Introduction to version control with Git

Scientific workflows: Tools and Tips 🛠️

2025-07-17

Scientific workflows: Tools and Tips 🛠️

📅 Every 3rd Thursday 🕓 4-5 p.m. 📍 Webex

- Slides and material provided online
  - Subscribe to the mailing list
- **Credit points**: send me a private message/email with your name and email

Why version control?

Why version control?

Git is like a Lab Notebook for your scripts

- **Tracks** every change in your scripts
- Helps you **recover** older versions
- Enables safe and easy **collaboration**
- Makes it easy to **share** your work with others

Today

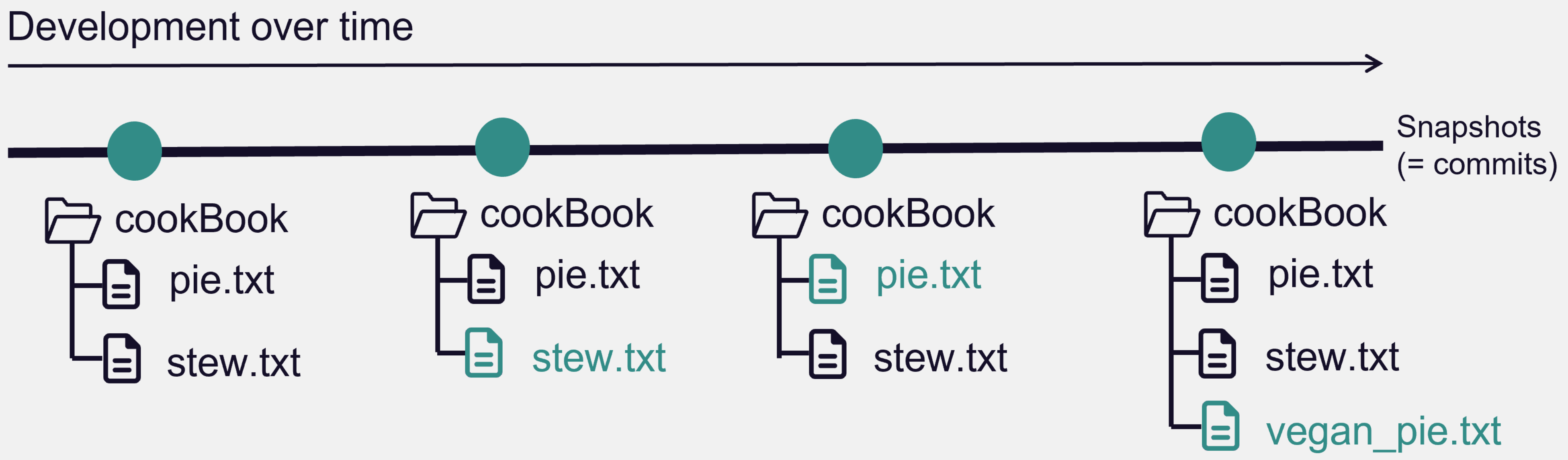
- Introduction to **Git**
- Simple Git workflow in **theory and practice**
- Publish your work on **GitHub**
- Find detailed how-to guides on the website

What is Git?

- **Open source and free** to use version control software
- Quasi **standard** for software development
- **Complete** and **long-term** history of every file in your project
- A whole universe of **other software and services** around it

What is Git?

- For projects with **mainly text files** (e.g. code, markdown files, …)
- Basic idea: Take snapshots (**commits**) of your project over time

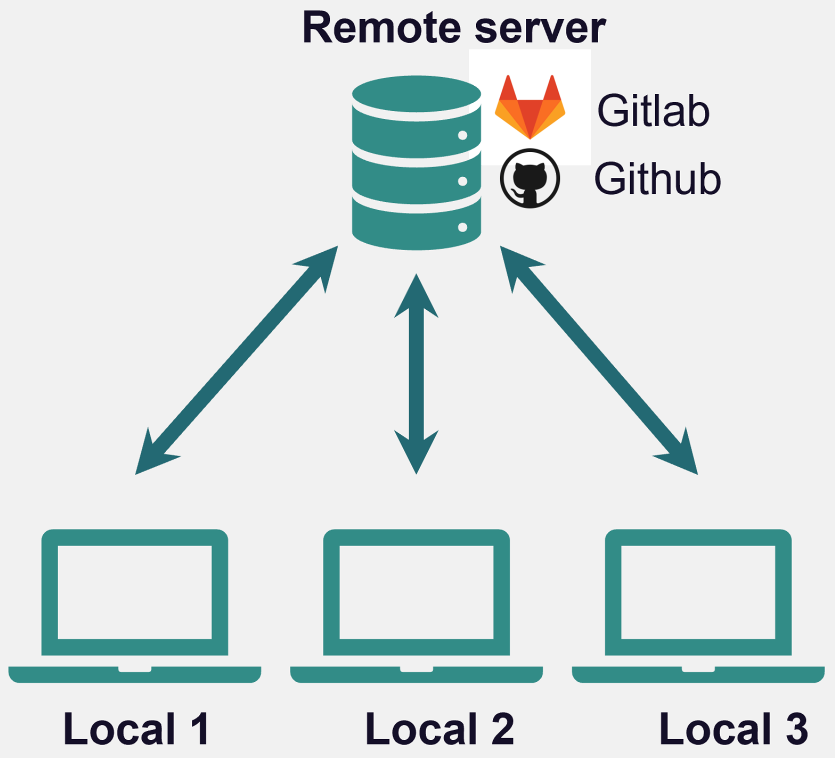


- A project version controlled with Git is a Git **repository** (**repo**)

Version control with Git

Git is a **distributed version control system**

Idea: many *local* repositories synced via one *remote* repo

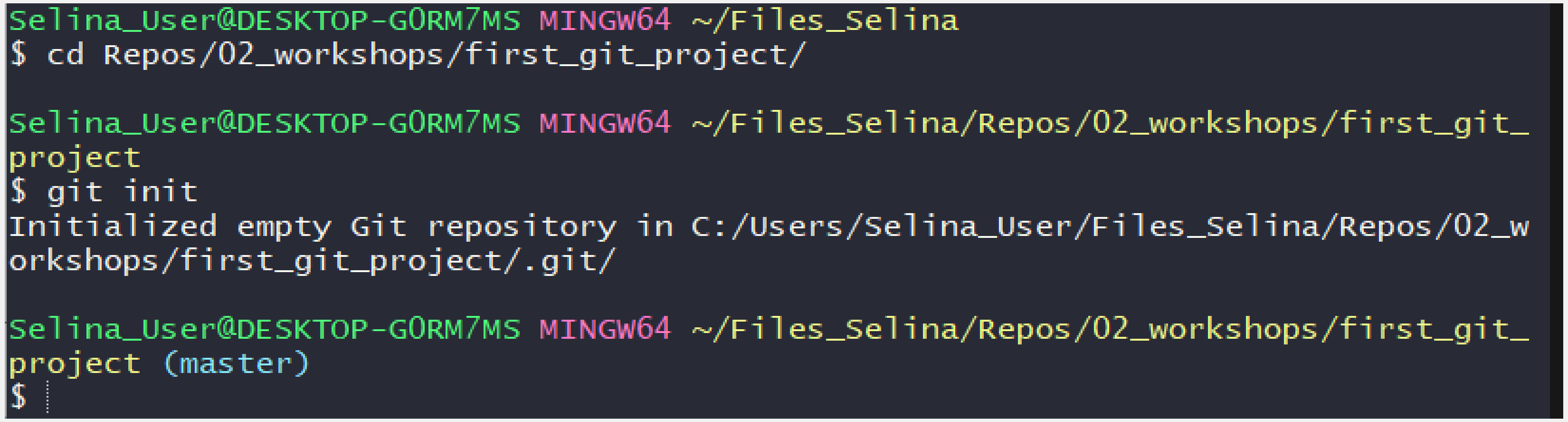


How to use Git

After you installed it there are different ways to interact with the software.

How to use Git - Terminal

Using Git from the terminal



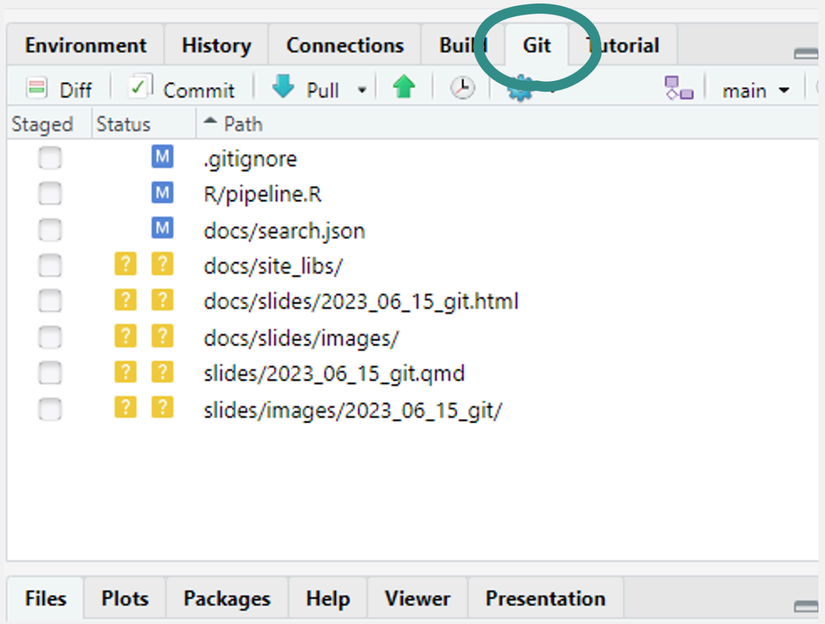
```
Selina_User@DESKTOP-GORM7MS MINGW64 ~/Files_Selina
$ cd Repos/02_workshops/first_git_project/

Selina_User@DESKTOP-GORM7MS MINGW64 ~/Files_Selina/Repos/02_workshops/first_git_
project
$ git init
Initialized empty Git repository in C:/Users/Selina_User/Files_Selina/Repos/02_w
orkshops/first_git_project/.git/

Selina_User@DESKTOP-GORM7MS MINGW64 ~/Files_Selina/Repos/02_workshops/first_git_
project (master)
$
```

