NICOLAUS COPERNICUS UNIVERSITY IN TORUŃ

RESEARCH UNIVERSITY
EXCELLENCE INITIATIVE

Introduction to Git version control

Iulia Emilia Brumboiu

iubr@umk.pl

27 October 2025

Today

- **Homework feedback**
  - Quick reminder of what we did last time,
  - General feedback on the Jupyter notebooks

- **Introduction to Git version control**
  - Git version control basics
  - Some of the most common git commands,
  - Generic project structure,
  - First class in a separate branch: IOData.

- **The Python project**

Part 1: Feedback

Last time:

- Introduction to using **conda** and working in a conda environment,
- Introduction to interactive programming with **Jupyter notebooks**.

You completed a notebook with 3 exercises meant to practice:

- Defining **Python routines** and **plotting** (Morse potential, gradient),
- **Thinking critically** about your implementation (checking cortectness of the gradient),
- Working with **h5py checkpoint files** (MP2 energy correction),
- Using useful **NumPy routines** (MP2 energy correction).

General feedback

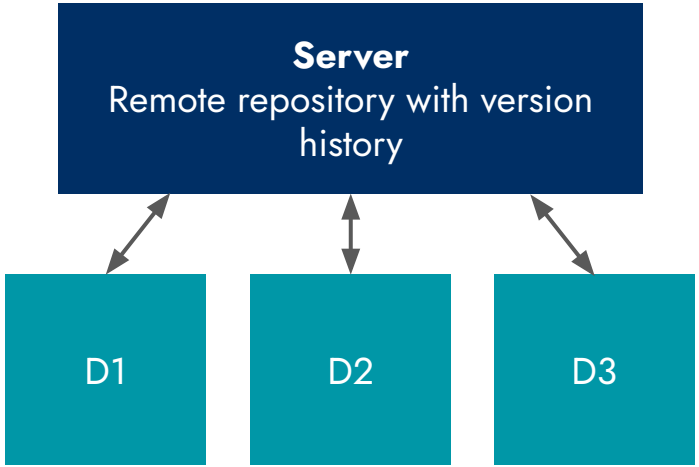
Part 2: Git version control

Git version control basics

Version control:

- **Record** changes to a set of files over time,
- Enable multiple developers to **collaborate** and develop different features of the same project,
- **Revert** selected files or the entire project to previous versions, **compare** changes over time, **record** project's **history** and who implemented what, etc.

