Introduction to version control with Git

Day 1: Concepts and a basic workflow

Selina Baldauf

September 16, 2023

Welcome to the workshop

Who am I?

- Scientific programmer theoretical ecology lab

. . .

Who are you?

- Large range of topics and reserach interests
- Different prior Git experience and expectations

Welcome to the workshop

Git is a huge topic because and git is very powerful. Most of the things are best to learn as you need them.

Aim of this workshop

Get started with simple workflows. Learn the rest as you need it.

Session 1

- Introduction to Git concepts
- Simple Git workflow for your own projects

Session 2

- Collaborate using Git and Github

Before we start

Organisation

- Today and tomorrow: 🕑 2.30 - 4 p.m.
  - Theoretical input + practical exercises
- Next Monday: 🕑 2.30 - 3.30 p.m.
  - Clarify questions and problems, talk about more advanced features, …
  - Until then: Start working on a small Git project
- Material is all online

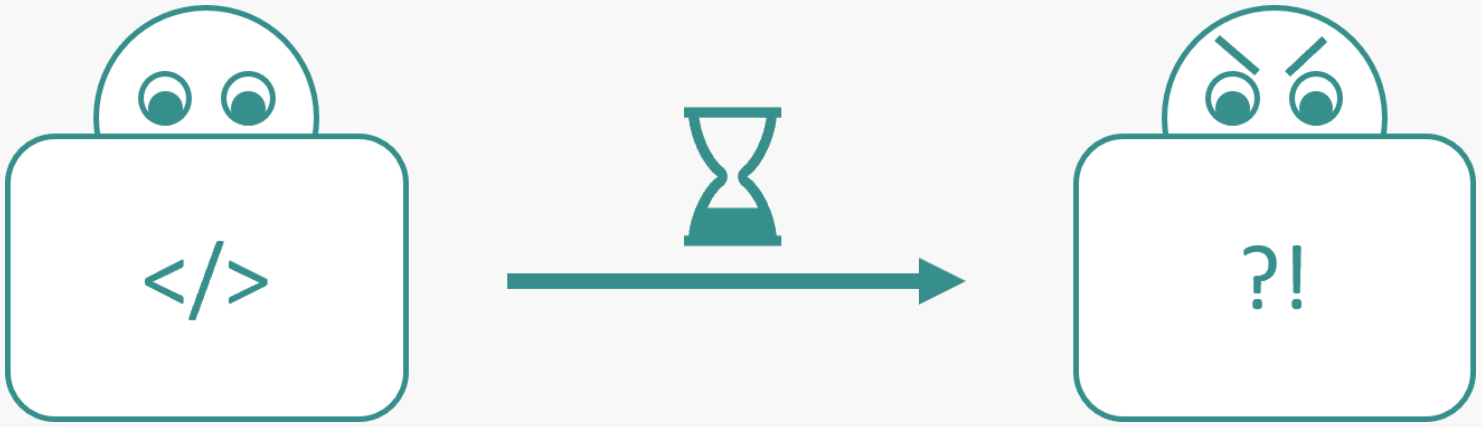
Before we start

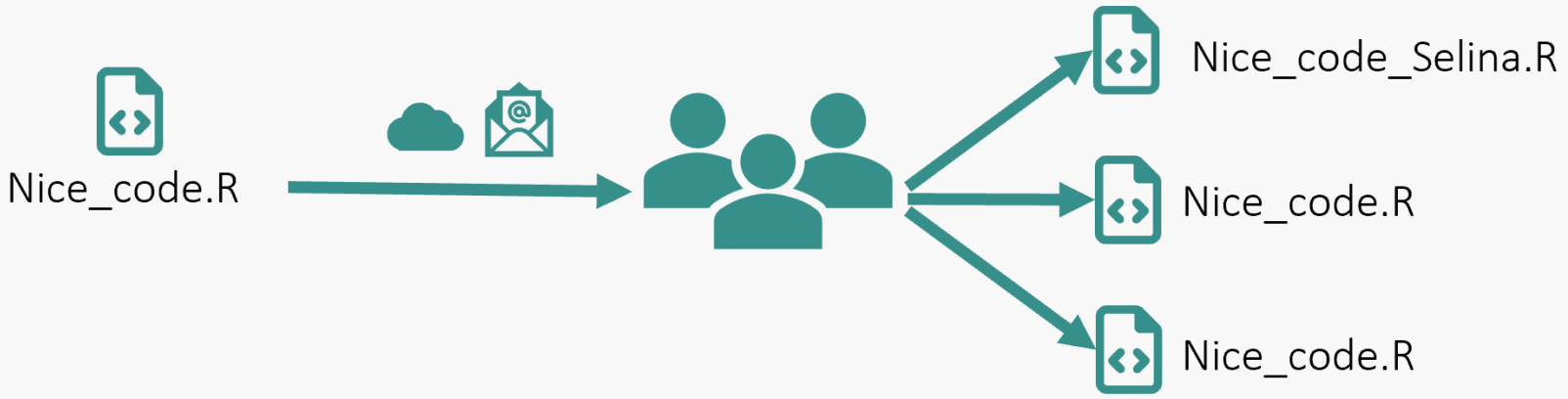
Did anyone have problems with the workshop preparation?

Let’s get started

Why version control?

