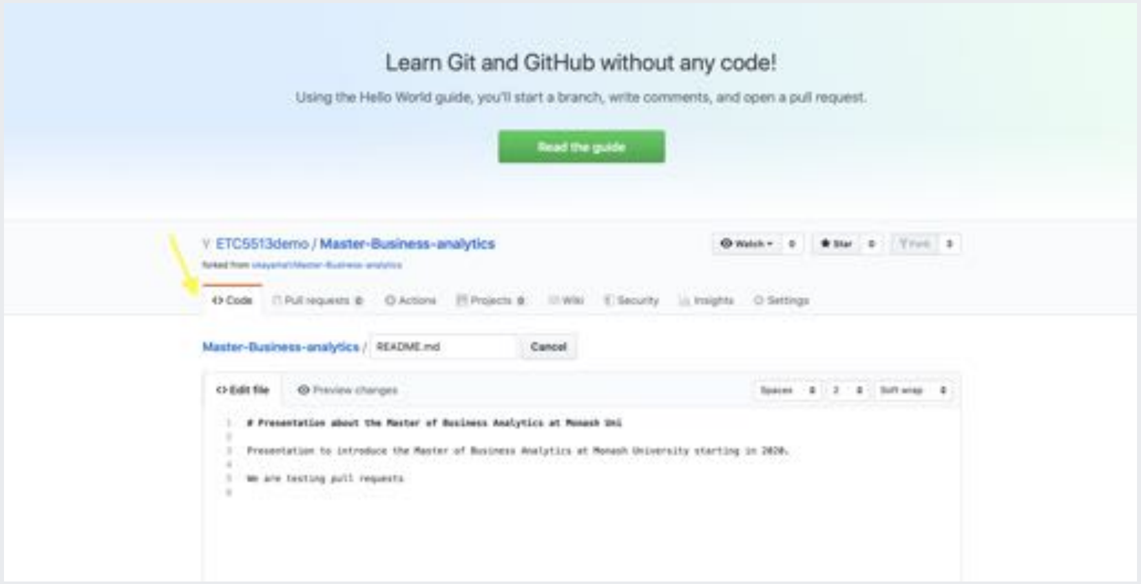


```
.DS_Store
Week3.Rmd
Week3.html
figs/commit_push.jpg
figs/mael-balland-V5hAryReZzo-unsplash.jpg
```

```
@@ -14,11 +14,11 @@ class: center, middle, inverse, title-slide

# ETC5513 Week 3
## Reproducible and Collaborative Practices
### Patricia Menendez
### Department of Econometrics and Business Statistics
### (updated: 2020-03-30)
### (updated: 2020-03-31)

---
```

Making a Commit

- Super easy:
  - After staging the files you want to commit...
  - Write a brief message (first line short, then as much after that as you want) and hit the commit button.



Learn Git and GitHub without any code!

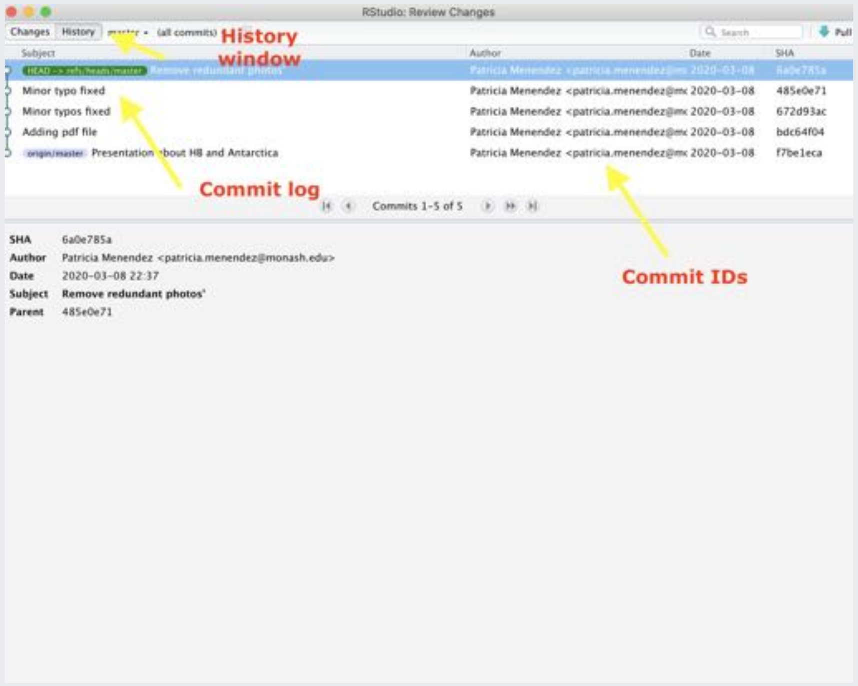
Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

```
# Presentation about the Master of Business Analytics at Monash Uni

Presentation to introduce the Master of Business Analytics at Monash University starting in 2020.

We are testing pull requests
```

The History window

Allow us to understand past commits.

- Easy inspection past commits.
- See what changes were made at each commit.



