Fachhochschule Kiel
University of Applied Sciences

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

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

Our Version Control Agenda

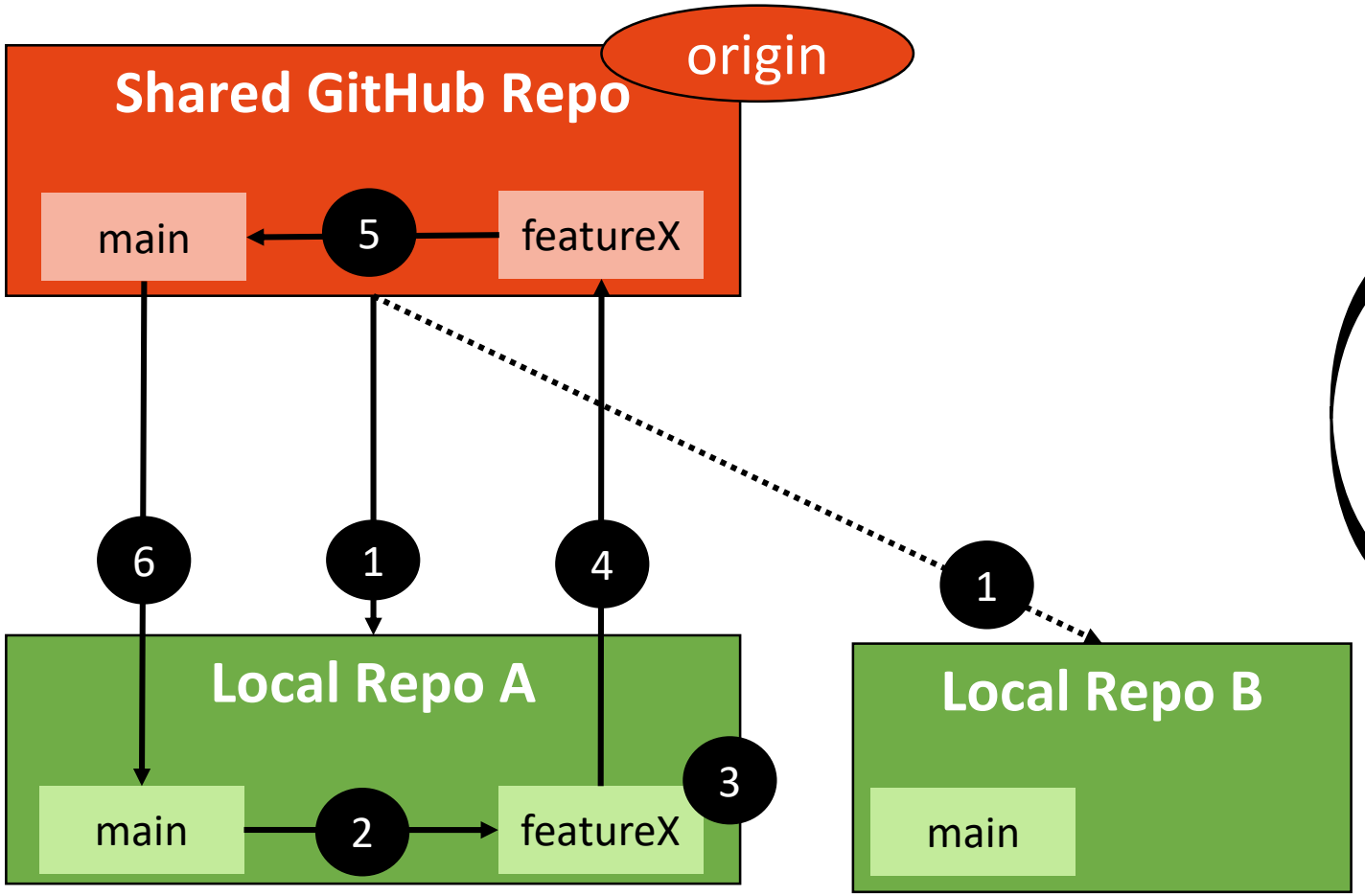
.gitignore

- **Text file that tells Git which files or directories are not to be tracked**
  - Python virtual environments
  - Jupyter Notebook checkpoint files
  - Large datasets
  - Large binary files, log files, …
  - Configuration or environment files with sensitive data (passwords, etc.)
- **Use cases**
  - Keep repo clean
  - Reduce repo size
  - Ensure that sensitive data are not publicly exposed
- **Template**: https://github.com/github/gitignore/blob/main/Python.gitignore

