anouri GmbH
Software & IT-Consulting

Training

Git Grundlagen

anouri GmbH
Software & IT-Consulting

Manuel Pieke

Geschäftsführer
anouri GmbH

SharePoint Architect
Software Architect/Developer
Scrum Master
ALM Consultant

M: +49 (0) 151 175 89 052
E: manuel.pieke@anouri.gmbh

What we want to do today

- What is a Version Control System?
- What is git?
- Thinking in git
- States behind the scenes
- Introduction to commmon git commands
- Prepare before we start
- Starting with the basics (Add/Commit)
- Branching and Merging
- .gitignore
- Working with other developers
- Merging
- Resolving conflicts
- Git flow
- Submodules
- LFS

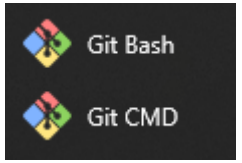
Before we start

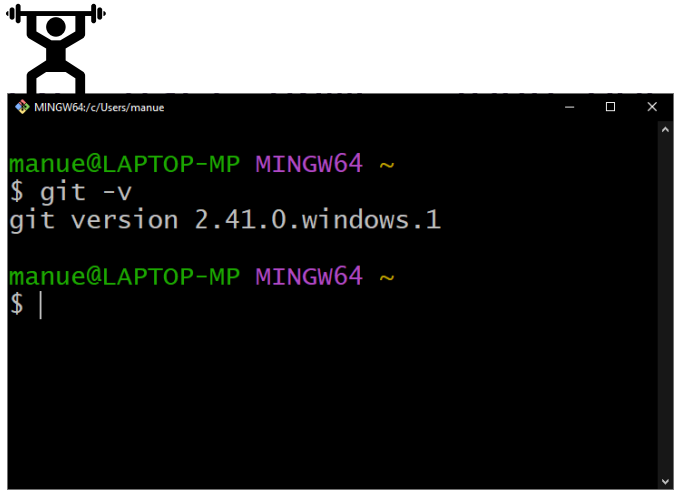
Has everyone installed git on their machines?

Open Console (Bash or CMD) and type:

```
› git --version / git -v
```

```
Version > 2.23
```





Cmd vs Bash

- Git Bash (Press q to quit listing) / Git CMD (windows)
- Basic Windows CMD- / Bash-Commands:
  - cd / cd: Change the current directory to another folder on your computer
  - dir / ls: Display the contents of the current directory
  - mkdir / mkdir: Create a new directory
  - ren / mv: Rename a file or folder
  - copy / cp: Copy a file from one location to another
  - xcopy / cp -r: Copy a folder and its contents from one location to another
  - del / rm: Delete a file
  - rd / rmdir: Remove an empty directory

Exercise

- Everything you should do as an exersize is marked with the icon:

We do not use a git GUI around here!

- GUI can be helpful in some situations
- But to really understand git, use CLI

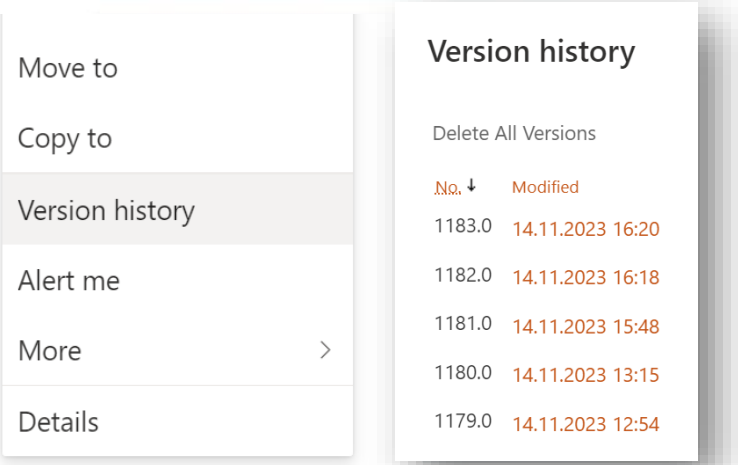
What is a version control system?

A version control system is a software tool that helps manage changes to files, especially source code, over time.

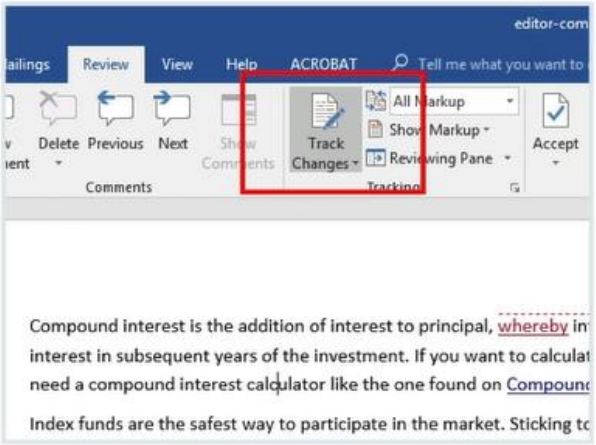
It tracks and records the modifications made by different developers and assigns a revision number or code to each change.

It also allows developers to work in parallel, communicate efficiently, and recall previous versions of the files if needed

Types of version control (local)



