
>. TechLabs
Winner of the
Google.org Impact Challenge
Germany 2018
git
Or
"How to work
together without
killing each other!"
Git Workshop

# Agenda

>. TechLabs

- Introduction
- Introduction to Git
- Collaborative working with Git
- Collaborative working done right
- How to fix things with Git
- Wrap-Up

# Introduction
Goal of today

>. TechLabs

- We learn how **to use Git**, and not how Git works
  - We are teaching you to drive, not how the engine works
  - For your own projects
  - For working with each other in a project (Hello project Phase!)
  - Opinionated workflow based on our experience
- We will focus on command line
  - If you understand how it works on the command line you can easily switch to GUI

# Introduction
Workshop Mode

>. TechLabs

Git Introduction

Git Collaboration

Git Collaboration 2

Git Fix


>. TechLabs

To save time during the workshop, we asked you to prepare beforehand:

- Git installation
- Set up git user (git --config global email/username)
- Have your GitHub username ready and be logged in into GitHub
- Run the test script

**Have all of you completed the preparations for this workshop?**


# Introduction

# Introduction into Git

## What is the problem?

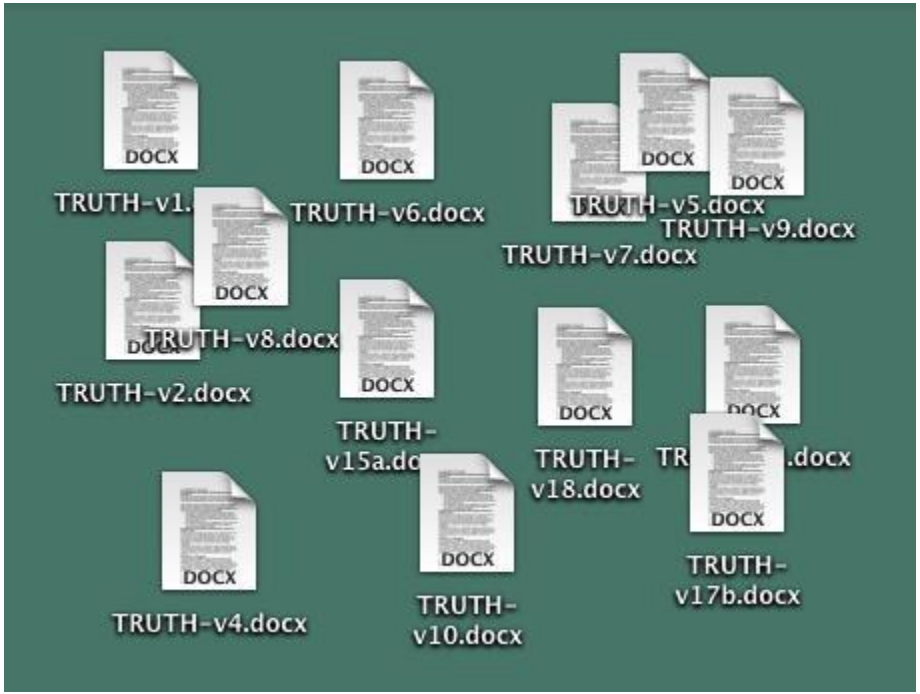
>. TechLabs

- You work on a project and want to make some changes to your code/text file
- You make some changes, go to bed and realize the next day that the changes you made aren't as great as you thought
- You would like to revert back to the state before you changed the code yesterday

# Introduction into Git

## Why do I need Version Control

>. TechLabs



>. TechLabs

- Git is a tool to **create save points** and helps you **handle these save points**.
- You can **modify** your files/code and be sure **will not ruin your work** because you created save points along the way.
- Protects yourself from changes of others.
- Projects changes of others from yourself. (No angry teammates anymore!)

# Introduction into Git

## What is Git?

>. TechLabs

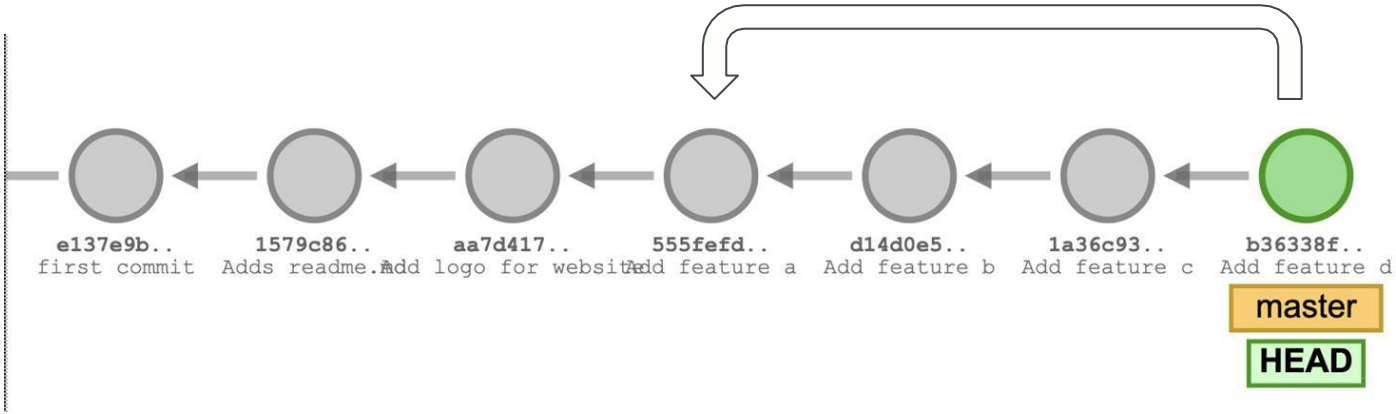
- Git is a distributed version control system
- A system that keeps records of your changes
  - **Allows you to revert any changes and go back to a previous state**
- Allows for collaborative development
- Allows you to know who made what changes, when and why

# Introduction into Git

Version Control done Right!

>. TechLabs

You can jump back anytime, and see the version of your files before



# Introduction into Git

But it takes time to learn GIT!

>. TechLabs

"But this is all so confusing, and scary. I will just use my old manual system"

This might work when you are alone, but:

We do know that the time you invest into learning Git now is less than the time you would spend doing version control across 6 people manually!

# Introduction into GIT

## Terminal

>. TechLabs

- Don't fear the terminal, it is your friend :)
- Most important commands for today

  -> pwd - prints the current path

  -> cd <path> - changes your current path to <path>

  -> touch <file> - creates a new file named <file> in your current path

  * If you do not know nano or vim, just use your favourite text editor
- All the Git commands will be run in your current path!
- **READ!** Even if a command fails git often tells you what command you need to run!

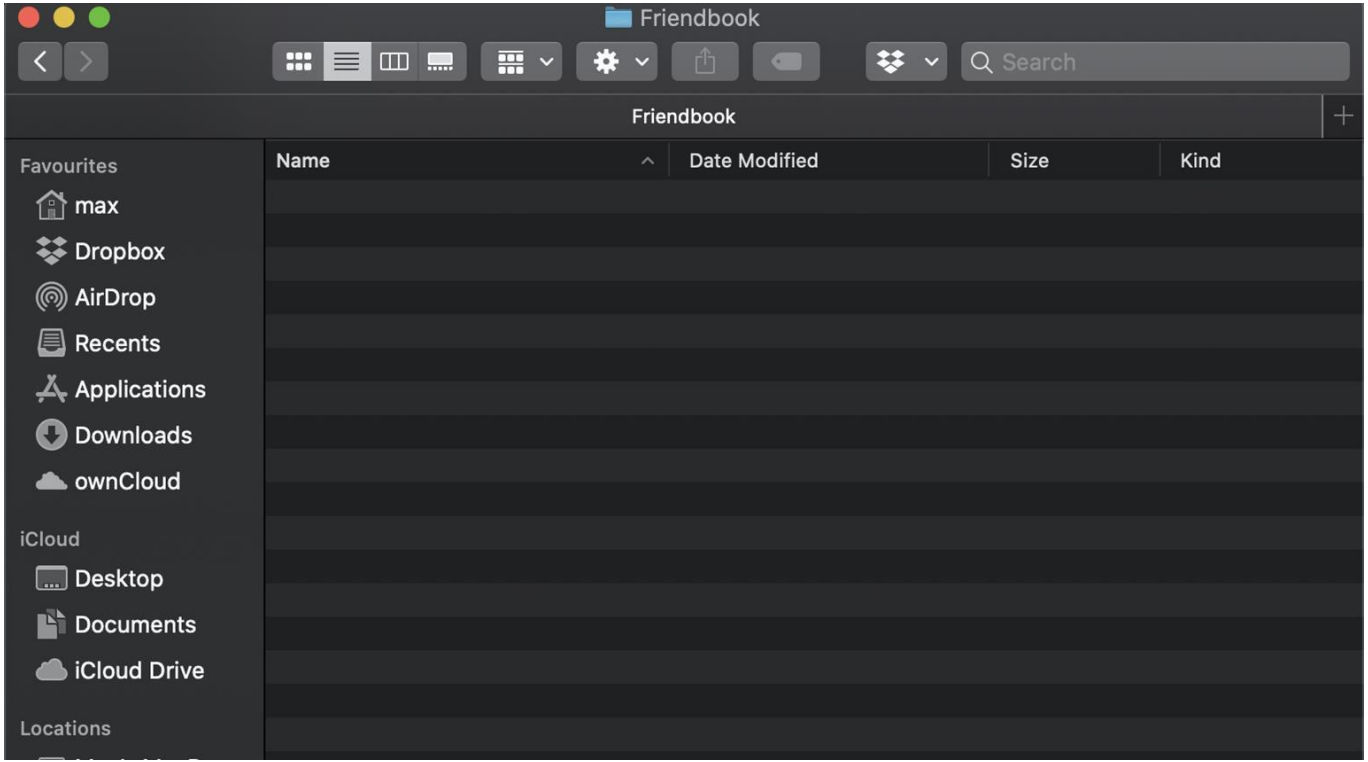
>. TechLabs

- Git Repository (Repo)
  - **A folder** where all changes should be tracked.

    Your project's repository contains all of your project's files and stores each file's revision history.
- Git Commit
  - A specific version(snapshot/save point) of the files in a repository.

# Introduction into Git
## Starting Point for examples

>. TechLabs




```
.
└── FriendBook/
```

```
$pwd
/Users/Max/projects/git_presen
tation
```

```
git init
```

```
.
└── FriendBook/
    ├── .git/
```

## git init

- **Initiates** a repository in your current folder!
  Transforms your current directory into a Git repository. It creates a ".git" folder, which stores all snapshots and other things  git needs to work. This .git folder can be  seen as database!
- Running *git init* in an existing repository does not do anything!

```
$git status
On branch main
nothing to commit, working
tree clean
```

# Introduction into Git
## Git status

>. TechLabs

- Displays the state of repository in your current path
- What files did you change?
- Which files are not taken care of yet by .git etc.
- Status output does not show you any information regarding the committed project history. For this, you need to use git log.

**As a beginner, run this each time before you want to run another command.**

```
$pwd
/Users/Max/projects/FriendBook
$git status

On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

$ ...some changes and adds later.
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:

            new file:   sebi.md
```

# Introduction into Git
## Git Log

>. TechLabs

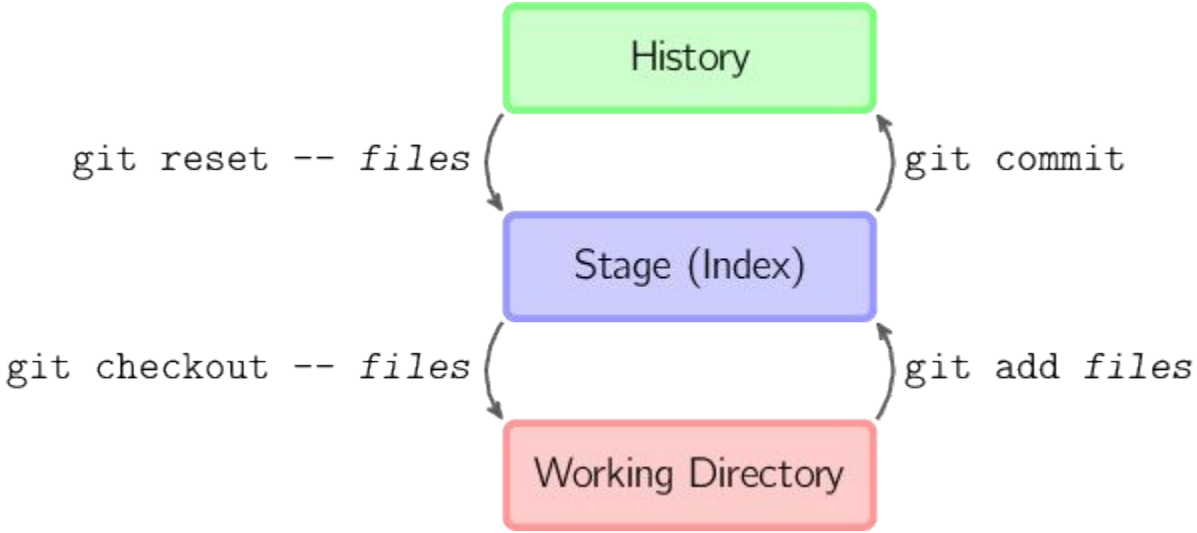
- The git log command displays a record of the commits in a Git repository.
- By default, the git log command displays a commit hash, the commit message, and other commit metadata. You can filter the output of git log using various options.

```
$git log

commit 0bcbc2508a305dbe701255b7427ba38887f7c702 (HEAD
-> main)
Author: Maximilian Schall
<maxscha@mail.de>  Date:     Thu Dec 3
20:55:53 2020 +0100

    Adds sebi.md

commit
3d73a54b1a046e392892b2d5cc97ff4219082355
Author: Maximilian Schall <maxscha@mail.de>
Date:Thu Dec 3 20:03:49 2020 +0100

    first commit
```

**Leave with q**

# Introduction into Git

## Git Staging Module



# Introduction into Git

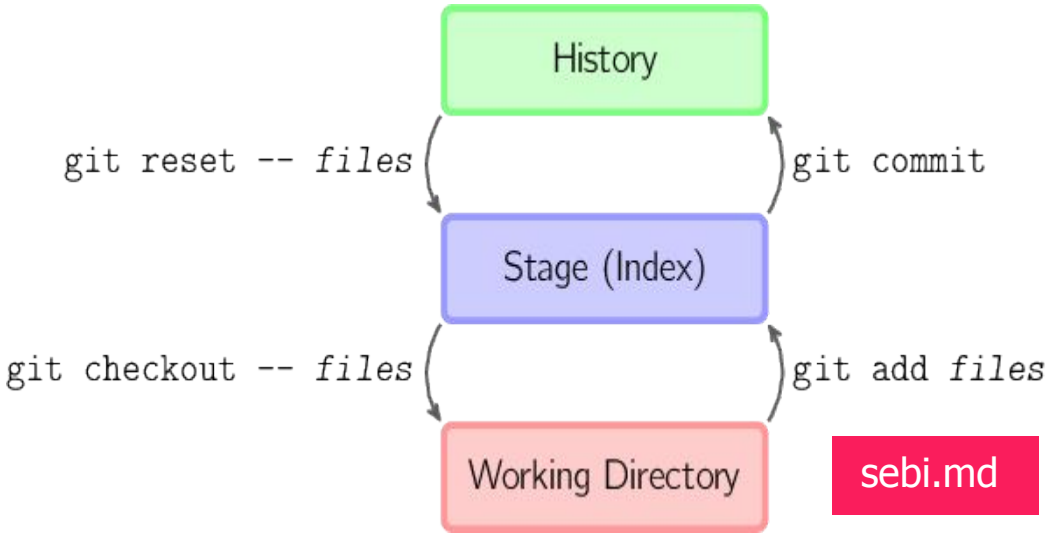
## New File

Not a git command, you can also just create a file in your text editor and save.

