Version Control with Git

(based on work of Luis Pedro Coelho)

Holger Dinkel

Version Control with Git

Version control is:

1. A record of what you & your collaborators have done,

Version Control with Git

Version control is:

1. A record of what you & your collaborators have done,
2. A way to see what's changed

Version Control with Git

Version control is:

1. A record of what you & your collaborators have done,
2. A way to see what's changed
3. A way to get back to different versions

What It Is

- A way to store and synchronize code (or any files).

What It Is

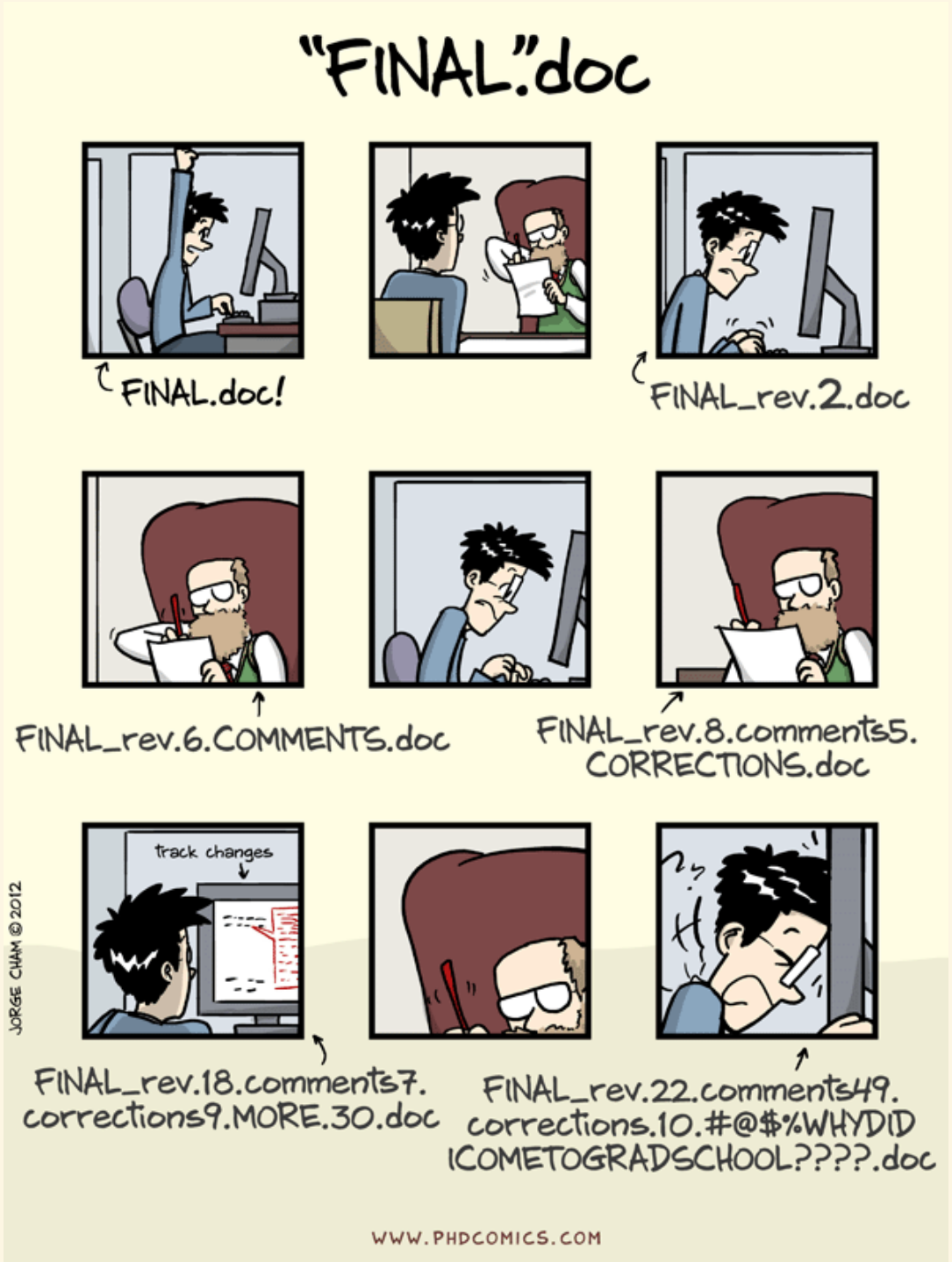
- A way to store and synchronize code (or any files).
- Documents collections of changes *upon your request*.