Two examples in which proper version control can be a life/time saver



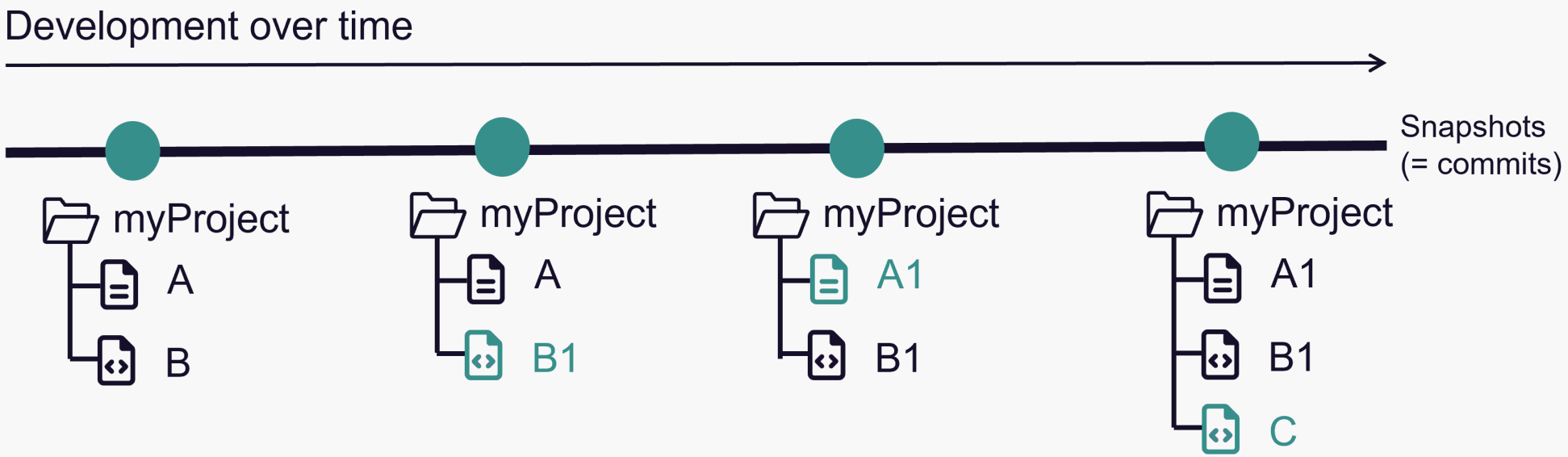


Version control with Git

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

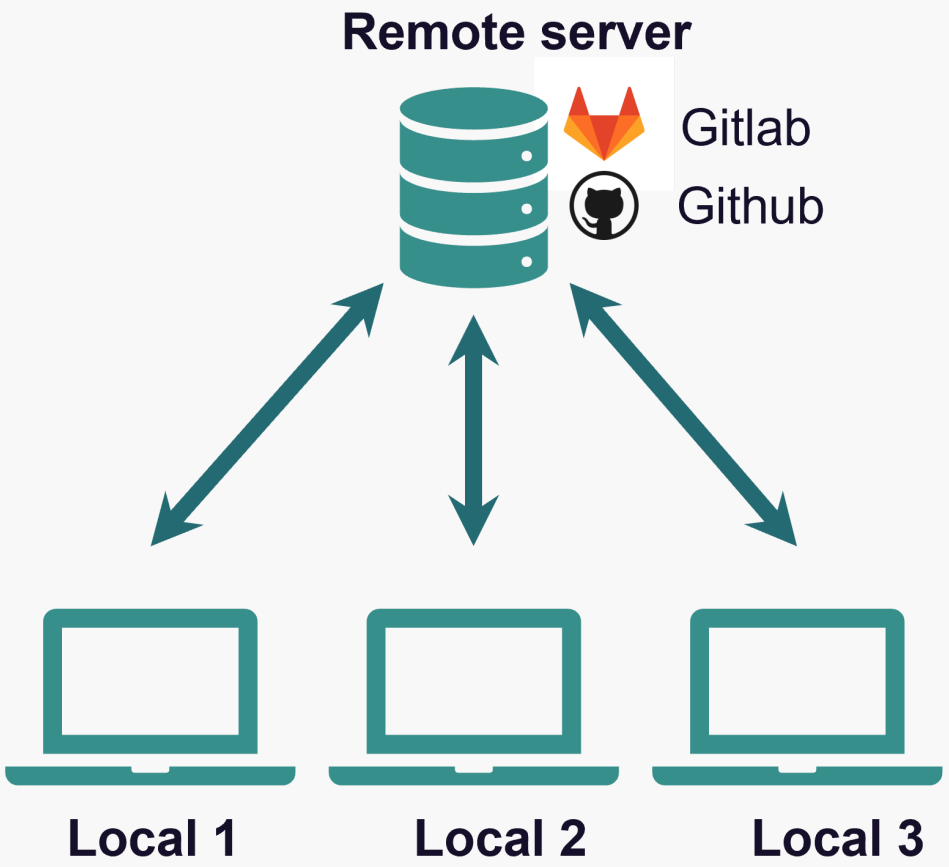
The basic idea of Git

- 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
- Everyone has a complete copy of the 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-G0RM7MS MINGW64 ~/Files_Selina
$ cd Repos/02_workshops/first_git_project/

