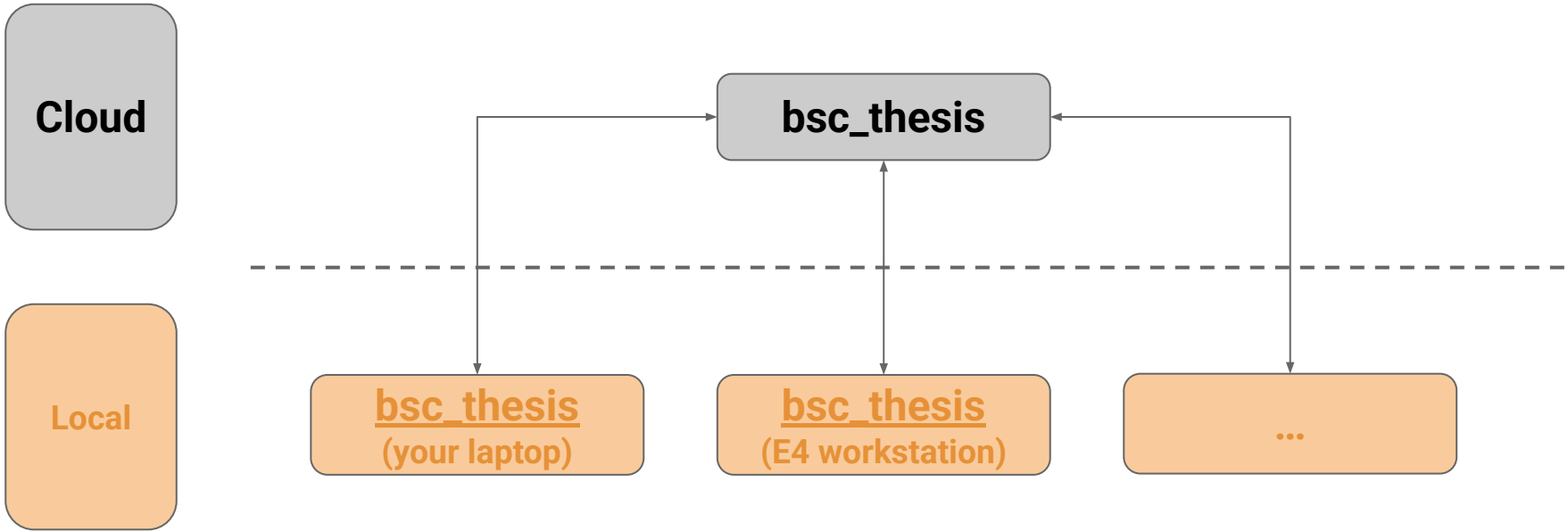
GIT – Bachelor programming course

(git gud)

Why git?

- standard across science and industry for:
  - version control
  - work in collaborations
  - **backup/save scripts, …**
  - **distributed access from different machines**

