

git Tutorial

Scientific Programming with Python

Nicola Chiapolini

University of Zurich
Faculty of Science

23 June 2025

Based on talk by Emanuele Olivetti https://github.com/emanuele/introduction_to_Git
 ⓘ ⓘ This work is licensed under the *Creative Commons Attribution-ShareAlike 4.0 License*.

Introduction

single+local

Branches

multi+remote/central

Behind the Scenes

MINI DEMO

Motivation to use Version Control

Problem 1

“Help! my code worked yesterday, but I can't recall what I changed.”

- ▶ track modifications
- ▶ access old version

Problem 2

“We would like to work together, but we don't know how!”

- ▶ concurrent editing
- ▶ merging
- ▶ development versions

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Intermezzo: Branches

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

Introduction

single+local

Branches

multi+remote/central

Behind the Scenes

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Intermezzo: Branches

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

Survey: Version Control

- ▶ Q1: Have you heard about *version control*?
- ▶ Q2: Do you use a version control software (cvs, svn, hg, bzr, git)?
- ▶ Q3: How much experience do you have with git?

Survey: Version Control

- ▶ Q1: Have you heard about *version control*?
- ▶ Q2: Do you use a version control software (cvs, svn, hg, bzr, git)?
- ▶ Q3: How much experience do you have with git?

Survey: Version Control

- ▶ Q1: Have you heard about *version control*?
- ▶ Q2: Do you use a version control software (cvs, svn, hg, bzr, git)?
- ▶ Q3: How much experience do you have with git?

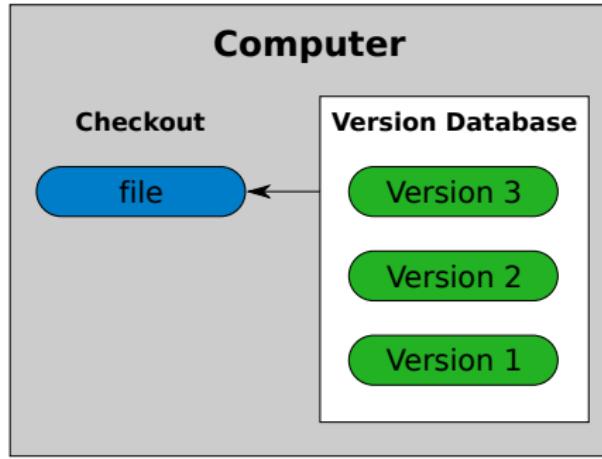
Uses for git

“*Version control* is a system that records changes to a file or set of files over time so that you can recall specific versions later.”

– <https://git-scm.com/book>

- ▶ checkpoints/backups/releases
- ▶ document developer effort
- ▶ collaboration across the globe
- ▶ for anything that's text
 - ▶ code
 - ▶ thesis/papers
 - ▶ system config files ([vcsh](#), [etckeeper](#))

Version Control: Local

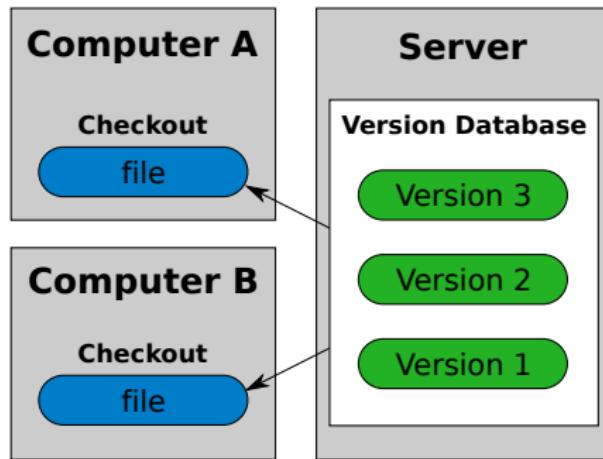


checkout working tree

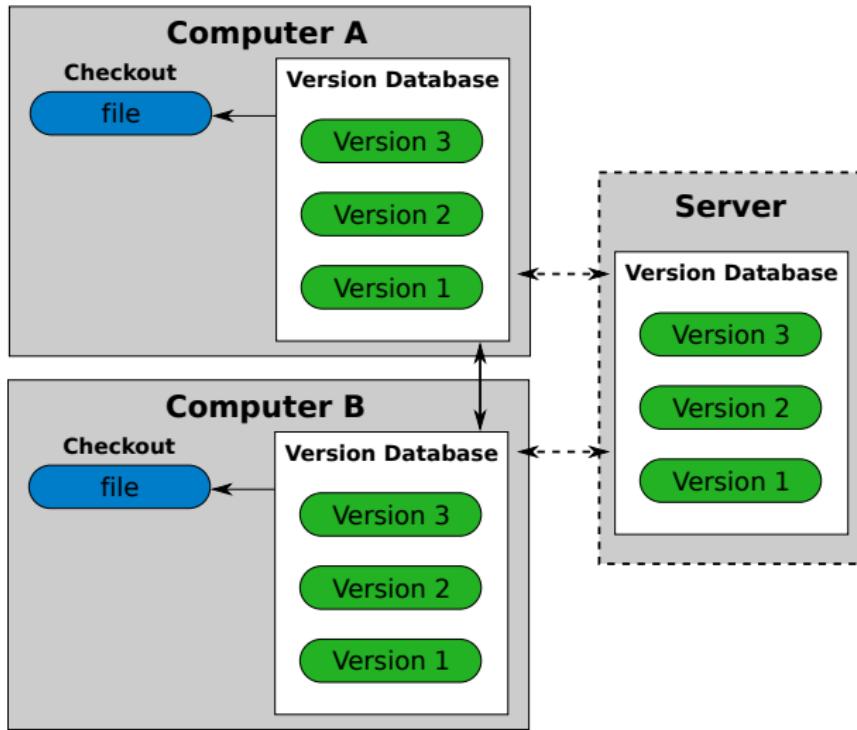
version database repository

There is always only one version of a file present in the working tree. Version Control allows you to change that file to different versions stored in the repository.

Version Control: Central



Version Control: Distributed



git: Help

```
usage: git [...]
       <command> [<args>]
```

These are common Git commands used in various situations:

[...]

'git help -a' and 'git help -g' list available subcommands and some concept guides. See 'git help <command>' or 'git help <concept>' to read about a specific subcommand or concept.

The Glossary

git help glossary
explains many terms that might be puzzling to new users.

git: Introduce yourself

```
git config --global user.name "Nicola Chiapolini"
```

```
git config --global user.email "nchiapol@physik.uzh.ch"
```

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Intermezzo: Branches

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

single+local: Init

```
git init
```

- ▶ Creates an empty git repository with one branch
 - ▶ a branch stores a line of development (see next section)
 - ▶ default branch is called `master`
- ▶ Creates the git directory: `.git/`
- ▶ Your prompt may change.
(If you added `$_git_ps1`)

working tree

index

master

- ▶ does not change your files

single+local: Init

```
git init
```

- ▶ Creates an empty git repository with one branch
 - ▶ a branch stores a line of development (see next section)
 - ▶ default branch is called `master`
- ▶ Creates the git directory: `.git/`
- ▶ Your prompt may change.
(If you added `$_git_ps1`)

working tree

index

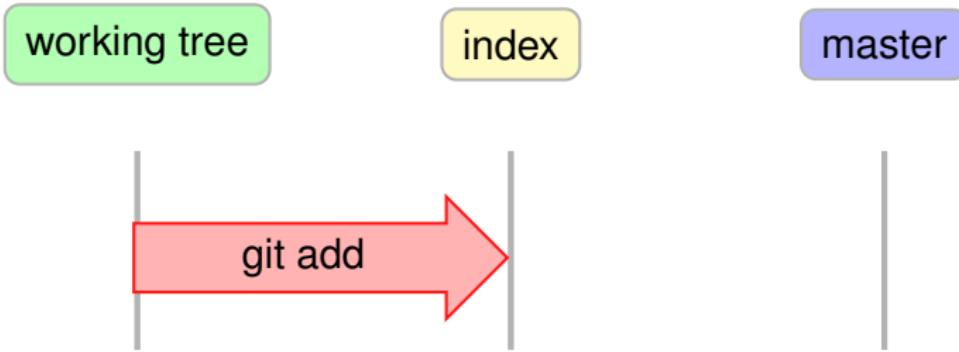
master

- ▶ does not change your files

single+local: Add

```
git add file1 [file2 ...]
```

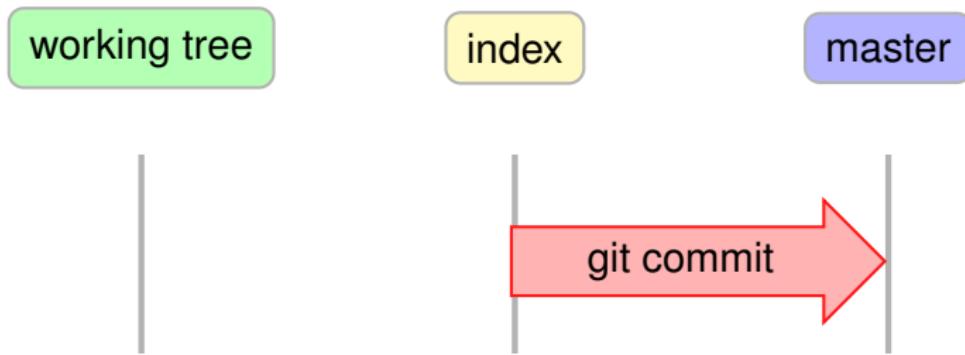
- ▶ Adds new files to be tracked by git
- ▶ Adds changes from working tree for next commit (**Confusion!**)
- ▶ DOES NOT add info on file permissions other than `exec/noexec`
- ▶ DOES NOT add directories *per se*.



single+local: Commit

```
git commit [-m "Commit message."]
```

Records changes from the index to master.