+ Most control
+ A lot of help/answers online

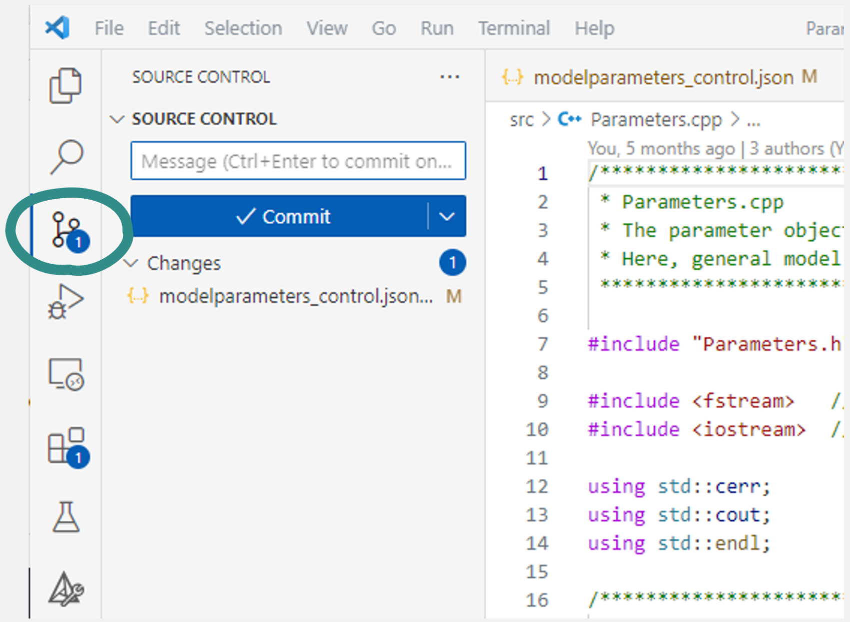
− You need to use terminal 😱

How to use Git - Integrated GUIs

A Git GUI is integrated in most (all?) IDEs, e.g. R Studio, VS Code



Staged	Status	Path
☐	M	.gitignore
☐	M	R/pipeline.R
☐	M	docs/search.json
☐	? ?	docs/site_libs/
☐	? ?	docs/slides/2023_06_15_git.html
☐	? ?	docs/slides/images/
☐	? ?	slides/2023_06_15_git.qmd
☐	? ?	slides/images/2023_06_15_git/


File
Edit
Selection
View
Go
Run
Terminal
Help
Para
SOURCE CONTROL
SOURCE CONTROL
Message (Ctrl+Enter to commit on...
✓ Commit
Changes
1
modelparameters_control.json... M
modelparameters_control.json M
src > Parameters.cpp > ...
You, 5 months ago | 3 authors (Y

```
/*********************
 * Parameters.cpp
 * The parameter objec
 * Here, general model
 *********************

#include "Parameters.h

#include <fstream>   /
#include <iostream>  /

using std::cerr;
using std::cout;
using std::endl;

/*********************
```

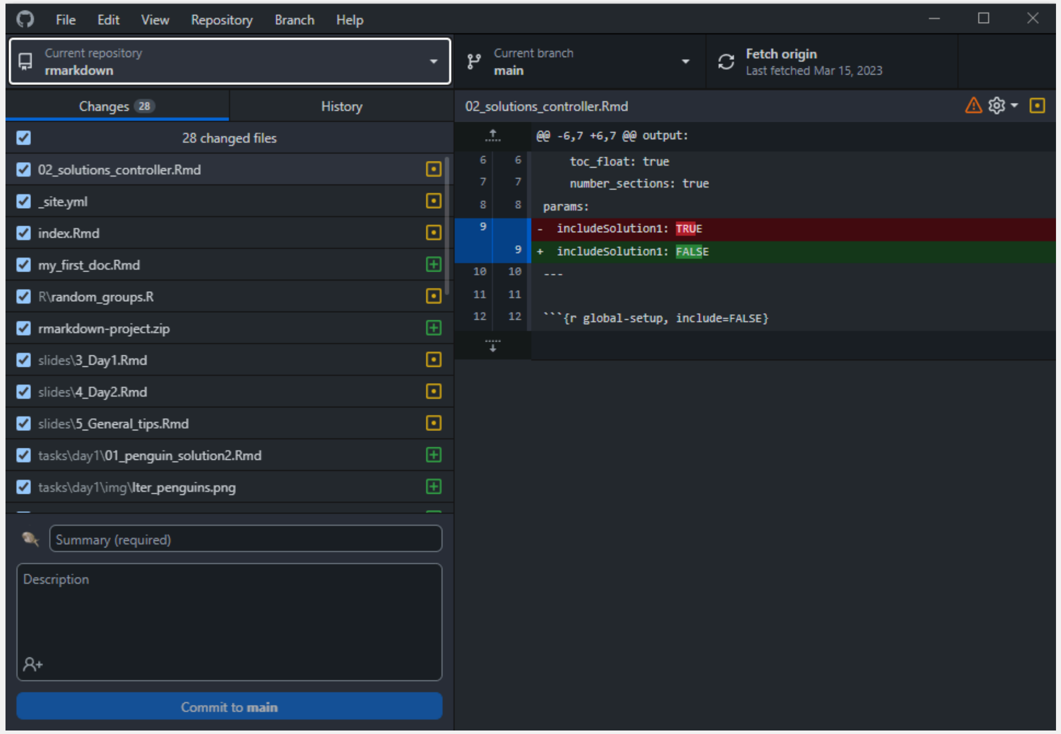
➕ Easy and intuitive

➕ Stay inside IDE

➖ Different for every program

How to use Git - Standalone GUIs

Standalone Git GUI software, e.g. GitHub Desktop, Source Tree, …



28 changed files

- 02_solutions_controller.Rmd
- _site.yml
- index.Rmd
- my_first_doc.Rmd
- R\random_groups.R
- rmarkdown-project.zip
- slides\3_Day1.Rmd
- slides\4_Day2.Rmd
- slides\5_General_tips.Rmd
- tasks\day1\01_penguin_solution2.Rmd
- tasks\day1\img\lter_penguins.png

02_solutions_controller.Rmd

~~~
@@ -6,7 +6,7 @@ output:
    toc_float: true
    number_sections: true
params:
-  includeSolution1: TRUE
+  includeSolution1: FALSE
---

