🔬 Scientific Programming with Python 🐍

Welcome to the Course!

Session 1

Introduction & Set Up

👥 Meet the Teaching Assistants

- **Konark** — 2nd Year, BSc Computer Science
- **Amirali** — 2nd Year, BSc Software, Data & Technology

📚 How the Course Works

- **Lectures** → delivered by the professor
- **Seminars**:
  - Warm-up problems in Jupyter — solved together
  - Core exercises — to be completed and submitted in class
- **Weekly Homework** → due before the next lecture

📝 Grading Scheme

- **67%** → Final written exam
- **33%** → Weekly homework

⚠️ To pass: you must score **at least 45% in each component**
(exam, weekly homework, and seminar core exercises)

💬 Communication Channels

- **Microsoft Teams**
  - Course channel: ask general questions, share problems others may also encounter
  - Private messages: for detailed or personal matters

🗓️ Today's Plan

- Install **Python**
- Set up **Git & GitHub Classroom**
- Submit your very first assignment → **Homework 0**

🚀 Let's get started!

Install Python

- Download from python.org/downloads
- Choose **Python 3.11+** (stable version)
- On Windows → ✅ check *"Add Python to PATH"* during install
- On macOS/Linux → often pre-installed. If missing or outdated:

macOS (with Homebrew):

```
brew install python
```

Ubuntu/Debian:

```
sudo apt update
sudo apt install python3 python3-pip -y
```

🛠️ Verify Installation

Check your version in the terminal:

```
python --version
```

✅ Expected output:

```
Python 3.x.x
```

📂 Virtual Environments (Advanced)

Keep project dependencies isolated:

```
python -m venv .venv
source .venv/bin/activate   # macOS/Linux
.venv\Scripts\activate      # Windows
```

Deactivate with:

```
deactivate
```

📦 Install Packages (coming soon...) with pip

Use `pip` to add dependencies:

```
pip install pytest
```

Install all dependencies from **requirements.txt**:

```
pip install -r requirements.txt
```

⚠️ Not working for some? The problem is 3️⃣!

- On **Linux/macOS**, older systems shipped with **Python 2** as `python`.
  - To avoid conflicts, **Python 3 is installed as** **`python3`**.
- On **Windows**, you usually only have Python 3, so the command is just **`python`**.
- In modern Linux/macOS, `python` may already point to Python 3, but it's safest to use `python3`.

👉 Same goes for **pip**

👉 **Rule of thumb:**

- macOS/Linux → use `python3` & `pip3`
- Windows → use `python` & `pip`

🧪 Test Your Setup

Run the Python REPL:

```
python
```

```
>>> print("Hello, world!")
```

🎉 If it works → you're ready to code!

🧭 Python Command Cheat Sheet

- **Run script:** `python file.py`
- **Interactive mode (REPL):** `python`
- **Install package:** `pip install <name>`
- **Install from requirements:** `pip install -r requirements.txt`
- **Virtual env:**
  - Create: `python -m venv .venv`
  - Activate: `.venv\Scripts\activate` (macOS/Linux → `source .venv/bin/activate`)
  - Deactivate: `deactivate`

⚠️ Add **3** for macOS/Linux

🐍 ✅ Python is Ready!

- You’ve installed Python
- Verified it runs on your system
- Know how to manage packages & virtual environments