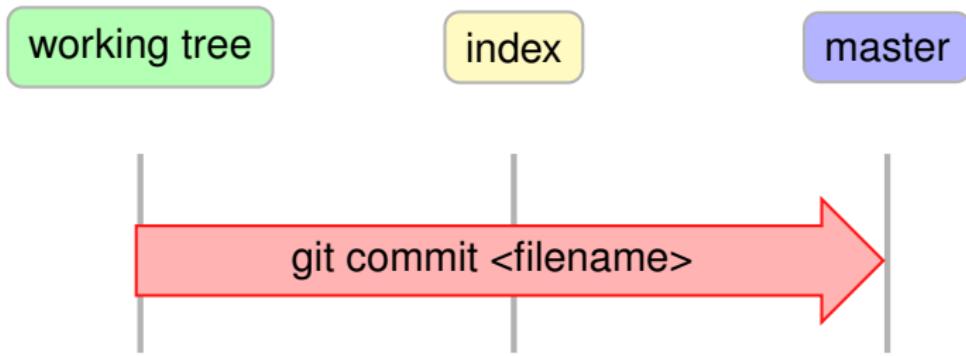


Config Tip: `git config [-global] core.editor "/usr/bin/kate"`

single+local: Direct Commit

```
git commit file1 file2 [-m "Commit message."]
```

Records all changes of `file1`, `file2` from working tree and index to master.



```
git commit -a[m "Commit message."]
```

Records all changes in working tree and index. *Be Careful!*

single+local: Diff

```
git diff [filename|...]
```

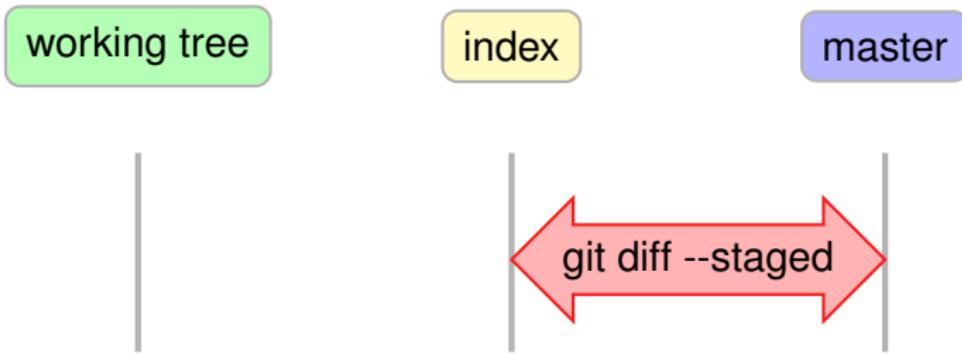
Shows changes between *working tree*
and *index*



single+local: Diff Staged

How do I see what is staged?

`git diff --staged` shows differences
between index and last commit.



single+local: Commit History

```
git log [--oneline] [--patch] [--graph] [file|branch]
```

Shows the history of a file or branch.



Config Tip: `git config [-global] log.date "iso"`

single+local: Old Changes

```
git diff <commit A> <commit B>
git show <commit>
```

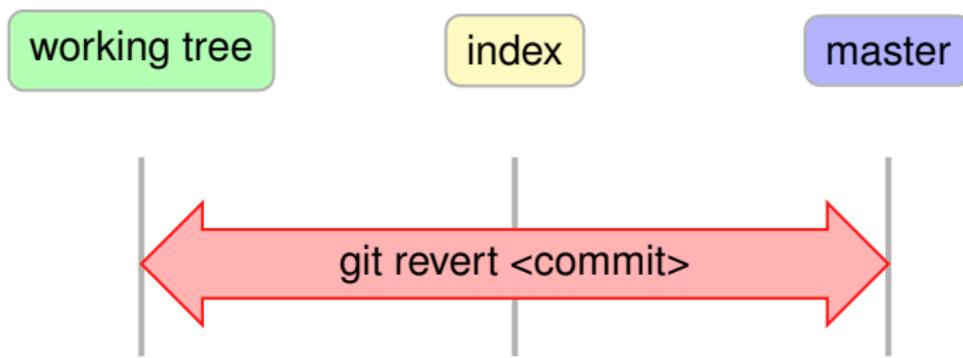
Shows the changes stored in commits.



single+local: Revert Commits

```
git revert <commit>
```

Creates a new commit reverting changes from <commit>.



Warning: If you are not reverting the last changes to a file, this will most likely create a conflict. See later on how to solve them.

single+local: Graphic Logs

qgit (or gitg or ...)

GUI to browse the git repository.



single+local: Changing Version (single file)

```
git restore [--source=...] <file>
```

working tree

index

master

A diagram illustrating the flow of the `git restore` command. It features three vertical grey lines representing the boundaries between the working tree, index, and master. A large red arrow points from the working tree towards the index, with the command `git restore <file>` written in white inside the arrow.

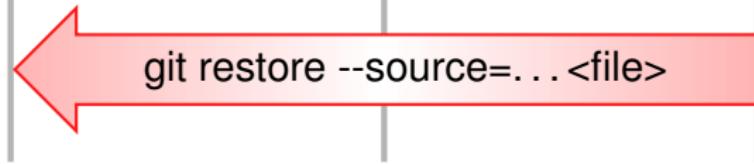
single+local: Changing Version (single file)

```
git restore [--source=...] <file>
```

working tree

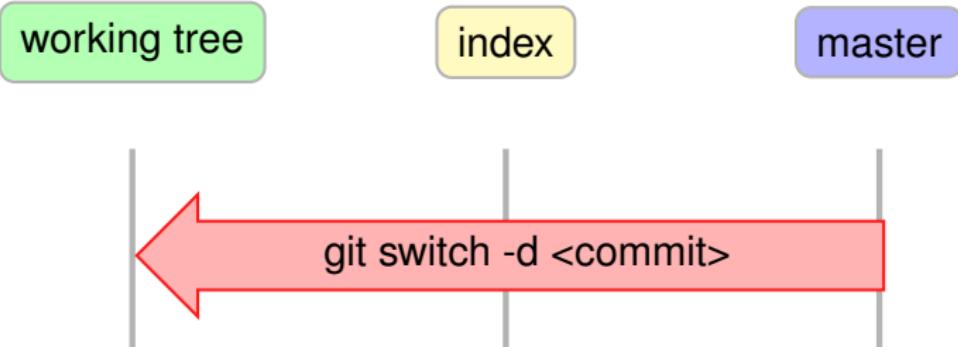
index

master



single+local: Changing Version (all files)

```
git switch --detached <commit>
```



single+local: (Re)move

Warning: whenever you want to *remove*, *move* or *rename* a tracked file use git:

```
git rm <filename>
```

```
git mv <oldname> <newname>
```

Remember to *commit* these changes!

```
git commit -m "File (re)moved."
```

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Intermezzo: Branches

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

single+local: Pitfalls

- ▶ don't store large binary files in git
 - ▶ a copy of the full file will be stored even on small changes
 - ▶ removing the file with `git rm` will not free much space
 - ▶ use `git-annex` or `git-lfs`
- ▶ don't put sensitive data into committed files
 - ▶ even if you change the file, the sensitive data is still in the history
- ▶ don't blindly follow instructions from random websites
 - ▶ there is a lot of bad advice for exotic things
 - ▶ try instructions on a test repository

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Intermezzo: Branches

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

Branches: Active Lines of Development

- ▶ So far: linear history stored in `master` branch
- ▶ could work on several branches in parallel
- ▶ separate version of each file in each branch

```
* 9bce (HEAD -> master) bugfix
| * af63 (binary) optimize
| * b1f4 add documentation
* | 4c48 increase search space
| * d458 use binary search
|/
* 1209 brute force search
* f96f initial commit
```

Why

- ▶ Develop new features without breaking the running version
- ▶ Test different ideas starting from the same base
- ▶ Synchronise with a remote server (see next section)

Branches: Common Commands

Create a new branch `git branch <branch-name>`

Switch to a different branch `git switch <branch-name>`

Create + switch in one go `git switch -c <branch-name>`

List branches `git branch [--list] [-a]`

Integrate changes `git merge <branch-name>`
includes all changes from `branch-name`
into the currently checked-out branch

Delete a branch `git branch -d <branch-name>`

Note:

Normal git commands only affect the branch currently checked out.