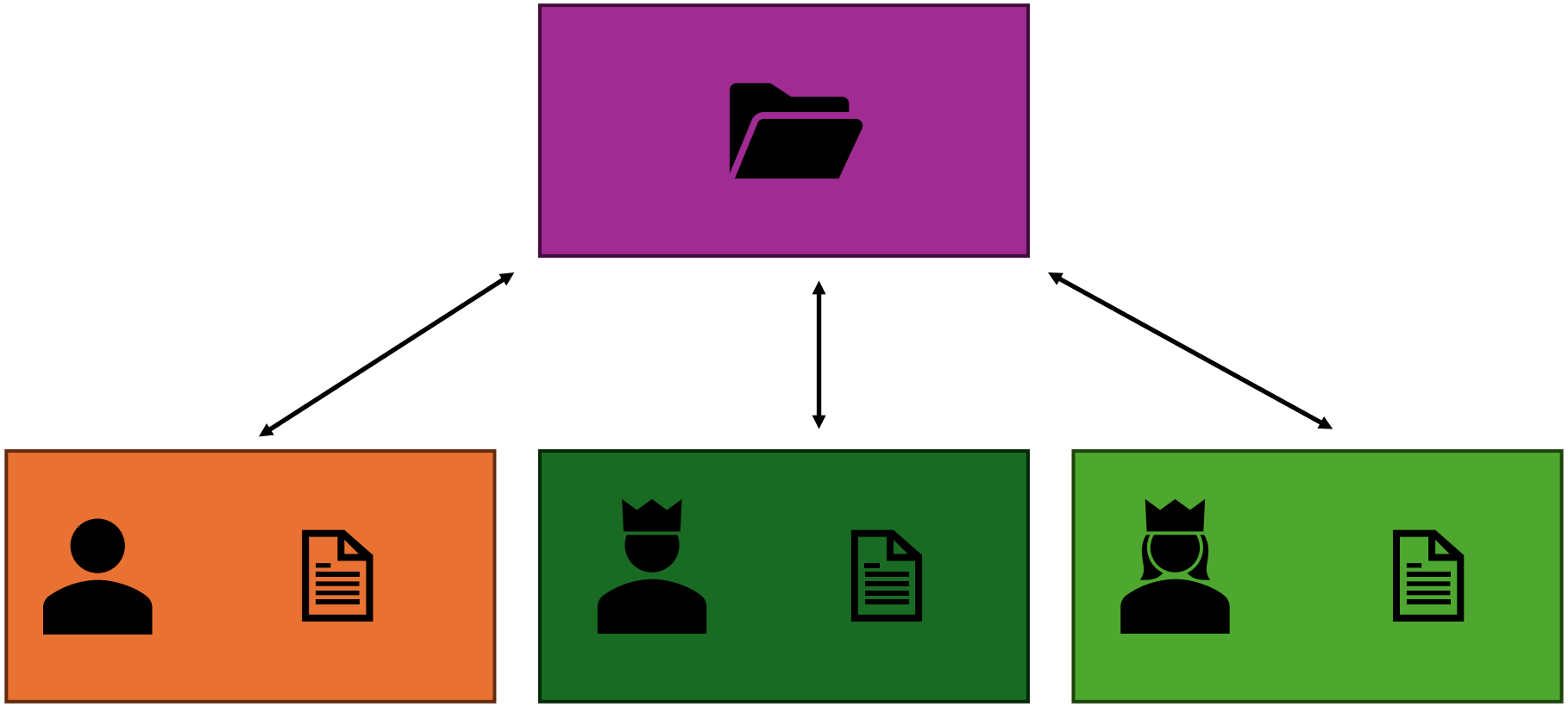
Version Control



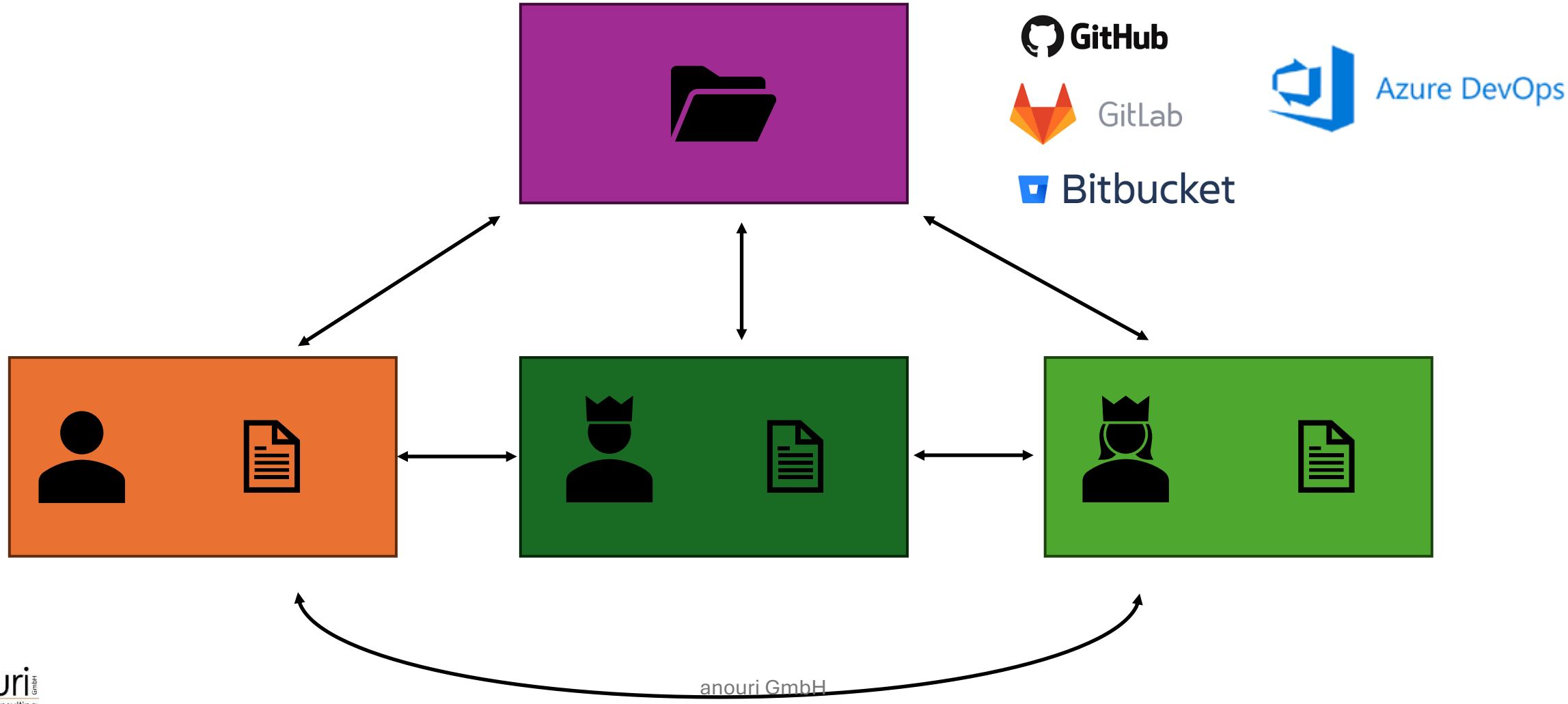
Version Control

Version Control

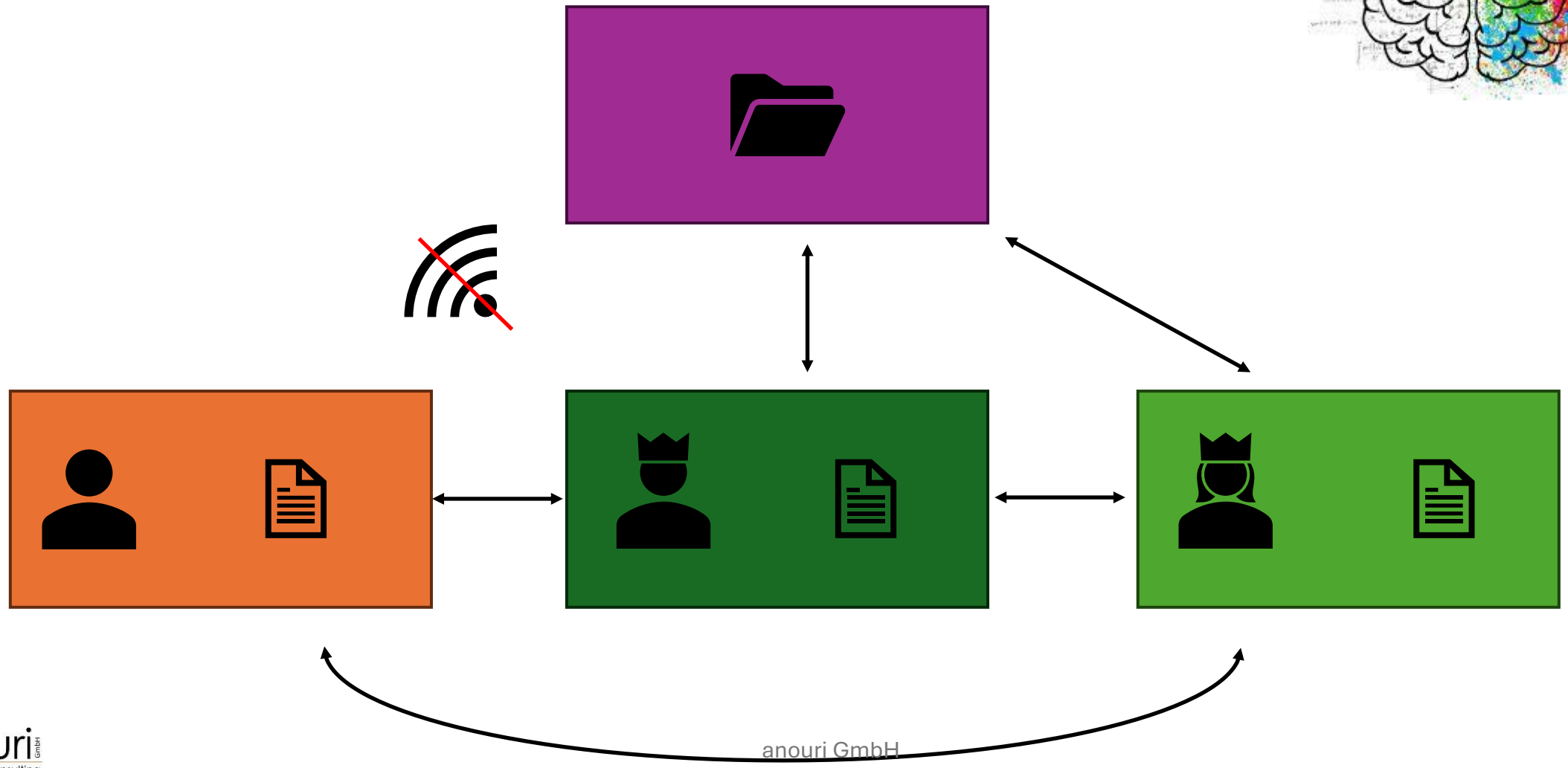
Centralized Version Control System



Distributed Version Control System



Designed for local working



When to use a Version Control System?

- Keep track of changes (bookkeeping, certification, process)
- Version your code (and artifacts*)
- Collaboration with multiple developers
- Go back in time (restore a previous version of the code)
- Keep it safe (backup)

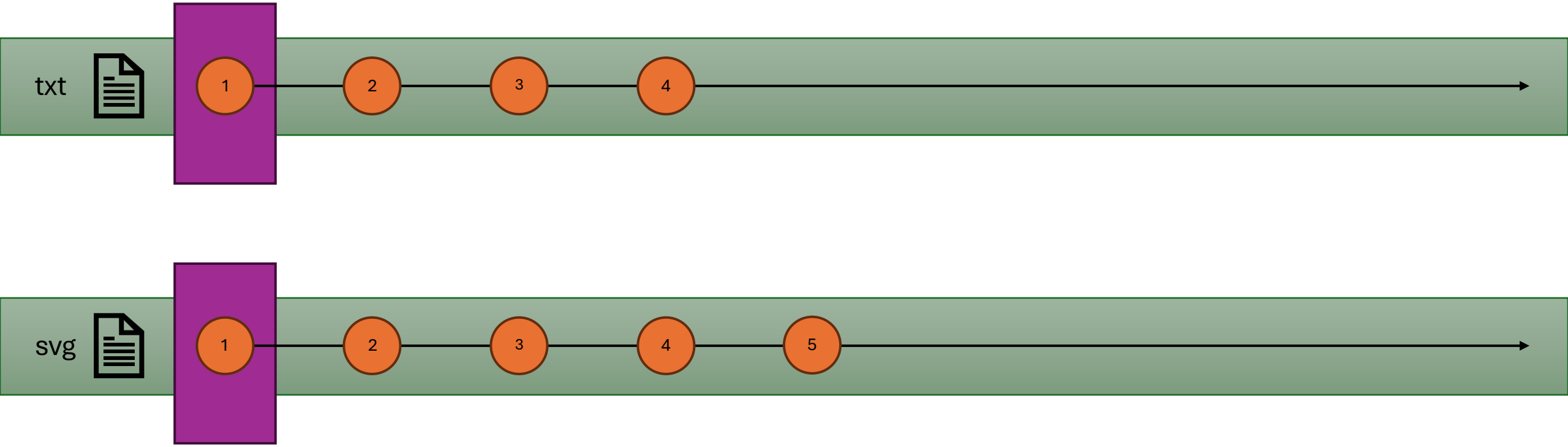
When NOT to use a Version Control System?

- Binary/larger files (LFS extension mentioned later)
- Dump of temporary files
- Extension of your local file system
- Backup system for documents
- Different use case → Dropbox/iCloud

What is git?

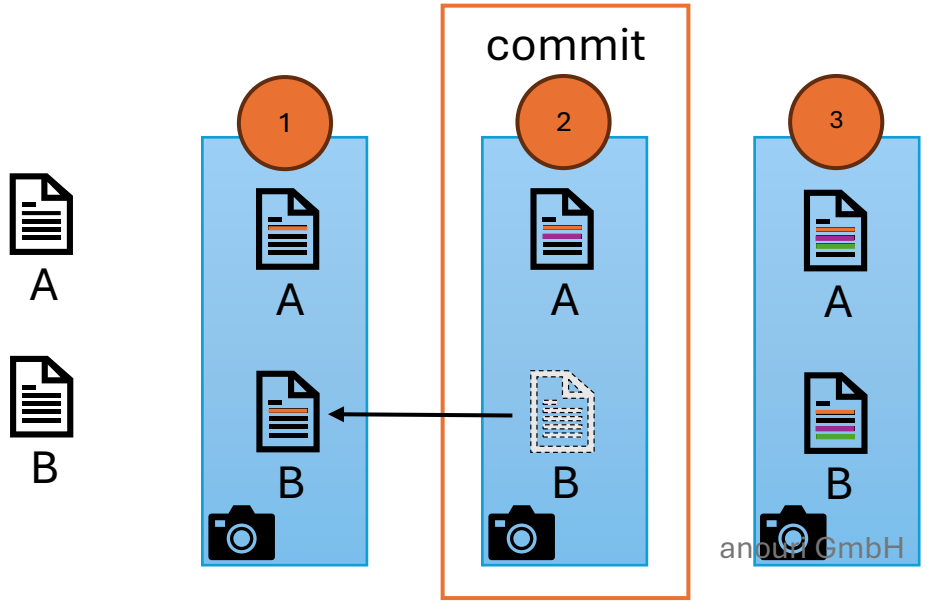
- Git is a (free and open source) distributed **v**ersion **c**ontrol **s**ystem that tracks changes in any set of computer files.
- It is designed to handle everything from small to very large projects with speed and efficiency.
- It is usually used for coordinating work among programmers collaboratively developing source code during software development.
- It was developed by Linus Torvalds in 2005.

Git – the time machine



Git-Introduction – thinking in git

- Other VCS
  - Stores files and their changes as delta for every file and version
- Git
  - Stores a snapshot of every file, history and metadata for any version at that specific point in time



Git-Introduction – thinking in git

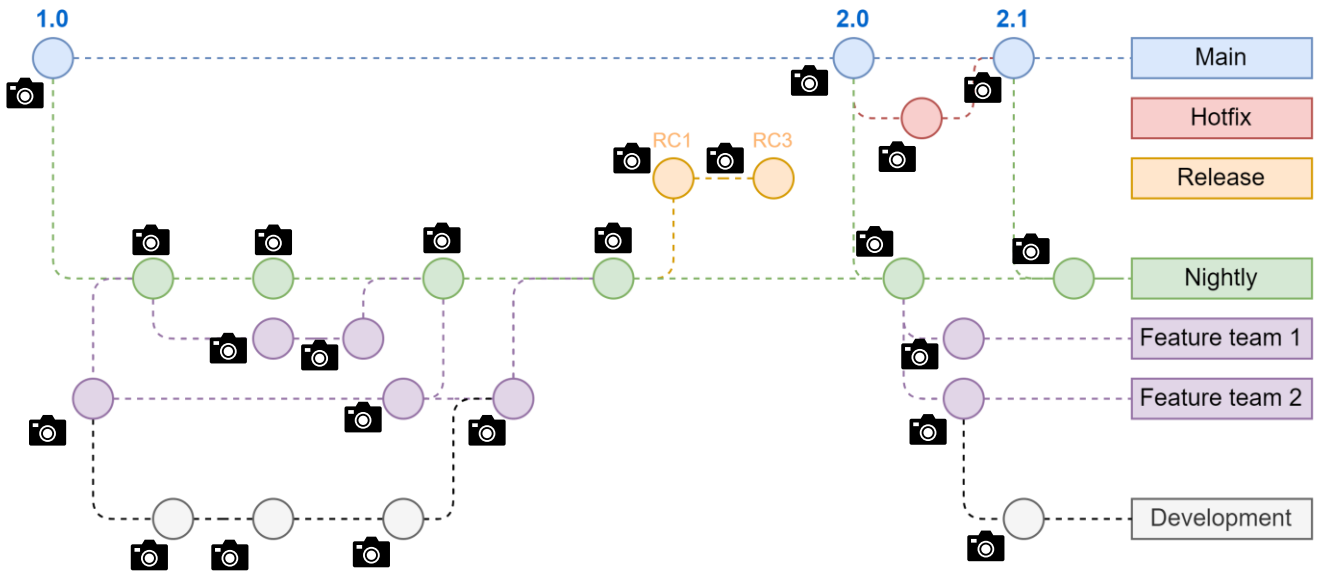
- Git does not do anything without you telling it to!

- Everything you do is done by you!
- What, When, How – you name it!

- Seems exausting, but you will love the freedom coming with that.