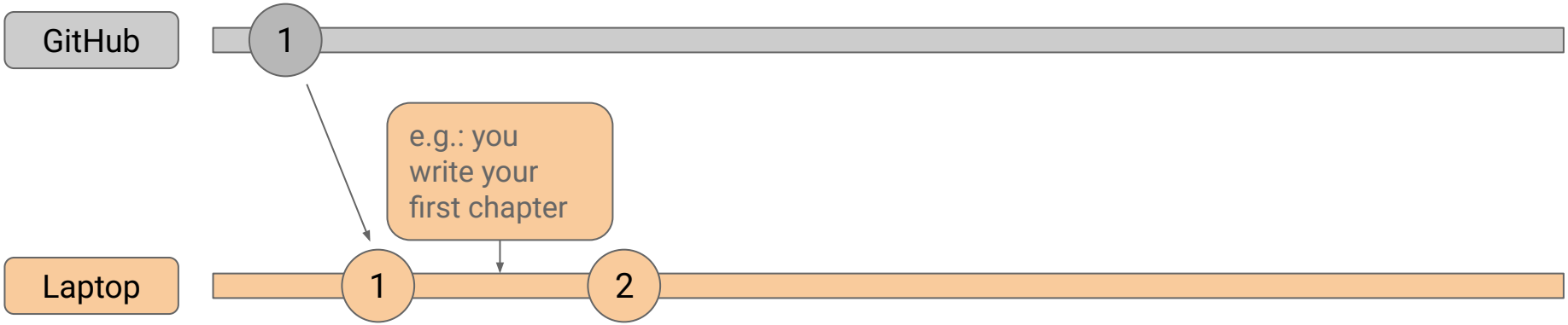
Structure


Cloud
Local
bsc_thesis
bsc_thesis
(your laptop)
bsc_thesis
(E4 workstation)
...