Subject	Author	Date	SHA
HEAD -> refs/heads/master Remove redundant photos'	Patricia Menendez <patricia.menendez@mc	2020-03-08	6a0e785a
Minor typo fixed	Patricia Menendez <patricia.menendez@mc	2020-03-08	485e0e71
Minor typos fixed	Patricia Menendez <patricia.menendez@mc	2020-03-08	672d93ac
Adding pdf file	Patricia Menendez <patricia.menendez@mc	2020-03-08	bdc64f04
origin/master Presentation about H8 and Antarctica	Patricia Menendez <patricia.menendez@mc	2020-03-08	f7be1eca

Commits 1-5 of 5

SHA	6a0e785a
Author	Patricia Menendez <patricia.menendez@monash.edu>
Date	2020-03-08 22:37
Subject	**Remove redundant photos'**
Parent	485e0e71

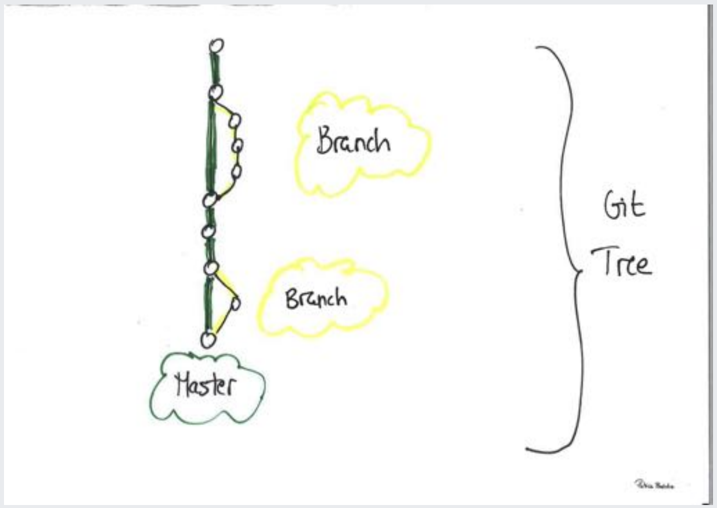
Important:

- We can interact between Git, GitHub and our local repository using the terminal only
- We can interact between Git, GitHub and our local repository using Rstudio

Branching

Each repository has one default branch, and can have multiple other branches. Branching is a great feature of version control!.

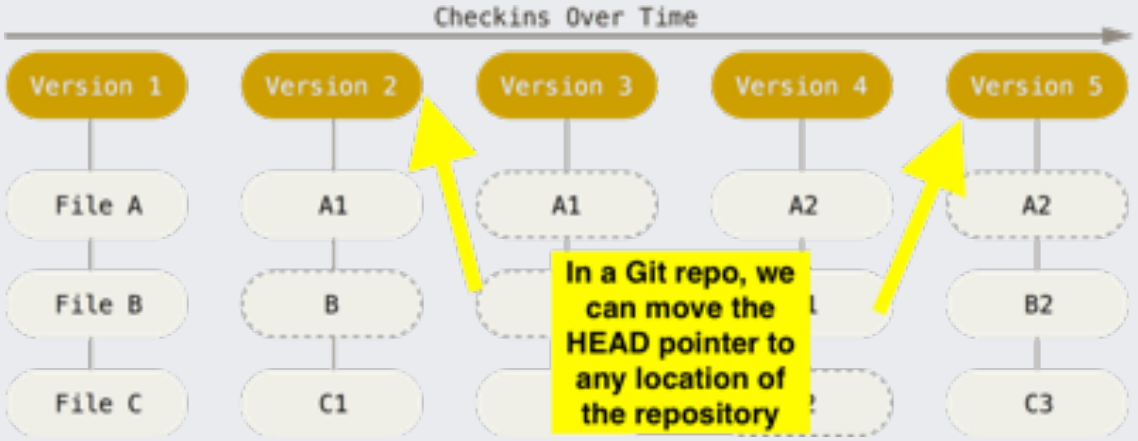
- It allows you to duplicate your existing repository
- Use a branch to isolate development work without affecting other branches in the repository
- Modification in a branch can be merged into your project.



Branching is particularly important with Git as it is the mechanism that is used when you are collaborating with other researchers/data scientists.

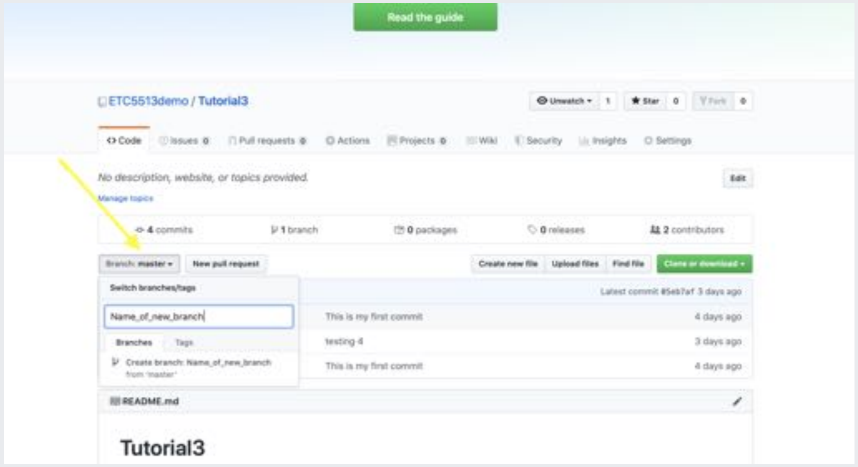
HEAD

HEAD is a pointer that Git uses to reference the current snapshop that we are looking at.



