DIS 3.3 · DATA INTEGRATION

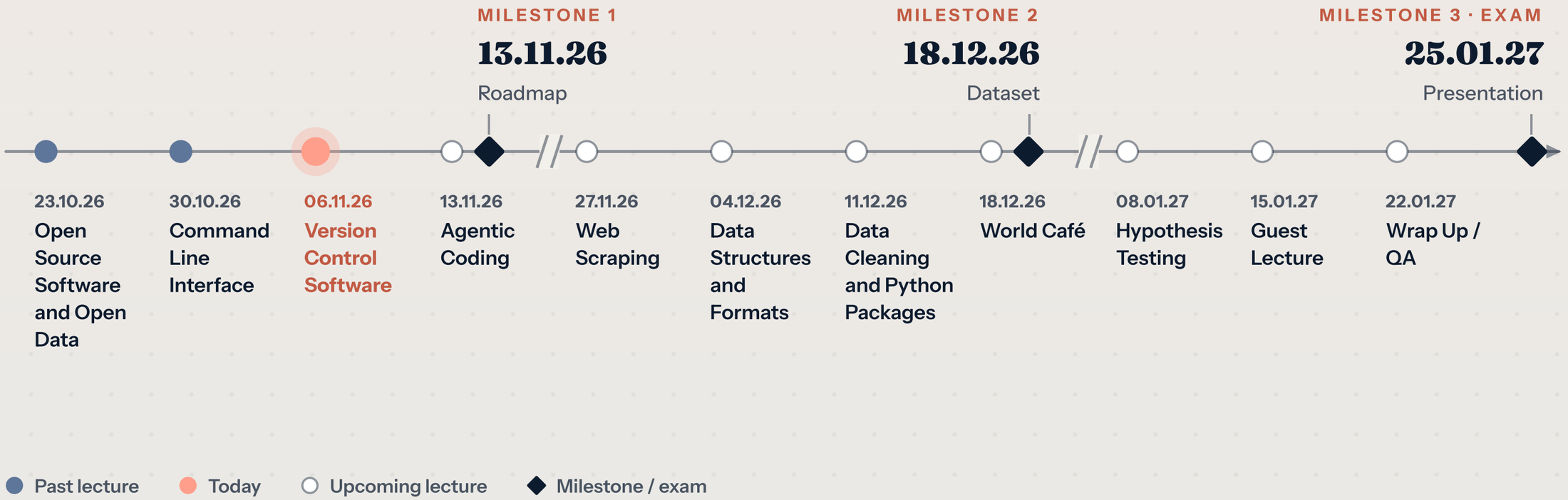
Version control.

Version control software with Git

Winter 2026/27 · Timo Breuer · TH Köln

```
$ git init
Initialized empty Git repository
$ git add README.md
$ git commit -m "Create README.md"
[main (root-commit) aaa829b] Create README.md
 1 file changed, 1 insertion(+)
$ git log --oneline
aaa829b (HEAD -> main) Create README.md
```

Timeline


MILESTONE 1
13.11.26
Roadmap
MILESTONE 2
18.12.26
Dataset
MILESTONE 3 · EXAM
25.01.27
Presentation
23.10.26
Open Source Software and Open Data
30.10.26
Command Line Interface
06.11.26
Version Control Software
13.11.26
Agentic Coding
27.11.26
Web Scraping
04.12.26
Data Structures and Formats
11.12.26
Data Cleaning and Python Packages
18.12.26
World Café
08.01.27
Hypothesis Testing
15.01.27
Guest Lecture
22.01.27
Wrap Up / QA
Past lecture
Today
Upcoming lecture
Milestone / exam

What is version control and why use it?

> **A system that records changes to a file or set of files over time, allowing you to recall specific versions.**

KEY FEATURES

Backup & restore

Track changes in code or documents.

Collaboration

Collaborate with multiple team members.

Version history

Revert to previous versions if mistakes are made.

Common tools Git SVN Mercurial

What is version control and why use it?

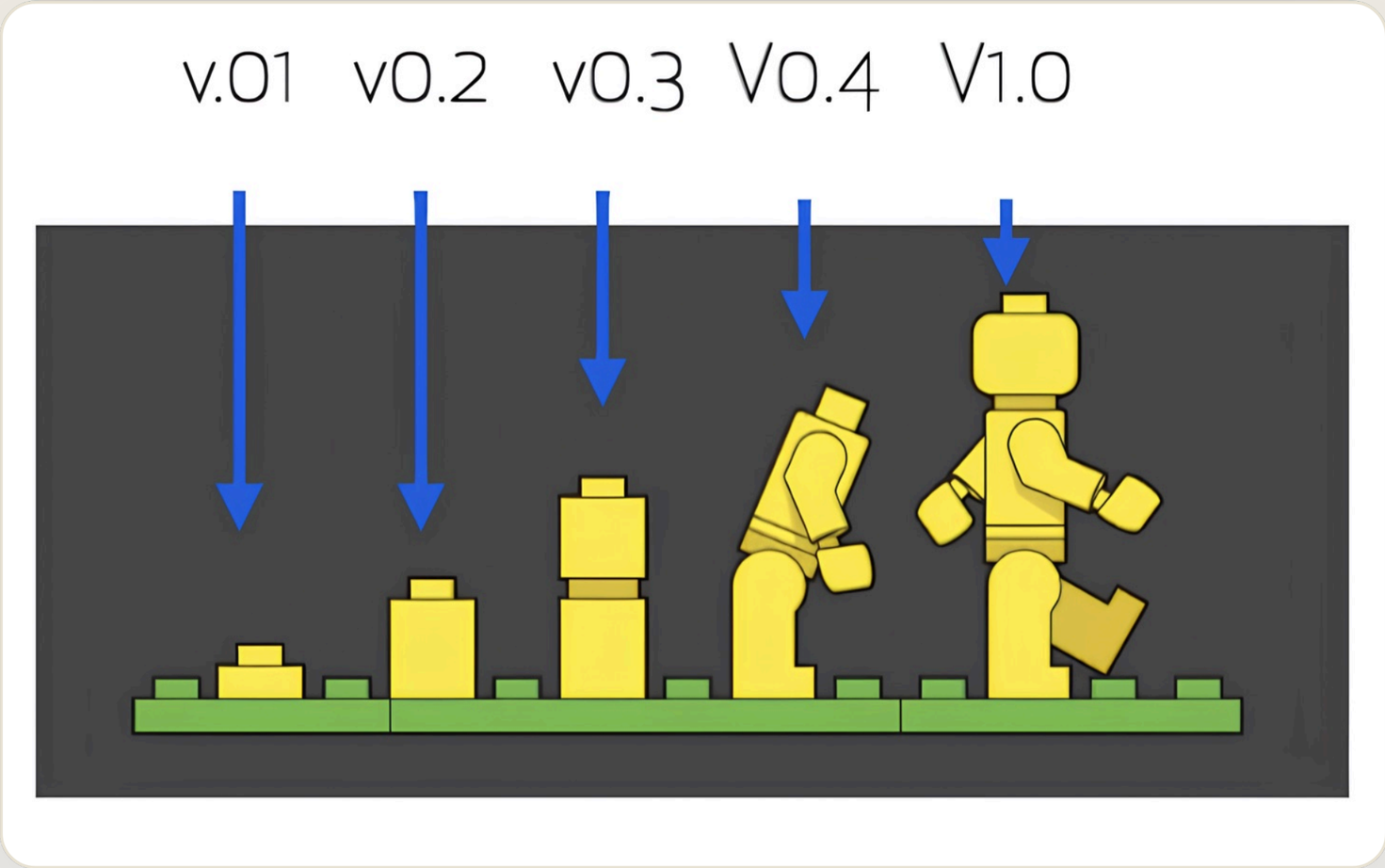
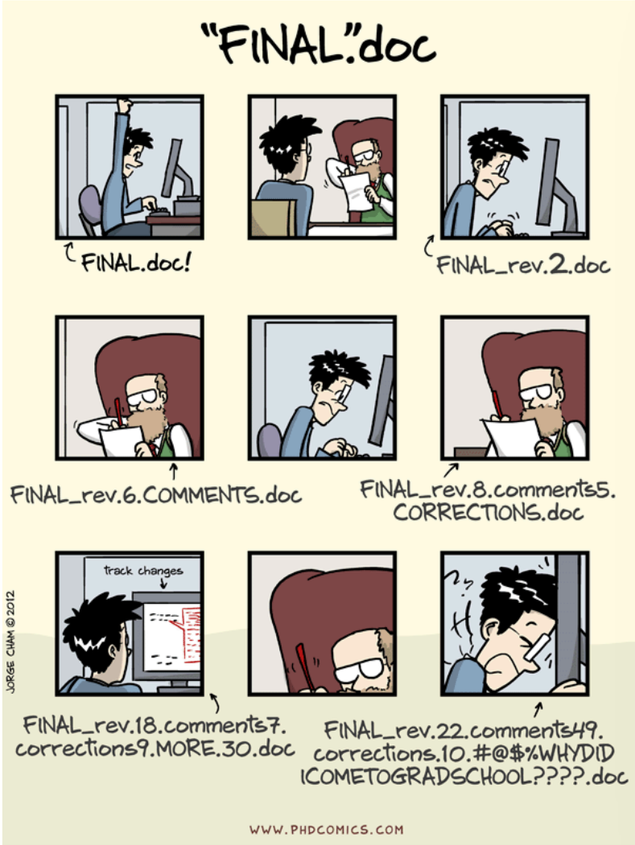