```{r global-setup, include=FALSE}
~~~

＋ Easy and intuitive

＋ Use for all projects

− Switch programs to use Git

How to use Git

Which one to choose?

- Depends on experience and taste
- You can mix methods because they are all interfaces to the same Git

- We will use GitHub Desktop
  - Beginner-friendly, intuitive and convenient
  - Nice integration with GitHub

Tip

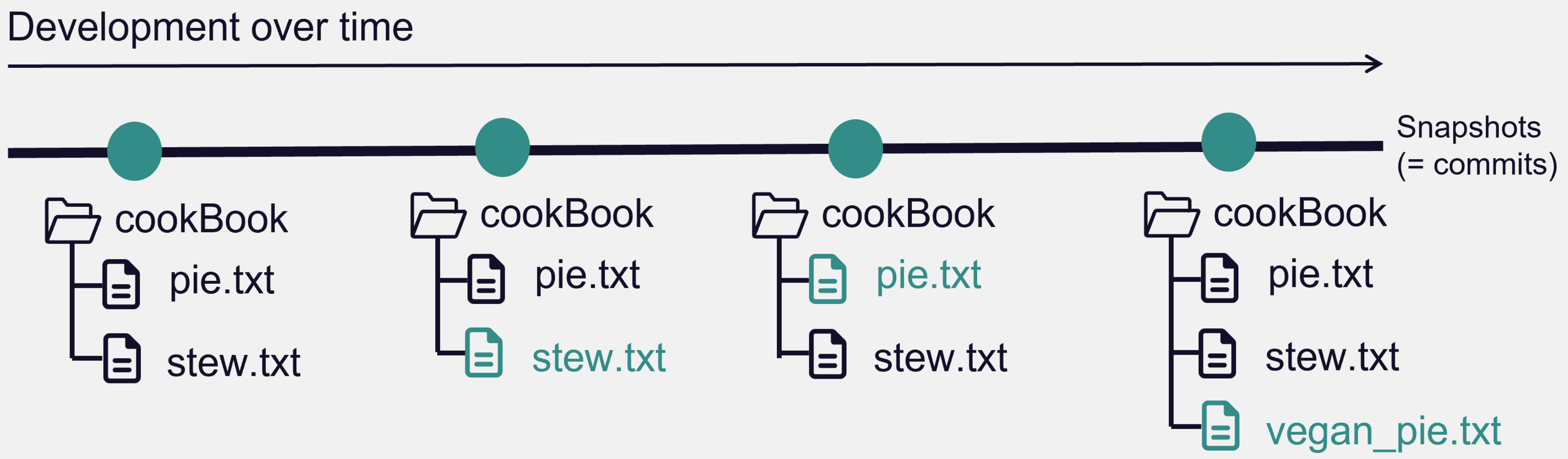
Have a look here to find **How-To guides for the other methods** as well.

The basic Git workflow

> `git init`, `git add`, `git commit`, `git push`

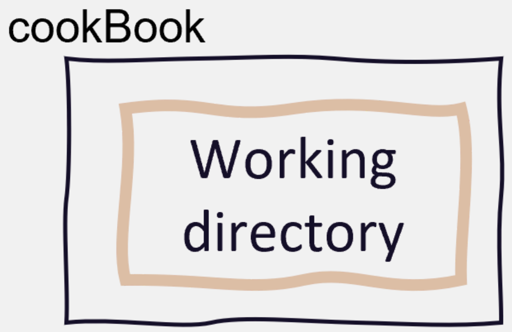
Example

A cook book project to collect all my favorite recipes.



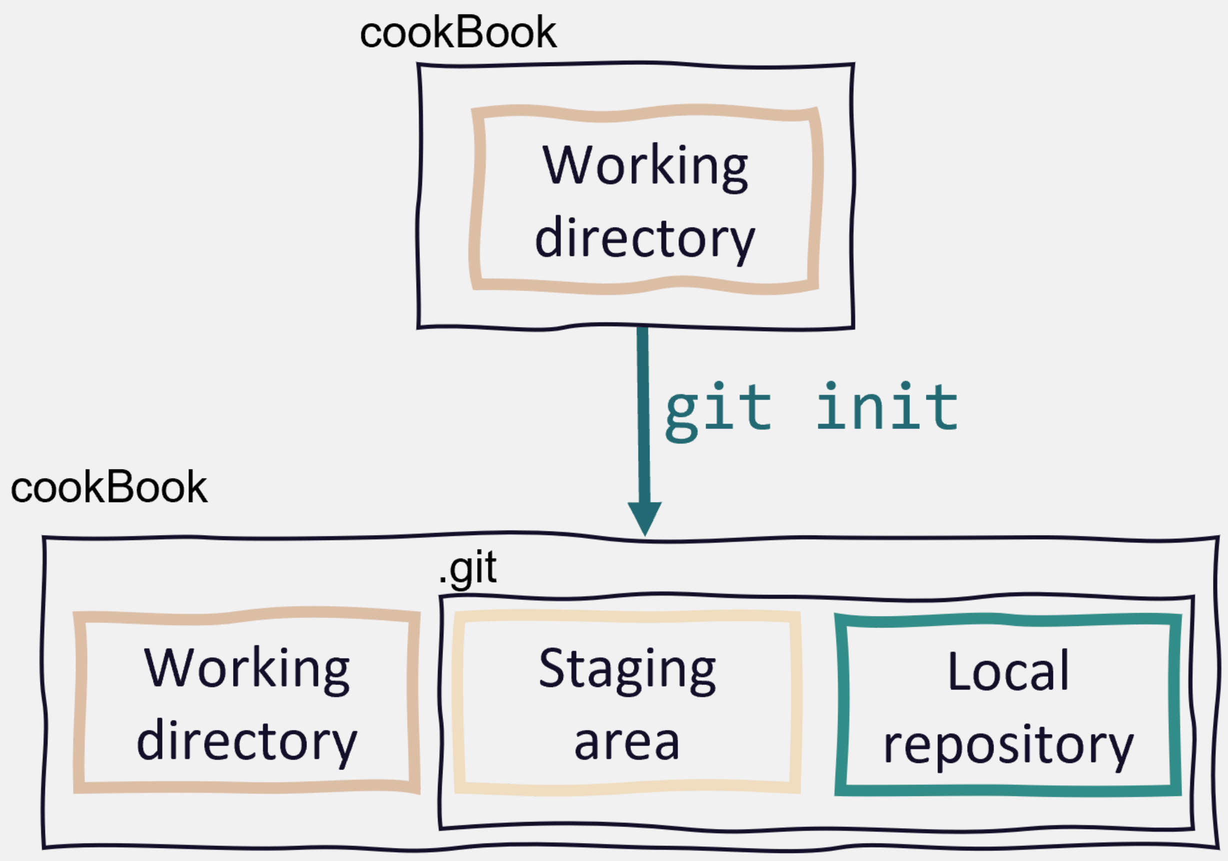
In real life this would be e.g. a data analysis project, your thesis in LaTex, a software project, …

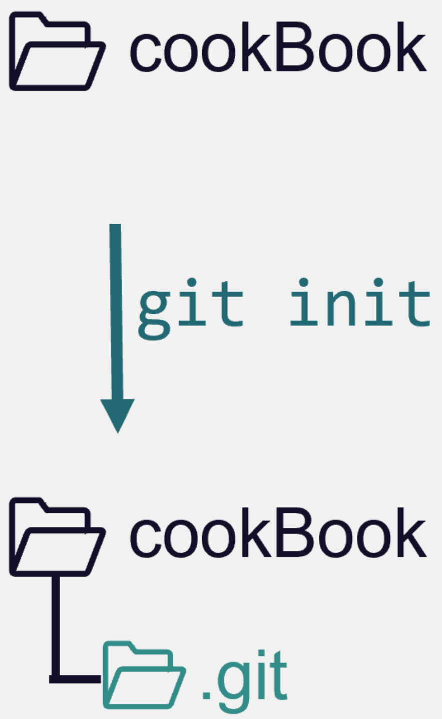
Step 1: Initialize a Git repository



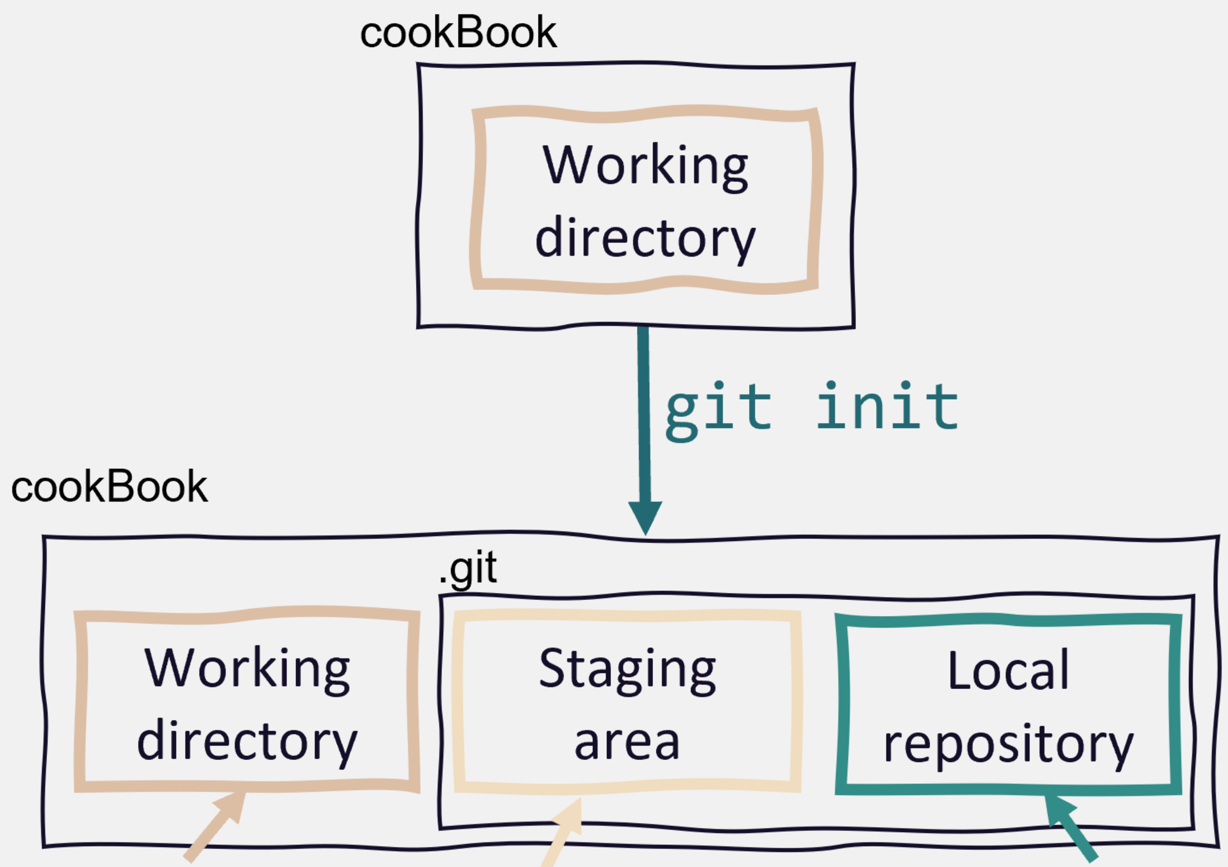
cookBook

Step 1: Initialize a Git repository





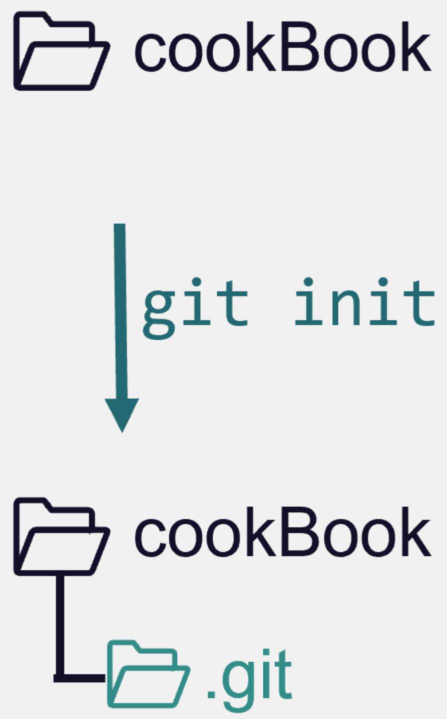
Step 1: Initialize a Git repository



Place for **files**

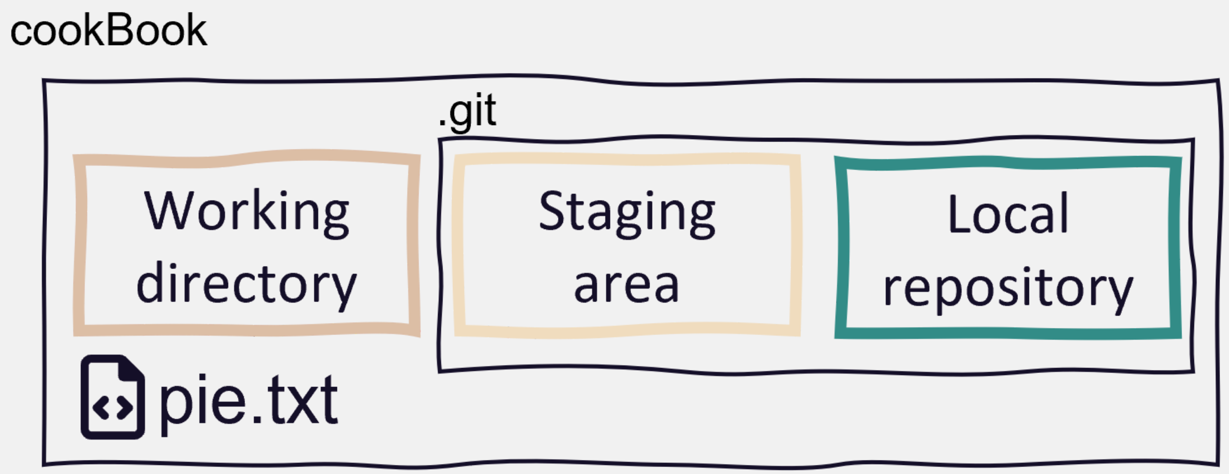