Scenario 1: Feature Branch Workflow

- **Setting**: you are part of a company data science team working on a predictive analytics project
- **Steps**
  - Initialization:
    - ✓The project resides in a central Git repository (e.g. hosted on Github, Gitlab, …)
    - ✓All team members clone this repository
  - Branching and Development:
    - ✓To develop a new predictive model, you create a new branch off the main branch
    - ✓You experiment with the data, train predictive models, and run tests
    - ✓You commit changes to this branch, keeping work isolated from the main branch
  - Integration:
    - ✓Once the feature is ready and tested, you push the branch to the central repo
    - ✓You create a pull request to merge it into the main branch.
    - ✓The team reviews the code before it is merged.

Scenario 1: Feature Branch Workflow



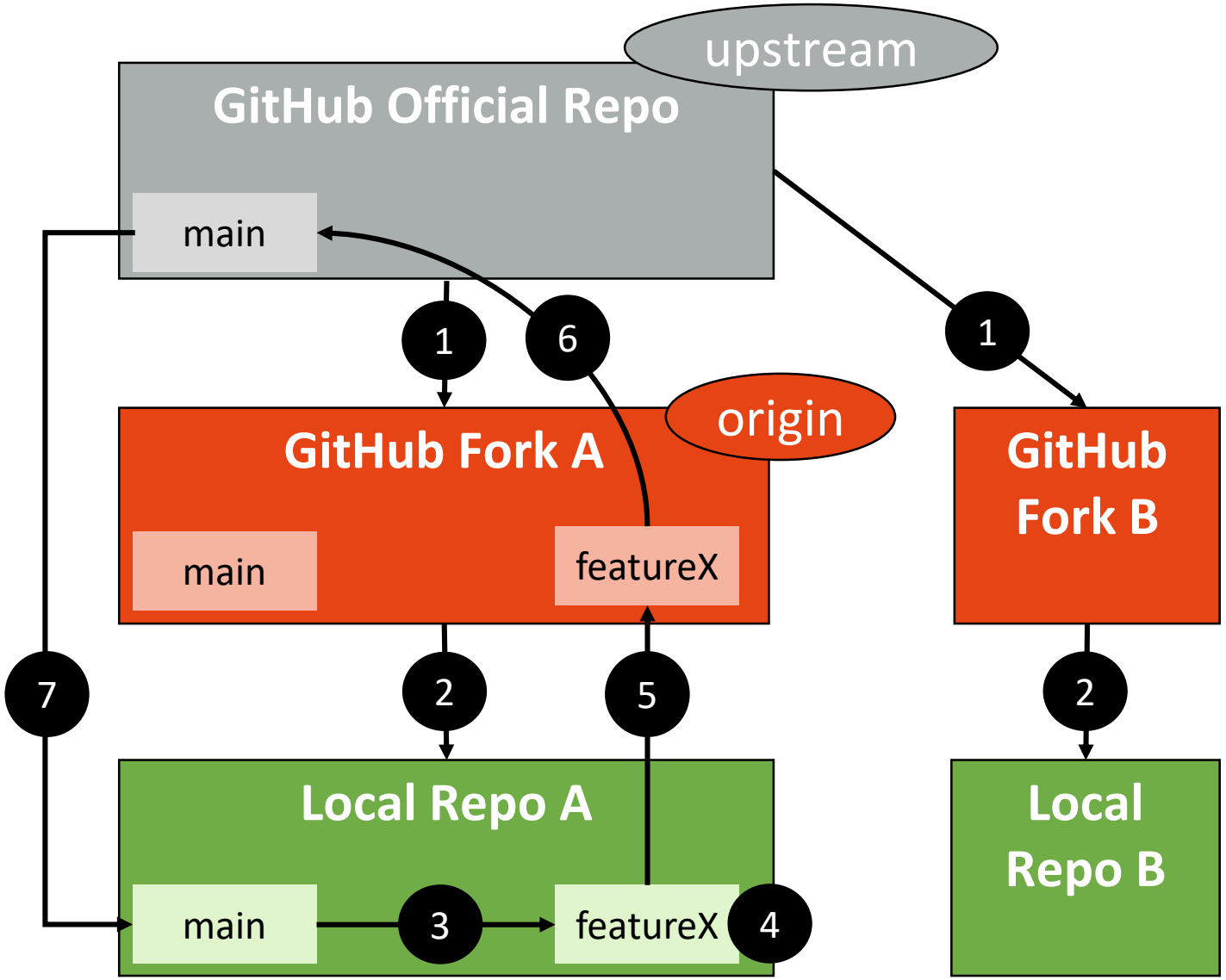
1. git clone <url>
2. git branch featureX
   git switch featureX
3. edit, git add, git commit
4. git push (--set-upstream)
5. Pull Request → git merge
6. git pull