🐙 Next Step → Git & GitHub 🚀

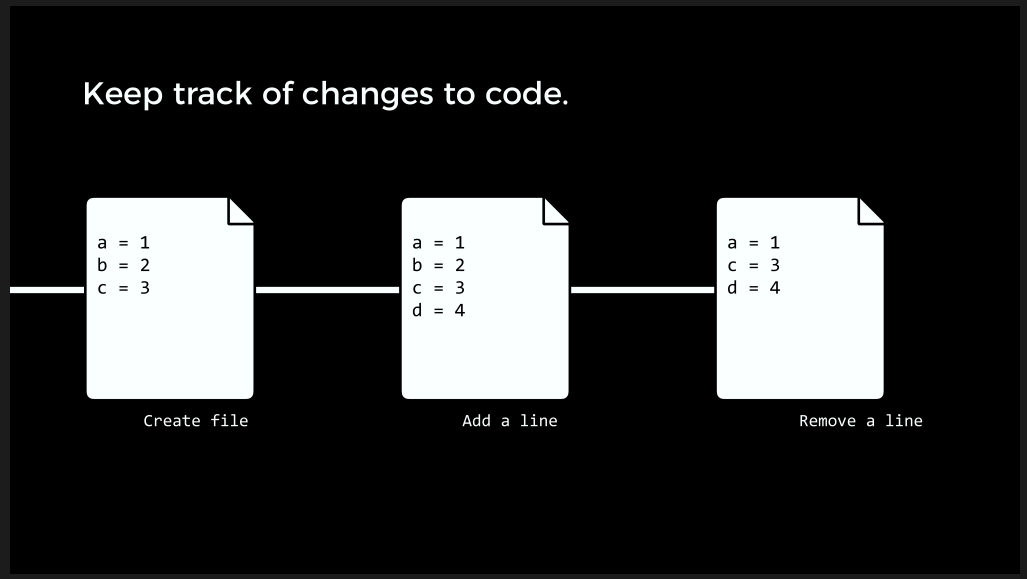
- Manage your projects with **version control**
- Collaborate with peers
- Submit homework via **GitHub Classroom**

✨ Let’s set up Git & GitHub together! ✨

🐙 Git & GitHub — Quick Introduction

A minimal toolkit to collaborate and track changes:

- Local work tracked by **Git**
- Shared remotely on **GitHub**
- Sync via **clone / pull / add / commit / push**



1 Install Git

- Download & install from git-scm.com/downloads
- Verify installation:

```
git --version
```

