Introduction to version control with Git

Day 2: Branching, Merging and collaboration workflows

Selina Baldauf

November 25, 2024

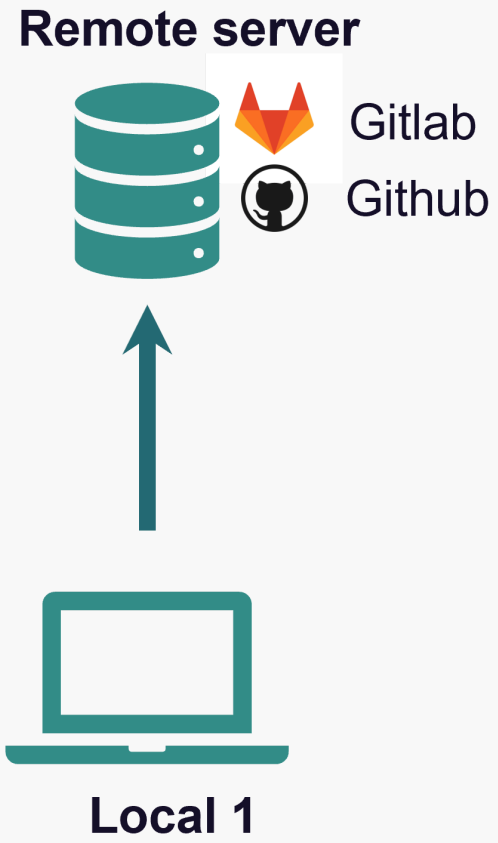
Before we start

Let's check if we are all set with the teams.

Recap

Basic Git workflow:

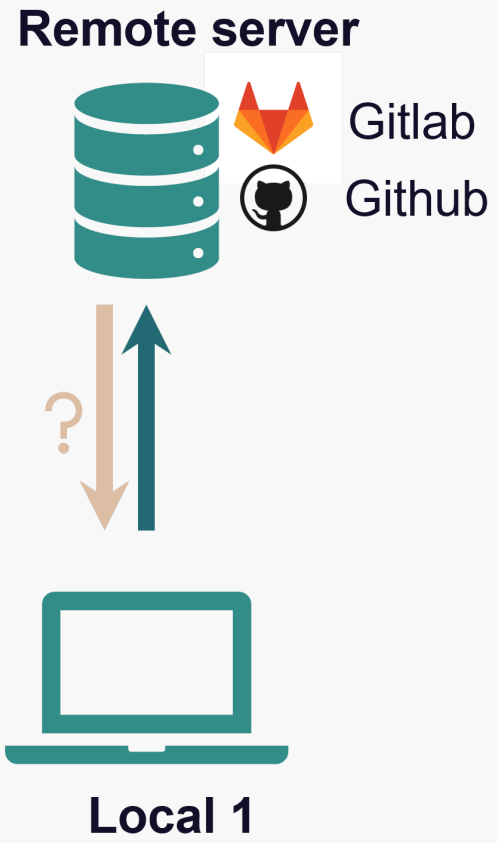
1. **Initialize** a Git repository
2. **Work** on the project
3. **Stage** and **commit** changes to the local repository
4. **Push** to the remote repository



