



# Training Data Stewards for Life Sciences

A Crash Course in Version Control and Git

Ricardo Leite

## Git

Created by Linus Torvalds, creator of Linux, in 2005

- Came out of Linux development community
- Designed to do version control on Linux kernel

### Goals of Git:

- Speed
- Support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects efficiently



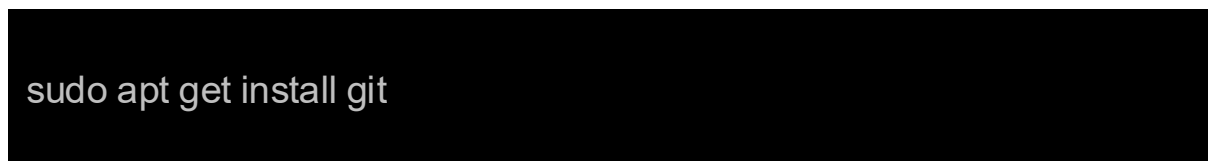
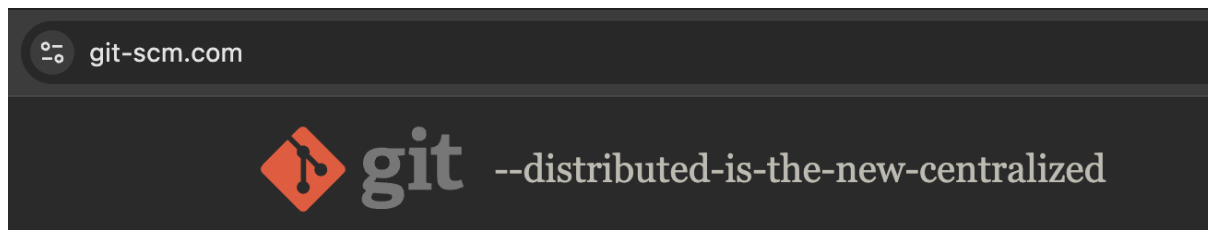
– A "git" is a cranky old man. Linus meant himself.

## Git



<https://xkcd.com/1597/>

## Installation Guide



<https://git-scm.com/doc>

## Free resources:

Git website: <http://git-scm.com/>

– Free on-line book: <http://git-scm.com/book>

– Git cheatsheet: <https://education.github.com/git-cheat-sheet-education.pdf>

– Elixir training lessons: e.g. <https://elixirestonia.github.io/2024-11-06-git/>

– Git for Computer Scientists:

- <http://eagain.net/articles/git-for-computer-scientists/>



## First time users

### ID please?

```
git config --global user.name "Your name"
```

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

### Color preferences

```
git config --global color.ui auto
```

### Let's check?

```
git config --list
```

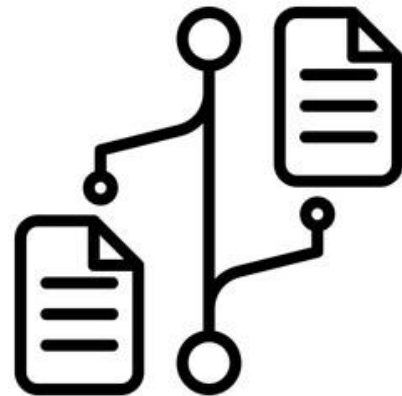
## Version Control

**Version control** (also known as revision control, source control, and source code management) is the software engineering practice of controlling, organizing, and tracking different versions in history of computer files; primarily source code text files, but generally any type of file. (wikipedia)



## Version Control

Many advantages:





## Version Control

### Many advantages:

Collaboration. Team Coordination: Multiple developers can work on the same project simultaneously without conflicts.

Backup and Restore. Automatic Backups: Every change is stored in the repository, providing a comprehensive backup of the project.

Version History.

Efficient Workflow.



Let's avoid:



Thesis.txt



Thesis -  
Copy.txt



Thesis -  
Copy (2).txt



Thesis -  
Final.txt



Thesis -  
Final -  
Copy.txt



Thesis -  
Final  
Final.txt



New Thesis  
- Final Final  
- Copy.txt



New Thesis  
- Final Final  
FINALLLL -  
Copy - Co...

## How it starts?

### for Linux:

```
$ cd /home/user/git_project
```

### for macOS:

```
$ cd /Users/user/git_project
```

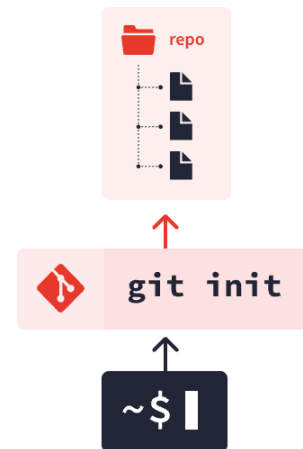
### for Windows:

```
$ cd C:/Users/user/git_project
```

### and type:

```
$ git init
```

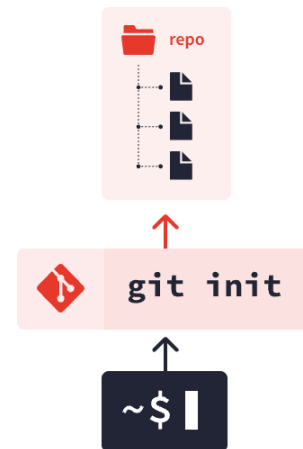
This creates a new subdirectory named `.git` that contains all necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet.