Delete branch (local and remote):
git branch -d featureX
git push origin --delete featureX

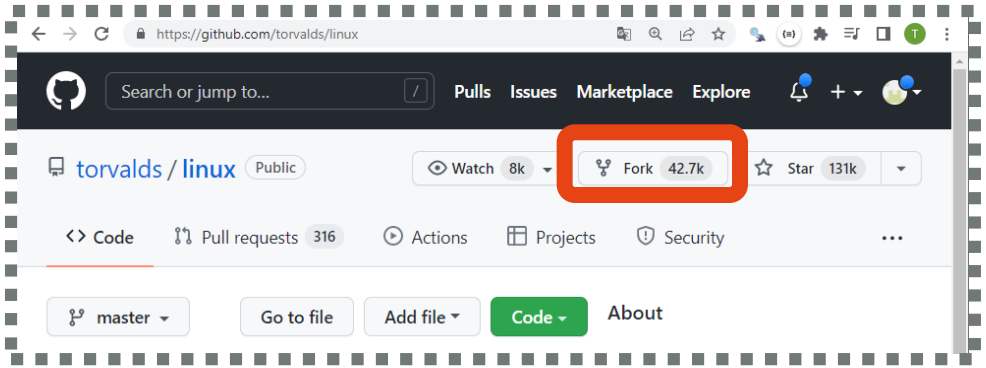
Scenario 2: Forking Workflow

- **Setting**:
  - You want to contribute to an open-source data science project (e.g. Pandas)
  - Forking workflow is used to manage contributions from many external collaborators who do not have direct write access to the project's main repository
- **Steps**
  - Initialization:
    - ✓Fork the main (upstream) GitHub repository to your own account, creating your personal copy of the Project on Github (origin)
    - ✓Clone the fork to your local system
  - Branching and Development → as in the feature branch workflow!
  - Integration:
    - ✓Once the feature is ready, push the branch to your fork
    - ✓Create a pull request to merge it into the main branch of the upstream repository
    - ✓The project maintainers review the code before it is merged

Forking Workflow

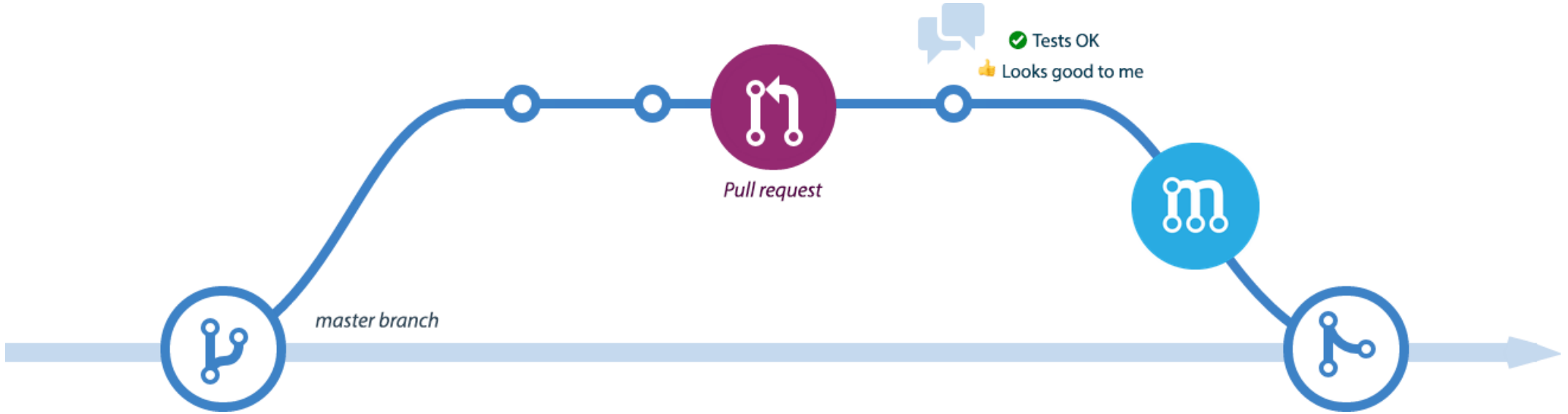


1. Fork
2. git clone <url>
3. git branch featureX
   git switch featureX
4. edit, git add, git commit
5. git push (--set-upstream)
6. Pull Request → git merge
7. (git remote add upstream <url>)
   git pull upstream