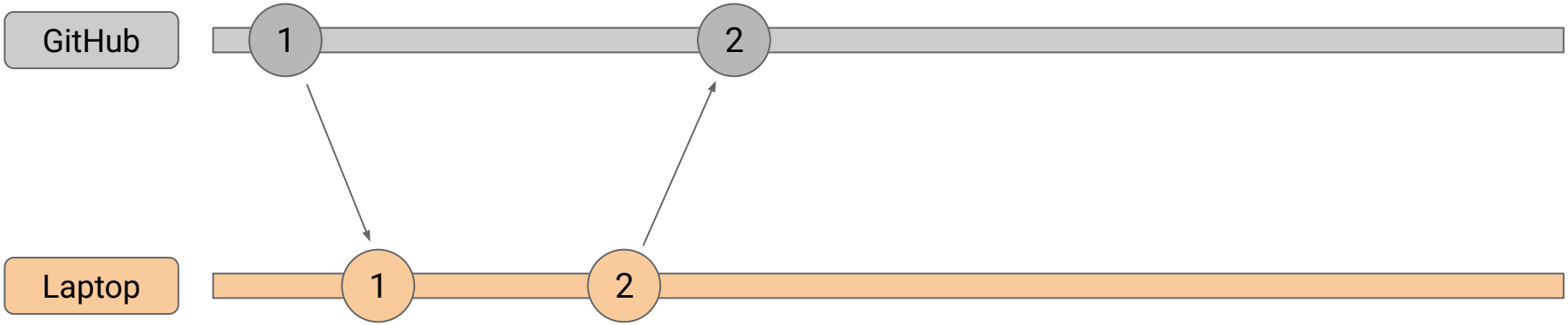
Exemplary Workflow

Exemplary Workflow

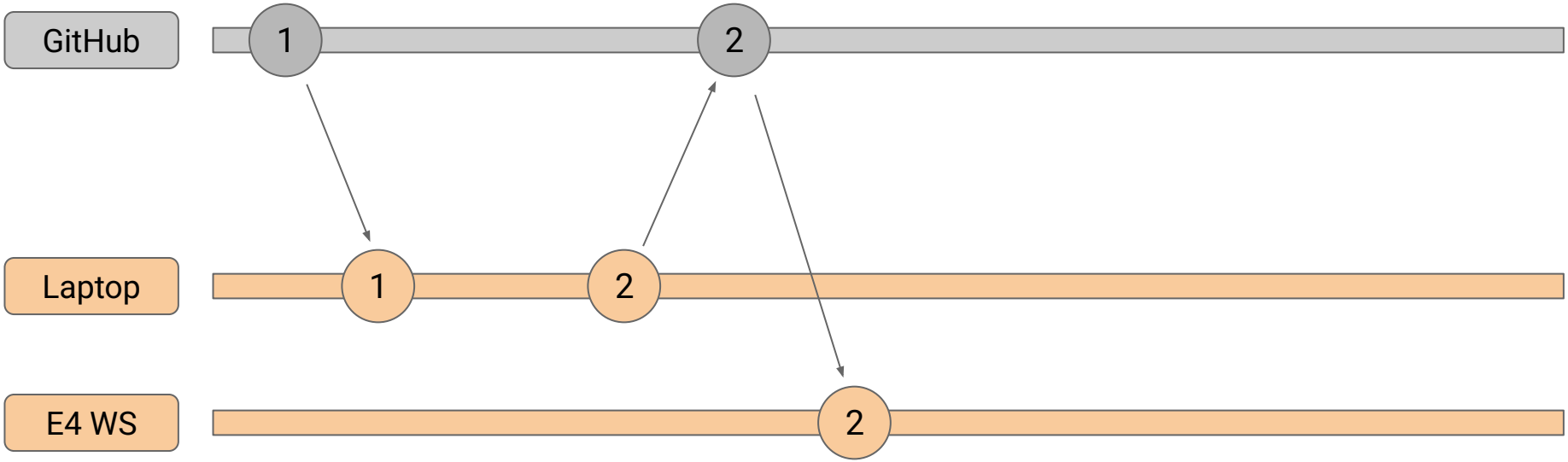
Exemplary Workflow



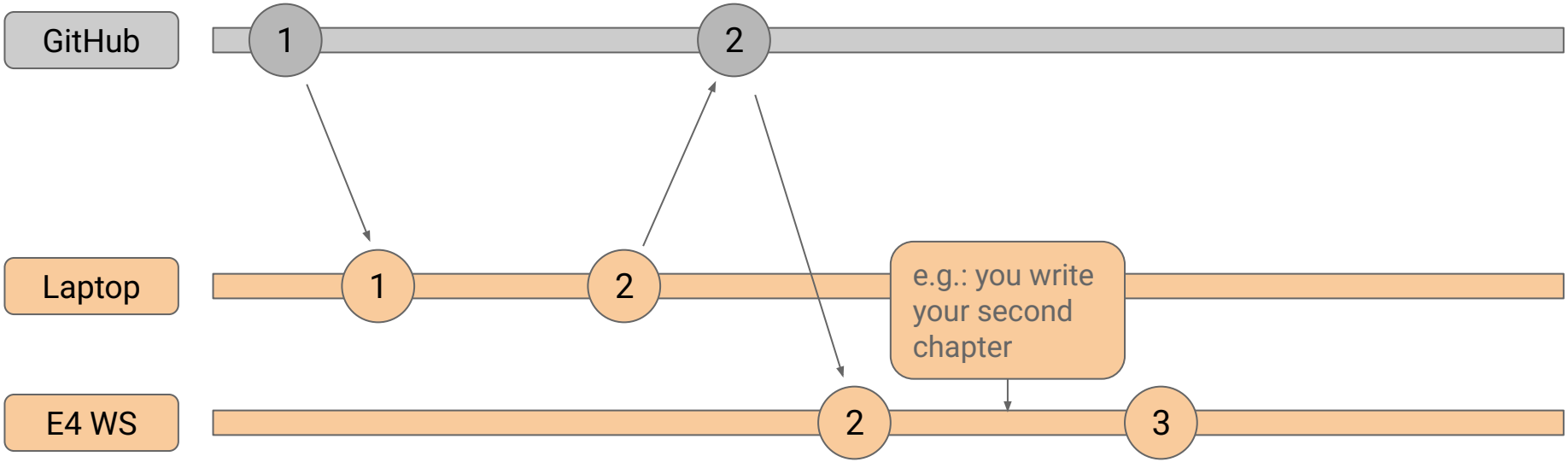
Exemplary Workflow



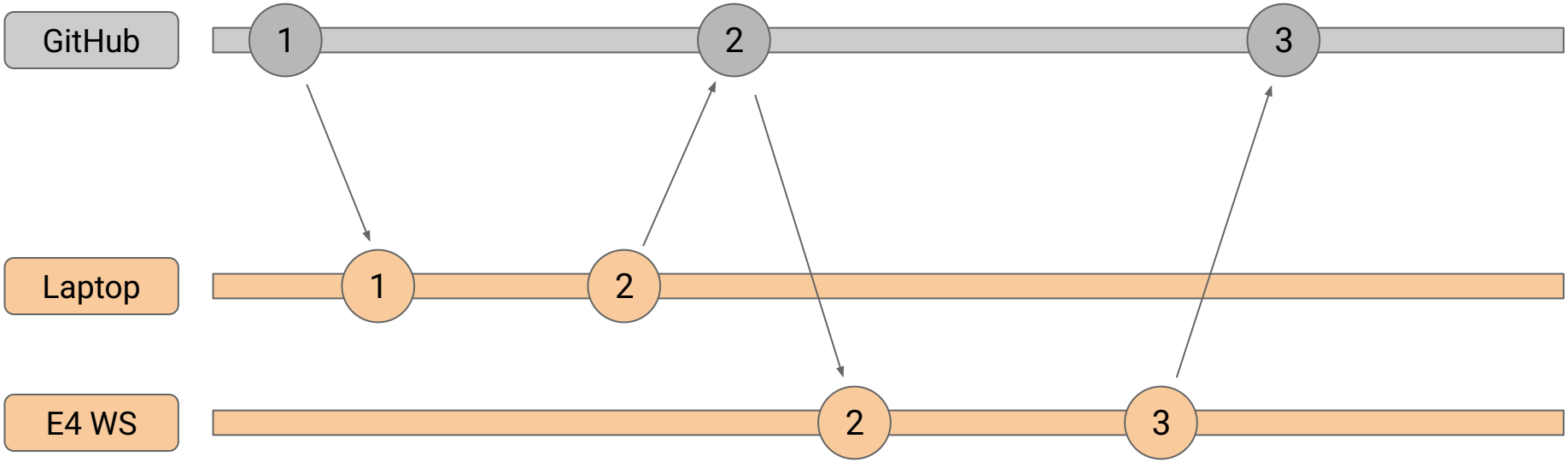
Exemplary Workflow



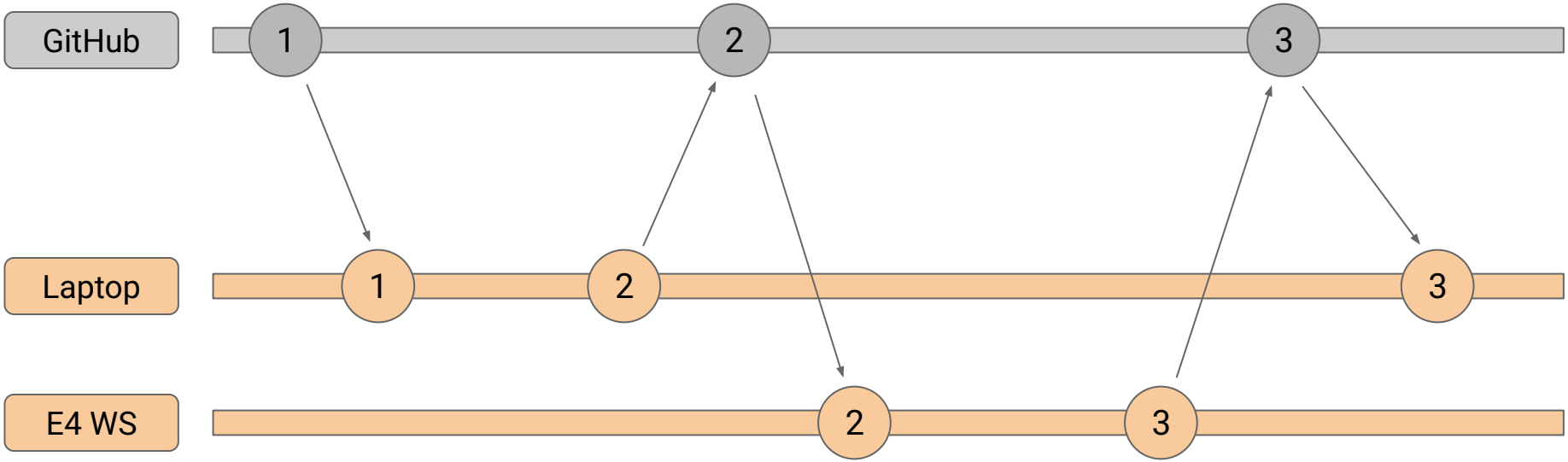
Exemplary Workflow



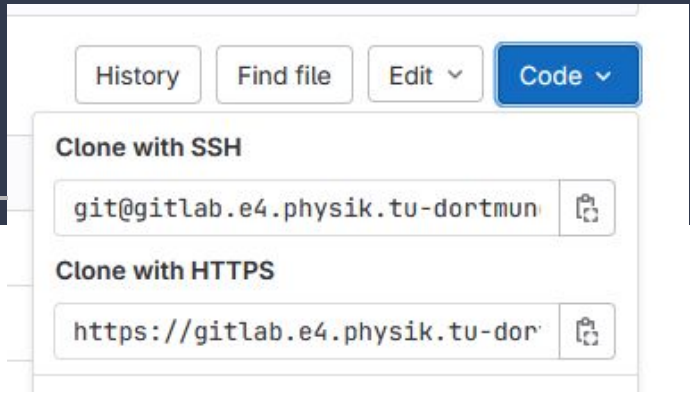
Exemplary Workflow