Creating branches from GitHub

- Create branches directly on GitHub. More info here



Latest commit #5eb7af 3 days ago

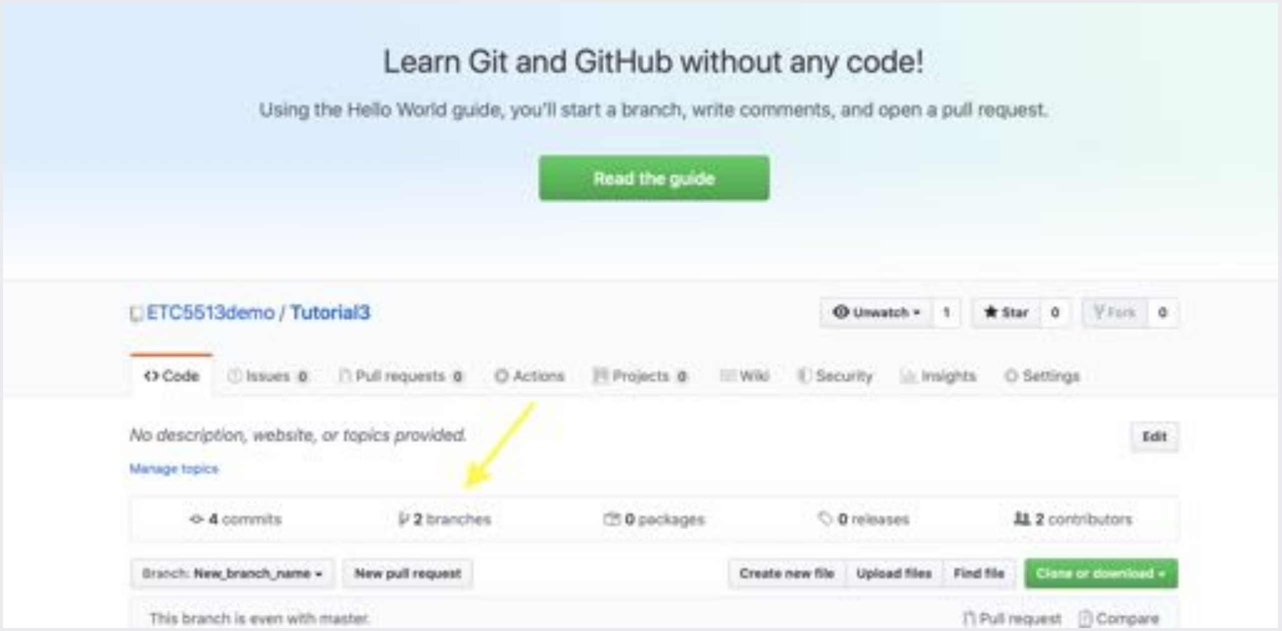
This is my first commit 4 days ago

testing 4 3 days ago

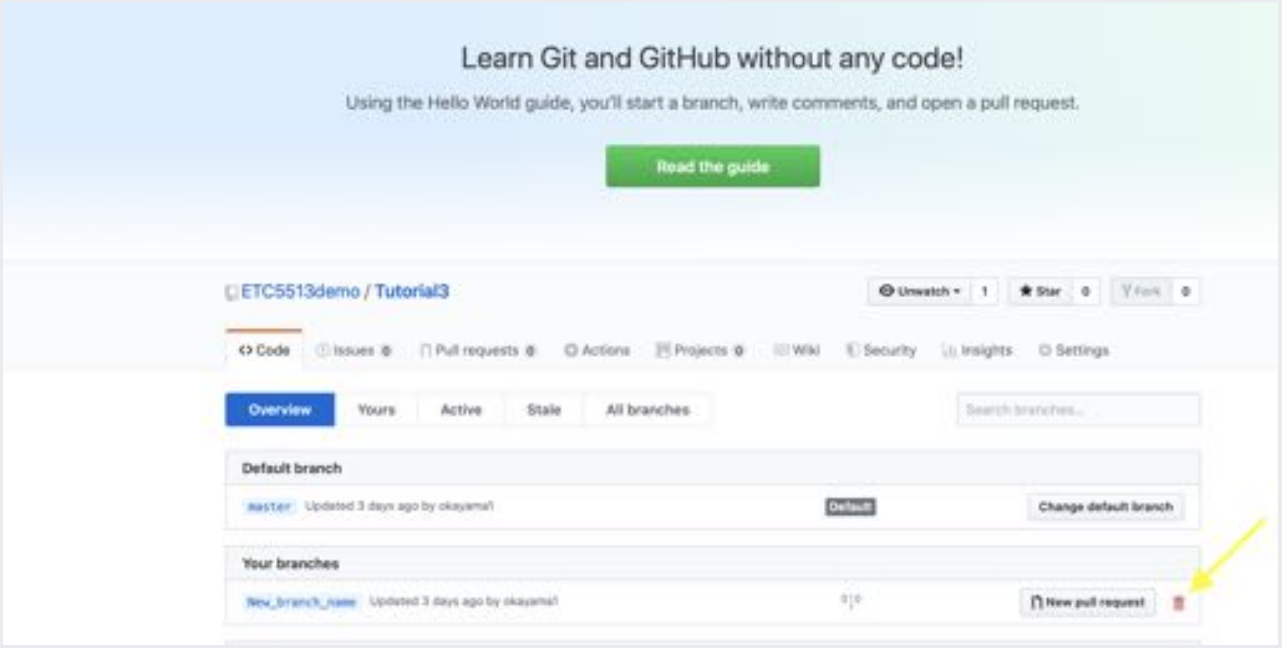
This is my first commit 4 days ago

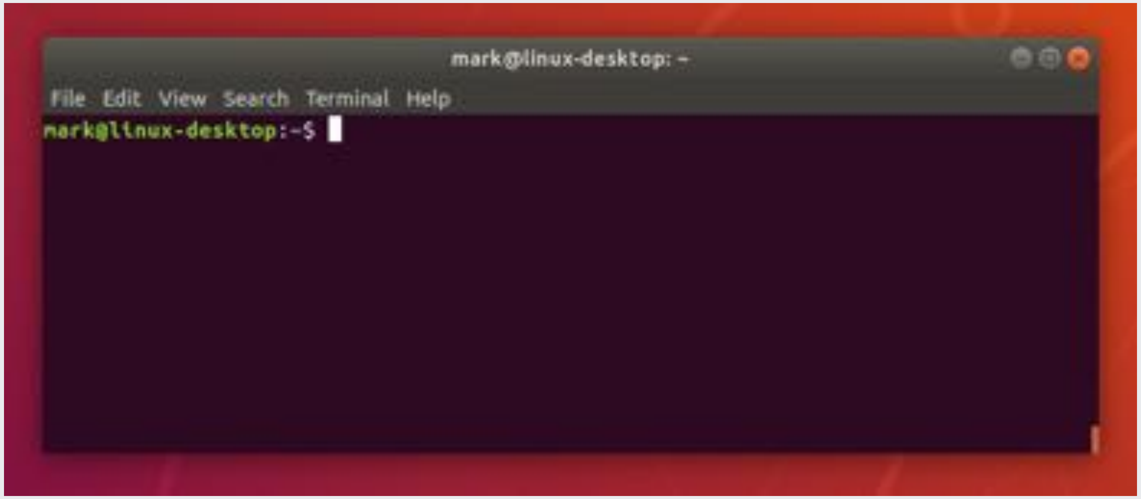
Deleting branches from GitHub

Branches



Deleting branches from GitHub





```
mark@linux-desktop:~$
```

For the time being we will be using the Terminal within Rstudio

Create branches using the Terminal/Shell/CLI

Using git branch and check out command

- git branch --> show us the branches we have in our repo and marks our current branch with *
- git branch newbranch_name --> creates a new branch but does not move the HEAD of the repo there.
- git checkout newbranch_name --> moves the HEAD to newbranch_name

Git HEAD and checkout

How does Git know what branch you’re currently on?

By using the pointer --> HEAD. In Git, this is a pointer to the local branch you are currently on.

Internally, the git checkout command updates the HEAD to point to either the specified branch or commit.