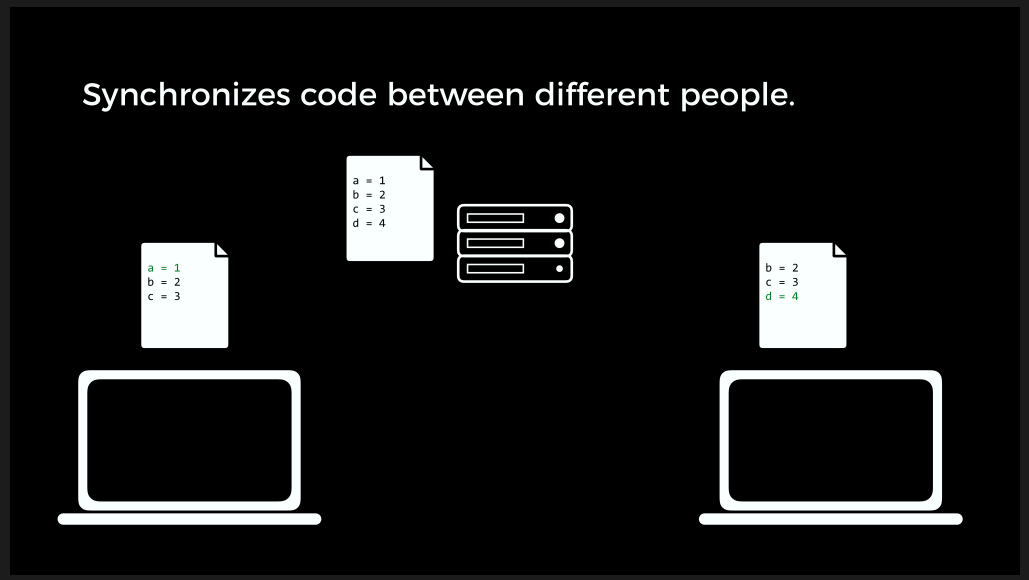
2 GitHub Setup

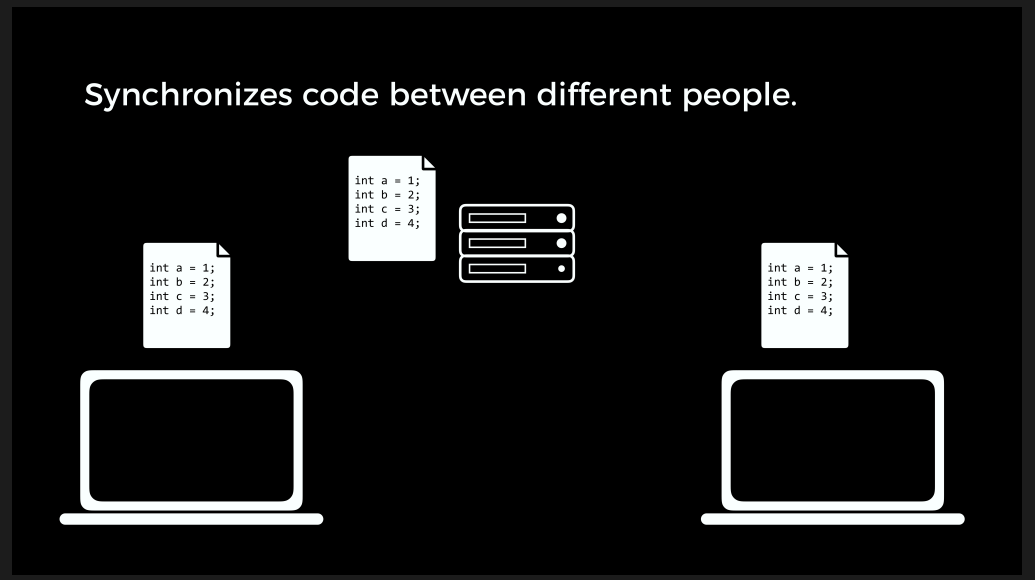
- Create an account on github.com
- Configure identity locally:

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

3️⃣ Repositories (Local ↔️ Remote)

- A **repository** = a project folder tracked by Git
- Two sides of the same repo:
  - **Local** on your computer
  - **Remote** on GitHub (the server)
- Collaborate by syncing changes through the remote





4 GitHub (the Server)

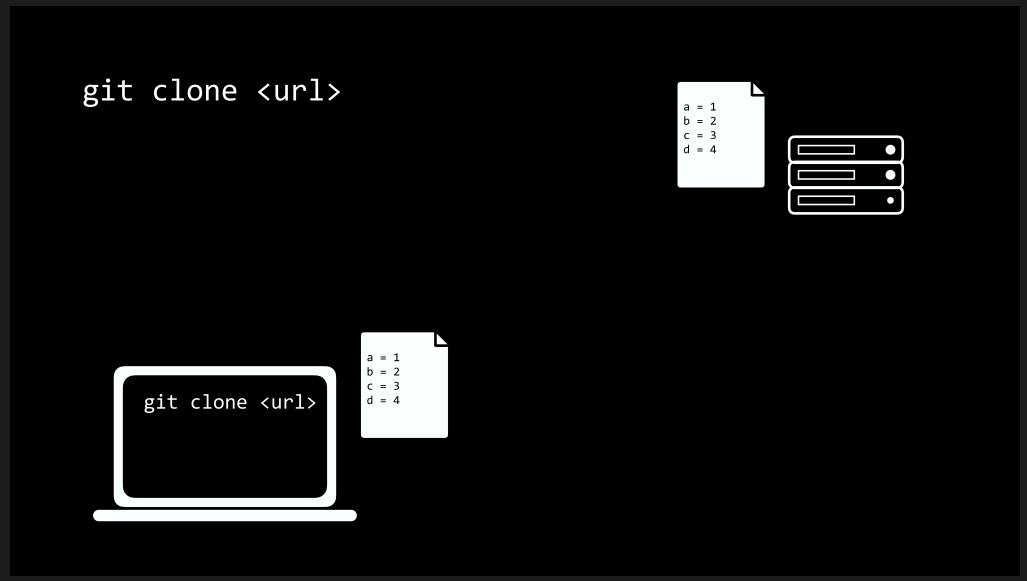
- Hosts repositories, issues, pull requests, CI, and collaboration tools
- Central place to share code and review contributions

