Fachhochschule Kiel
University of Applied Sciences

Tools and Programming for Data Science
Version Control with Git and GitHub (I)

Study Program Data Science
Prof. Dr. Tillmann Schwörer

Our course agenda

- **Introduction and overview**
- **NumPy**: Basic data handling with Numpy arrays
- **Pandas**
  - Exploratory data analysis
  - Data consolidation
  - Data cleaning
- **Data visualization using Matplotlib and Seaborn**
- **Interacting with APIs**
- **Interacting with SQL databases**
- **Version Control with Git and GitHub**
- **Advanced Python**

Python foundations

Data types

Operators

Functions

Control flow and iterators

Programming concepts & paradigms

See also Precourse Programming

Tooling

Installation

Visual Studio Code

Jupyter Notebooks

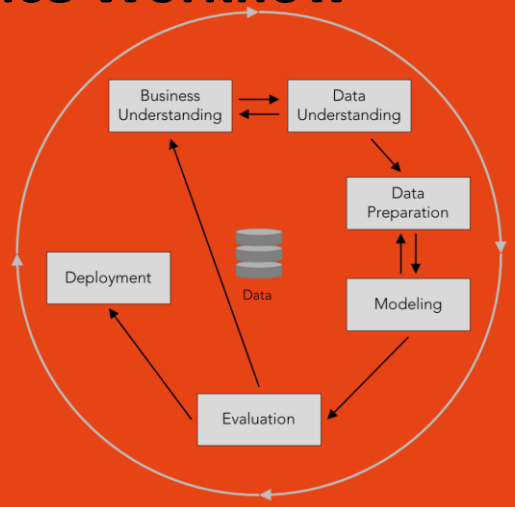
Packages

Virtual Environments

Git and Github

Python

Data Science Workflow

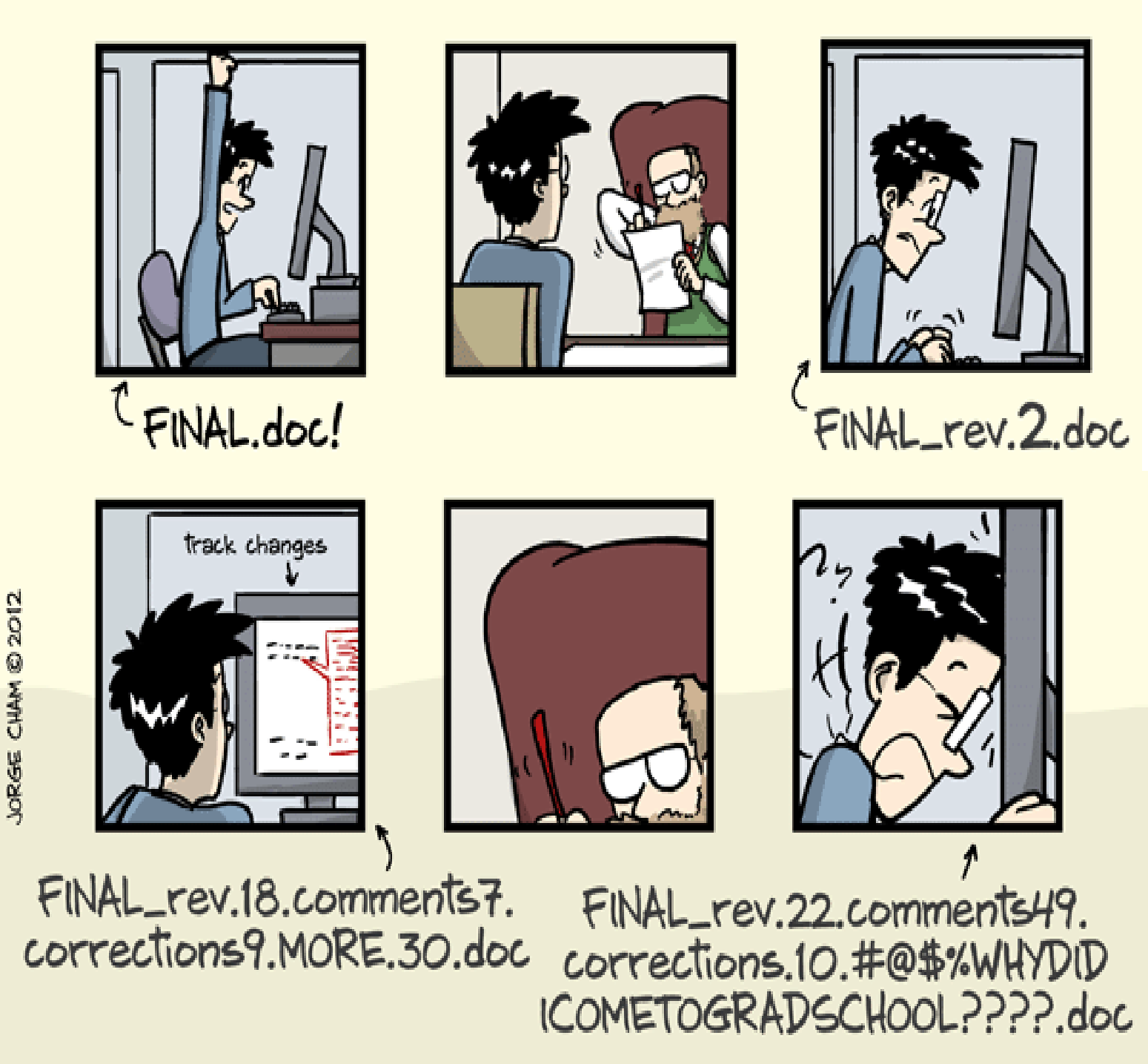


NumPy pandas matplotlib

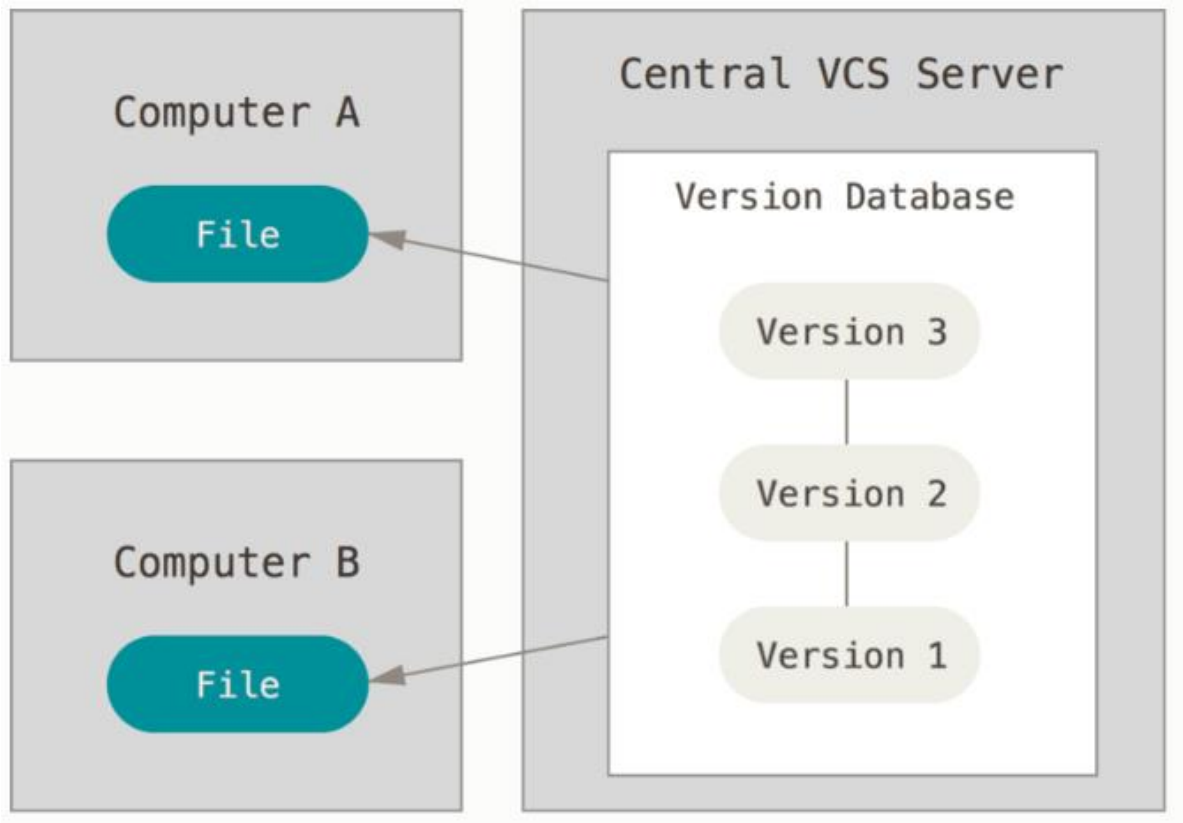
Version control system

- A version control system records changes in files over time
- Advantages
  - Understand what, when, and who changed?
  - Compare different states of the project
  - Revert to previous states
  - Experiment safely in branches
  - Collaborate with others on the same project
- Git:
  - Distributed version control system
  - Created by Linus Torvals in 2005

Why Version Control?