Selina_User@DESKTOP-G0RM7MS 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-G0RM7MS MINGW64 ~/Files_Selina/Repos/02_workshops/first_git_
project (master)
$
```

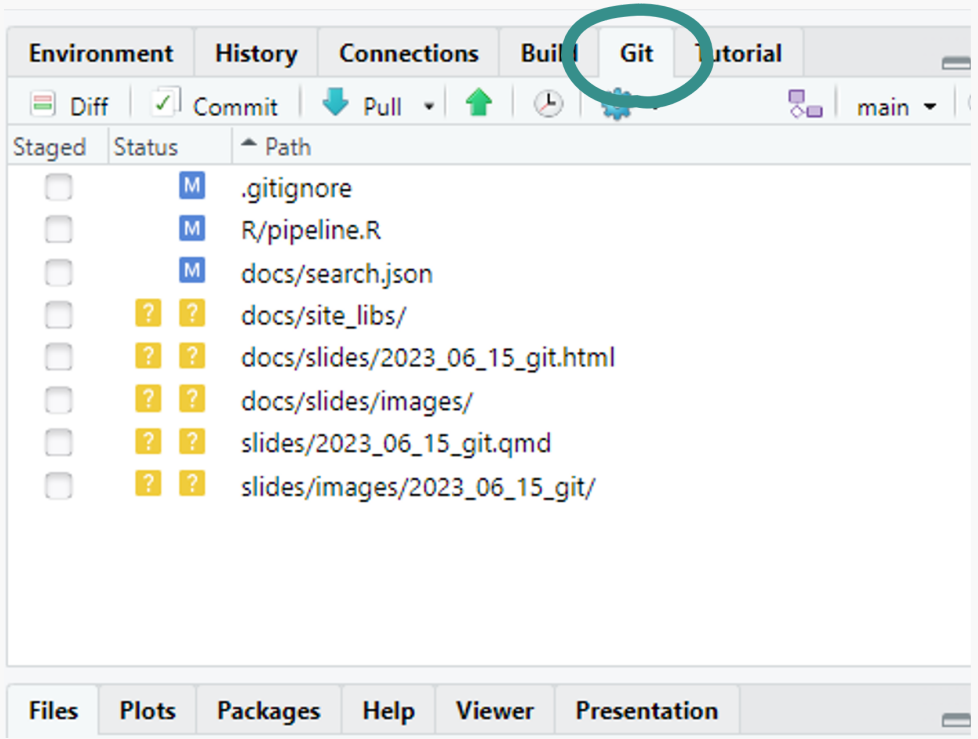
➕ Most control

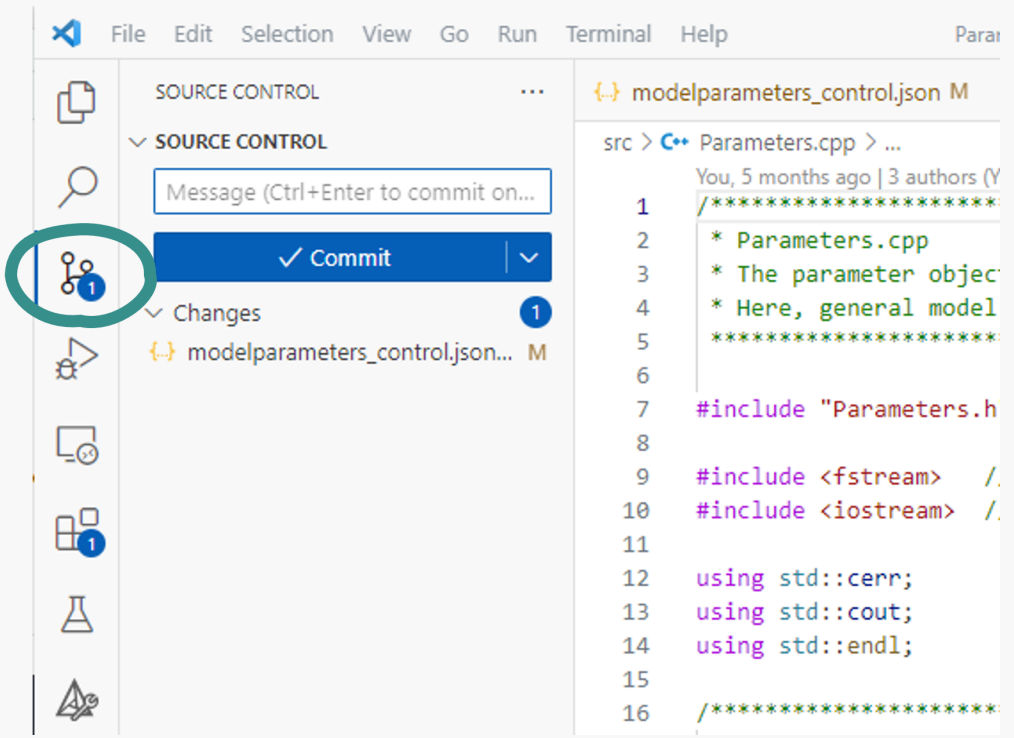
➕ A lot of help/answers online

➖ 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





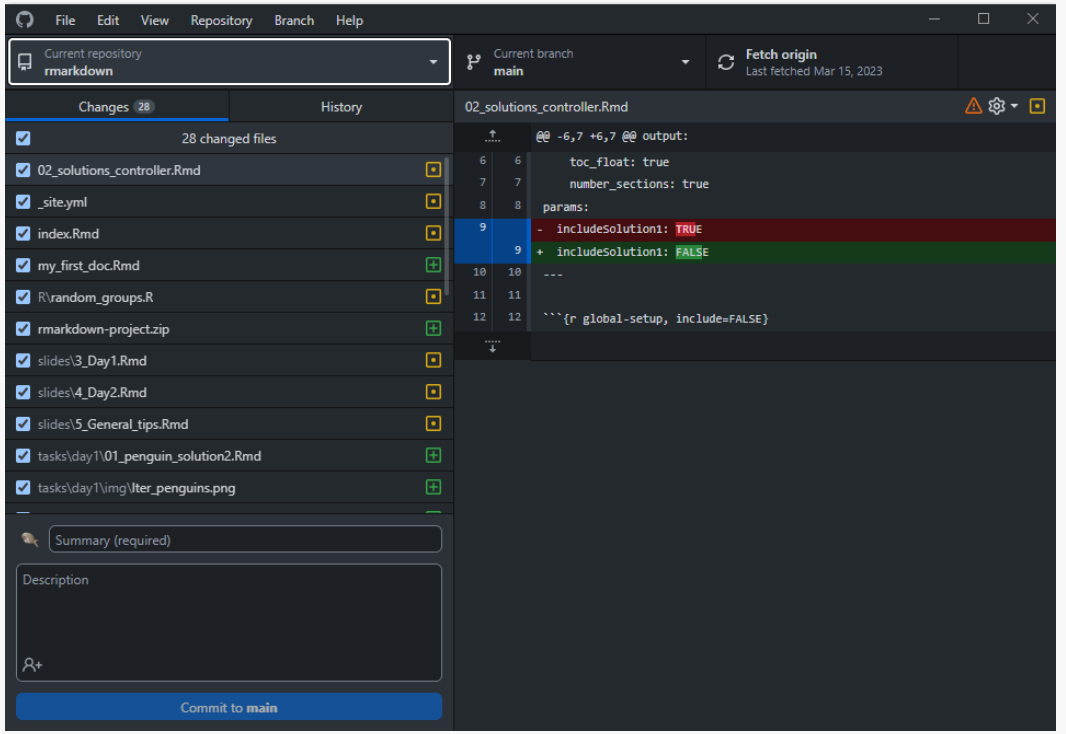
➕ Easy and intuitive

➕ Stay inside IDE

➖ Different for every program

How to use Git - Standalone GUIs

Standalone Git GUI software, e.g. Github Desktop



➕ Easy and intuitive

➕ Nice integration with Github

➖ Switch programs to use Git

How to use Git

Which one to choose?