## Creating :

Git will create/copy a hidden .git subdirectory in your project folder. This is where Git stores all the metadata and history it needs to manage your project's versions,

After running either of these commands, you can use the `ls -a` command to verify that the hidden .git subdirectory was created in your project folder.



## Making changes, staging, and committing them :

Imagine you're writing a very important book, and each letter you write is like a file in your project.

### **git init (Getting your Desk Ready):**

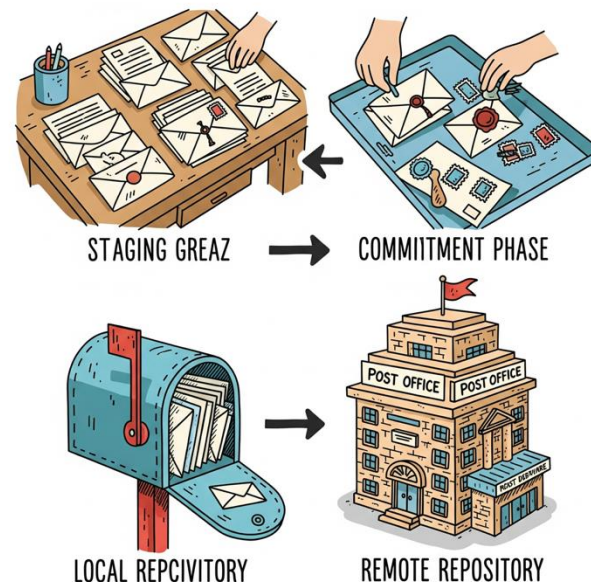
Before you start writing, you need a dedicated workspace. git init is like setting up your filing cabinet (your Git repository) in your office. It's empty at first, but it's ready to organize your work.

### **Creating a file (Writing a Letter):**

You write the first draft of a chapter. This is like creating a new letter.

### **git status (Checking your Desk):**

You look at your desk. You see the letter you just wrote, lying there. It's a "new" item, not yet filed away. This is like git status showing you an "untracked file."



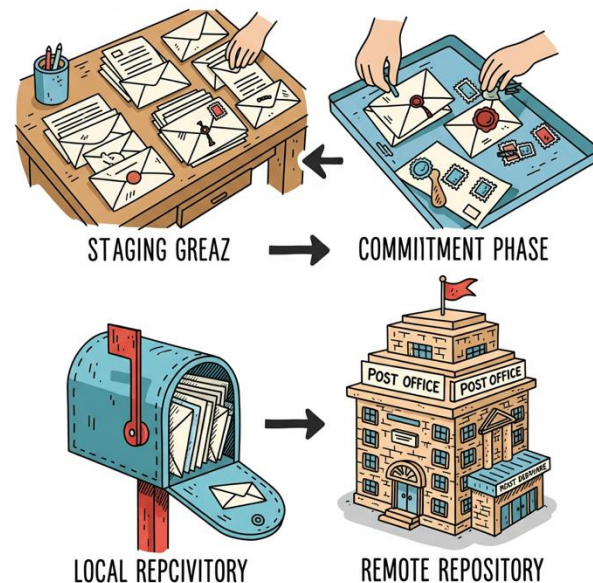
## **git add <filename> (Putting the Letter in an Envelope):**

You decide this chapter is important, so you put the letter into an envelope. This action is `git add`. The envelope is like the staging area. The letter is now "staged" – it's ready to be sent, but not yet sealed and stamped. You can add multiple letters to the same envelope if they belong together.

## **git commit -m "message" (Sealing the Envelope and Stamping It):**

Once all the letters for a particular thought or chapter are in the envelope, you seal the envelope and put a stamp on it. The "stamp" is like the unique commit ID (those long strings of numbers and letters). The message you write on the envelope (e.g., "First draft of Chapter 1") is your commit message. This tells you exactly what's inside this sealed envelope.

Once committed, the envelope is sealed and stamped, making it an immutable record of that specific version of your work. You can always go back and open that specific sealed envelope to see what the letters looked like at that exact point in time.



## Local Repository (Your Mailbox/Collection of Envelopes):

All these sealed, stamped envelopes are sitting in your outbox or mailbox right next to your desk. This is your local Git repository. You have a complete history of all your sealed work, but it hasn't left your office yet.