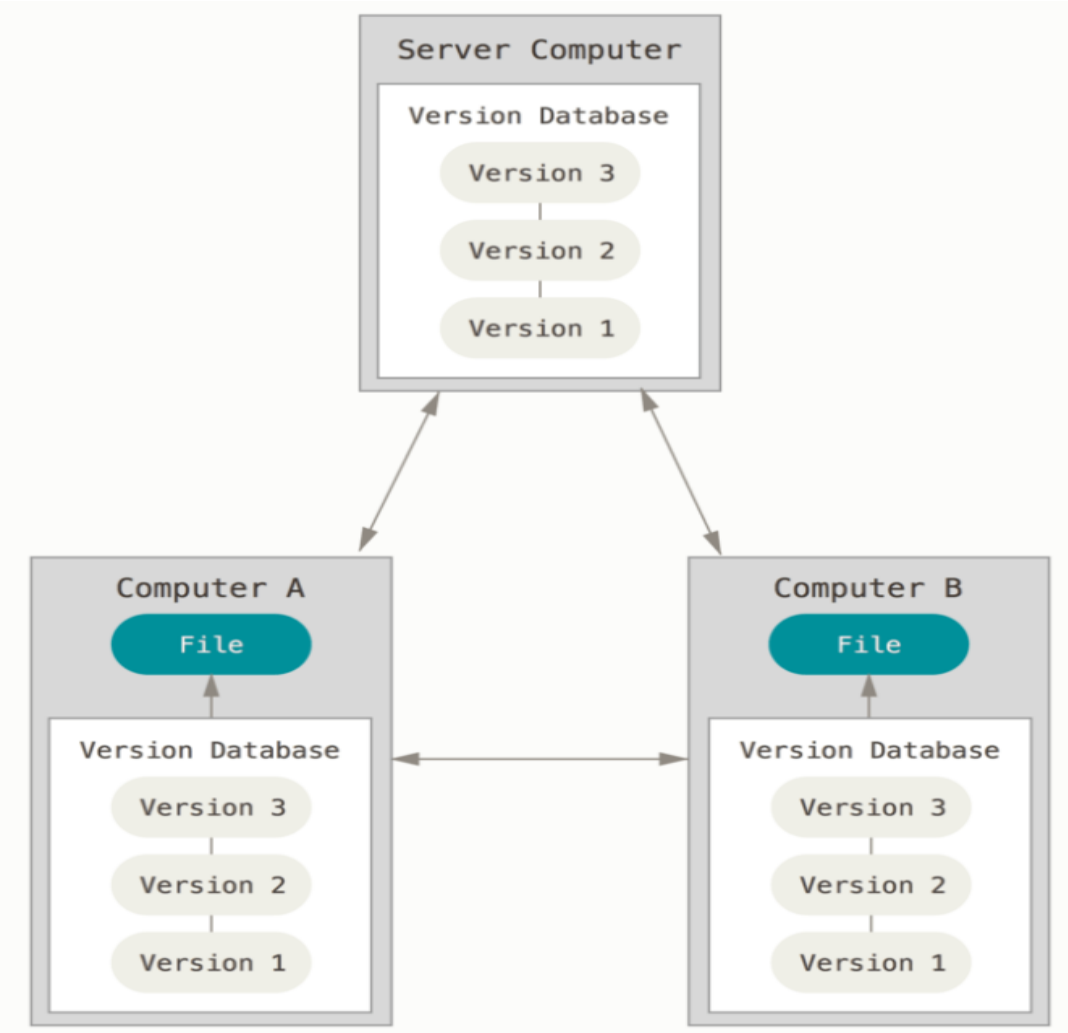


Centralized Version Control System



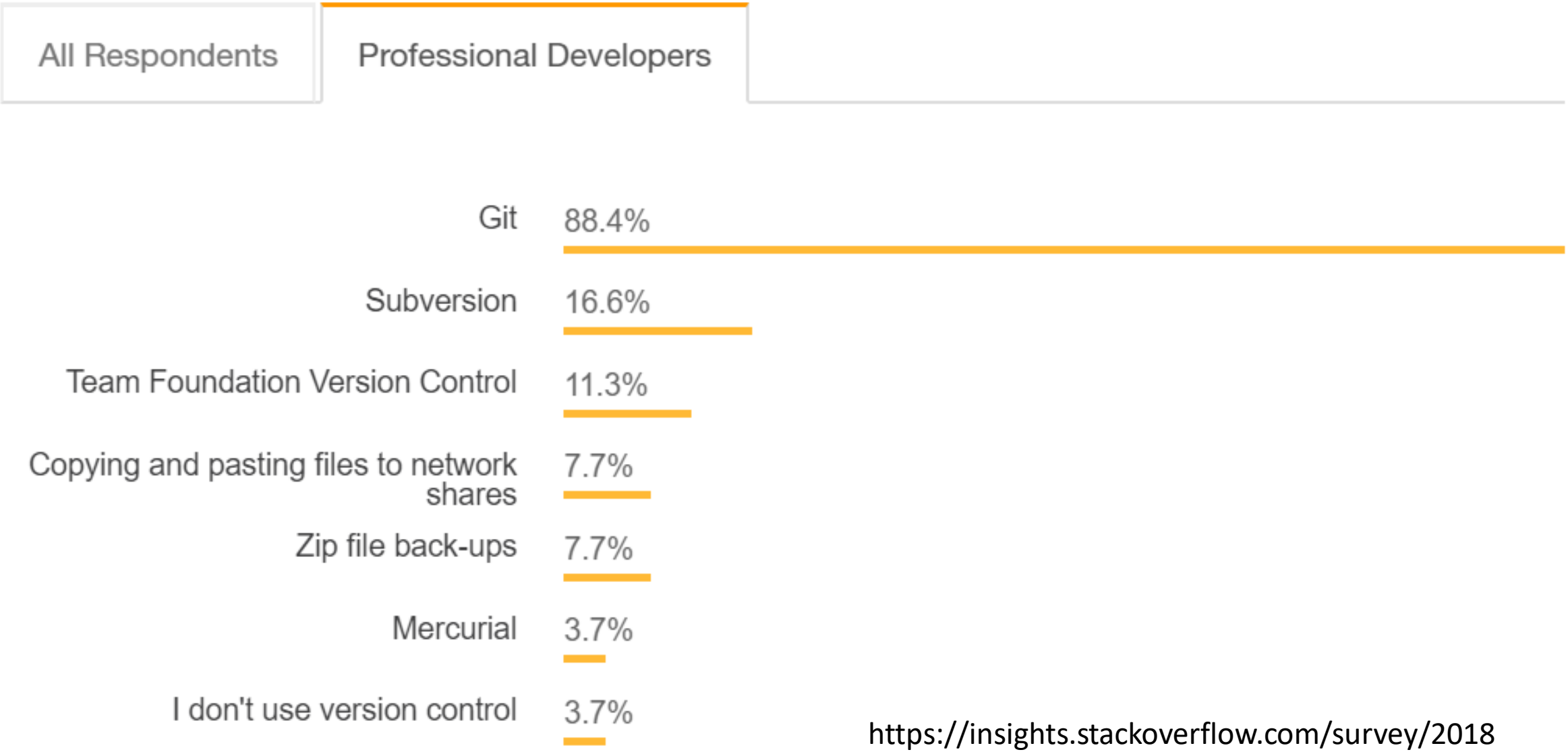
- How it works
  - Project history is stored on a central server
  - Users checkout files, edit local copies, and commit to central server
- Pros and Cons
  - \+ Allows for access and permission control
  - \+ Easier, more linear history
  - \- Central server is a single point of failure
  - \- High network dependency
  - \- Working in branches is more difficult