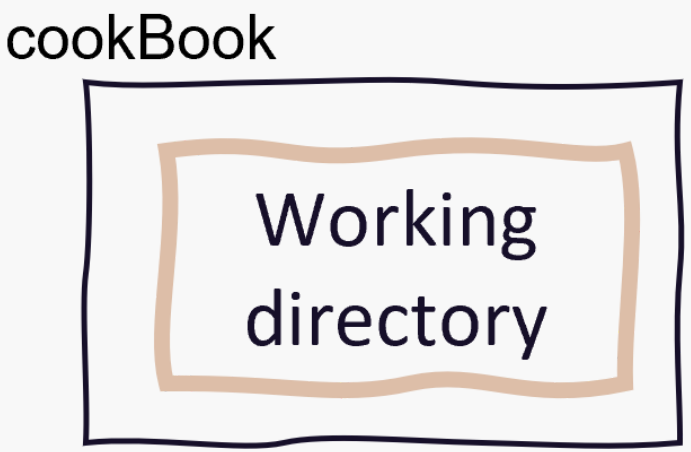
- Depends on experience and taste
- You can mix methods because they are all interfaces to the same Git
- We will use Github Desktop
  - I'll still mention the corresponding terminal terminology at times

> **Tip**
>
> Have a look at the website where you find **How-To guides for the other methods** as well.

The basic Git workflow

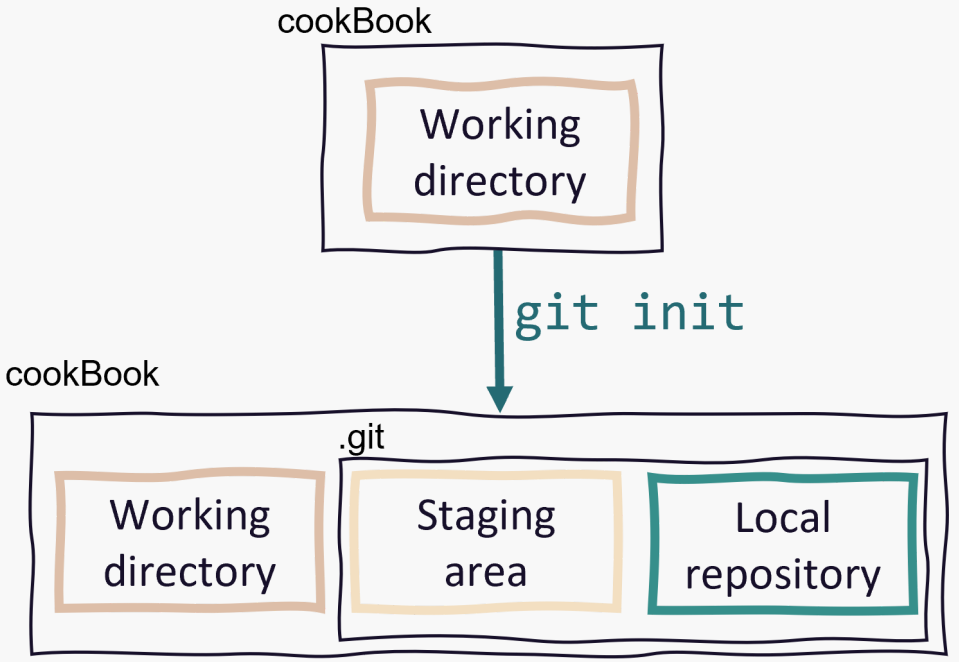
> `git init`, `git add`, `git commit`, `git push`

Step 0: An empty project



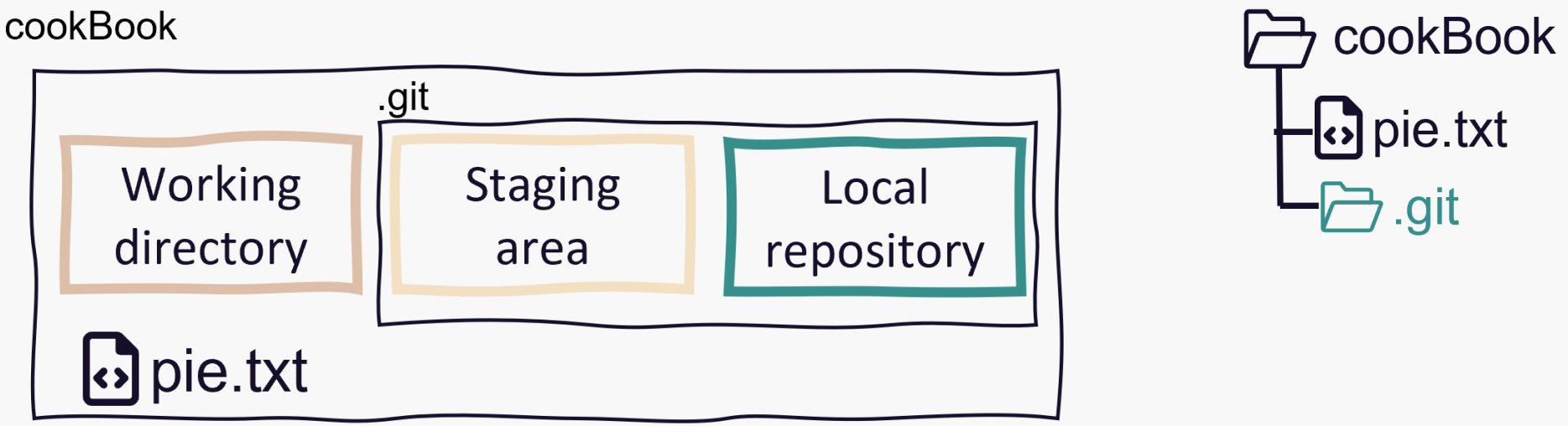
Step 1: Initialize a git repository

- Adds a (hidden) `.git` folder to your project
  - You don't need to interact with this folder directly