A **list of changes** that go into the next snapshot (commit)

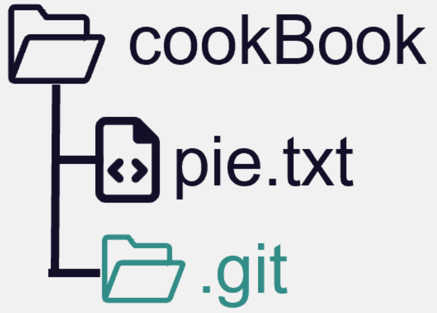
The **heart of your project**: This is where the **commit history** lives



Step 2: Add and modify files

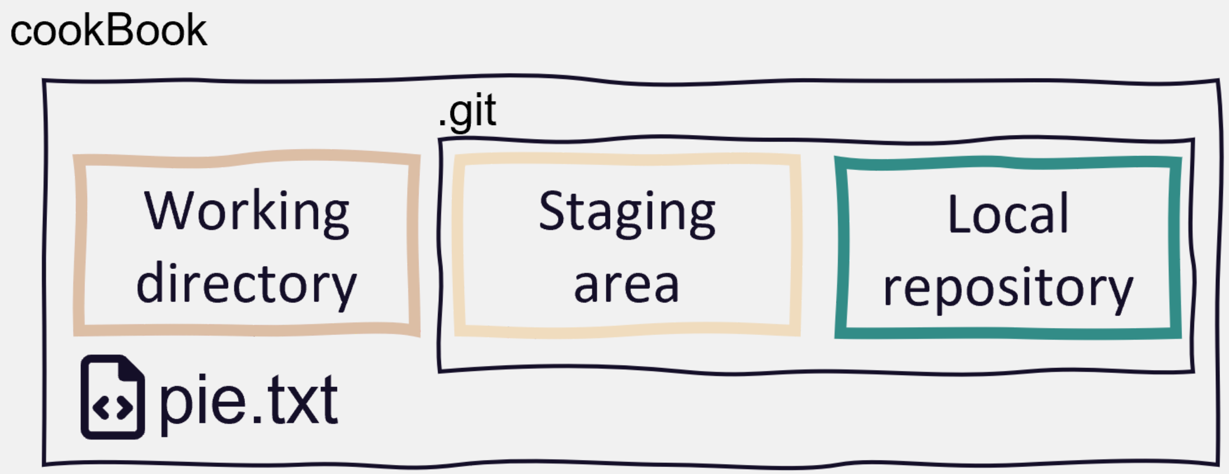
Git detects any changes in the working directory

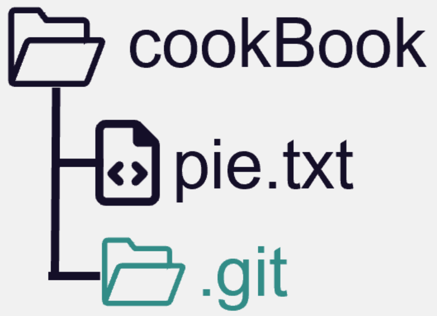




Step 2: Stage changes

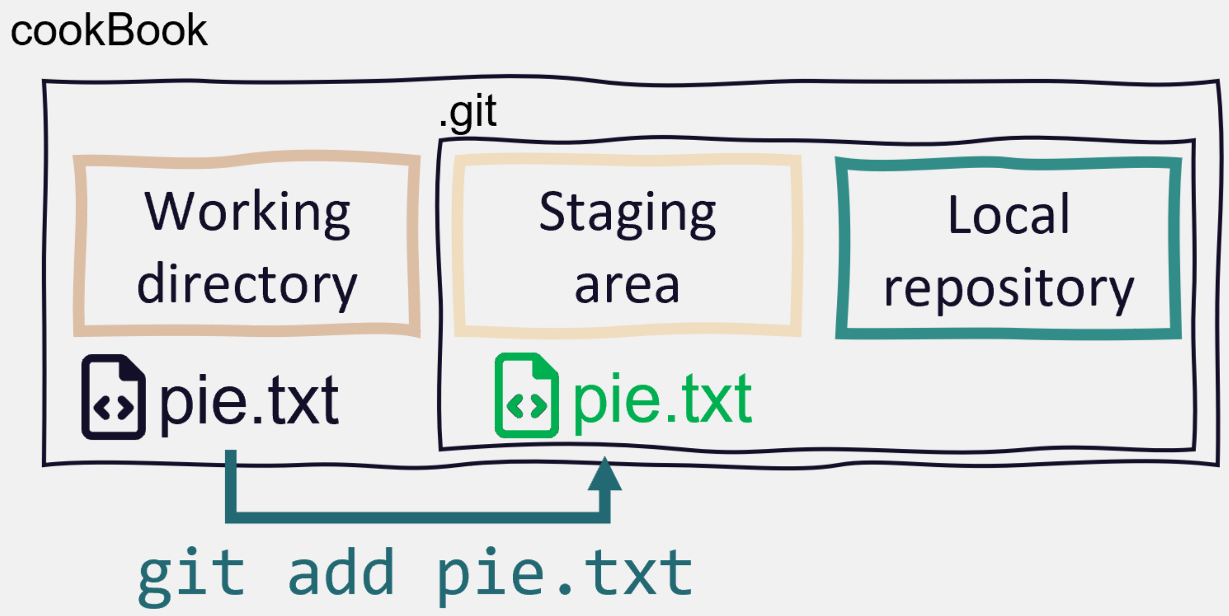
Staging a file means to **list it for the next commit**.



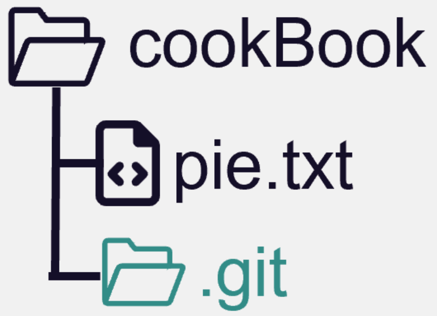


Step 2: Stage changes

Staging a file means to **list it for the next commit**.

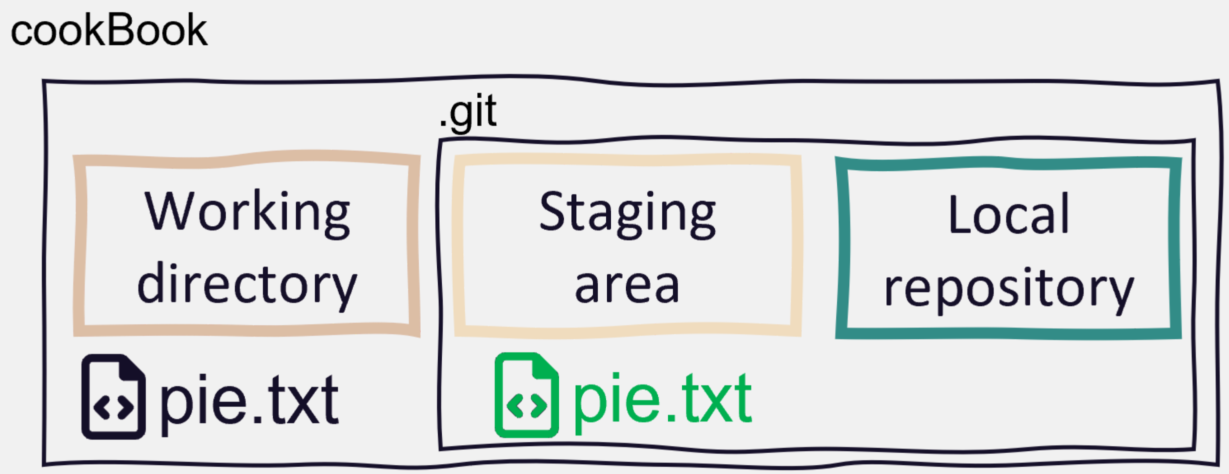


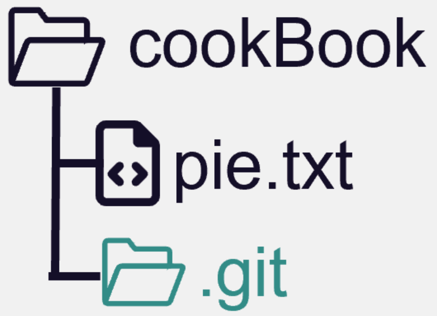
```
git add pie.txt
```



Step 3: Commit changes

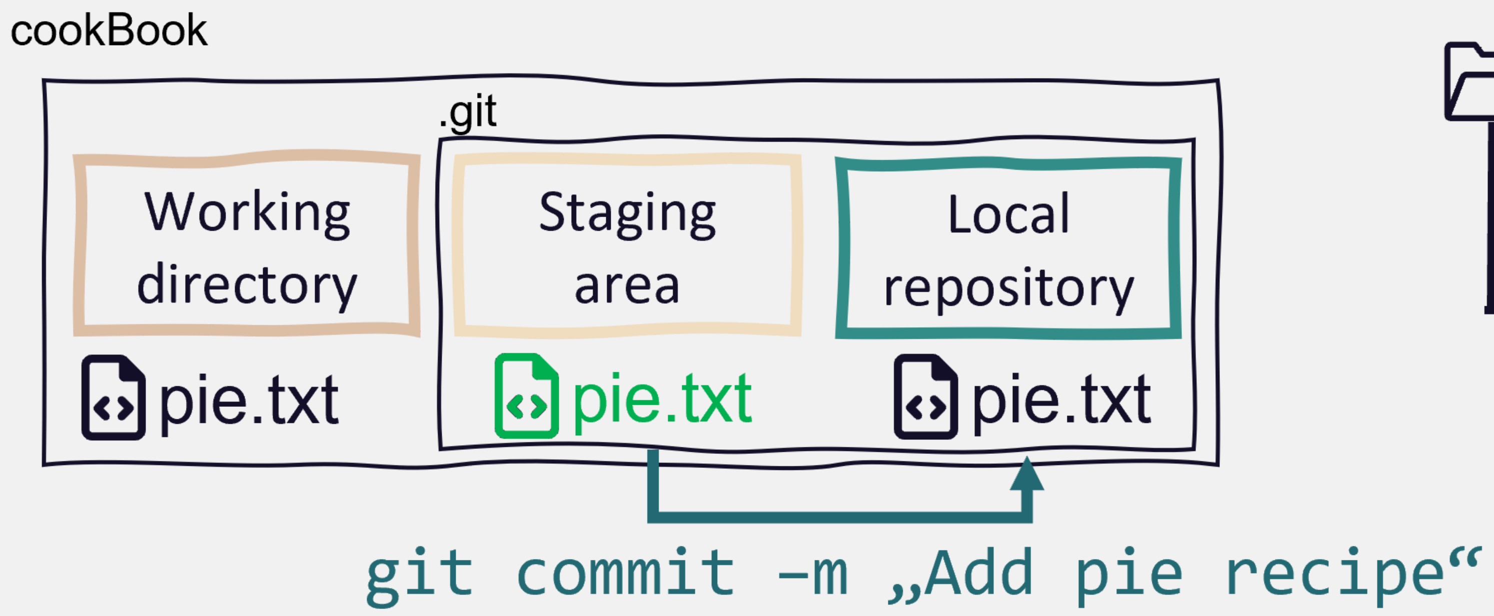
Commits are the snapshots of your project state



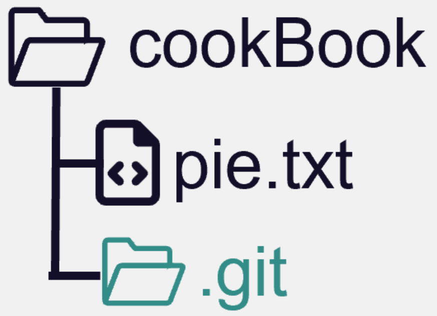


Step 3: Commit changes

Commits are the snapshots of your project state

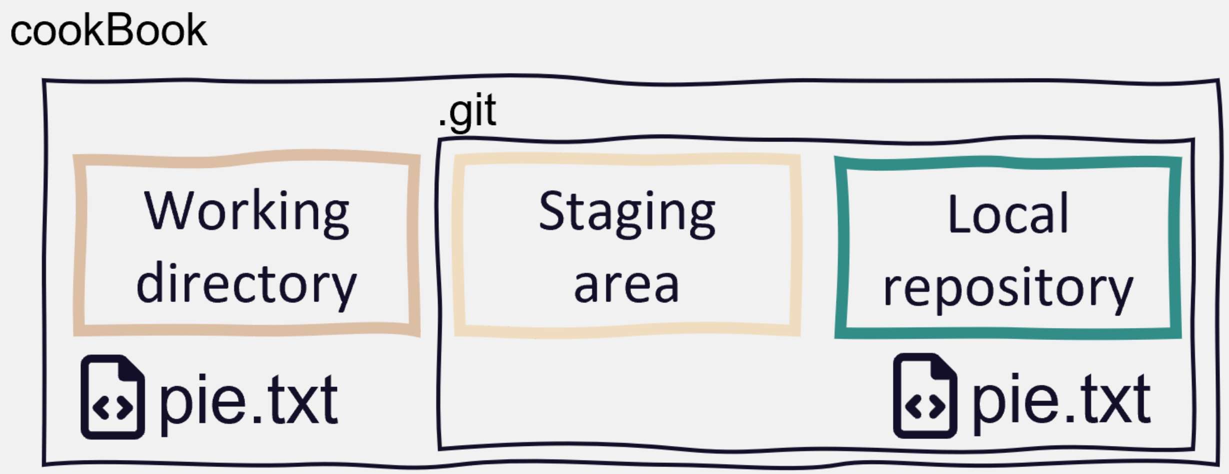


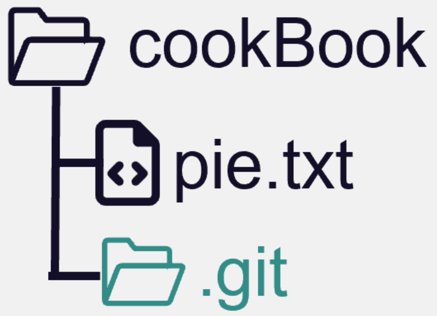
```
git commit -m „Add pie recipe“
```



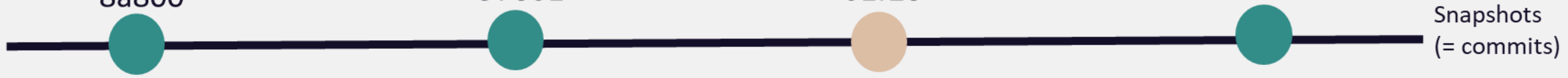
Step 3: Commit changes