Updating those new branches in the remote repo in GitHub

- We can just update the empty branch into GitHub by `git push origin newbranch_name`

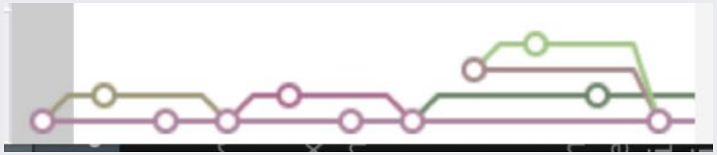
Alernatively if we had files or changes added into that branch:

- `git add .` (--> adding all the modified files into the staging area)
- `git commit -m "Updating new newbranch_name"`
- `git push origin newbranch_name`

Merging branches

Switch to the branch that you want to add the stuff into (let's say that is master). Then

- Merge changes into master --> git checkout master
- Anytime we can use git status --> to check the status of our repo
- git merge newbranch_name -m “Merging branches”
- git push origin master updating the remote repository too

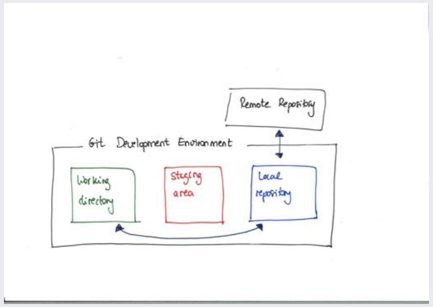


Deleting branches using cli

Deleting branches from your **local** repository

- git branch -a $\rightarrow$ list all the branches
- Move to master [git checkout master]
- Delete unwanted branch git branch -d Name_of_branch $\rightarrow$ delete branch called Name_of_branch