Image source: github.com/education

Life without version control



Name
Super Cool Report v1.xlsx
Super Cool Report v2.xlsx
Super Cool Report v3.1.xlsx
Super Cool Report v3.xlsx
Super Cool Report v4.xlsx
Super Cool Report v4a.xlsx
Super Cool Report v4b.xlsx
Super Cool Report v5.xlsx
Super Cool Report vFinal.xlsx
Super Cool Report vFinal_1.xlsx
Super Cool Report vFinal_2.xlsx
Super Cool Report vFinal_Final.xlsx
Super Cool Report vFinal_Final-UPDATED.xlsx
Super Cool Report vFinal_Final-UPDATED_NEW.xlsx

Which one is the final version?

Without version control, versions pile up as copies with ever-longer names.

Image sources: phdcomics.com/comics/archive.php?comicid=1531 · reddit.com/r/ProgrammerHumor/comments/oz4pfb/version_controlcan_anyone_relate/

Scope of today's lecture and roadmap

01
Definition of Git

02
Basic concepts

03
Common workflow and example

04
Graphs

05
Branching & merging

06
History & undo

07
Remote repositories

08
Best practices

What is Git?

> **Git is a distributed version control system that tracks changes in files and coordinates work on those files among multiple people.**

KEY FEATURES

- Distributed
- Version control
- Collaboration tool

BENEFITS

- Tracks changes over time
- Easy collaboration: branching and merging capabilities
- Backup and recovery

Do not confuse Git with GitHub!

GitHub is a hosting platform for Git, owned by Microsoft. Git was created by Linus Torvalds in 2005 for the Linux kernel.

Basic Git concepts and terminology

Term	Meaning
Repository	A storage space where your project resides, including all files and history.
Commit	A snapshot of your project at a specific time.
Branch	A separate line of development.
Merge	Combining branches together.
Clone	Copying a repository.
Pull / Push	Fetching changes from a remote repository / sending changes to a remote repository.
Fork	A personal copy of someone else's repository, often used in open-source projects.

Term	Meaning
Staging area	A space where changes are prepared before committing.
Revert	Undo changes by creating a new commit that negates a previous one.
Rebase	Replay your commits on top of another commit, giving a linear history.
Conflict	Occurs when changes from different commits or branches clash, needing manual resolution.
Tag	A label marking specific commits, often used for releases.
HEAD	A pointer to what you have checked out — usually the tip of your current branch.
Checkout	Switch between branches or specific commits.