What It Is

- A way to store and synchronize code (or any files).
- Documents collections of changes *upon your request*.
- Keeps your work safe and up-to-date *across machines*!

A common approach. . .

- just add something to the name of your file
  - date
  - comment
  - status change
  - . . .


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Do's & Don'ts

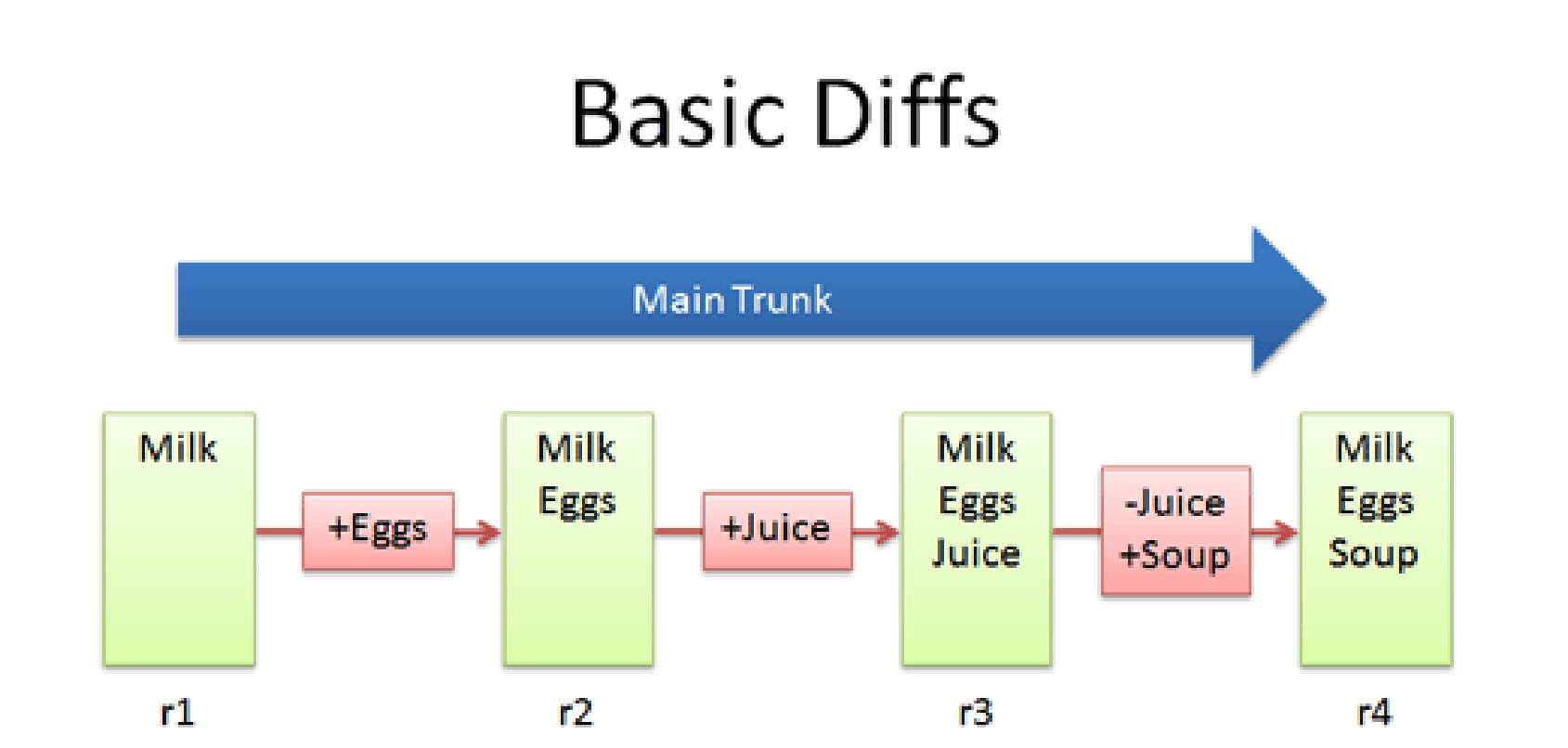
Do use git for

- textfiles
- documents
- configuration files

Do *NOT* use git for:

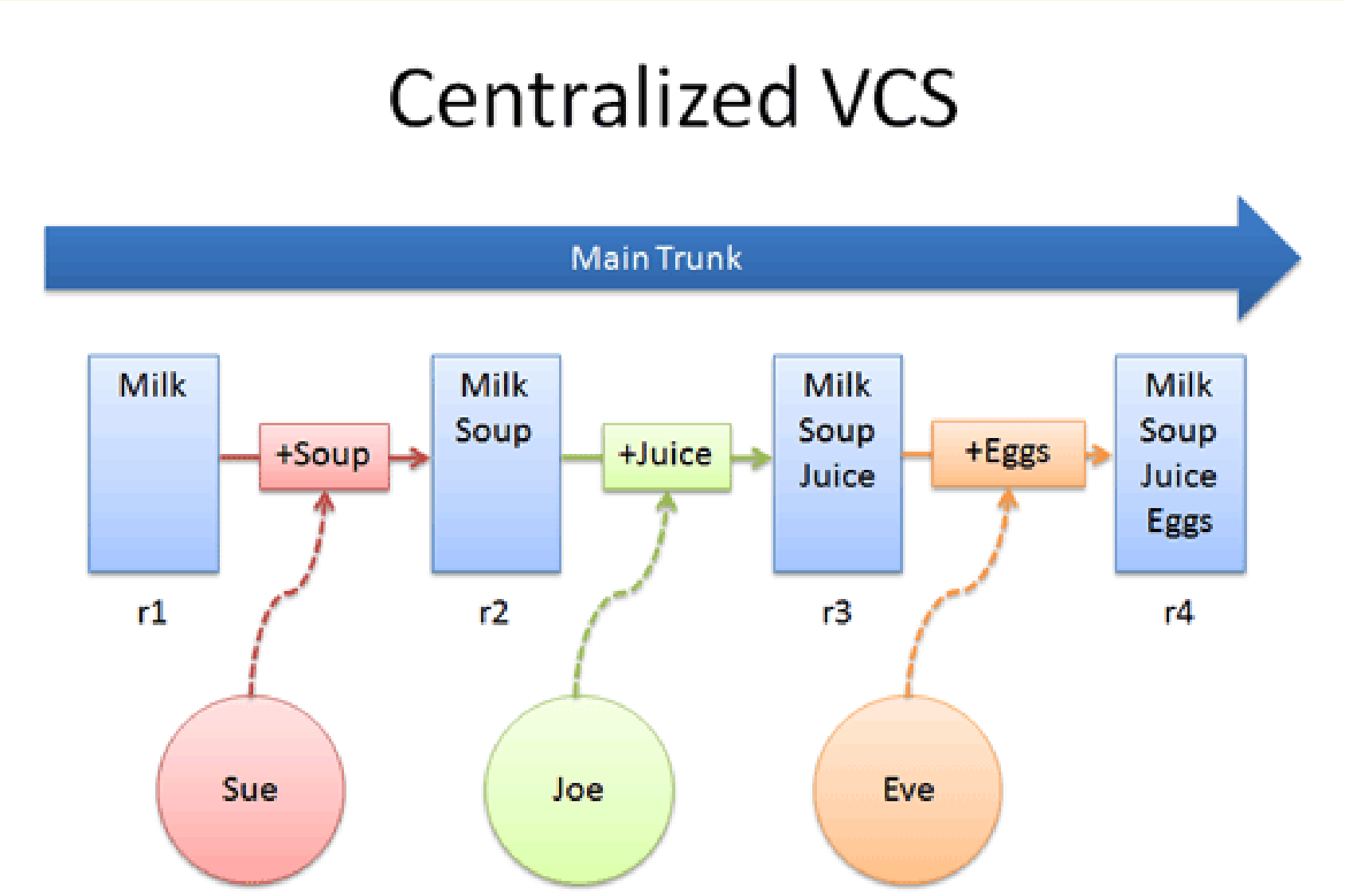
- passwords
- large files
- heavily changing files

Basic Version Control



(from betterexplained.com)

Working together



(from betterexplained.com)

Other problems we can solve

Imagine multiple copies of important code and data across machines:

- Which copy has “the fix”
- Sharing with yourself can be hard, but
- sharing with others is downright treacherous

What if there’s a conflict?