Recap

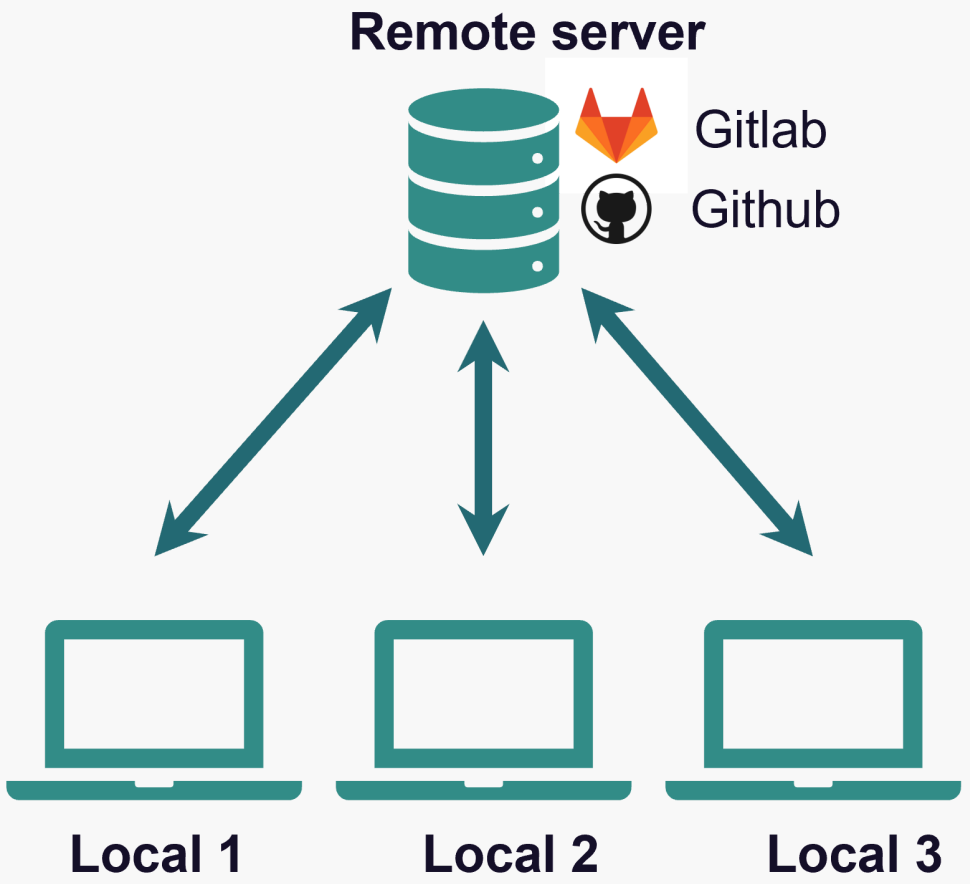
Basic Git workflow:

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**

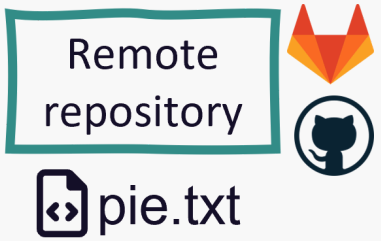


- Idea: many *local* repositories synced via one *remote* repo
- Collaborate with
  - **yourself** on different machines
  - your **colleagues** and friends
  - **strangers** on open source projects

Get a repo from a remote

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

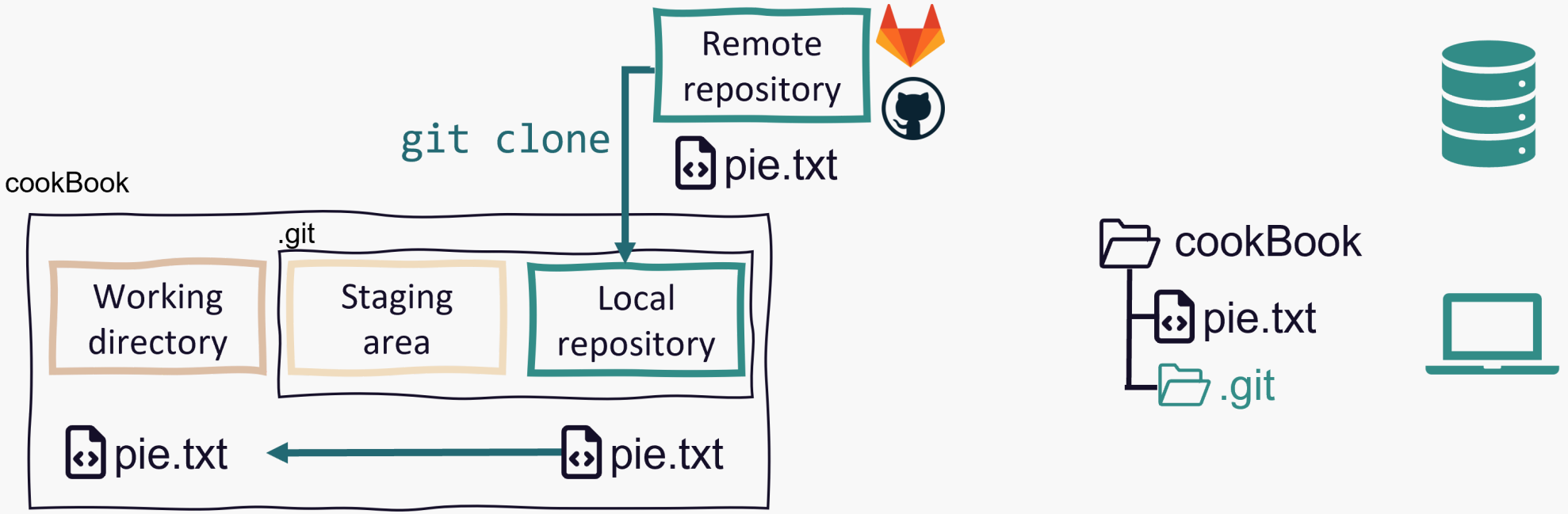
- Get a **full copy** of the remote repo



