Lib4RI

Federico Cantini
federico.cantini@lib4ri.ch

Frank Hösli
frank.hoesli@lib4ri.ch

Git into Version Control: an Introduction to Git for Scientists

CC BY NC SA

Today

Part	Topic	Start from
0	Why version control	
1	The command line	
2	Configuring Git	
3	Your first repository	`starter/`
4	Exploring history	`cp1`
	Break	
5	Undoing things	`cp2`
6	Branching and merging	`cp3`, `cp3b`
7	Advanced branching	`cp4`
8	Where to go next	

How today works

- You type along: every command shown here is also in the handout
- The course folder `git-course/` you unzipped before the course contains:
  - `starter/`: plain files you copy into your project
  - `cp1` … `cp4`: ready-made repositories (checkpoints), right next to your project
- Switching to a checkpoint is always `cd ../cp1`, `cd ../cp2`, … Lost? Jump to the next one and rejoin
- Running example: `temperature-study`, a script, a data file and a paper draft
- You never run the script: it is just text that changes over time
- Questions are welcome at any time

CC BY NC SA

Why version control

CC BY NC SA

Motivation

Have you ever:

- Made changes you didn’t mean to?
- Made some changes, but can’t remember what they were?
- Wanted to try something but was worried about breaking something else?
- Went crazy trying to figure out which version of your code produced certain output?
- Ended up with thesis_final_v3_REALLY_final.docx?

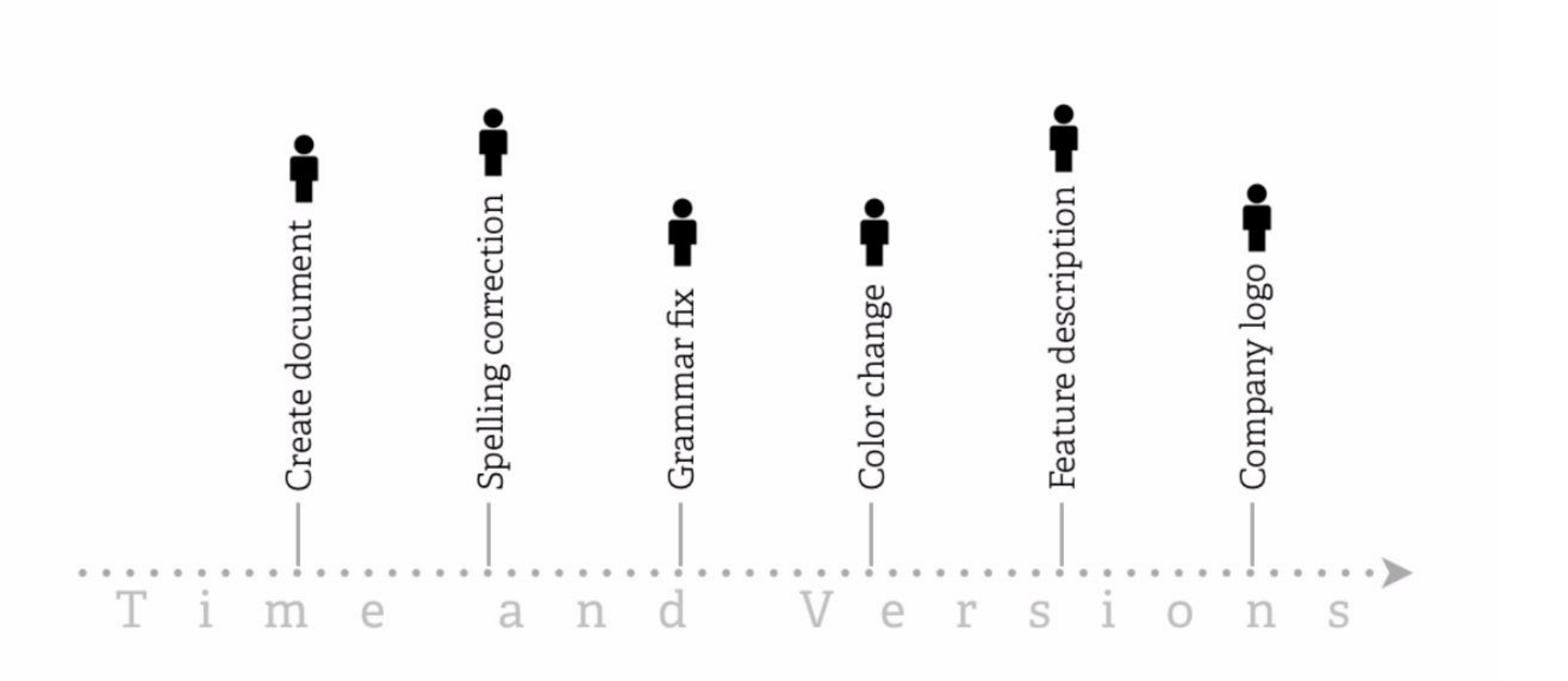
Version control

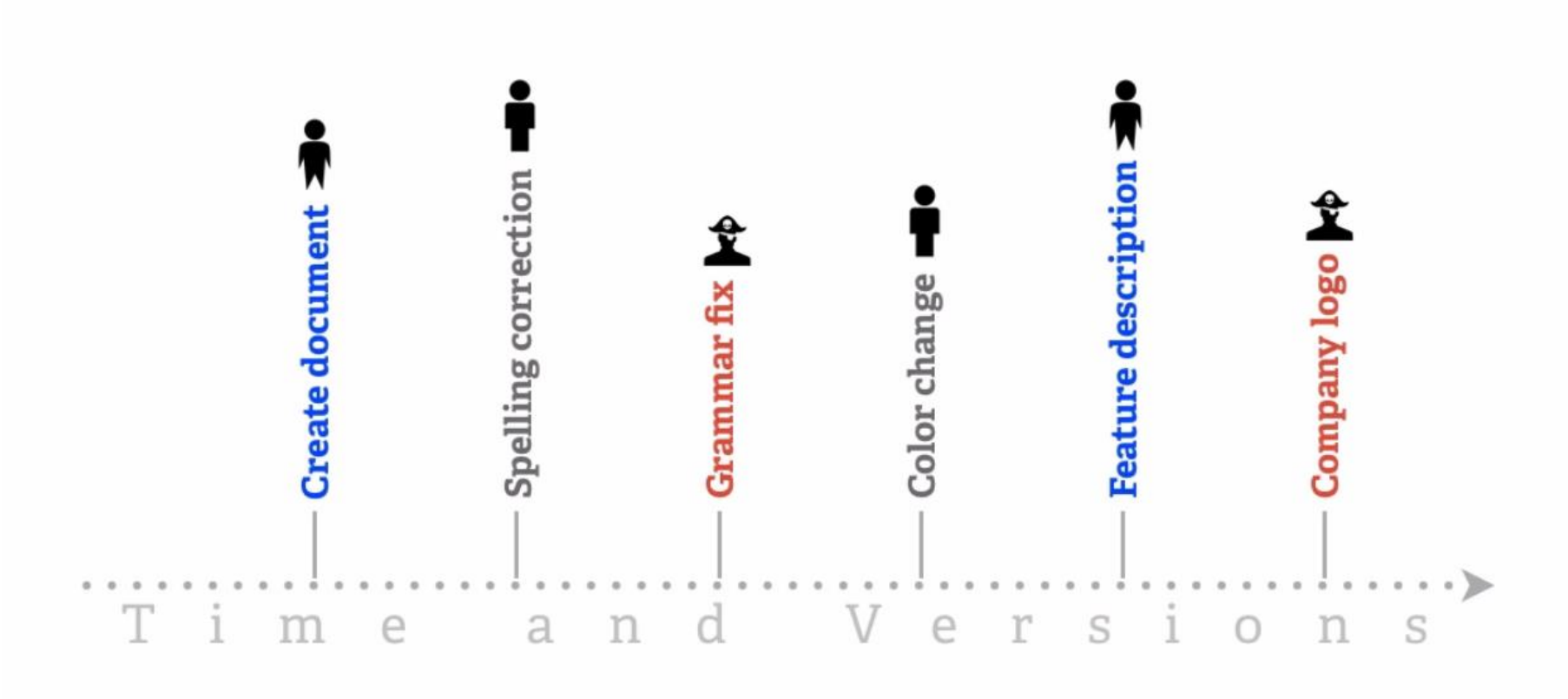
- Version control is a system that records changes to a file or set of files over time
- It allows to:
  - revert selected files back to a previous state
  - revert the entire project back to a previous state
  - compare changes over time
  - work concurrently with others
  - see who last modified something and when