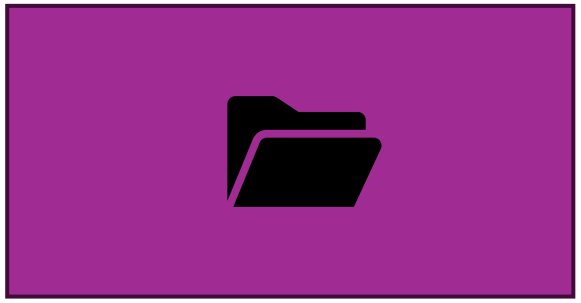
Git-Introduction – thinking in git

- Git is Designed for Non-Linear Development - Branching



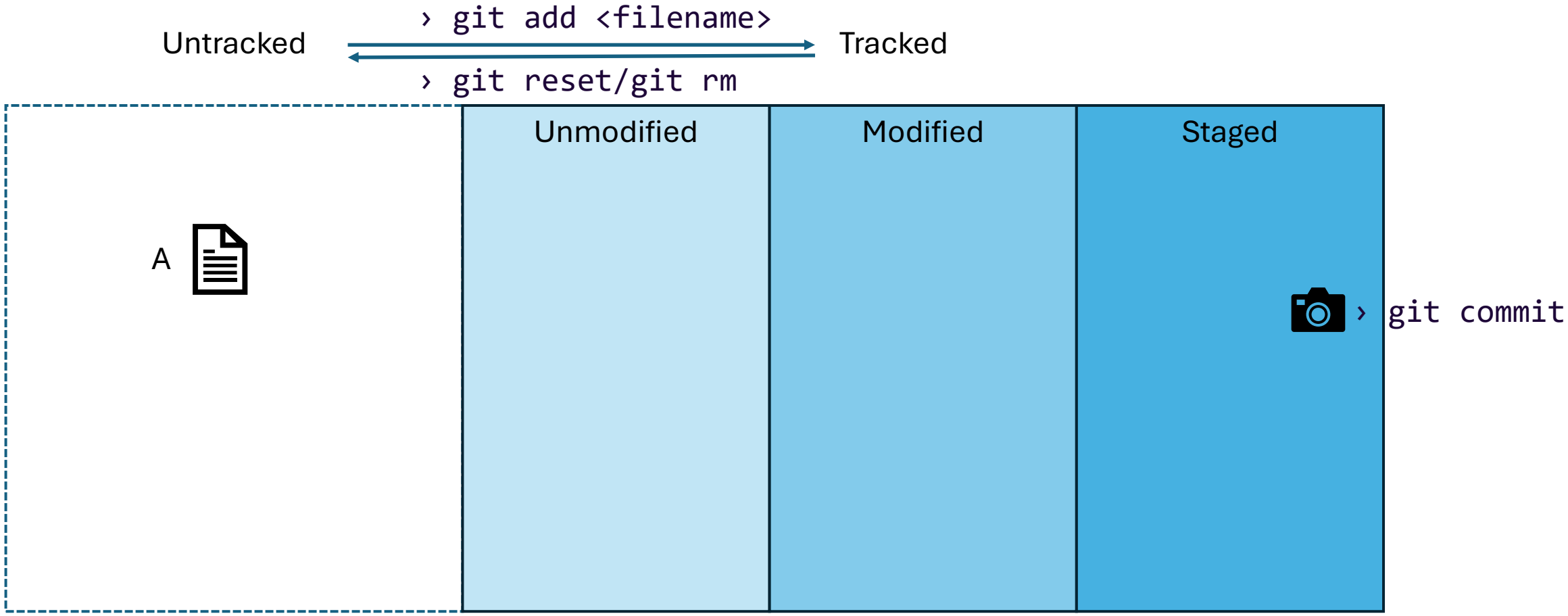
Git repository

- Also referred as „repo“ or “git project“



```
git init
```

States files can be in



Git „Basic“ Commands

- `git init` Create new .git repository and begin tracking
- `git add` Move modified files into the staging area (add to tracking)
- `git rm` removes fils from tracking
- `git status` Shows the status of the files
- `git commit` Create a snapshot and commit to .git
- `git config` Set and replace git configurations
- `git log` Shows the committed snapshot history
- `git show` Shows details to the last commit (object)
- `git diff` Shows changes between your working dir. and staging area

Git „Branches“ Commands

- `git branch` List, create, or delete branches
- `git checkout` Switch between branches (aka: `git switch`)
- `git merge` Move changes from one branch to another

Git „Repositories“ Commands

- `git clone` Copies entire repo into local .git directory
- `git remote` Create and show linked repos
- `git push` Send updates to associated repos
- `git pull` Retrieve and integrate changes from other repos
- `git fetch` Retreive but do not integrate changes from other repos

Git „Undoing“ Commands

- `git revert` Create a new commit which is undoing the previous commit (safe command)
- `git reset` Removes files from staging area (warning!)

Help-Command

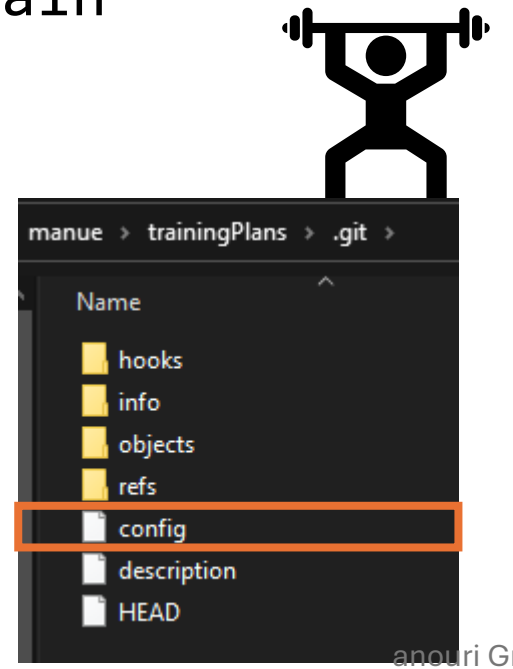
- `git <command> --help`
- `git add –-help`
- `git commit –-help`

5 min – Get familiar with your console and execute some help commands

git help internet page: https://git-scm.com

Create new project

```
> mkdir trainingPlans_A
> cd trainingPlans_A
> git init
> git branch -m main
> Dir / ls -a
> git status
```



Texteditor

- All Text editors can be used.
- I prefer Visual Studio Code
- Code . → opens VS Code

Git config levels

`--system` (on system level, for all users on the pc)

`--global` (applies to all repos of a user)

`--local` (specific to a single repo)

Git needs user and mail

```
> git config --global --list
> git config --global user.name "John Doe"
> git config --global user.email john@doe.org
> git config --global --list
> git config --global init.defaultBranch main
> git config --global core.editor "code --wait --new-window"

> git config --global color.ui auto

> git config --global --unset user.name
```

Other configuration

Windows uses a different line ending (\r\n) compared to UNIX systems (\n). This can cause problems when sharing files between different systems.

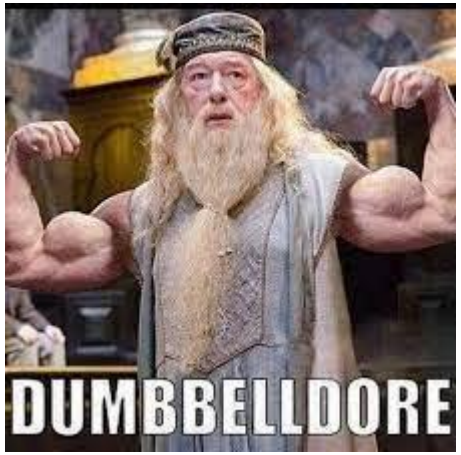
```
git config core.eol lf
# Configures to use UNIX line endings

git config core.eol crlf
# Configures to use DOS line endings
```

While we are at it: setting of encoding is done in .gitattributes:

```
echo '*.html encoding=utf-8' >> .gitattributes
```

Git - gitattributes Documentation (git-scm.com)

Our Szenario for exersizes



- Lets assume I am a fitness gym trainer
- Create training plans for different goals:
  - Cardio
  - Increase strength and mass
  - Loose fat
  - Define muscles
- And different body parts:
  - Legs, Biceps, Chest, Triceps, Back, Shoulders

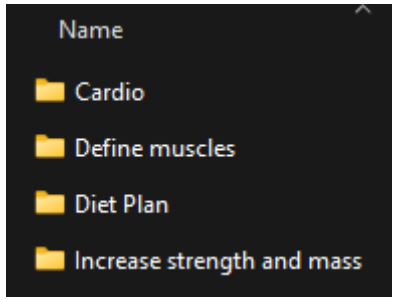
Create Folders

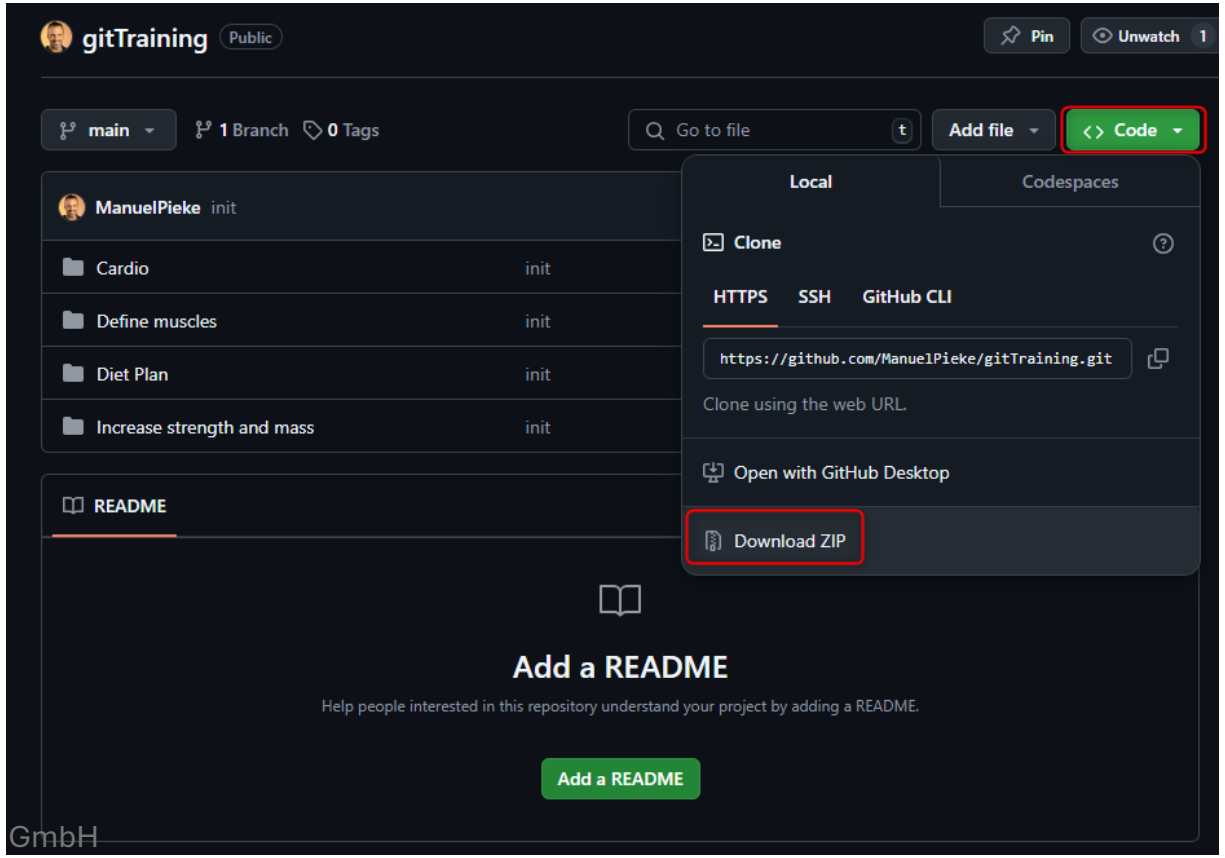
Download Zip from:

https://github.com/ManuelPieke/gitTraining

Download PDFs seperately

```
> git init
```





Git add

```
> git add .
```
(adds everything, but be careful)

```
> git status
```

But we do not want this. So reset and try again.

```
> git reset
> git status
```

Be aware of casing

Only add one folder:

```
> git add "Cardio"
> git commit -m "add files Cardio"
```

Then add the rest by yourself.