Step 2: Modify files and stage changes

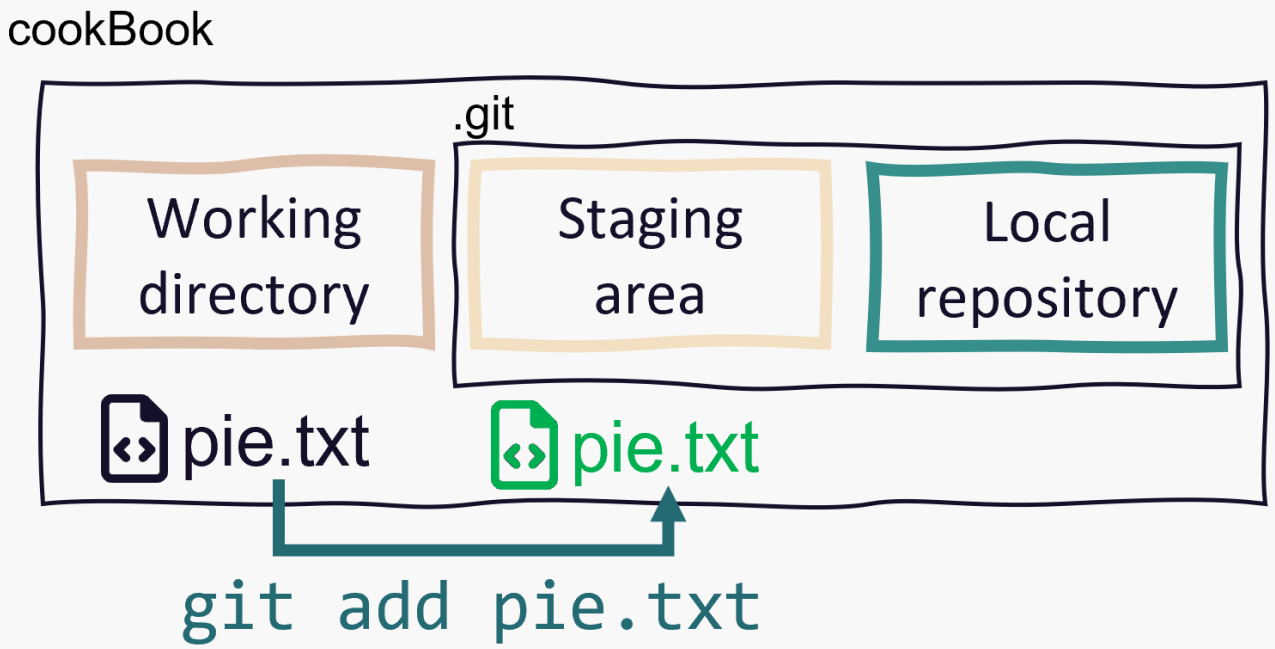
Git detects any changes in the working directory

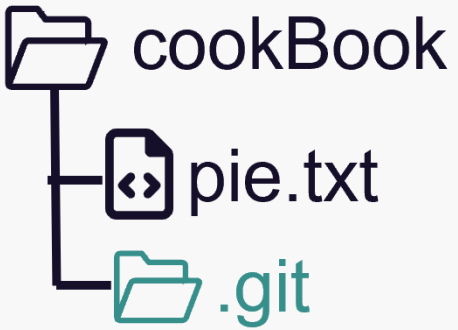


Step 2: Modify files and stage changes

Stage file to be part of the next commit (snapshot)

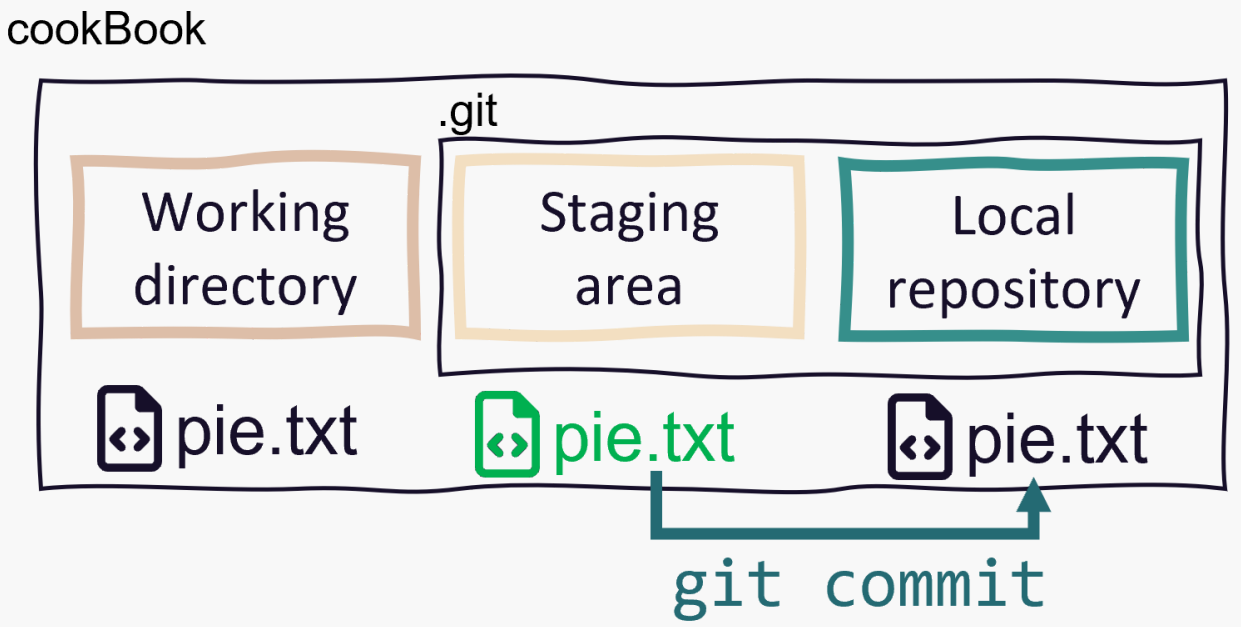
- In the terminal use `git add`
- In GUIs just a check box

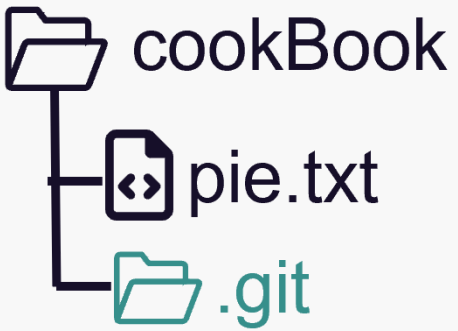




Step 3: Commit changes

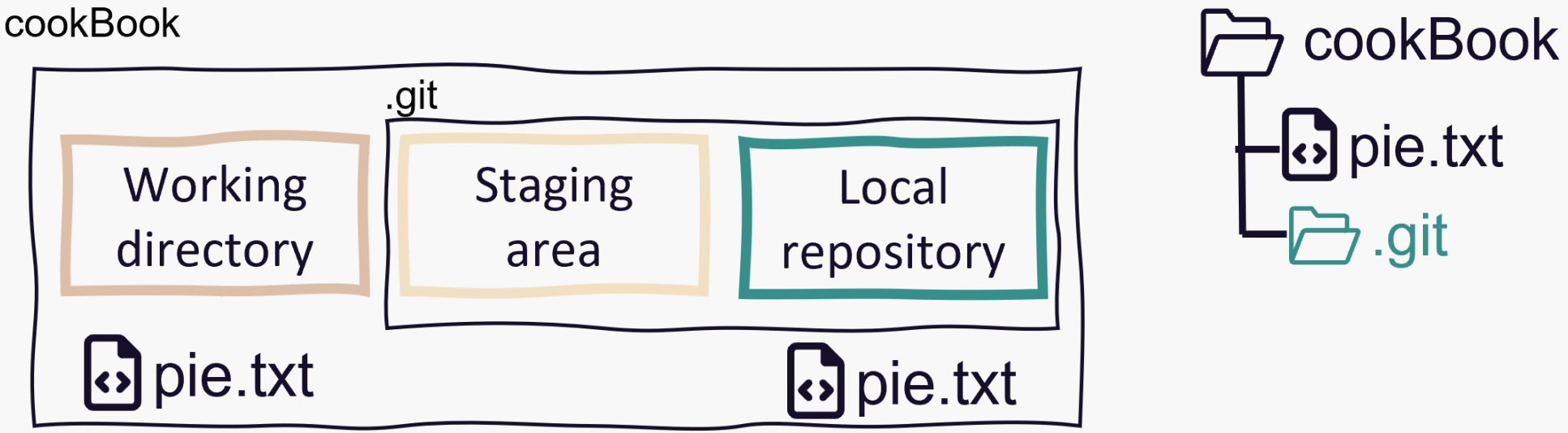
- Commits are the snapshots of your project state
- Commit a bundle of changes from staging area to local repo
- Collect meaningful chunks of work in the staging area, then commit





Step 3: Commit changes