Perfectly reasonable (but actually harder)

Cloud storage

+

well-considered naming schemes

=

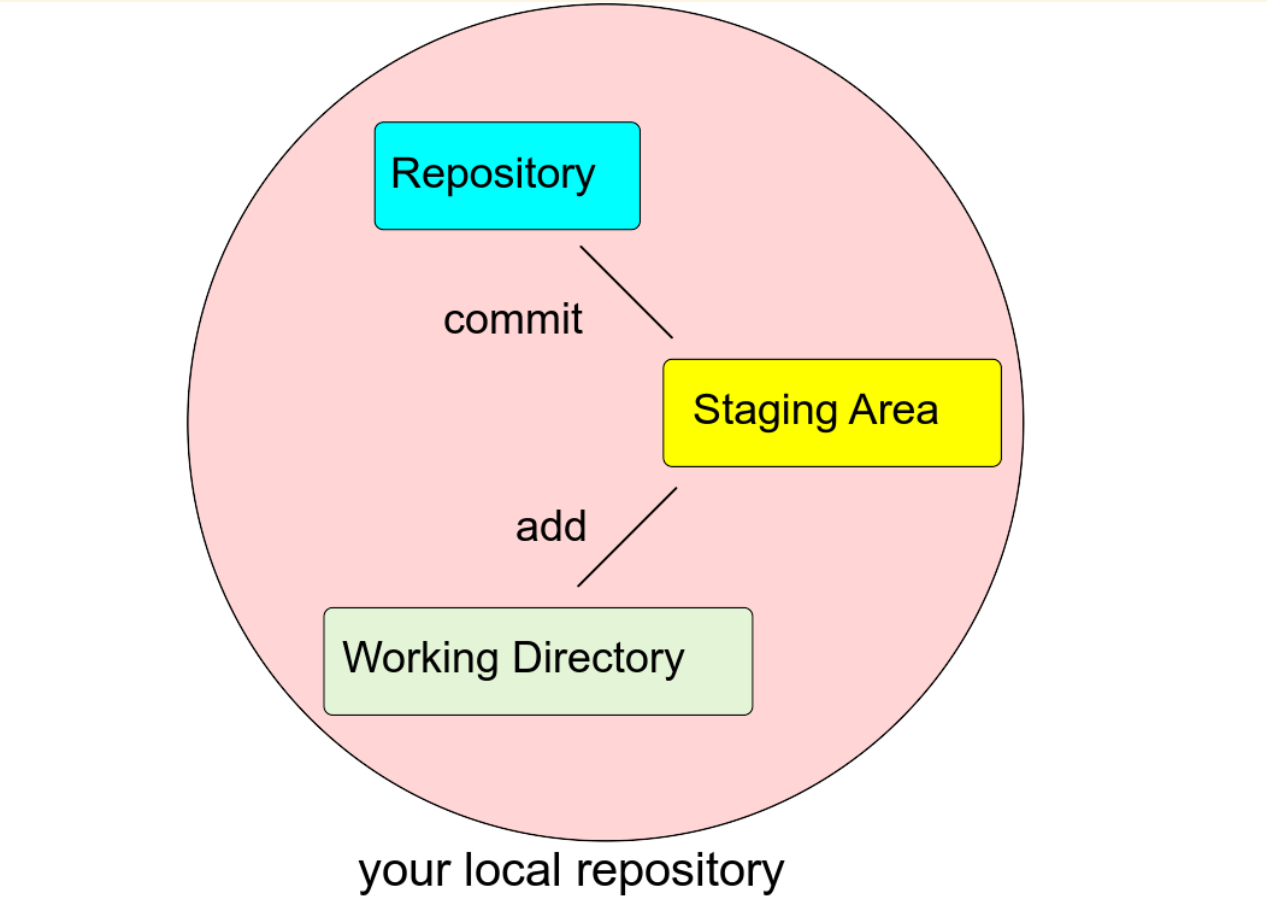
Maybe good enough?