Exemplary Workflow



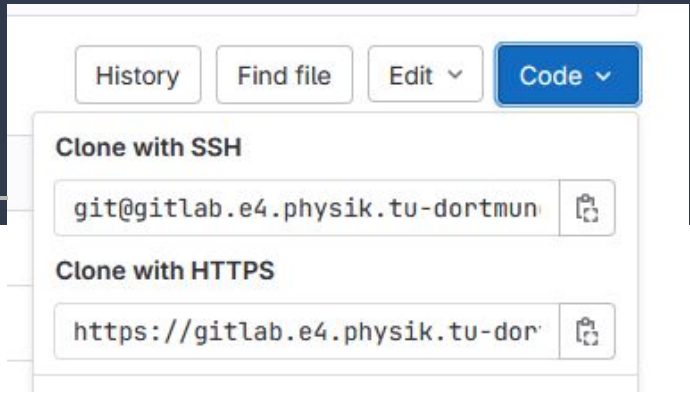
Workflow in practice



First:

1. Create a repository (e.g. on github or our gitlab)
2. Clone repository on your local machine (`git clone ……`)
3. Work in local repo (behaves like a traditional folder)

Workflow in practice



First:

1. Create a repository (e.g. on github or our gitlab)
2. Clone repository on your local machine (`git clone ……`)