## Remote Repository (The Post Office / Friend's Mailbox):

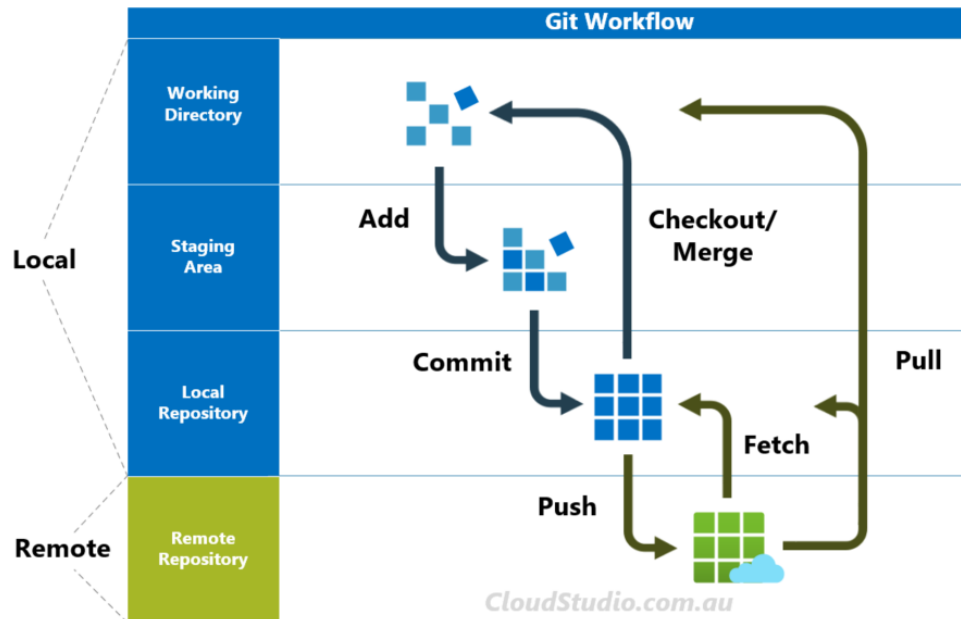
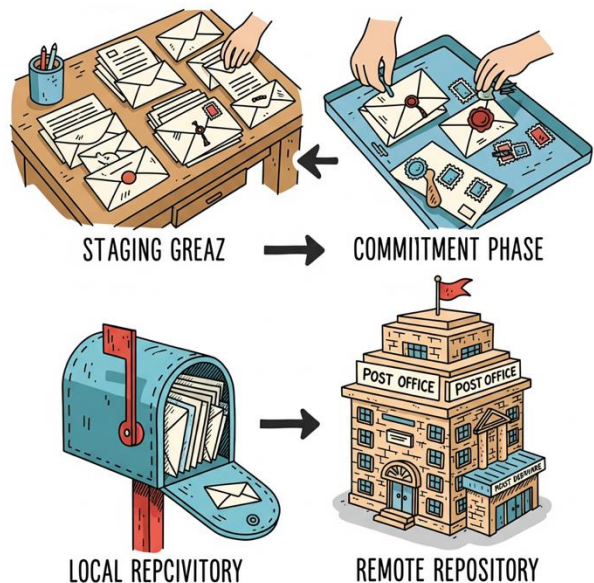
You want to share your work with a publisher or a co-author, or maybe just back it up safely outside your office. This is the remote repository – like the Post Office or a friend's mailbox.

## git push (Sending the Envelopes to the Post Office):

When you're ready to share your work, you take all the sealed, stamped envelopes from your outbox and send them off to the Post Office (or your friend's mailbox). This is git push. You are sending your committed changes from your local repository to the remote repository.

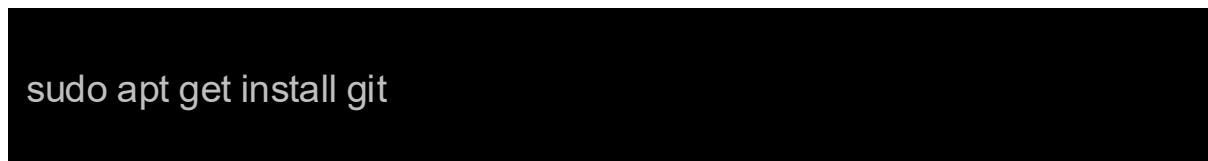
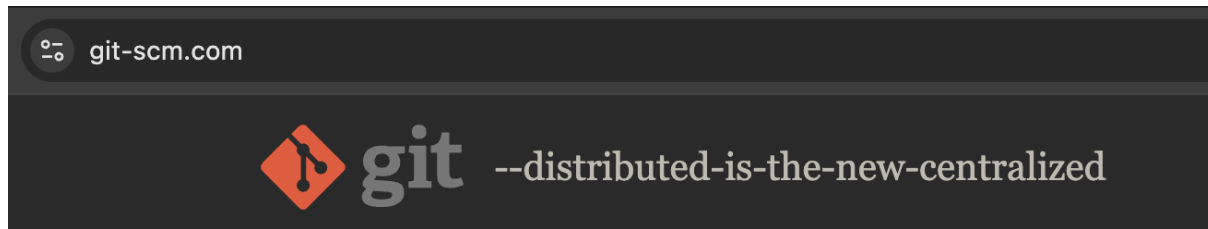
## git pull (Receiving New Mail):

If your co-author also sends you new chapters, you need to go to the Post Office and pick up new mail. This is git pull. You're getting the latest changes from the remote repository down to your local one.