Git is like a desk

Working directory
where you write



Staging area
rough draft, in a manila folder

Repository
final draft in the filing cabinet

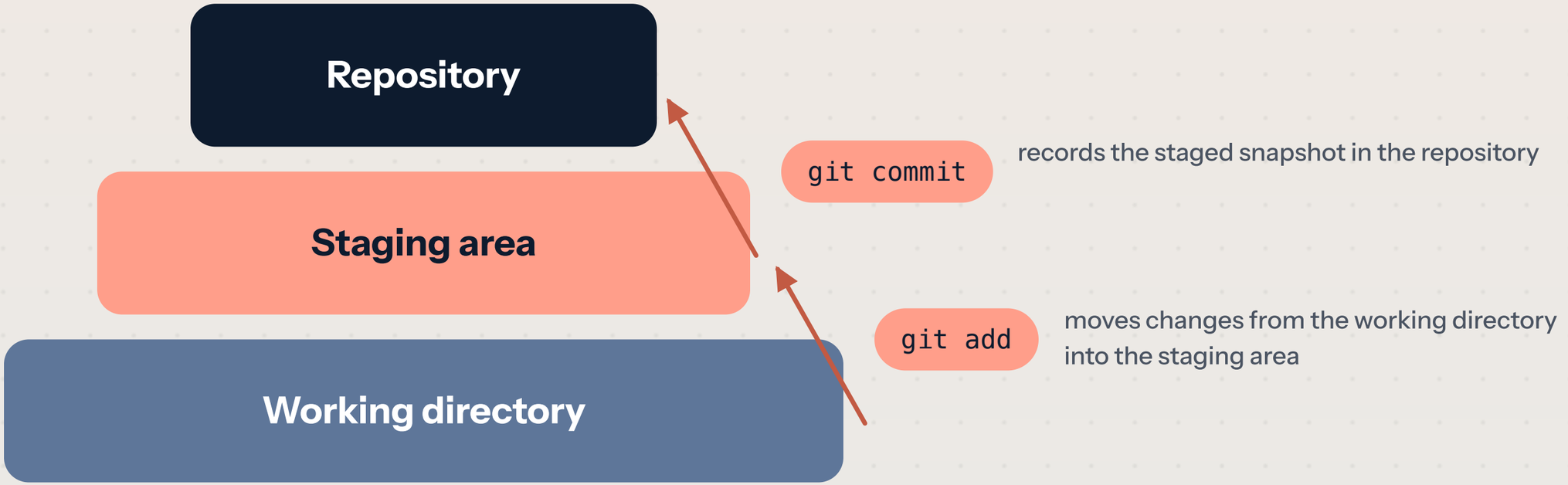
Image source: github.com/education

Git is like a desk



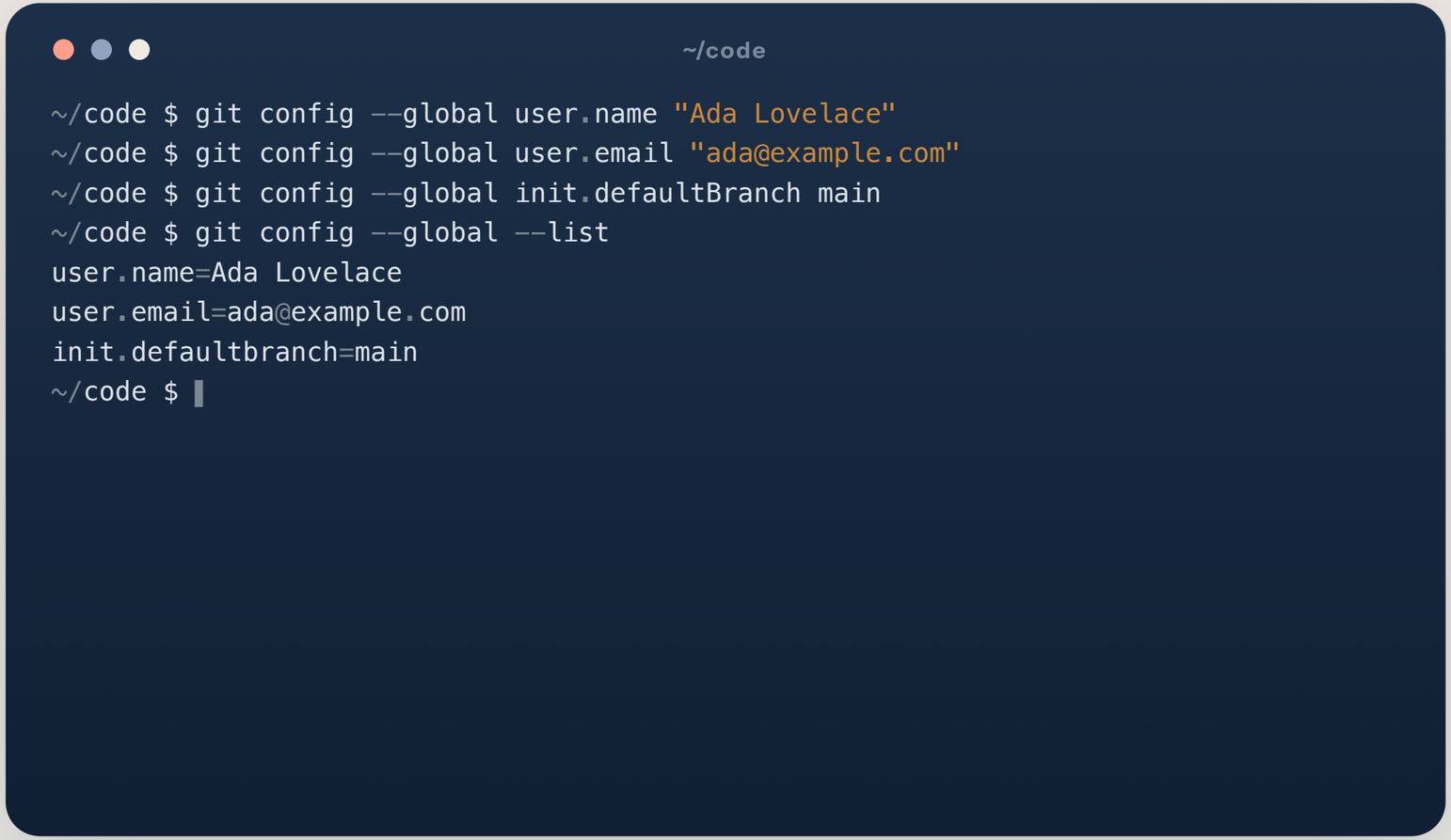
After: github.com/education

Git is like a desk



After: github.com/education

Before you start: configure Git once



Name & e-mail

Stored in every commit. Use the e-mail of your GitHub account so your commits are linked to your profile.

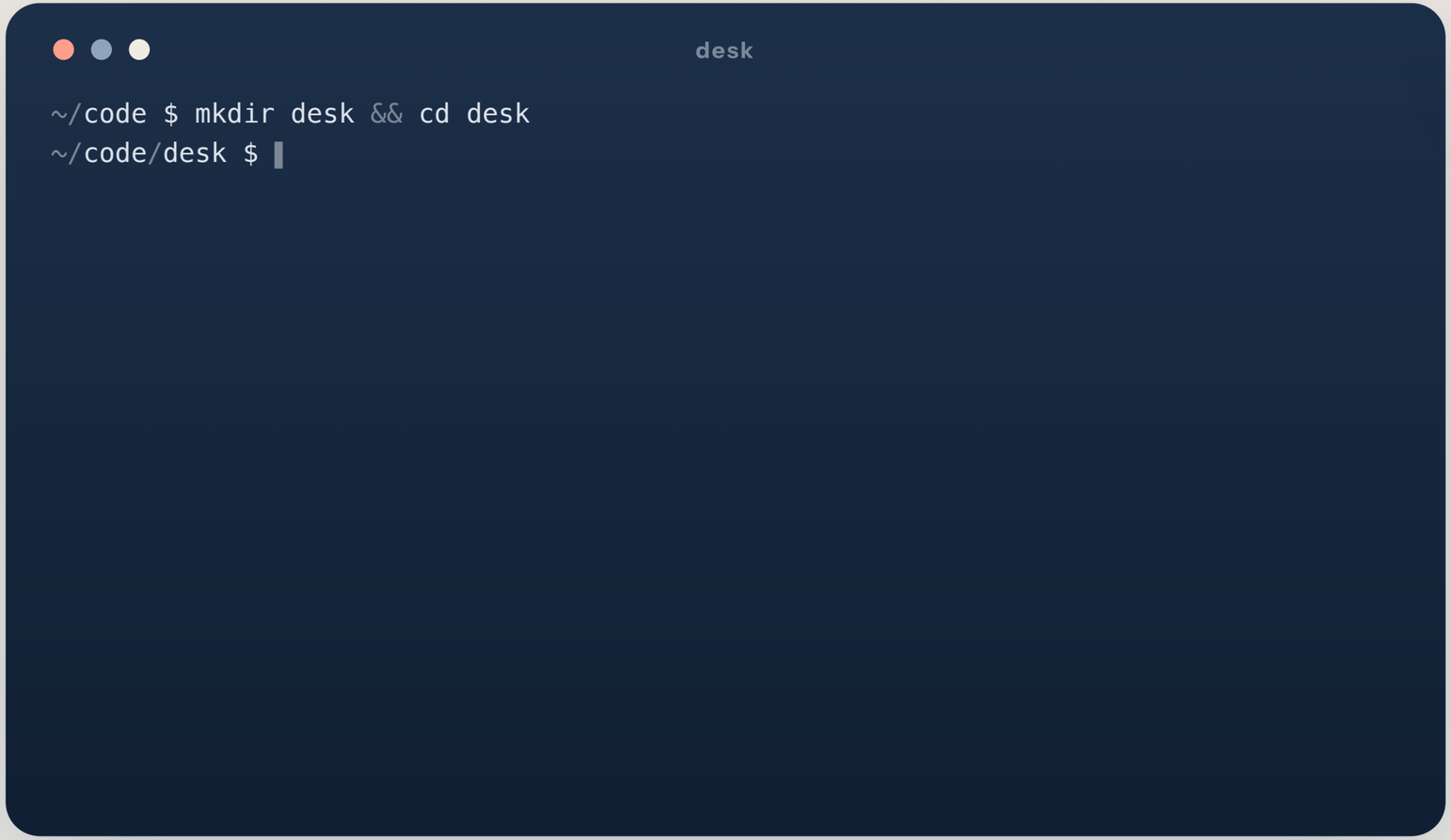
Default branch

Name the first branch `main`. Older Git versions still default to `master`.

Once per computer

`--global` saves the settings in `~/.gitconfig` for all your repositories.

A first example




```
01  mkdir desk && cd desk
02  git --version
03  git init
04  echo "…" > README.md
05  git status
06  git add README.md
07  git commit -m "…"
```

Create a project folder and move into it.

A first example: verify installation




```
01  mkdir desk && cd desk
02  git --version
03  git init
04  echo "…" > README.md
05  git status
06  git add README.md
07  git commit -m "…"
```

Check that Git is installed.

A first example: initialize a repository




```
01  mkdir desk && cd desk
02  git --version
03  git init
04  echo "…" > README.md
05  git status
06  git add README.md
07  git commit -m "…"
```

Turn the folder into a Git repository. Git creates a hidden **.git** folder.

A first example: create a file




```
01  mkdir desk && cd desk
02  git --version
03  git init
04  echo "…" > README.md
05  git status
06  git add README.md
07  git commit -m "…"
```

Create a first file in the working directory.

A first example: check the status

desk

```
~/code $ mkdir desk && cd desk
~/code/desk $ git --version
git version 2.39.5 (Apple Git-154)
~/code/desk $ git init
Initialized empty Git repository in /Users/timo/code/desk/.git/
~/code/desk (main) $ echo "This file documents the project" > README.md
~/code/desk (main) $ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

nothing added to commit but untracked files present (use "git add" to track)
~/code/desk (main) $
```

```
01  mkdir desk && cd desk
02  git --version
03  git init
04  echo "…" > README.md
05  git status
06  git add README.md
07  git commit -m "…"
```

Ask Git what it sees: README.md is **untracked**.

A first example: adding your changes

desk

```
~/code/desk $ git --version
git version 2.39.5 (Apple Git-154)
~/code/desk $ git init
Initialized empty Git repository in /Users/timo/code/desk/.git/
~/code/desk (main) $ echo "This file documents the project" > README.md
~/code/desk (main) $ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

nothing added to commit but untracked files present (use "git add" to track)
~/code/desk (main) $ git add README.md
~/code/desk (main) $
```

```
01  mkdir desk && cd desk
02  git --version
03  git init
04  echo "…" > README.md
05  git status
06  git add README.md
07  git commit -m "…"
```

Stage the file: it moves into the **staging area**.