Distributed version control

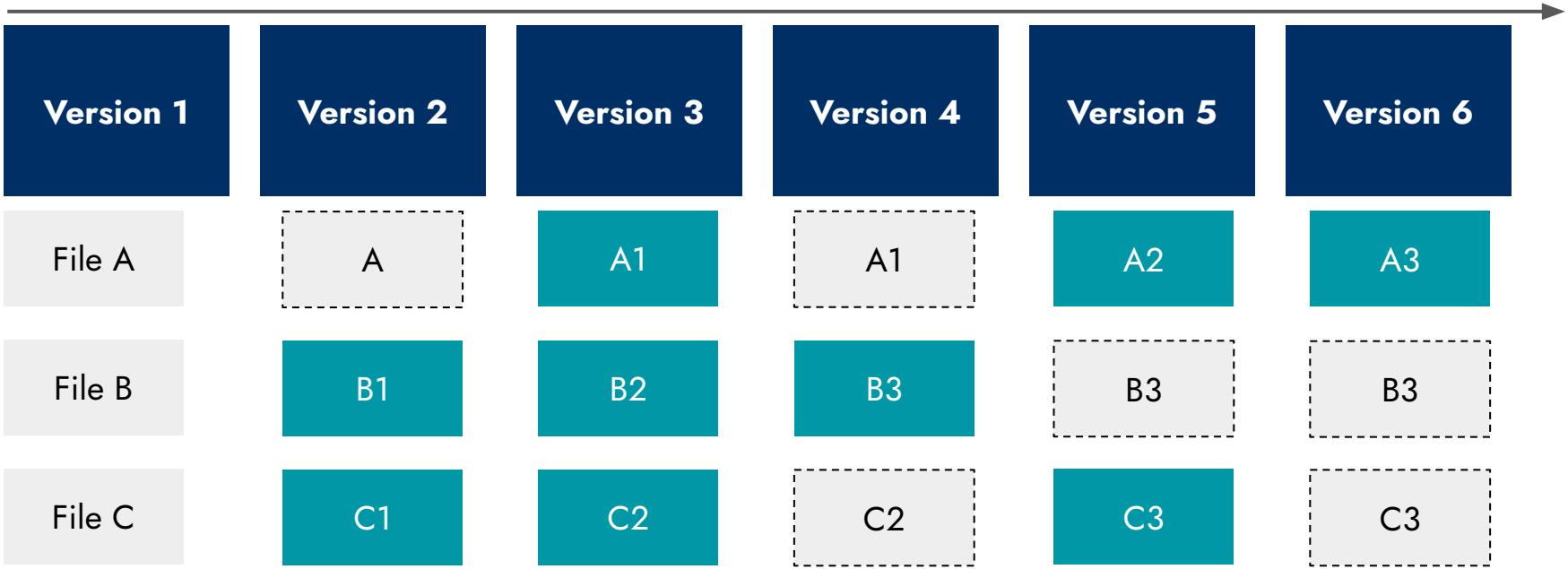


Each local **clone** fully mirrors the repository, including its full history.

https://git-scm.com/book/en/v2

Git version control basics

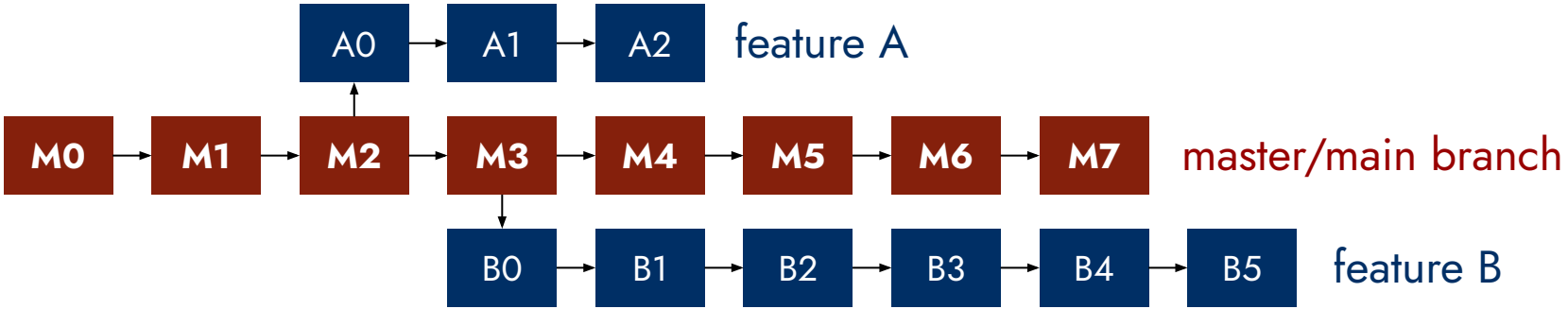
Data is stored as **snapshots** of the project over time.



https://git-scm.com/book/en/v2

Git version control basics

Working with branches



https://git-scm.com/book/en/v2

Branch naming conventions

- Avoid long confusing names
- Describe the type of branch: feature, bugfix, docs, etc.
- Examples: **feature/name_of_feature**; **bug/name_of_fix**.

Regular branches

dev: main development branch, where changes are made, reviewed and tested without touching the master branch.

master/main: stable default branch in repo.

test/QA: code for testing.