>. TechLabs

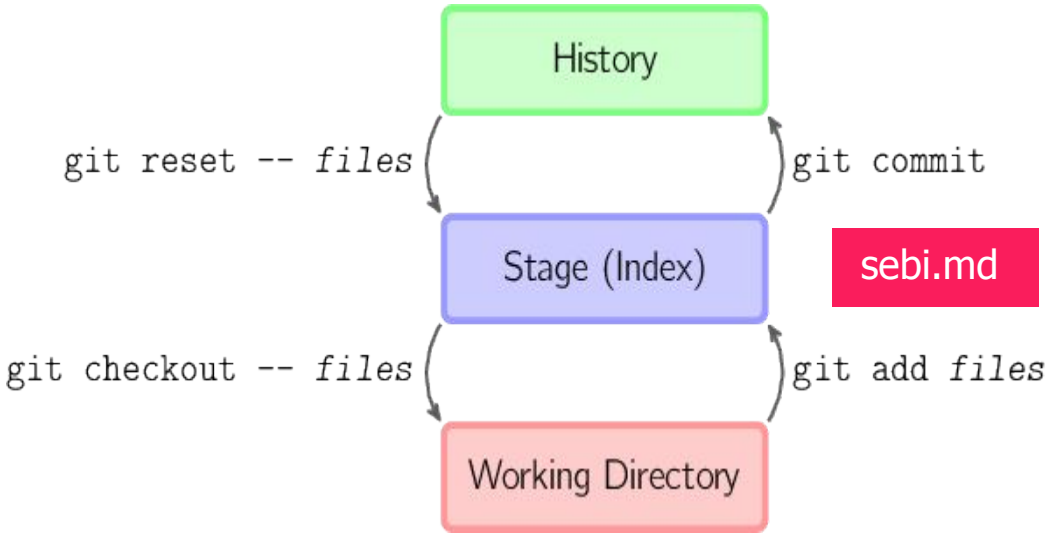
```
touch sebi.md
```

```
.
└── FriendBook
    ├── .git
    └── sebi.md
```




```
$git status
On branch master
Untracked files:
            sebi.md
nothing added to commit
but  untracked files
present
```

# Introduction into Git
## Git add



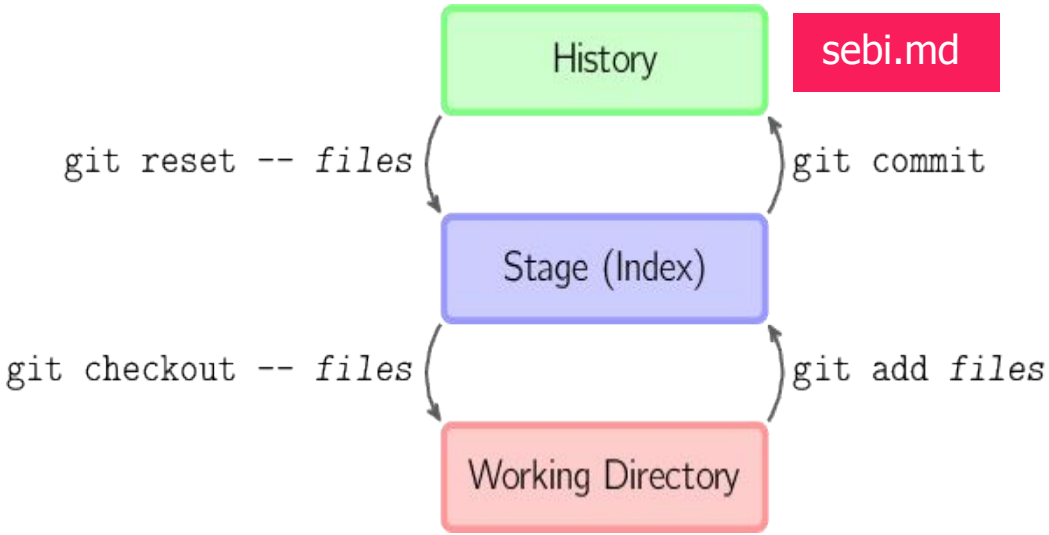

```
git add sebi.md
```

```
.
└── FriendBook
    ├── .git
    └── sebi.md
```

```
$git status
On branch main
Changes to be committed:

            new file:
sebi.md
```

# Introduction into Git

## Git commit




```
git commit
```

```
.
└── FriendBook
    ├── .git
    └── sebi.md
```

```
$git status
On branch main
nothing to commit,
working  tree clean
```

Adds all files in staging to a new commit!

>. TechLabs

- Git requires you to describe every commit you create!
- This is called git commit message.
- Make sure to write some lines what you changed and why?
- In the preparation document we set Visual Studio Code as your default commit message text editor.

>. TechLabs

## What if I want to change a file?

```
edit sebi.md
```

```
git add sebi.md
```

```
git commit
```

# Introduction into Git

But I would like to see my first version of sebi.txt?

>. TechLabs

```
$git log

commit
0bcbc2508a305dbe701255b7427ba38887f7c702
(HEAD -> main)
Author: Maximilian Schall
<maxscha@mail.de> Date:     Thu Dec 3
20:55:53 2020 +0100
    Adds sebi.md

commit          3d73a54b1a046e392892b2d5cc97ff4219082355

Author: Maximilian Schall <maxscha@mail.de>
Date:Thu Dec 3 20:03:49 2020 +0100

    first commit
```

< Name of a commit

```
$git checkout 3d73a54b1a046e392892b2d5cc97ff4219082355

.... Have a look at the first version of sebi.md

$git checkout main
```

# Introduction into Git

What happens if I run a git command outside?

```
$pwd
/Users/Max/projects
$git add -A
fatal: not a git repository
(or any of the
parent
directories): .git
```

Well except for git init

# Introduction into Git

## Tasks

>. TechLabs

Everyone in the team needs to do these tasks, but communicate with each other about your questions and learnings!

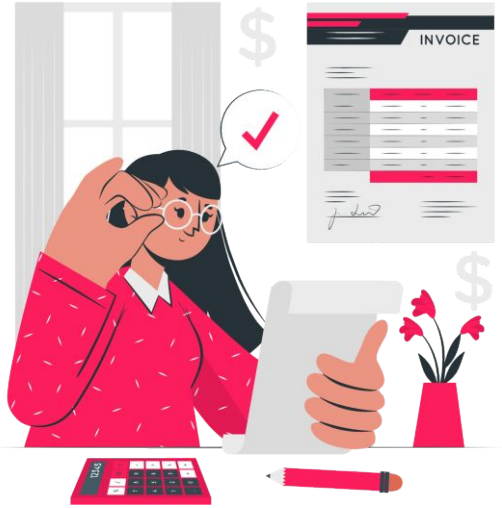
1. Create a new folder and open the command line/terminal in there.
2. Initialize the git repository.
3. Create a file with **yourlastname.txt** in the folder and write 1-2 sentences about you in that file (You can create the file with vs-code, any text editor, etc.) & run git status
4. Add these files to git staging-area & run git status.
5. Commit the staged file & run git status.
6. Check the git logs.
7. Add two more sentences to your file.
8. Add and commit again.
9. Check the logs.
10. Jump back in time, and look at an old commit.
11. Switch back to current version

**15 min**

>. TechLabs

# Questions?

Look at the cheat sheet and ask your group members



# Introduction into Git
## Solution

>. TechLabs

- Create a new folder and open the command line/terminal in there.
- Initialize the git repository.
    - git init
- Create a file with **yourlastname.txt** in the folder and write 1-2 sentences about you in that file (You can create the file with vs-code, any text editor, etc.) & run git status
    - touch yourlastname.txt
    - *edit* yourlastname.txt
    - git status
- Add these files to git staging & run git status.
    - git add yourlastname.txt
    - git status
- Commit the staged file & run git status.
    - git commit -m "yourlastname.txt"
    - git status
- Check the logs.
    - git log
- Add two more sentences to your file.
- Add and commit again.
- Check the logs.
- Jump back in time, and look, add an old commit.
    - git checkout <commit_hash>
- Switch back to main
    - git checkout main (older versions: git checkout master)

>. TechLabs

# How was it?


# Git Collaboration

# Git Collaboration

What will you learn?

>. TechLabs

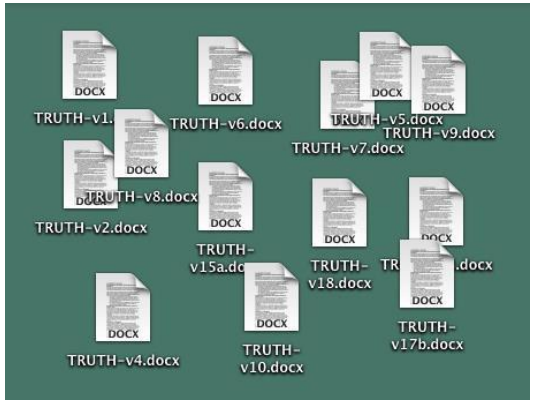
- Well know you understood how do to (basic) versioning in your local repository!
- Next we will learn how to **work together on one project**
- Why is Git great for working together?
  - Everyone can work on it at the “same time”
  - Awesome conflict resolution (e.g. P1 and P2 works on F1 at the same time)

# Git Collaboration

How could you do it?

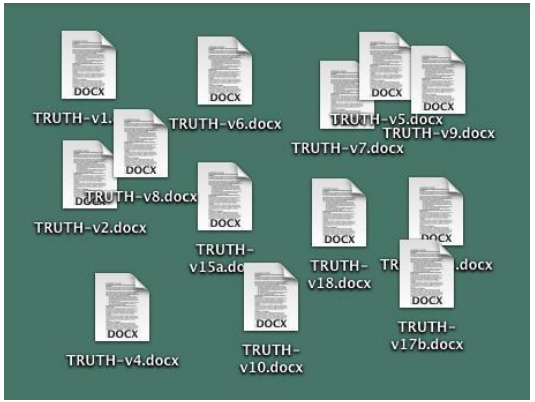
>. TechLabs

“ Can you send me your changes so I can add them to my local version? “



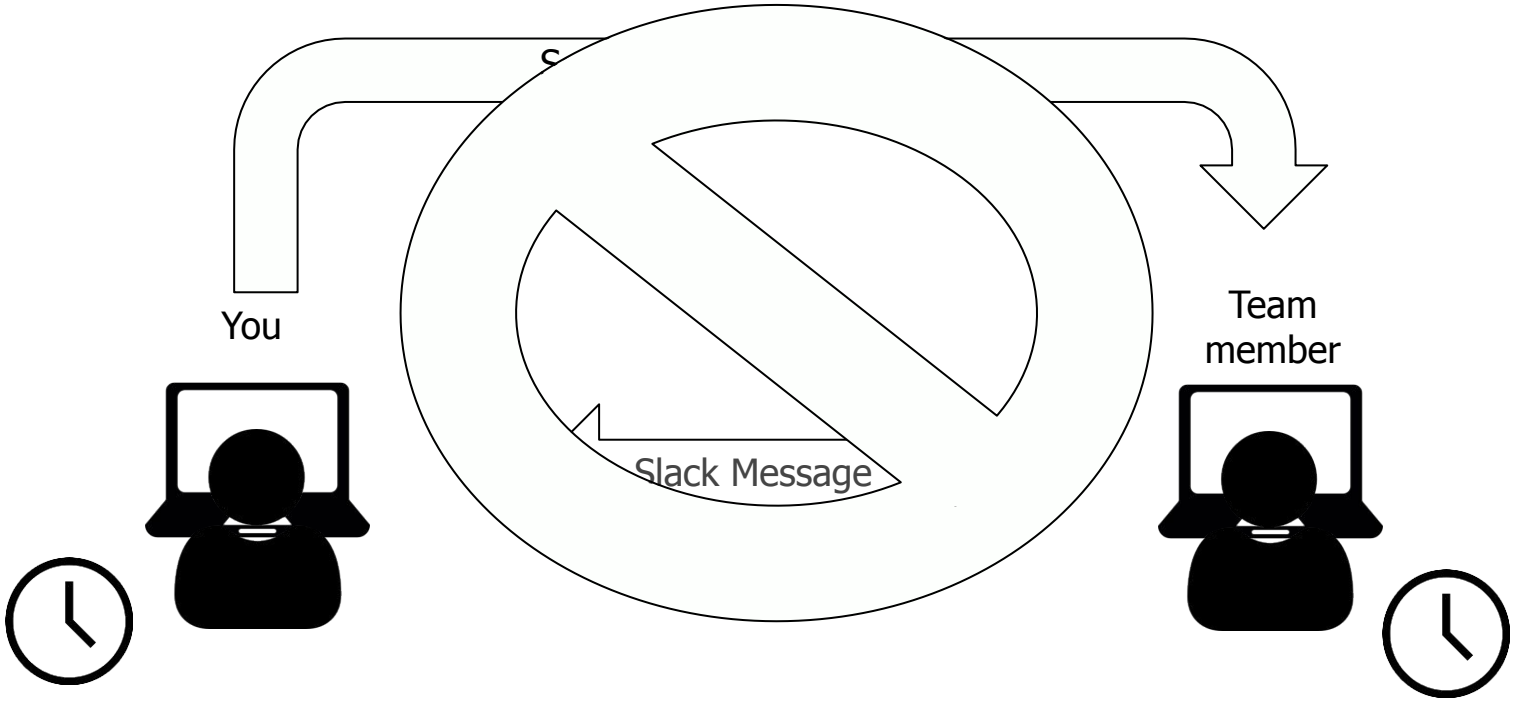

Slack Message

“Sure”



# Git Collaboration

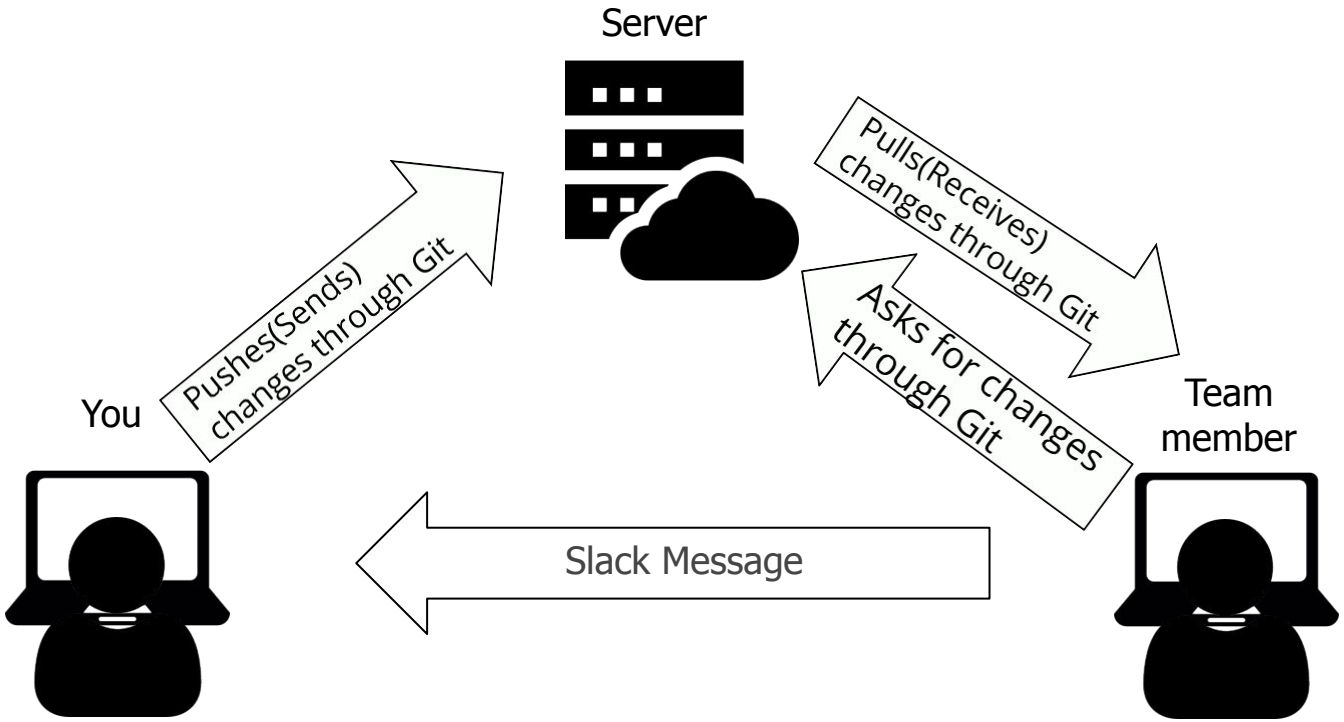
How to properly do it?