Changes are part of Git history and staging area is clear again



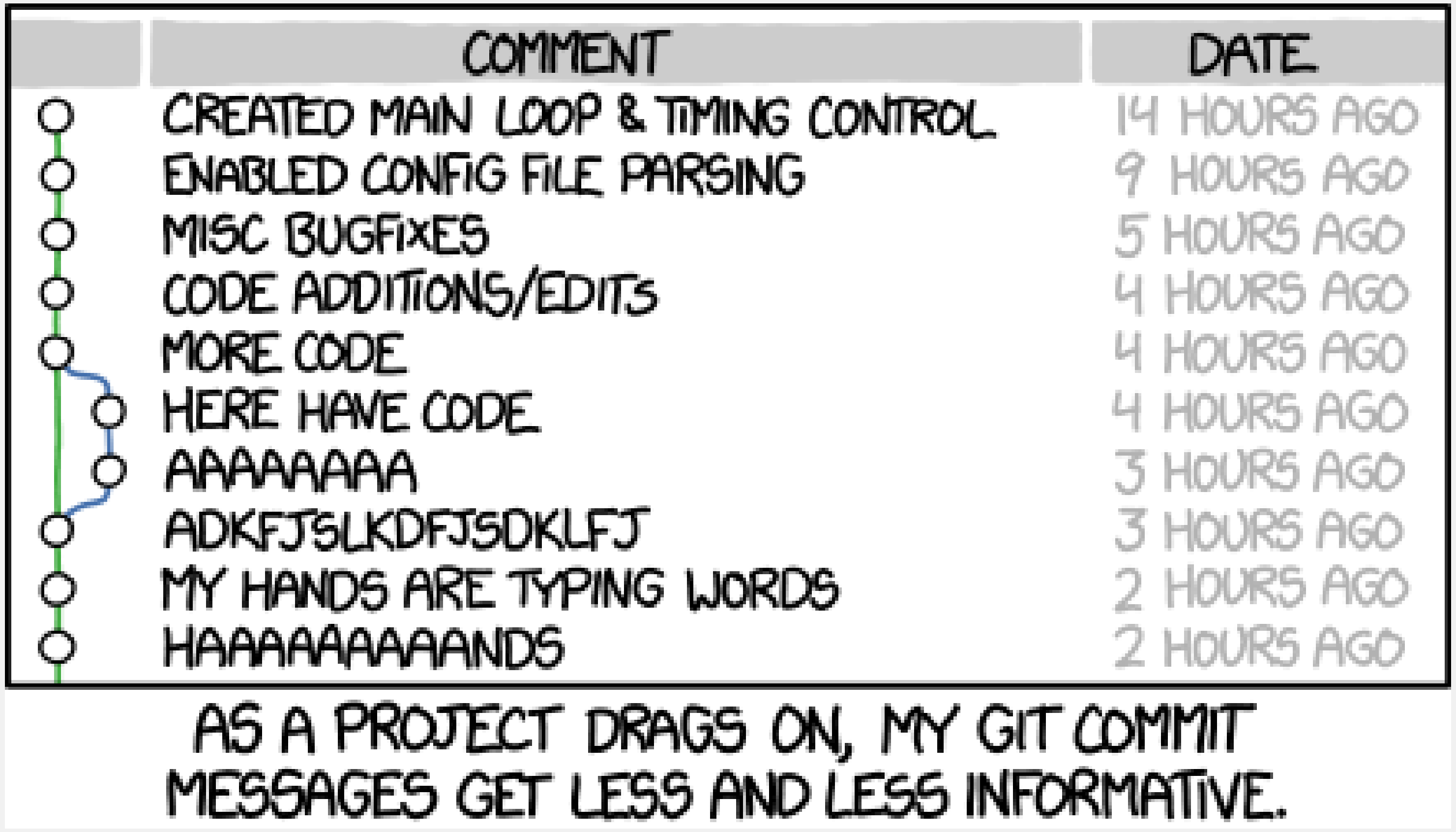


The commit history



8a800	97061	61f20	d96a0
Add pie recipe This is my favourite pie in the world. The recipe comes from my grandfather and he learned it from his neighbor.	**Add lentil stew recipe**	**Replace milk with soy milk** This is better for the environment and for cows	**Add cooking duration** Alice reminded me that I forgot to add the cooking duration. This was bad because the food got burned. This is now fixed.
+Best pie ever +100 ml milk +2 eggs	+stew.txt	- 100 ml milk +100 ml soy milk	+Bake time: 45 min +Cook time: 30 min
Selina Baldauf <selina.baldauf@fu-berlin.de.de>	Selina Baldauf <selina.baldauf@fu-berlin.de.de>	Bob Example <bob.example@email.com>	Selina Baldauf <selina.baldauf@fu-berlin.de.de>
Wed Jul 19 18:23:15 2023 +0200	Wed Jul 19 20:23:15 2023 +0200	Wed Jul 20 15:23:15 2023 +0200	Tue Aug 20 12:25:15 2023 +0200

Good commit messages



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

xkcd on commit messages

Good commit messages

Good commit messages are descriptive and helpful.

✓