Temporary (delete after merge)

bugfix: bug found and resolved during production and testing.

feature: implementation of a new feature.

hotfix: patch to fix issue after release.

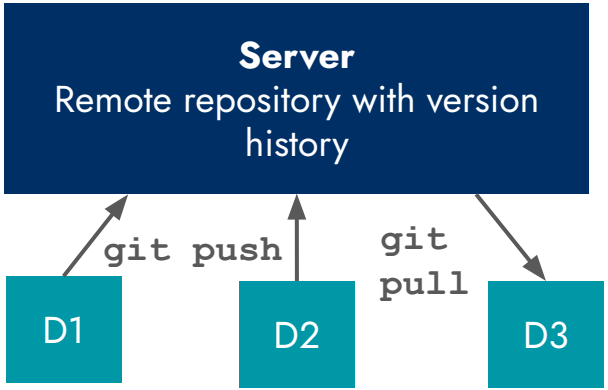
WIP: work in progress.

Git version control basics

Basic commands

- `git help`: lists a few useful commands
- `git clone [url]`: clone a repository
- `git config --list`: lists the config info.
- `git config --global user.email "[email]"`
- `git status`: lists modified files.
- `git add [file]`: add file to staging area
- `git commit -m [msg]`: record to version history
- `git push`: send committed changes to remote
- `git pull`: grab all changes from remote
- `git branch [name]`: create new branch [name]
- `git checkout [name]`: switch to branch `[name]`

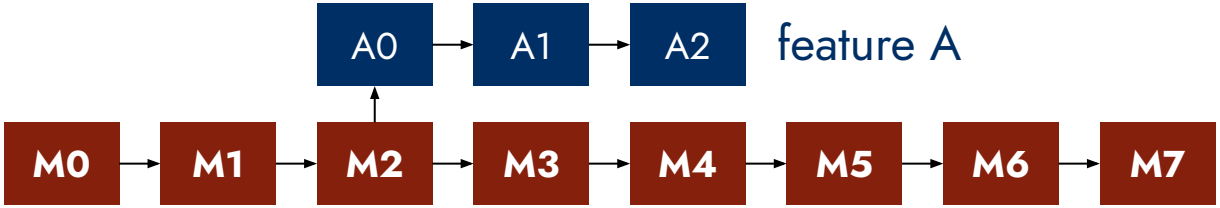
Distributed version control



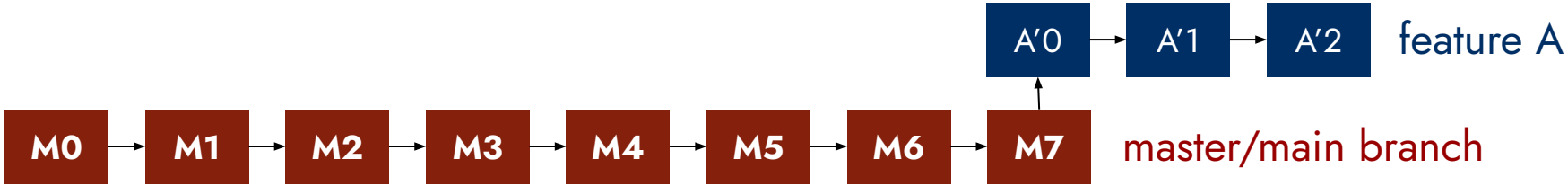
https://git-scm.com/book/en/v2

Git version control basics

Merging master/main into branch



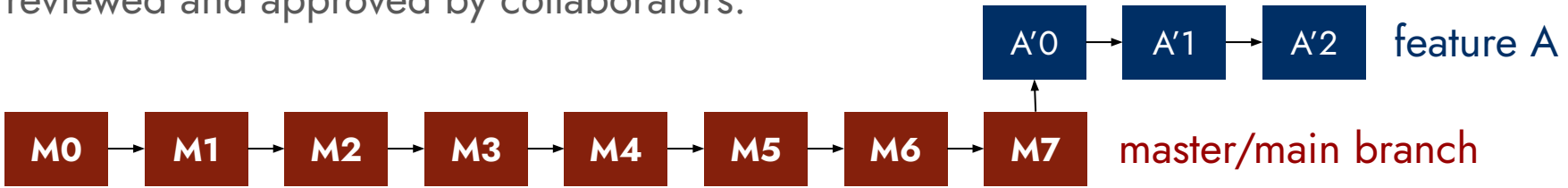

```
git checkout featureA && git merge master
```