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

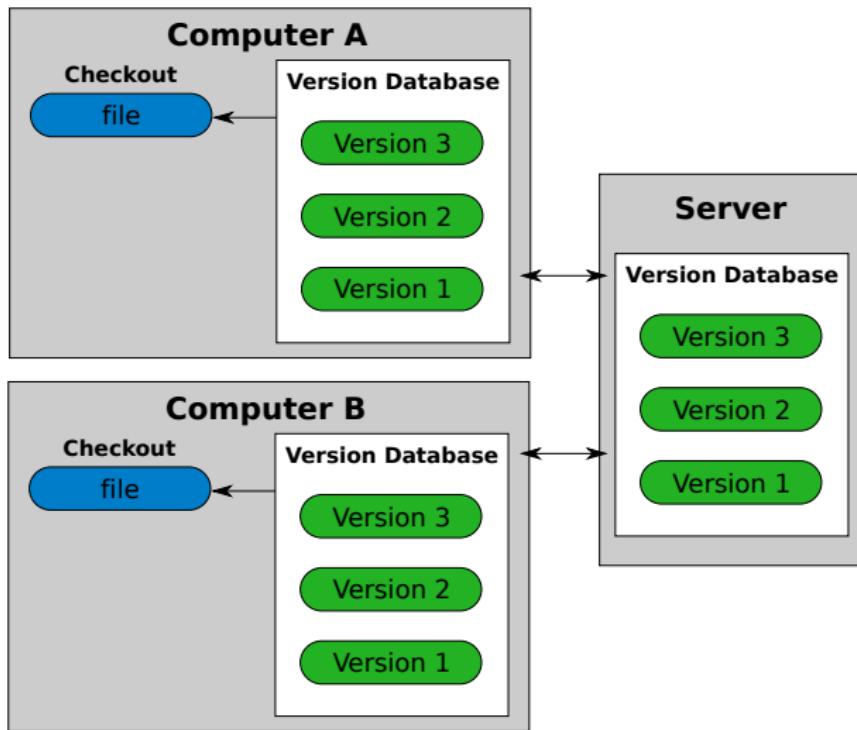
Intermezzo: Branches

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

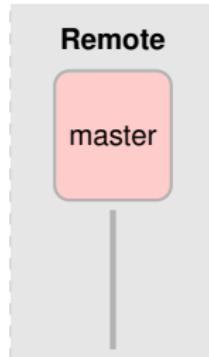
multi+remote/central: Setup



multi+remote/central: Clone

```
git clone <URL>
```

Creates **two** local copies of the **whole** remote branch.



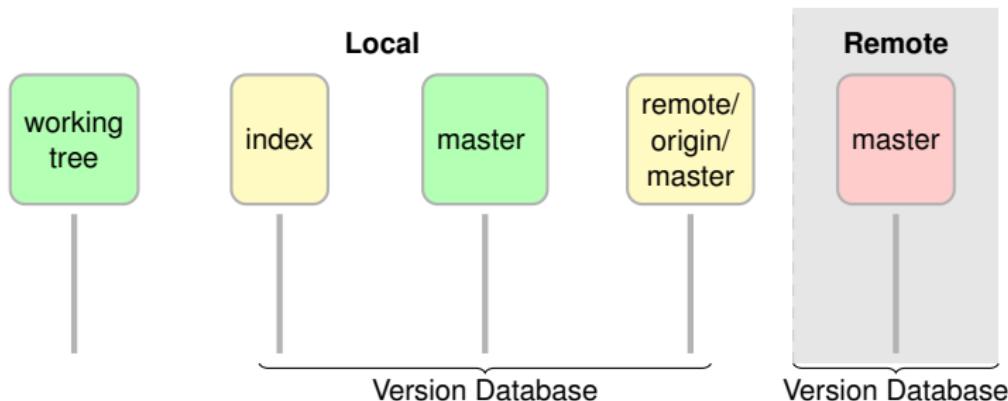
Hint

`git remote -v` shows **name** and **URL** of the remote repository.

multi+remote/central: Clone

```
git clone <URL>
```

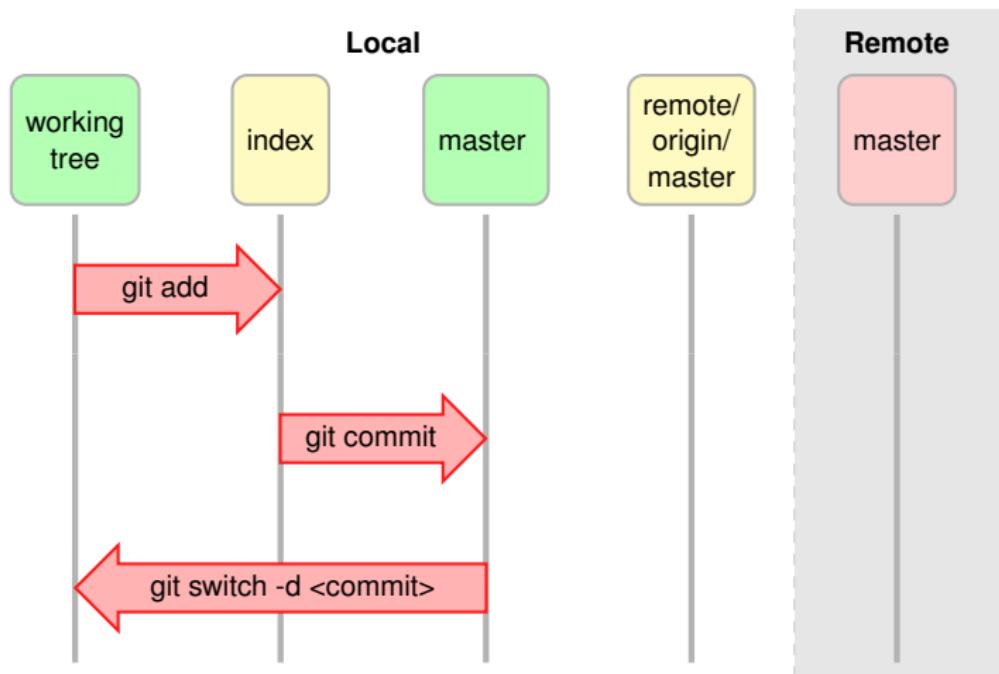
Creates **two** local copies of the **whole** remote branch.



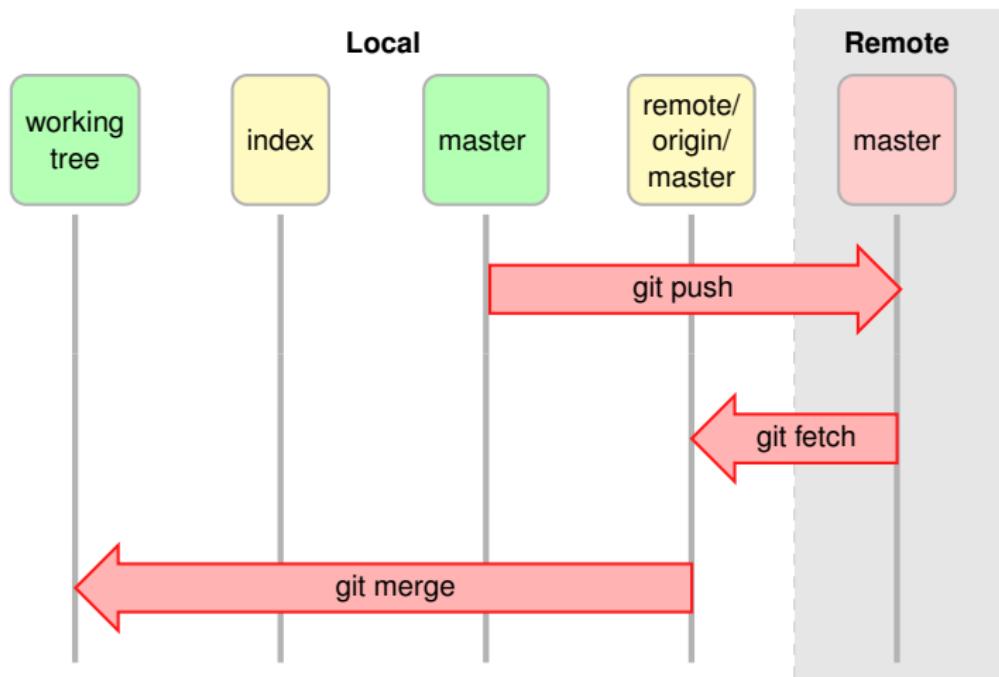
Hint

`git remote -v` shows **name** and **URL** of the remote repository.