```
Add pie recipe

This is my favorite pie in the world.
The recipe comes from my grandfather and
he learned it from his neighbor.
```

✗

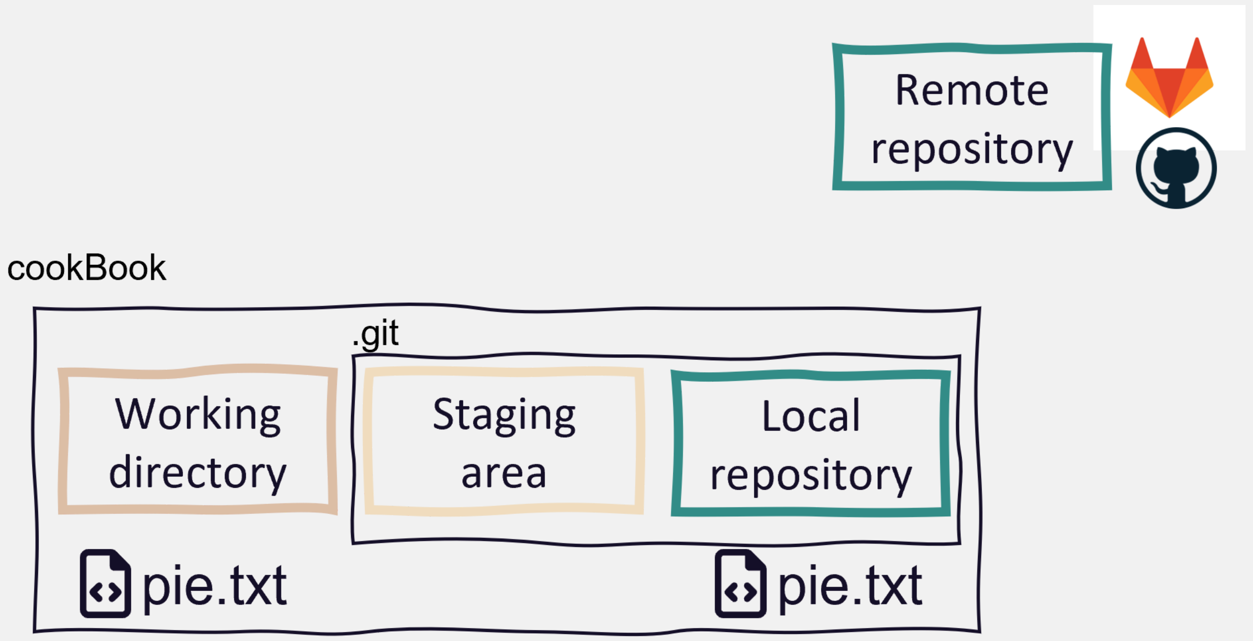
```
added a file.