Decentralized Version Control System



- How it works
  - Each user has a full copy of project history
  - Most operations can be done offline
  - Network dependeny only when changes are synchronized with central repository
- Pros and Cons
  - \- No access control possible
  - \- More complex
  - \+ Multiple backups
  - \+ Low network dependency
  - \+ Working in branches is easy
  - \+ Ideal for complex, distributed projects

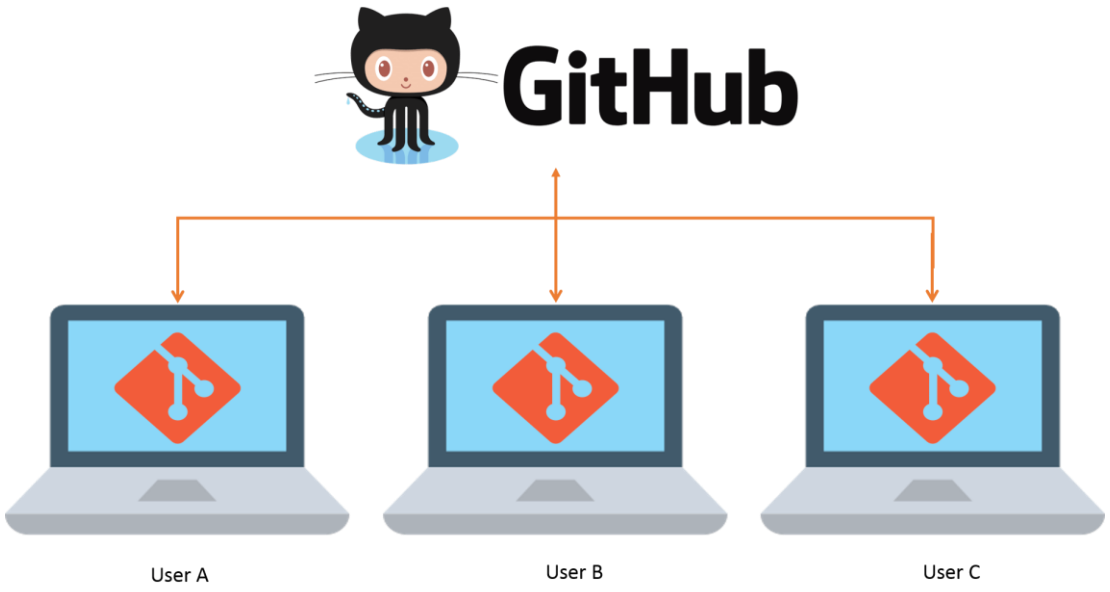
Most popular version control systems

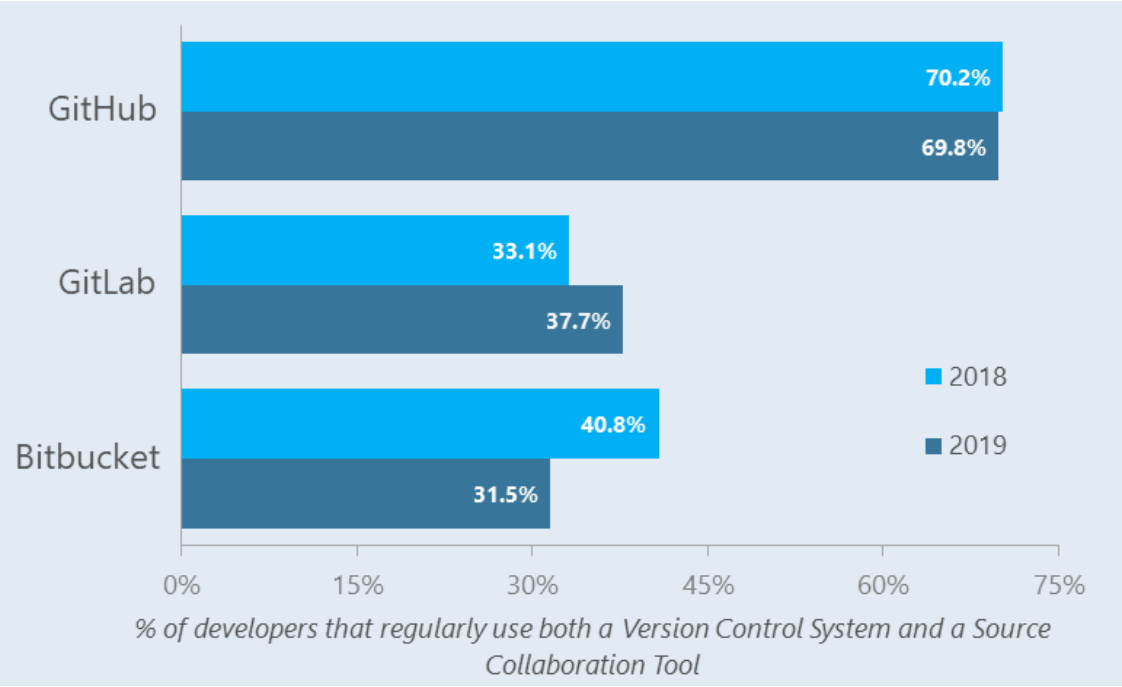


https://insights.stackoverflow.com/survey/2018

GitHub

- Hosting service for Git repositories
- Collaboration plattform: bug tracking, feature requests, task management, wikis, etc.

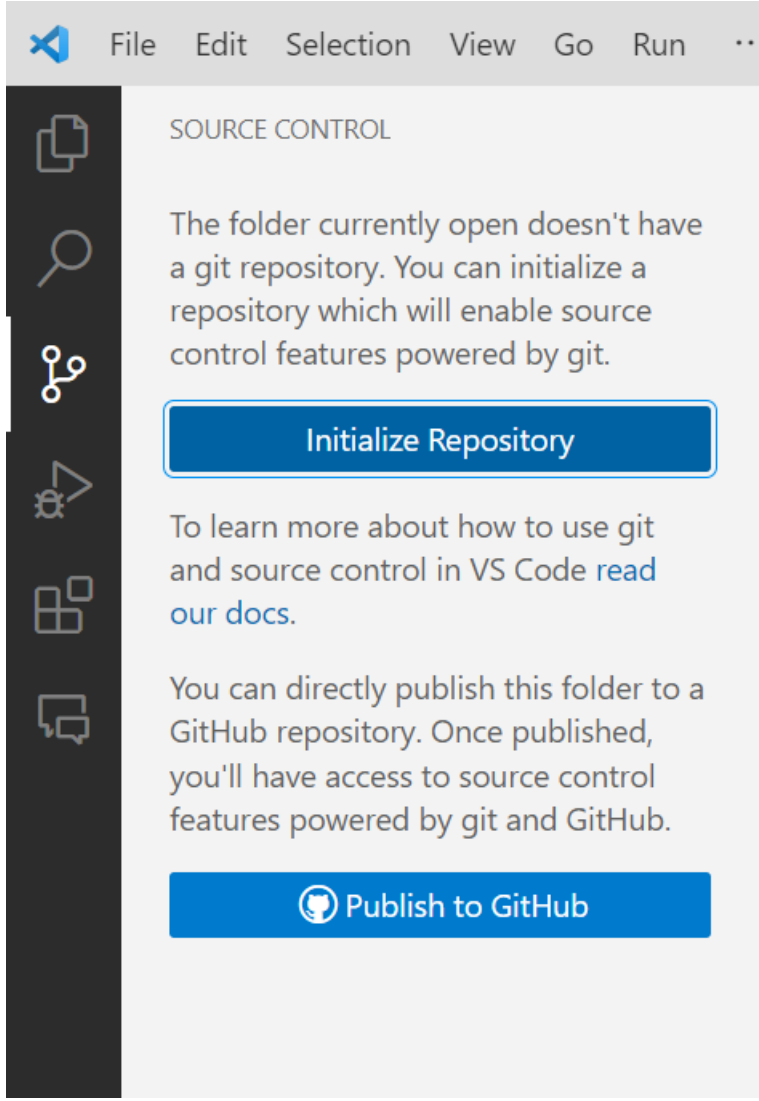




Our Agenda


Standard Git Workflow
Undo changes
Branching and Merging
Handling merge conflicts
Collaboration workflows