multi+remote/central: Commands



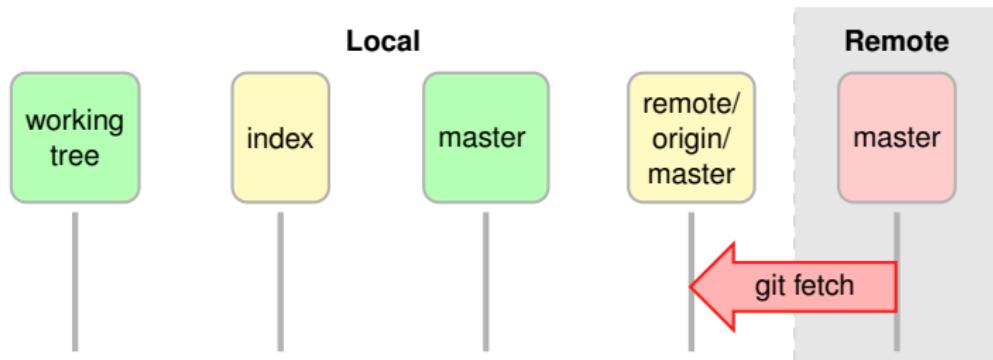
multi+remote/central: Commands



multi+remote/central: Fetch

`git fetch`

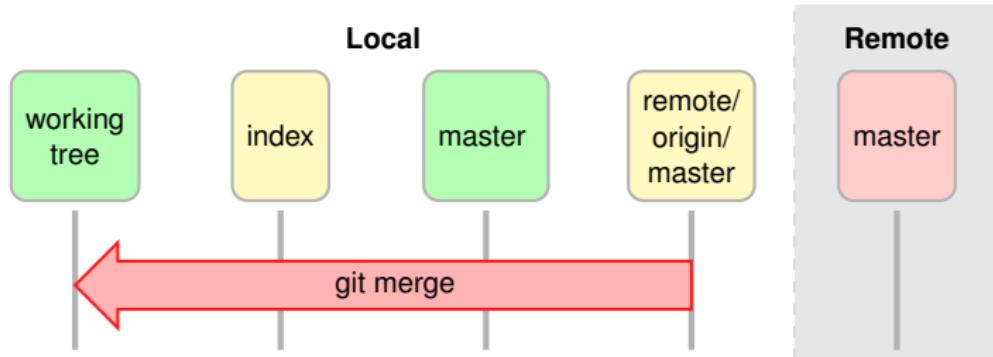
- ▶ Updates origin master from remote master
- ▶ local master, index and working tree not changed



multi+remote/central: Merge

git merge

- ▶ combines changes from both sources
- ▶ **Warning:** can generate *conflicts*!



git fetch + git merge = git pull

multi+remote/central: Conflicts

Conflict!

```
...
<<<<<< yours:sample.txt
Conflict resolution is hard;
let's go shopping.
=====
Git makes conflict resolution easy.
>>>>> theirs:sample.txt
...
```

multi+remote/central: Resolving Conflicts

1. See where conflicts are:

```
git diff
```

2. Edit conflicting lines.
3. Add changes to the index:

```
git add file1 [...]
```

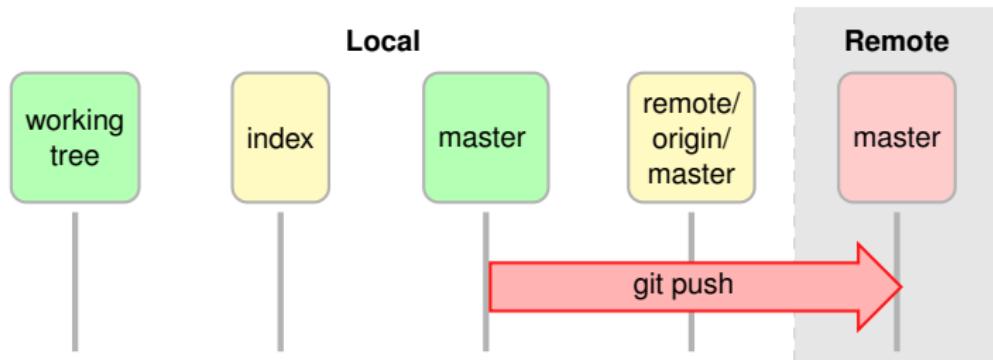
4. Commit changes:

```
git commit -m "Conflicts solved."
```

multi+remote/central: Push

`git push`

- ▶ Updates *remote master*.
- ▶ Requires `fetch+merge` first.



Introduction

single+local

Branches

multi+remote/central

Behind the Scenes

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Intermezzo: Branches

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

Lessons Learned

- ▶ pushing to a central server can be problematic
→ a setup where everybody pulls can help here
- ▶ be careful, what you commit
(no `git add *`)

Reference: Setting up a central remote repository.

access to repository via ssh

On *remote* server create **bare+shared** repository:

- ▶ `mkdir newproject`
- ▶ set up proper *group* permissions: `chmod g+rws newproject`
- ▶ `cd newproject`
- ▶ `git --bare init --shared=group`

Everybody clones:

```
git clone ssh://remote.example.com/path/newproject
```

Introduction

single+local

Branches

multi+remote/central

Behind the Scenes

Outline

Introduction

Single developer + local repository

Demo/Exercise: single+local

Intermezzo: Branches

Multiple developers + remote central repository

Demo/Exercise: multi+remote/central

Behind the Scenes

Behind the Scenes: Setup

```
git init; git add [...]; git commit -m "A: init"
```

a

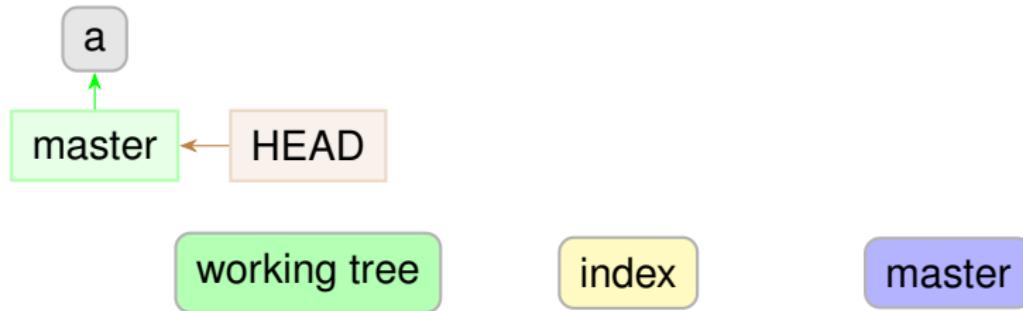
working tree

index

master

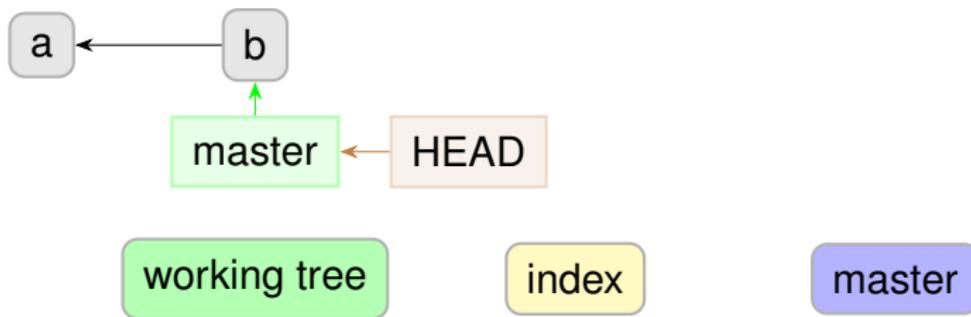
Behind the Scenes: Setup

```
git init; git add [...]; git commit -m "A: init"
```



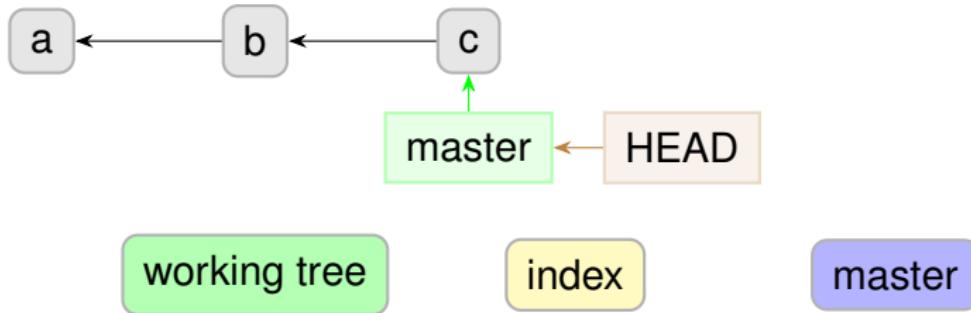
Behind the Scenes: Setup

```
git commit -am "B"
```