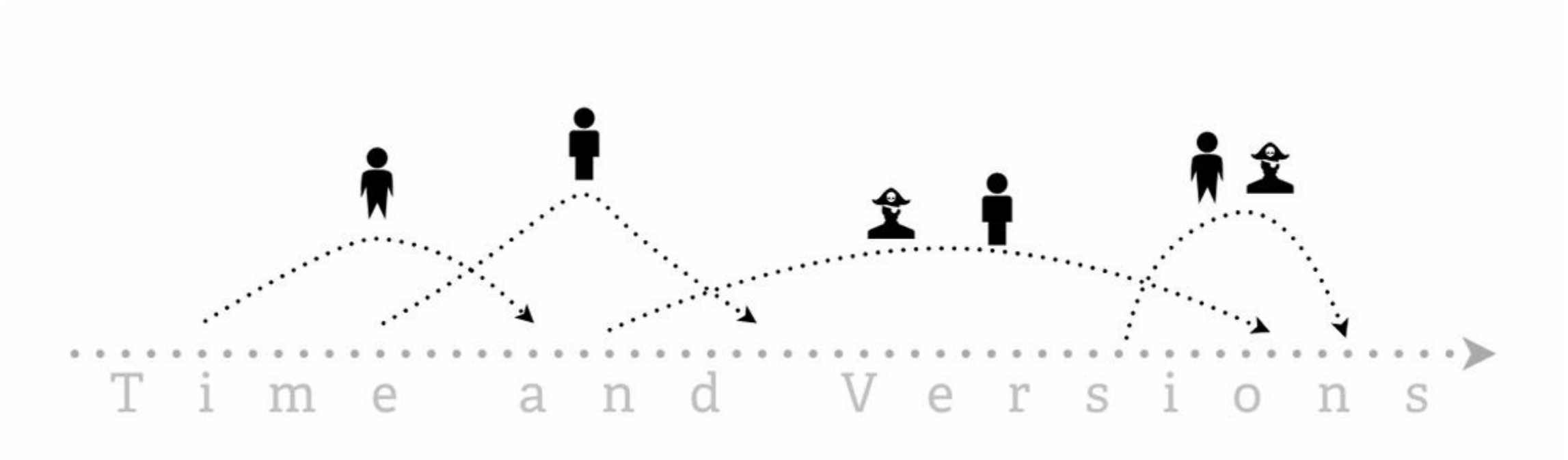
- If you break things you can (usually) easily recover

Pro Git 1.1 About Version Control

CC BY NC SA


Create document
Spelling correction
Grammar fix
Color change
Feature description
Company logo
Time and Versions


Create document
Spelling correction
Grammar fix
Color change
Feature description
Company logo
Time and Versions


Time and Versions



https://git-scm.com

- First release April 2005
- Started by the team developing the Linux kernel
- De facto standard Version Control System
- Free/open source software (GPL v.2 license) https://git-scm.com/about/free-and-open-source
- Available for GNU/Linux, macOS and Windows https://git-scm.com/downloads
- Works with any programming language (any text file indeed and possibly other file types)
- Command line interface
- GUI clients available https://git-scm.com/downloads/guis

- **git ≠ GitHub ≈ GitLab ≈ Codeberg ≈ …**

"Git logo" by Jason Long is licensed under CC BY 3.0

Pro Git 1.3 What is Git?

Our reference: Pro Git

- **Pro Git** by Scott Chacon and Ben Straub, free online: https://git-scm.com/book
- Readable on the web, or as PDF and e-book, in many languages
- Most images in this presentation come from it
- Each part of today points to the matching chapter: read it afterwards

Getting started

CC BY NC SA

Command-line survival kit

Command	What it does
`pwd`	Print the folder you are in
`ls` / `ls -a`	List the files / including hidden ones
`ls -l` (or `ll`)	List files with details: size, date, permissions
`cd <folder>`	Change into a folder
`cd ..` / `cd ~`	Go up one folder / go to your home folder
`mkdir <name>`	Create a new folder
`cp <from> <to>`	Copy a file, e.g. `cp ../starter/paper.md .`

- Linux and macOS: **Terminal** · Windows: **Git Bash**
- Tab completes file and folder names · Up arrow recalls previous commands
- `nano <file>` edits a file: Ctrl+O to save, Ctrl+X to exit

▶ Live demo

Pro Git 1.4 The Command Line

CC BY NC SA

Configuring Git

```
$ git config --global user.name "Your Name"
$ git config --global user.email your.name@institute.ch
$ git config --global core.editor nano
$ git config --global init.defaultBranch main
$ git config --global core.autocrlf input
$ git config --list
```

- `--global`: the setting applies to all your repositories
- Name and email are stamped into every commit you make
- The editor opens whenever Git needs a message from you
- `core.autocrlf input`: no line-ending conversions, no confusing warnings on Windows
- Help: `git help <command>` or `git <command> -h`