## Installation Guide



## Exercises:

```
$ git -- version
```

If you have git installed, you need to insert your name and email

```
# Set your Git name
git config --global user.name "Your Name"

# Set your Git email
git config --global user.email "your@email.com"

# Configures default initial branch
git config --global init.defaultBranch "main"
```

Create a test directory and cd into it:

```
$ mkdir testgit && cd testgit
```

Next, initiate an empty git repository:

```
$ git init
```

You should get output similar to this:

```
Initialized empty Git repository in /home/"yourusername"/testgit/.git/
```

All files in this directory are now being tracked by Git. Create a new file so that we can try a commit:

```
$ echo "Git Crash Course" > readme.txt
```

Now, if you check the status of the repository, you'll notice the new readme.txt file listed as untracked:

```
$ git status
```

You'll get output like this:

```
On branch main
```

```
No commits yet
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
readme.txt
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

As the message suggests, you can add files to be later committed with the git add command:

```
git add readme.txt
```

Now, if you run git status again, you'll notice that the readme.txt file has moved to staged, which means it is now part of the changes that should be committed. The output also points out this is a new file, with no previous history to Git.

```
On branch main
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file: readme.txt
```

To commit changes, you can use the following command, which commits staged files and uses an inline message to identify this commit:

```
$ git commit -m "first commit"
```

You'll get output similar to this:

```
[main (root-commit) xxxxxxx] first commit  
1 file changed, 1 insertion(+)  
create mode xxxxxx readme.txt
```

Your first commit. You can check all recent changes in a repository with the git log command:

```
$ git log
```

just created your first commit. You can check all recent changes in a repository with the git log command:

```
$ git log
```

You should get output similar to this, showing your first commit to the repo:

```
commit xxxxxxxxxxxxxxxxxxxxxxxxxxxx (HEAD -> main)
```

```
Author: blabla
```

```
Date: Mon Jul 7
```

```
first commit
```

```
(END)
```

To exit the log view, press **q**.

Let's change the readme.txt file. Edit using notepad++ or using or preferred editor. Let's check now

```
$ git status
```

You should get output similar to this, showing your first commit to the repo:

```
$ git add readme.txt
```

```
$ git commit -m "I have changed the text"
```

```
( [main 2f52edb] I have changed the text
```

```
1 file changed, 2 insertions(+), 1 deletion(-)
```

```
$git status
```

```
On branch main
```

```
nothing to commit, working tree clean
```



Let's check the log:

```
$ git log  
commit 2f52edb487c3972a79bf68f433c9a65a90bbb86a (HEAD -> main)  
Author: Ricardo Leite <rleite@mac.lan>  
Date: Sun Jul 6 11:07:54 2025 +0100
```

I have changed the text

```
commit 10c1a1e885bf5969498d562dd3cc5da0a94fb8f1  
Author: Ricardo Leite <rleite@mac.lan>  
Date: Sun Jul 6 11:05:29 2025 +0100
```

Assing my file to this version

Let's check the log:

```
$ git show 2f52edb487c3972a79bf68f433c9a65a90bbb86a  
commit 2f52edb487c3972a79bf68f433c9a65a90bbb86a (HEAD -> main)  
Author: Ricardo Leite <rleite@mac.lan>  
Date: Sun Jul 6 11:07:54 2025 +0100
```

I have changed the text

```
$ diff --git a/file.txt b/file.txt  
index 849a4c7..0a2e27d 100644  
--- a/file.txt  
+++ b/file.txt  
@@ -1,2 @@  
-practicing git  
+Hi!  
+I'm practicing git today
```

Let's check the log:

```
$ git show 2f52edb487c3972a79bf68f433c9a65a90bbb86a
commit 2f52edb487c3972a79bf68f433c9a65a90bbb86a (HEAD -> main)
Author: Ricardo Leite <rleite@mac.lan>
Date: Sun Jul 6 11:07:54 2025 +0100
```

I have changed the text