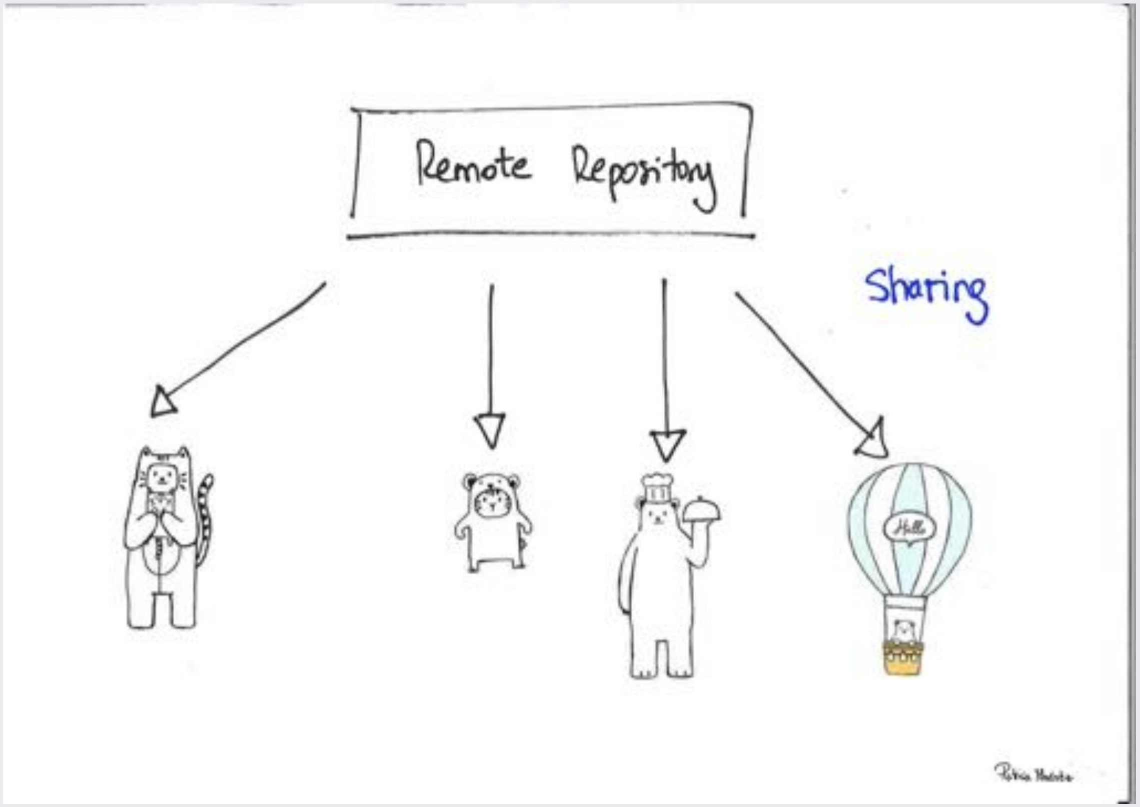
You cannot delete a branch if your HEAD is on that branch



Deleting branches using cli

Deleting branches from your **remote repository** (GitHub)

1. `git push origin --delete Name_of_branch`



How to go back to your previous branch?

- git checkout branchname
- Imagine that you have two branches:
  - Master
  - Alternative_analysis
- To check in which branch you are currently --> git branch --> Alternative_analysis
- To go back to Master (assuming that you were in there) git checkout master

Merging branches sucessfully

Suppose we have two branches: master and new_development

1. For merging --> go to master branch git checkout master
2. git merge new_development
3. This will incorporate the changes made in the branch new_development into master.

If those steps are successful your new_development branch will be fully integrated within the master branch.

Merging branches with conflicts

However, it is possible that Git will not be able to automatically resolve some conflicts,

```
# Auto-merging index.html
# CONFLICT (content): Merge conflict in index.html
# Automatic merge failed; fix conflicts and then com
```

Important: **DO NOT PANIC**

You will need to resolve the conflicts

- You will have to resolve them **manually**.
- This normally happens when two branches the same file but with two different versions. In that case Git is not able to figure out which version to use.

Resolving merging conflicts

- First thing to do --> figure out which files are those affected by the conflict
- git status

```
git status
# On branch master
# You have unmerged paths.
#   (fix conflicts and run "git commit")
#
# Unmerged paths:
#   (use "git add <file>..." to mark resolution)
#
#     both modified:      example.Rmd
#
# no changes added to commit (use "git add" and/or "
```

Resolving the conflict

- Open the file with a text editor
- Go to the lines which are marked with

  <<<<<<, ====== , and >>>>>>

- Edit the file
- git add filename
- git commit -m "Message"
- git push origin master

Resolving conflicts

When you open the conflict file in a **text editor such as Rstudio Cloud / Rstudio**, you will see the conflicted part marked like this:

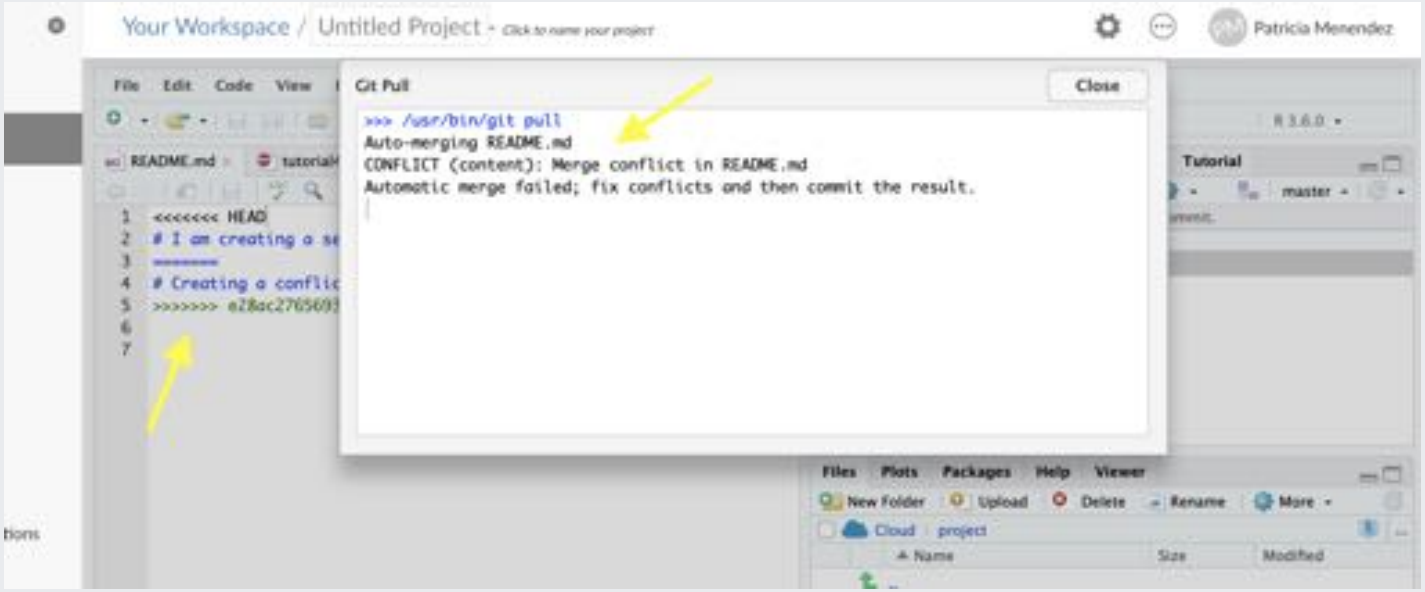
```
/* code unaffected by conflict */
<<<<<<< HEAD
/* code from master that caused conflict */
=======
/* code from feature that caused conflict */
```

When Git encounters a conflict, it adds <<<<<<< and ======= to highlight the parts that caused the conflict and need to be resolved.

Resolving conflicts in practice

- Open the file in a text editor (for example Rstudio / Rstudio Cloud)
- Decide which part of the code you need to keep in the final master branch
- Removed the irrelevant code and the conflict indicators
- Run git add to stage the file/s and git commit to commit the changes --> this will generate the merge commit.

Resolving the conflict

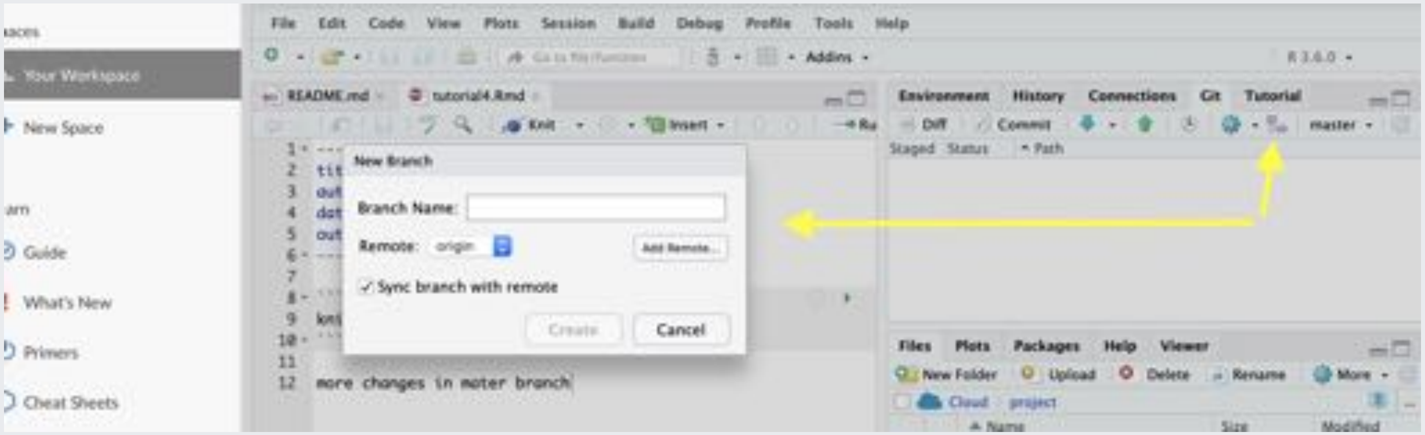


```
>>> /usr/bin/git pull
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