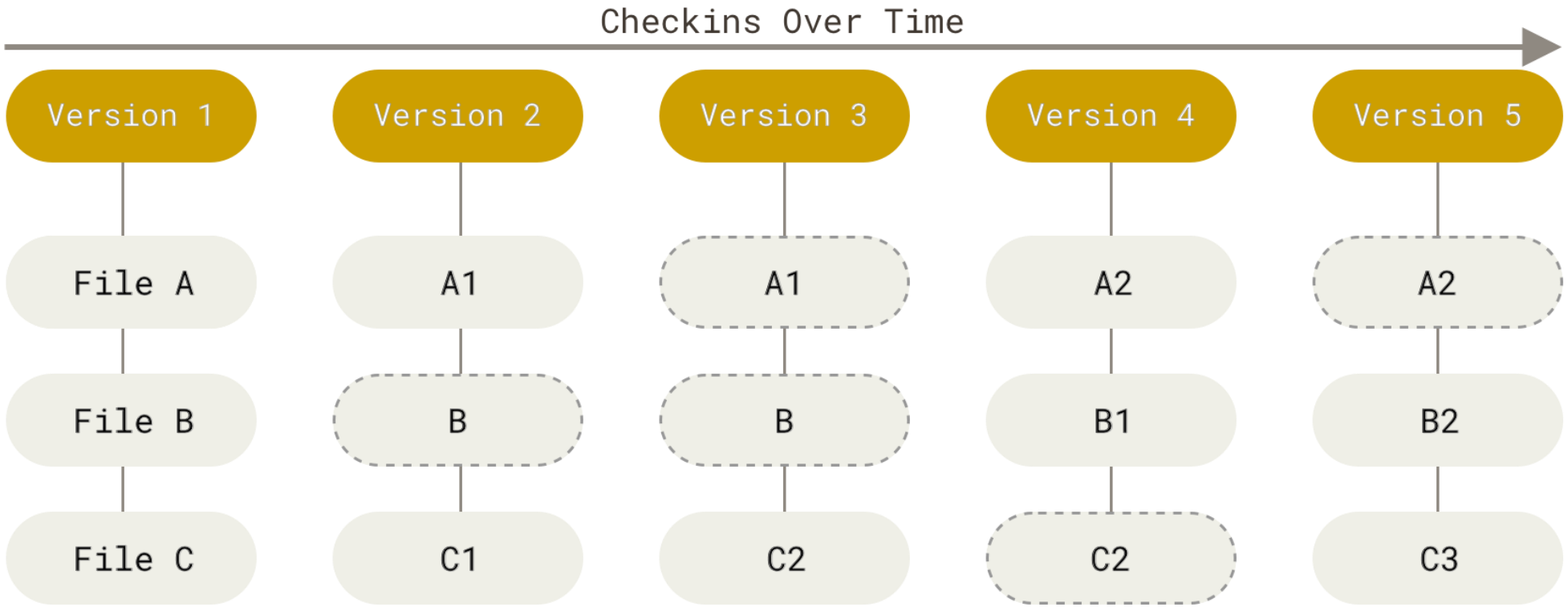
► **Live demo**

Pro Git 1.6 First-Time Git Setup

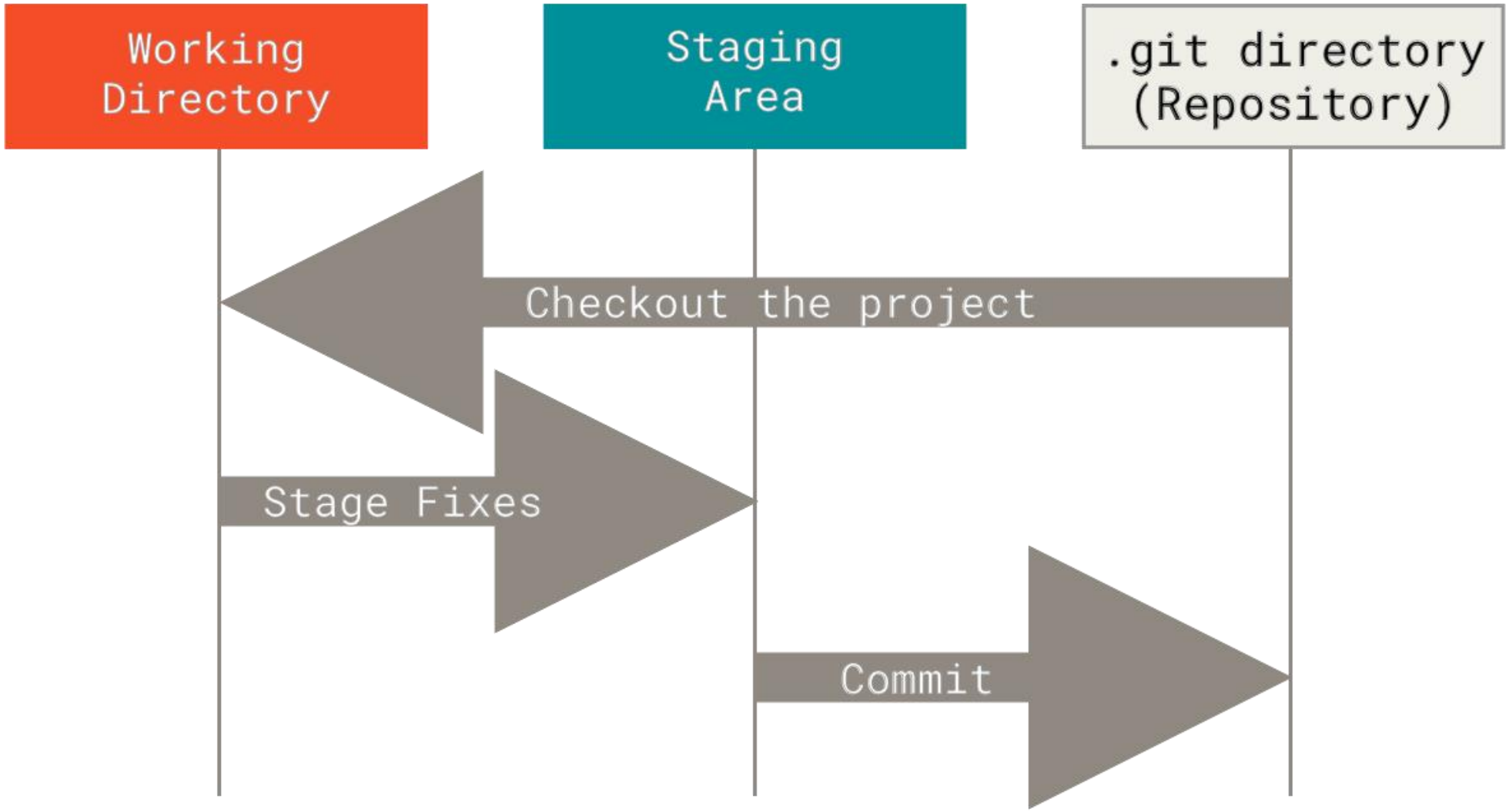
Pro Git 1.7 Getting Help

Using Git locally

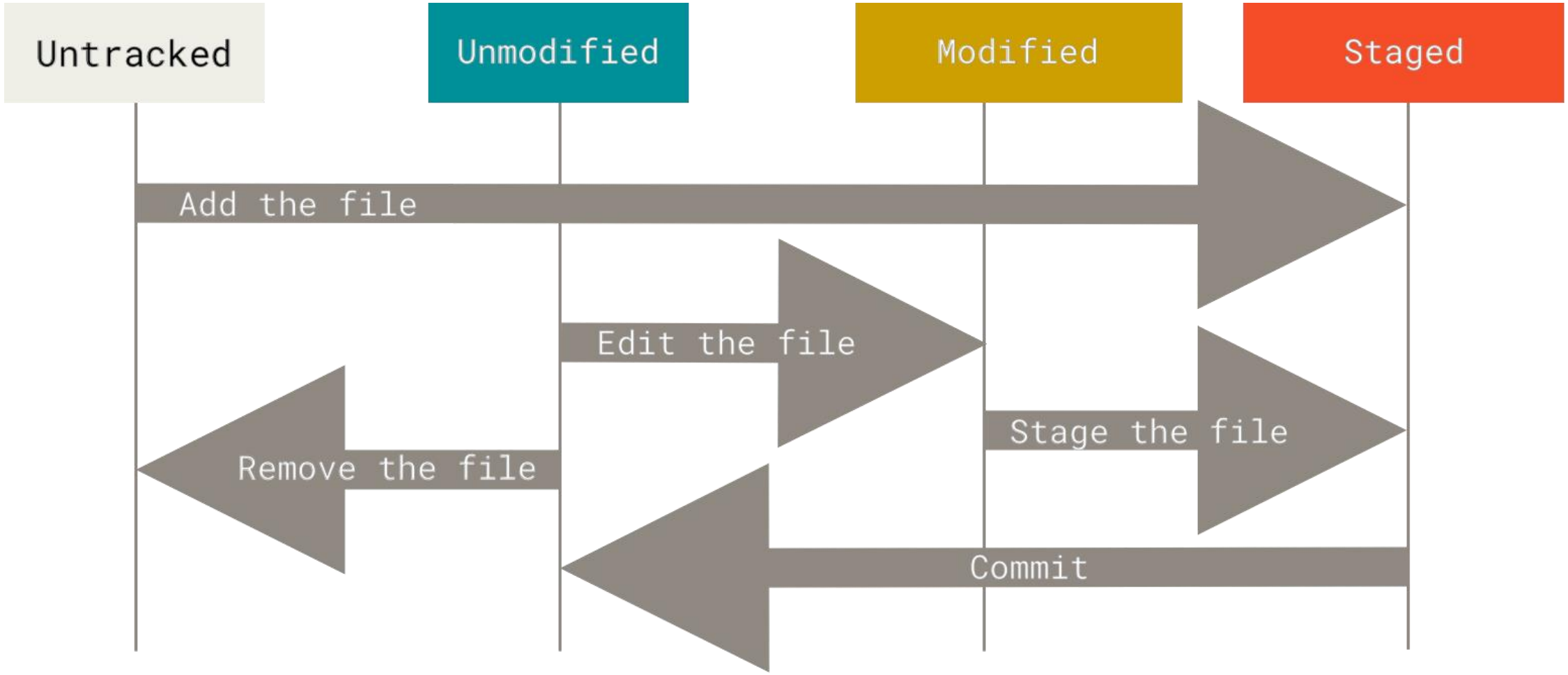
Locally track changes



Pro Git 1.3 What is Git?



Pro Git 1.3 What is Git?



Pro Git 2.2 Recording Changes to the Repository

Your first repository

```
$ mkdir temperature-study
$ cd temperature-study
$ git init
$ git status
$ git add README.md
$ git commit -m "Add README"
$ git log
$ git diff            # after editing again: what changed?
```

- Run `git status` after every step: watch the file move through the three areas
- One logical change, one commit: the staging area is how you compose it
- Commit messages: a short summary in the imperative ("Add", "Fix", "Clarify")

Pro Git 2.1 Getting a Git Repository Pro Git 2.2 Recording Changes to the Repository

What belongs in Git

- Git is made for text: code, scripts, LaTeX, Markdown, small data files
- Keep out: outputs you can regenerate, temporary files, large raw data
- List what to ignore in a file called `.gitignore`, one pattern per line:

```
results/
*.tmp
```