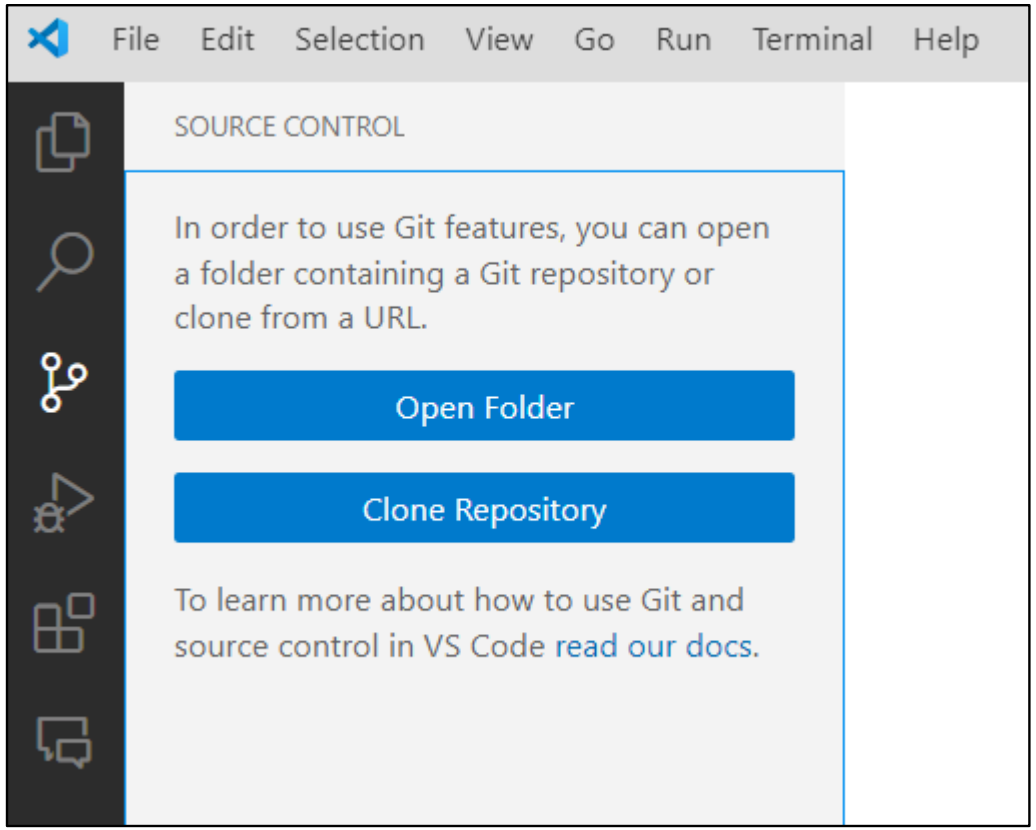
Initialize New Repository



- Use git init to start tracking changes in some directory

```
git init
```

Clone Existing Repository



Use Case:

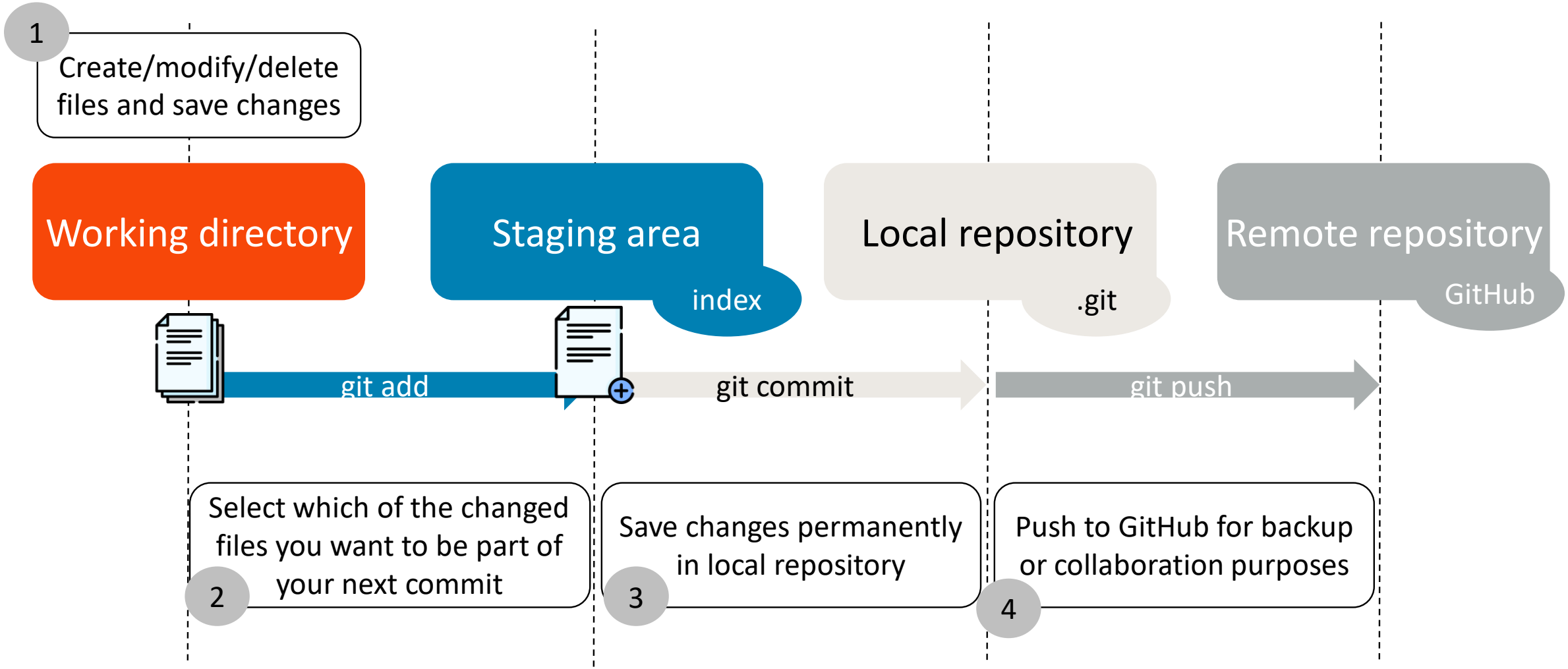
- Get a local copy of an existing git repository hosted e.g. on Github, Gitlab, Bitbucket

Example:

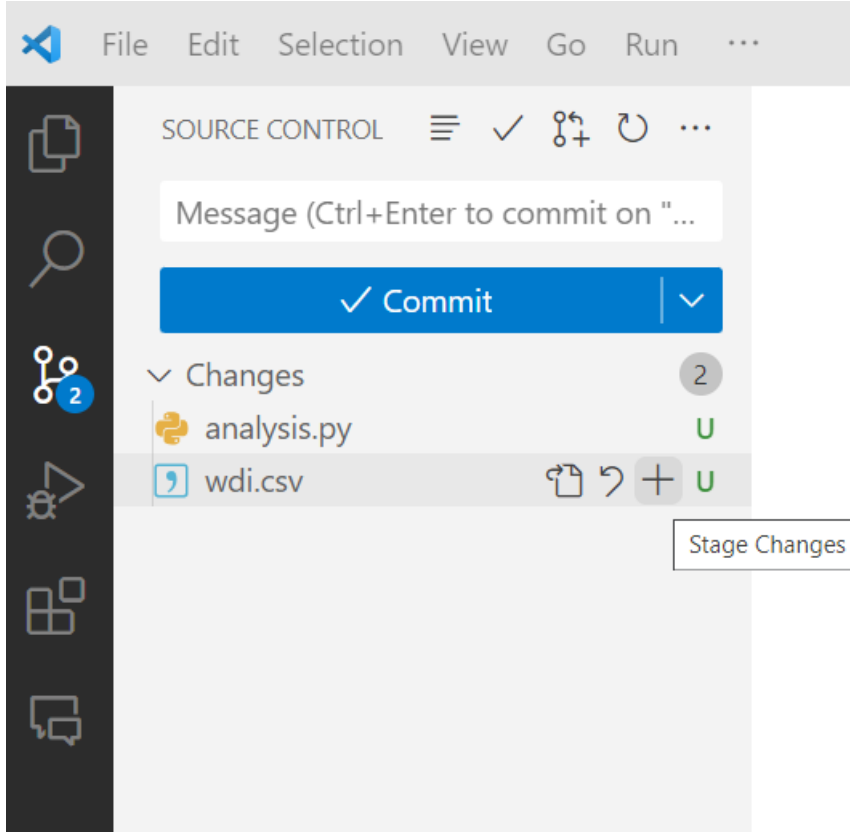
git clone https://github.com/pandas-dev/pandas.git