This is really good.
```

See here for more details on good commit messages.

Step 4: Share changes with the remote repo

Use remote repos (on a server) to **backup**, **synchronize**, **share** and **collaborate**

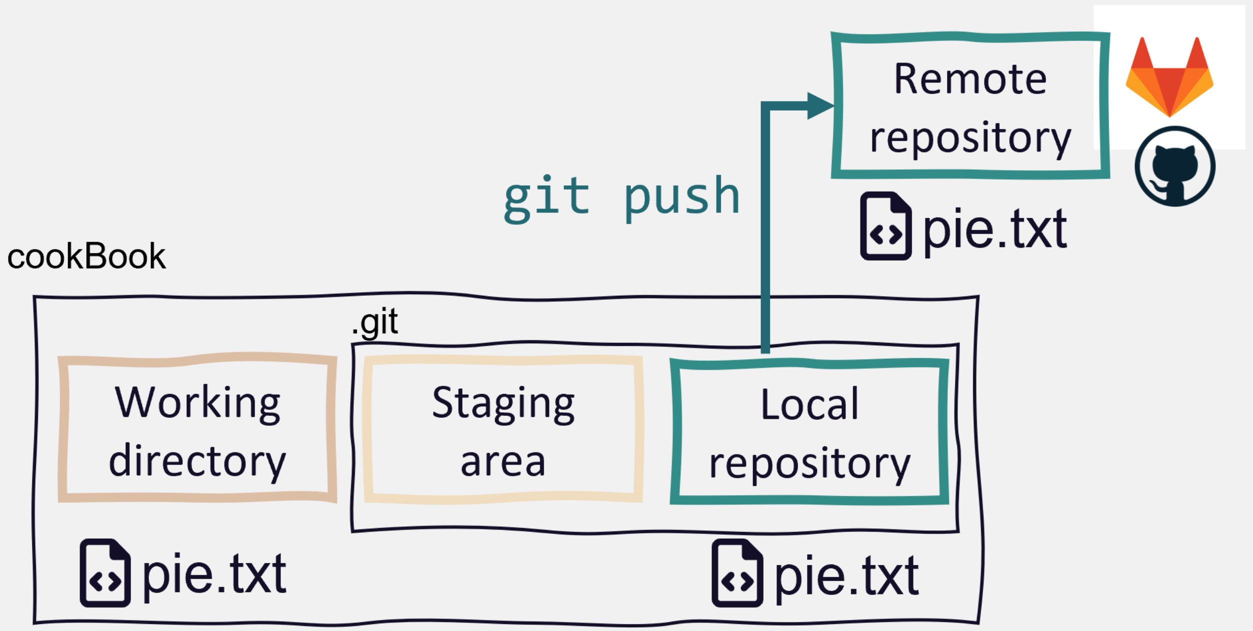
- can be **private** (you + collaborators) or **public** (visible to anyone)



Step 4: Share changes with the remote repo

Use remote repos (on a server) to **backup**, **synchronize**, **share** and **collaborate**

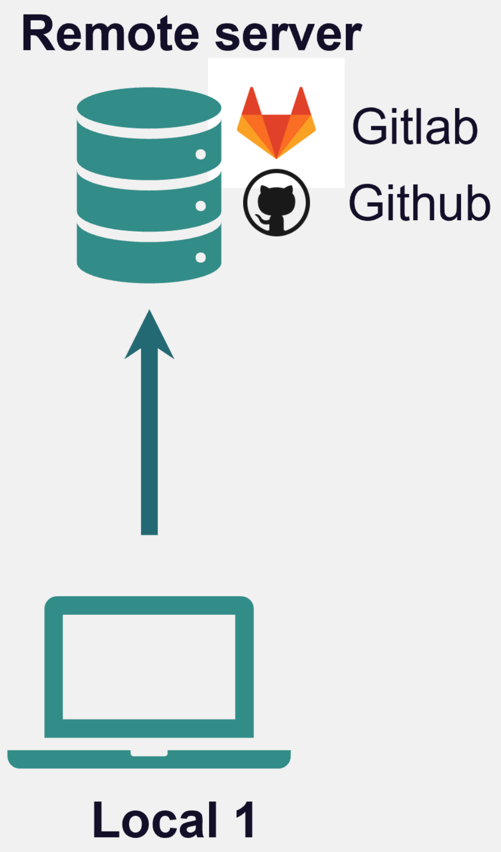
- can be **private** (you + collaborators) or **public** (visible to anyone)



Recap

Basic Git workflow:

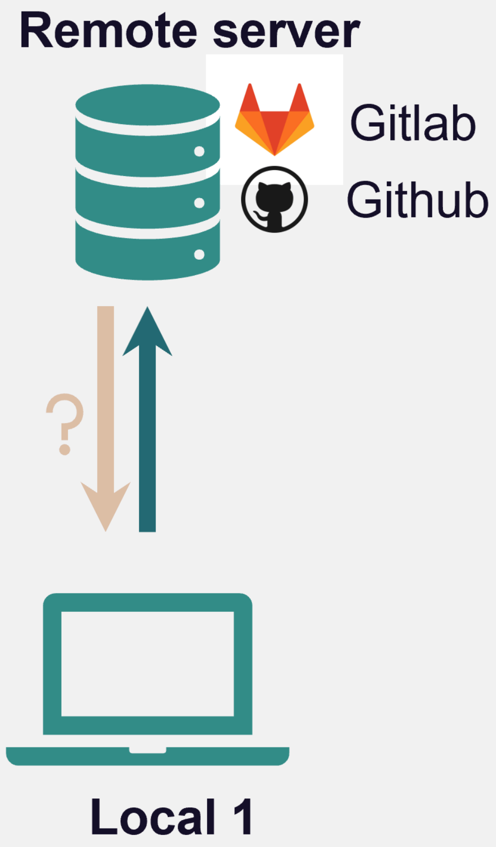
1. **Initialize** a Git repository
2. **Work** on the project
3. **Stage** and **commit** changes to the local repository
4. **Push** to the remote repository



Recap

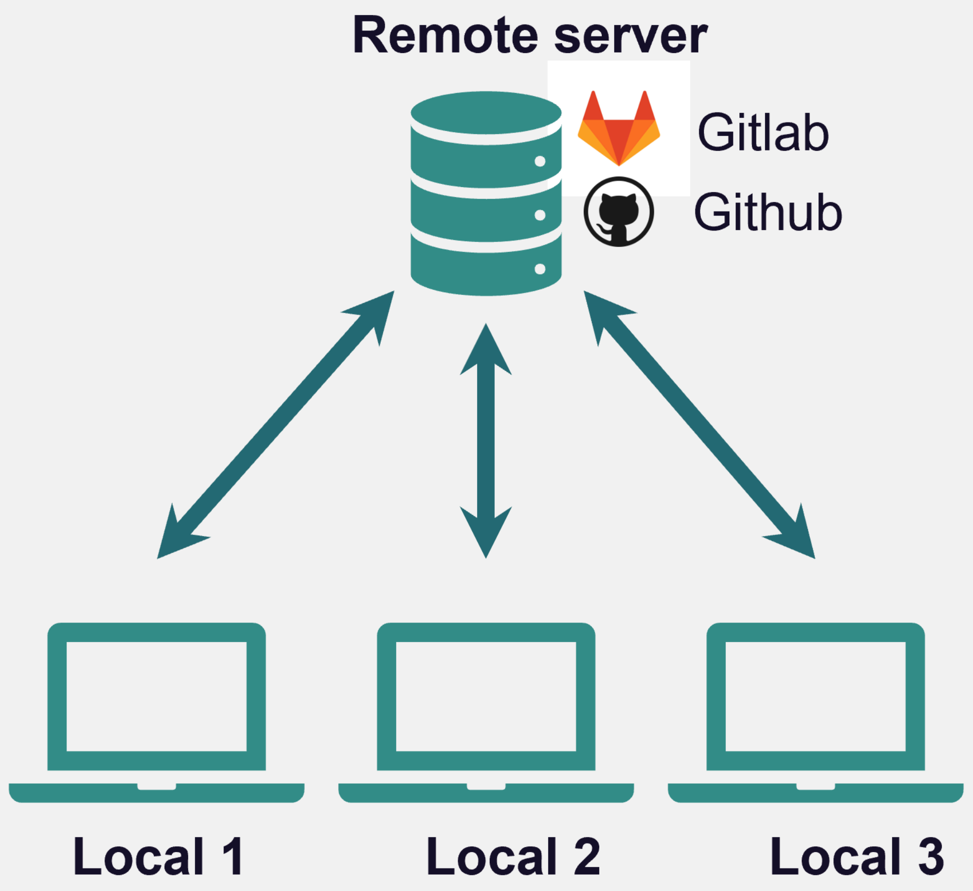
Basic Git workflow:

1. **Initialize** a Git repository
2. **Work** on the project
3. **Stage** and **commit** changes to the local repository
4. **Push** to the remote repository



Recap

Git is a **distributed version control system**

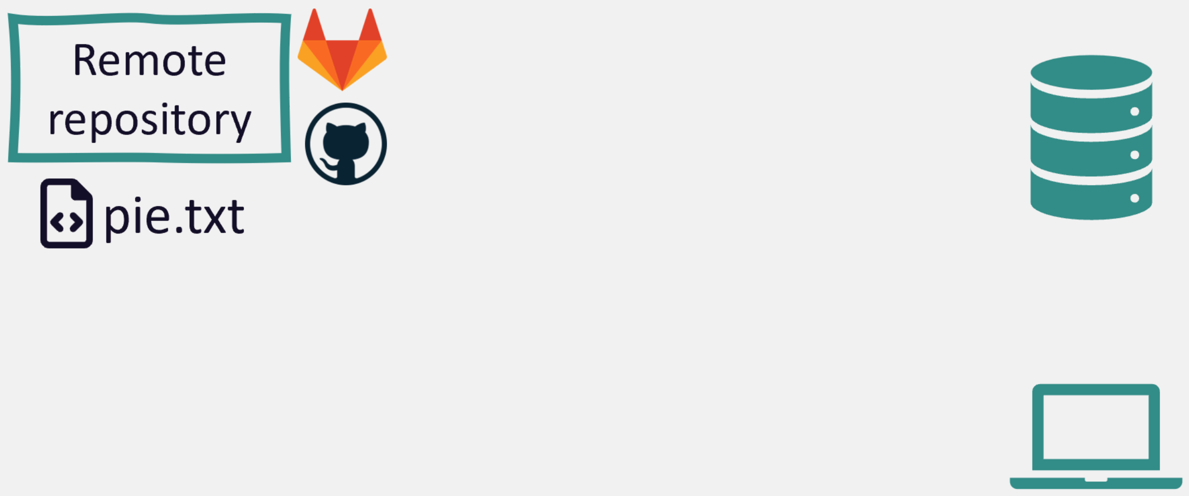


- Idea: many *local* repositories synced via one *remote* repo
- Collaborate with
  - **yourself** on different machines
  - your **colleagues** and friends
  - **strangers** on open source projects

Get a repo from a remote

In Git language, this is called **cloning**

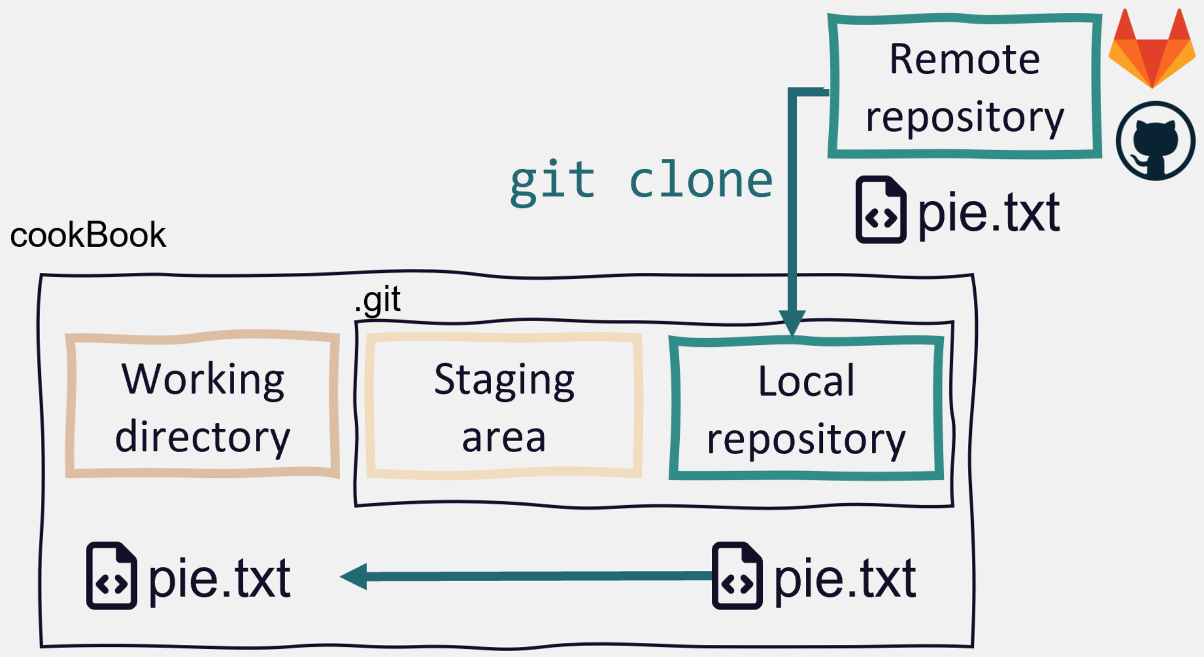
You can clone all public repositories and private repositories if you are a owner/collaborator

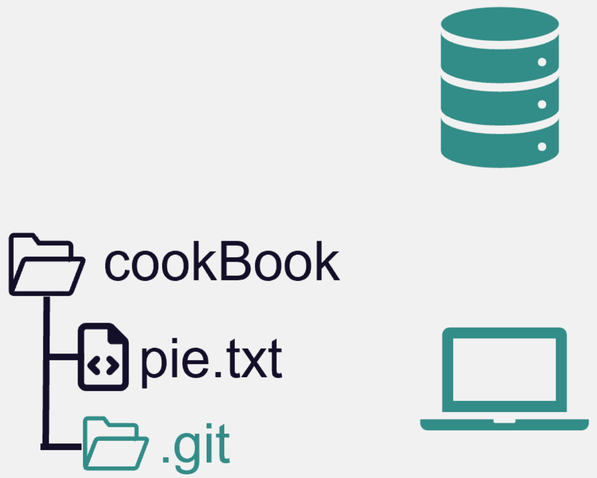


Get a repo from a remote

In Git language, this is called **cloning**

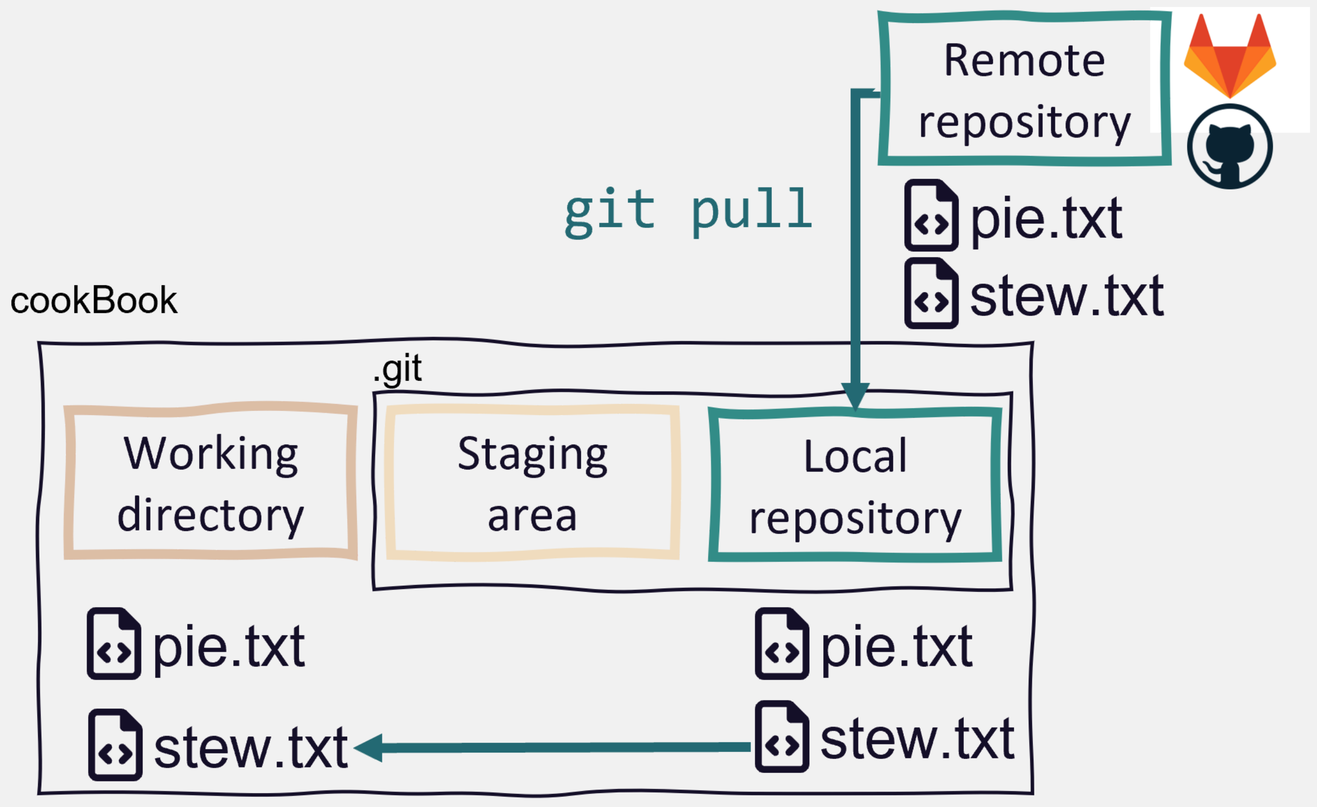
You can clone all public repositories and private repositories if you are a owner/collaborator



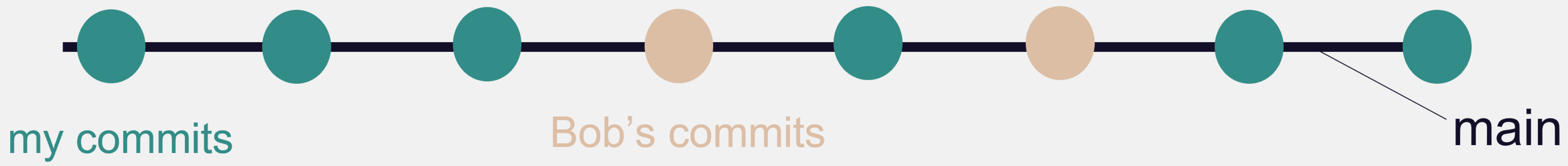


Get changes from the remote

- Local changes, publish to remote: `git push`
- Remote changes, pull to local: `git pull`

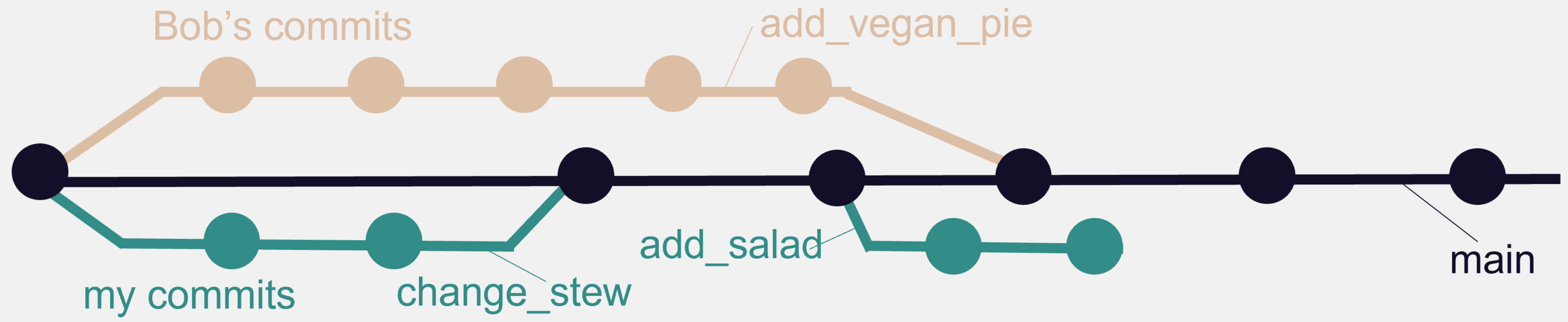


A simple collaboration workflow



- One remote repo on GitHub, multiple local repos
- Idea: Everyone works on the same branch
  - **Pull before** you start **working**
  - **Push after** you finished **working**

A branching-merging workflow



- One remote repo on GitHub, multiple local repos
- Idea: Everyone works on the their **separate branch**
  - **Merge** branch with the main when work is done
- Check out the How-To guide for details

Publishing your work

Remote repositories

- There are **commercial** and **self-hosted** options for your remote repositories
  - Commercial: GitHub, Gitlab, Bitbucket, …
  - Self-hosted: Gitlab (maybe at your institution?)
- Please be aware of your institutional guidelines
  - Servers outside EU
  - Privacy rules might apply depending on type of data