Undo changes

- Unstage changes with `git reset <staged file>`

- `git rm --cached <file>` (unstages file without deleting it)
- `git rm --force <file>` (deletes the file for good)

- `git restore --staged <file>` (discards changes)

- Undo working directory changes (loose content) with `git restore <file>`

- Great explanation:
- https://linuxhint.com/difference-between-git-rm-cached-and-git-reset-file/

Commit ID and added files

- Commit ID is unique (40 char ID, but only 7 are mostly enough)
- Added files are shown

```
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Cardio/cycling.md
        new file:   Cardio/jogging.md
        new file:   Define muscles/biceps.md
        new file:   Define muscles/breast.md
        new file:   Diet Plan/dietplan-week01.md
        new file:   Increase strength and mass/max-biceps.md


manue@LAPTOP-MP MINGW64 /d/Quellcode/cariad/trainingPlansV1 (main)
$ git commit
[main (root-commit) 0b78c8b] Add Cardio, define muscles, diet plan
and inclrease strength
 6 files changed, 101 insertions(+)
 create mode 100644 Cardio/cycling.md
 create mode 100644 Cardio/jogging.md
 create mode 100644 Define muscles/biceps.md
 create mode 100644 Define muscles/breast.md
 create mode 100644 Diet Plan/dietplan-week01.md
 create mode 100644 Increase strength and mass/max-biceps.md
```

Make changes

Open Text Editor („code .“ for VS Code)

Edit file biceps.md, add title and save.

```
> git status
> git add <filename>
> git status
> git commit –m 'update title for biceps training'
```

→ Try this with the chest file as well.

Notes on commits

Commit quite often.

Avoid mixing whitespace changes with functional code changes.

Avoid mixing two unrelated functional changes.

Avoid sending large new features in a single giant commit.

If a code change can be split into a sequence of patches/commits, then it should be split. Less is not more. More is more.

Important: Always commit working code!

Note on commit messages

A properly formed git commit subject line should always be able to complete the following sentence:

If applied, this commit will <your subject line here>

Describe why a change is being made.
The first commit line is the most important.
Do not assume the reviewer understands what the original problem was.
Do not assume the code is self-evident/self-documenting.
Describe any limitations of the current code.
...

Check out these links for more information about good commits:
https://gist.github.com/robertpainsi/b632364184e70900af4ab688decf6f53
https://wiki.openstack.org/wiki/GitCommitMessages#Git_Commit_Good_Practice

Branching

```
> git branch feature
```
(Creates new feature branch)

```
> git status
> git branch -a
```
(Shows current branches)

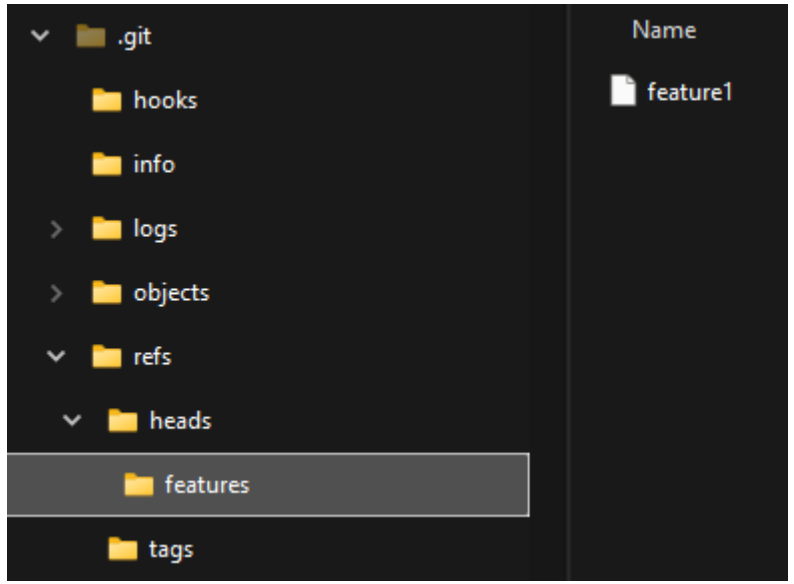
```
> git checkout feature
> git branch -a
```
(Shows current branches)

Also possible:

```
> git branch features/feature1
```

Shortcut:

```
> git checkout -b feature
```
(Creates branch and checks it out)



What is HEAD?

HEAD is a pointer to where you are.

Check it out:
https://a-a-ron.github.io/visualizing-git/#free

`git reset HEAD~` (keep changes)
`git reset --hard HEAD~` (discard changes)

```
git reset HEAD~2
```

Good explanation:
https://www.becomebetterprogrammer.com/git-head/

Branching

`git branch`: manage branches without changing context

- List branches: `git branch [--remote] [-va] [-a]`
- Create a branch: `git branch <branch-name> [<origin if not current HEAD>]`
- Rename a branch: `git branch -m <old-name> <new-name>`
- Delete a branch (merge check): `git branch -d <branch-name>`
- Delete a branch (force): `git branch -D <branch-name>`

Checkout

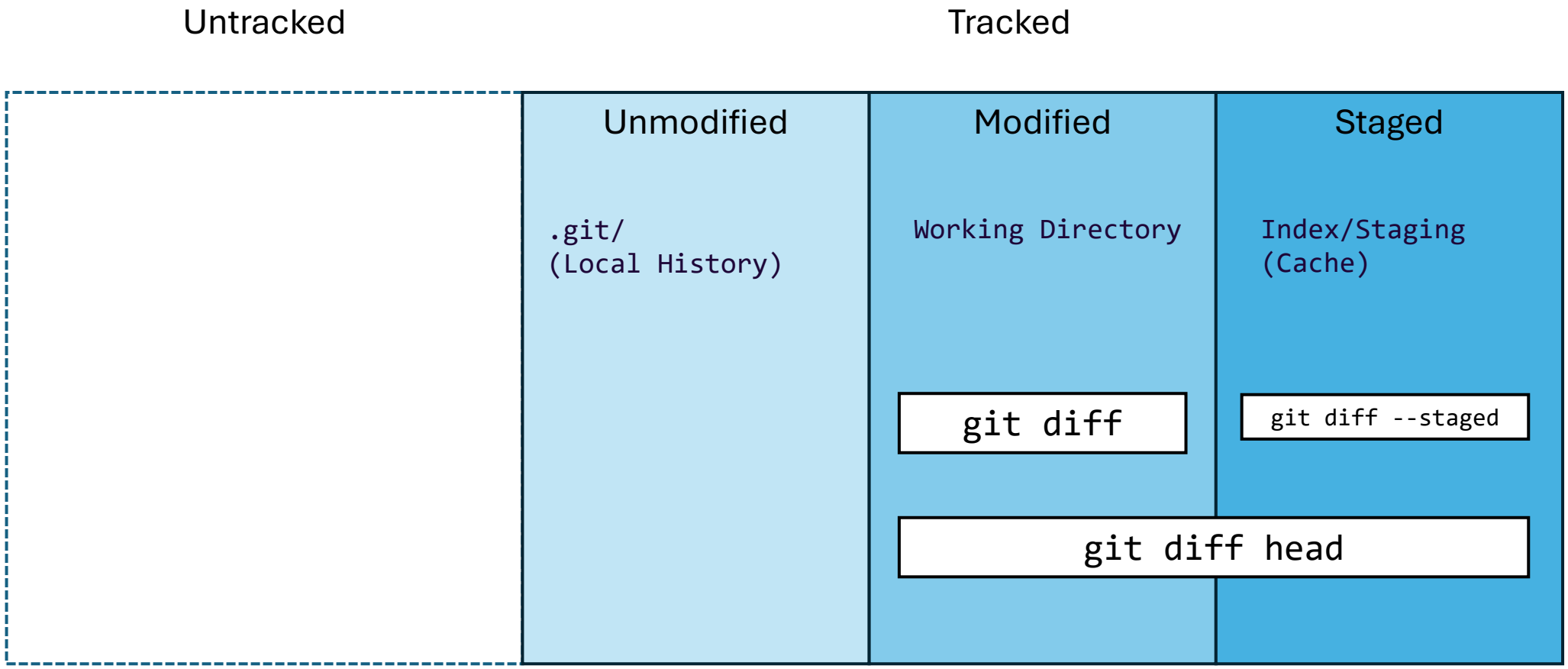
`git checkout`: change context (the working directory), the HEAD

- Switch to a branch: `git checkout <branch-name>`
- Create a branch and switch to it: `git checkout -b <branch-name> [<origin if not current HEAD>]`
- Copy file from a different branch to the current context: `git checkout <branch-name> --<file path>`

Changes

- Do multiple changes on the new feature branch.
- There are commited files with changes and modified but not staged files.
- Now we want to check out the differences of the files.

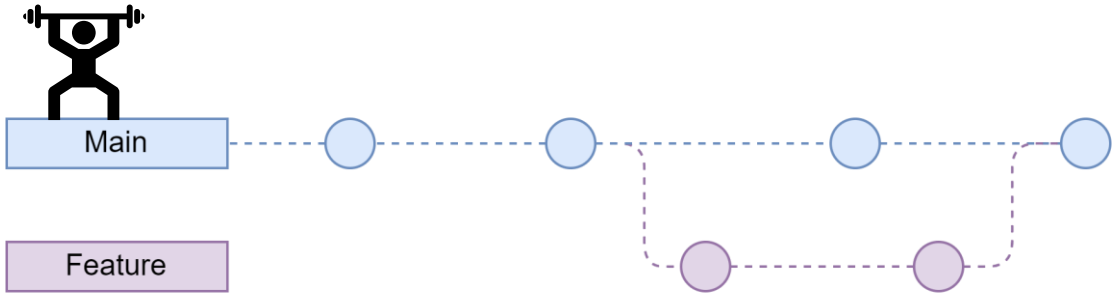
git diff

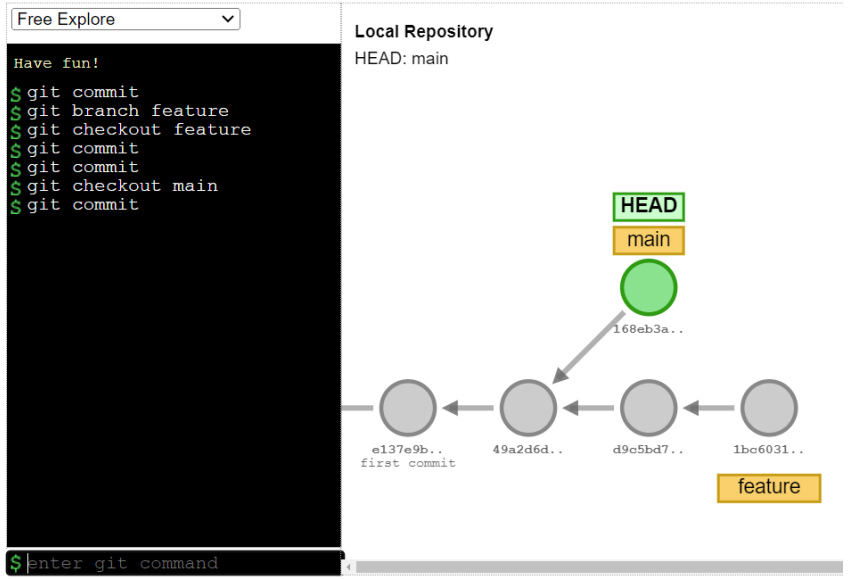

Untracked
Tracked
Unmodified
Modified
Staged
.git/
(Local History)
Working Directory
Index/Staging
(Cache)
git diff
git diff --staged
git diff head

Perpare the merge

- Switch to main branch
- Open Text editor and modify diet-plan-week01.rd file (change food)
- Stage and commit the file
- Be sure to check out the branch you want to merge into:

```
> git checkout main
```



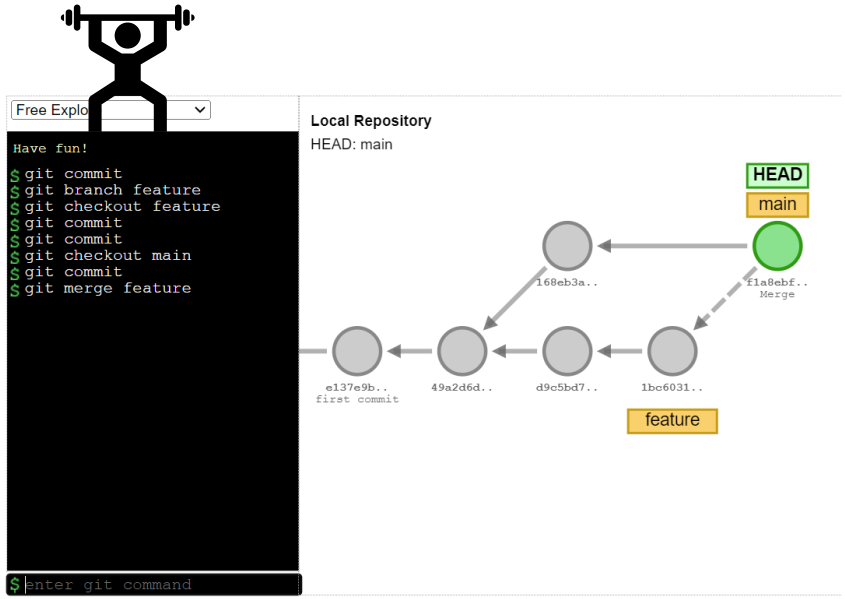


Do the merge

```
> git merge feature -m "Merge feature branch into main"
```

Git decides on the merge strategy it uses. More to that later on.