# Git Collaboration

How to properly do it?

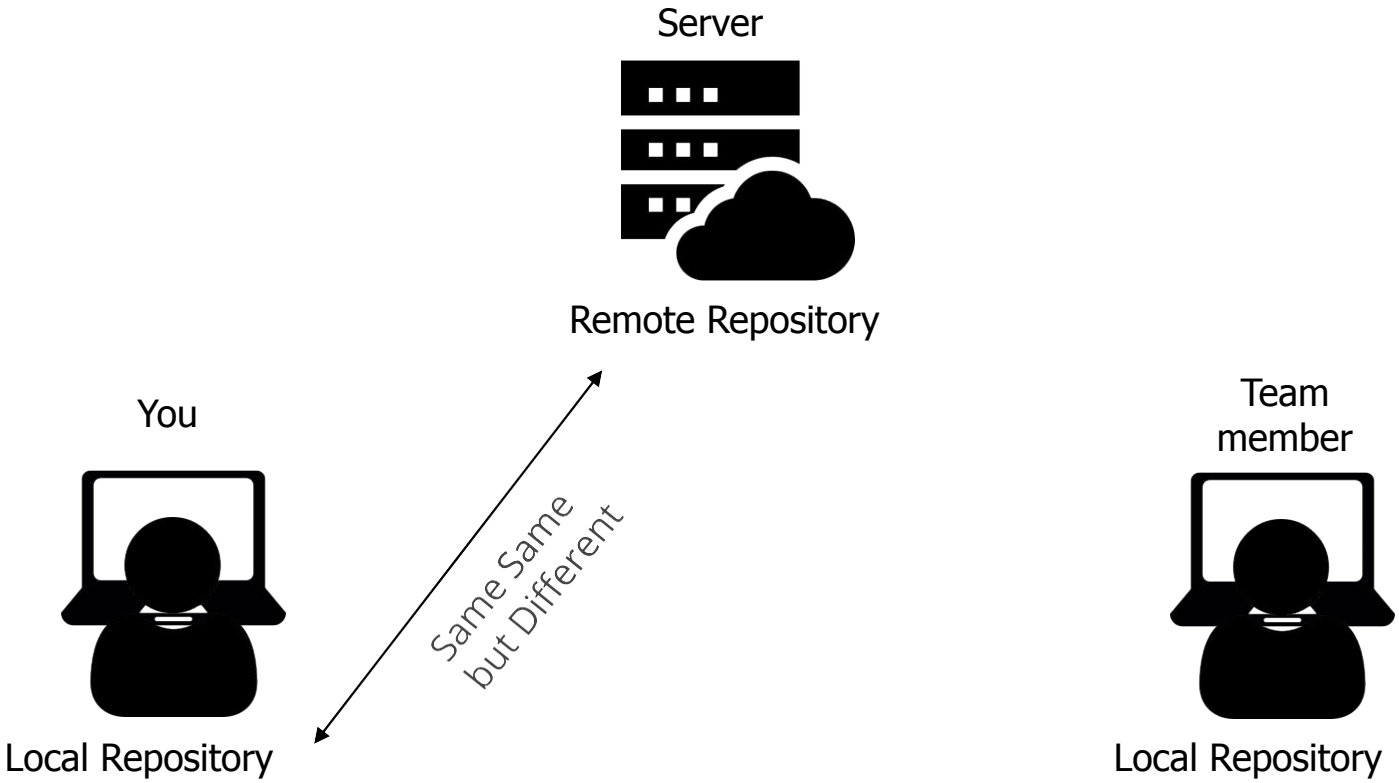
>. TechLabs



# Git Collaboration

## Local/Remote Repository

>. TechLabs



# Git Collaboration

## Where to host the Remote Repository

>. TechLabs

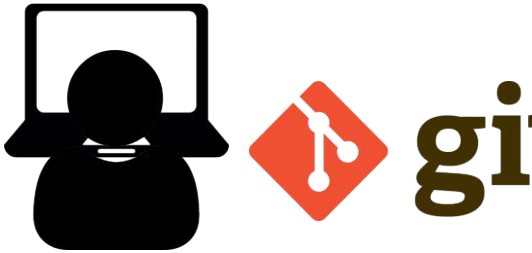

e.g. GitHub OR GitLab

Remote Repository

You


Local Repository

Team member



Local Repository

# Git Collaboration

## GitHub Collaboration

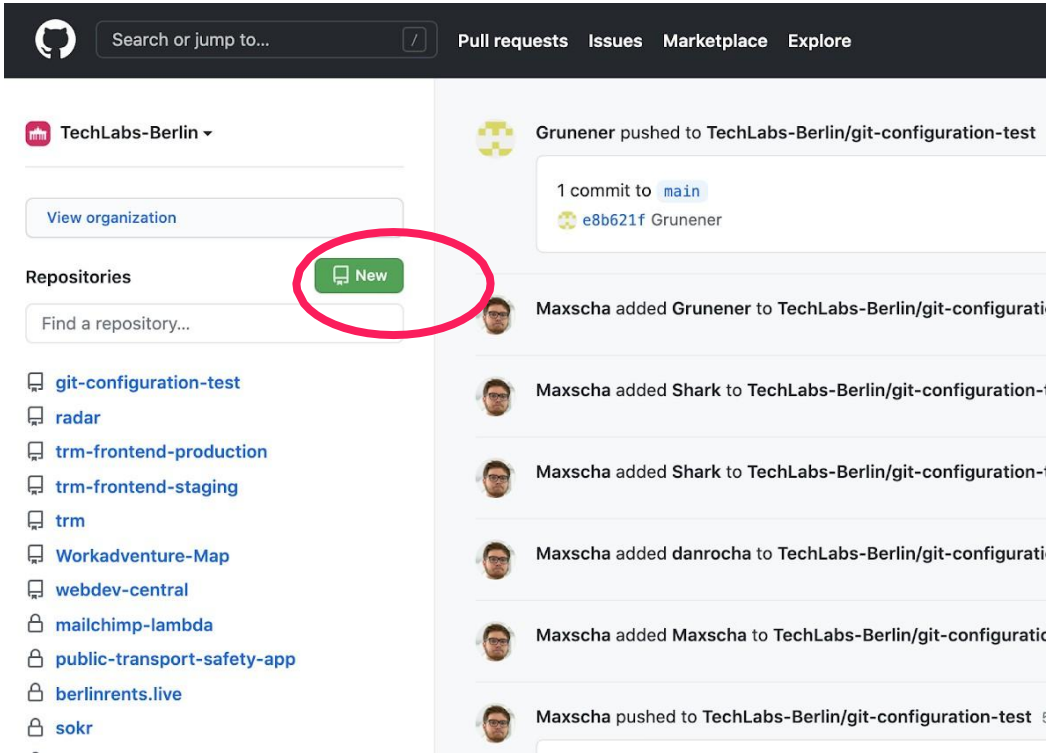
- GitHub is a cloud service, to host & share your repositories online
- You can see it as a "Dropbox for repositories"
- **Git != Github**
  - They have both different functionalities and are for different tasks
  - Git: Working **LOCALLY** on your machine & communicating with GitHub
  - GitHub: Hosting your repository, and can be seen more as a project management tool

- It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project.

# Git Collaboration

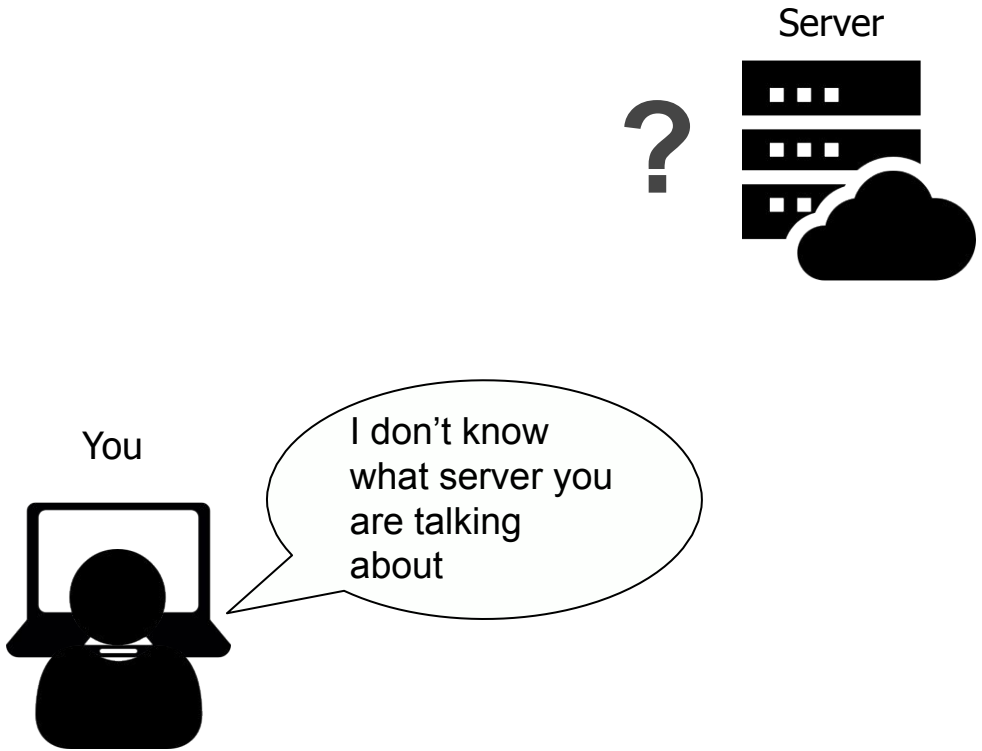
## Github New Repository

>. TechLabs



# Git Collaboration

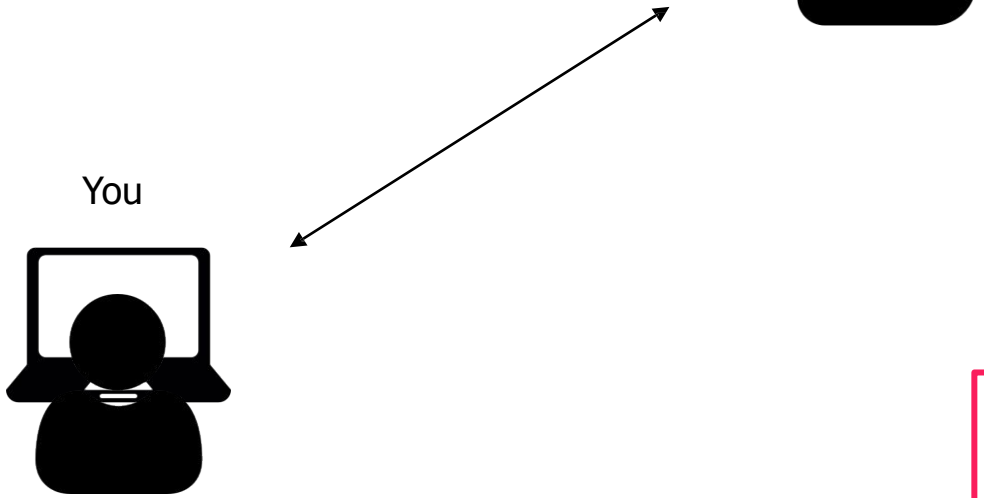
git remote add



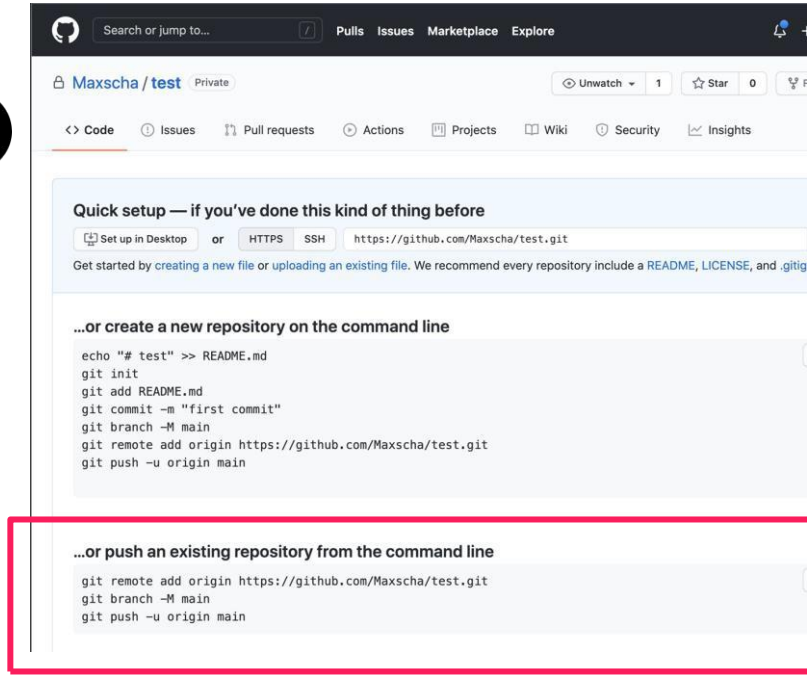
# Git Collaboration

git remote add

```
git remote add origin
https://github.com/user/frien
d  book.git
```



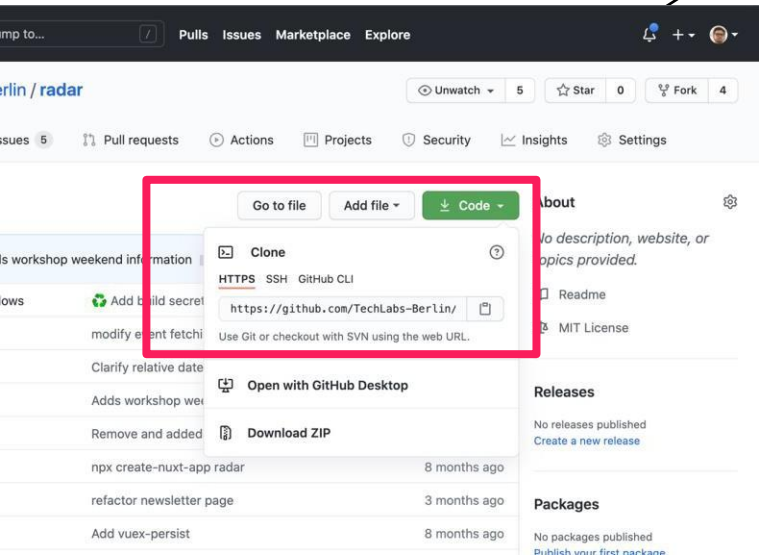

Alternative: Just copy the code from GitHub



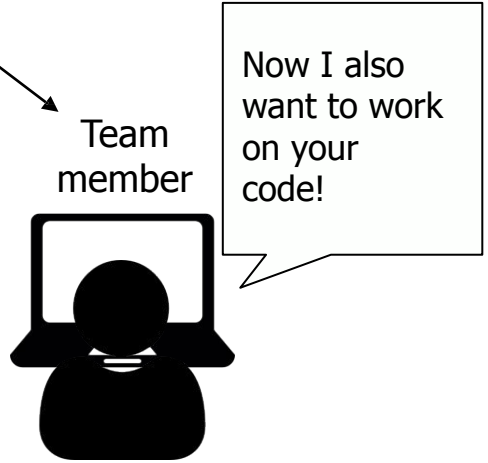
# Git Collaboration
git clone

```
git clone
https://github.com/user/frien
d  book.git
```