```
$ diff --git a/file.txt b/file.txt
index 849a4c7..0a2e27d 100644
--- a/file.txt
+++ b/file.txt
@@ -1,2 @@
-practicing git
+Hi!
+I'm practicing git today
```



**Local vs remote repository**

## Local repository

A local repository is a complete copy of your project's Git history stored directly on your computer. All your commits, branches, and version history are contained within the .git folder in your project directory.

### Advantages :

Speed: All operations (commits, diffs, branching, merging) are incredibly fast

Offline Work: You can work on your project and make commits even if you don't have internet access.

Privacy: Your work remains private to your machine until you explicitly choose to share it.

Experimentation: It's easy to experiment

### Disadvantages of a Local Repository:

No Collaboration: Other developers cannot access or contribute to your work unless you share it manually

No Backup: If your local machine crashes or gets lost, your entire project history could be gone

No Central Source of Truth: Without a remote repository, it's hard to determine the "definitive" version of the project if multiple people are working on it.

## Remote Repository

A remote repository is a version of your project hosted on a server. It acts as a central hub where developers can upload (push) their local changes and download (pull) changes made by others.

### Advantages of a Remote Repository:

**Collaboration:** Multiple developers can work on the same project, share their code, and integrate changes.

**Backup and Disaster Recovery:** Your project's history is safely stored off-site.

**Centralized Source of Truth:** Provides a single, definitive version of the codebase that everyone can refer to.

**Accessibility:** Your code can be accessed from anywhere with an internet connection.

**Integration:** Integration/continuous delivery pipelines, automating builds, tests, and deployments.

### Disadvantages of a Remote Repository:

**Internet Dependency (for sync):** You need an internet connection to push or pull changes

**Potential for Conflicts:** With multiple people pushing changes, merge conflicts can arise

**Security Concerns:** Depending on where the remote is hosted, there might be security considerations for sensitive code.

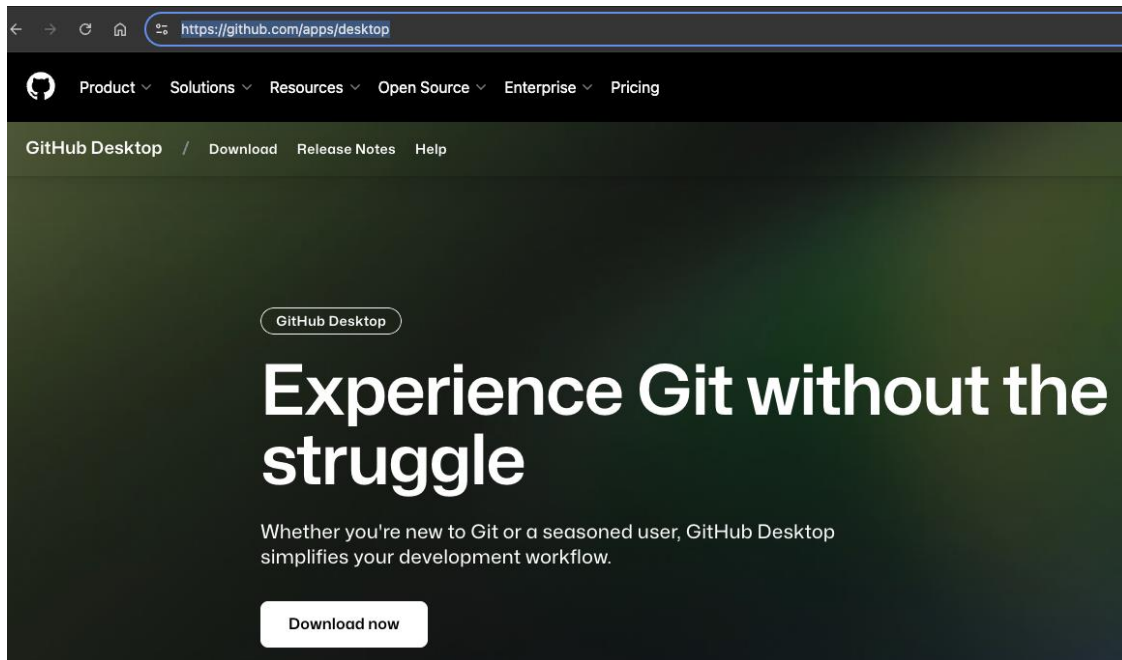
How to start your Github journey:

<https://docs.github.com/>

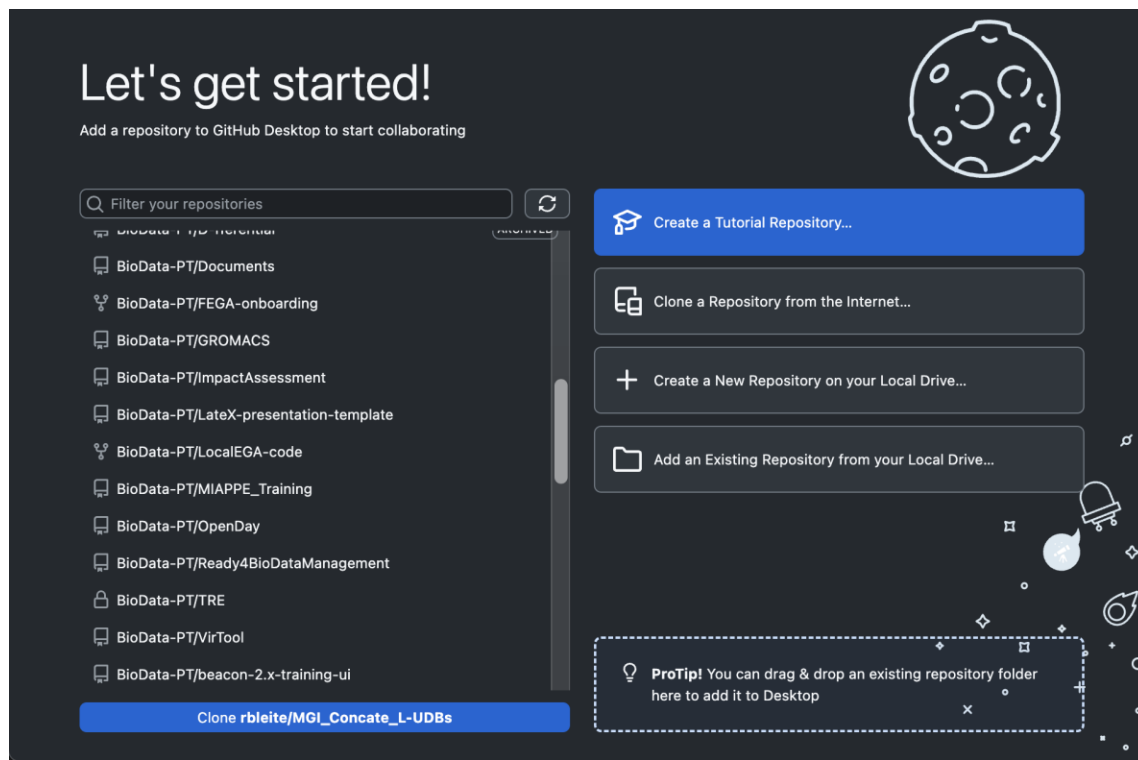


Github Client:

<https://github.com/apps/desktop>







## Advance:

### Create ssh key

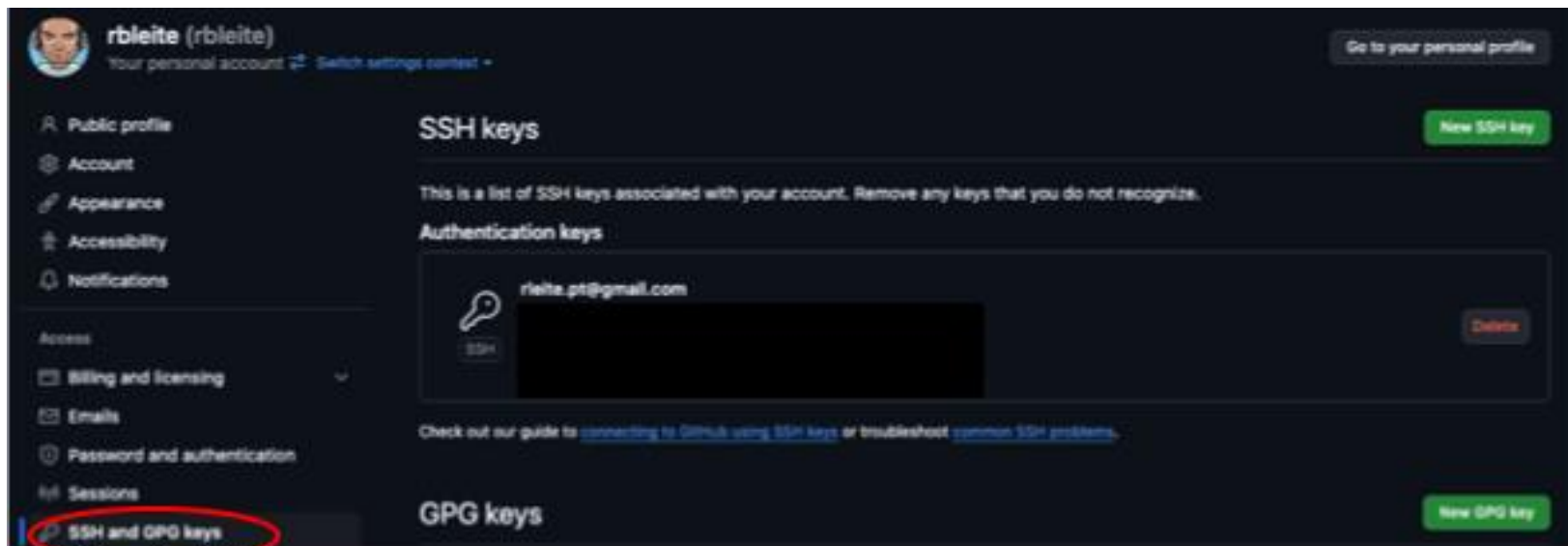
```
$ ssh-keygen -t ed25519 -C your_email@example.com
```

The id\_ed25519.pub file is the public portion of your SSH key. You'll use the content of this file to set up your SSH key within GitHub. The id\_ed25519 file without extension is the private portion of your SSH key, and should never be shared.

### Adding an SSH Key to your GitHub Account

To add your SSH key to your GitHub account, access the menu “SSH and GPG Keys” under your Settings.