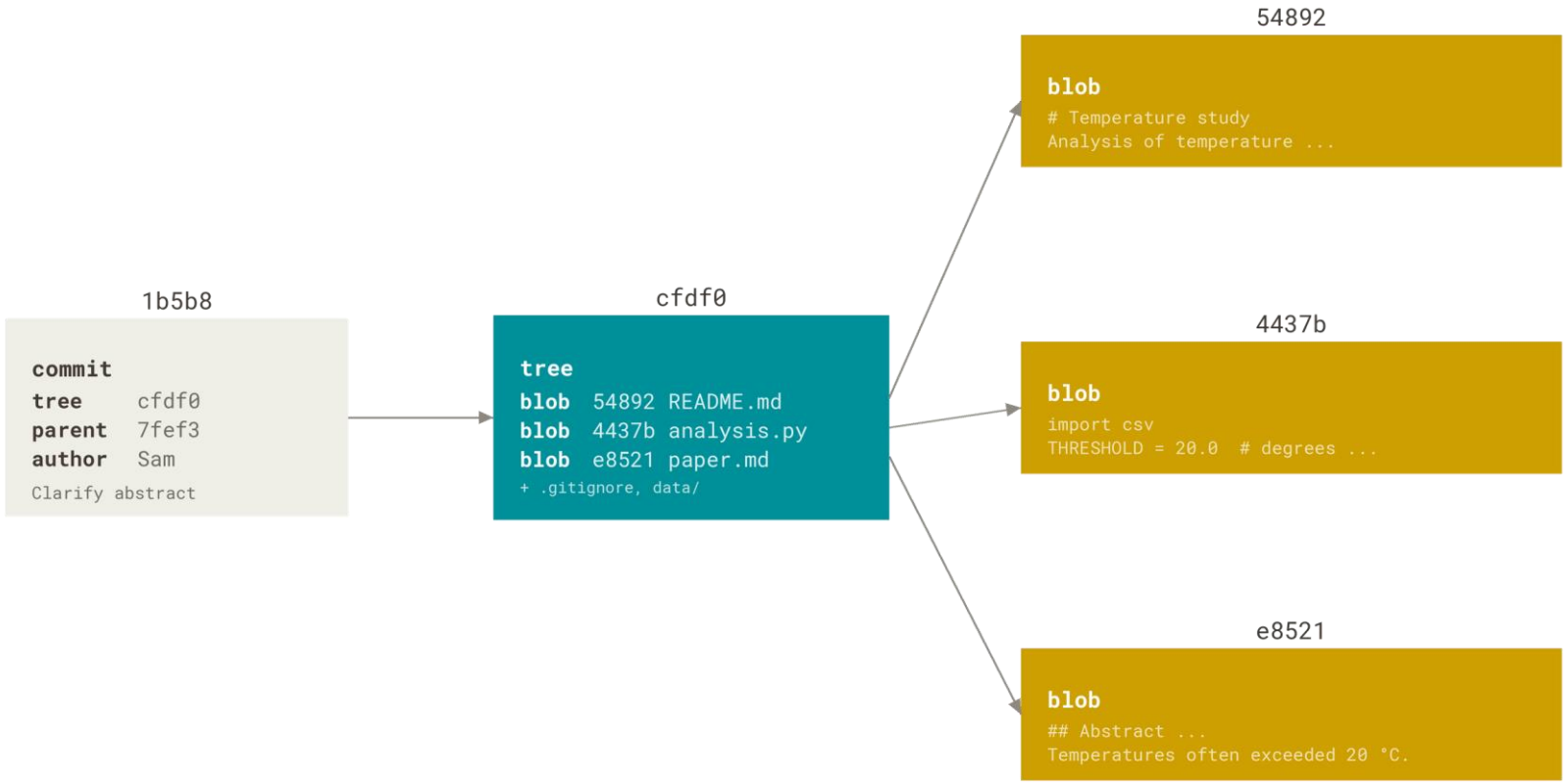
- Git tracks files, not folders: an empty folder never shows up
- Rename and delete tracked files with `git mv` and `git rm`

► **Live demo**

Pro Git 2.2 Recording Changes to the Repository

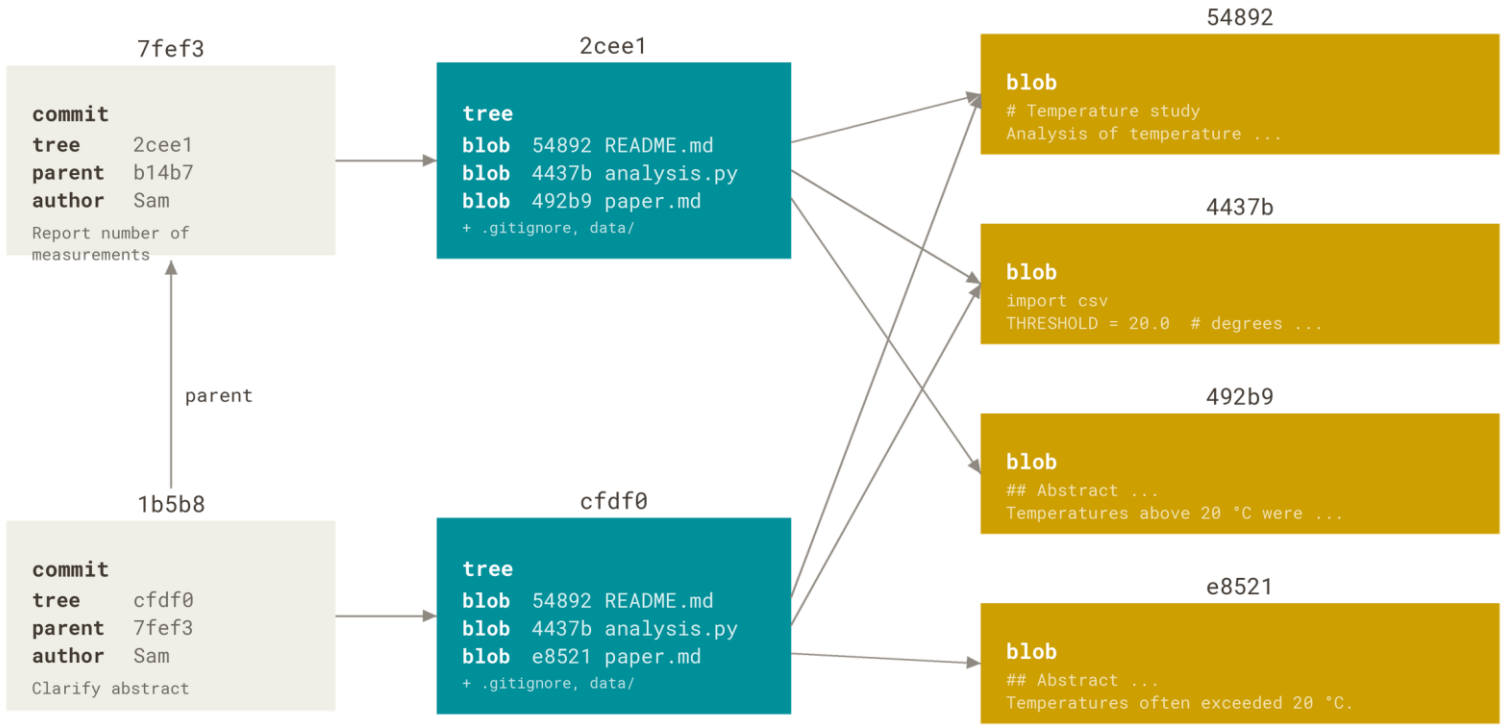
CC BY NC SA

A peek inside .git



Pro Git 10.2 Git Objects

What a new commit stores



Only paper.md changed: one new blob, one new tree, one new commit. README.md and analysis.py are reused, not copied.

Pro Git 10.2 Git Objects

CC BY NC SA

The objects in your repository

```
$ git cat-file -p HEAD            # the commit: tree, parent, author
$ git cat-file -p HEAD^{tree}     # its tree: the snapshot
$ git cat-file -p HEAD~1^{tree}   # the snapshot one commit earlier
$ cat .git/HEAD
$ cat .git/refs/heads/main
```

Object	What it stores
Blob	The content of one file
Tree	A folder: names pointing to blobs and other trees
Commit	One tree (the snapshot), the parent commit, author, date and message

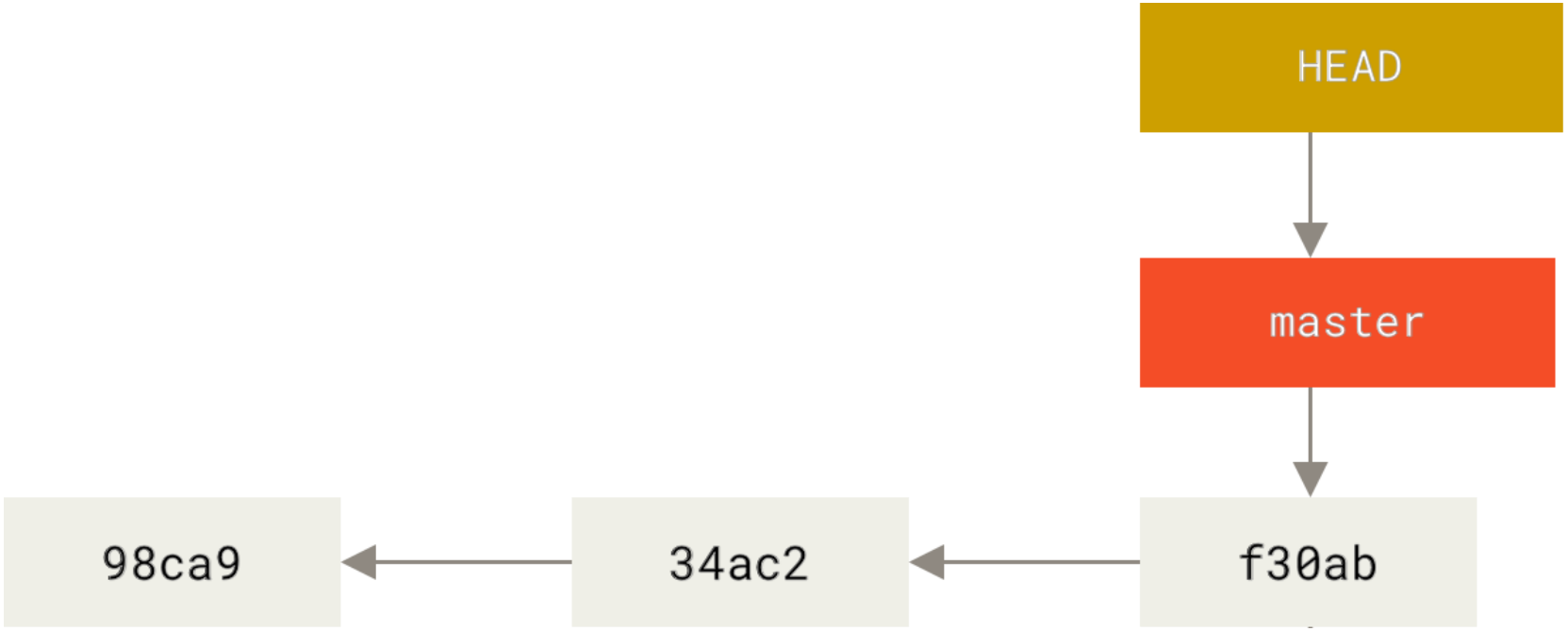
Every object is named by the hash of its content