Alternative: Just copy the code from GitHub

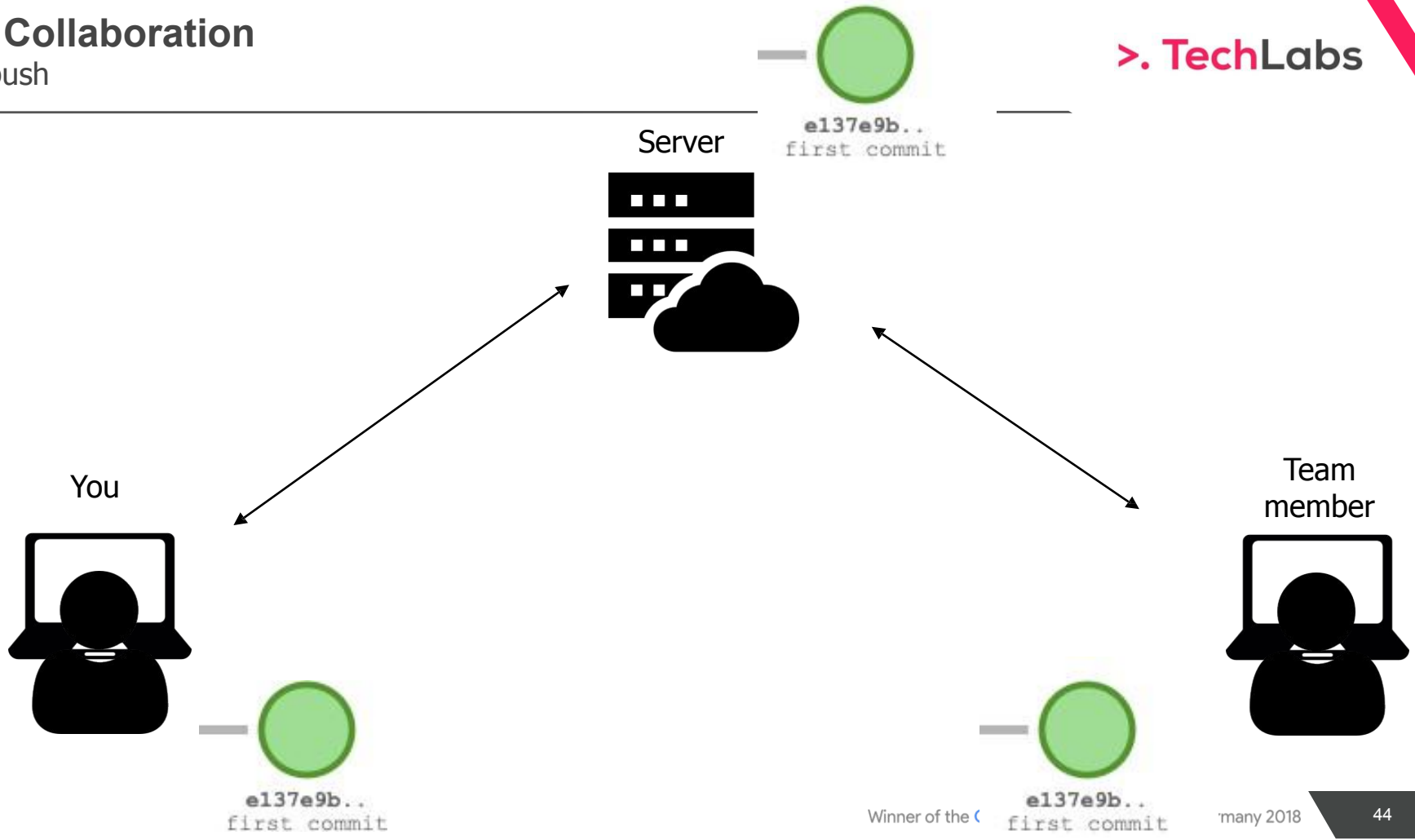






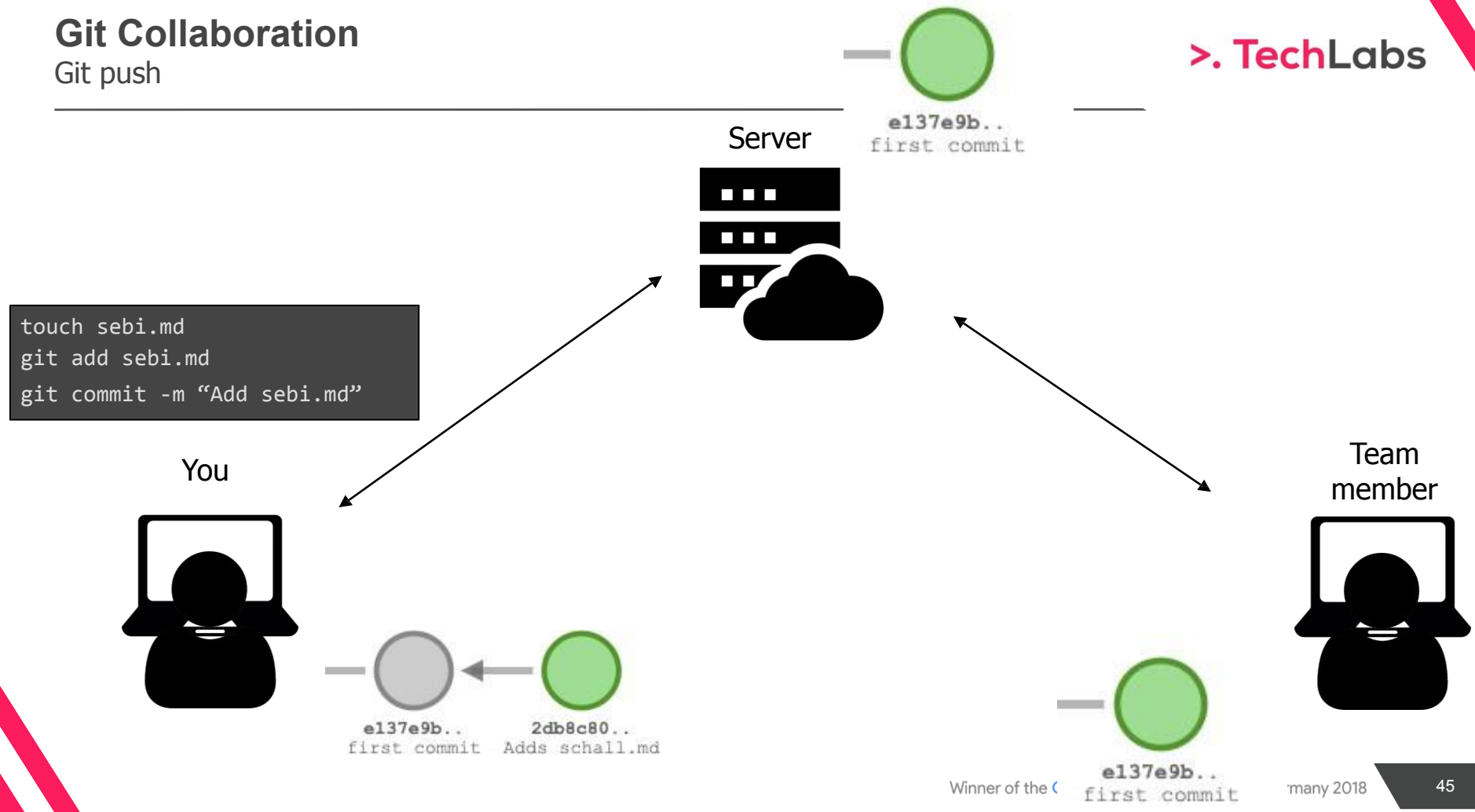
# Git Collaboration

Git push



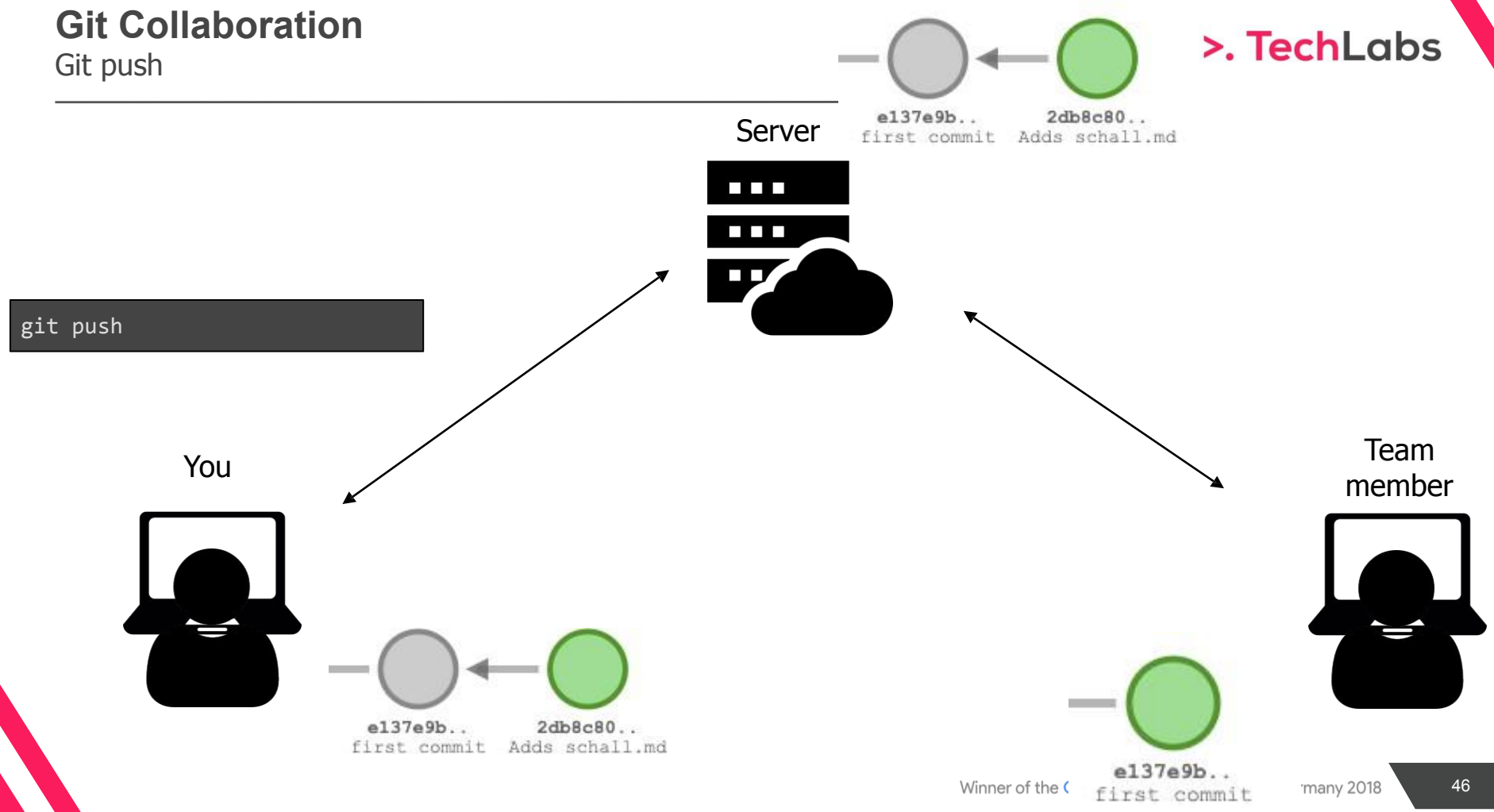
# Git Collaboration

## Git push



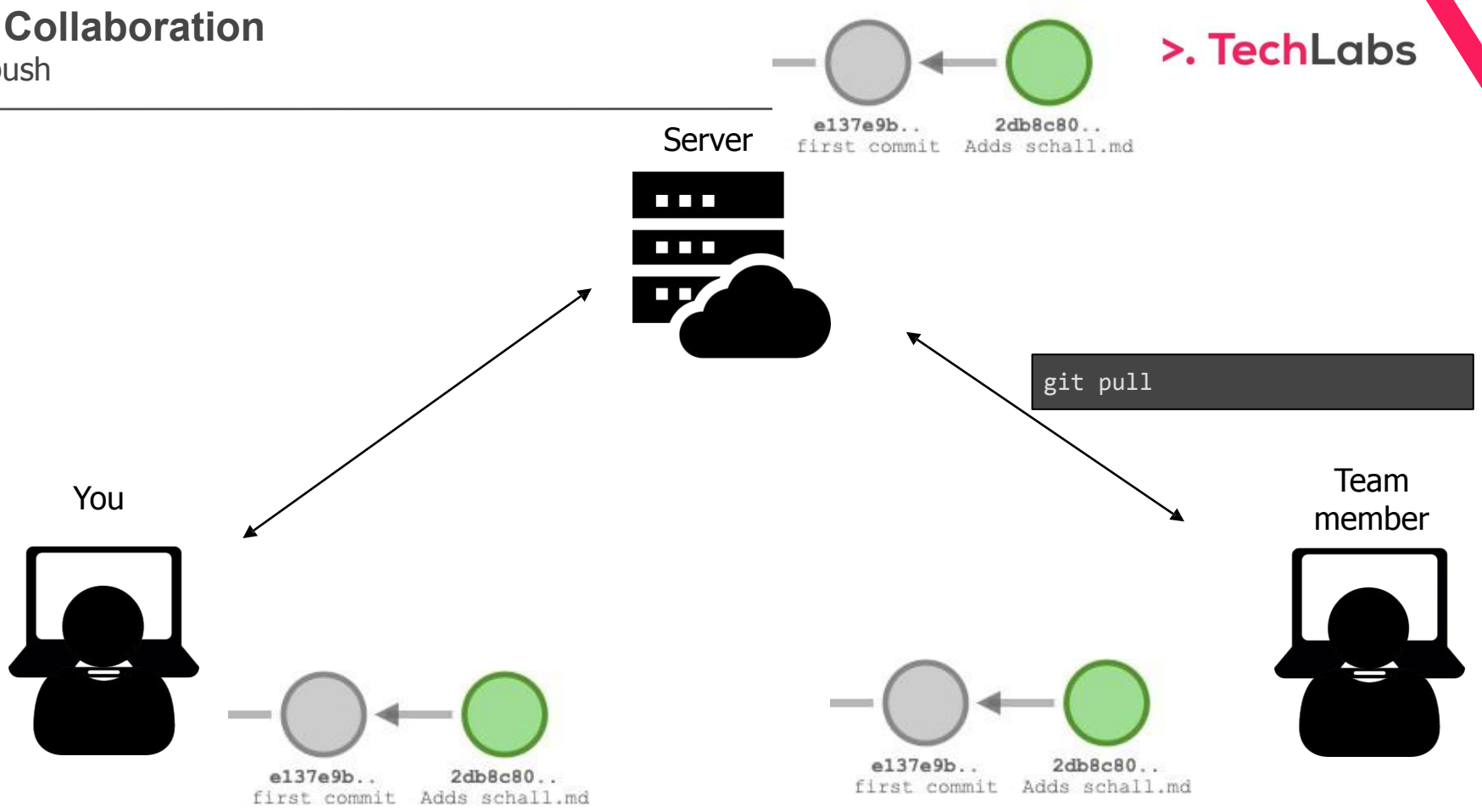
# Git Collaboration

## Git push



# Git Collaboration

## Git push



# Git Collaboration

## Tasks

>. TechLabs

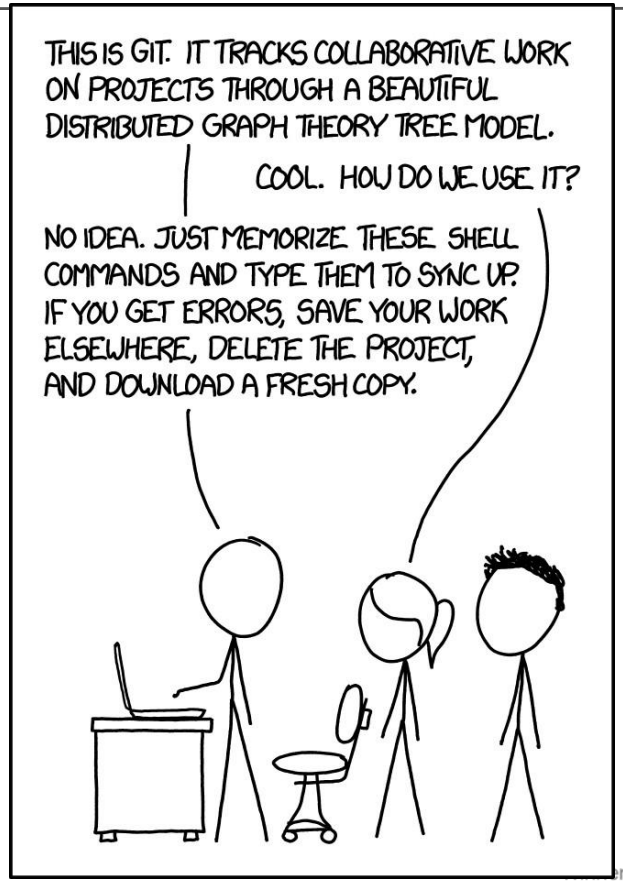
**Everyone in the team needs to do these tasks, but communicate with each other about your questions and learnings!**