## SSH keys:



The screenshot shows the GitHub account settings for user 'rbleite'. The left sidebar contains navigation links: Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and licensing, Emails, Password and authentication, Sessions, and SSH and GPG keys (which is circled in red). The main content area is titled 'SSH keys' and includes a 'New SSH key' button. Below the title, it states: 'This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.' Under the heading 'Authentication keys', there is one key listed for the email 'rbleite.pt@gmail.com'. The key's content is redacted with a black box, and there is a 'Delete' button next to it. At the bottom of the main area, there is a link to a guide: 'Check out our guide to [connecting to GitHub using SSH keys](#) or troubleshoot [common SSH problems](#).' Below this, there is a section for 'GPG keys' with a 'New GPG key' button.

## Clone

The screenshot shows the GitHub repository page for `octocat / Spoon-Knife`. The repository is public and has 764 watches, 152k forks, and 13.1k stars. The 'Code' button is highlighted with a red circle. A dropdown menu is open, showing the 'Clone' option, which is also highlighted with a red circle. The clone URL `https://github.com/octocat/Spoon-Knife.git` is displayed and highlighted with a red circle. The repository description states: 'This repo is for demonstration purposes only.' The README section includes a greeting 'Well hello there!' and instructions on how to fork a repository.

**Code** Issues 2.4k Pull requests 5k+ Actions Projects Wiki Security Insights

**Spoon-Knife** Public Watch 764 Fork 152k Star 13.1k

main 3 Branches 0 Tags

Go to file Add file <> Code

octocat Pointing to the guide for forking

README.md Pointing to the guide for forking

index.html Created index page

styles.css Create styles.css

**Clone**

HTTPS SSH GitHub CLI

`https://github.com/octocat/Spoon-Knife.git`

Clone using the web URL.

Open with GitHub Desktop

Download ZIP

**README**

**Well hello there!**

This repository is meant to provide an example for *forking* a repository on GitHub.

Creating a *fork* is producing a personal copy of someone else's project. Forks act as a sort of bridge between the original repository and your personal copy. You can submit *Pull Requests* to help make other people's projects better by offering your changes up to the original project. Forking is at the core of social coding at GitHub.

After forking this repository, you can make some changes to the project, and submit a [Pull Request](#) as practice.

For some more information on how to fork a repository, [check out our guide, "Forking Projects"](#). Thanks! ❤️

**Releases**

No releases published

**Packages**

No packages published

**Languages**

HTML 58.1% CSS 41.9%