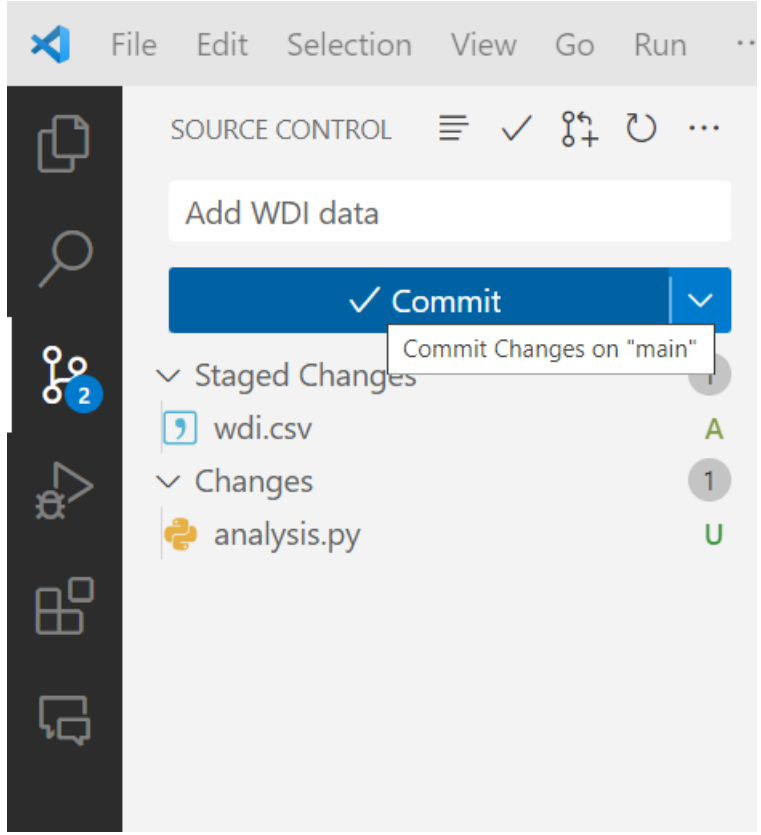
```
git clone <repourl>
```

Standard Git Workflow



Add and Commit





`git add`

`git commit`

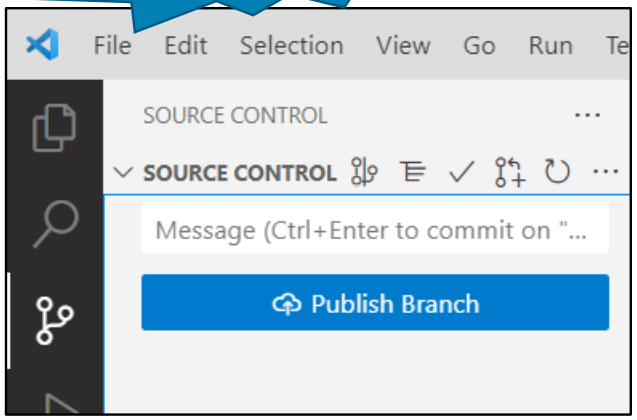
- Files in the working directory are untracked (U) so far
- git add wdi.csv starts tracking and stages the changes of file wdi.csv (A) → select files and review them before the actual commit
- git commit saves the staged changes to the commit history
- A Commit message must be provided

Add and commit commands

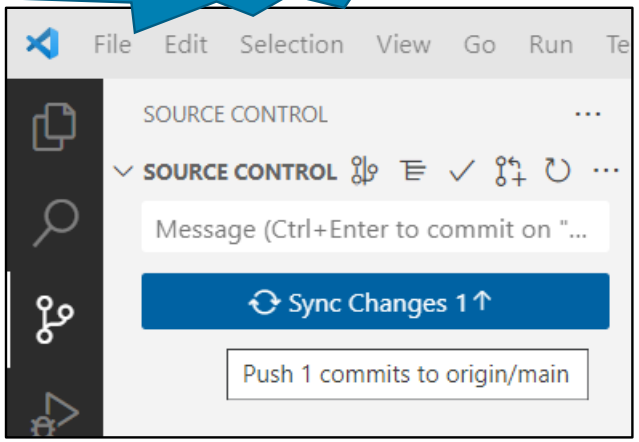
Command	Explanation
git add <file1>	Add file1 to the index
git add <file1> <file2> …	Add all specified files to the index
git add .	Add all changed files to the index
git commit	Opens a text editor that prompts you for a commit message
git commit -m „message“	Commit with a commit message
git commit -m „title“ – m „description“	Commit with a more detailed message (title + description)
git commit --amend	Make changes to the most recent commit, possibly also modifying the commit message
git commit -am „message“	Add all files and commit in one step