1. Select one person in your group...
   a. … to create a GitHub repo and to upload (push) his existing repo there.
   b. … to add the others as a collaborator in that repository (Invitation comes via e-mail, so check your mail)
2. Everyone clone that repo (make sure not to clone it within your existing repository)
   a. Check before if you are in a repository.
   b. Use git status to check
3. Copy your lastname.txt from your old repository to the other member's repository, git add, git commit, git push (You can do the coping in the file explorer)
4. Change something in the text file of your group, add, commit, & push.
5. Play around with the repository, add and commit some things, pull and coordinate with your team members.

20 min

# Problems?

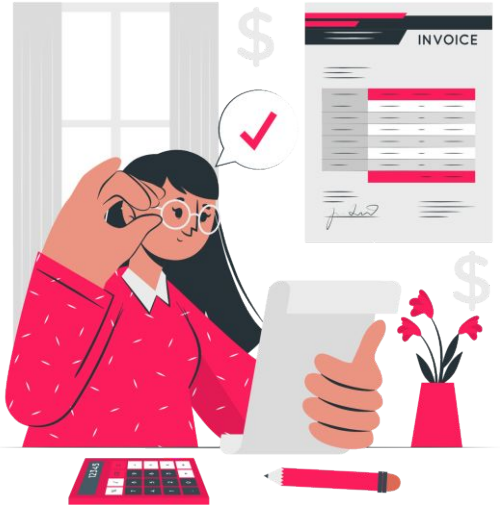
>. TechLabs



>. TechLabs

# Questions?

Look at the cheat sheet and ask your group members



# Git Collaboration
Solution

>. TechLabs

1. Select one person in your breakout room...
   a. … to create a GitHub repo and to upload his existing repo there.
   b. … to add the others as a collaborator in that repository (Invitation comes via e-mail, so check your mail)
2. Everyone clone that repo (make sure not to clone it within your existing repository)
   a. Check before if you are in a repository.
      i. pwd
   b. Use git status to check
      i. git status
3. Copy your lastname.txt from your old repository to the other member's repository, git add, git commit, git push(You can do the coping in the file explorer)
   a. git add <filename>; git commit -m “message”; git push
4. Change something in the text file of your group, add, commit, & push.
5. Play around with the repository, change push & pull and coordinate with your team members until time is over.


# Git Collaboration 2

# Git Collaboration 2

Or how to really work together without problems

>. TechLabs

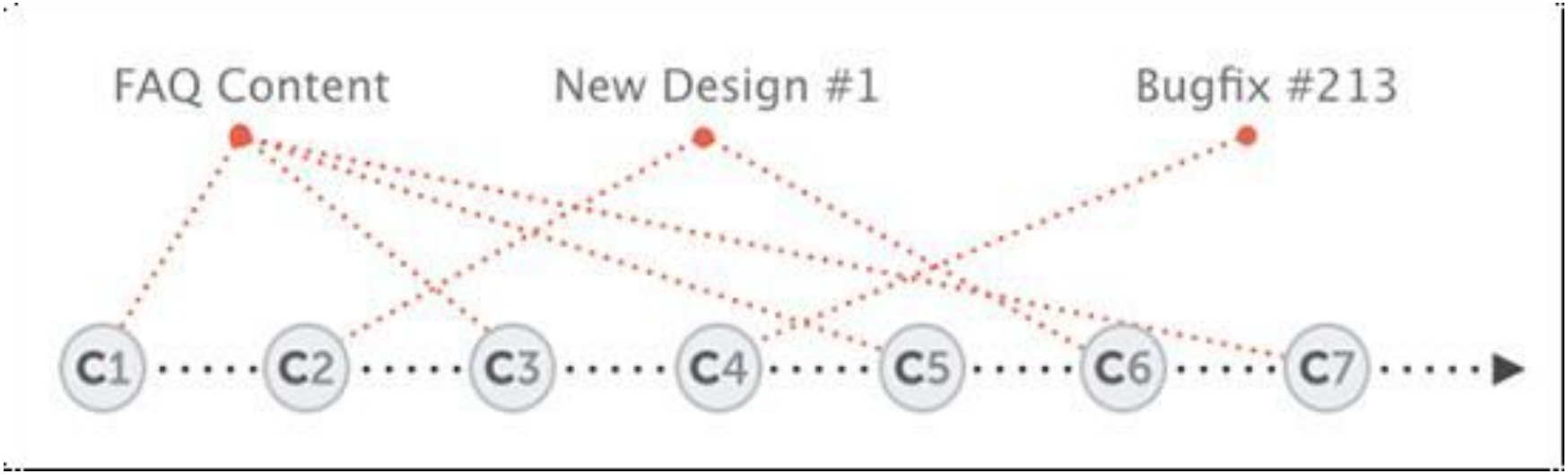
- You want to work in different contexts until you are done
- Creating contexts is easy

- Combining these contexts is hard, but git has helpful ways with this
  - (It is probably the most complex topic for a git beginner, but once you put your head around it will be kinda easy)

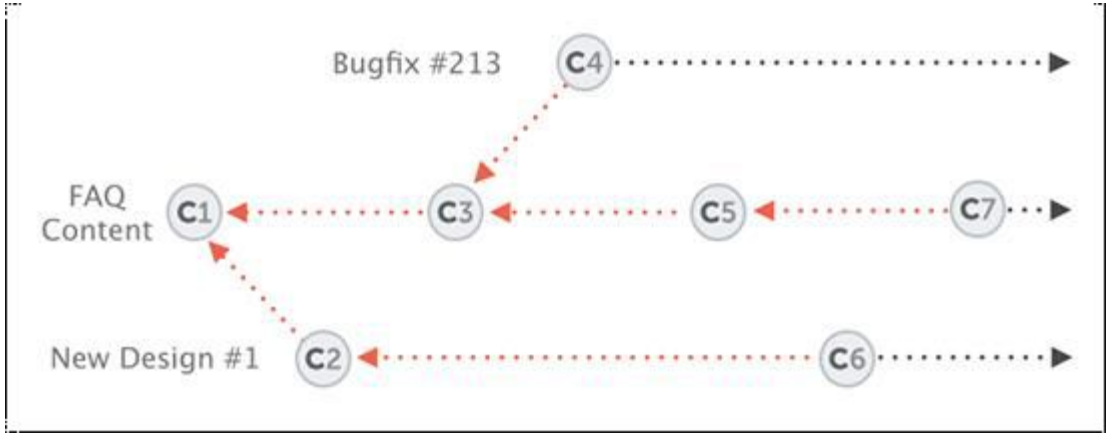
# Git Collaboration 2

Working on contexts!

>. TechLabs



# Git Collaboration 2
## Branches




bugfix

main

new_design

# Git Collaboration 2

## Default branch naming

>. TechLabs

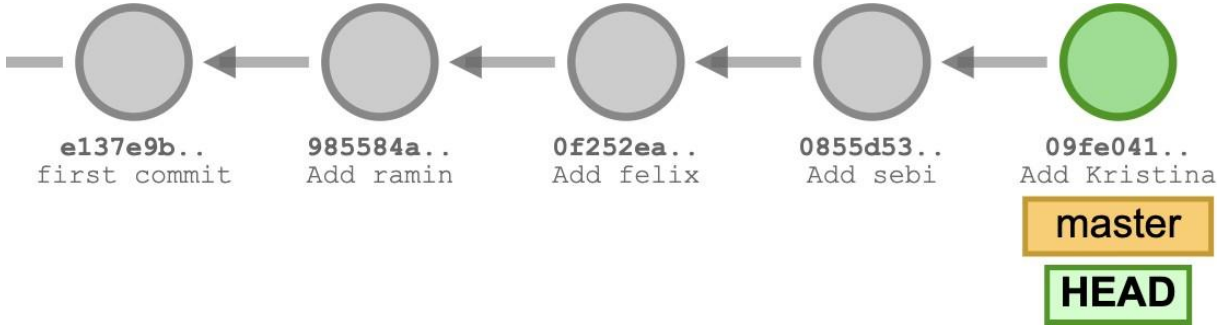
- **Master** was the common naming for the default branch until quite recently
- After the brutal death of George Floyd and the Black Lives Matter protests, tech companies wanted to show their support for the black community by abandoning non-inclusive terms such as master, slave, blacklist, and whitelist
- Your setup should already include that per default also local repositories have their initial branch called **main**

- Just keep an eye if you work with older repositories.
  -> *git config --global init.defaultBranch <Name>* to set the the default name for initial branches; *git branch -m <Name>* to rename branch you are currently in)

# Git Collaboration 2

Examples with head & checkout

>. TechLabs




```
.
└── FriendBook/
    ├── .git
    ├── sebi.md
    ├── ramin.md
    ├── felix.md
    └── kristina.md
```

# Git Collaboration 2

Examples with head & checkout


```
git checkout 0f252ea
```

```
.
└── FriendBook/
    ├── .git
    ├── ramin.md
    ├── felix.md
```

# Git Collaboration 2

Examples with head & checkout


```
git checkout 0855d53
```

```
.
└── FriendBook/
    ├── .git
    ├── sebi.md
    ├── ramin.md
    └── felix.md
```

# Git Collaboration 2

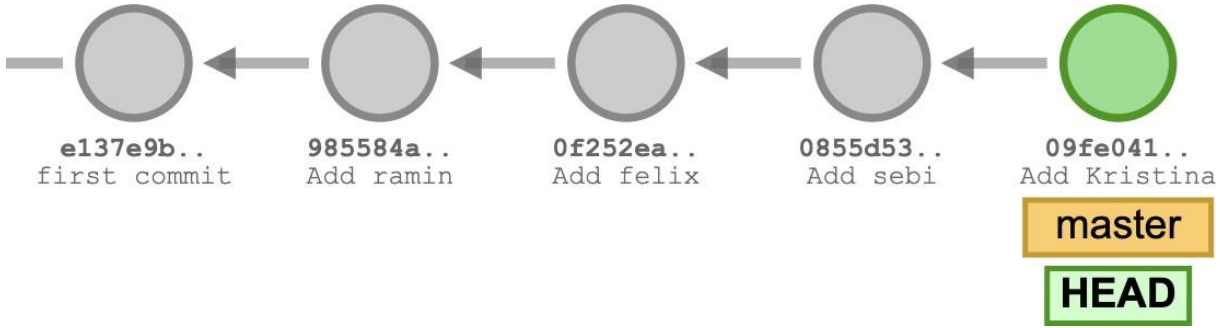
Examples with head & checkout


```
git checkout 985584a
```

```
.
└── FriendBook/
    ├── .git
    ├── ramin.md
```

# Git Collaboration 2

Examples with head & checkout




```
git checkout main
```

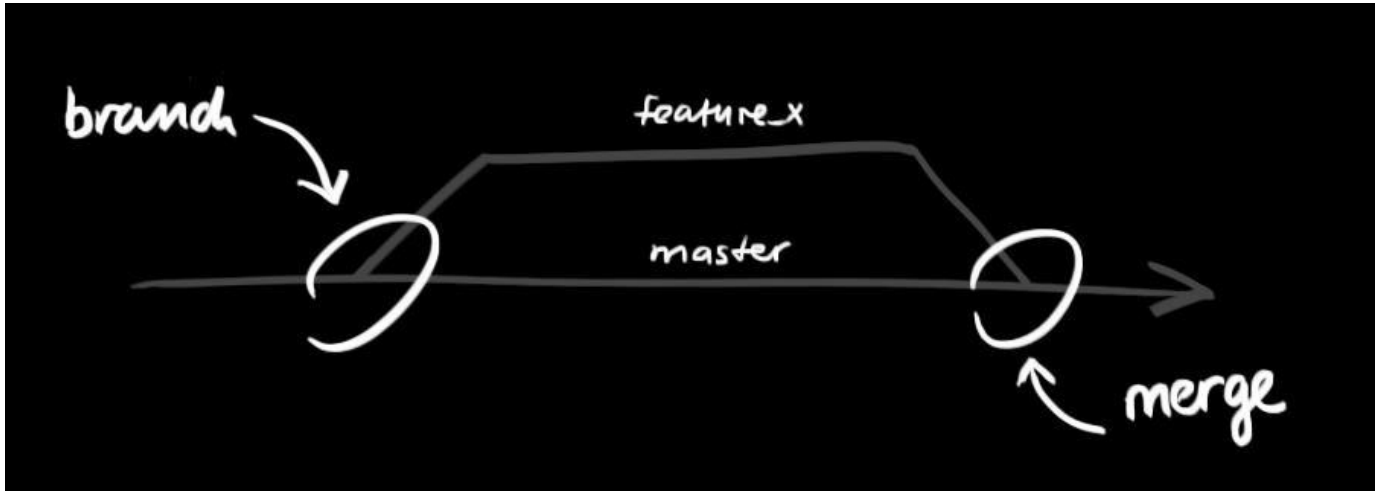
```
.
└── FriendBook/
    ├── .git
    ├── sebi.md
    ├── ramin.md
    ├── felix.md
    └── kristina.md
```

# Git Collaboration 2

Branches!

>. TechLabs

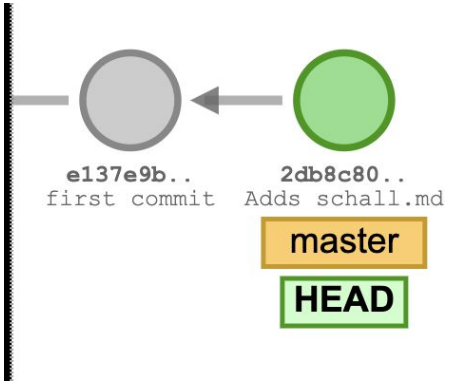
A branch is a possibility to develop feature_x in a save context, without interfering with other work. It allows easy separation, comparison and roll_back.