Unchanged files are not copied: both snapshots point to the same blob

A branch is a 41-byte file containing a commit hash

Pro Git 10.2 Git Objects Pro Git 10.3 Git References

CC BY NC SA



Pro Git 3.1 Branches in a Nutshell

Exploring history

Reading the history

```
$ cd ../cp1
```

Command	What it shows
`git log --oneline --graph --all`	Compact history of all branches
`git log -p <file>`	Commits that changed a file, with the changes
`git show <commit>`	One commit: message and changes
`git show <commit>:<file>`	A file as it was in that commit
`git diff` / `git diff --staged`	Unstaged / staged changes
`git diff <a> <b>`	Differences between two commits

- Naming a commit: the first 7 characters of its hash, or `HEAD`, `HEAD~1`, `HEAD~2`, …

Pro Git 2.3 Viewing the Commit History Pro Git 7.1 Revision Selection

Detective questions

1. Why is the threshold 25 °C and not 20 °C?
2. The paper once reported −105.55 °C. Where did it come from, and what fixed it?
3. What did the abstract look like in the very first draft?
4. Compare `git diff` and `git diff --staged` before and after `git add`

Finished early?

5. What changed in the paper between the first draft and now?
6. Which commits changed a line containing THRESHOLD?
7. The abstract and the script disagree. Since when?

► **Live demo**

Pro Git 2.3 Viewing the Commit History

CC BY NC SA

Break · 15 minutes

CC BY NC SA

Undoing things

Fixing mistakes

```
$ cd ../cp2
```

Command	What it does
`git restore <file>`	Discard changes in the working directory (permanent!)
`git restore --staged <file>`	Unstage a file, keeping its changes
`git commit --amend`	Replace the last commit
`git revert <commit>`	Add a new commit that undoes an older one

- Commits never change: `--amend` creates a new commit and moves the branch
- `git log --format=fuller` shows author **and** committer
- `revert` adds history instead of erasing it: the safe choice for older commits

Pro Git 2.4 Undoing Things Pro Git 7.8 Advanced Merging

Reset moves the branch pointer

Mode	Branch pointer	Staging area	Working directory
`--soft`	moved	kept	kept
`--mixed` (default)	moved	reset	kept
`--hard`	moved	reset	reset

```
$ git reset --hard HEAD~3      # the accident
$ git reflog                   # everywhere HEAD has been
$ git reset --hard HEAD@{1}    # the rescue
```

Almost nothing in Git is ever really lost

► **Live demo**

Pro Git 7.7 Reset Demystified

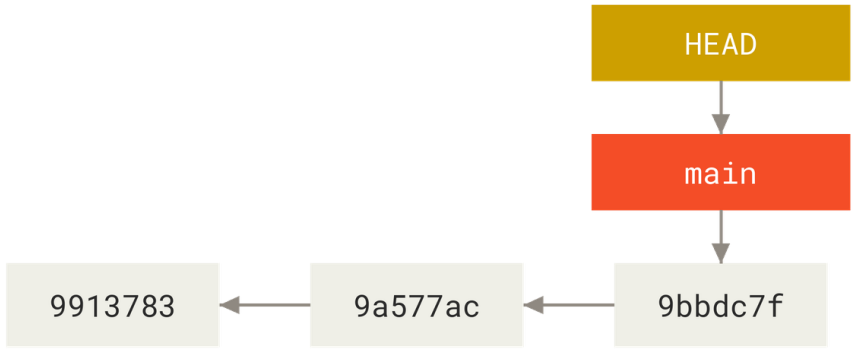
Pro Git 7.1 Revision Selection

CC BY NC SA

Branching and merging

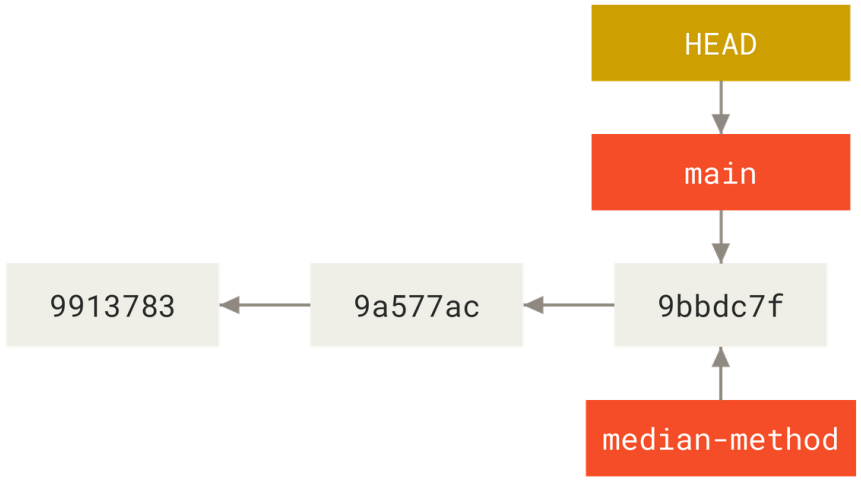
CC BY NC SA

A branch is a pointer to a commit; HEAD says which branch you are on



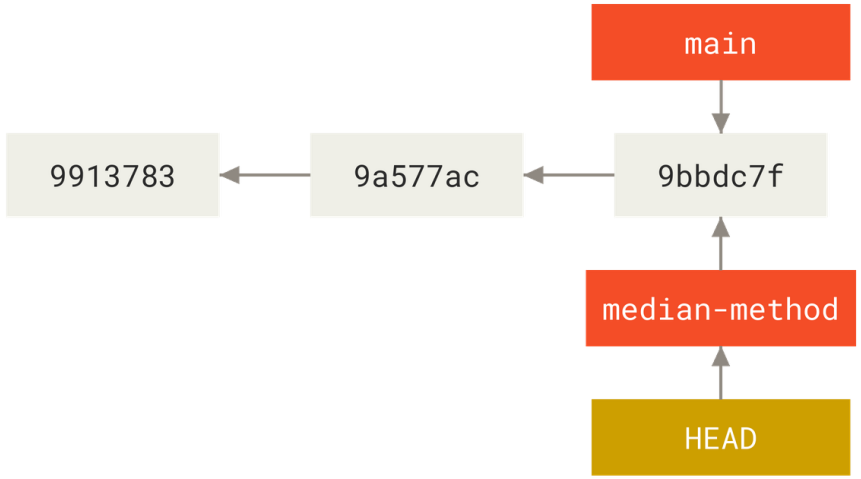
Pro Git 3.1 Branches in a Nutshell

```
$ git branch median-method
```