Get a repo from a remote

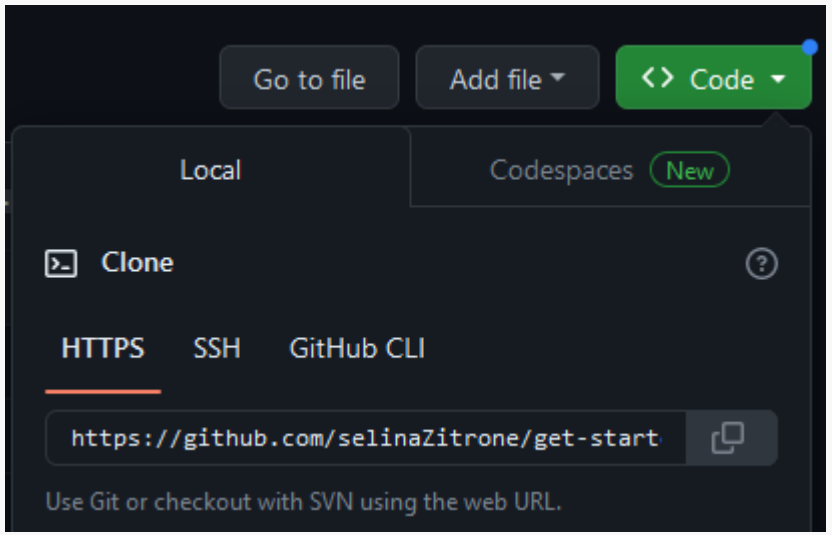
In Git language, this is called **cloning**

- Get a **full copy** of the remote repo



Get a repo from a remote

To clone a repo, you need to know the repo's URL



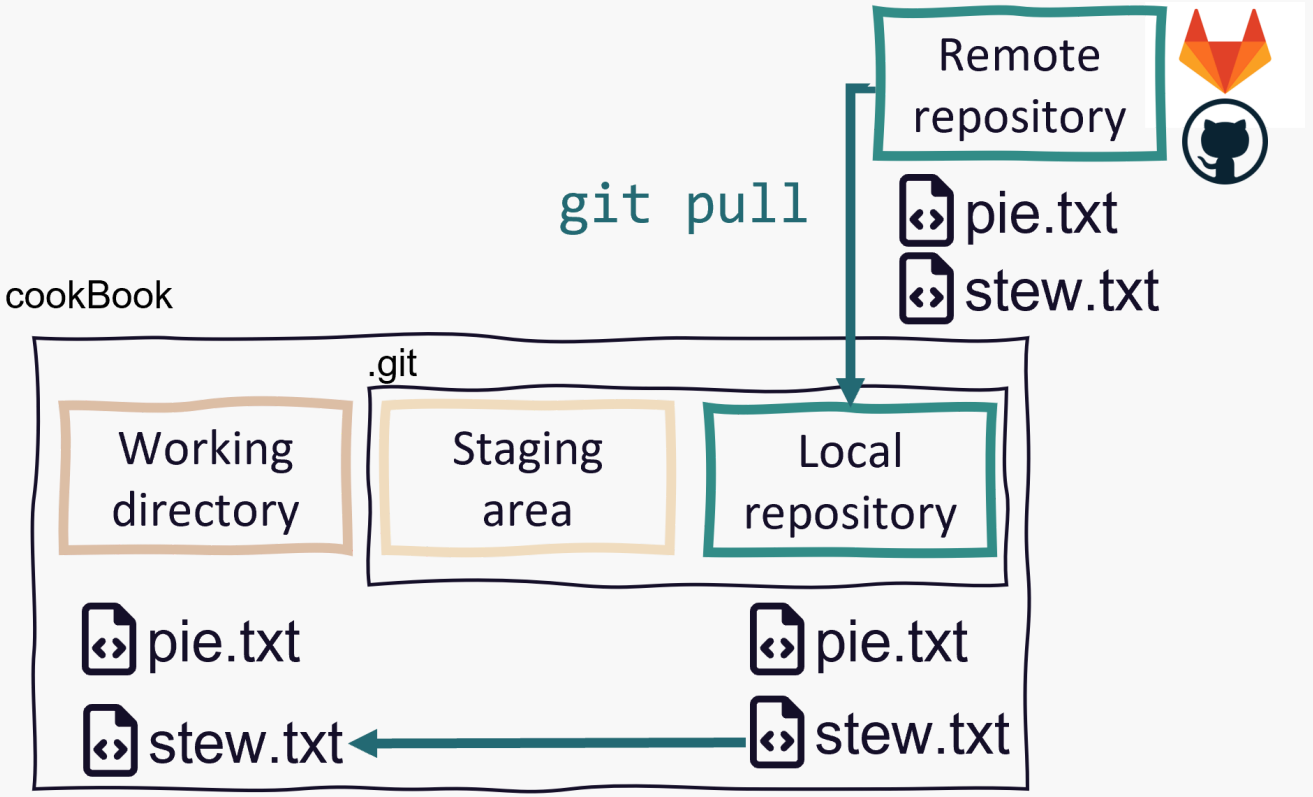
- You can clone all public repositories
  - You can only push if you are authorized
- You can clone private repositories if you are a owner/collaborator