3. Work in local repo (behaves like a traditional folder)

Then regularly: (e.g. at the end of each day/week)

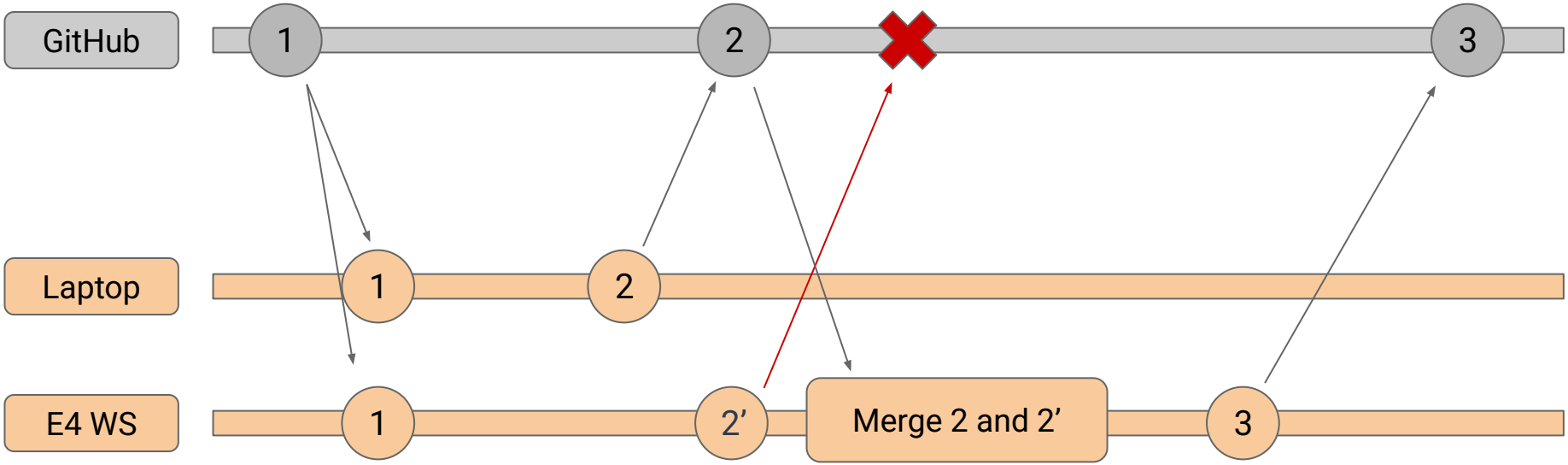
1. Track your changes: `git add --all/<yourfile>`
2. Create a commit with description what you changed since your last commit: `git commit -m "Finished chapter 2"`
3. Push the changes to the cloud repo: `git push`