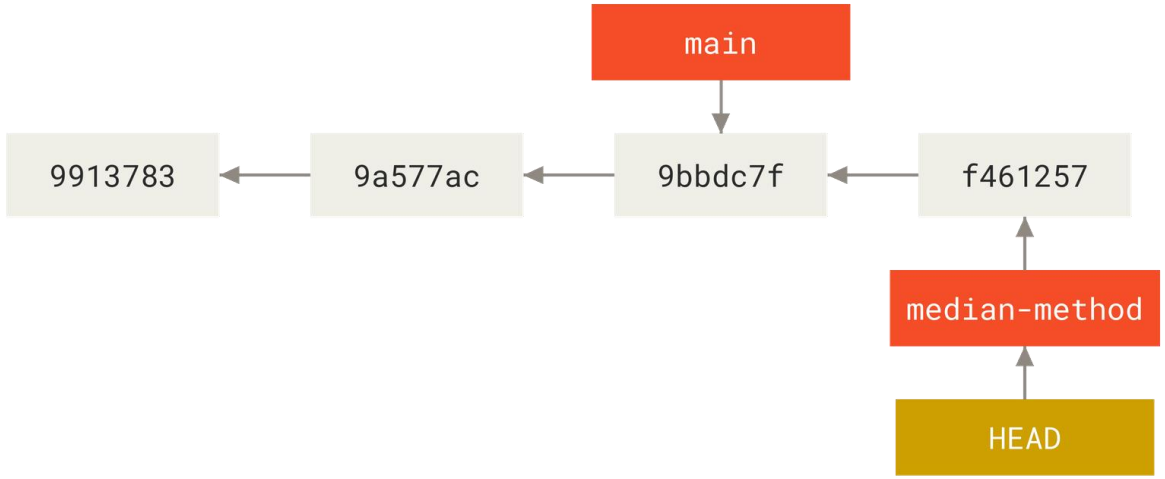
Pro Git 3.1 Branches in a Nutshell

```
$ git switch median-method
```



Pro Git 3.1 Branches in a Nutshell

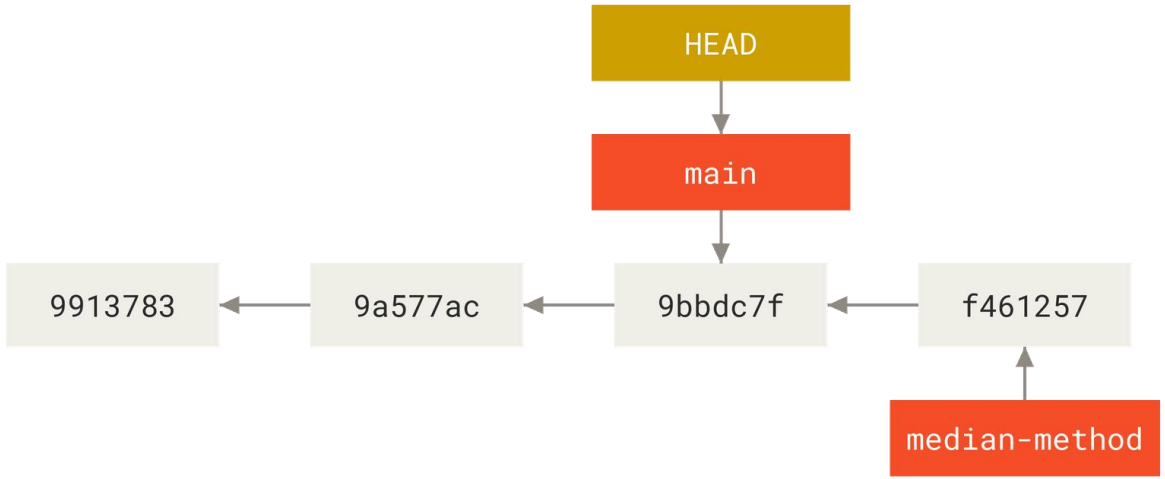
```
$ git commit -am "Add median as robust estimate"
```



Pro Git 3.1 Branches in a Nutshell

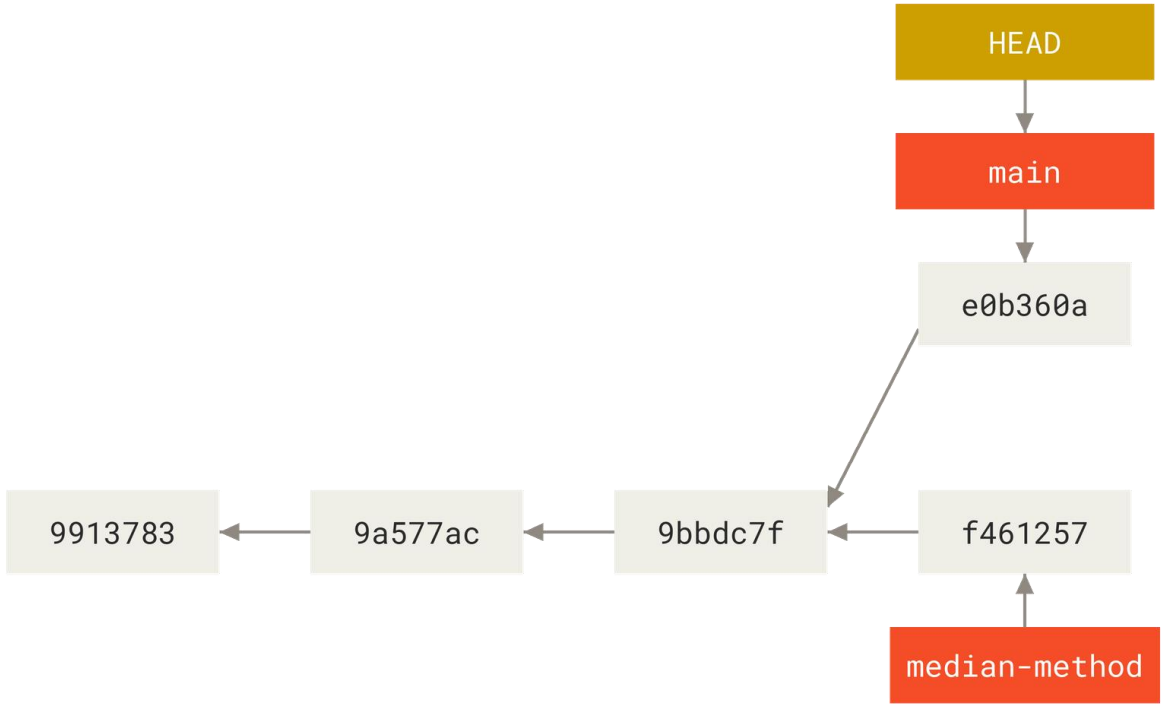
CC BY NC SA

```
$ git switch main
```



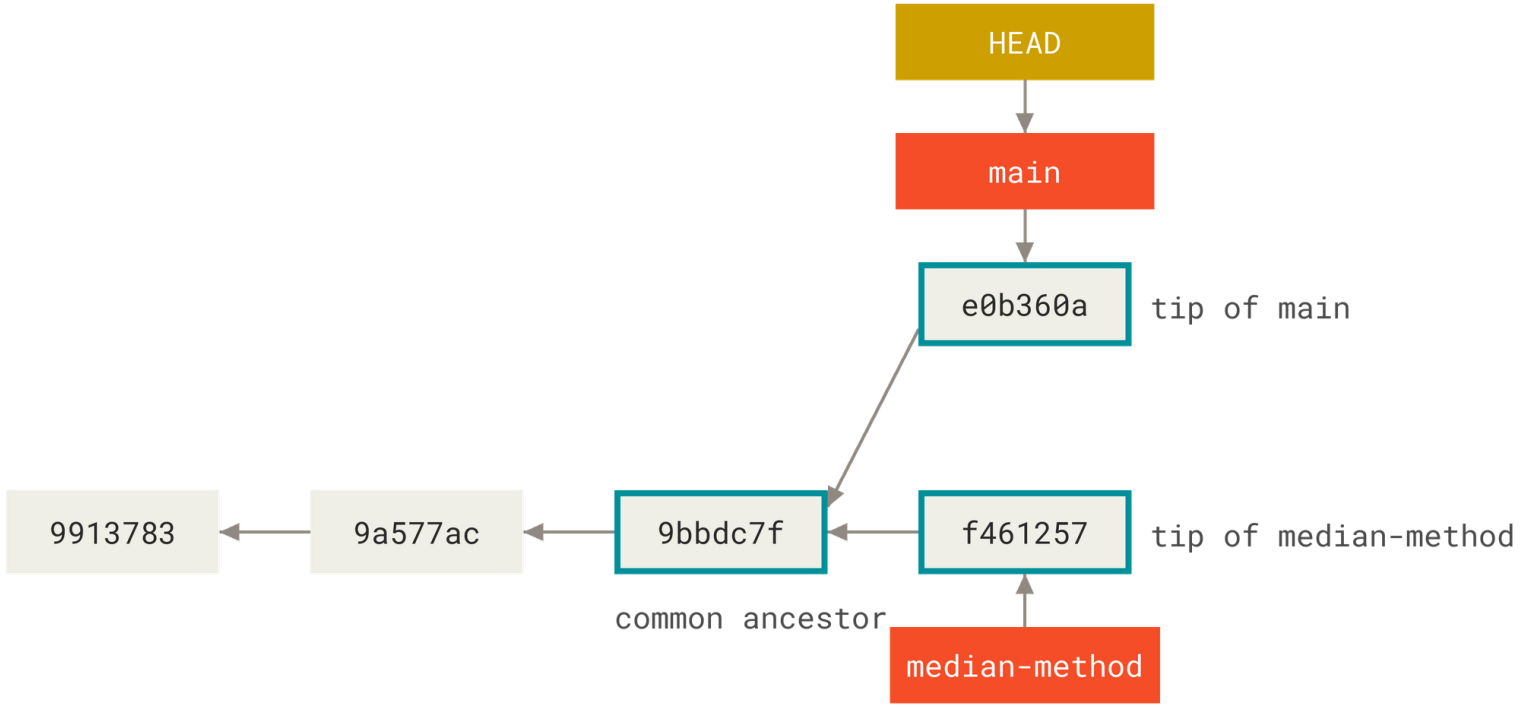
Pro Git 3.1 Branches in a Nutshell

```
$ git commit -am "Start discussion section"
```