Now you (5 min)

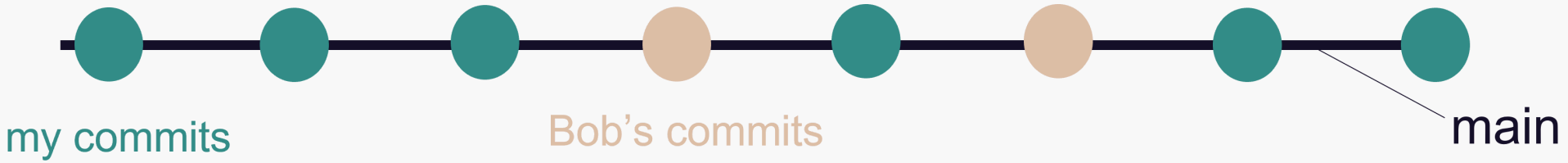
> Clone your team mate's cook Book repo
> Details in Task 2 "Clone"

Get changes from the remote

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

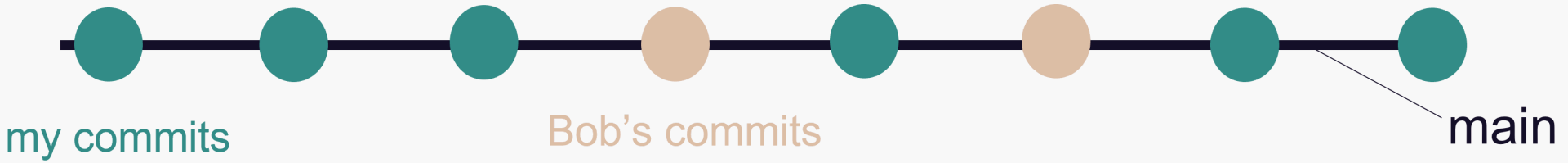


A simple collaboration workflow



- 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**

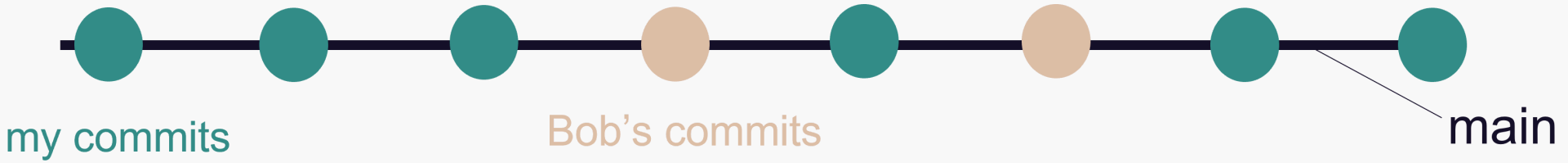
A simple collaboration workflow



This works well if

- Repo is not updated often
- You don't work on the same files simultaneously
- No need to discuss changes before they are integrated
- You collaborate with yourself