- After a commit, the staging area is clear again
- Changes are now part of the project's Git history



Now you

> Add some recipes to your cook book

How to write good commit messages?

xkcd on commit messages

How to write good commit messages?

See here for more details but some general rules:

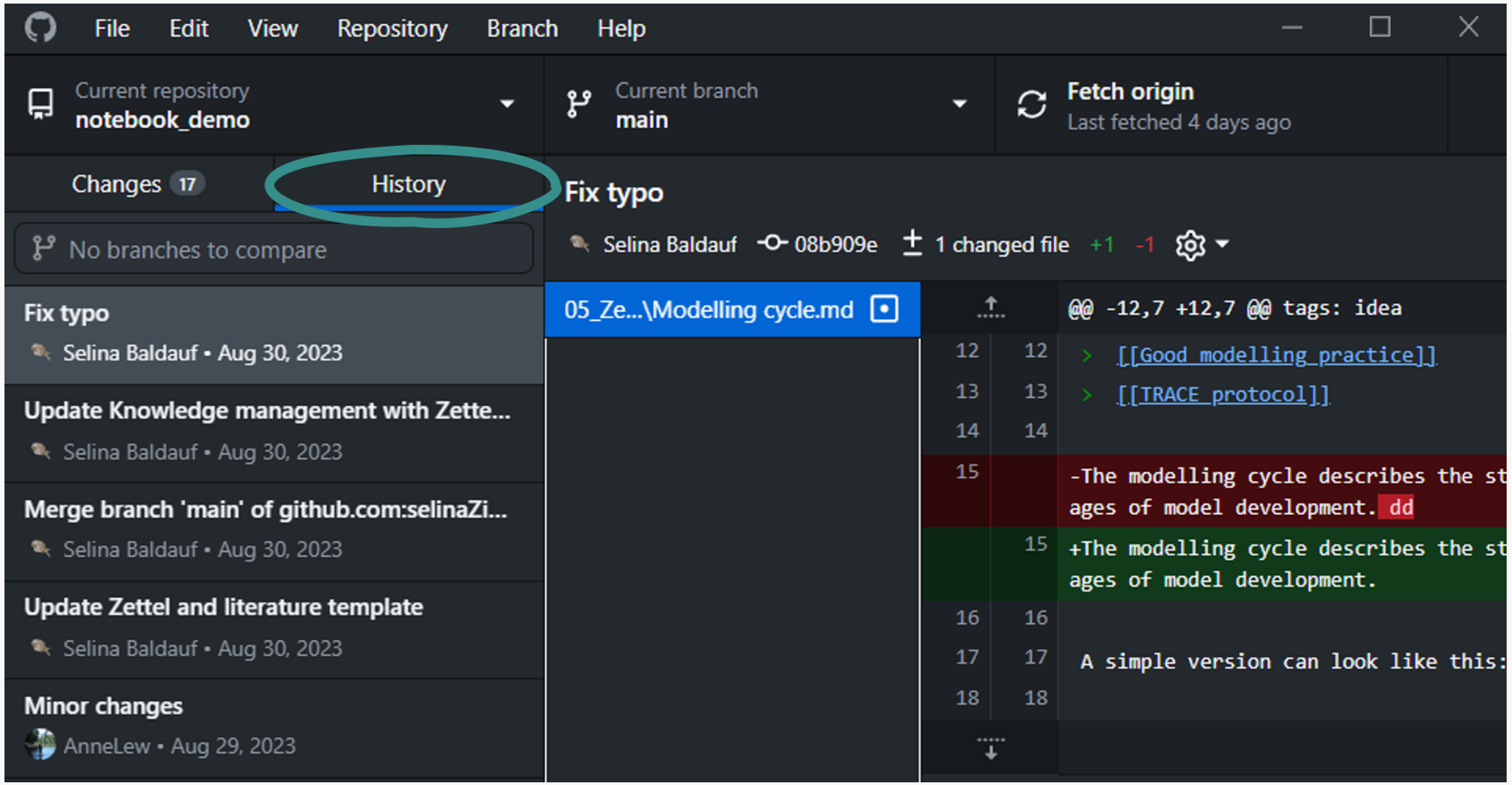
1. Limit summary line to 50 characters
2. Capitalize summary line
3. Do not end summary line with period
4. Use imperative mood in the subject line
5. Use the *Description* to explain **what** and **why**, not **how**

How to write good commit messages?

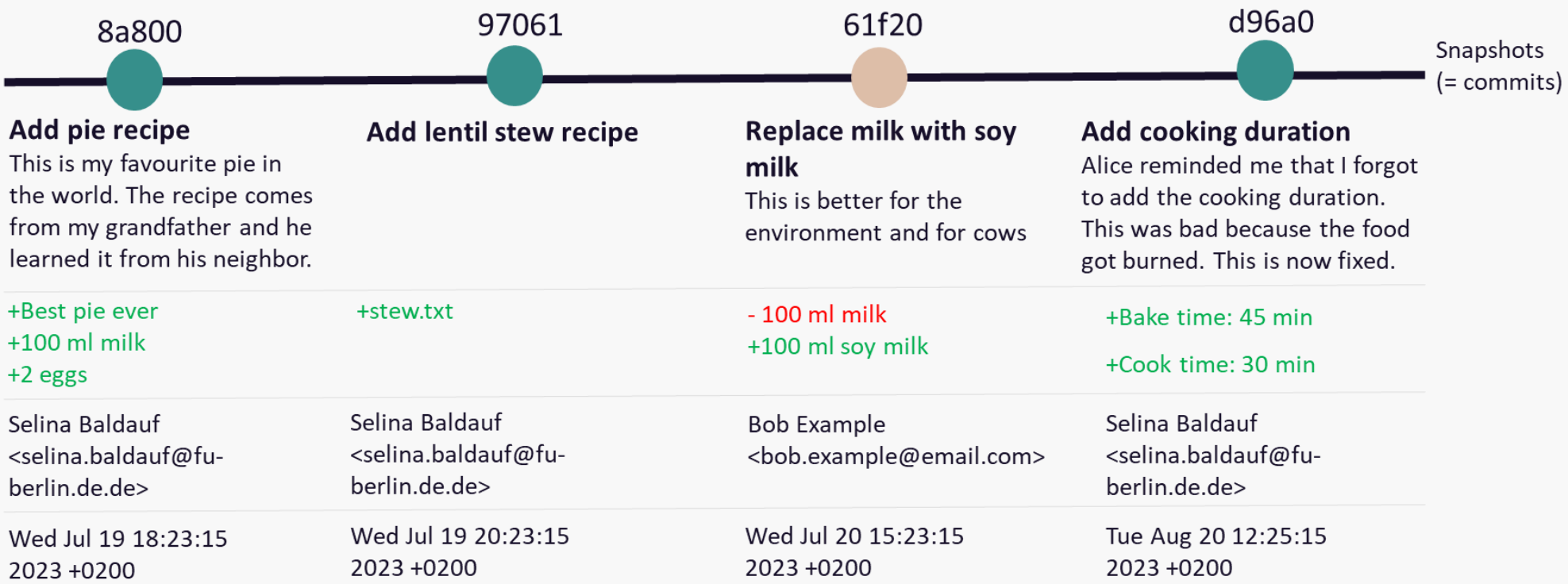
```
Limit model temperature range