5 Clone a Repository

Copy a remote repo to your machine:

```
git clone <url>
```

- Creates a local folder with full commit history



6 Stage Changes

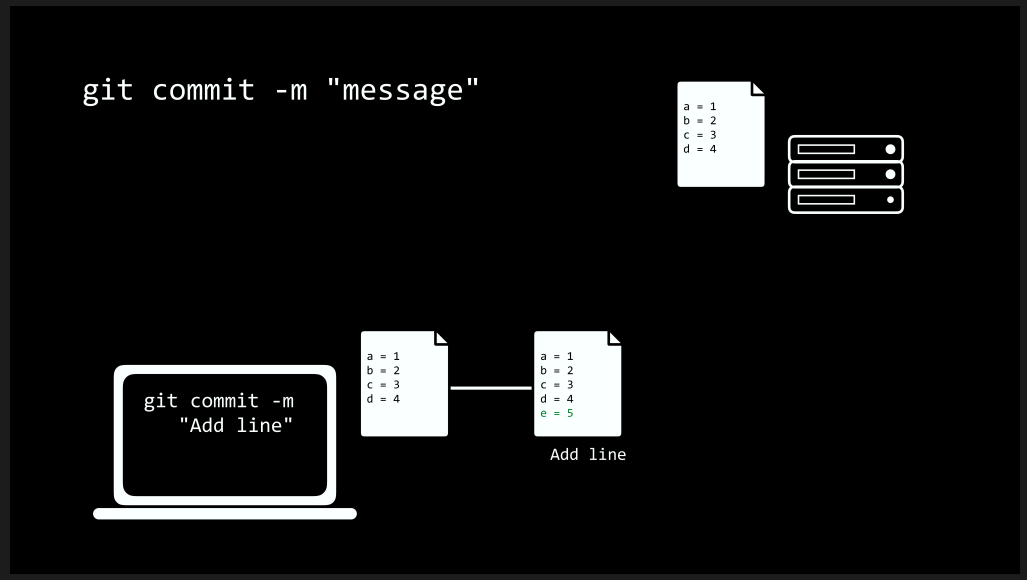
Select what to include in the next commit:

```
git add <filename>
# or
git add .
```

7 Commit Changes

Save a snapshot with a clear message:

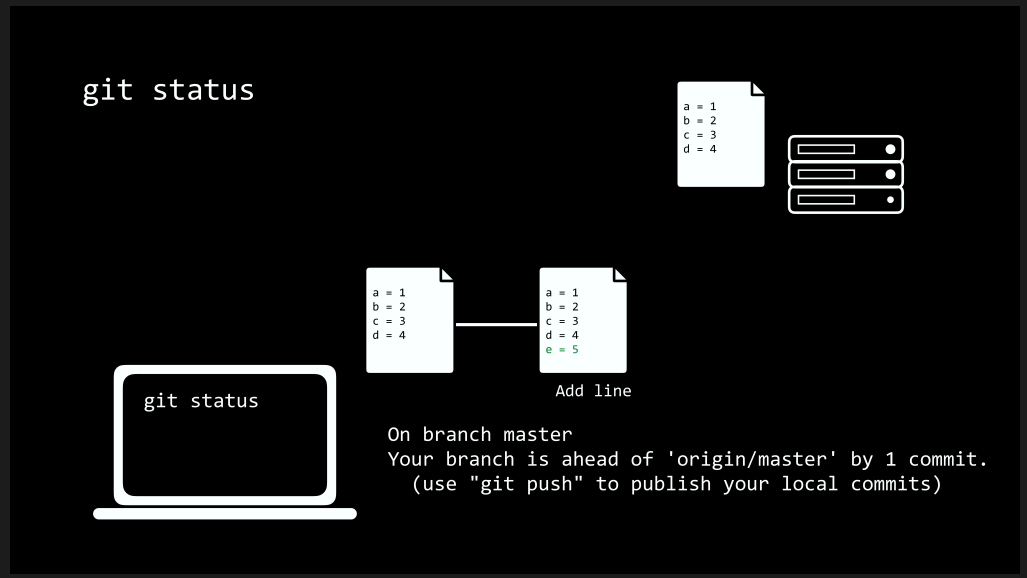
```
git commit -m "Describe what changed"
```