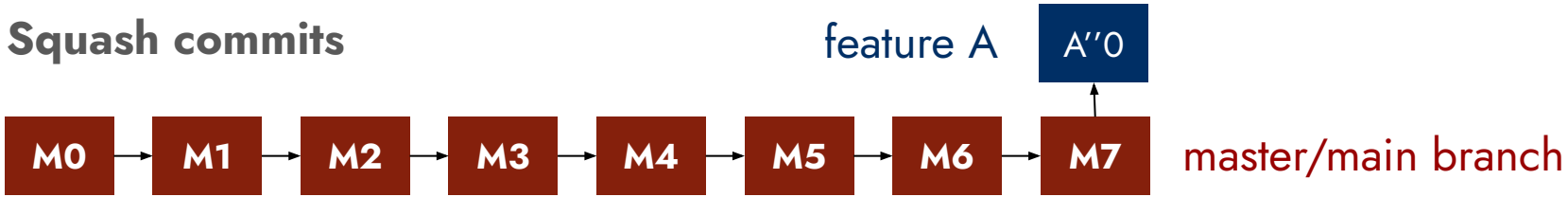
https://git-scm.com/book/en/v2

Merging branch into master

Good practice: do a **merge request**; allows comparing the code, new code reviewed and approved by collaborators.



Squash commits



Feature merged.

Tasks for today

- **Create a repository** for your project on Github or Gitlab,
- Clone the repository, update README.md and create a .gitignore file containing *.swp, build/ and *egg.info/
- Create a **basic project structure** with a pyproject.toml (see example here) and a new folder called **name/** (replace name with the name of your project)
- Add, commit, and push these changes to your main branch.
- Create a **new branch** called **feature/iodata** and implement a Python module which can read/write NumPy arrays to/from a checkpoint file.
- **Push** the new branch to your repository with a **descriptive commit message** following commit conventions.
- **Make a merge request** to merge the new branch into master/main.

Command to build and install based on **pyproject.toml**

```
python3 -m pip install --no-build-isolation -v .
```

Homework

- Add me (username **iubr**) as a collaborator with the role “developer” to your Github repositories.

Next time we will:

- Discuss in more details how to structure a Python project.

Part 3: Projects

Project: Develop your own Python software

Project evaluation:

1. Github or Gitlab **repository**:
    - Repository should contain the following components:
        - README with installation instructions,
        - Proper project structure,
        - Python tests and script/notebook examples,
        - Software manual,
        - One C/C++ class which is exposed to Python.
    - I should be able to install and run your software,
    - The code should follow Python coding conventions,
    - The Git commits should follow (more or less) a set of commit conventions.

2. Short **project presentation**:
    - 2026/02/02
    - 10 min. Jupyter notebook demonstration of your software

Project: Develop your own Python software

You can either choose to work on:

1. A Python project which you **design yourself,**
2. One of the Python projects **suggested** by me.

If you would like to work on **your own idea**, try to define a:

- Self-contained project,
- Useful for your (PhD) work,
- Not too complicated.

Project: Develop your own Python software

By/on 13 November:

- Decide which project you would like to do:
  - Select from my list of projects (list to be posted on course website),
  - Design your own.
- During the **class on 13 November**, we'll:
  - Set the projects,
  - Discuss the corresponding code structure.

Project ideas on the course website:

https://sites.google.com/view/python3-umk/projects

Thank you for today!

Questions?