Try doing it yourself:
https://a-a-ron.github.io/visualizing-git/#free



git log

```
> git log
> git log --oneline
> git log --oneline --graph
> git branch -d feature
> git branch -a
```

- See the log of your local repository running `git log --decorate --graph --oneline`
- Try to use different parameter combination and using `–c, --summary or --name-only`
- Check out: `git log --help`

.gitignore

- A gitignore file specifies intentionally untracked files that Git should ignore. Files already tracked by Git are not affected;

Syntax:

- `path/to/file.txt  # ignore a specific file)`
- `*.txt # ignore all txt files (in the root folder)`
- `path/to/* # ignore all files in path/to`
- `path/to/*.txt # ignore all txt files in path/to`
- `path/**/*.txt # ignore all txt files in all folders under path/to`
- `**/*.pyc # ignore all pyc files everywhere in the repository`

.gitignore

- Create a file named .gitignore in the root project folder and tell it to ignore all .txt files
- Create a “some.txt” file
- Add and commit the .gitignore file
- What happens with the text file?

Little help:

```
path/to/file.txt  # ignore a specific file
```

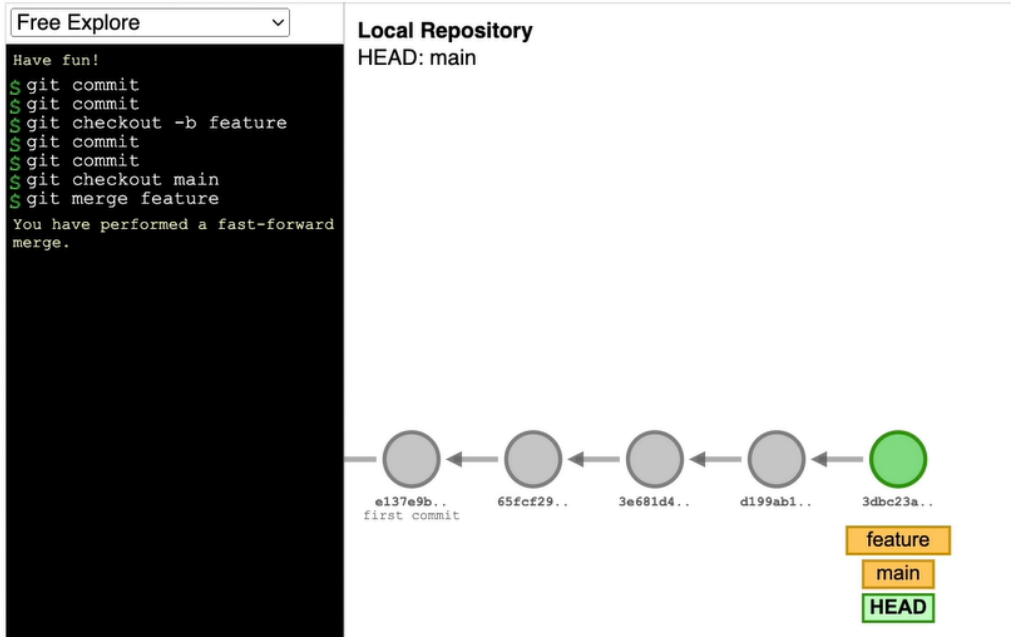
- `*.txt # ignore all txt files (in the root folder)`
- `path/to/* # ignore all files in path/to`
- `path/to/*.txt # ignore all txt files in path/to`
- `path/**/*.txt # ignore all txt files in all folders under path/to`
- `**/*.pyc # ignore all pyc files everywhere in the repository`

Creating a merge

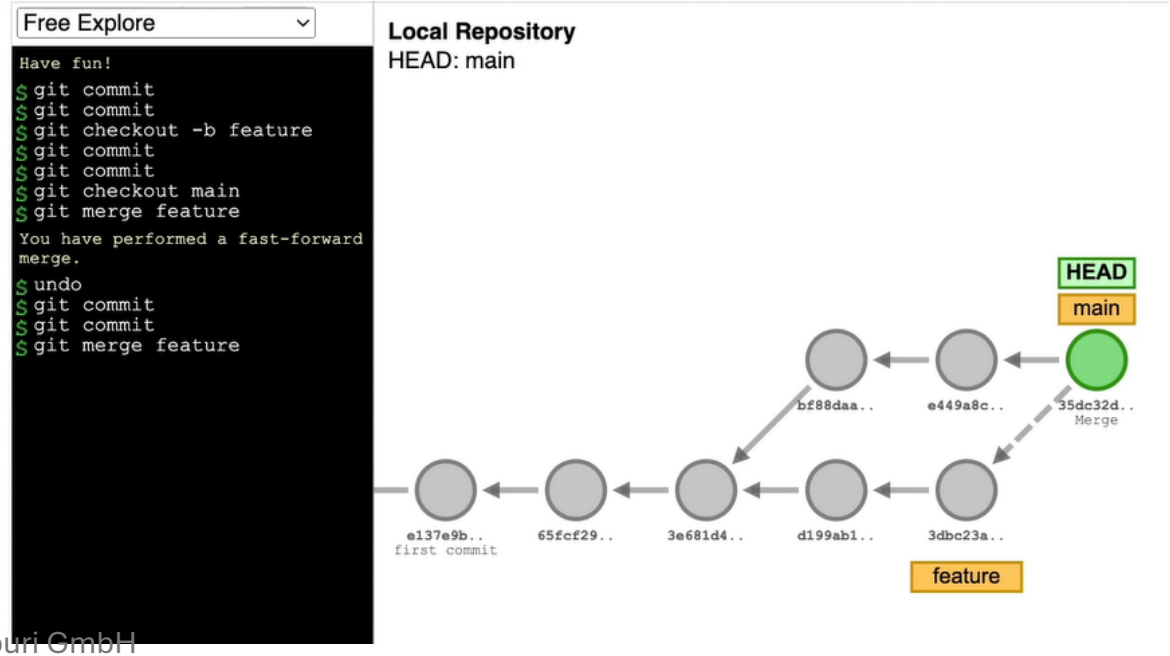
- Create a new branch `firstChange`
- `Do changes to a file`
- `Back to main and create a second branch secondChange`
- `Do changes to the same file in the same line`
- `Back to main`

Merge strategies

- Fast forward (linear history)