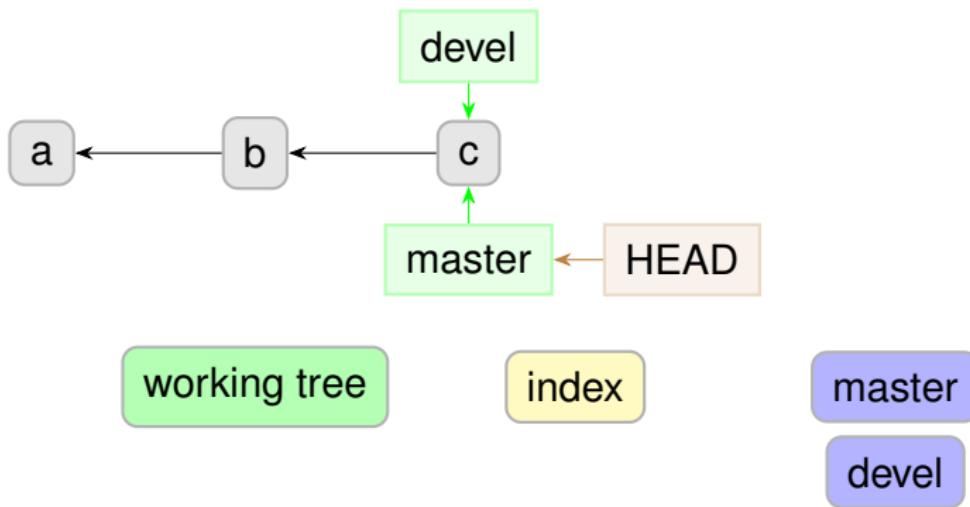
Behind the Scenes: Setup

```
git commit -am "C"
```



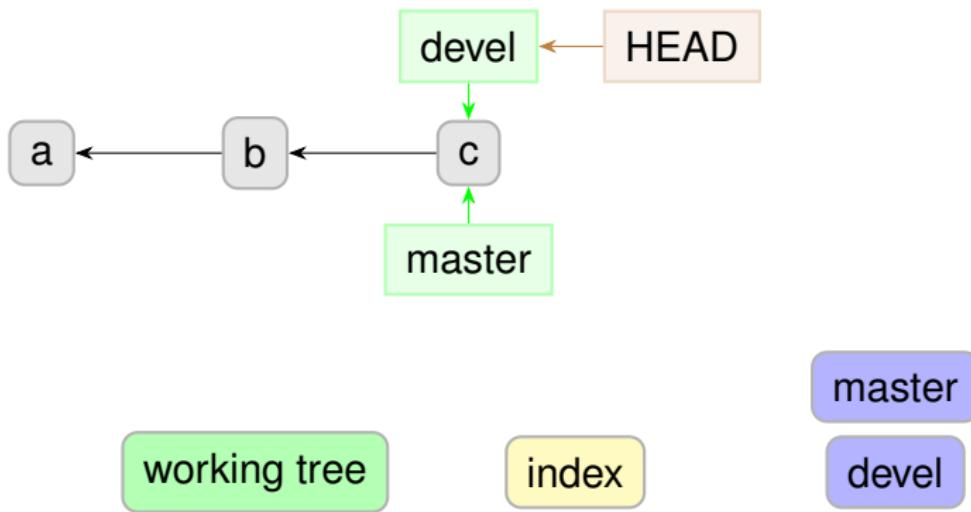
Behind the Scenes: Branches

```
git branch devel
```



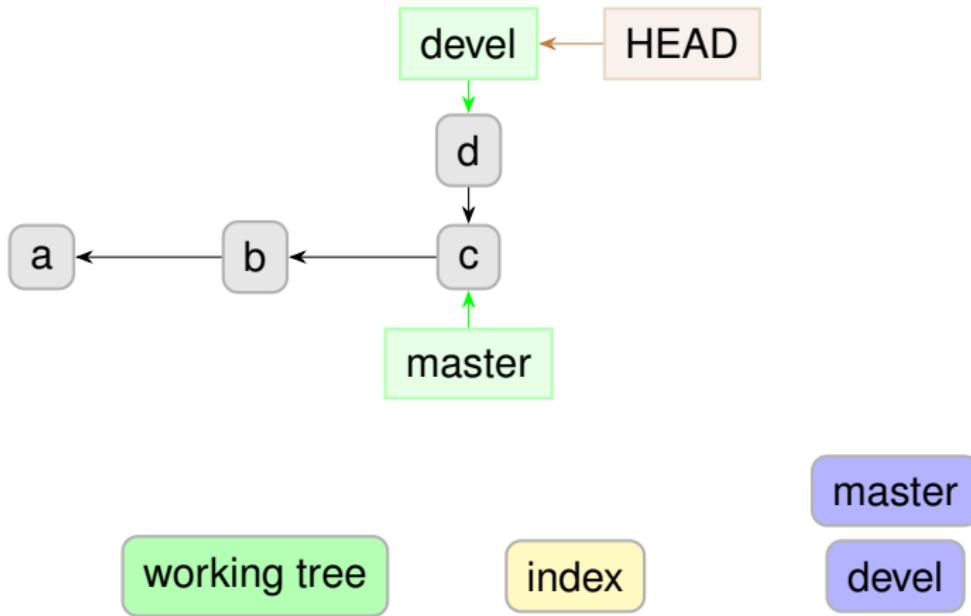
Behind the Scenes: Branches

```
git switch devel
```



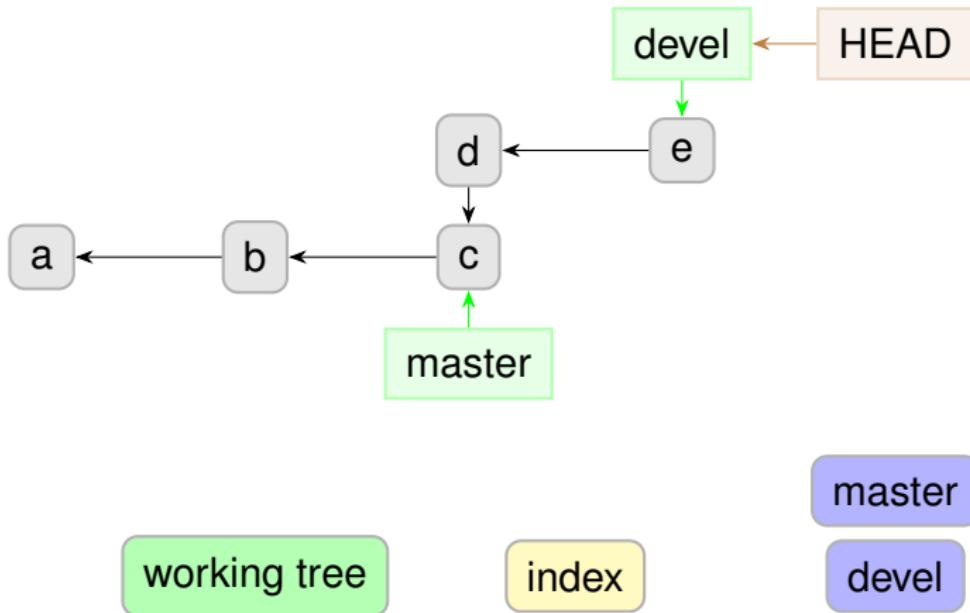
Behind the Scenes: Branches

```
git commit -am "D"
```



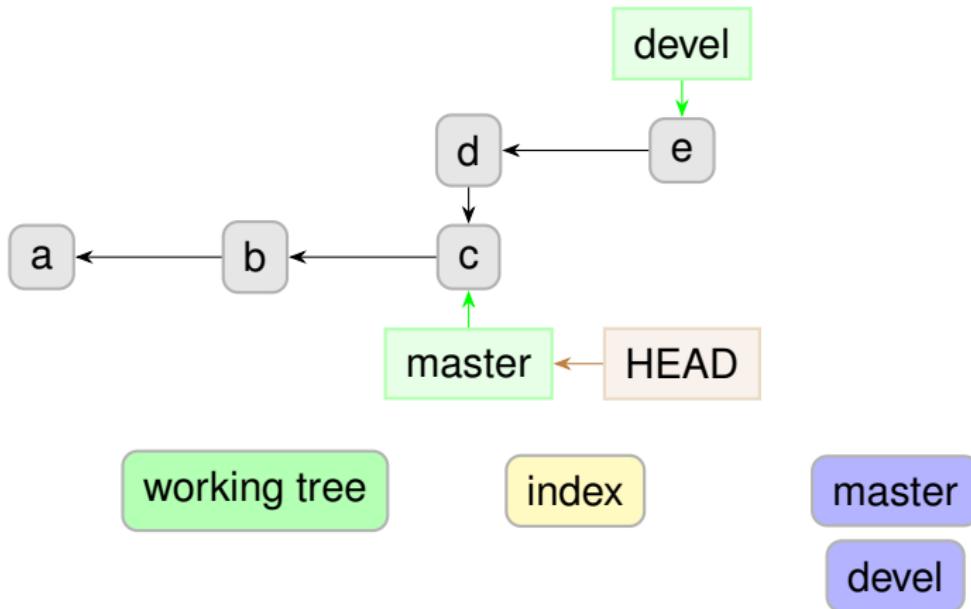
Behind the Scenes: Branches

```
git commit -am "E"
```