I modified the temperature range the mode
operate on to an upper limit of 40°C. Thi
fixes the following problems:

- no more unrealistic results because
temperature cannot exceed the meaningful
- the program does not terminate with eri
code 123 anymore
```

```
limited model temperature range.

Temperatures above 40°C are unrealistic.
```

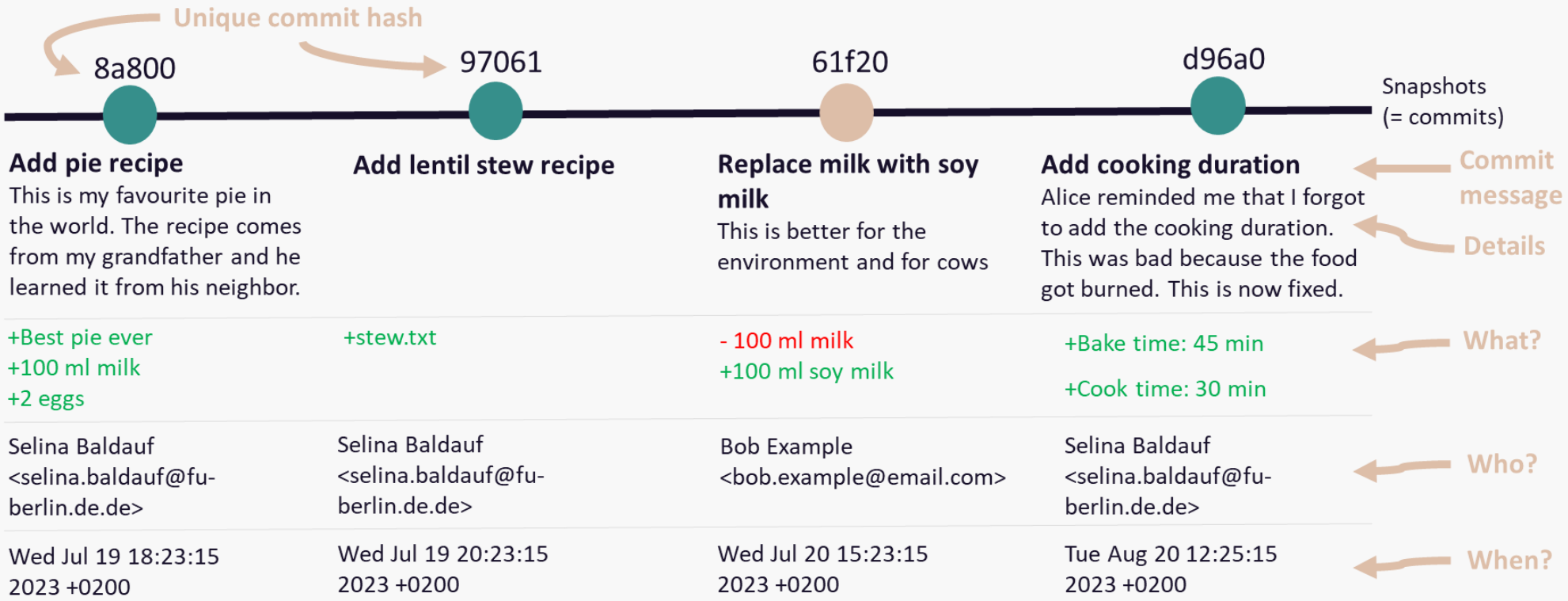
The commit history



The commit history

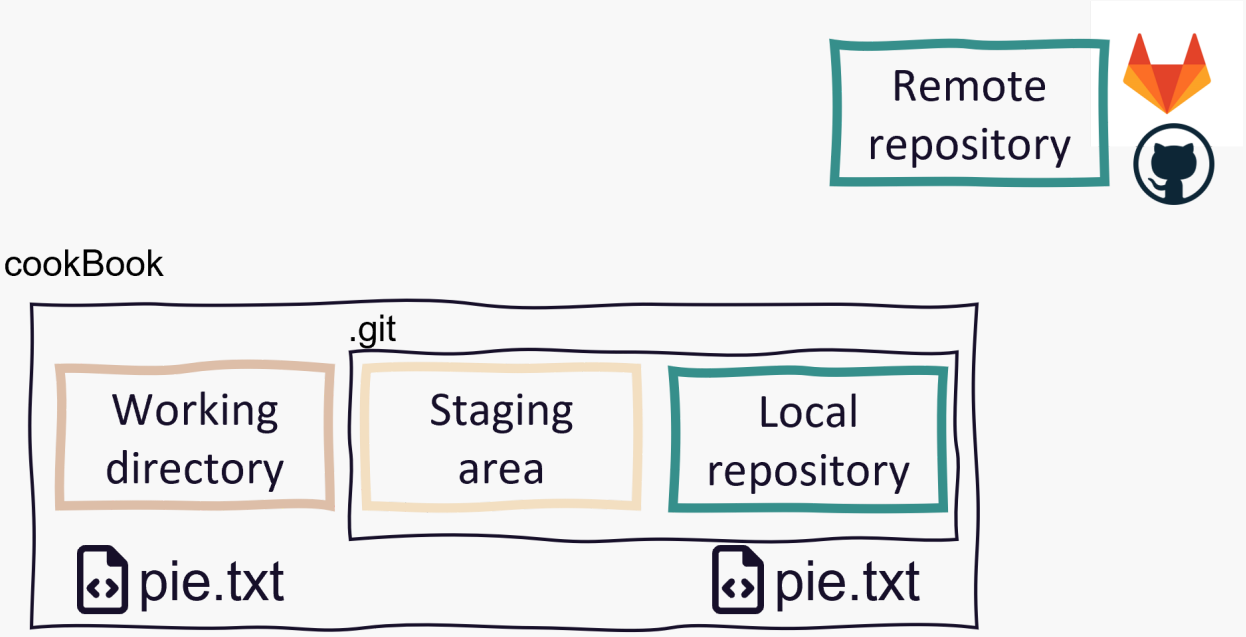


The commit history



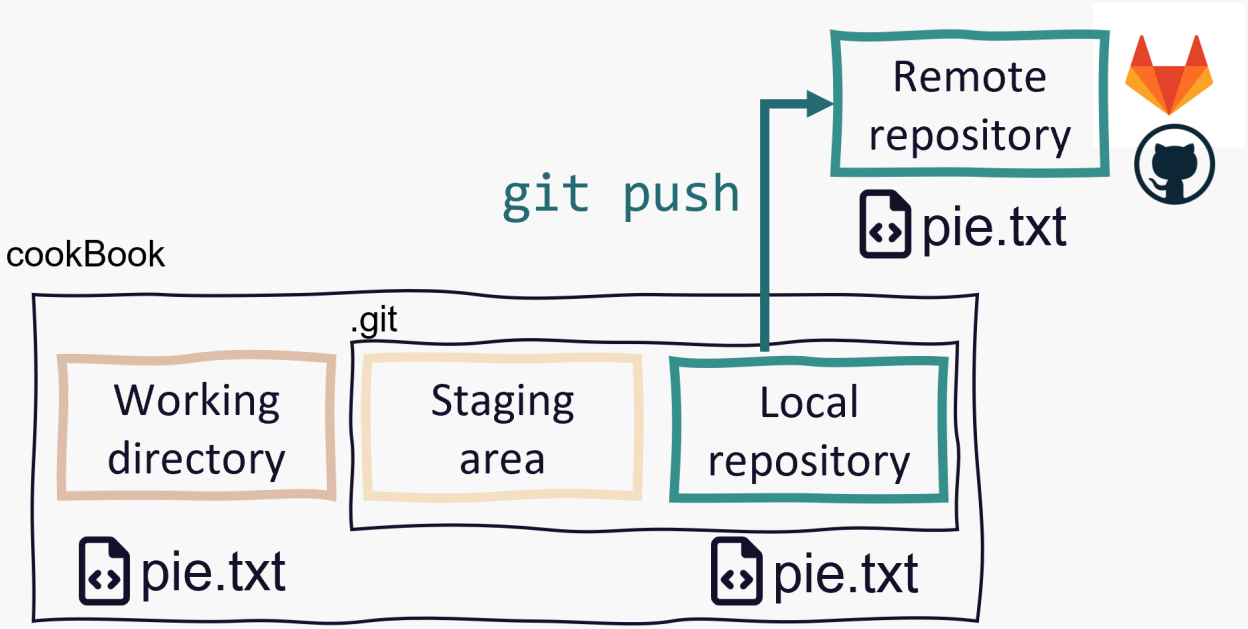
Step 4: Create and connect a remote repo

- Use remote repos (on a server) to *synchronize*, *share* and *collaborate*
- Remote repos can be *private* (you + collaborators) or *public* (visible to anyone)



Step 5: Share changes with the remote repo

- Push your local changes to the remote with `git push`



Now you

> Publish your cook book on Github (please make it public)

A word on remote repositories

- There are commercial and self-hosted options for your remote repositories
  - Commercial: Github, Gitlab, Bitbucket, …
  - Self-hosted: Gitlab (maybe at your institution?)
- For the commercial options please be aware of your institutional guidelines
  - Servers are likely outside EU
  - Privacy rules might apply

Summary of the basic steps

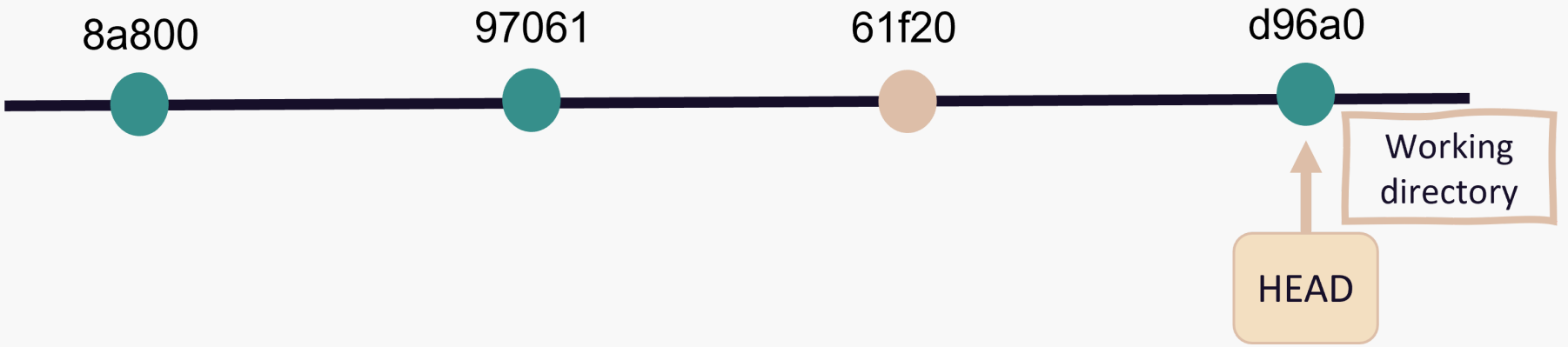
- `git init`: Initialize a git repository