You have to be in a shell inside the respective folder

Working on multiple devices

- you have to clone the repo on each new machine
- get the last changes from the Cloud Repo: `git pull`
- after you are finished push your changes to the cloud repo again
- if you forgot to push your changes on a machine you might run into a ***Merge Conflict***

Merge Conflicts

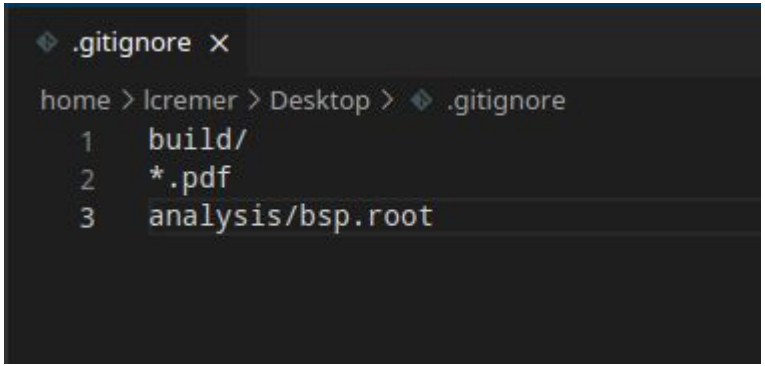


Merge Conflicts

- sometimes merge conflicts can be solved automatically by git (e.g. if the conflicting changes are in different files)
- if not, git will ask you to solve the merge conflict by yourself → go to all files with conflicting lines (marked by git) and edit them as you want to keep them
- if you are using VSCode + git extension, there is a nice overlay to check what you want to keep
- if in doubt ask your supervisor or me for help (ChatGPT is also pretty useful)

gitignore

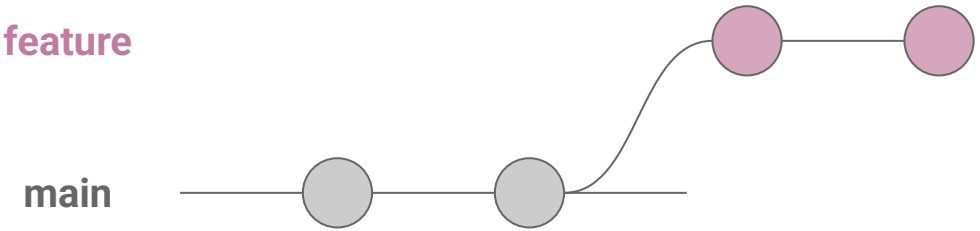
- usually there are a lot of files which you don't want to track with git
  - data files
  - output files (e.g. plots)
  - cache files
- you can create/edit `.gitignore` (in the main folder of your repo)
- provide a list of files/folders that should be ignored (wildcards are supported)