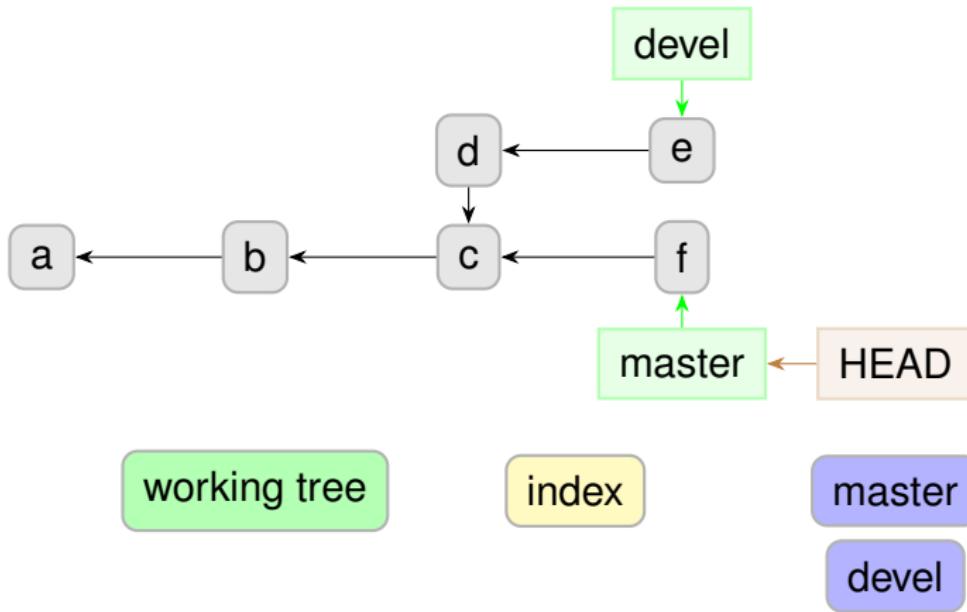
Behind the Scenes: Branches

```
git switch master
```



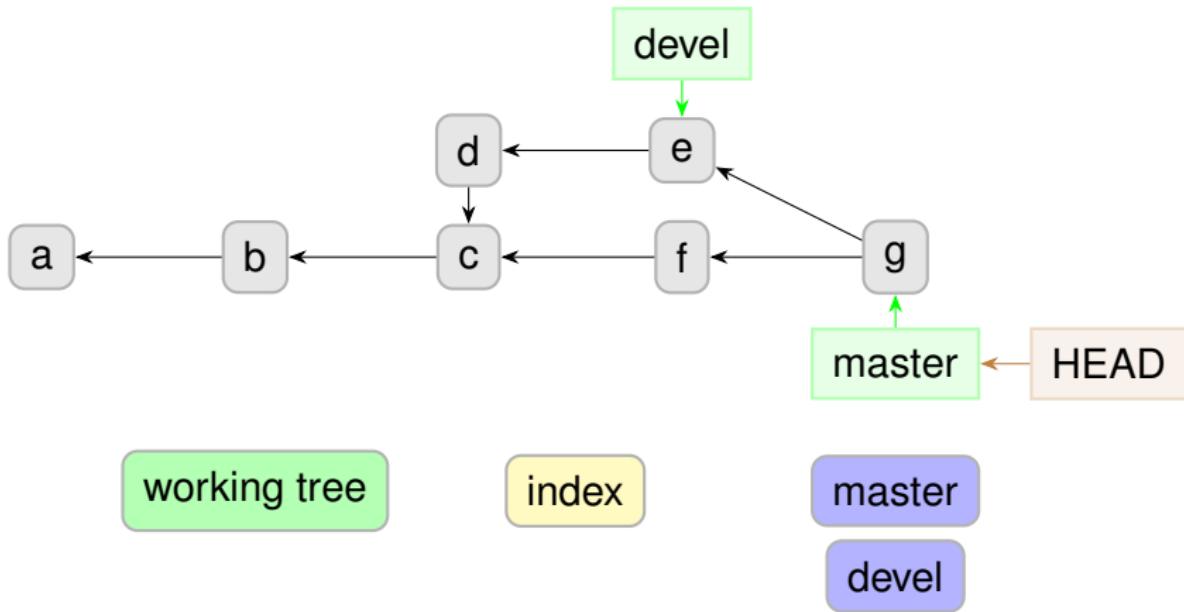
Behind the Scenes: Branches

```
git commit -am "F"
```



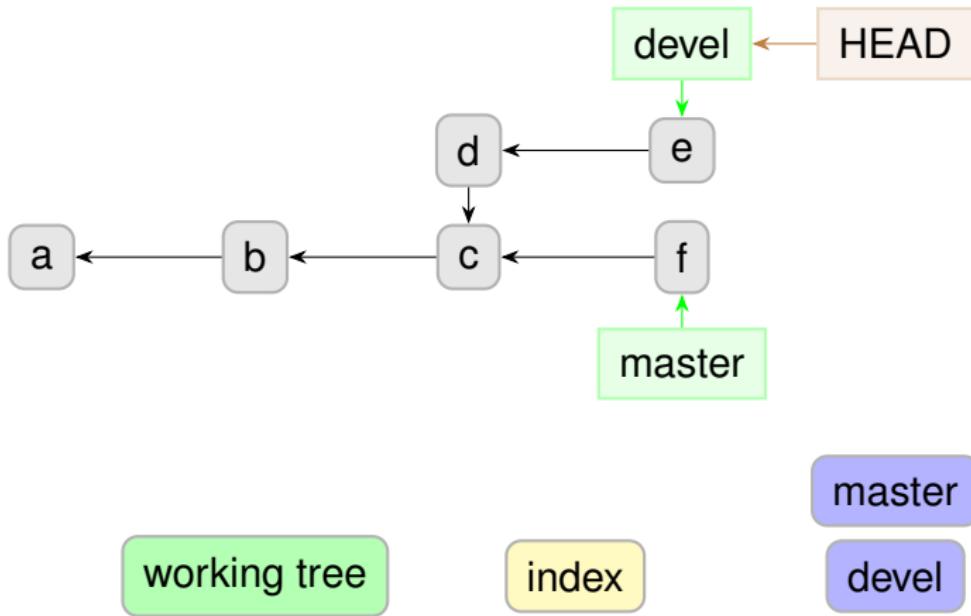
Behind the Scenes: Branches

git merge devel



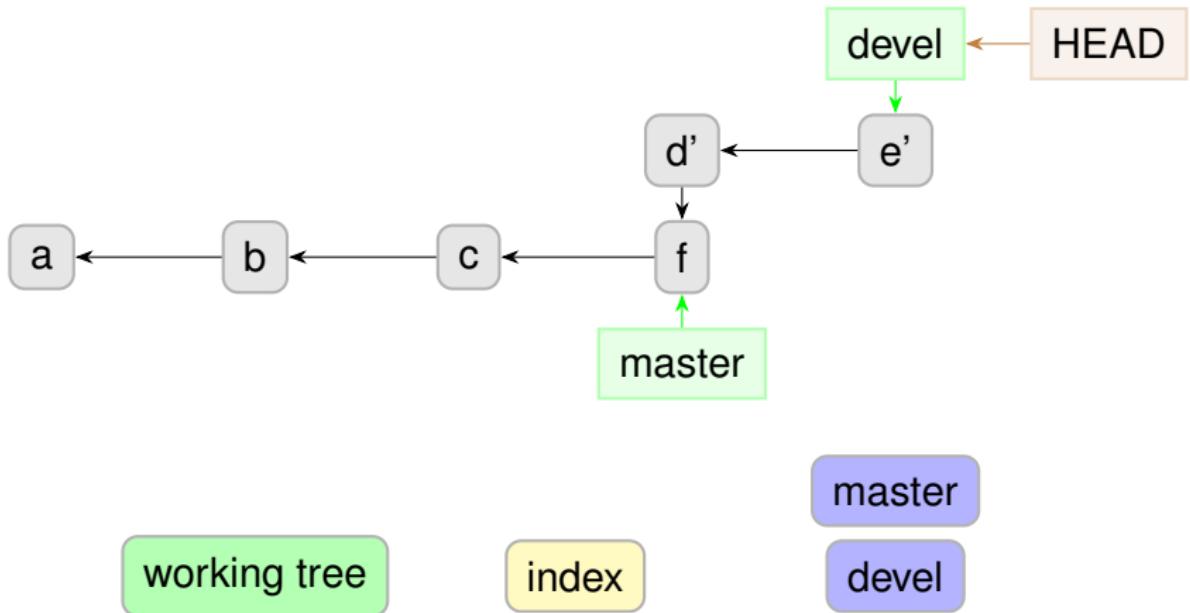
Behind the Scenes: Rebase

git switch devel



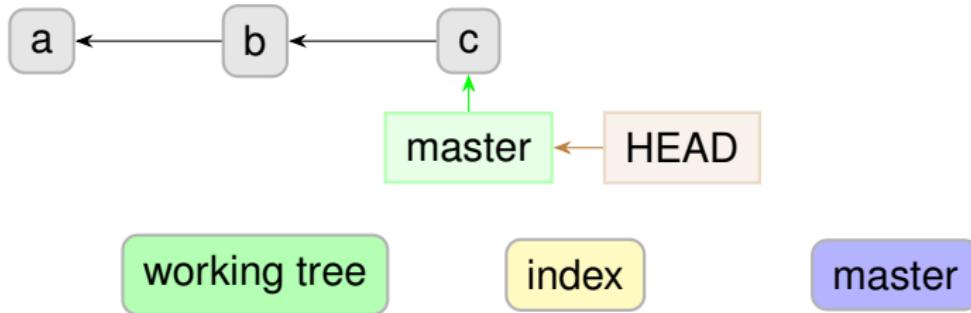
Behind the Scenes: Rebase

```
git rebase master
```