Public repositories

- Making a repository public is a good way to publish and share your work
- Always add a README.md file
- Always add a LICENSE file
  - This is important to clarify how others can use your work
- Connect your repo with Zenodo to get a DOI

If you are interested, browse some nice GitHub repositories for inspiration (e.g. Git training repository, Computational notebooks, Repo to publish code from a manuscript)

Outlook

- Git can do much more than we covered today
  - Complex collaboration workflows with code review steps
  - Rolling back to previous versions
  - Ignoring files from the repository
  - ...
- GitHub et al. offer many more features
  - Issues, pull requests, code review, project management, ...
  - Host websites, wikis, ...
- Start with the basic workflow and build on that

Next lecture

Summer/Conference break in August and September!

Topic of next lecture t.b.a.

📅 16th October 🕓 4-5 p.m. 📍 Webex

🔔 Subscribe to the mailing list

📧 For topic suggestions and/or feedback send me an email

Thanks for your attention

Questions?

Summary of the basic steps

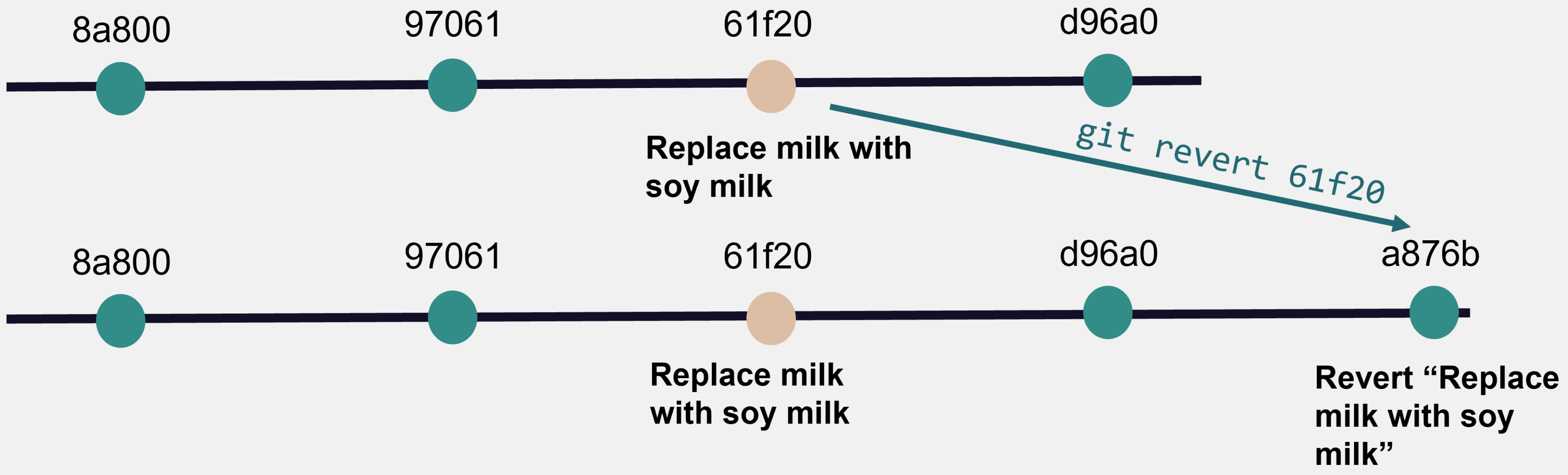
- `git init`: Initialize a git repository
  - Adds a `.git` folder to your working directory
- `git add`: Add files to the staging area
  - This marks the files as being part of the next commit
- `git commit`: Take a snapshot of your current project version
  - Includes time stamp, commit message and information on the person who did the commit
- `git push`: Push new commits to the remote repository
  - Sync your local project version with the remote e.g. on GitHub

Undo things

> `git revert`

Revert changes

- Use `git revert` to revert specific commits
- This does not delete the commit, it creates a **new commit that undoes a previous commit**
  - It's a safe way to undo commited changes

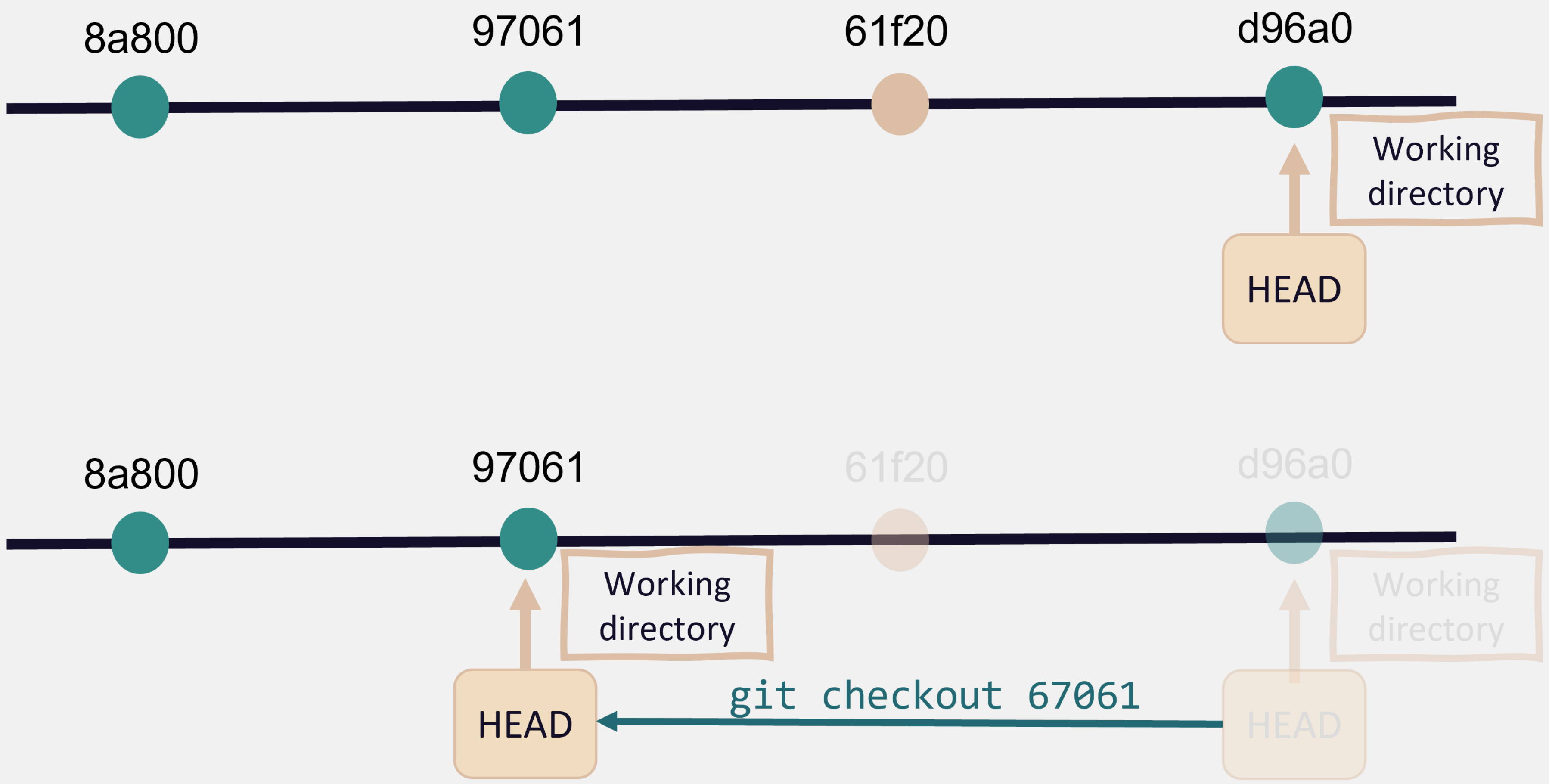

8a800
97061
61f20
d96a0
Replace milk with soy milk
git revert 61f20
8a800
97061
61f20
d96a0
a876b
Replace milk with soy milk
Revert "Replace milk with soy milk"

Go back in time

> `git checkout`

Checkout a previous commit

Take your work space back in time temporarily with `git checkout`



Ignoring files with `.gitignore`

Ignore files with `.gitignore`

- Useful to ignore e.g.
  - Compiled code and build directories
  - Log files
  - Hidden system files
  - Personal IDE config files
  - …

Ignore files with `.gitignore`

- Create a file with the name `.gitignore` in working directory
- Add all files and directories you want to ignore to the `.gitignore` file

Example

```
*.html    # ignore all .html files
*.pdf     # ignore all .pdf files

debug.log # ignore the file debug.log

build/    # ignore all files in subdirectory build
```

See here for more ignore patterns that you can use.