Pro Git 3.1 Branches in a Nutshell

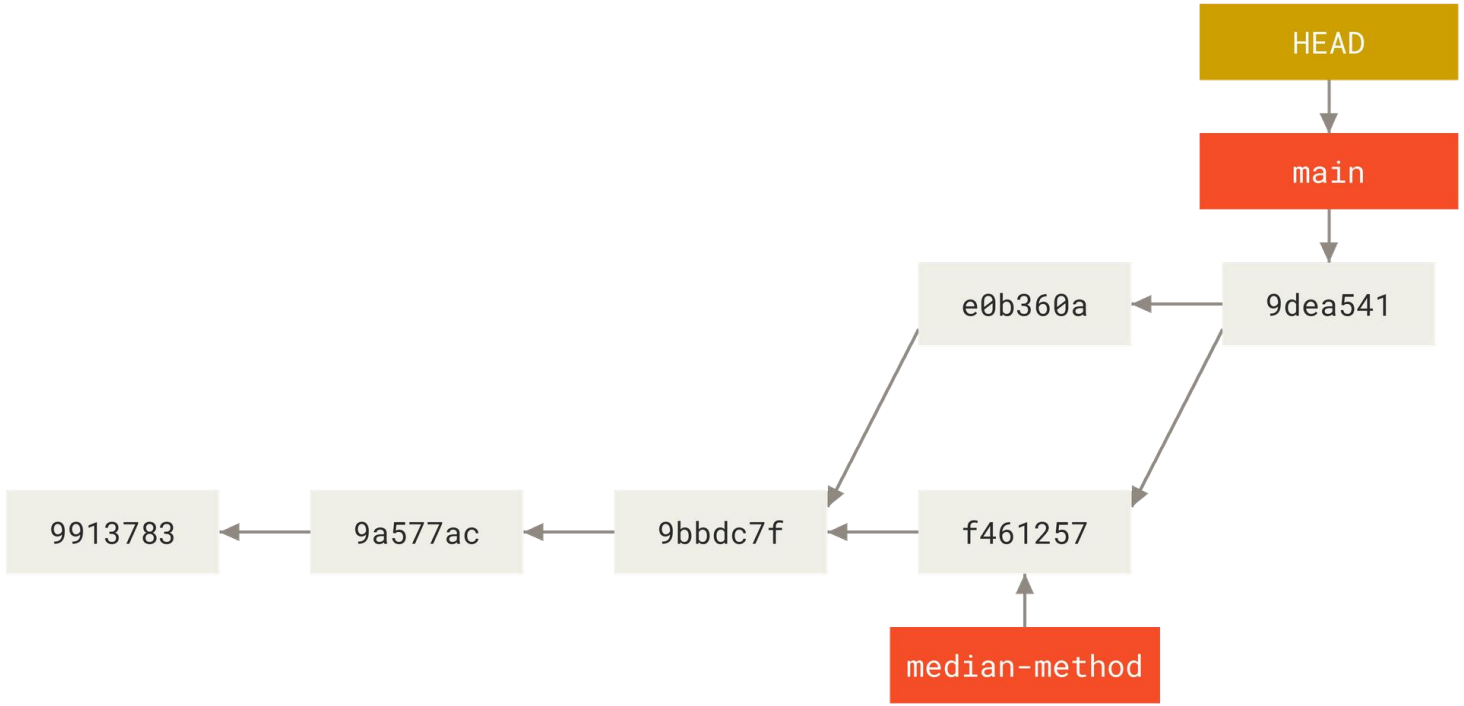
Three-way merge: Git combines both tips, using their common ancestor



Pro Git 3.2 Basic Branching and Merging

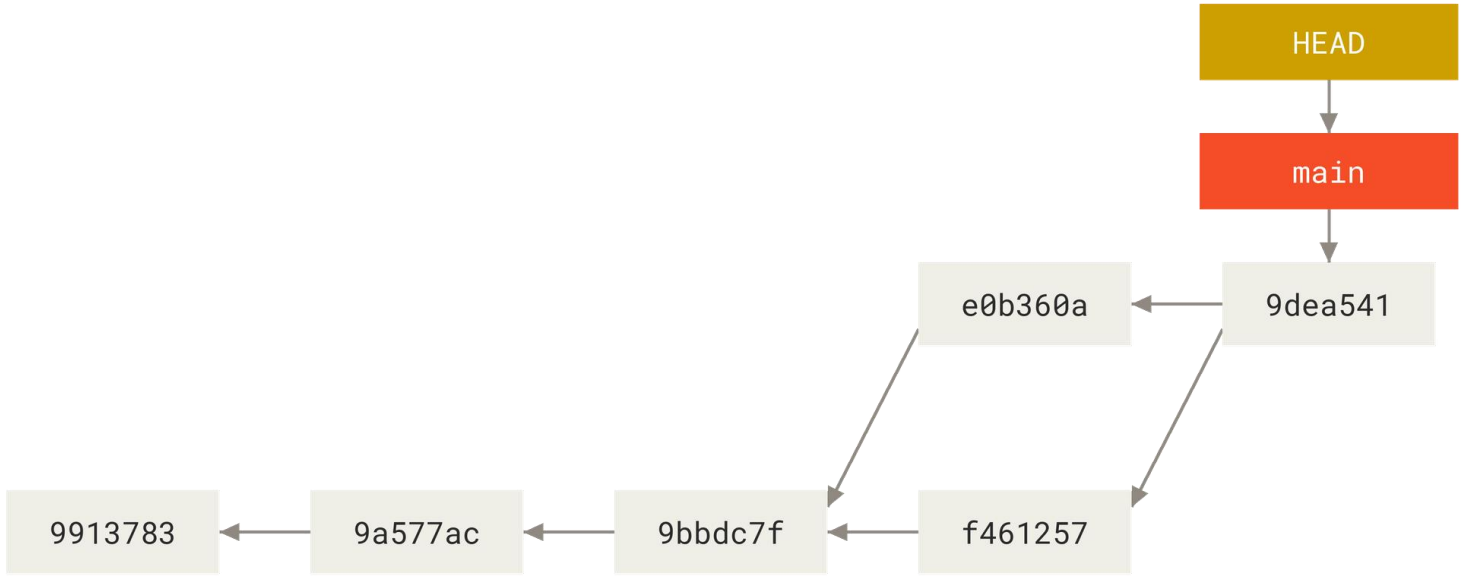
CC BY NC SA

```
$ git merge median-method
```



Pro Git 3.2 Basic Branching and Merging

```
$ git branch -d median-method    # the label goes, the commits stay
```



Pro Git 3.2 Basic Branching and Merging

Merge conflicts

```
$ cd ../cp3b
$ git merge coauthor-marc
CONFLICT (content): Merge conflict in paper.md

<<<<<<< HEAD
Temperatures exceeded 25 °C in 3 of 10 measurements.
=======
Afternoon temperatures regularly exceeded 25 °C.
>>>>>>> coauthor-marc
```

- Decide what the text should say, write it, delete the three marker lines
- Then `git add paper.md` and `git commit`
- Panic? `git merge --abort` takes you back to before the merge

► **Live demo**

Pro Git 3.2 Basic Branching and Merging

CC BY NC SA

Advanced branching

CC BY NC SA

Stash: park unfinished work

```
$ cd ../cp4
$ git switch main        # refused: changes would be overwritten
$ git stash              # park your changes
$ git switch main        # urgent fix, commit
$ git switch new-figure
$ git stash pop          # your work is back
```

- `git stash list` shows what is parked

► **Live demo**

Pro Git 7.3 Stashing and Cleaning

Rebase: replay commits on a new base

```
$ git rebase main        # replay your branch on top of main
$ git rebase -i main     # squash, reword or reorder your commits
```