A simple collaboration workflow



This workflow starts to be problematic when

- People push often/forget to pull regularly
  - Potential conflicts on main
- You just want to experiment
  - Everything goes directly to main

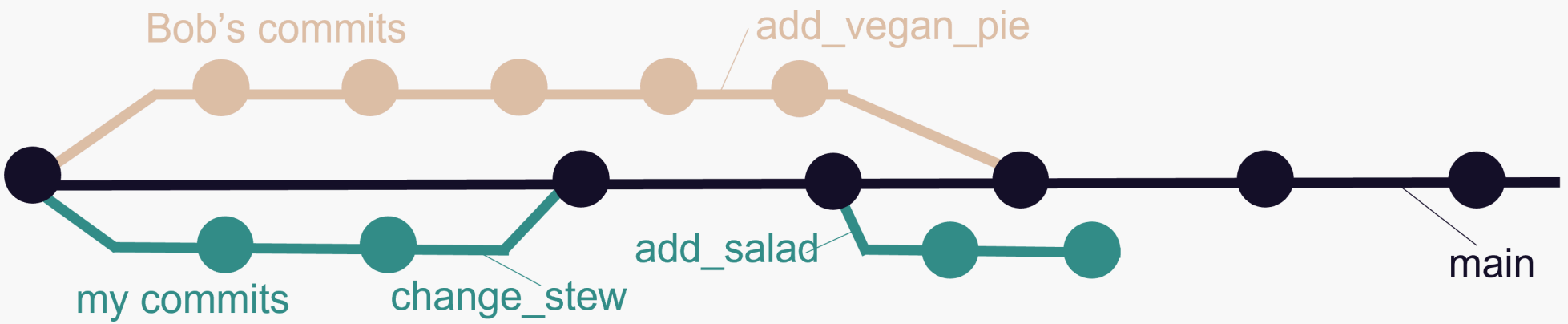
Let's give it a try

- Make sure you are in the repository of your team mate
- Open a recipe in the cook book of your team mate
  - **Repository -> Show in Explorer**
- Change something in there
- Commit the change and push it

Get the changes of your team mate from the remote.

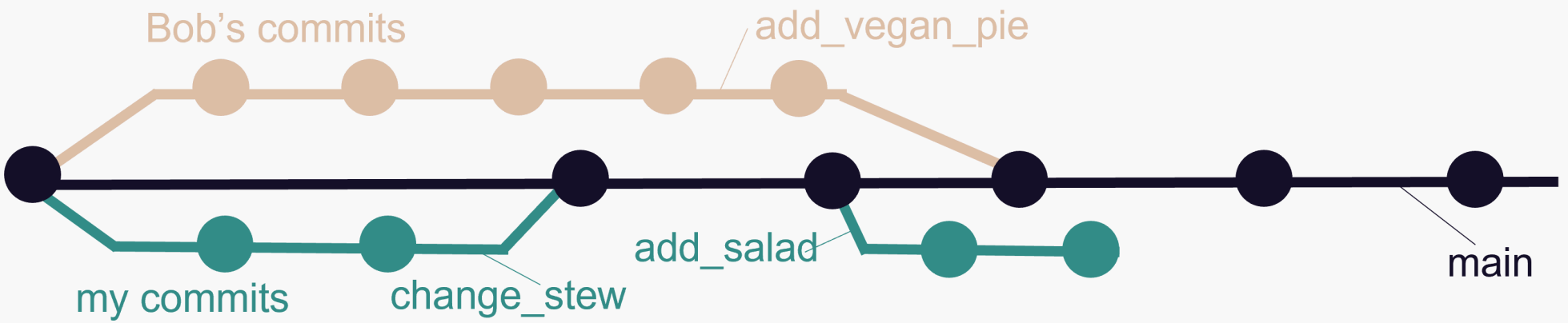
- Switch to your own cook book repository
- Pull the changes (Same button as the push button)
- Have a look at the commit history to see what changed

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
- **Pull before** and **push after** working

A branching-merging workflow



Advantages of this approach

- Guarantee that main always works
- Potential conflicts don't have to be solved on main
- You can experiment without messing up the main