A first example: committing your changes

desk

```
~/code/desk (main) $ echo "This file documents the project" > README.md
~/code/desk (main) $ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

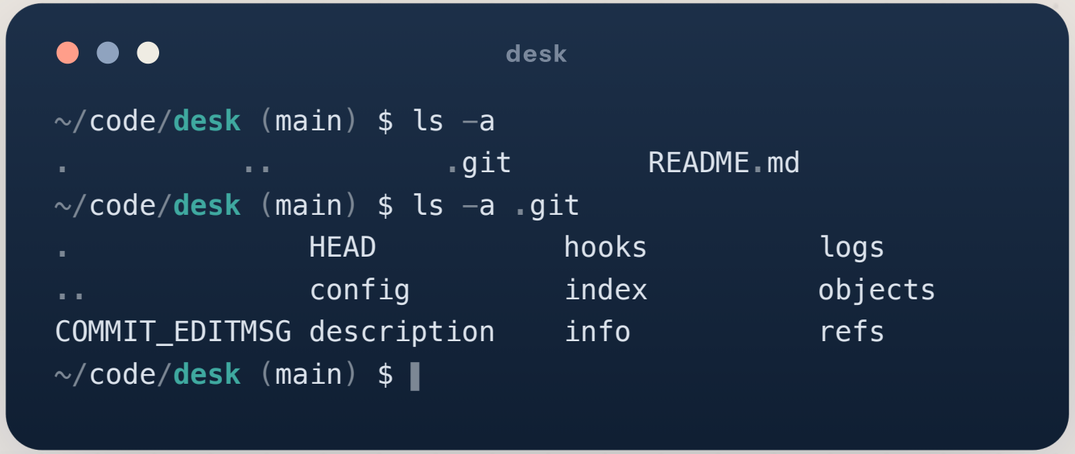
nothing added to commit but untracked files present (use "git add" to track)
~/code/desk (main) $ git add README.md
~/code/desk (main) $ git commit -m "Create README.md"
[main (root-commit) aaa829b] Create README.md
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
~/code/desk (main) $
```

```
01  mkdir desk && cd desk
02  git --version
03  git init
04  echo "…" > README.md
05  git status
06  git add README.md
07  git commit -m "…"
```

Commit: the staged snapshot is stored in the **repository**.

What is the .git folder?

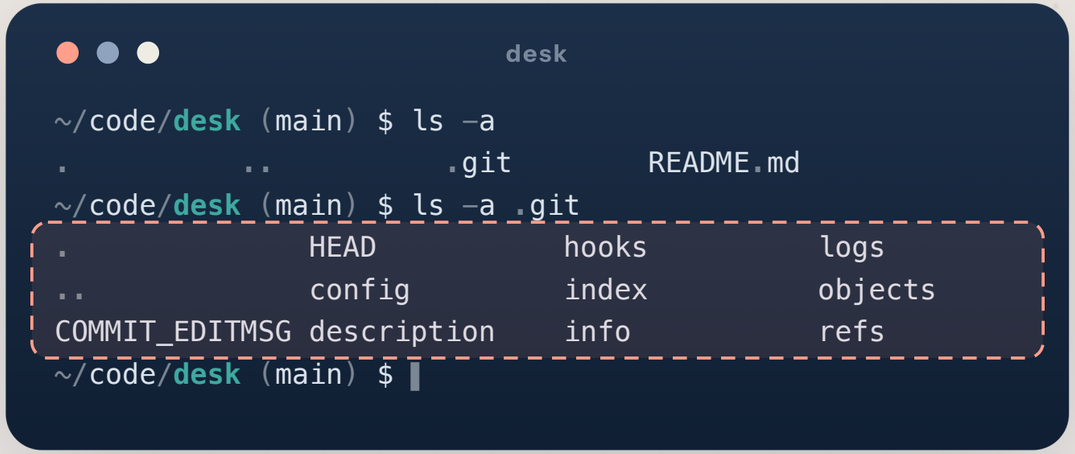
- A hidden directory that Git automatically creates when you initialize a repository using `git init`.
- It contains all the metadata and object data for the version control of your project.
- It tracks the entire version history of your project.
- Without it, Git would not function, as this is where it stores all commit data and change tracking.



! Never delete or modify the `.git` folder — unless you want to remove Git tracking from your project!

What is the .git folder?

Entry	Key contents of the .git folder
`HEAD`	Points to the current branch or commit you're working on.
`config`	Configuration settings (user info, remotes, etc.) for the repository.
`refs/`	Stores pointers to branches and tags.
`objects/`	Stores all content (commits, trees and blobs) as compressed data.
`index`	The staging area where files are prepared before being committed.
`logs/`	Records every action that updates your HEAD or branches.



What is the .git folder?

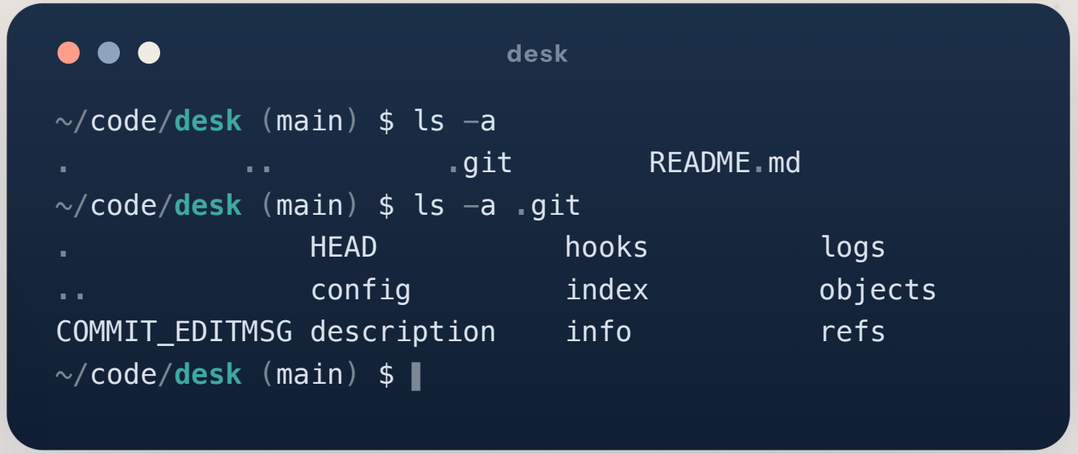
Want to learn more?

If you want to learn more about the .git folder, take a look at this resource:



git-school / git-under-the-hood

github.com/git-school/git-under-the-hood



Ignoring files with .gitignore

- A plain text file in the root of your repository: one pattern per line for files Git should **not** track.
- Ignore what can be regenerated or must stay private: environments, caches, secrets, large data.
- Commit the `.gitignore` itself, so the whole team ignores the same files.
- Templates for many languages: **github.com/github/gitignore**

.gitignore

```
# Environments and caches
.venv/
__pycache__/
.ipynb_checkpoints/

# Secrets: never commit these
.env

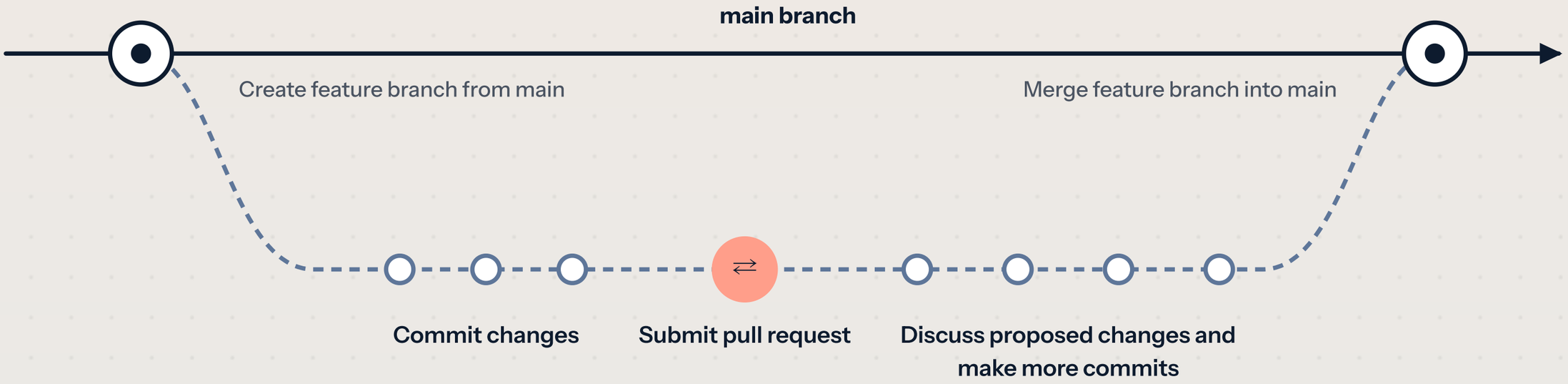
# Large or raw data
data/raw/

# Operating system clutter
.DS_Store
```

! Committed a secret by accident? Deleting it in the next commit is not enough — it stays in the history. Revoke the key and create a new one.