8 Check Status

See what's modified, staged, or untracked:

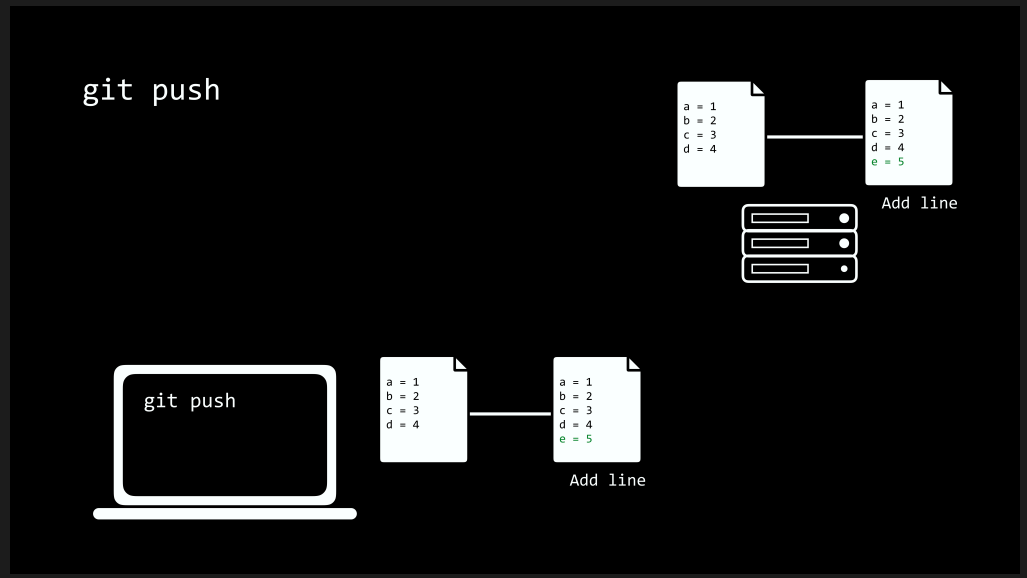
```
git status
```



9 Push to GitHub

Upload your local commits to the remote:

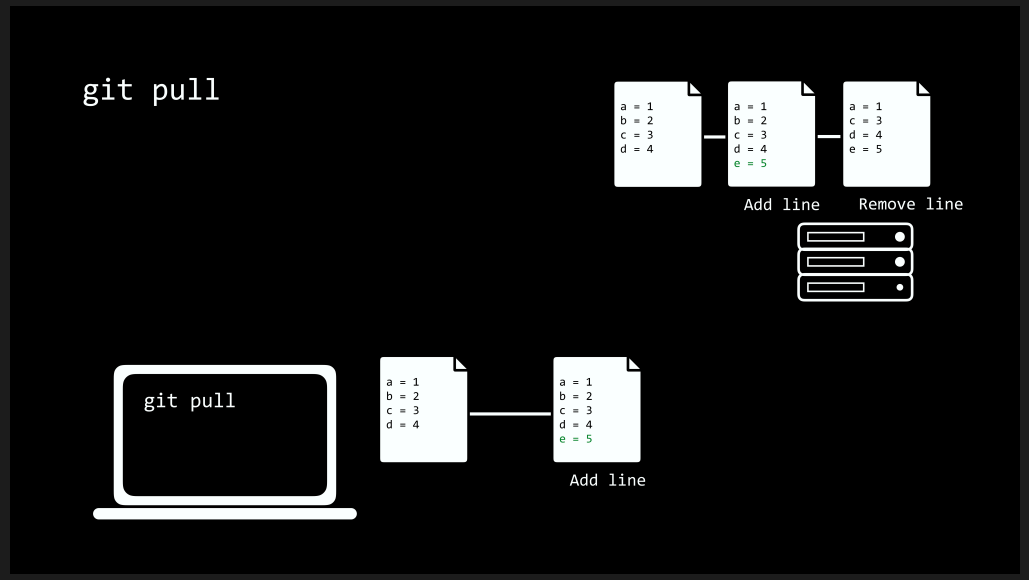
```
git push
# on the first push, Git may ask you to set an upstream branch
git push --set-upstream origin <branch>
# or shorter:
git push -u origin <branch>
```