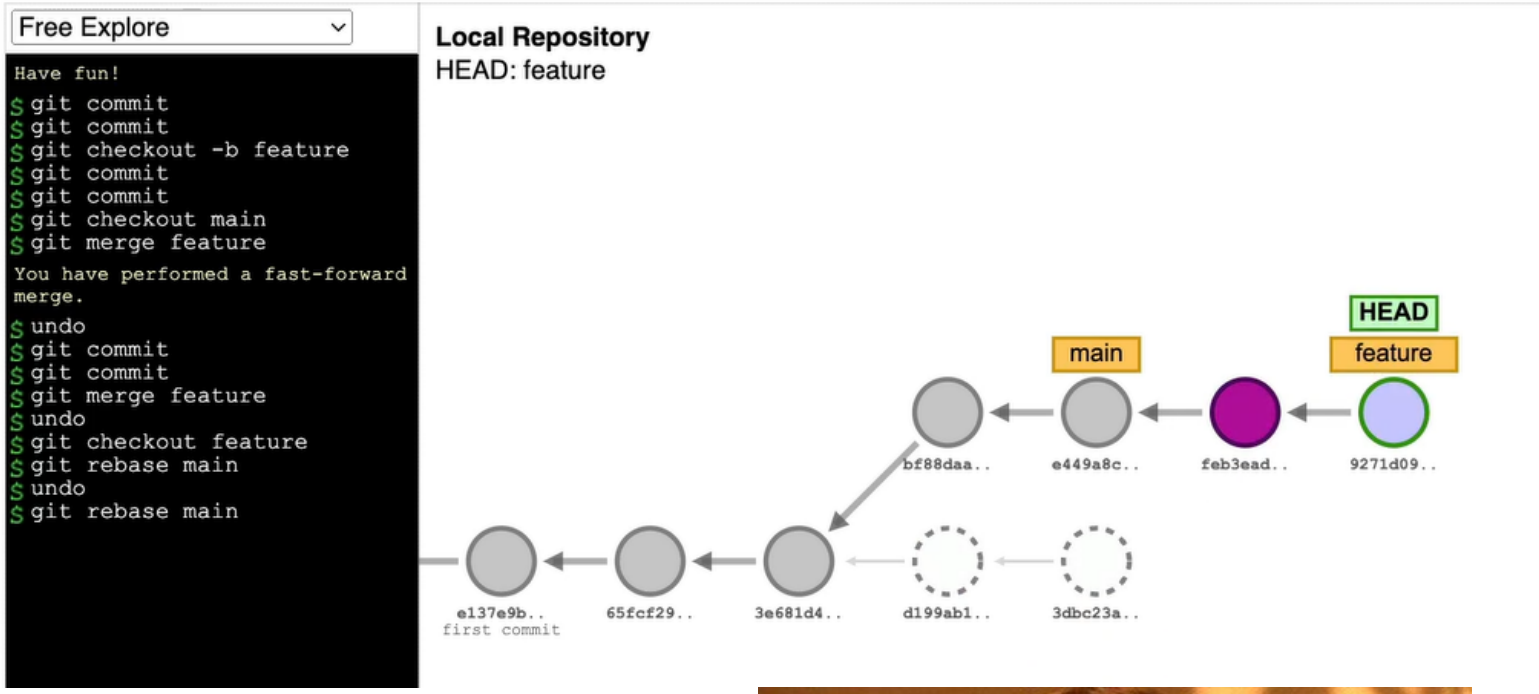
- 3-Way merge



Merge strategies

- Rebase

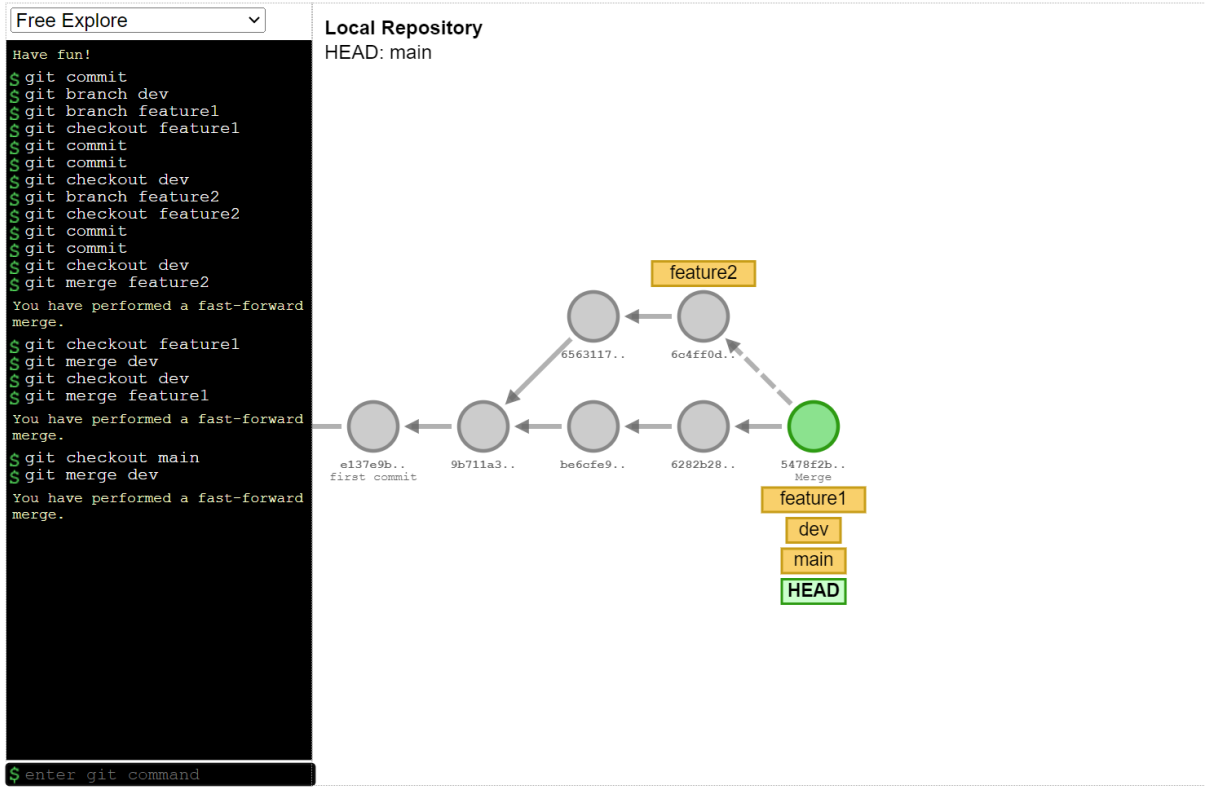
→You get a clean history

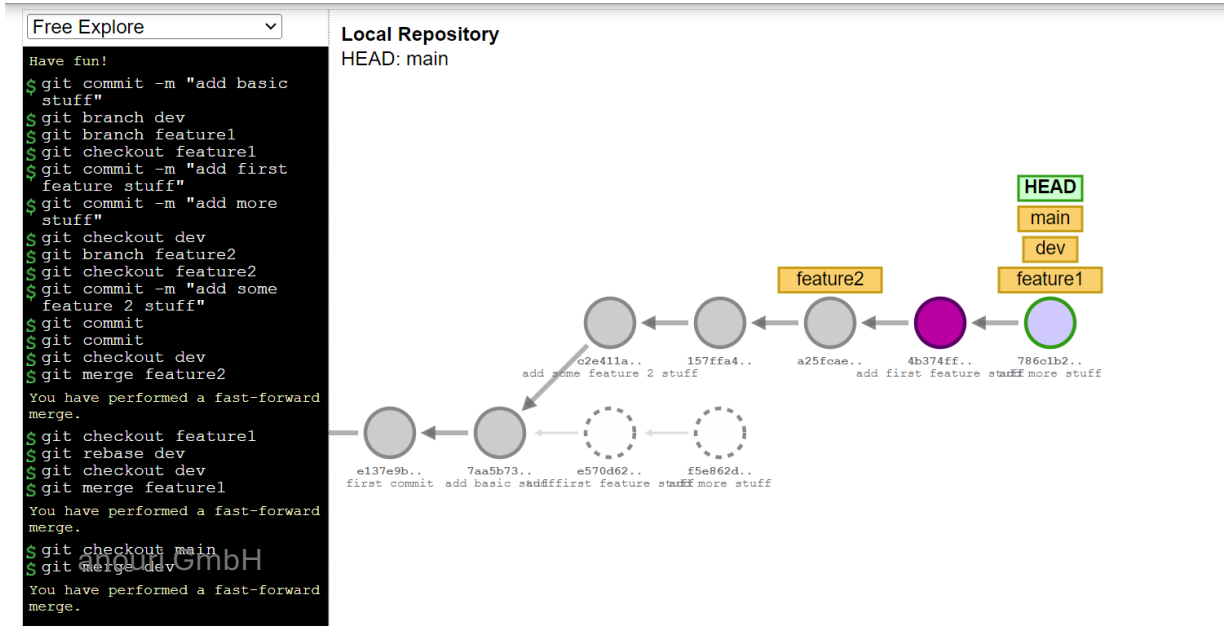


Warning: basically do not rebase shared commits because you will rewrite history.

You need to know what you are doing!

Rebase vs Merge





Merge Conflicts

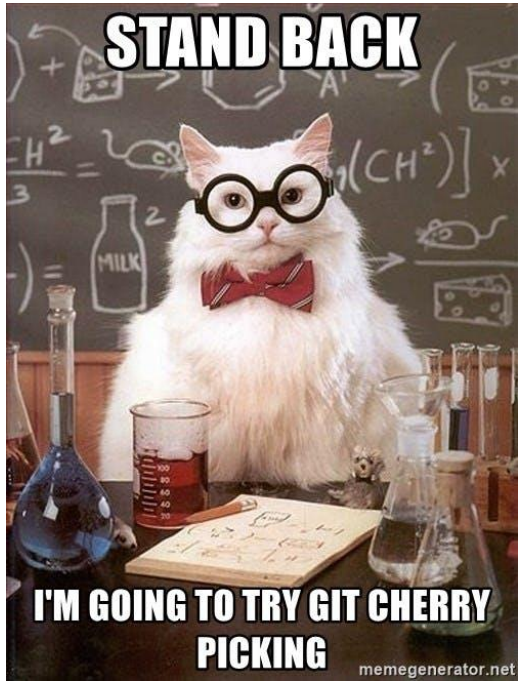
- When more than one person changes the same line in a file and…
- When someone deletes content in a file, but another person edits the same content, …
- When someone deletes a file, but another person edits it, …
- … and they try to merge the change to the same branch

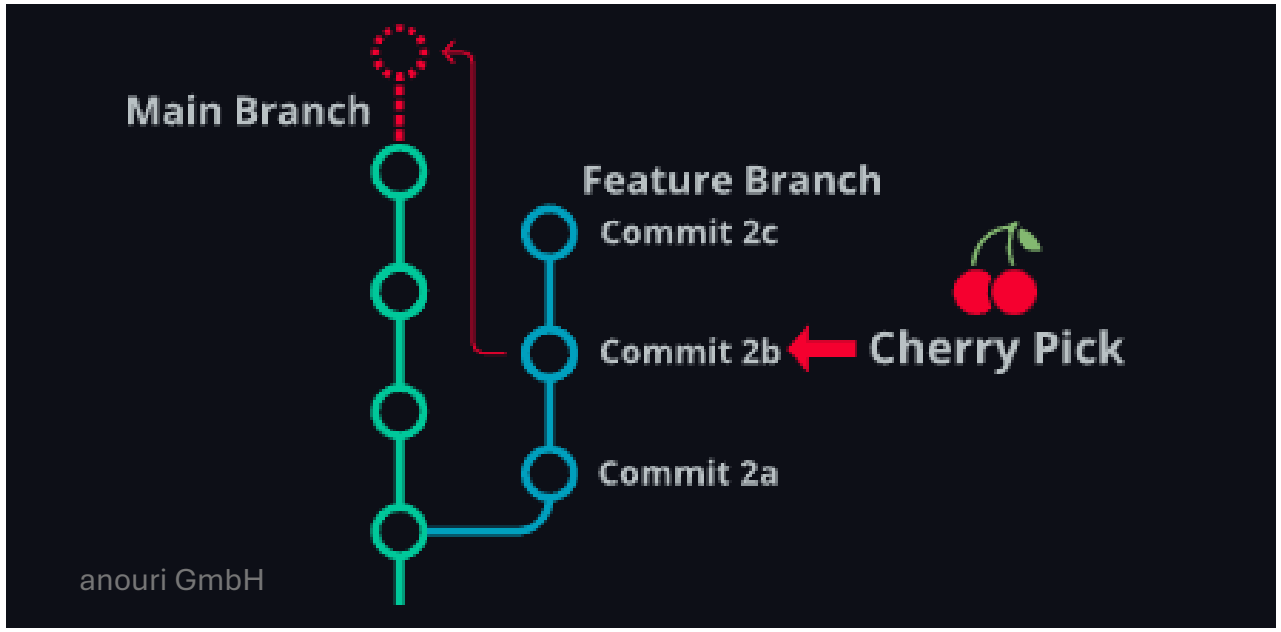
Cherry Picking

Switch to branch where cherry shall be inserted

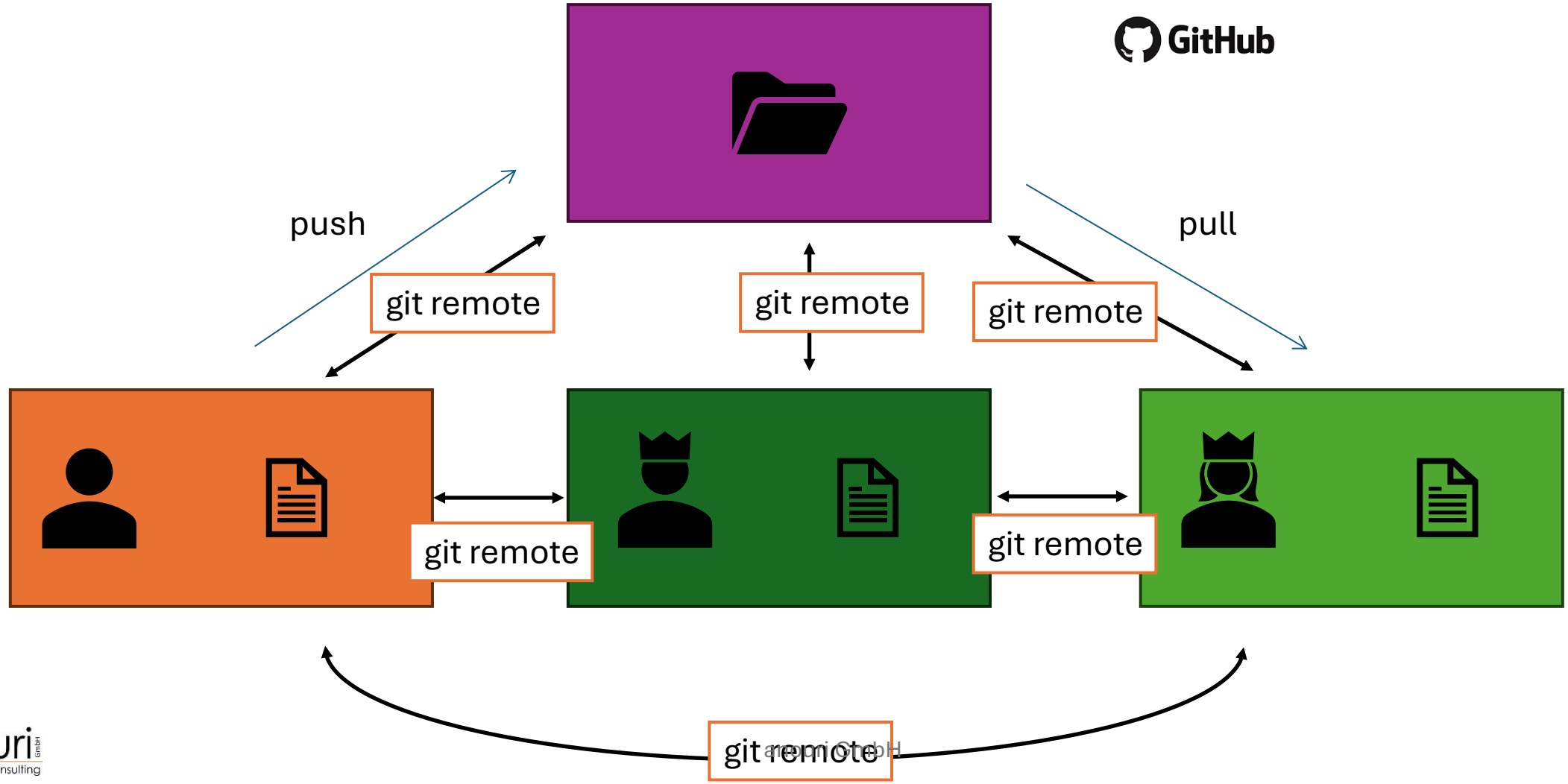
```
> git cherry-pick <shaNumber/branch>
```

Best for Hotfixes in different branches.