Common workflow

GITHUB FLOW



Source: training.github.com/downloads/github-git-cheat-sheet.pdf

Git's commit history as a DAG

DIRECTED ACYCLIC GRAPH

Graph

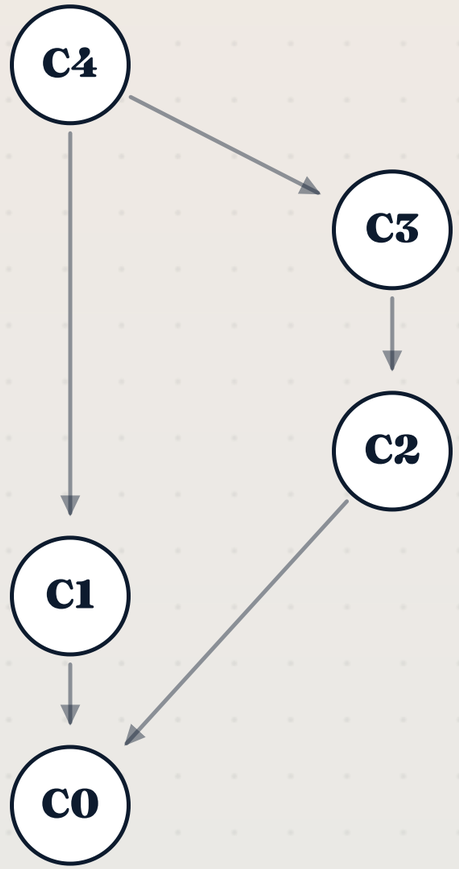
A set of **nodes** (vertices) connected by **edges** (links).

In Git

Commits are **nodes**, relationships between commits are **edges**.

Directed acyclic

Each commit points to its parent commit(s) — only the very first commit has none. No commit points back to its descendants: edges have a direction, and there are no cycles.



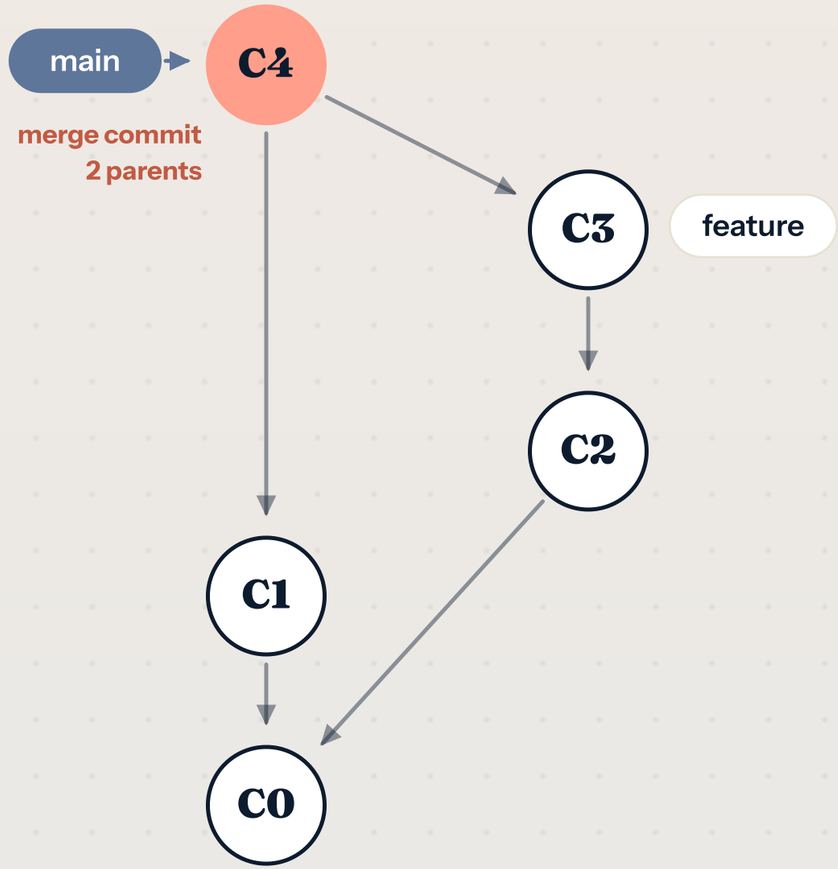
Git's commit history as a DAG

Branch: a pointer (label) to a commit in the graph.

- Points to the latest commit in a series.
- As you commit, the branch moves forward.
- Branches allow for multiple diverging paths in the commit graph.

Merging: joining two lines of development.

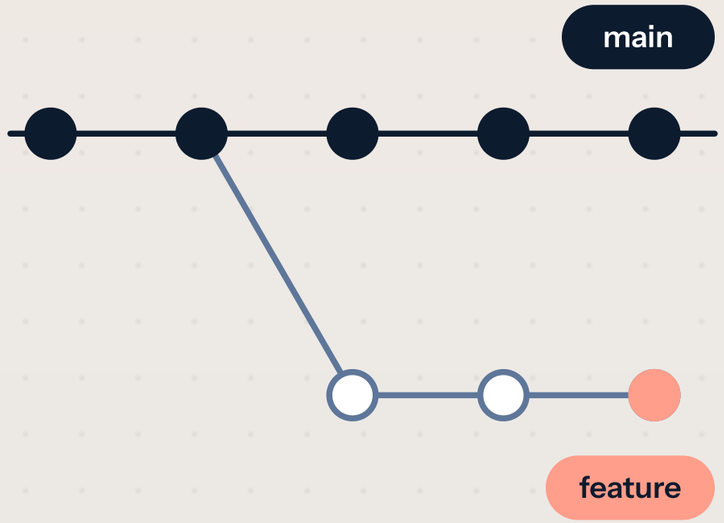
- **Fast-forward merge:** the current branch pointer is just moved forward to the target branch. No new node is added, the two branches simply converge.
- **Three-way merge:** combines two branches that have diverged. A new node is added with two parents (one from each branch).



What is a branch?

- A branch is an independent line of development. Keep code clean by isolating different pieces of work.
- Allows you to work on new features or bug fixes without affecting the main codebase.
- Visualize branches as parallel versions of the code. Easily integrate new features when ready.

Example: develop a new feature without affecting the main branch, i.e., keeping it clean!



Git branch commands overview

Command	What it does
`git branch <branch_name>`	Create a new branch
`git branch`	List all branches
`git branch -m <new_name>`	Rename a branch
`git branch -d <branch_name>`	Delete a branch
`git switch <branch_name>`	Switch to a branch
`git switch -c <branch_name>`	Create and switch
`git status`	Check the current branch
`git checkout -b <branch_name>`	Create and switch
`git checkout <branch_name>`	Switch to a branch

Visualizing branches

You can see the branching history using:

```
git log --oneline --graph --decorate --all
```

PythonDataScienceHandbook

```
~/code/PythonDataScienceHandbook (master) $ git log --oneline --graph --decorate --all
* c340151 (origin/dependabot/pip/pillow-10.0.1) Bump pillow from 3.4.2 to 10.0.1
| * eabac5f (origin/dependabot/pip/scipy-1.10.0) Bump scipy from 0.17.1 to 1.10.0
|/
* d662314 (HEAD -> master, origin/master, origin/HEAD) Add source material from second edition
| * cc569ee (origin/notebooks-v2) Add v2 notebooks
|/
| * ce26f23 (origin/v2) remove contents cells from chapters 00 and 01
| * 4d709a9 Update 02.05
| * e890daf Update 02.03, 02.04
| * 68ff5a4 Update jupytext version
| * 231dc69 Update 02.00, 02.02, 02.02
| * 7d64b94 Update requirements
| * 175a5dc update requirements.txt to most recent versions of packages
| * 4ee6907 Update 01.07 and 01.08
| * 153182a Update 01.06
| * 75160a7 update 01.05
```

What is merging in Git?