Push to Remote Repository

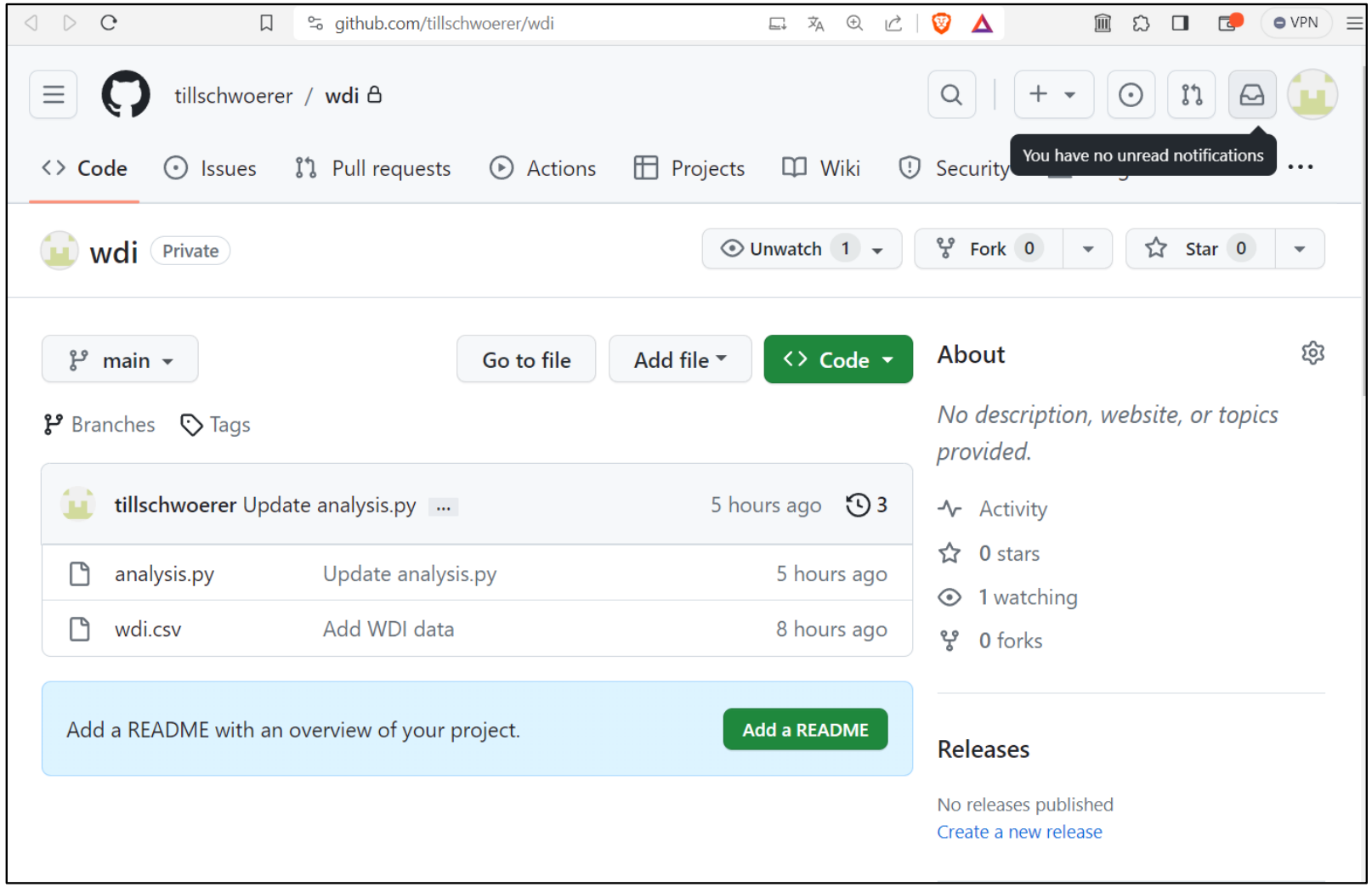
Only once



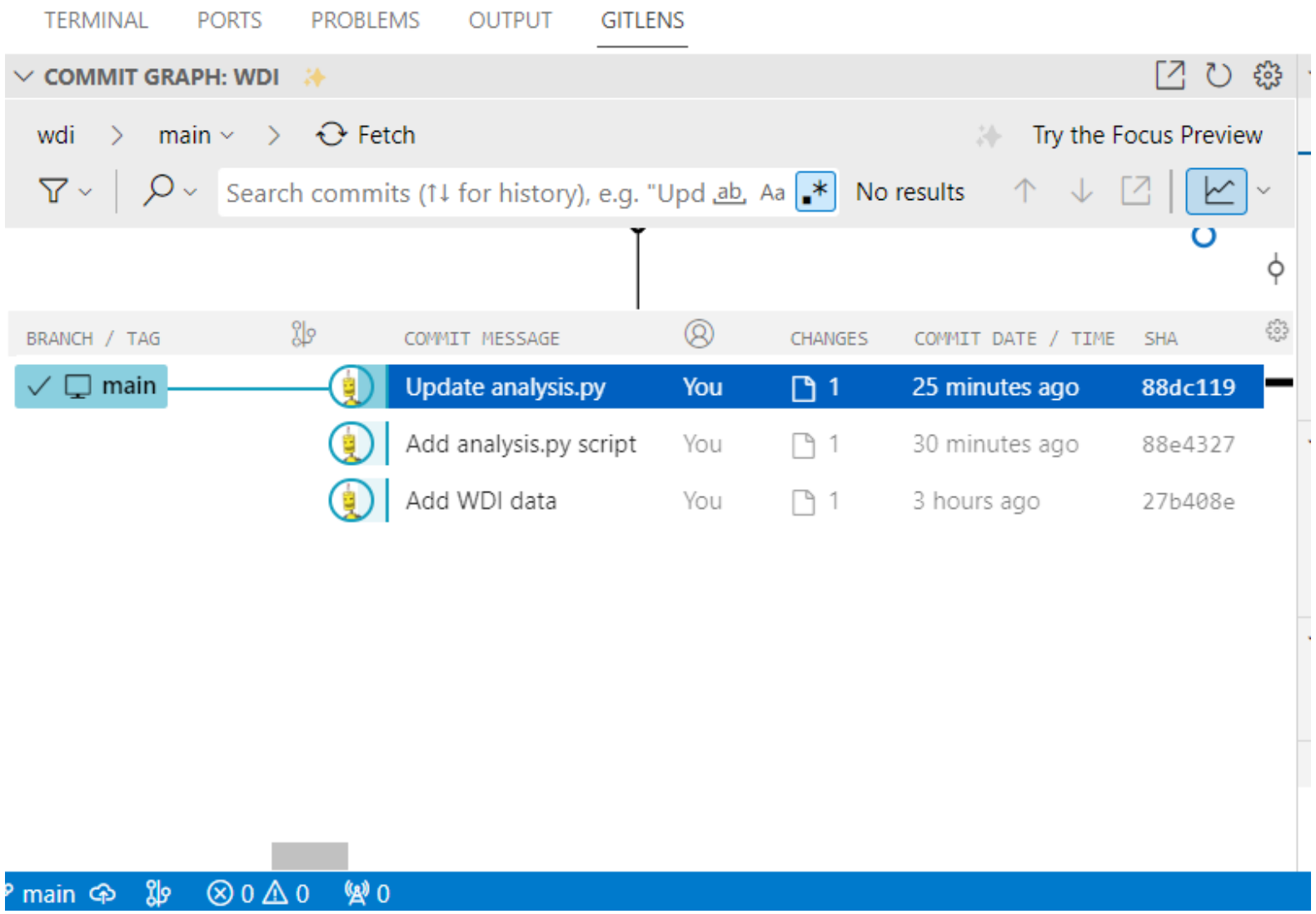
Repeatedly




```
git push
```



Commit History



`git log`

- git log shows the commit history
- For nice visual representations, use e.g. the GitLens extension
- Each commit is uniquely identified by a 40-digit commit id/hash (often abbreviated by the first 7 digits)

Comparing versions

- Comparing commits works best for **text based files** (.txt, .csv, .py, .R, .md, .yaml, json, …) due to color highlighting of additions and deletions

```
git diff
```

analysis.py (88e4327) ↔ analysis.py (88dc119)

C: > Users > tschwoer > Documents > github > wdi > analysis.py

```
1   import pandas as pd
2
3   # Read data
4   df = pd.read_csv("wdi.csv)
5
6   # Describe data
7   df.describe()
```

```
1   import pandas as pd
2
3   # Read data
4   df = pd.read_csv("wdi.csv)
5
6   # Describe data
7   df.describe()
8+      You, 39 minutes ago • Update analysis.py
9+  df.head()
```

Comparing versions

git diff

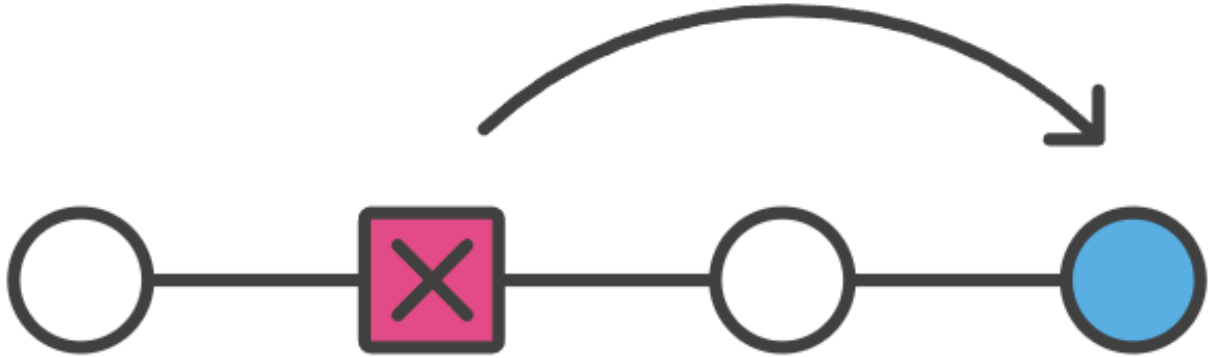
Command options	Example	Explanation
git diff		Compare working directory with last commit
git diff <commit1>	git diff 88e4327 git diff HEAD~1 git diff HEAD~2	Compare commit 884327 with last commit Compare second last with last commit Compare third last with last commit
git diff <commit1> <commit2>	git diff 88e4327 88dc119 git diff HEAD~1 HEAD	Compare commits 88e4327 and 88dc119 Compare second last with last commit
git diff <commit1> -- filename	git diff HEAD~1 -- analysis.py	Compare only the versions of file analysis.py

Git revert

- Adds a new commit that reverts the bad commit
- Leaves the commit history intact