  - Adds a `.git` folder to your working directory
- `git add`: Add files to the staging area
  - This marks the files as being part of the next commit
- `git commit`: Take a snapshot of your current project version
  - Includes time stamp, commit message and 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

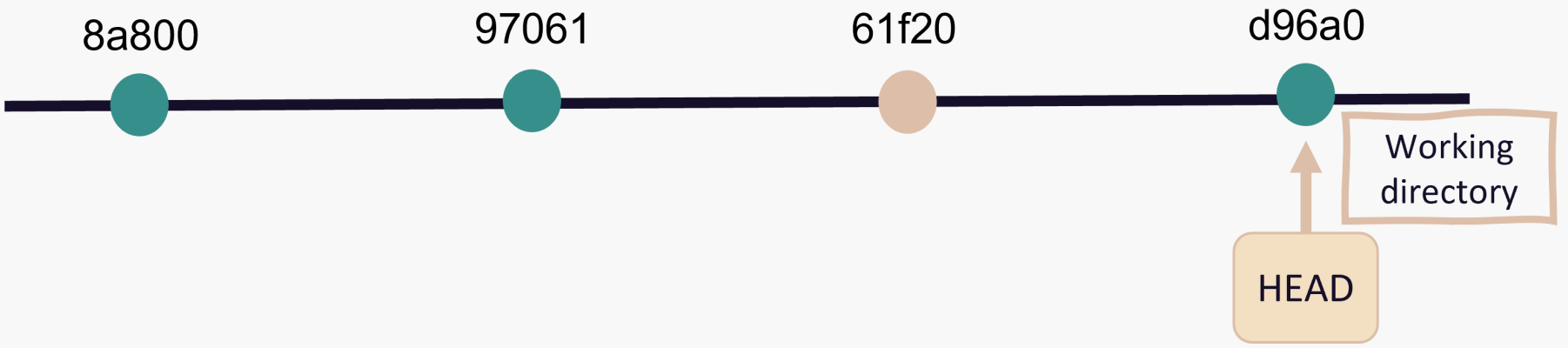
Go back in time with Git

> git log, git checkout, git revert

Checkout a previous commit

- Bring your work space back in time temporarily with `git checkout`





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

Now you

> Revert a part of your recipe

Other good things to know

> Ignore files, publish your projects

Ignore files with `.gitignore`

- Use a `.gitignore` file to list files/folders that you don't want to track with git

- 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.

Publish your repositories

Github/Gitlab are a good way to publish and share your work.

Advantages of publishing your code

- Others can build on your work
- Citations
- Reproducibility
- Get feedback

Publish your repositories

You can increase the quality/complexity of your repo by

- Adding a README.md file
- Connecting the repo with Zenodo to get a DOI
- Create a Github pages website alongside your repo
- Encourage people to write issues if they find problems
- ...

Thanks for your attention

> Questions?

Basic Git workflow