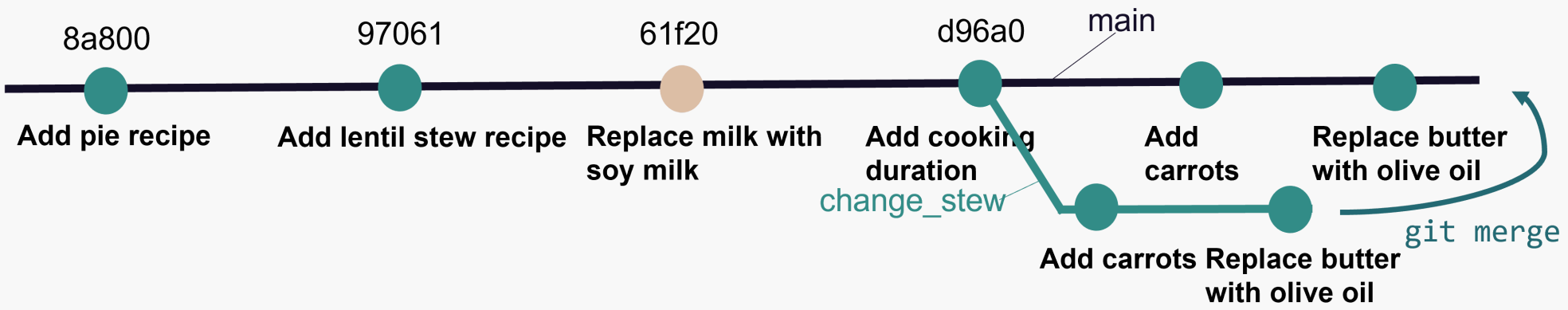
Working on a separate branch

The steps to create and work on a separate branch are easy:

- Create a local branch and switch to it
- Work on the branch like you are used to
  - Make changes, **stage** and **commit**

Merging changes from a branch

To bring changes to the main branch you need to **merge** them.

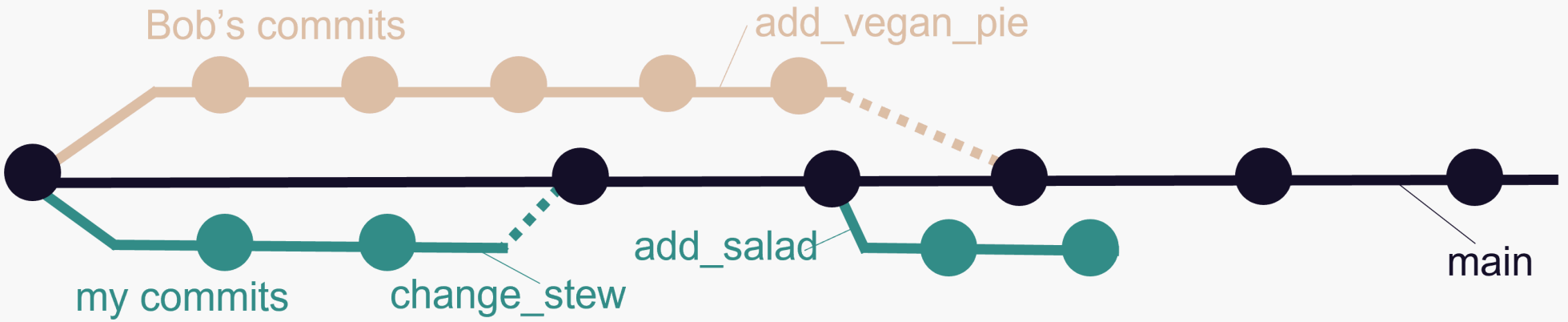


Git merge brings the commits from the branch to main

Now you (10 min)

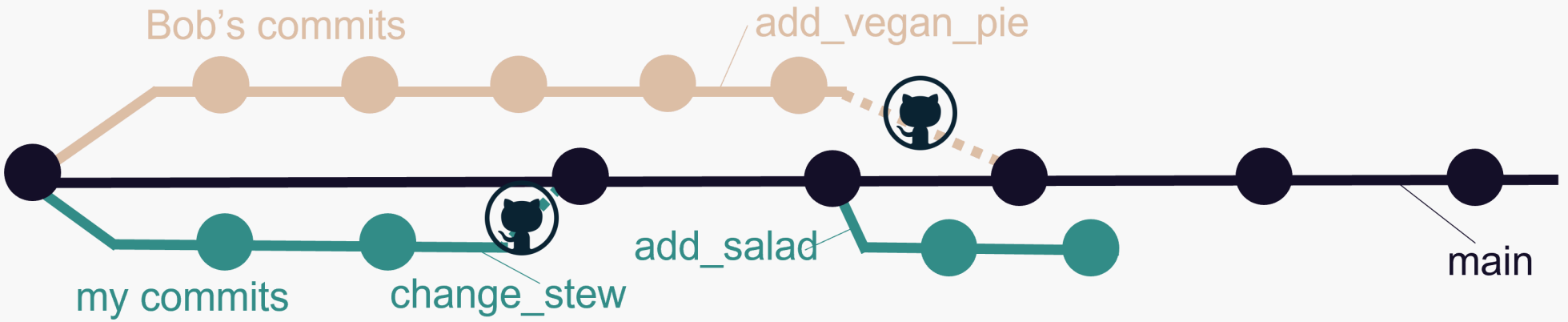
> Create a branch and merge it in your team mate's cook book
> Complete task 2 "Branch and merge"