– Commit history is less clean

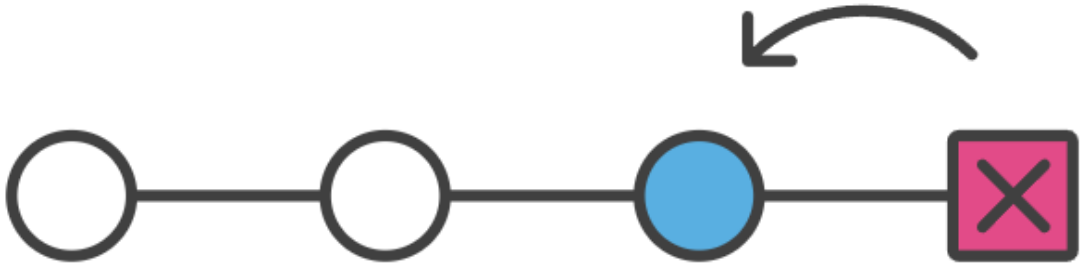
\+ Can be safely done even if you have already pushed to GitHub



```
git revert <badcommit>
```

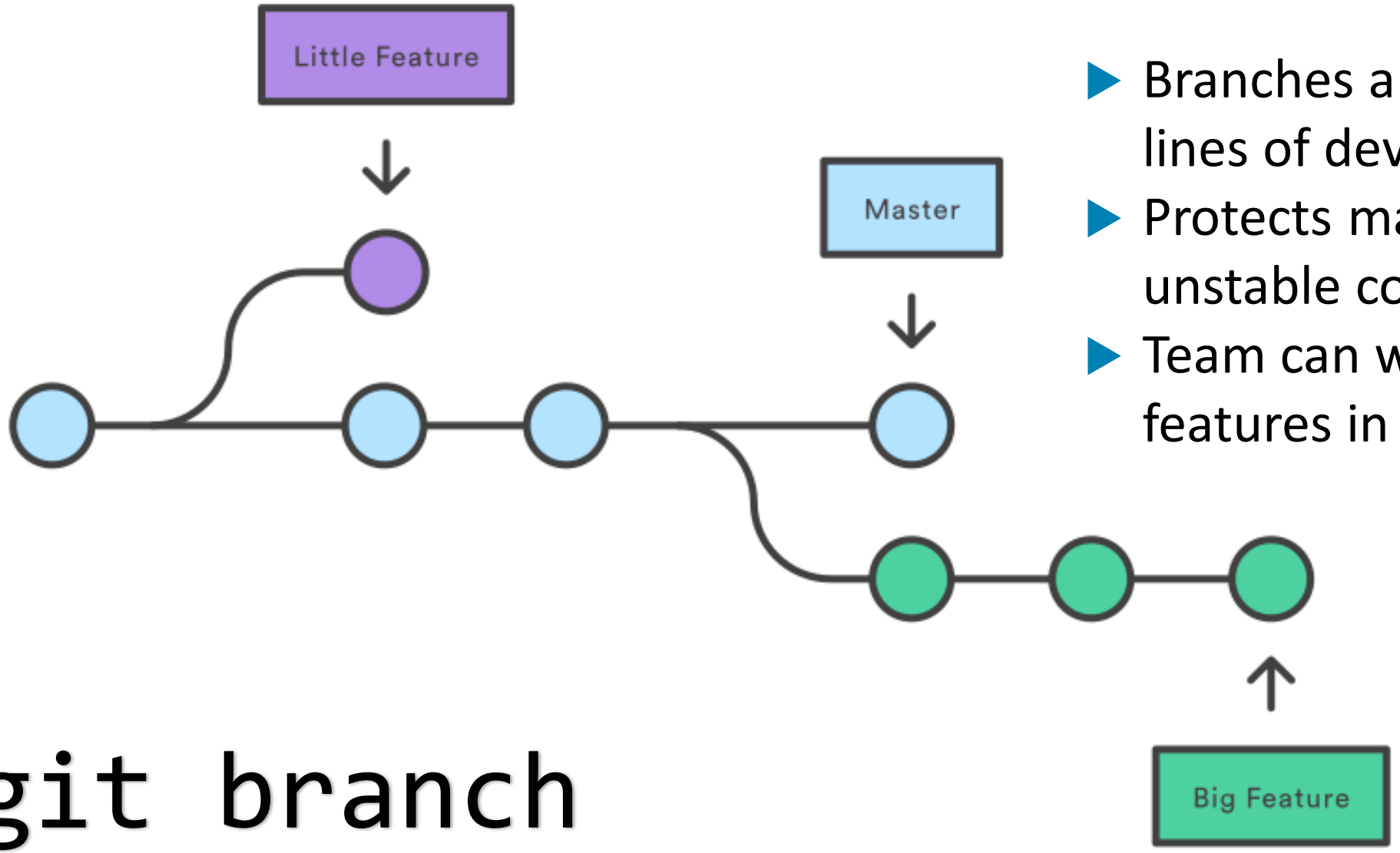
Git reset

- Rewrites the commit history
  - --hard: resets irreversably (caution!)
  - --soft: keeps changes in working directory and index
  - --mixed: keeps changes in working directory (default)
- Advantage: Commit history is clean
- Caution: use only for changes that have not been pushed to Github yet



```
git reset <targetcommit>
```

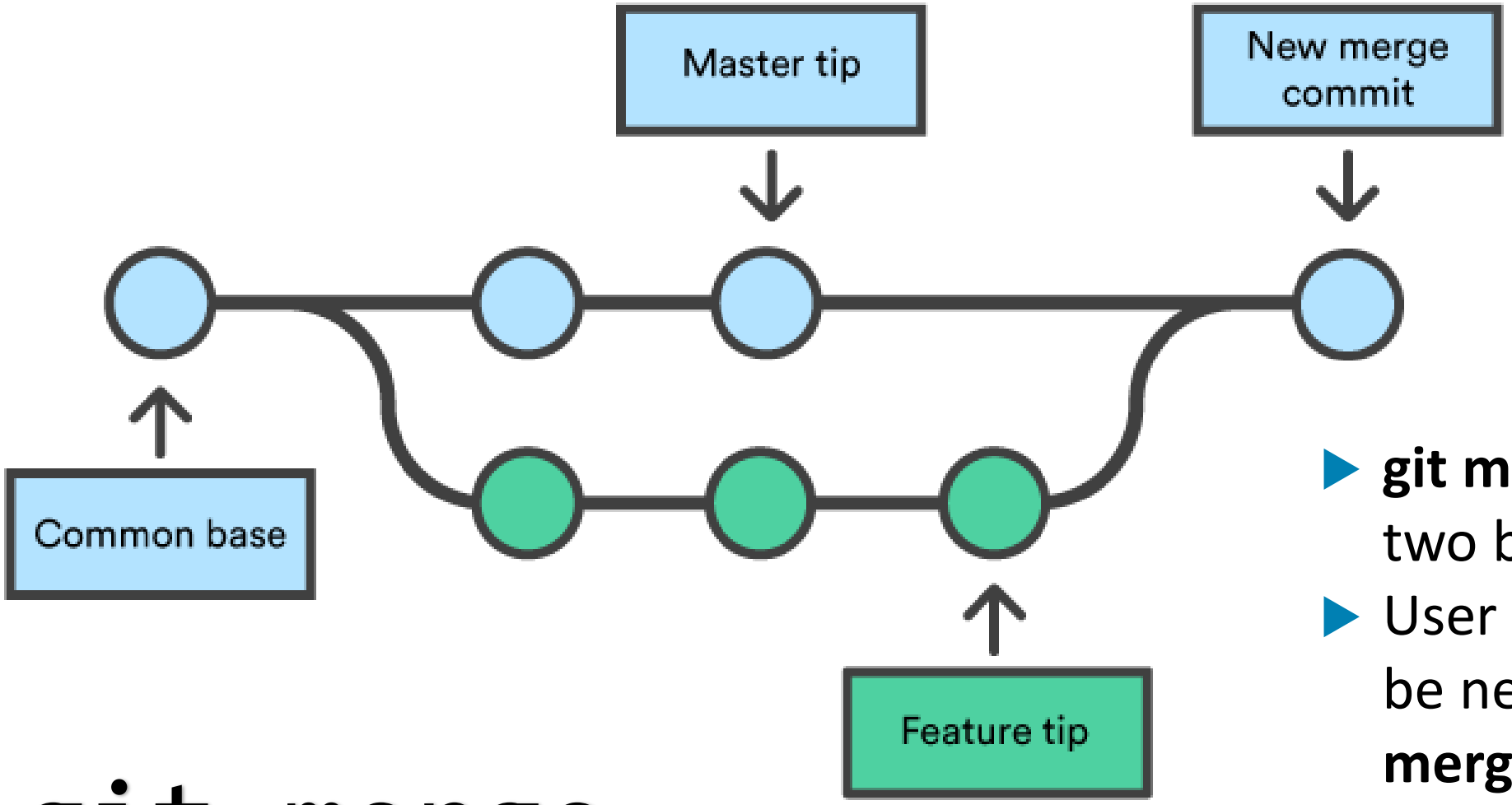
Branching



- Branches are independent lines of development
- Protects main branch against unstable code
- Team can work on separate features in parallel

```
git branch
```