## Clone

The screenshot shows the GitHub repository page for `octocat / Spoon-Knife`. The repository is public and has 764 watches, 152k forks, and 13.1k stars. The 'Code' button is highlighted with a red circle. A dropdown menu is open, showing the 'Clone' option, which is also highlighted with a red circle. The clone URL `https://github.com/octocat/Spoon-Knife.git` is displayed and highlighted with a red circle. The repository description states: 'This repo is for demonstration purposes only.' The README section includes a greeting 'Well hello there!' and instructions on how to fork a repository.

octocat / Spoon-Knife

Code Issues 2.4k Pull requests 5k+ Actions Projects Wiki Security Insights

Watch 764 Fork 152k Star 13.1k

main 3 Branches 0 Tags

Go to file Add file <> Code

octocat Pointing to the guide for forking

README.md Pointing to the guide for forking

index.html Created index page

styles.css Create styles.css

Clone

HTTPS SSH GitHub CLI

`https://github.com/octocat/Spoon-Knife.git`

Clone using the web URL.

Open with GitHub Desktop

Download ZIP

README

Well hello there!

This repository is meant to provide an example for *forking* a repository on GitHub.

Creating a *fork* is producing a personal copy of someone else's project. Forks act as a sort of bridge between the original repository and your personal copy. You can submit *Pull Requests* to help make other people's projects better by offering your changes up to the original project. Forking is at the core of social coding at GitHub.

After forking this repository, you can make some changes to the project, and submit a [Pull Request](#) as practice.

For some more information on how to fork a repository, [check out our guide, "Forking Projects"](#). Thanks! ❤️

This repo is for demonstration purposes only.

Readme

Activity

13.1k stars

764 watching

152k forks

Report repository

Releases

No releases published

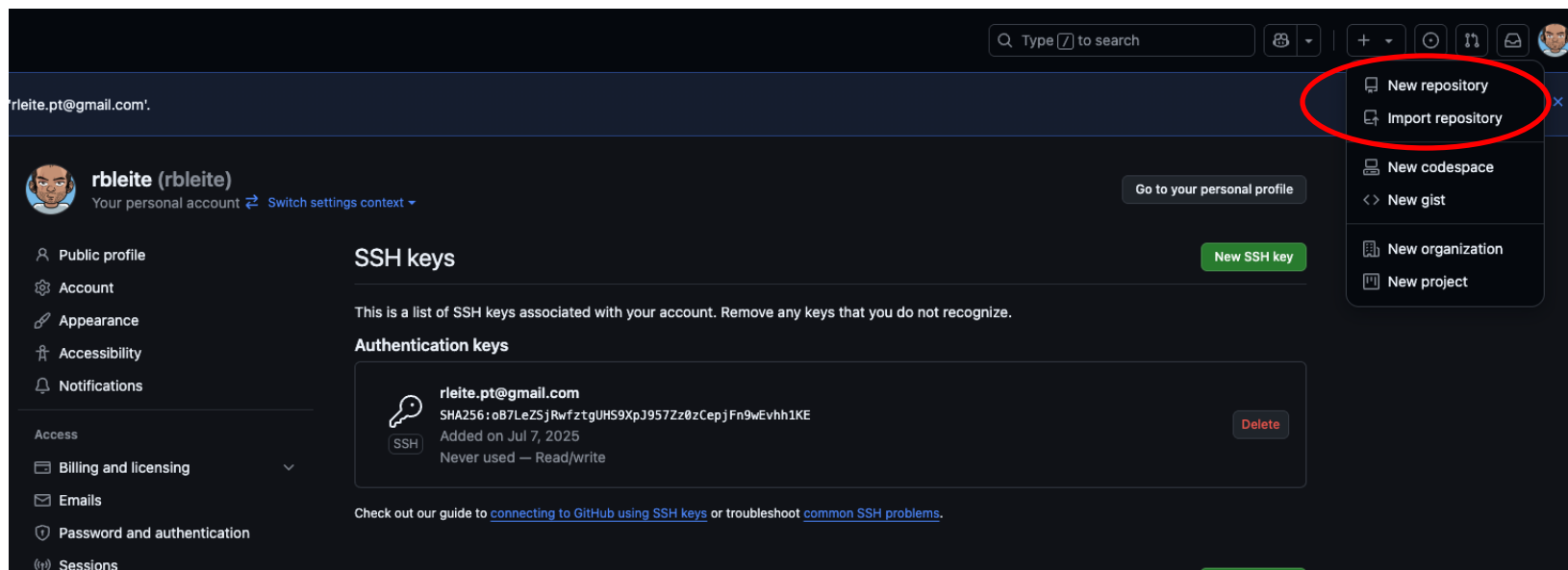
Packages

No packages published

Languages

HTML 58.1% CSS 41.9%

## New or import repository



The screenshot shows the GitHub web interface for a user named 'rbleite (rbleite)'. The top navigation bar includes a search bar and a menu with icons for adding new content. A red circle highlights the 'New repository' and 'Import repository' options in this menu. The left sidebar shows the user's profile and navigation links. The main content area displays 'SSH keys' and 'Authentication keys' sections. A single SSH key is listed for the email 'rbleite.pt@gmail.com', added on July 7, 2025, and marked as 'Never used'. A 'Delete' button is visible next to the key. At the bottom, there is a link to a guide on connecting to GitHub using SSH keys.

## New repository


### 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 (\*).*

Owner \*

 rbleite ▾


Repository name \*

 Test1 is available.


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

Description (optional)

---

☒  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

▾

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

Choose a license

▾

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

## Branch

All new repository have a default branch called “main”

To list:

```
$ git branch
```

Let's create a new branch called “another” while keeping in the main one

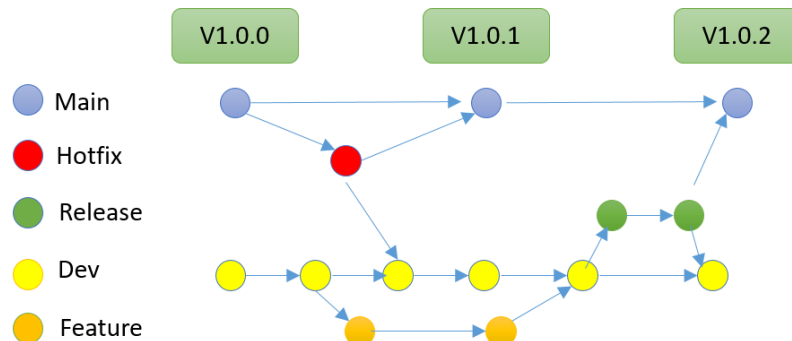
```
$ git branch another main
```

Create a branch and changing to this new one

```
$ git checkout -b “anotherone”
```

To remove

```
$ git branch -r “anotherone”
```







## Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

## Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

## Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my\_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new\_branch

```
$ git branch new_branch
```

Delete the branch called my\_branch

```
$ git branch -d my_branch
```

Merge branch\_a into branch\_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

## Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

## Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

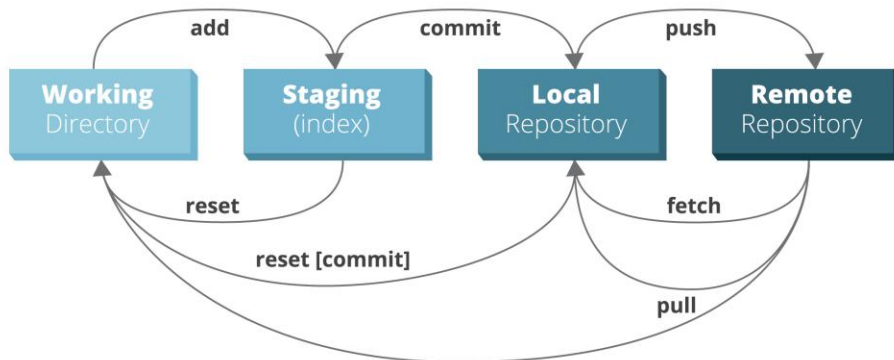
```
$ git push
```

## Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit <https://training.github.com/> for official GitHub training.



# Training Data Stewards for Life Sciences

Presentation title

THANK YOU!

Questions?