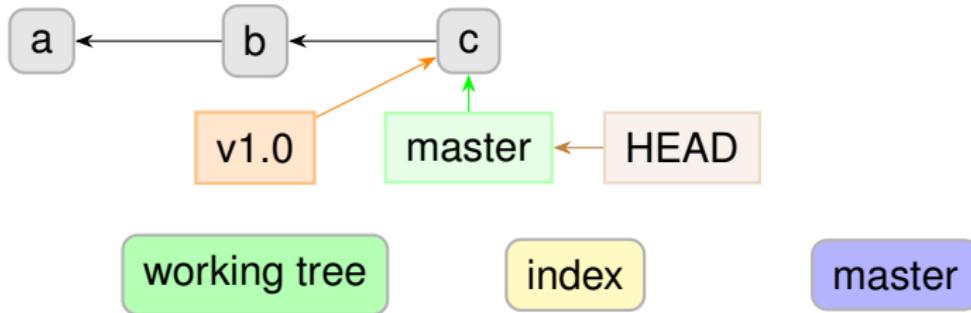
Behind the Scenes: Setup

```
git commit -am "C"
```



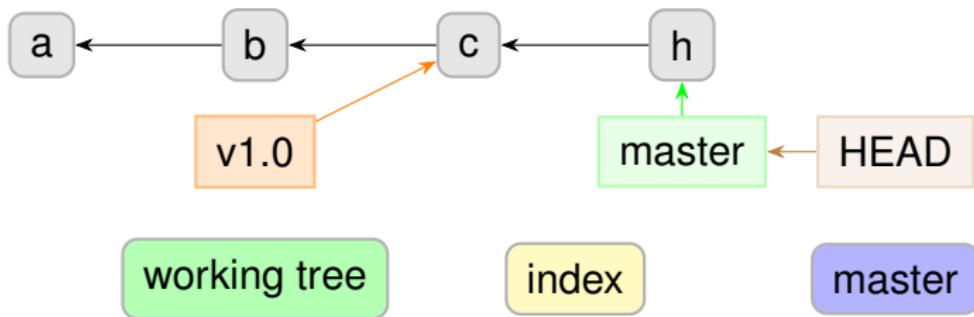
Behind the Scenes: Tags

```
git tag [-m "my message"] v1.0
```



Behind the Scenes: Tags

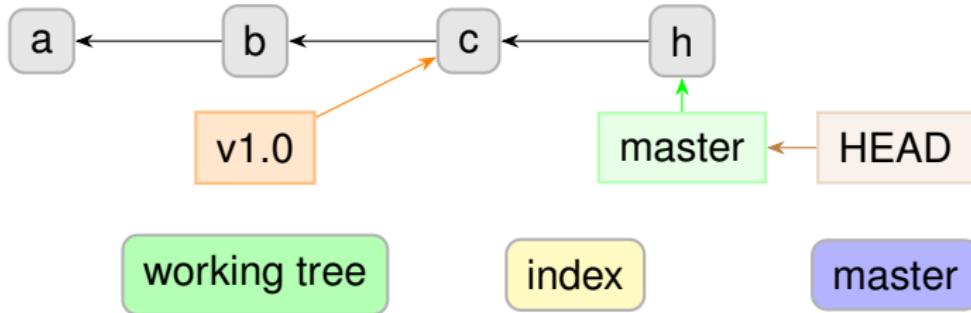
```
git commit -am "H"
```



Behind the Scenes: Tags

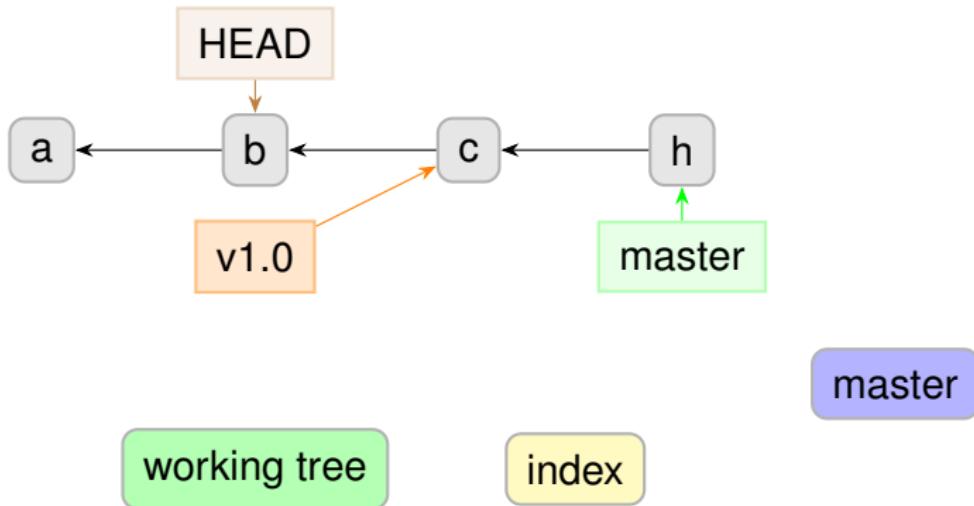
```
git commit -am "H"
```

to **share** tags: `git push origin <tag>` or `git push --tags`



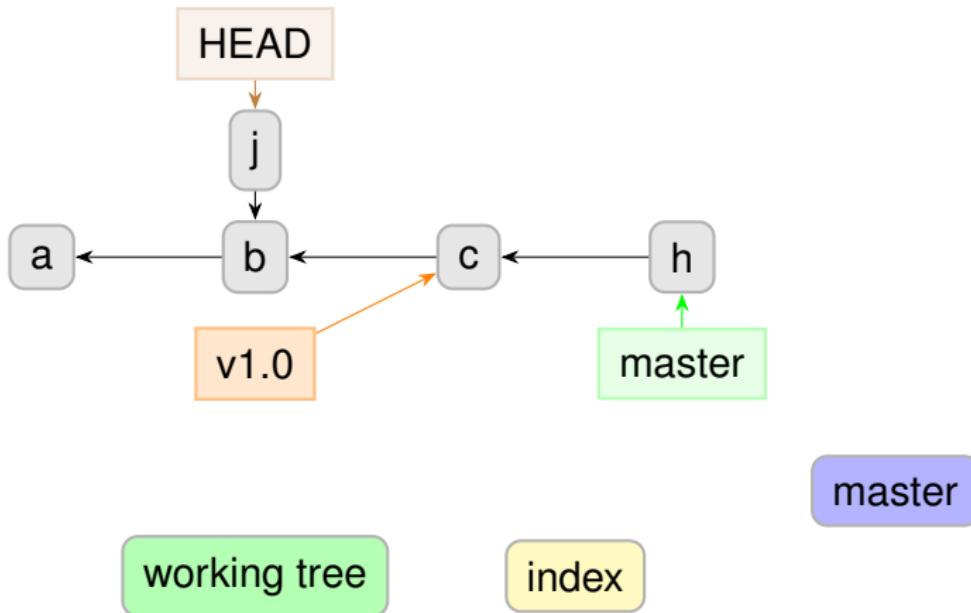
Behind the Scenes: Detached HEAD

```
git switch -d b
```