“Advanced”:
Working in a team

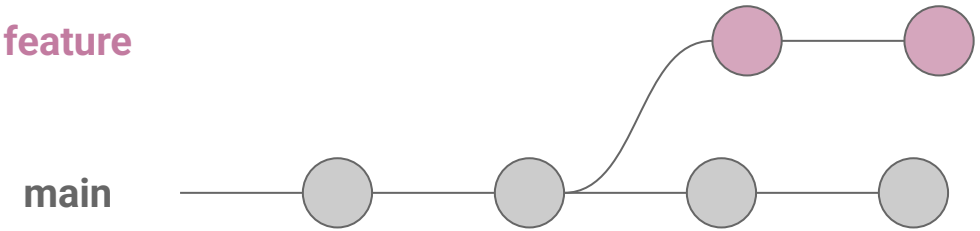
Branches & Rebasing

- especially when working in a team, you don't want to develop on ***main*** → ***main*** should be the clean & stable branch including only *finished* changes the group/maintainers agreed on
- so if you e.g. want to develop a new feature for the main repository you create a "feature branch", which you use for the development of the feature
- after your feature is ready you request to merge the "feature branch" back into ***main***



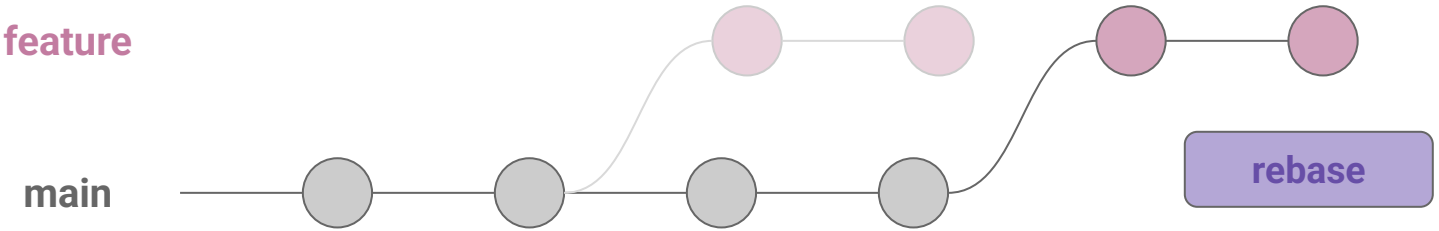
Branches & Rebasing

- especially when working in a team, you don't want to develop on ***main*** → ***main*** should be the clean & stable branch including only *finished* changes the group/maintainers agreed on
- so if you e.g. want to develop a new feature for the main repository you create a "feature branch", which you use for the development of the feature
- after your feature is ready you request to merge the "feature branch" back into ***main*** → *rebase/merge* (usually you want to rebase for a cleaner history)



Branches & Rebasing

- especially when working in a team, you don't want to develop on ***main*** → ***main*** should be the clean & stable branch including only *finished* changes the group/maintainers agreed on
- so if you e.g. want to develop a new feature for the main repository you create a "feature branch", which you use for the development of the feature
- after your feature is ready you request to merge the "feature branch" back into ***main*** → *rebase/merge* (usually you want to rebase for a cleaner history)



In practice

- create & switch to a new branch: `git checkout -b my-new-branch`
- switch between branches: `git checkout <branch>`

In practice

- create & switch to a new branch: `git checkout -b my-new-branch`
- switch between branches: `git checkout <branch>`
- rebasing: pull latest changes and go to your feature branch and then use `git rebase main`
    - if the rebase runs into a conflict, you will have to edit the respective files and mark them ready with `git add <file>`
    - after all conflicts are fixed: `git rebase --continue`
    - if you want to abort the rebase: `git rebase --abort`

In practice

- create & switch to a new branch: `git checkout -b my-new-branch`
- switch between branches: `git checkout <branch>`
- rebasing: pull latest changes and go to your feature branch and then use `git rebase main`
    - if the rebase runs into a conflict, you will have to edit the respective files and mark them ready with `git add <file>`
    - after all conflicts are fixed: `git rebase --continue`
    - if you want to abort the rebase: `git rebase --abort`
- if rebase is finished you can push the changes via `git push --force` (since you changed the history of your branch, you need --force)
- now your branch is ready to be merge into the main branch in the cloud repo (→ Merge request)