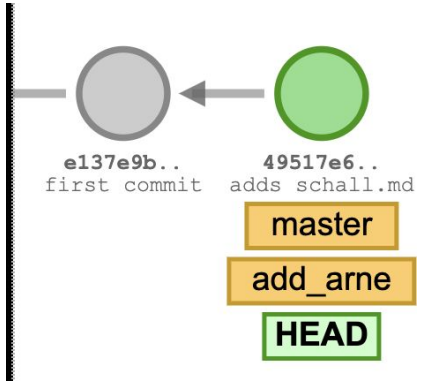


# Git Collaboration 2

How to create branches?

```
git checkout -b add_arne
```

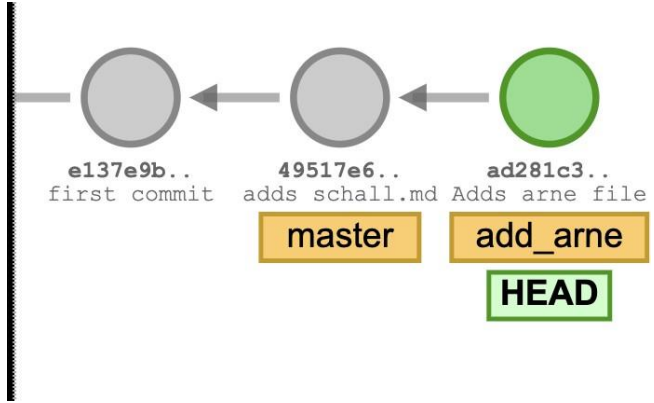


# Git Collaboration 2

How to work in branches

```
touch arne.md
git add
arne.md
git commit -m “adds arne file”
```

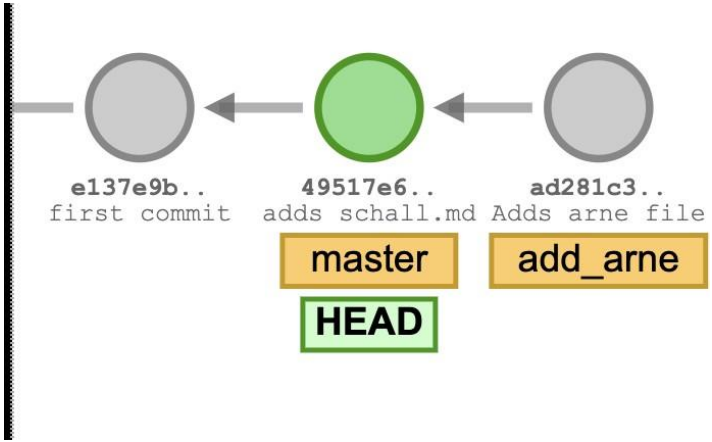


# Git Collaboration 2

How to switch between branches

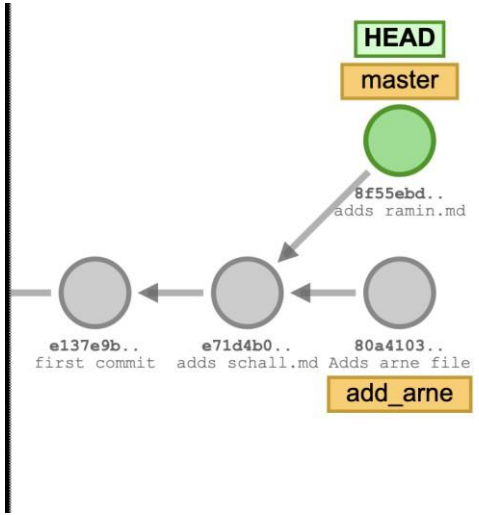
>. TechLabs

```
git checkout main
```



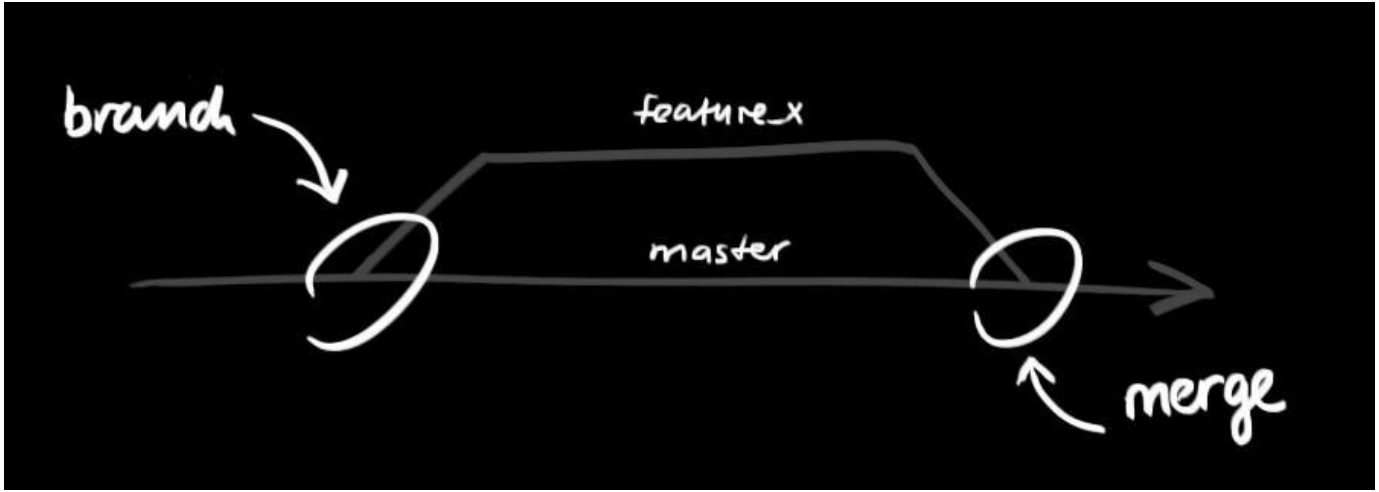
# Git Collaboration 2

Add commit to main



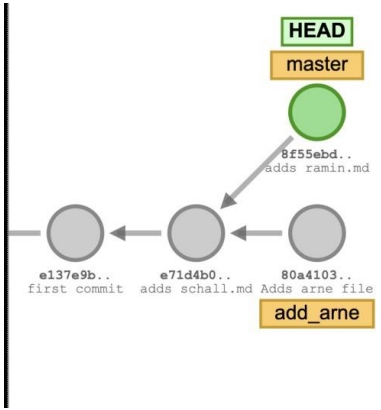
# Git Collaboration 2

What are branches?

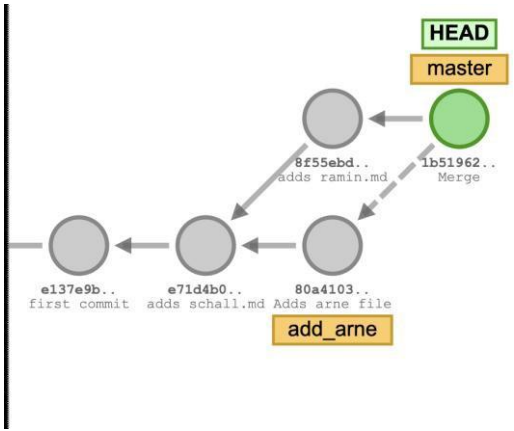


>. TechLabs

## Explain **git merge**




```
git merge add_arne
```



# Git Collaboration 2

## When to merge?

>. TechLabs

- When to merge a branch into main is a hard question!
  - Too early and you will need to recreate the context to often
  - Too late and the contextes moved too far away from each other

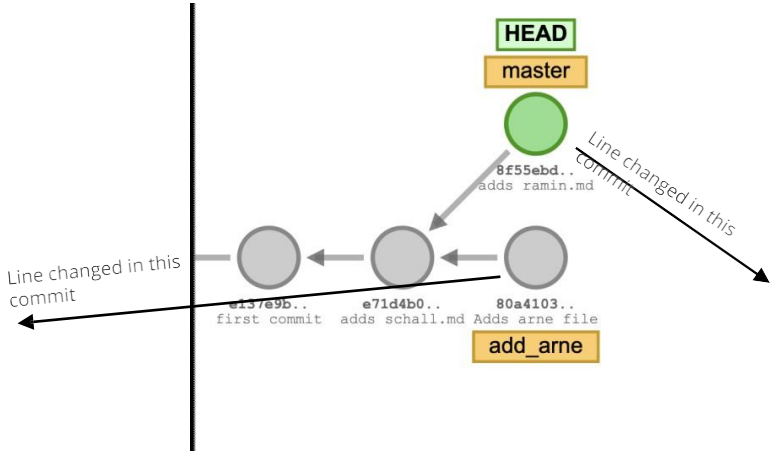
- Merge when:
  - A feature/bugfix is done
  - If the feature takes longer than a week to implement, find a subset of that feature to implement within a week and merge that

>. TechLabs

- When do **merge conflicts** happen?
  - When the same files and lines get edited in two different branches and these branches are merged together

**sebi.txt (in branche add_arne)**

Name: Arne
Bday: 34.12.94
Favorite Food: Spätzle




**sebi.txt (in branche main)**

Name: Arne
Bday: 34.12.94
Favorite Food: Maultaschen

# Git Collaboration 2

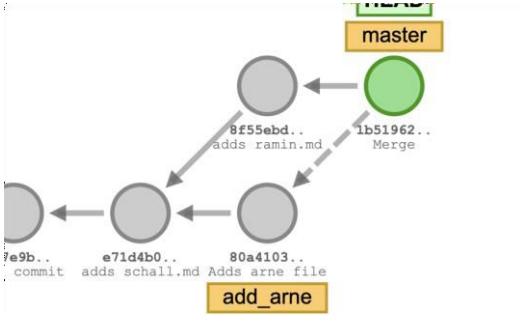
What is the correct line?

>. TechLabs

Assume the same line (number) of the same document in different branches.

Now there is a merge… Which line is the right one?

**Favorite Food: Spätzle?**




**Favorite Food: Maultaschen?**

# Git Collaboration 2

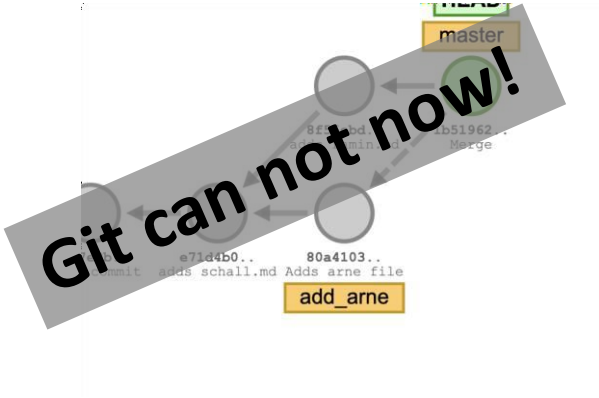
What is the correct line?

>. TechLabs

Assume the same line (number) of the same document in different branches.

Now there is a merge… Which line is the right one?

**Favorite Food: Spätzle?**




**Favorite Food: Maultaschen?**

# Git Collaboration 2

How to fix

>. TechLabs

**sebi.txt**

Name: Arne
Bday: 14.12.94

```
<<<<<<<<<< HEAD
Favorite Food: Spätzle
==========
Favorite Food: Maultaschen
>>>>>>>>>> add_arne
```

Your decision!

1. Fix it in your text editor, by choosing the line(s) and removing all the other things
2. Add fixed file
3. commit

```
$git status
On branch main
 You have unmerged paths.
   (fix conflicts and run "git commit")

 Unmerged paths:
   (use "git add <file>..." to mark resolution)

        both modified:   contact.html

no changes added to commit (use "git
add"  and/or "git commit -a")
```

# Git Collaboration 2

## Merging is hard

>. TechLabs

- Probably the most difficult topic today, try to create merge conflicts in your own projects to solve them.
- If you are merging conflicts from branches with other people work together with them
- Your code is not more important than code from others!

# Git Collaboration 2

And how to do this remote?

>. TechLabs

```
git pull
```

```
git push
```

```
git checkout --track
origin/<branch-name>
```