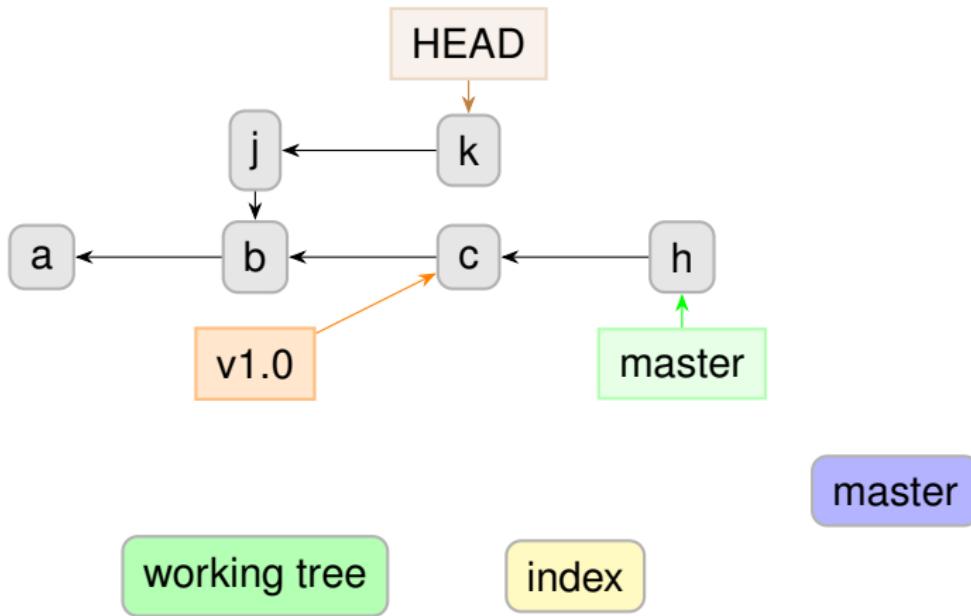
Behind the Scenes: Detached HEAD

```
git commit -am "J"
```



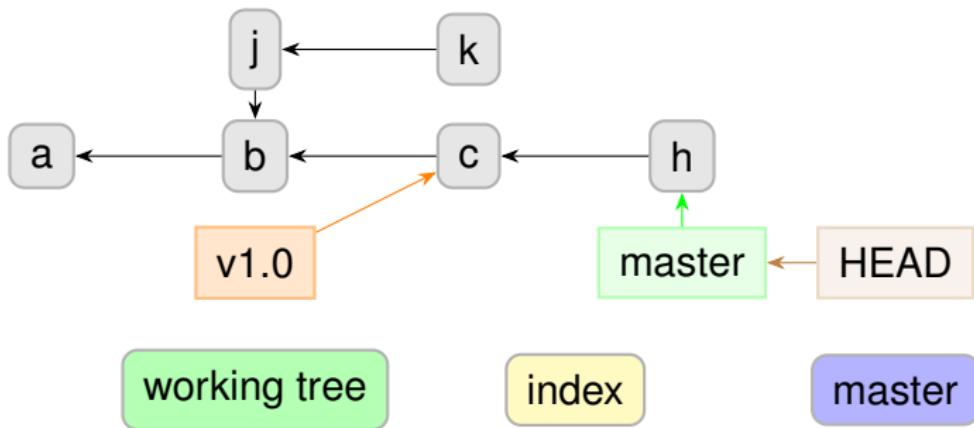
Behind the Scenes: Detached HEAD

```
git commit -am "K"
```



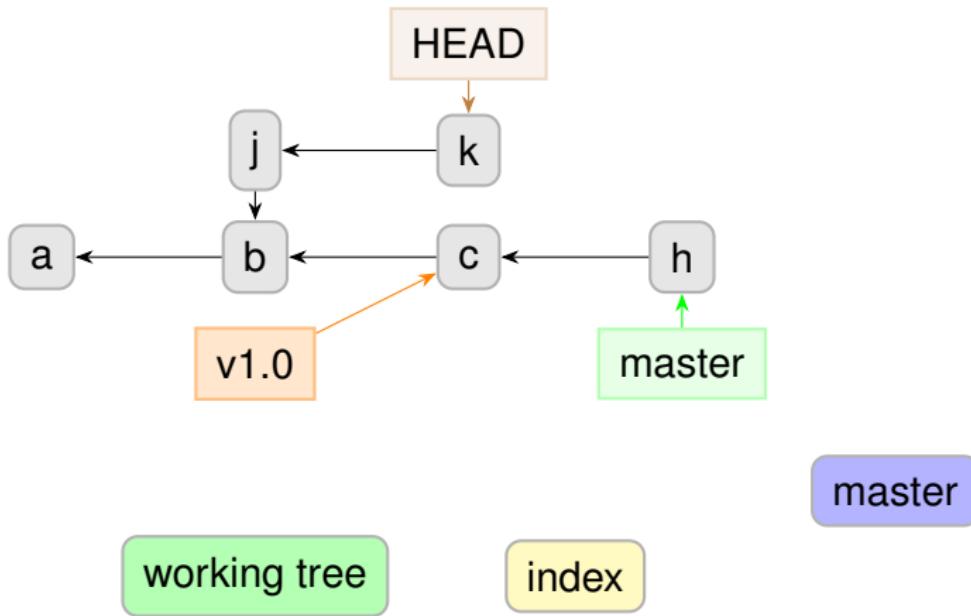
Behind the Scenes: Detached HEAD

```
git switch master
```



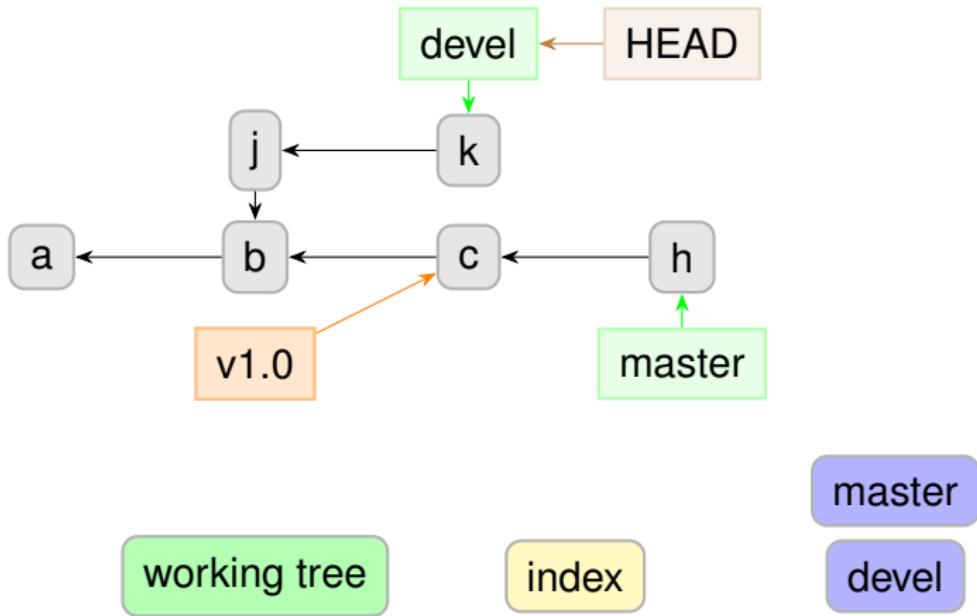
Behind the Scenes: Detached HEAD

```
git commit -am "K"
```



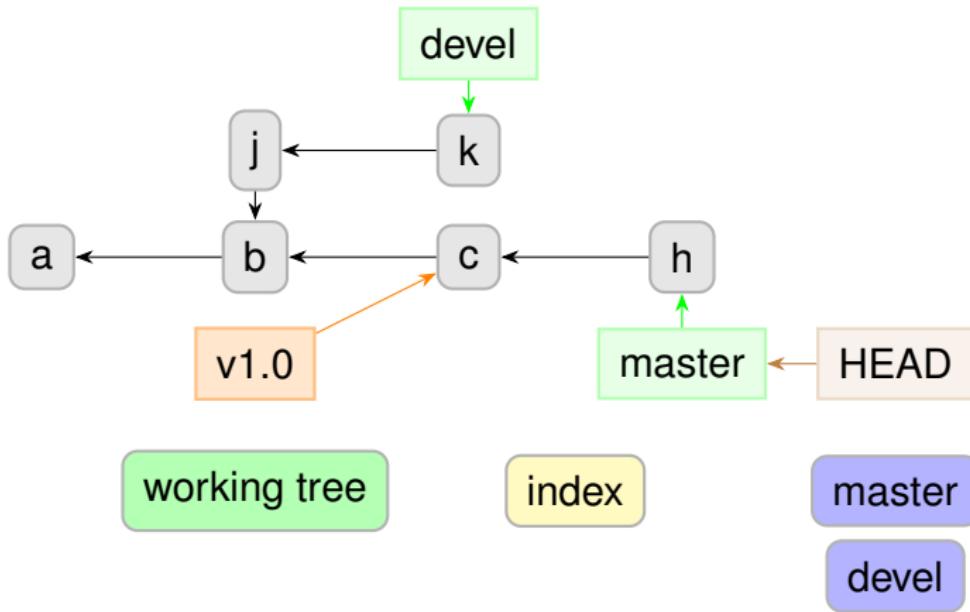
Behind the Scenes: Detached HEAD

```
git switch -c devel
```



Behind the Scenes: Detached HEAD

git switch master



Questions?

Understanding how git works:

- ▶ git foundations, by Matthew Brett:
<http://matthew-brett.github.io/pydagogue/foundation.html>
- ▶ The git parable, by Tom Preston-Werner: <https://tom.preston-werner.com/2009/05/19/the-git-parable.html>

Excellent guides:

- ▶ “Pro Git” book: <https://git-scm.com/book/en/v2> (FREE)
- ▶ git magic:
<http://www-cs-students.stanford.edu/~blynn/gitmagic/>