Combining two branches together — often used to bring the changes from a feature branch back into the main branch.

Fast-forward merge

When there is a direct path between the current branch and the branch being merged. Simply moves the pointer forward.

Three-way merge

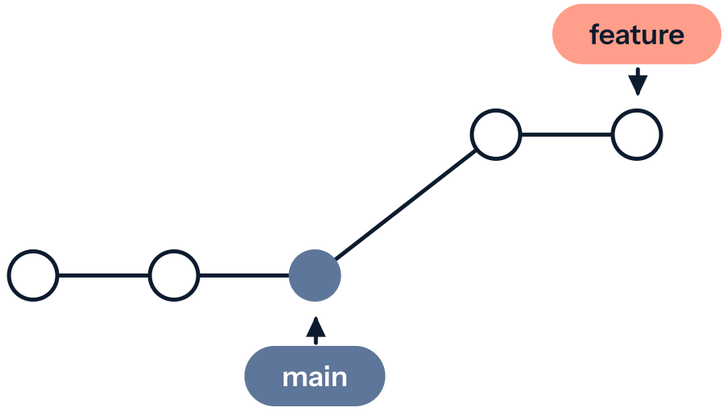
Occurs when the branches have diverged and have different commits. Git creates a new commit that reconciles the differences.

Command `git merge <branch_name>`

What is merging in Git?

Fast-forward merge

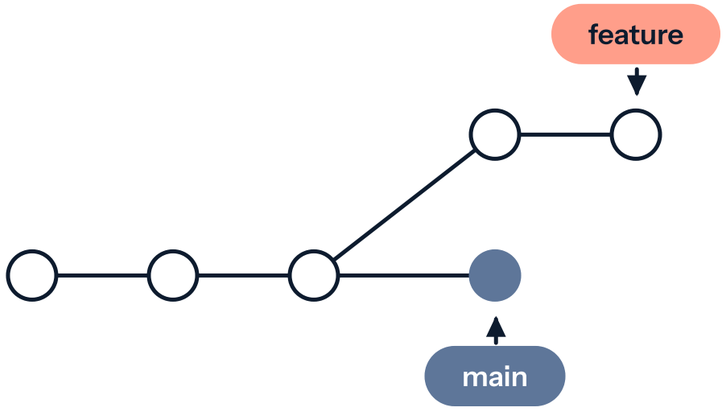
BEFORE MERGING



main has not moved since **feature** branched off.

Three-way merge

BEFORE MERGING



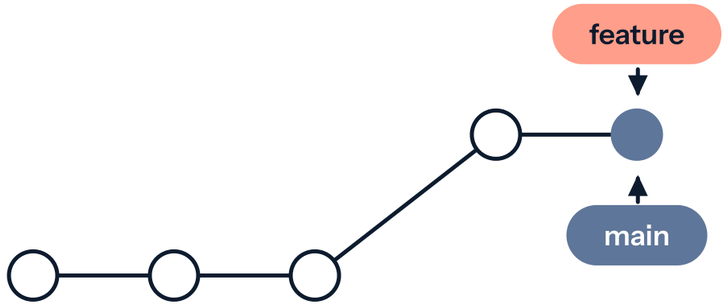
main and **feature** have both moved on: they diverged.

After: atlassian.com/git/tutorials/using-branches/git-merge

What is merging in Git?

Fast-forward merge

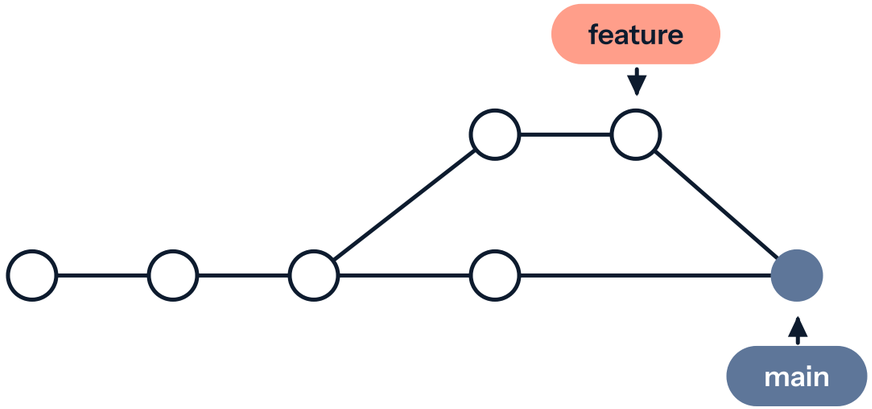
AFTER MERGING



The **main** pointer simply moves forward. No new commit.

Three-way merge

AFTER MERGING



A new **merge commit** with two parents joins both lines.

After: atlassian.com/git/tutorials/using-branches/git-merge

Handling merge conflicts

- Merge conflicts happen when Git cannot automatically resolve differences.
- Manual intervention is required to resolve conflicts.
- Conflict markers: `<<<<<<<`, `=======`, `>>>>>>>`

EXAMPLE

```
<<<<<<< HEAD
Current branch changes.
=======
Merging branch changes.
>>>>>>> feature/new-feature
```

Marker	Meaning
`<<<<<<< HEAD`	Start of the version on your current branch
`=======`	Separates the two versions
`>>>>>>> feature/new-feature`	End of the version from the branch being merged

How to resolve merge conflicts



Git marks the conflict. You decide what the result should be.

Best practices for branching and merging

01

Descriptive names

Use descriptive branch names, e.g. `feature/new-feature`, `bugfix/issue-123`

02

Short-lived branches

Keep branches short-lived to avoid large, complex merges.

03

Stay up to date

Regularly pull changes from the main branch into your working branch.

04

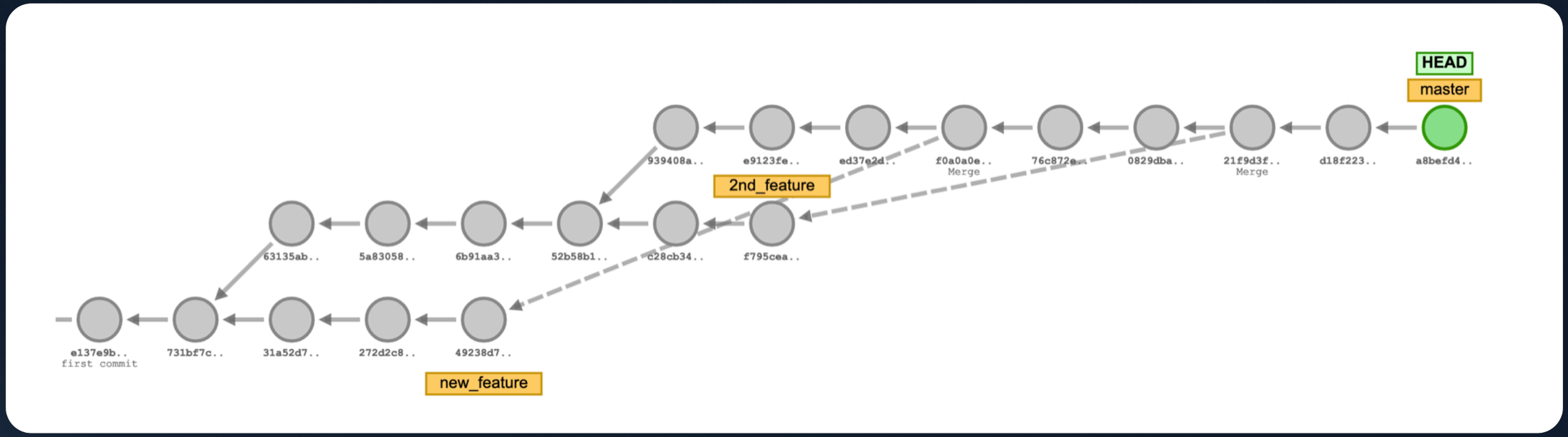
Review before merging

Always review code before merging, e.g. with pull requests.