Merging



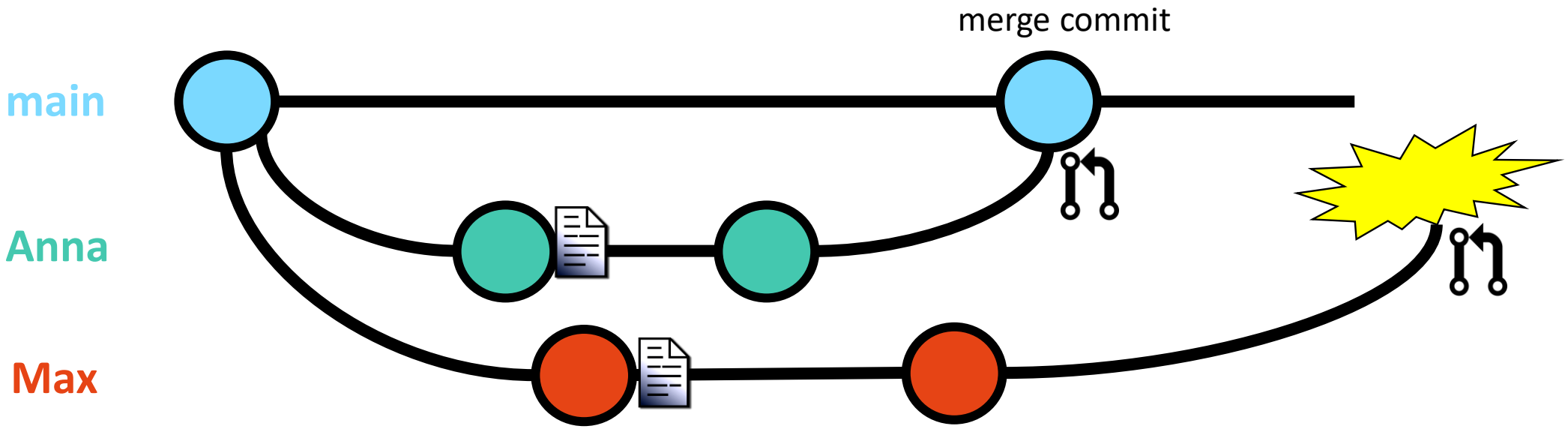
- **git merge** combines two branches into one.
- User intervention may be needed to resolve **merge conflicts**

```
git merge
```

Branching and Merging

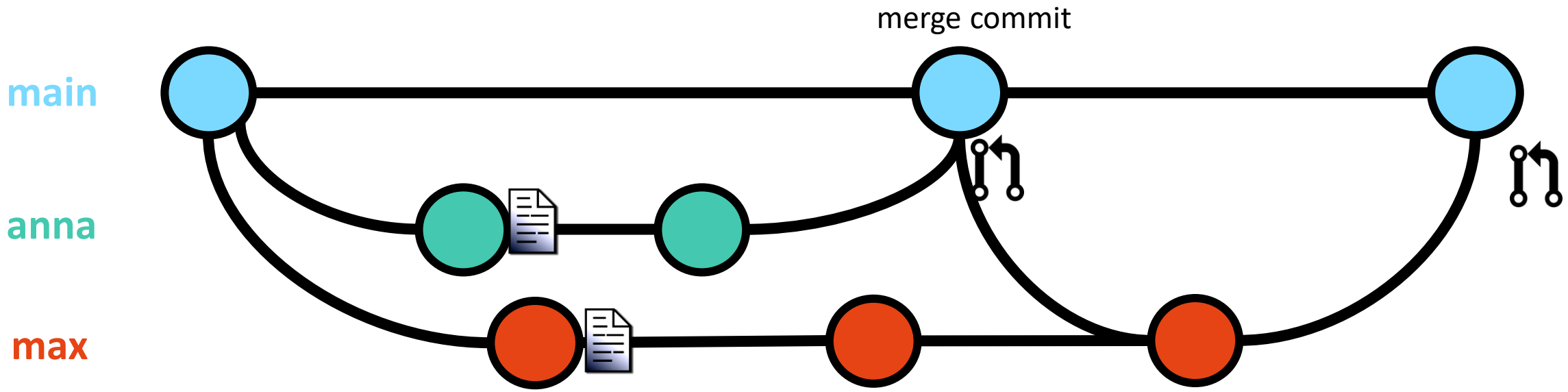
git branch	Lists available branches
git branch <name>	Creates a branch with specified name
git branch -d <name>	Delete a branch, unless it has unmerged changes
git branch -D <name>	Force delete a branch even if it has unmerged changes
git switch <name>	Switch to a branch
git switch -c <name>	Create and switch to new branch
git merge <name>	Merge the specified branch into the active branch

Merge conflicts



- Anna and Max edit the same file, commiting to their local repos each
- Anna opens a PR and her changes are merged into the main branch
- Max opens a PR, but a merge conflict shows up

Merge conflicts



Option 1: resolve conflict locally“

Option 2: resolve conflict remotely (on Github)

Merge conflicts

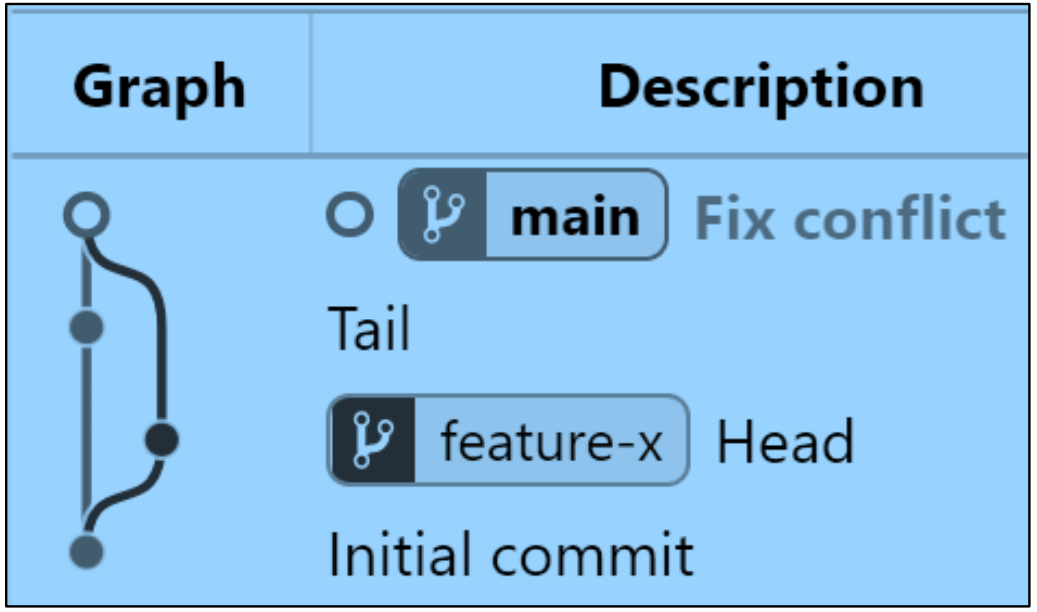

```
git merge feature-x
```

```
Auto-merging analysis.py
CONFLICT (content): Merge conflict in analysis.py
Automatic merge failed; fix conflicts and then commit the result.
```

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<<·HEAD (Current Change)
df.tail()
=======
df.head()
>>>>>>>·feature-x (Incoming Change)
```

```
df.head()
df.tail()
```

```
git commit -m "Fix conflict"
```



Further aspects

- **.gitignore:**
  - define files that you don’t want to track
  - e.g. password files (.env), .ipynb_checkpoints, very large binary files
- **Large files**
  - GitHub file size limit: 100 MB
  - Git Large File Storage: tracking files up to 2 GB
  - Git LFS stores references to the file in the Github repository. The actual file is stored separately.