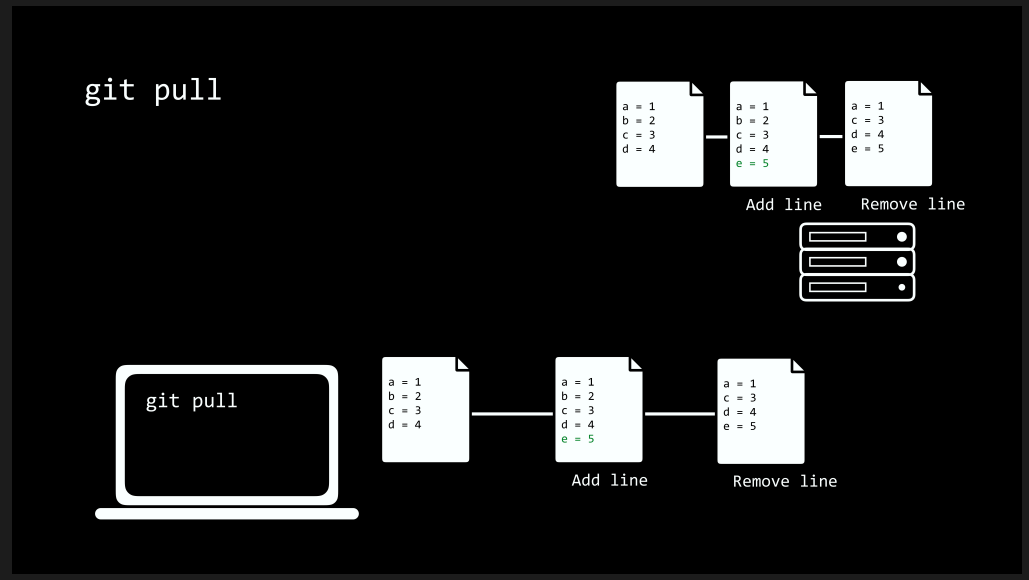


🔟 Pull from GitHub

Download and integrate remote changes:

```
git pull
```





1️⃣1️⃣ View History

Inspect past commits:

```
git log
```

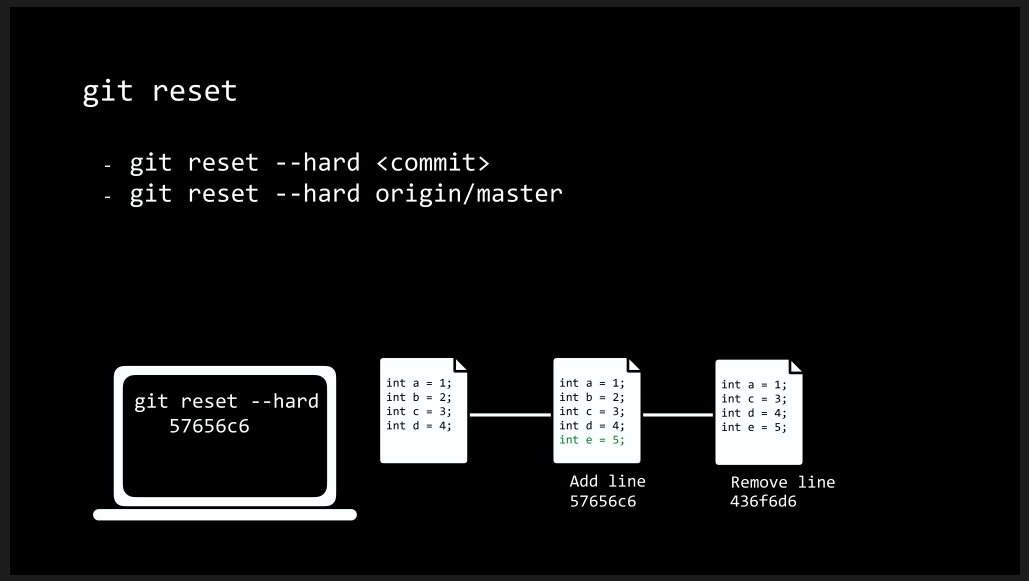
Useful flags: `--oneline`, `--graph`, `--decorate`

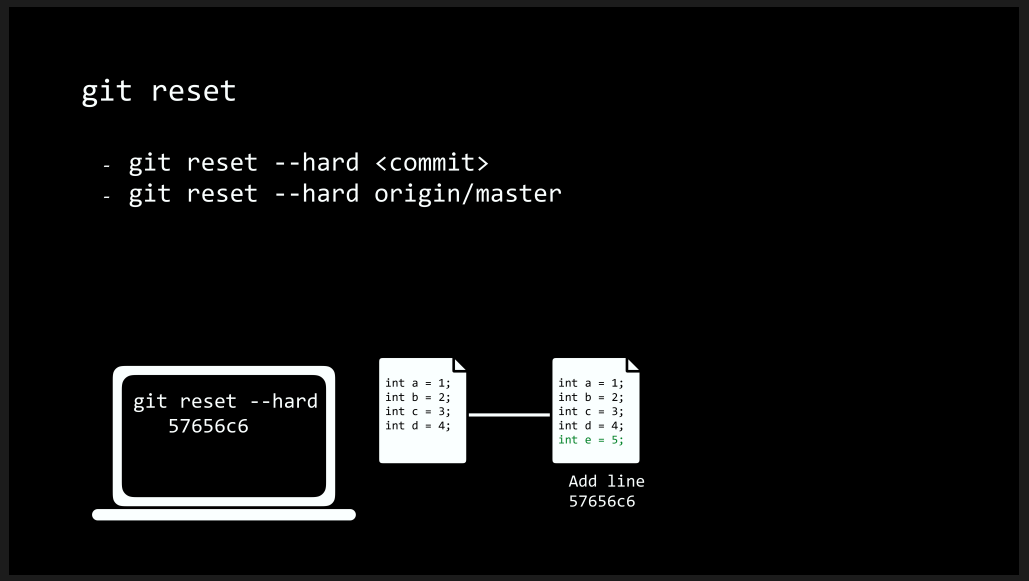
1️⃣2️⃣ Reset (HARD) — Use with Caution

⚠️ **Dangerous:** discards local changes after the target commit.

```
git reset --hard <commit>
```

- Resets working directory and index to the given commit
- Irreversible if changes weren't pushed or saved elsewhere





Git Command Cheat Sheet

- **Setup:** `git config --global user.name` , `user.email`
- **Clone:** `git clone <url>`
- **Status:** `git status`
- **Stage:** `git add <file>` / `git add .`
- **Commit:** `git commit -m "<message>"`
- **Push/Pull:** `git push` / `git pull`
- **History:** `git log`
- **Reset (danger):** `git reset --hard <commit>`

✅ Wrap-Up

- You know the course structure & grading
- You can install Git, configure GitHub, and use core commands
- Next: **HW0 submission via GitHub Classroom**

? Any Questions?