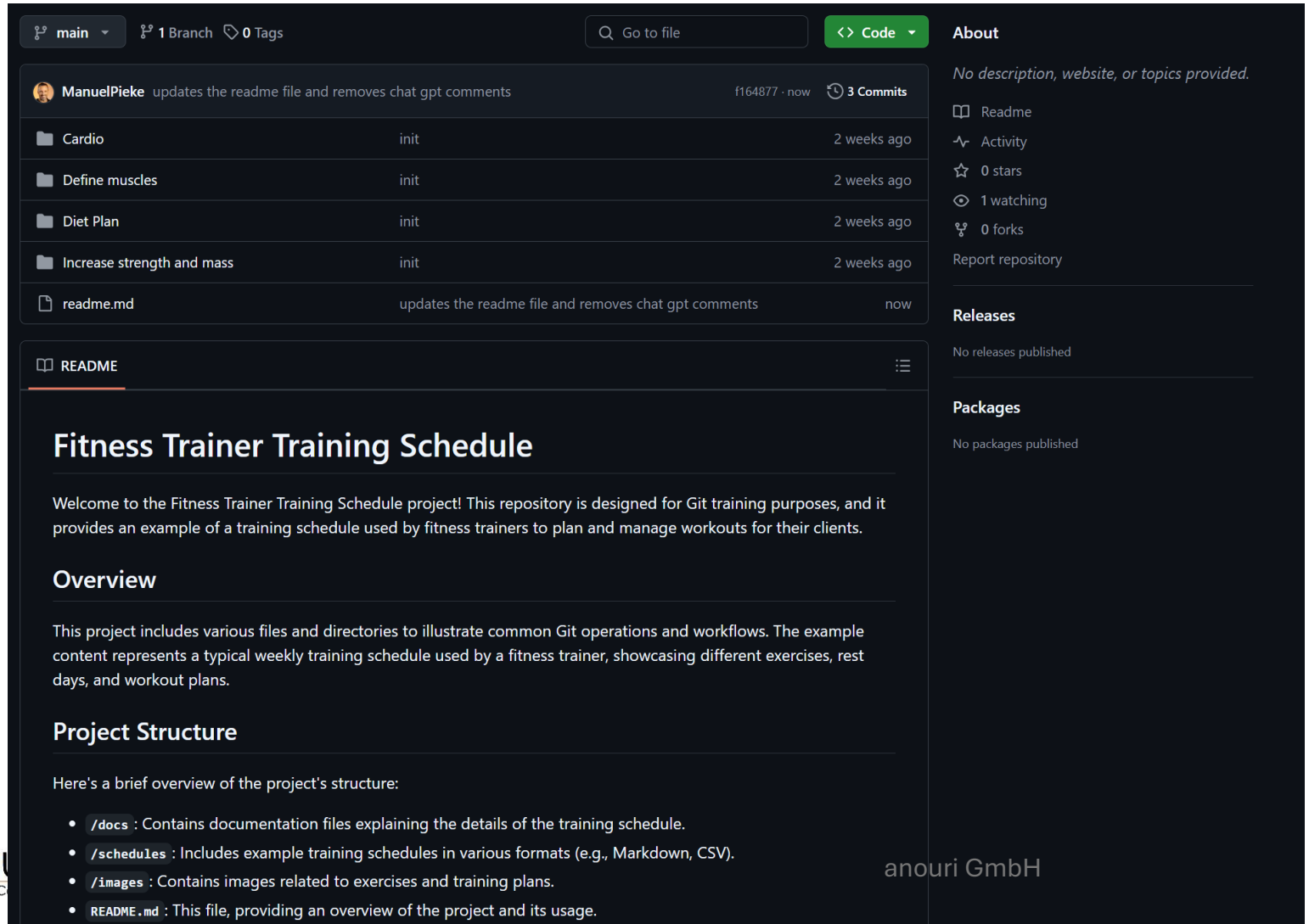




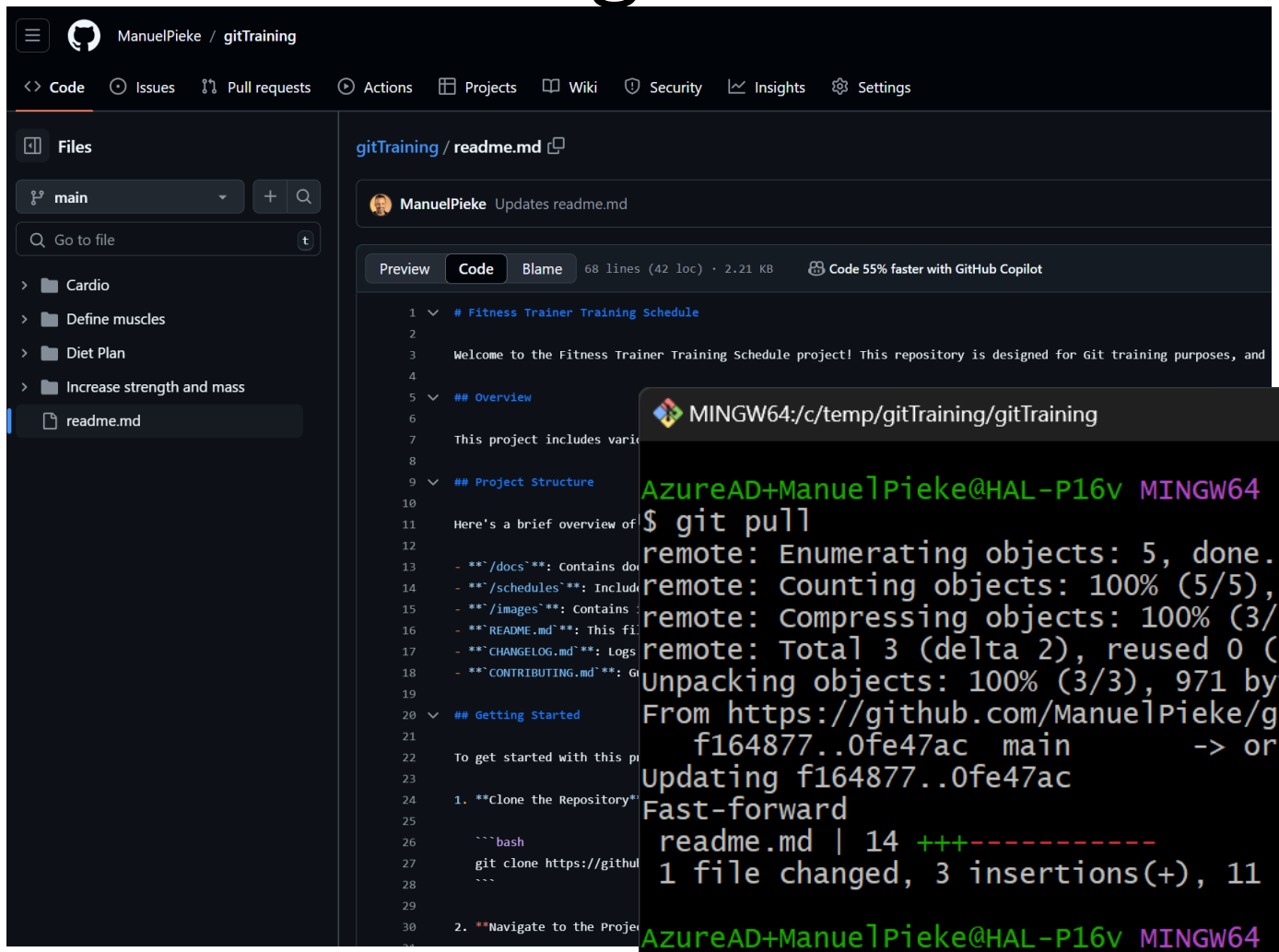
Sharing local work with others

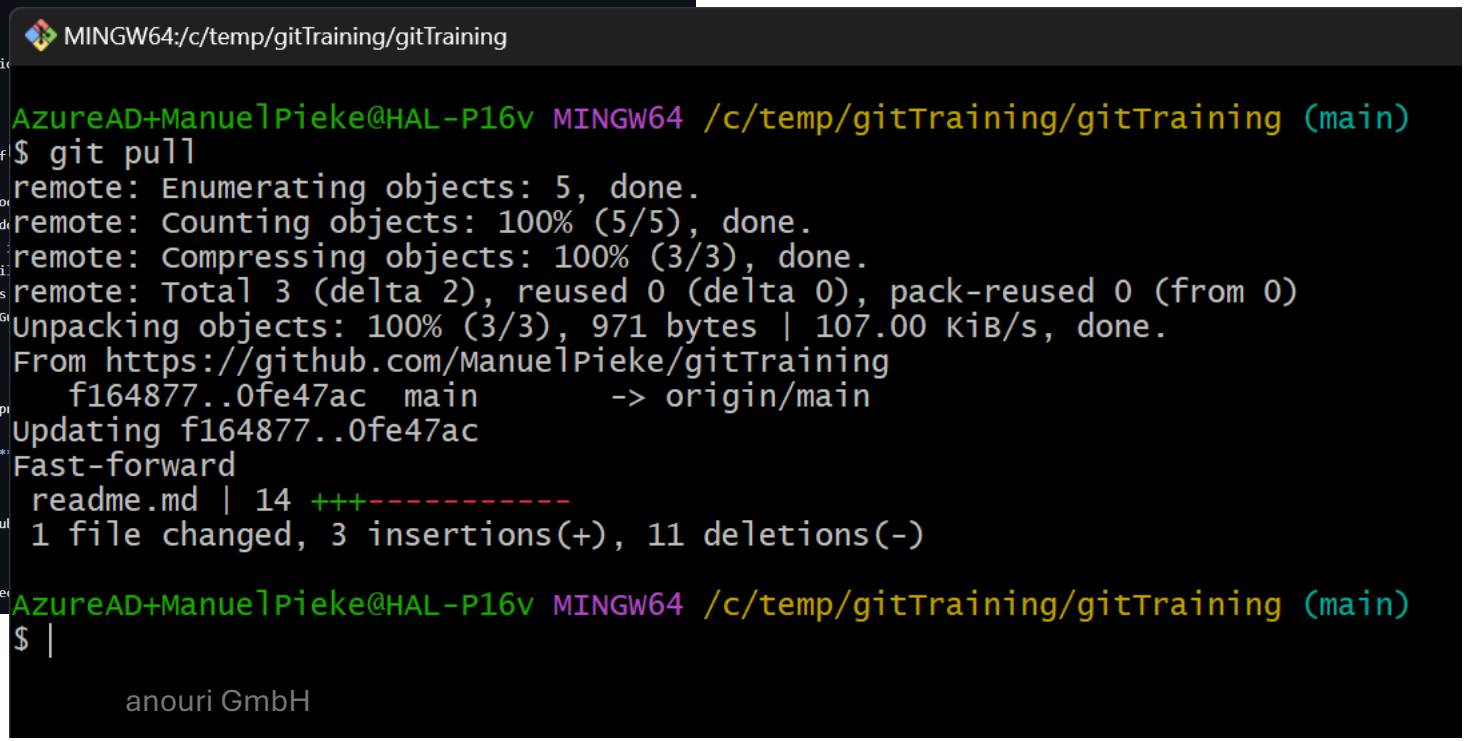


Demo – Push changes to GitHub



Demo – Pull changes from GitHub





Demo – Remote and local updates

```
AzureAD+ManuelPieke@HAL-P16v MINGW64 /c/temp/gitTraining/gitTraining (main)
$ git push
To https://github.com/ManuelPieke/gitTraining.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/ManuelPieke/gitTraining.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

AzureAD+ManuelPieke@HAL-P16v MINGW64 /c/temp/gitTraining/gitTraining (main)
```

Note on git push --force

If you have to force a git push, technically you **can** with the --force option, but procedurally you **shouldn't** because someone may have already pulled and somewhere a kitten will die.