Demo: branching & merging

Try it yourself in the browser:

git-school.github.io/visualizing-git



Viewing your history with git log

`git log` shows

Commit hash

A unique identifier for each commit.

Author

Who made the commit.

Date

When the commit was made.

Commit message

A description of the changes made.

PythonDataScienceHandbook

```
~/code/PythonDataScienceHandbook (master) $ git log -n 3
commit d66231454ef753818dc9213c9b5942e067266966 (HEAD -> master,
origin/master, origin/HEAD)
Author: Jake VanderPlas <jakevdp@google.com>
Date:   Fri May 5 16:20:45 2023 -0700

    Add source material from second edition

commit 8a34a4f653bdbdc01415a94dc20d4e9b97438965
Author: Jake VanderPlas <jakevdp@gmail.com>
Date:   Fri Nov 30 15:24:10 2018 -0800

    Change deprecated pd.datetools.timedelta to datetime.timedelta

commit be23269c7eb119e093a6d5ce91e464f5e686d9ab
Merge: 97c8c91 edf3947
Author: Jake Vanderplas <jakevdp@gmail.com>
Date:   Thu Aug 30 10:23:00 2018 -0700

    Merge pull request #145 from choldgraf/35binder
```

What is a commit hash?

40 character SHA-1 string, generated from the commit's contents and metadata

- A unique identifier assigned to each commit in Git.
- Allows you to refer back to specific changes in your project.
- Change a single byte and the hash changes — so history cannot be altered unnoticed.

Look at an old version:

```
git checkout <commit-hash>
```

Detached HEAD — look only. Back with `git switch main`.

PythonDataScienceHandbook

```
~/code/PythonDataScienceHandbook (master) $ git log -n 3
commit d66231454ef753818dc9213c9b5942e067266966 (HEAD -> master,
origin/master, origin/HEAD)
Author: Jake VanderPlas <jakevdp@google.com>
Date:   Fri May 5 16:20:45 2023 -0700

    Add source material from second edition

commit 8a34a4f653bdbdc01415a94dc20d4e9b97438965
Author: Jake VanderPlas <jakevdp@gmail.com>
Date:   Fri Nov 30 15:24:10 2018 -0800

    Change deprecated pd.datetools.timedelta to datetime.timedelta

commit be23269c7eb119e093a6d5ce91e464f5e686d9ab
Merge: 97c8c91 edf3947
Author: Jake Vanderplas <jakevdp@gmail.com>
Date:   Thu Aug 30 10:23:00 2018 -0700

    Merge pull request #145 from choldgraf/35binder
```

Undoing things

You want to ...	Command	Note
See what you changed	`git diff`, `git diff --staged`	Read-only, always safe
Unstage a file, keep the changes	`git restore --staged <file>`	Safe
Discard uncommitted changes in a file	`git restore <file>`	Changes are gone for good
Fix the last commit (message, forgotten file)	`git commit --amend`	Only if not pushed yet
Undo a commit that is already pushed	`git revert <commit-hash>`	Safe: adds a new commit
Move the branch back to an older commit	`git reset --hard <commit-hash>`	Dangerous: rewrites history

Already pushed? Use `git revert` — never rewrite history that others have.

Remote repositories

A remote repository is hosted on the internet or another network.

It enables multiple developers to collaborate and work on the same project by providing a centralized location for code storage.

COMMON PLATFORMS

GitHub
github.com

Bitbucket
bitbucket.org

GitLab
about.gitlab.com

Codeberg
codeberg.org

Cloning a repository

Cloning a repository means creating a copy of a remote repository on your local machine. This allows you to work on the project offline, with full access to the entire history and version control.

1

Get the repository URL

Navigate to the repository on platforms like GitHub or GitLab. Copy the repository's **HTTPS** or **SSH** URL.

2

Run the git clone command

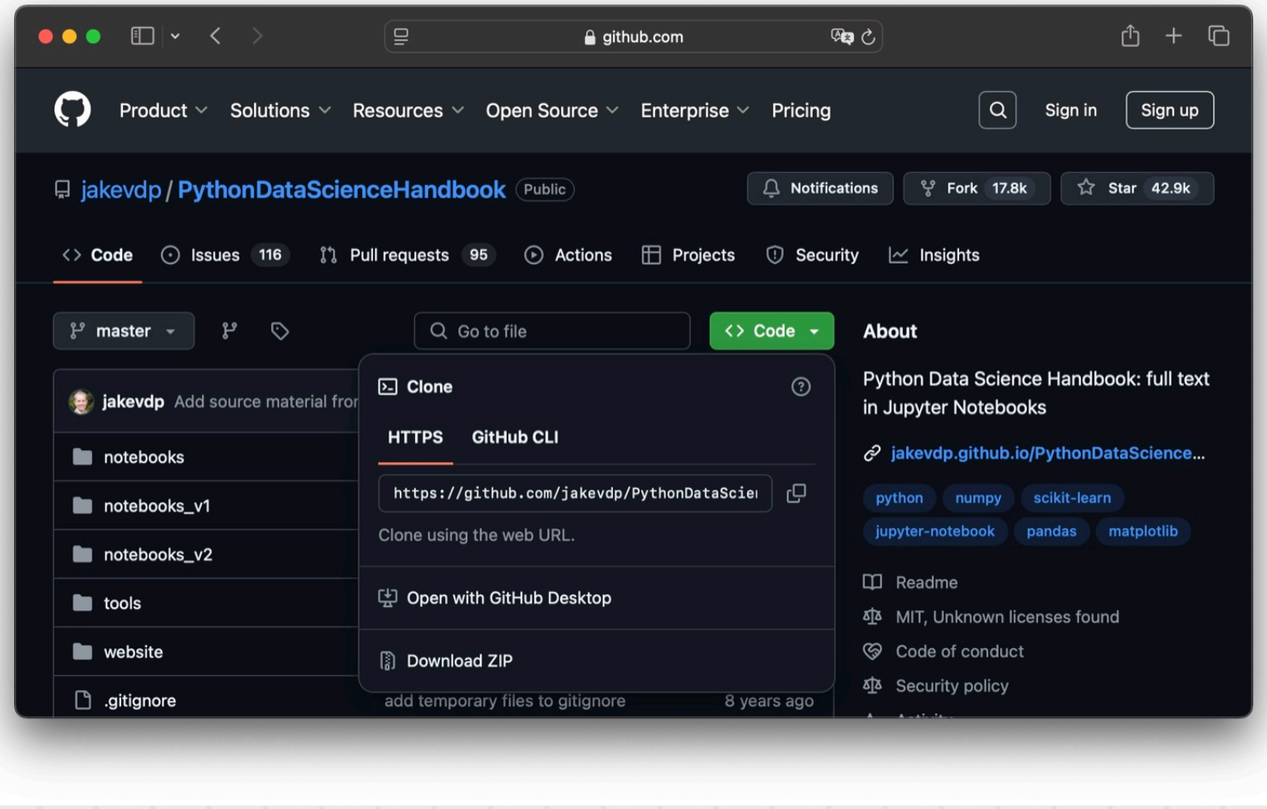
```
git clone <repository-url>
```

3

Access the cloned repository

After cloning, a new directory is created on your local machine containing the project files. You can now make changes, create branches, and push updates back to the remote repository.

How to find a repository's URL on GitHub?



1. Open the repository on GitHub.
2. Click the green **Code** button.
3. Choose **HTTPS** (or SSH) and copy the URL.

```
git clone https://github.com/…
```

Syncing with remote repositories

Syncing keeps your local repository up to date with changes made by other collaborators and lets you share your own updates. Syncing often keeps merge conflicts small.

Command	Action	What it does
`git fetch`	**Fetch changes from remote**	Downloads changes from the remote repository without merging them into your local repository.
`git merge`	**Merge changes locally**	After fetching, you can merge the changes into your current branch.
`git pull`	**Pull changes (fetch + merge)**	Combines fetching and merging in one step to directly update your local repository with remote changes.
`git push`	**Push your changes to remote**	Upload your local commits to the remote repository.

git pull = git fetch + git merge

Pushing your commits

Pushing is the process of uploading your local commits to a remote repository, making them available for others to see and collaborate on.