A branching-merging workflow with GitHub



- 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~~

A branching-merging workflow with GitHub



- 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~~
  - **Create a pull request** on GitHub to ask for a merge
- **Pull before** and **push after** working

A branching-merging workflow with GitHub

A pull request is basically asking your collaborators:

> What do you think of my changes? Can we integrate them in main or do we still need to change something?

GitHub has nice features for pull requests, e.g.:

- **Provide context and explanations** for your changes
- Collaborators can easily **compare versions**
- Collaborators can **discuss and comment** on your changes

A pull request is merged on GitHub when **everyone agreed on the code**.

Now you (10 min)

> Create a pull request on your team mate'S repo
> Complete task 3 "Pull requests"

Thanks for your attention

> Questions?

Next week

- Monday 2 - 3 on Webex (link via email)
- Until then: work with Git on your own if you can
  - Pick something you find most interesting/useful to you
- Collect questions/problems/discoveries
- More Git topics

Some ideas

- **Start working** with Git on one of **your research projects**
- **Publish** one of your projects on GitHub including a nice README
- **Practice collaboration** by contributing to your team mate's cook using pull requests
  - Your team mate can answer your pull requests and request some changes :)

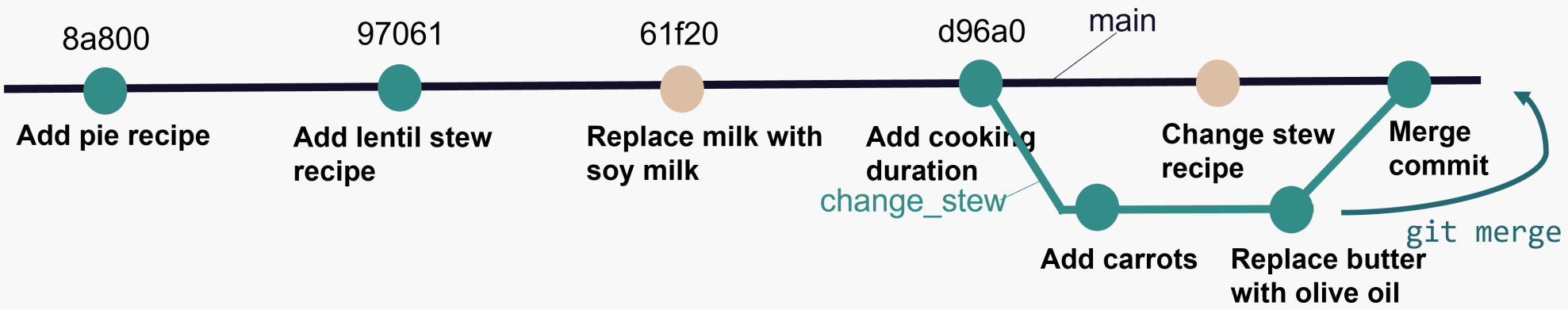
Some ideas

- **Check out the How-To guides** if you want to
  - Recap GH Desktop
  - learn about Git in the terminal
  - learn about Git + R
- If you find a mistake on my websites
  - **Edit the page** on GitHub (and make a Pull request) or **report an issue**
- Checkout the additional resources

Appendix

Merging changes from a branch

To bring changes to the main branch you need to **merge** them.



If there was a commit on a common file in main, a *merge commit* is introduced.

Merging changes from a branch

To bring changes to the main branch you need to **merge** them.

- Mostly merging happens without problems, but…
- … if the same line was edited on separate branches…
- … there will be a merge conflict 😱

Merge conflicts need to be solved manually. You need to chose which of the conflicting versions you want to keep.