Some Information: [When and Where to Use Git Push — force: A Comprehensive Guide | by Bdbose | Medium]

Create a merge conflict

- `Merge firstChange to main`
- `Merge secondChange to main`

Resolve Merge Conflict

```
AzureAD+ManuelPieke@HAL-P16v MINGW64 /c/temp/gitTraining/gitTraining (main)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 939 bytes | 52.00 KiB/s, done.
From https://github.com/ManuelPieke/gitTraining
   0fe47ac..05ed759  main       -> origin/main
Auto-merging readme.md
CONFLICT (content): Merge conflict in readme.md
Automatic merge failed; fix conflicts and then commit the result.

AzureAD+ManuelPieke@HAL-P16v MINGW64 /c/temp/gitTraining/gitTraining (main|MERGING)
```

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
# Fitness Training Plans
=======
# Fitness Trainer Training Plans
>>>>>>> 05ed759b035ba591934d4e3d165afb523eb26b1d (Incoming Change)

Welcome to the Fitness Trainer Training Schedule project! This repository is designed fo

## Overview
```

readme.md

C: > temp > gitTraining > gitTraining > readme.md > # Fitness Training Plans

```
1  # Fitness Training Plans
2
3  Welcome to the Fitness Trainer Training Schedule project! This repository i
4
5  ## Overview
6
```

```
> git add .
> git commit …
> git push
```

Reduce Conflicts

- Use standard formatting throughout the whole team
- Do small and direct changes and merge frequently
- Talk to eachother to reduce simultaneously working on files

Simple Git Workflow

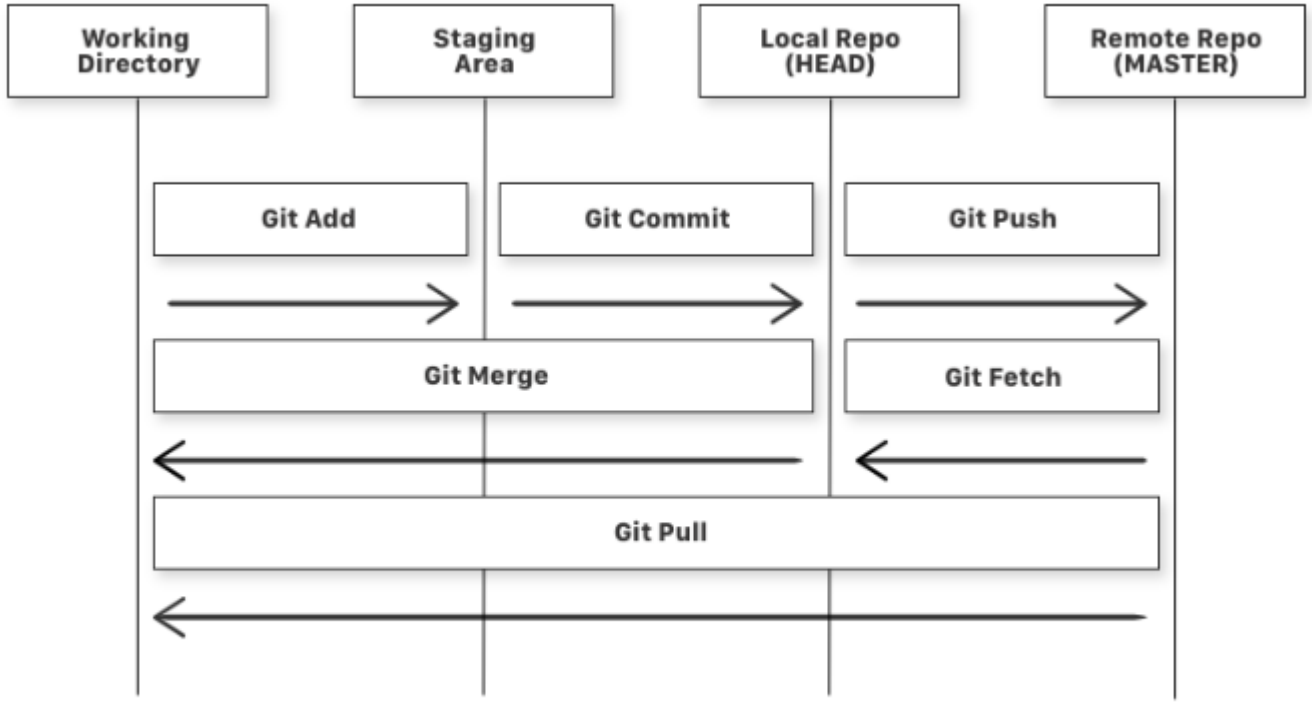


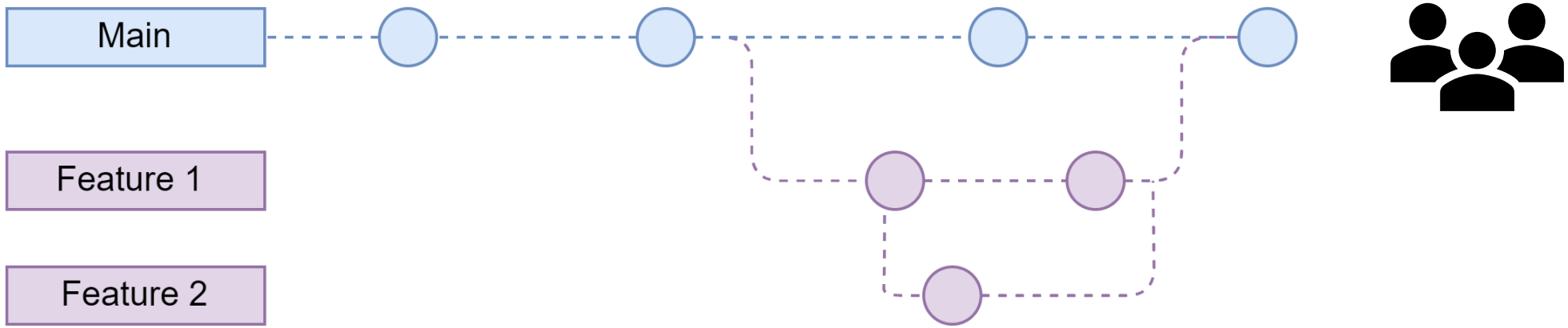
Diagram of a simple Git Workflow

Basic Flow

- For small projects – perhaps only personal projects:

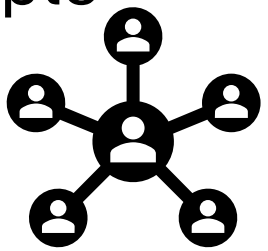
Feature Branch Flow

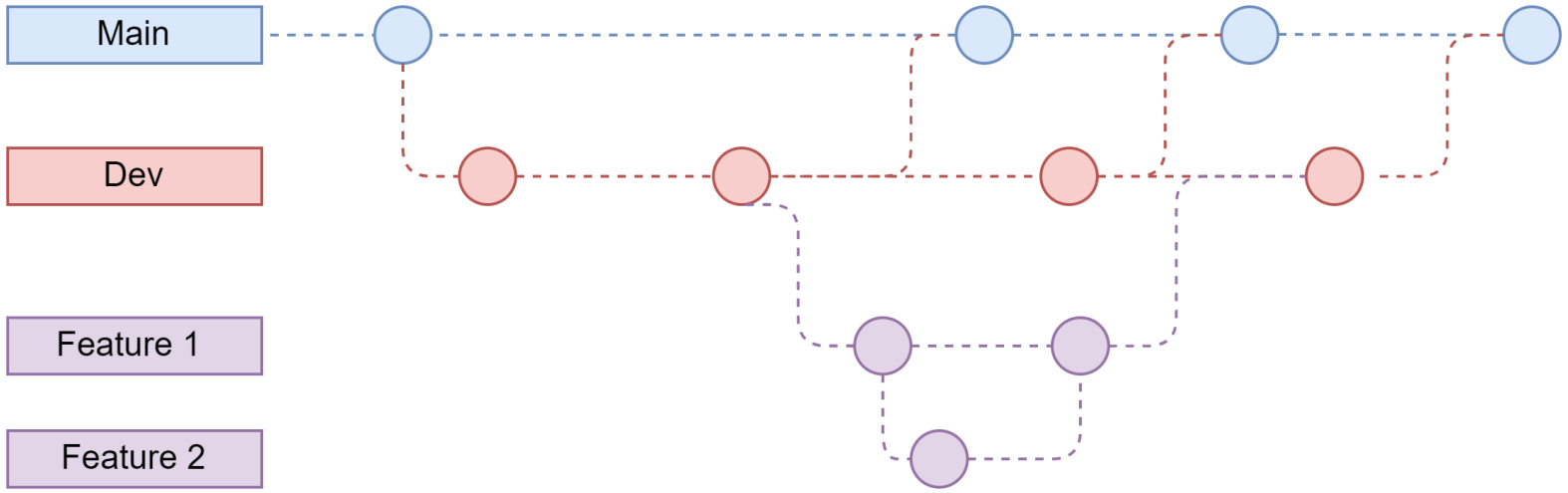
- For multi person projects – with one version in production:



git Flow

- For multi person/team projects – with safe main and multiple releases:

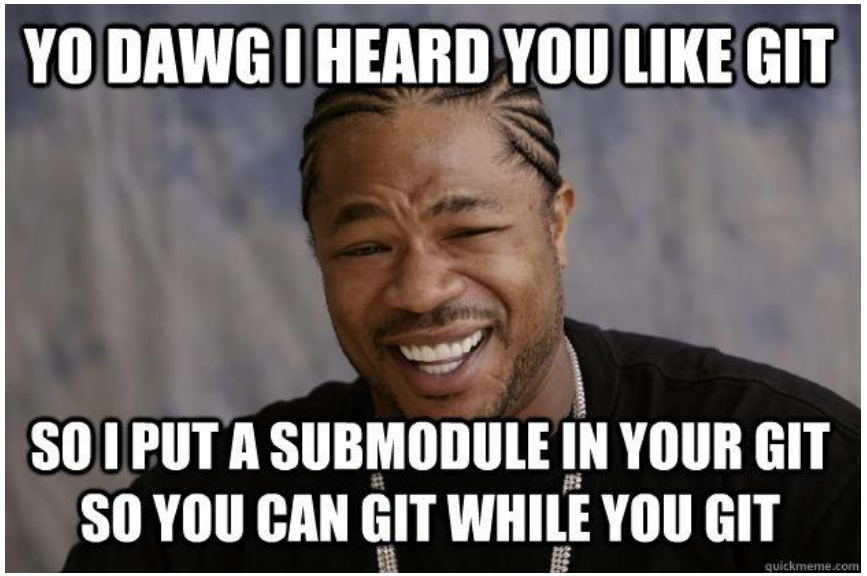




Additional Branches like Release, Bugfixes etc. can be added on the way.

git Submodules

- One Project uses one or more other projects as submodule
- Develop them seperately
- Libraries sometimes can be difficult to maintain, alternatively you can integrate other repos



- More on Submodules: https://git-scm.com/book/en/v2/Git-Tools-Submodules
- https://www.atlassian.com/git/tutorials/git-submodule

git LFS

Git Large File Storage (LFS) replaces large files such as audio samples, videos, datasets, matlab model files, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.

Cool Resources

- https://learngitbranching.js.org/
- Git - Downloading Package (git-scm.com)

Questions?


git remote add origin
git remote origin add
git add remote origin
git add origin remote
JAKE-CLARK.TUMBLR


git push -f


In case of fire
1. git commit
2. git push
3. exit building
Git > Your life


git commit
git push
git add .


YOU USE GIT VIA A GUI
I USE GIT VIA COMMAND LINE
WE ARE NOT THE SAME


When you're dead but remember you forgot to git commit git push your last code iterations

Vielen Dank für Ihre Aufmerksamkeit

anouri GmbH

Alte Leipziger Straße 62
63571 Gelnhausen
Germany - Hessen

Ansprechpartner:

Manuel Pieke
M: +49 (0) 151 175 89 052
E: manuel.pieke@anouri.gmbh

Und hier fängt die Geschichte an...