**READ THE OUTPUT!**

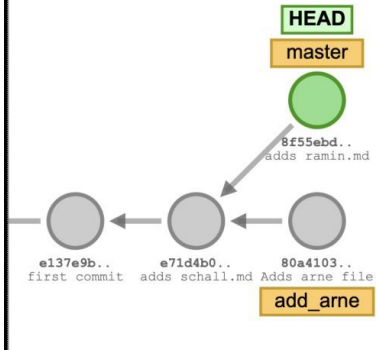
# Git Collaboration 2

How to do this remote

>. TechLabs


Server

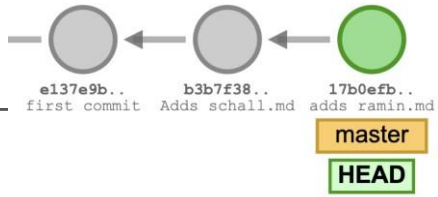
You




Team member

# Git Collaboration 2

How to do this remote

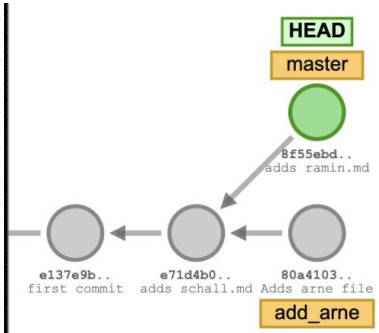



TechLabs

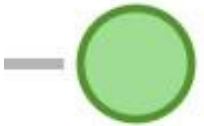
Server

```
git push
```

You




Team member

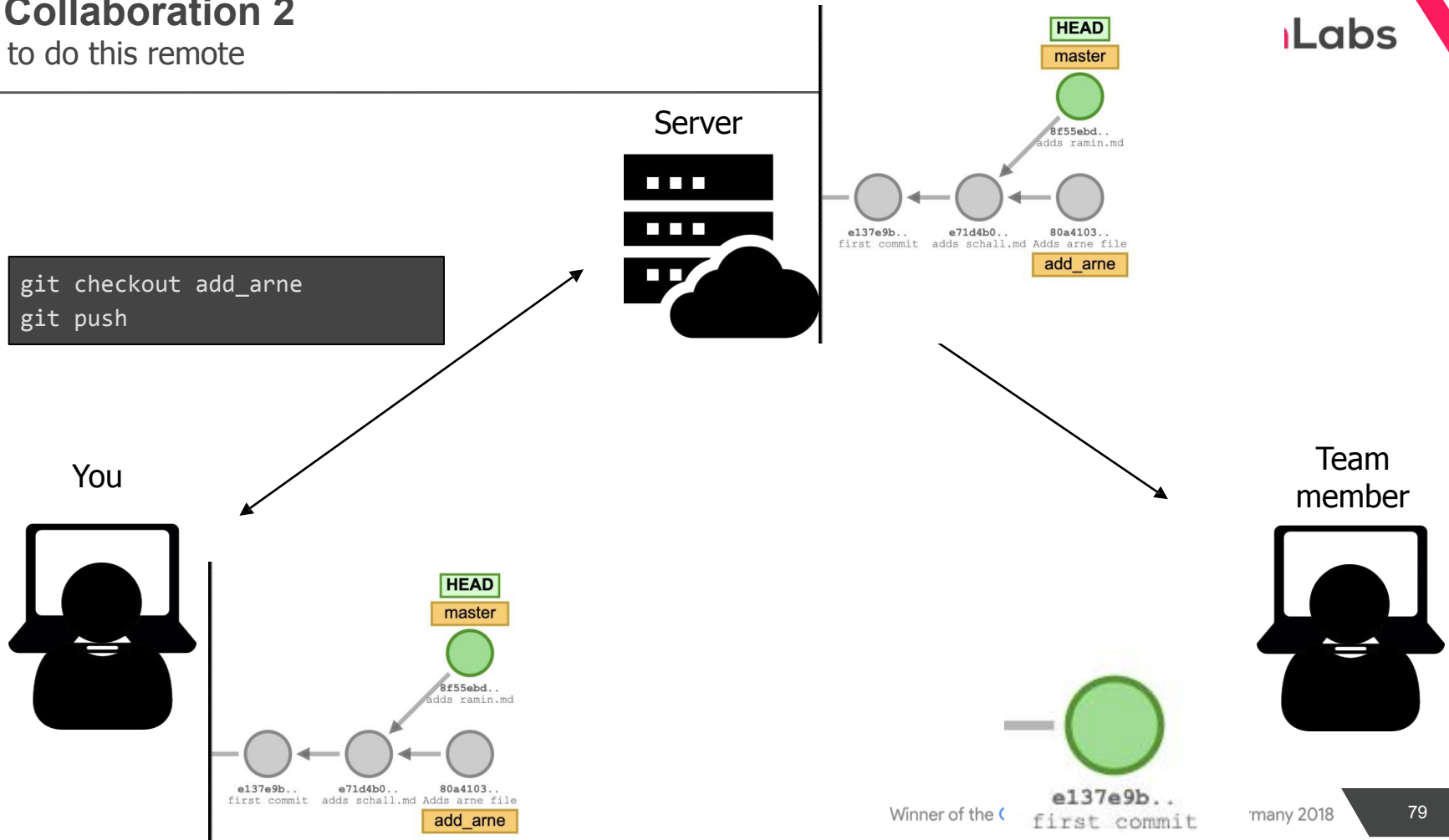



Winner of the

many 2018

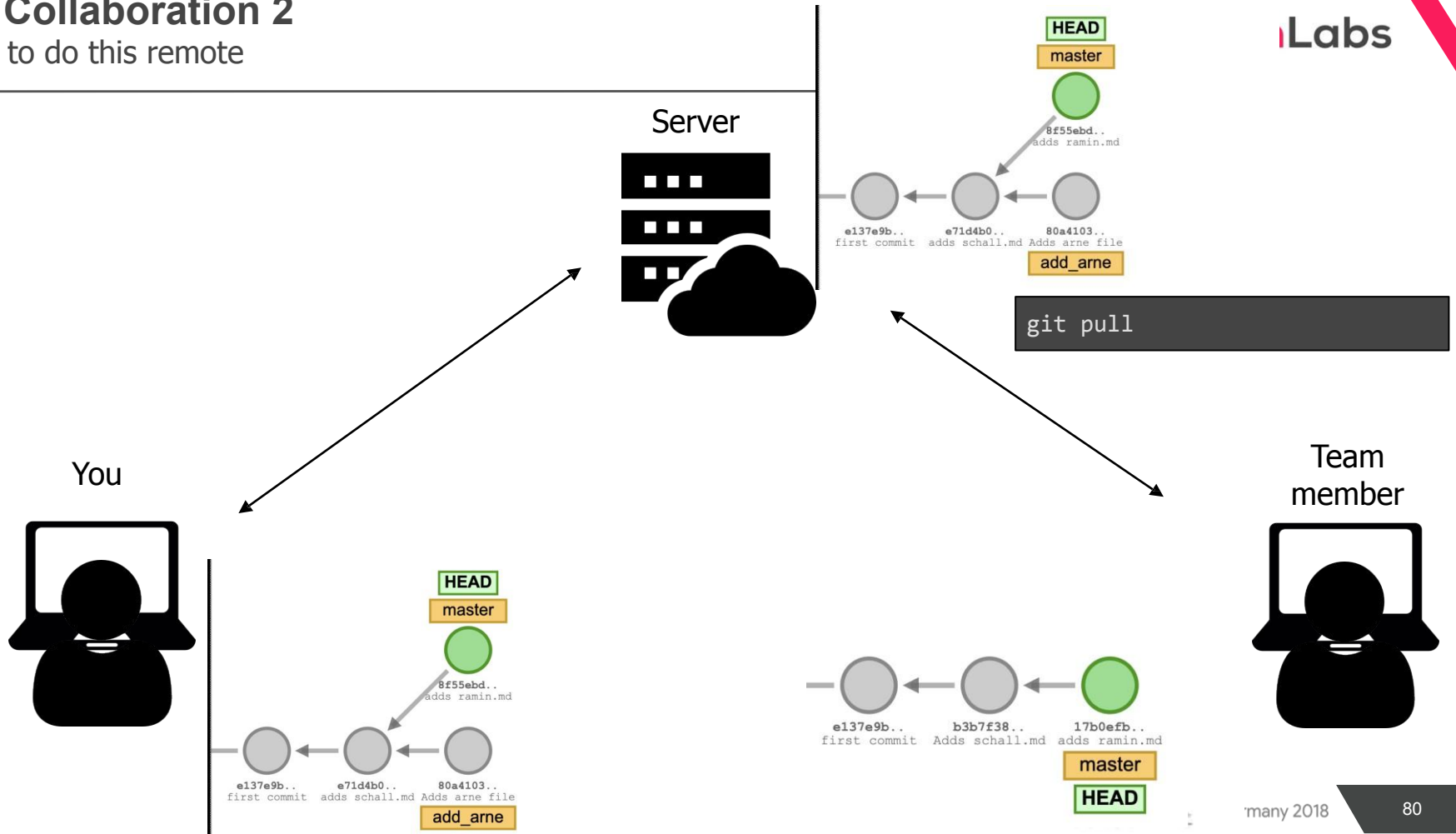
# Git Collaboration 2

How to do this remote



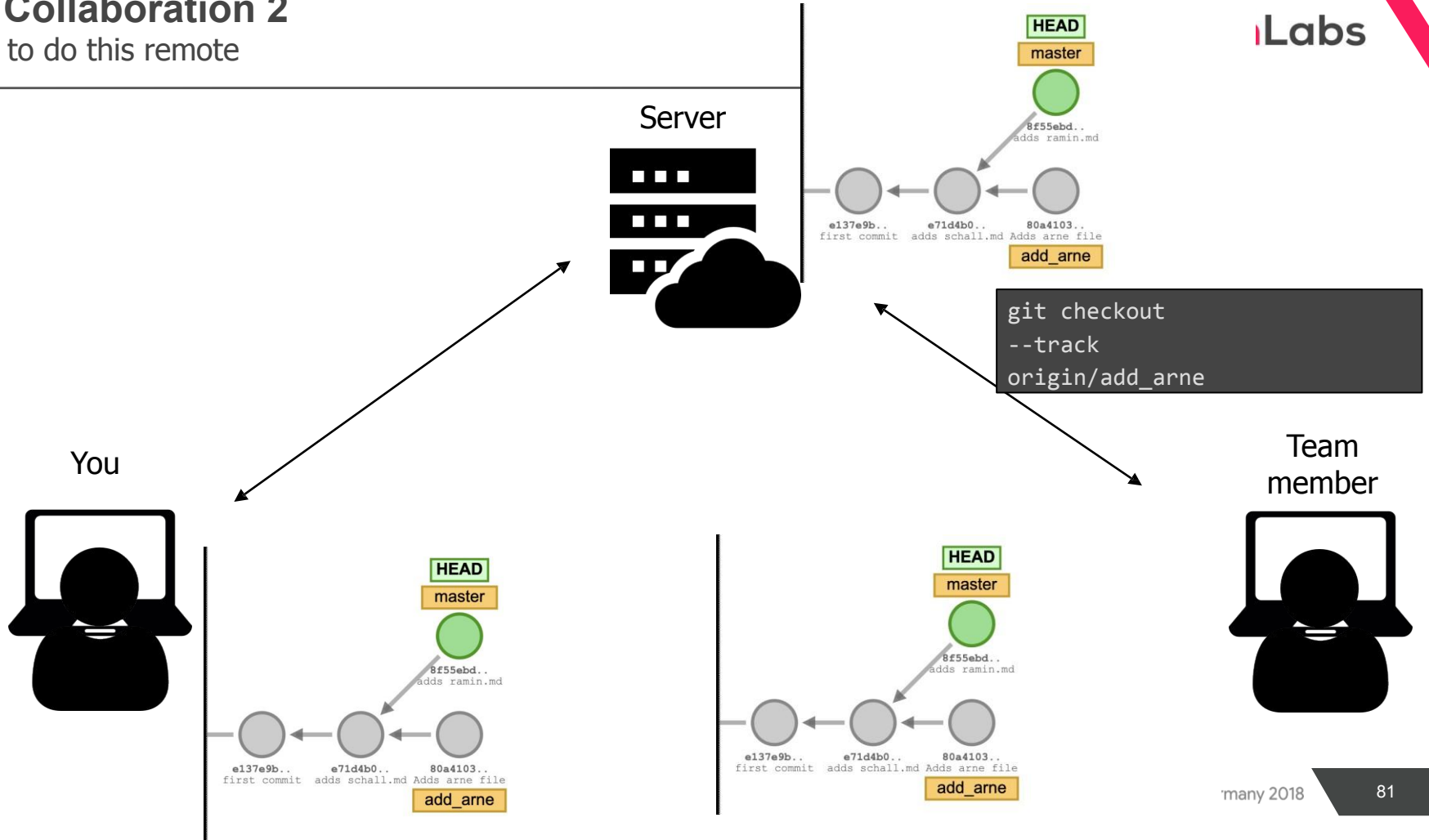
# Git Collaboration 2

How to do this remote



# Git Collaboration 2

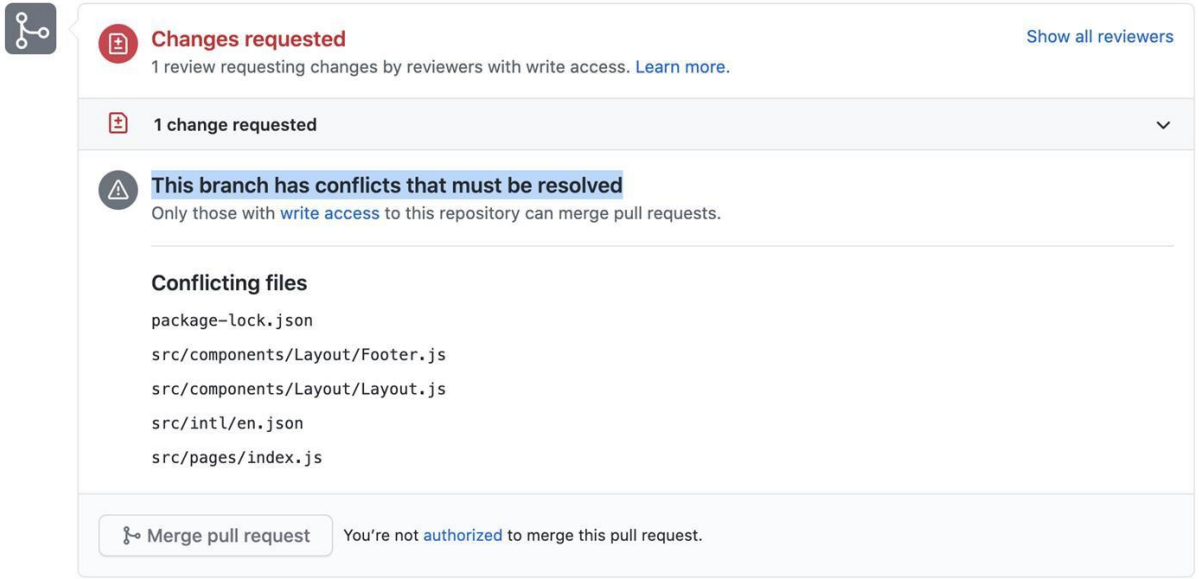
How to do this remote



# Git Collaboration 2

## Merge conflicts Online

```
git checkout main

git pull

git checkout <branch>

git merge main
[ ... resolve any conflicts ... ]

git add [files that were conflicted]

git commit

  git push
```



# Git Collaboration 2

## Branch Protection

>. TechLabs

- Github allows to configure permissions for branches more granular
- Branch protection forbids to push directly to main
- It is seen as good practice in the industry
- Is used for our tasks, just follow the steps in the cheat sheet

# Git Collaboration 2

Local Tasks

>. TechLabs

**Everyone in the team needs to do these tasks, but communicate with each other about your questions and learnings!**

1. Work in your local repository from Part 1
2. Create a new branch with a name of your choice
3. Make sure you are in that new branch
4. Add a new file with some content
5. Add and commit
6. Return to the main branch
7. Check Git status & and think about what happened to the file you have just added
8. Merge the created branch into main
9. Check status and file content
10. Make sure you are in main
11. Now try to create a merge conflict
    a. Make a new branch and change 1-2 lines in a file, add & commit
    b. Go back to main, make a different change in the same lines in the file you have just changed, add & commit
12. Merge your new branch into main
13. Solve the merge conflict

20 min

>. TechLabs

# How was it?


# Wrap-Up

# Wrap-Up
## Summary

>. TechLabs

- What did you learn!
  - Work in local repositories
  - Work collaboratively in remote repositories with your team members
  - Branching and reviewing
  - How to get a repository in the state I would like to have!
- So, can I use Git now?
  - Yes, you can use it
  - It is okay to have the cheat sheet printed out next to you while working with Git!
  - And sometimes you need to ask or google commands. (Reach out to us so we can adapt the cheat sheet)