```
<<<<<<< HEAD
# I am creating a se
=======
# Creating a conflic
>>>>>>> e28ac2765693
```

Your Workspace / Untitled Project - Click to name your project
Patricia Menendez
Git Pull
Close
File Edit Code View
README.md
tutorial
Tutorial
master
Files Plots Packages Help Viewer
New Folder Upload Delete Rename More
Cloud project
Name Size Modified

Creating branches from Rstudio

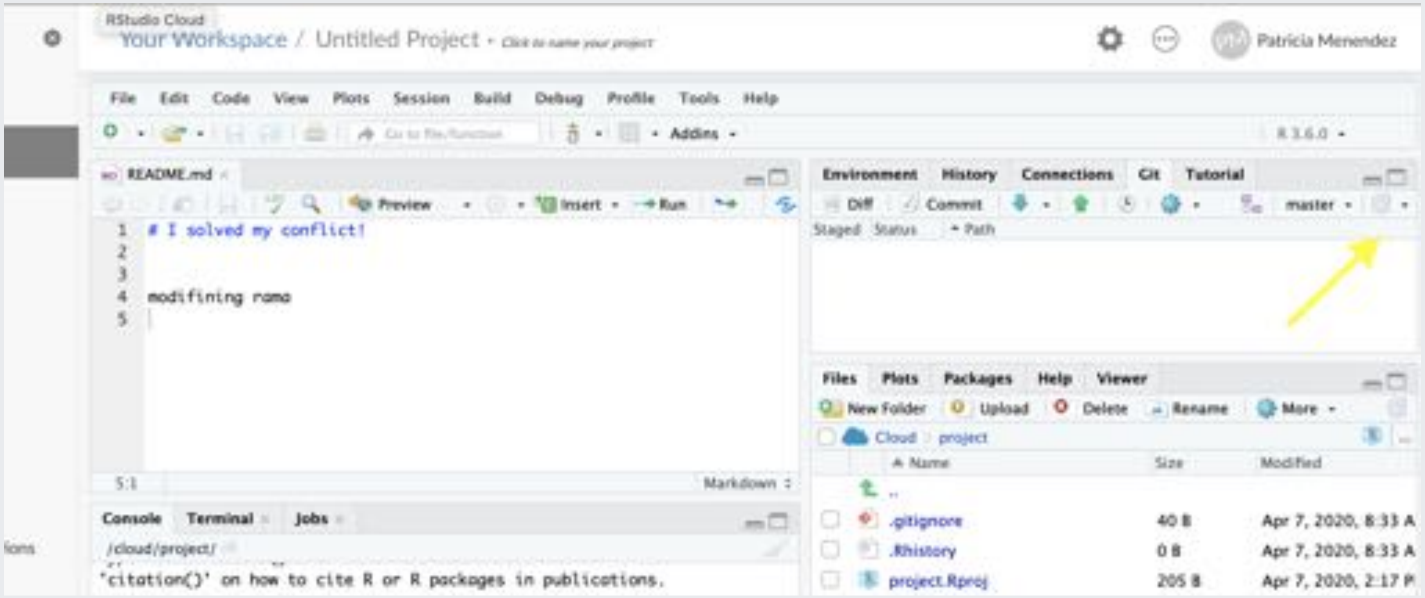


```
more changes in moter branch
```

Important: When we create a branch using Rstudio the branch is created both in the local and in the remote repository (at the same time.)

Keep refreshing Rstudio cloud

Otherwise some of your branches and changes might not be updated.

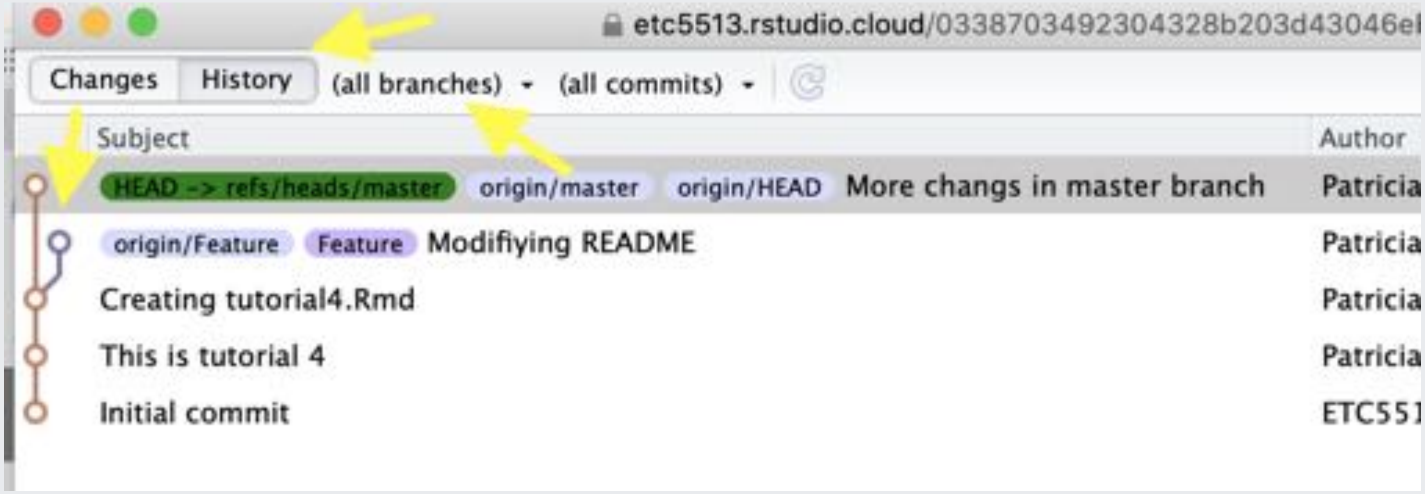

RStudio Cloud
Your Workspace / Untitled Project
Patricia Menendez
File Edit Code View Plots Session Build Debug Profile Tools Help
Addins
R 3.6.0
README.md
Preview
Insert
Run
Environment History Connections Git Tutorial
Diff Commit
master
Staged Status Path
Files Plots Packages Help Viewer
New Folder Upload Delete Rename More
Cloud > project
Name Size Modified
Markdown
Console Terminal Jobs