- Same changes, new commits: every hash changes
- Interactive rebase turns "wip", "fix", "fix again" into one clean commit
- Afterwards, merging into main is a fast-forward

Golden rule: never rebase commits that someone else already has

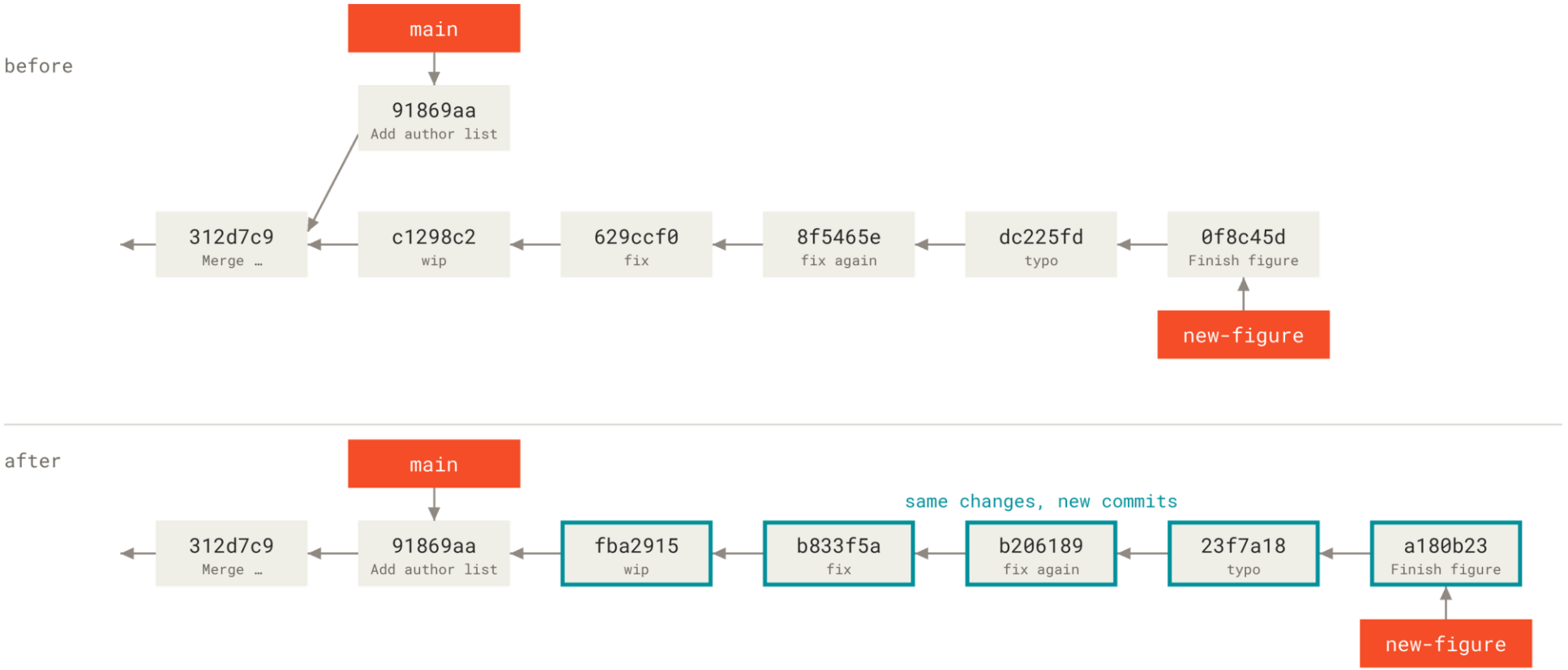
► **Live demo**

Pro Git 3.6 Rebasing

Pro Git 7.6 Rewriting History

CC BY NC SA

```
$ git rebase main    # replay the commits of new-figure on top of main
```



Pro Git 3.6 Rebasing

Tags: mark a version forever

```
$ git switch main          # tags go on main
$ git tag -a submitted-v1 -m "Version submitted to the journal"
$ git tag
$ git show submitted-v1
$ git diff submitted-v1    # what changed since submission?
```

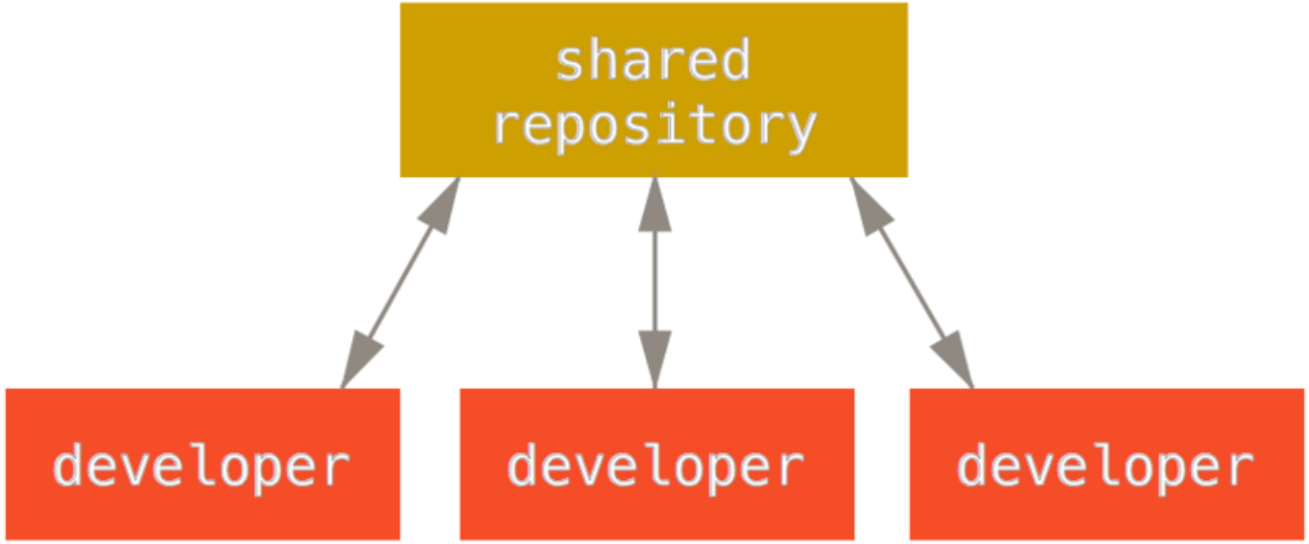
- Annotated tags (`-a`) record who tagged, when and why
- The tag always points to the exact state you submitted

► **Live demo**

Pro Git 2.6 Tagging

CC BY NC SA

Where to go next


shared
repository
developer
developer
developer

A first look at sharing

```
$ cd ..                # back to the course folder
$ git clone cp4 my-copy
$ cd my-copy
$ git log --oneline    # the complete history came along
$ git remote -v        # the copy knows where it came from
```

- This connection between repositories is the basis of working with others
- Remotes, `push` and `pull`: the natural next step after today

Pro Git 2.1 Getting a Git Repository Pro Git 2.5 Working with Remotes

Command line, GUI and large data

- Everything you did today also works in a GUI, on the same repository: switch at any time
- Git for Windows includes `git gui` (staging, committing) and `gitk` (history, branches)
- Many more clients: https://git-scm.com/downloads/guis

- Git is made for text. For large datasets or binary files: Git LFS, git-annex, DVC

Pro Git A1.1 Graphical Interfaces

Keep learning with Pro Git

Free online: git-scm.com/book

- Chapters 2 and 3: consolidate everything from today
- Chapter 7: Stashing and Cleaning, Rewriting History, Reset Demystified
- Chapter 10: Git Internals, for the curious
- Section 2.5 and chapter 5: remotes and working with others

- Also worth looking up: `git blame` and `git cherry-pick`

A complementary course

K295 – Branch into Git: Collaborate Effectively with Version Control

- Another introductory course, from a different perspective
- Starts from GitHub and working together with Git
- Part of the Lead Campus course offer: login.lead-campus.ch
- Course details: course page, or scan the QR code

Git resources

- Git website download page (Git, GUI clients) *https://git-scm.com/downloads*
- Git website documentation page
  (Reference manual, “Pro Git” book, videos) *https://git-scm.com/doc*
- Git cheat sheet *https://git-scm.com/cheat-sheet*
- GitHub Git cheat sheets *https://training.github.com/*

Unless otherwise noted, images are from Pro Git Book (*https://git-scm.com/book*) and are licensed under CC BY-NC-SA 3.0.

This presentation is licensed under CC BY-NC-SA 4.0.

Lib4RI – Excellent Services for Excellent Research.

https://www.lib4ri.ch
info@lib4ri.ch
T: + 41 58 765 57 00