- Feel free to ask about Git basics
- Setup issues? We can help you fix them now
- No question is too small — if you’re stuck, probably others are too :)

🚀 Let’s Get Started with HW0

- We’ll **submit HW0 together right now**
- Feel free to **ask your peers** if you get stuck
- **Pair programming** is always better 👩‍💻👨‍💻
- And of course, don’t hesitate to **ask us** for help

🎓 GitHub Classroom — Step by Step

What it is:

- You accept an **assignment link** (shared on Teams).
- GitHub Classroom creates a **private repo** for you.
- You work locally and **push** to submit. You can push updates until the deadline.

1 Accept the Assignment

- Open the **Classroom link** from Teams.
- Sign in to **GitHub** (or create an account).
- Wait for the private repo to be created (name usually includes your username).
- Confirm the repo is **private** and belongs to the course organization.

2️⃣ Clone Your Repo

Use HTTPS (recommended for beginners):

```
git clone <repo-url>
cd <repo-name>
```

📂 Project Structure — Do Not Modify

- **.gitignore** → handles which files are ignored by Git
  (e.g., IDE files, caches, virtual environments)
- **requirements.txt** → lists dependencies needed for grading
  ⚠️ Do not edit or remove
- **tests/** & **.github/** → contains autograder test cases
  ⚠️ Never change or delete, otherwise grading will fail

👉 Only work inside the provided source files

Install Dependencies →

```
pip install -r requirements.txt
```

🔧 Test Your Code Before Committing

Always run the tests to make sure your code works

```
pytest
#or
pytest tests/task_name.py # to run tests for a specific task
```

- Fix errors before committing
- Green tests ✅ → safe to `git add` and `git commit`

3 Do the Work Locally

- Open the project in your editor.
- Implement your solutions, run tests.

Stage and commit your changes:

```
git add <files> # or: git add .
git commit -m "HW0: implemented"
```

4 Push to Submit

Push your commits to GitHub:

```
git push
# if Git asks for an upstream on first push:
git push --set-upstream origin <branch>
# shorter:
git push -u origin <branch>
```

🔍 Verify Your Submission

- Go to your repository on GitHub
- Open the **Pull Requests (PRs)** tab
- Check the feedback from the autograder
- If the checks pass ✅ → you're done
- If they fail ❌ → fix your code locally, commit, and push again (the PR updates automatically)

🧭 GitHub Classroom Cheat Sheet

1️⃣ **Accept Assignment** (Teams link → private repo)
2️⃣ **Clone Repo** → `git clone <url>` → `cd <repo>`
3️⃣ **Install Dependencies** →

```
pip install -r requirements.txt
```

4️⃣ **Work Locally** → edit code, run `pytest`
5️⃣ **Commit** → `git add .` → `git commit -m "HWx: solution"`
6️⃣ **Push** → `git push`
7️⃣ **Check GitHub** → PR shows autograder

- ✅ Green → done
- ❌ Red → fix, commit & push again

✨ Follow this order for every HW submission ✨

🎉 All Set Up!

- ✅ Python installed & working
- ✅ Virtual environment ready (don't worry about it UwU)
- ✅ Git & GitHub configured
- ✅ GitHub Classroom workflow clear

✨ You’re ready to start coding! ✨

📢 Homework 1

- We will now share the **HW1 link**
- Deadline: **before next week's lecture**
- Submit via GitHub Classroom (same process as HW0)

🙌 Thank You!

- That’s all for today
- Good luck with **HW1**
- See you in the next session 👋