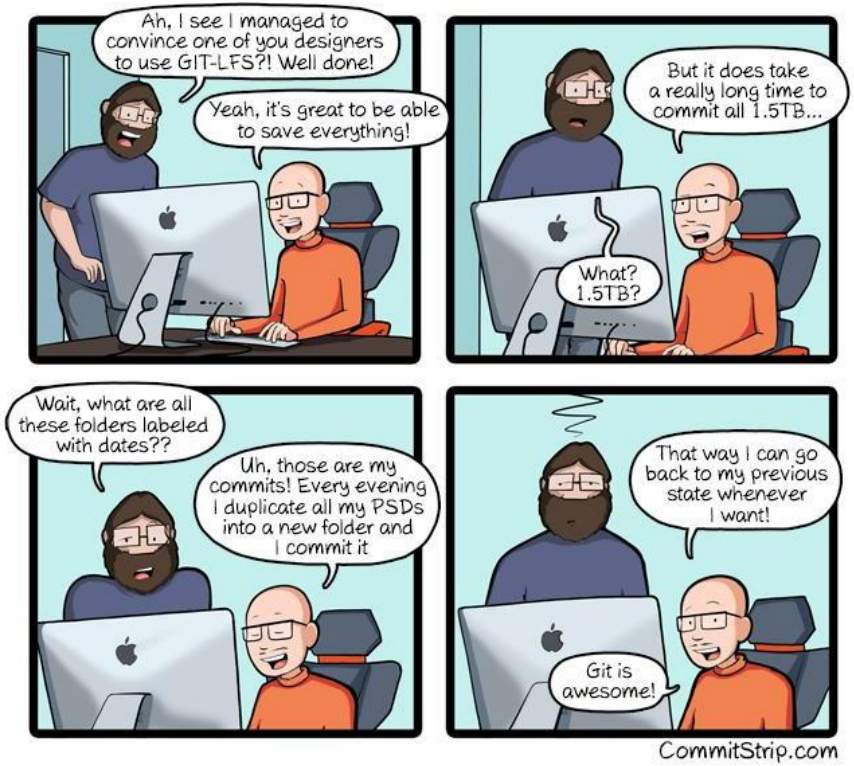
# Wrap-Up
## What to learn further

>. TechLabs

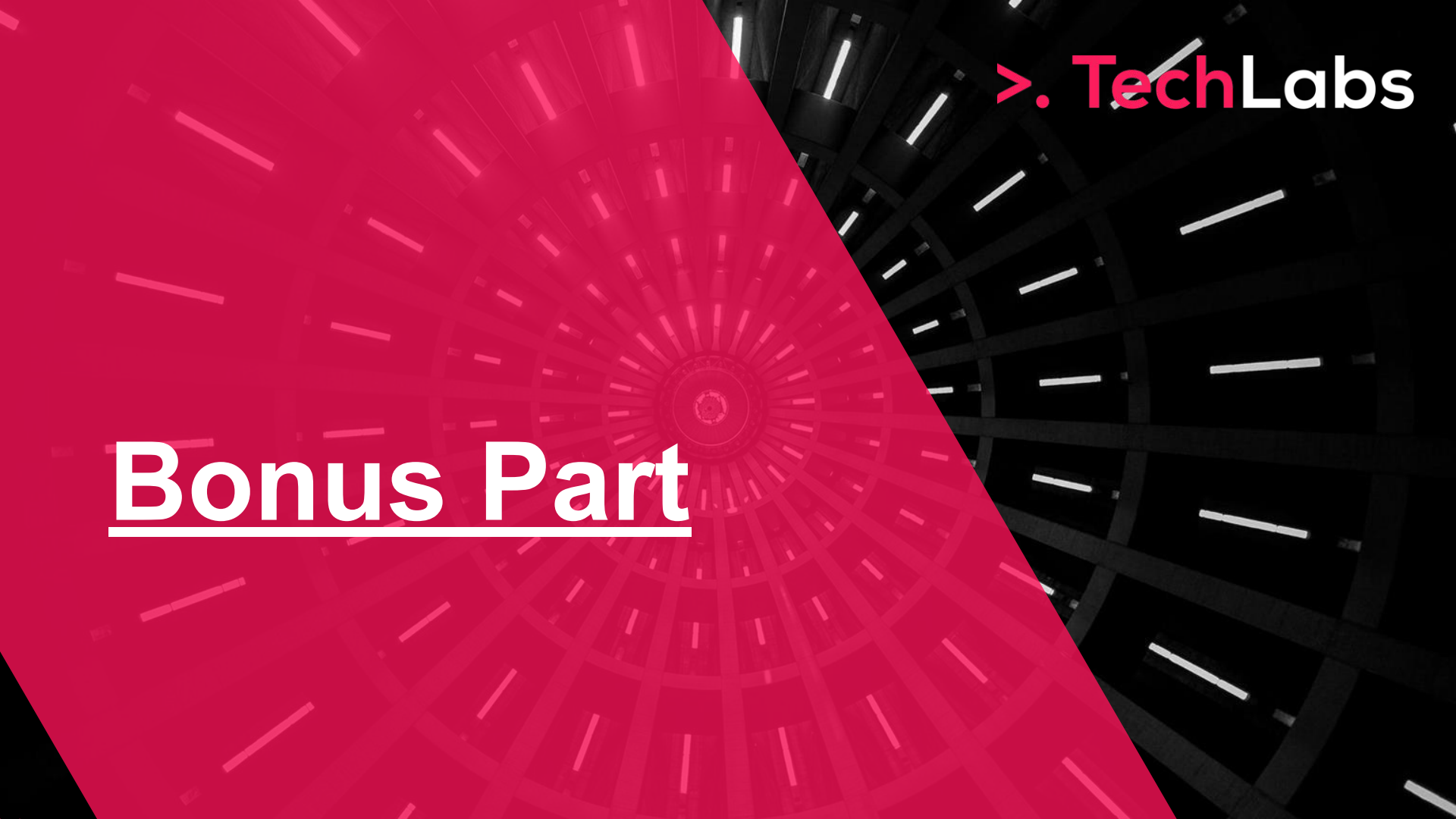
- Practice Git:
  - Learn Git even more interactive:
    - https://learngitbranching.js.org/
    - https://github.com/jlord/git-it-electron#what-to-install (We recommend this one after the workshop, if you have some more time!)
    - http://git-school.github.io/visualizing-git/
- You want to learn more about:
  - Git Internas & Details
    - https://git-scm.com/book/en/v2
  - Git Branching and Merging
    - https://learngitbranching.js.org/
  - Git merge conflicts:
    - https://www.youtube.com/watch?v=__cR7uPBOIk
- Suggest next steps here
  - Practice!
  - Set-Up the same Repository twice on your computer (in different folder) with the same remote repository and practice!
  - Think about non-programming tasks you could use git for and use it!
- You still don‘t like the terminal?
  - Invest the time to get more familiarized with it
    - https://www.youtube.com/watch?v=oxuRxtrO2Ag
  - Use a Gui Git Tool
    - Recommendation: https://git-fork.com/

# Wrap-Up
Not just code!

>. TechLabs



# Thanks for participating!




# Bonus Part

# Git Collaboration 2

Task 1 - **Bonus**

>. TechLabs

Everyone in the team needs to do these tasks, but communicate with each other about your questions and learnings!

1. Now we do this collaboratively, so we work in the same repo as in part 2
2. Repository Owner: Turn on the Branch Protection (Look at the cheat sheet how to do this)!
3. Everyone creates a new branch add/lastname_description locally in that repository
4. Add a file and commit to your new branch.
5. Push the new branch online as a remote branch
6. Create a merge-request to merge your new branch into main.
7. The original owner now needs to review and merge all the merge-requests.

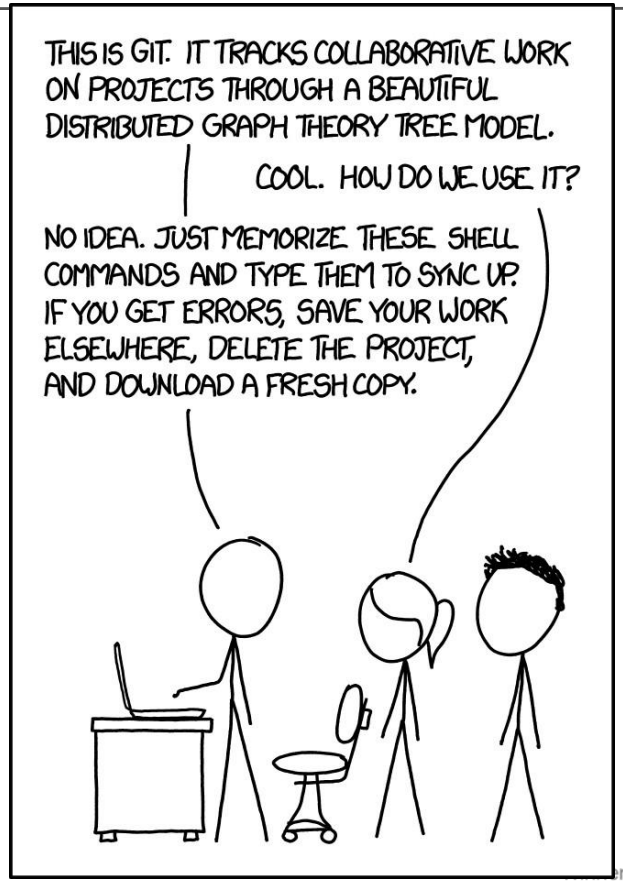
20 min


# Git - Repair my Stuff

(Bonus Time)

# Git Repair my Stuff

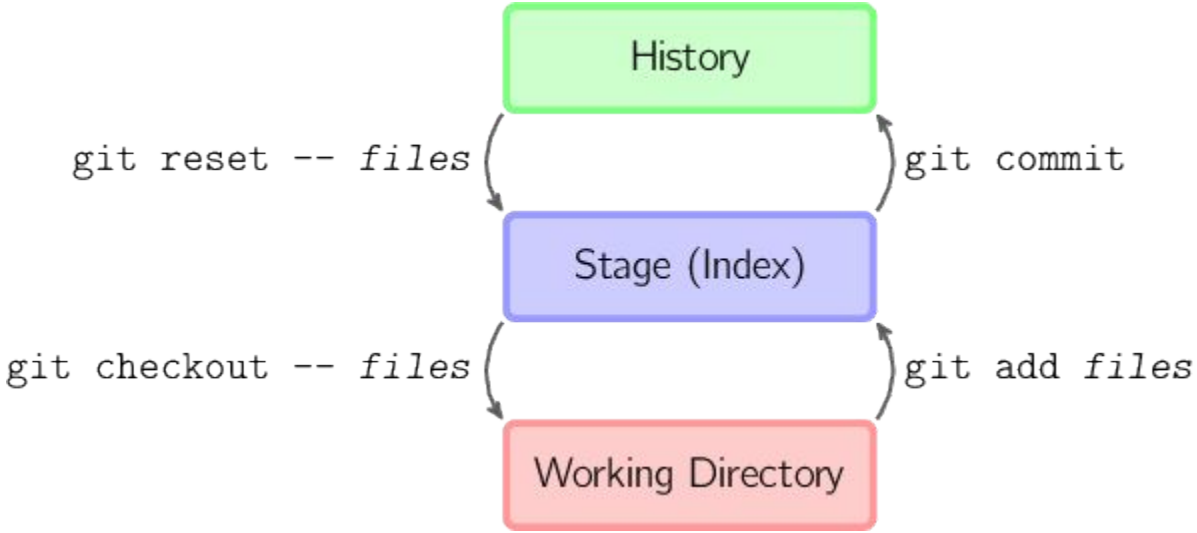
Something is not working!



# Git Repair my Stuff

## History and Stages

>. TechLabs

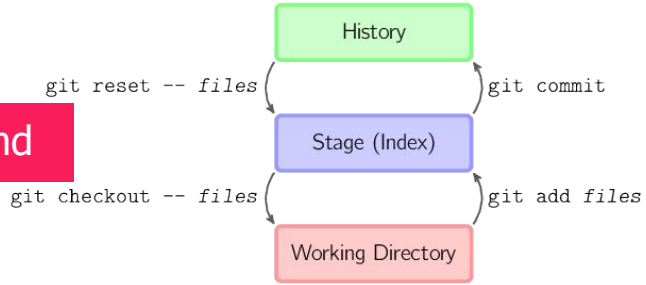


# Git Repair my Stuff

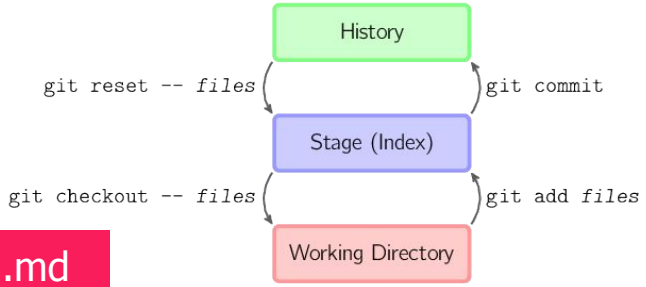
Moves from staging to working directory

>. TechLabs

## git checkout -- <bad filename>



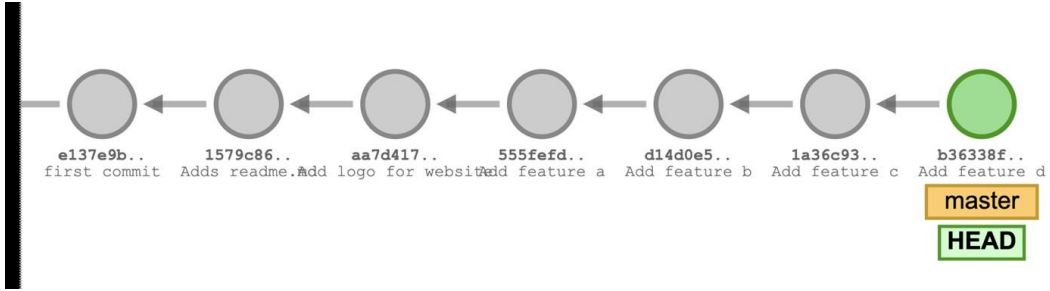

sebi.md



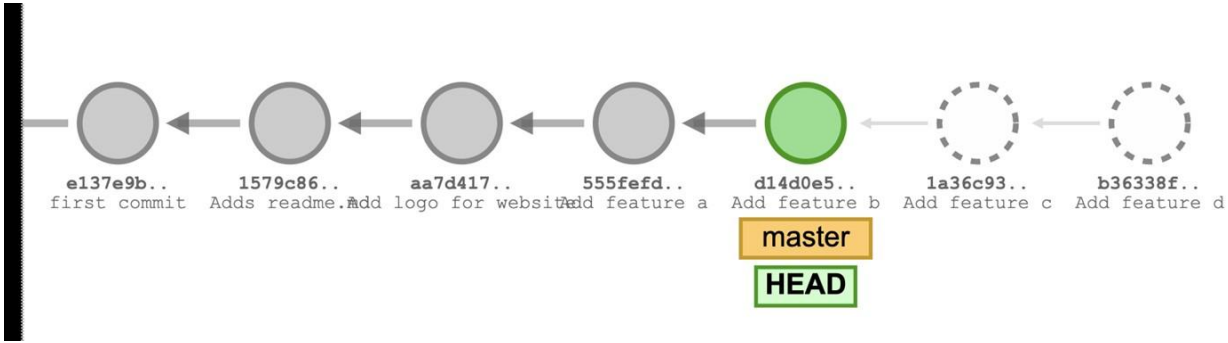

sebi.md

# Git Repair my stuff

## Undo Local Changes




```
git reset --hard <last good SHA>

git reset --hard d14d0e5
```



# Git How to Repair

## Undo a public change

>. TechLabs

Scenario: You just ran git push, sending your changes to GitHub, now you realize there's a problem with one of those commits. You'd like to undo that commit.

Undo with: git revert <SHA>

What's happening: git revert will create a new commit that's the opposite (or inverse) of the given SHA. If the old commit is "matter", the new commit is "anti-matter"—anything removed in the old commit will be added in the new commit and anything added in the old commit will be removed in the new commit.

This is Git's safest, most basic "undo" scenario, because it doesn't alter history—so you can now git push the new "inverse" commit to undo your mistaken commit.

>. TechLabs

git push --force

Do NOT use (UNLESS YOU KNOW EXACTLY WHAT YOU ARE DOING)

- Checks:
  - What branch am I in? -> If its main don`t use it
  - What branch do I want to push -> If its master don`t use it
  - What is the current status of the remote branch -> Did other people use it? -> don`t use it
- Only use if you are 110% sure that you are pushing to a non- main branch, and that no one else works on that branch!
- YOU CAN DELETE A LOT OF PROGRESS AND YOU CAN MAKE YOUR TEAM MEMBERS VERY VERY ANGRY!

>. TechLabs

A gitignore file specifies intentionally untracked files that Git should ignore. Files already tracked by Git are not affected; see the NOTES below for details.

No need to write your own: Just choose the right one from here
- https://github.com/github/gitignore

# Git How to repair

Tasks (if we have time)

>. TechLabs

Everyone in the team needs to do these tasks, but communicate with each other about your questions and learnings!

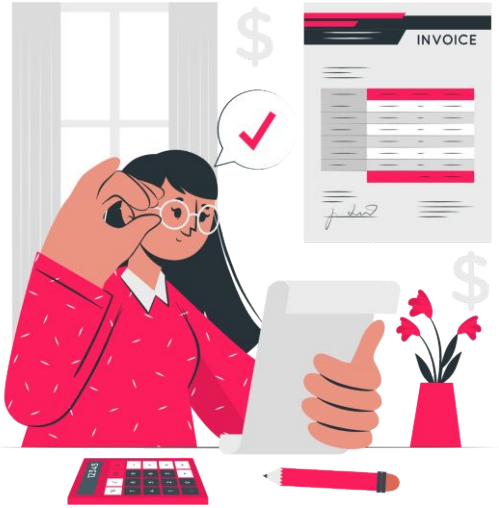
1. Commit wrong things to your coworker's repo (No review this time push straight to main).
2. Remove the wrong stuff from your project.
3. Add a useful .gitignore to your repo.

20 min

>. TechLabs

## Questions?

Look at the cheat sheet and ask your group members



>. TechLabs

# How was it?