```
# I solved my conflict!


modifining romo

```

```
/cloud/project/
'citation()' on how to cite R or R packages in publications.
```

Name	Size	Modified
.gitignore	40 B	Apr 7, 2020, 8:33 A
.Rhistory	0 B	Apr 7, 2020, 8:33 A
project.Rproj	205 B	Apr 7, 2020, 2:17 P

Diff window in Rstudio



Subject	Author
HEAD -> refs/heads/master origin/master origin/HEAD More changs in master branch	Patricia
origin/Feature Feature Modifiying README	Patricia
Creating tutorial4.Rmd	Patricia
This is tutorial 4	Patricia
Initial commit	ETC551

Few things to get that are worth

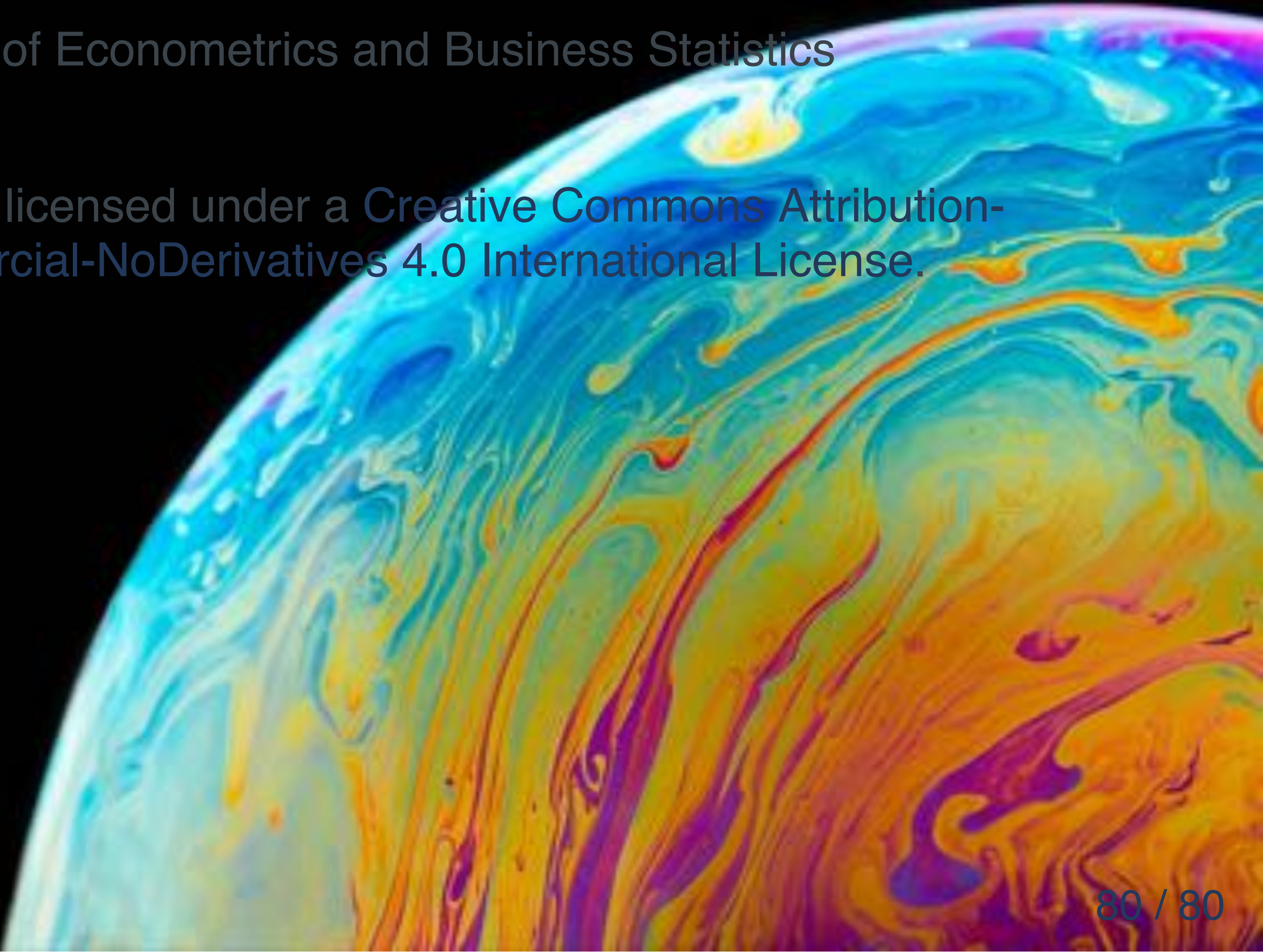
Please follow the link below and get the GitHub Education pack. Then you can use:

- Atom (Install Atom from this link) and
- Gitkraken (from the GitHub education pack for students)

https://education.github.com/students

Three important topics

- GitHub issues --> Great for collaboration. Please see read more here.
- Licenses --> Choose the right License
- .gitignore --> See more here



Lecturer: Dr Patricia Menéndez

Department of Econometrics and Business Statistics