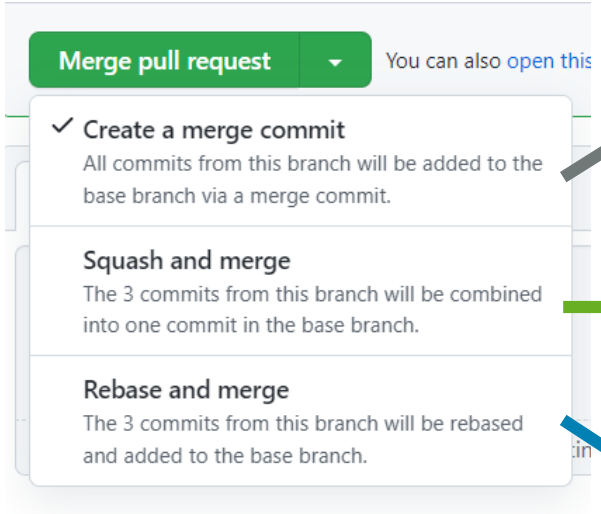


Pull Request

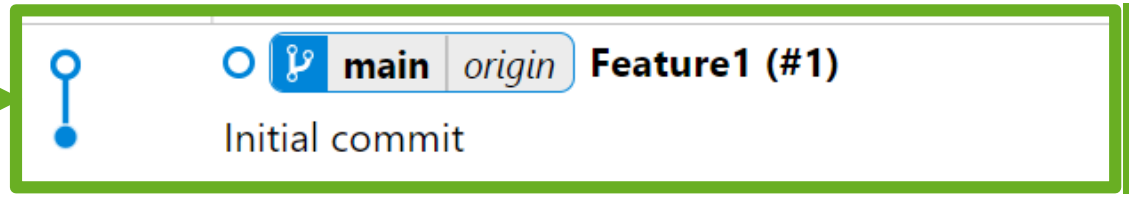
- **A Pull request is a GitHub feature**, not a git command: you request to pull changes from your feature branch (“merge request” on Gitlab)
- May involve multiple iterations of discussions, code reviews, and follow-up commits, before the commit is merged into the main branch



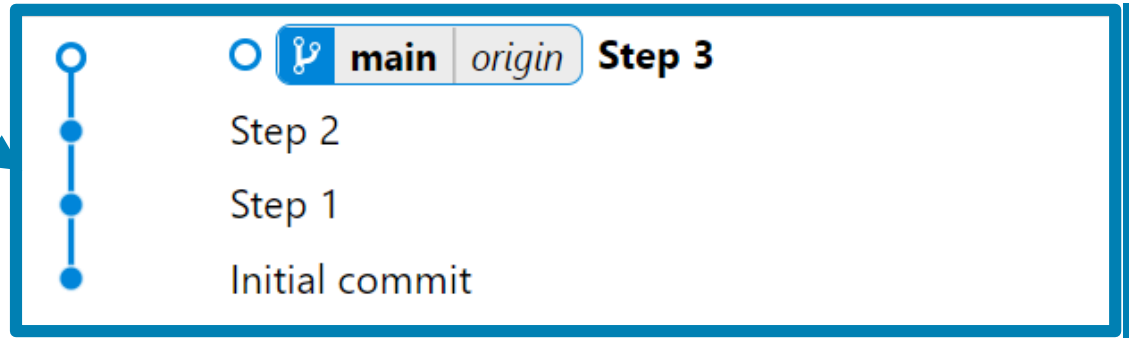
Merging a Pull Request



+ Truthful history of actions
- May pollute project history



+ Short and linear project history
+ Pull requests can still be identified
- Actual single commits are not logged anymore



+ Linear project history
+ Single commits are all part of the history
- Feature branches / pull requests cannot be identified any more