Lecture II — Version Control with Git

AI-Assisted Programming for PhD Researchers

Dr. Tobias Vlćek
Helmut Schmidt University — September 2026

Why Git

Code Now Changes Fast

- An agent can rewrite half your project in a minute
- That is genuinely useful, and genuinely dangerous
- Without a history, you cannot **see** what changed

. . .

- You cannot **judge** it, and you cannot **undo** it

What Git Gives You

- A **time machine**: undo any change, back to any point
- A **microscope**: see exactly what changed, line by line
- A **safety net**: branch off and experiment without fear
- A **lab notebook**: a dated, labeled record of every step

Git and Reproducibility

- "Which version of the code produced Figure 3?"
- A commit hash answers that — exactly, forever [1]
- Version control is a core skill for reproducible research [2]
- Journals and supervisors increasingly ask you to prove it

Setup

Install Git

Pick the line for your system:

```
# macOS – installs with the developer tools
xcode-select --install       # or: brew install git

# Windows – download the installer, accept defaults
winget install Git.Git       # from git-scm.com

# Linux – use your distribution's package manager
sudo apt install git         # Debian / Ubuntu
```

Your Turn — Verify and Introduce Yourself

Your turn: confirm Git works, then tell it who you are.

```
git --version
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

> **i** Note
>
> Use the same email you will use on GitHub — it links your commits to your account.

Your Turn — Check Your Python Toolchain

Your turn: confirm `uv` is installed from setup.

```
uv --version
```

Missed the setup step? This one command fixes it now:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Every lab runs Python through `uv run` — nothing else to configure.

Repositories

What Is a Repository?

- A folder whose **full history** lives in a hidden `.git` directory
- Every version of every file, back to the first commit
- It is entirely **local** — on your machine, in your control
- Nothing leaves your computer unless you explicitly push it

Getting One: init or clone

- `git init` — start tracking a **new** folder of your own
- `git clone URL` — copy an **existing** repository to your machine

```
git init                    # new project, here
git clone URL               # existing project, from a remote
```

Your Turn — Clone the Course Project

Your turn: get the starter project and look around.

```
git clone https://github.com/beyondsimulations/aiprog-lab-starter
cd aiprog-lab-starter
ls
git log --oneline
```

The `log` output is the project's history — every commit before you arrived.

What You Just Inherited

- A **data set** and an **analysis script** from a previous PhD student
- It will **not run yet**: dependencies, paths, and assumptions are stale
- That is not a mistake; it is the normal starting point
- Fixing inherited code is tomorrow's reality, and today's material

The Everyday Loop

Status, Add, Commit

- `git status`: what have I changed?
- `git add`: put changes in the staging area
- `git commit`: record what is staged, with a message

. . .

The staging area is a **shopping cart**: you fill it deliberately, then check out.

Anatomy of a Good Commit

- **One logical change** per commit, not a day's worth of work
- Message in the **imperative**, and it says **why**

. . .

```
bad:   git commit -m "stuff"
good:  git commit -m "Fix humidity parsing for n/a values"
```

Your Turn — First Commit

Your turn: make your first commit in the starter repo.

```
# create notes.md with one line about the project's state
git add notes.md
git commit -m "Add project state notes"
git log --oneline
```

Your commit is now the newest line in the history.

Partial Staging

- `git add -p` walks you through changes **hunk by hunk**
- Say yes or no to each piece before it is staged
- This is exactly how you accept an agent's diff **selectively**

. . .

- Take a good change, drop a bad one — review at staging time

Looking at History

log — What Happened

- `git log --oneline --graph` — compact history with branch structure

```
git log --oneline --graph
```

- Each line is one commit; each hash is a **citable state** of your project
- Point at a hash and you point at an exact, reproducible version

diff — What Changed

- `git diff`: changes you have **not** staged yet
- `git diff --staged`: changes about to go into your next commit
- `git show HEAD`: the full content of the **last** commit

. . .

This is **the** tool for reviewing what an agent actually did.

Your Turn — Inspect

Your turn: read the starter repo's history.

```
git log --oneline --graph
git show --stat HEAD
```

Scroll the log and find the commit where the **data set** was first added.

Undoing

Three Levels of Undo

Match the command to how far you want to go back:

Command	Undoes
`git restore <file>`	An unstaged change in your working file
`git restore --staged <file>`	Staging — keeps the edit, unstages it
`git reset --soft HEAD~1`	The last commit — keeps its changes staged

Your Turn — Break and Restore

Your turn: break a file on purpose, then bring it back.

```
# in your editor, delete half of legacy_analysis.py and save
git diff                      # confirm the damage
git restore legacy_analysis.py
git diff                      # clean again - nothing to show
```

Nothing is scary while your last commit is safe.

The One Safety Rule

> ⚠ Warning
>
> Never rewrite history you have already **pushed or shared** — others may have built on it. Everything still **local and unpushed** is yours to rewrite freely.

Branches

Why Branches (Especially with Agents)

- A branch is an **isolated line of work** off your main history
- Let the agent go wild on a branch — `main` stays clean
- Experiment fails? Delete the branch; `main` never knew
- Merge back **only** what you have reviewed and understood

Branch Basics

```
git switch -c try-idea        # create a branch and move onto it
# ... make changes and commit them ...
git switch main               # go back to main
git merge try-idea            # bring the work in
```

Four commands: branch, work, return, merge.

Your Turn — Branch and Merge

Your turn: run the full branch cycle once.

```
git switch -c add-readme-note
# edit README, then:
git add README.md && git commit -m "Add note to README"
git switch main
git merge add-readme-note
git branch -d add-readme-note
```

Tags and Checkpoints

Tags Mark Moments

- The labs tag reference states — clean checkpoints you can jump to
- Fell behind or broke something? Jump to a known-good tag:

```
git fetch --tags
git checkout tags/lab-01-done -b catchup
```

No shame in catching up — the goal is the **next** exercise, not pride.

Remotes (Just Enough)

GitHub in One Slide

- A **remote** is a copy of your repository somewhere else (e.g. GitHub)
- `git push` sends your commits up; `git pull` brings others' down
- You will push your capstone if you want feedback on it — the cheatsheet has the exact push and pull commands

Continue Your Journey

Next Up

- **Lunch**, then your first agent session at 13:30
- Lecture III — First Steps with an Agent
- Git cheatsheet — every command from today, one page
- Course literature and references

Bibliography

[1] K. Ram, "Git can facilitate greater reproducibility and increased transparency in science," *Source Code for Biology and Medicine*, vol. 8, no. 1, p. 7, Dec. 2013, doi: 10.1186/1751-0473-8-7.

[2] J. Bryan, "Excuse Me, Do You Have a Moment to Talk About Version Control?," *The American Statistician*, vol. 72, no. 1, pp. 20–27, Jan. 2018, doi: 10.1080/00031305.2017.1399928.