1 Ensure your changes are committed

Before pushing, all changes must be committed locally.

```
git add <file>
git commit -m "Your commit message"
```

2 Push to the remote repository

Use the git push command to upload your changes to the remote branch.

```
git push origin <branch-name>
git push origin main
git push --set-upstream origin <branch-name>
```

3 Ensure you’re up to date

If others have made changes, use git pull to sync before pushing, to avoid conflicts.

```
git pull
```

Authenticating with GitHub

Cloning a public repository needs no login. To **push**, GitHub must know who you are — and it does not accept your account password on the command line.

1

HTTPS + token

Create a personal access token on GitHub (Settings → Developer settings). Paste it when Git asks for your password.

```
https://github.com/…
```

2

SSH key

Create a key pair once and add the **public** key to your GitHub account. Then use the SSH URL of a repository.

```
ssh-keygen -t ed25519
```

3

GitHub CLI

The simplest route: `gh` logs you in through the browser and sets up Git for you.

```
gh auth login
```

Tokens and private keys are passwords: never commit them, never share them.

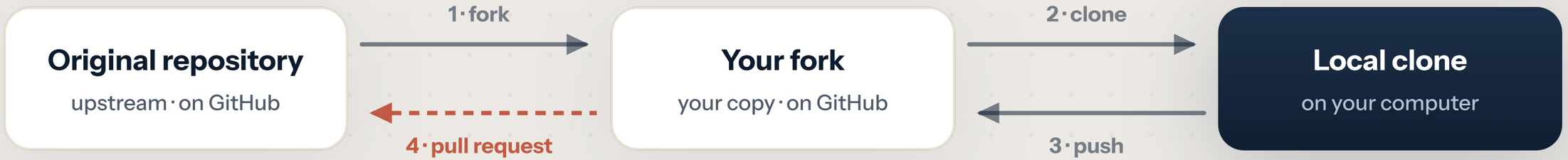
Pull requests and forks

Pull request

A request to merge one branch into another, made on the platform — not a Git command. Teammates review the diff, comment and approve before it is merged. GitLab calls it a *merge request*.

Fork

Your own copy of someone else's repository on the platform. You push to your fork, then open a pull request to the original repository (*upstream*).



> Working in one shared team repository? Then you need no fork — branch and open pull requests directly.

Git workflow example

1. **Clone** the repository.
2. **Create** a new branch.
3. **Make** changes.
4. **Commit** changes.
5. **Push** to remote.
6. **Open** a pull request.
7. **Merge** into the main branch.

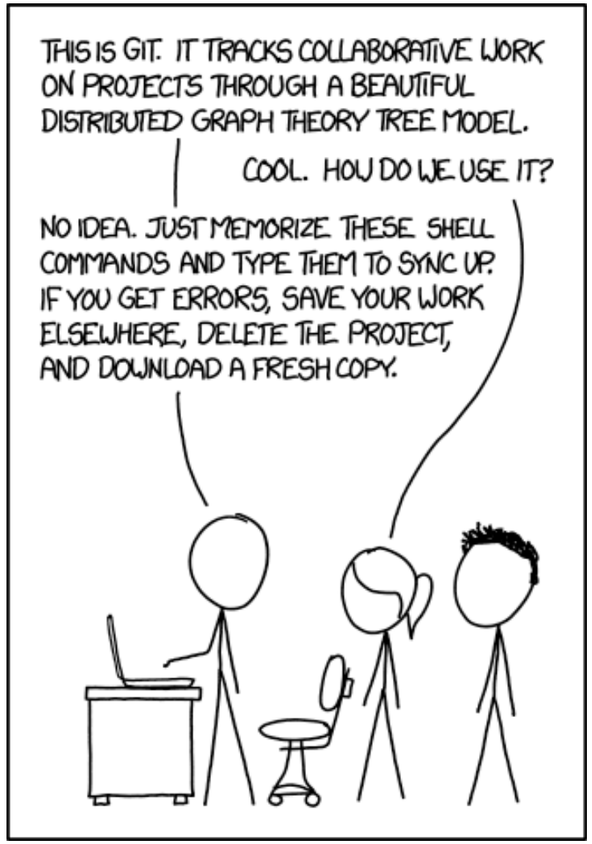


Image source: xkcd.com/1597

Best practices

- **Write clear and concise commit messages**
  Describe what and why.
- **Commit often**
  Small, frequent commits make it easier to track changes.
- **Use branches**
  Keep the main branch clean and stable.
- **Review code**
  Use pull requests for code review before merging.
- **Logical grouping**
  Do not mix typos with code features — use separate commits instead.



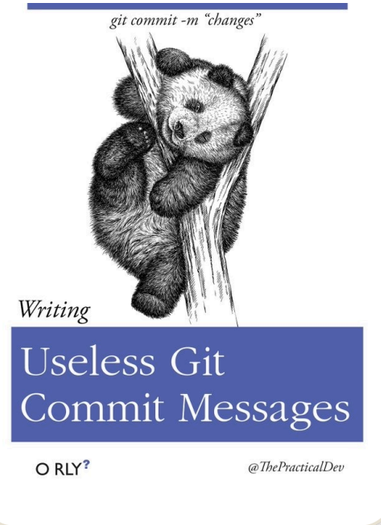


Image sources: xkcd.com/1296 · orlybooks.com/books/useless-git-commit-message

NOT LIKE THIS

```
git commit -m "fix typo, add supercool feature"
```

Version control for data projects

01

Code yes, raw data no

Git is built for text. Keep large raw data out of the repository and document in the README where it comes from.

02

Mind the size limit

GitHub blocks files larger than 100 MB. For large or binary data, use Git LFS or DVC (Data Version Control).

03

Tame your notebooks

Notebooks store their outputs inside the file, so diffs get noisy. Clear outputs before committing, or use `nbstripout`.

04

Make results reproducible

Commit your `requirements.txt` and tag the version behind your results: `git tag v1.0`

Git as a safety net for AI agents

1 **Start from a clean state**

Commit your own work first: `git status` should say *nothing to commit*.

2 **Give the agent its own branch**

`git switch -c agent/cleaning` keeps `main` untouched while the agent works.

3 **Read the diff, not the summary**

`git diff` shows what actually changed — not what the agent says it changed.

4 **Merge only what you understand**

Review it like a teammate's pull request. Not convinced? Delete the branch.



More on this next week: **Agentic Coding**, 13.11.26.

Common Git commands

Command	What it does
`git init`	Initializes a new Git repository in the current directory.
`git clone`	Creates a local copy of a remote repository.
`git add`	Stages changes to be committed.
`git commit`	Saves the staged changes to the repository with a message.
`git status`	Shows the status of changes (staged, unstaged, untracked) in the repository.
`git log`	Displays the commit history.
`git branch`	Lists branches or creates a new branch.

Command	What it does
`git checkout`	Switches to a different branch or a specific commit.
`git merge`	Combines changes from one branch into another.
`git pull`	Fetches changes from a remote repository and merges them into your current branch.
`git push`	Uploads local commits to a remote repository.
`git diff`	Shows the changes between commits or between the working directory and the index.
`git revert`	Creates a new commit that undoes the changes of a previous commit.
`git stash`	Temporarily saves changes that aren’t ready to be committed.

Check your understanding

Show answers →

01

You edited `clean.py`. Which command shows exactly what you changed?

02

What is the difference between `git fetch` and `git pull`?

03

When does `git merge` create a new merge commit?

04

You pushed a commit with a bug. How do you undo it without rewriting history?

Questions?

Next lecture: Agentic Coding on 13.11.26.
Milestone 1 is due: your roadmap.

RESOURCES FROM TODAY

- github.com/git-school/git-under-the-hood — inside the .git folder
- git-school.github.io/visualizing-git — branching & merging, visualized
- training.github.com — the GitHub Git cheat sheet
- atlassian.com/git/tutorials — merging explained
- git-scm.com/book — Pro Git, the free Git book
- github.com/github/gitignore — .gitignore templates