How do you manage files now?
NB: you are still getting work done, right?

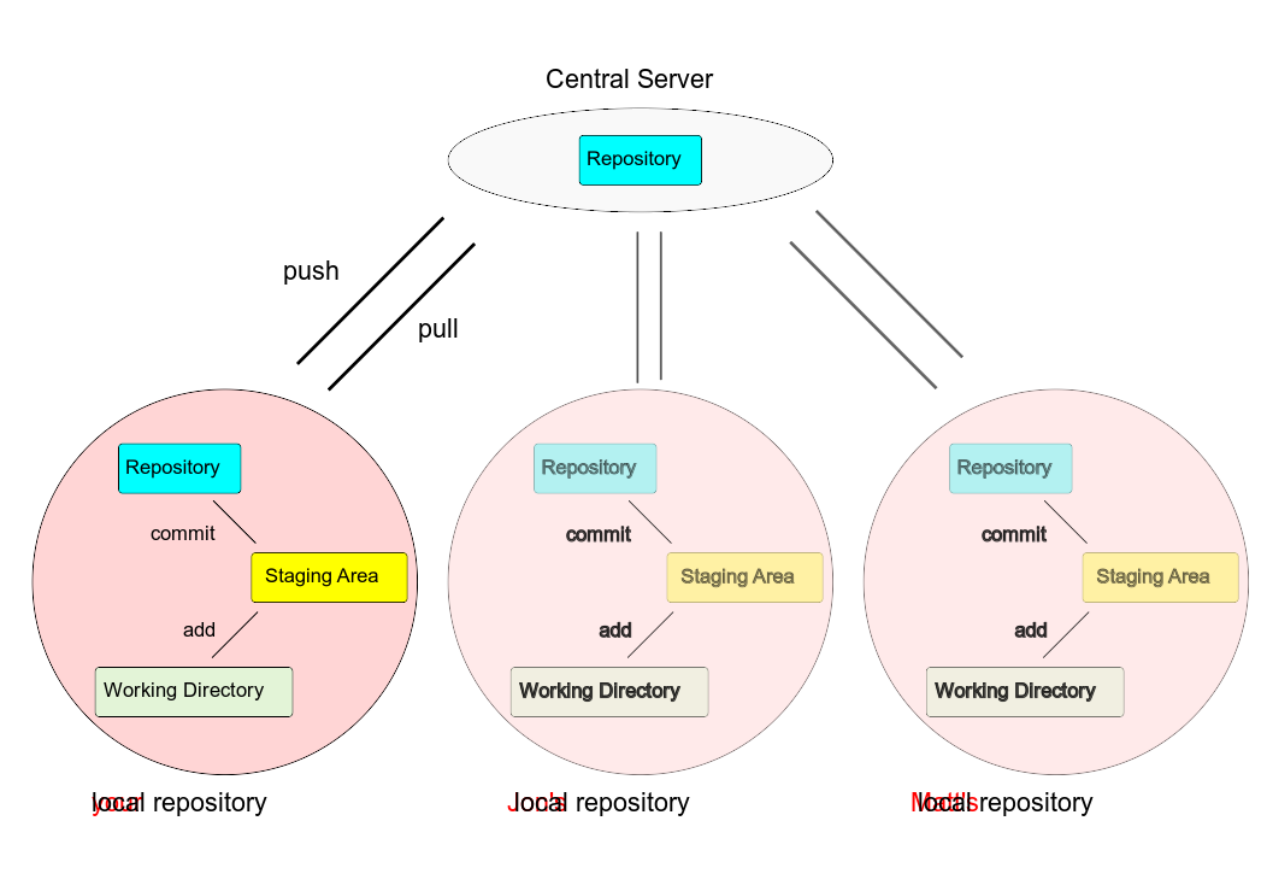
Git solves a lot of problems at once

- A record of what you and your collaborators have done
- A way to see what's changed
- A way to go "back in time" to previous versions

Two step process



Git repositories



Informative commit messages

(from http://xkcd.com/1296)

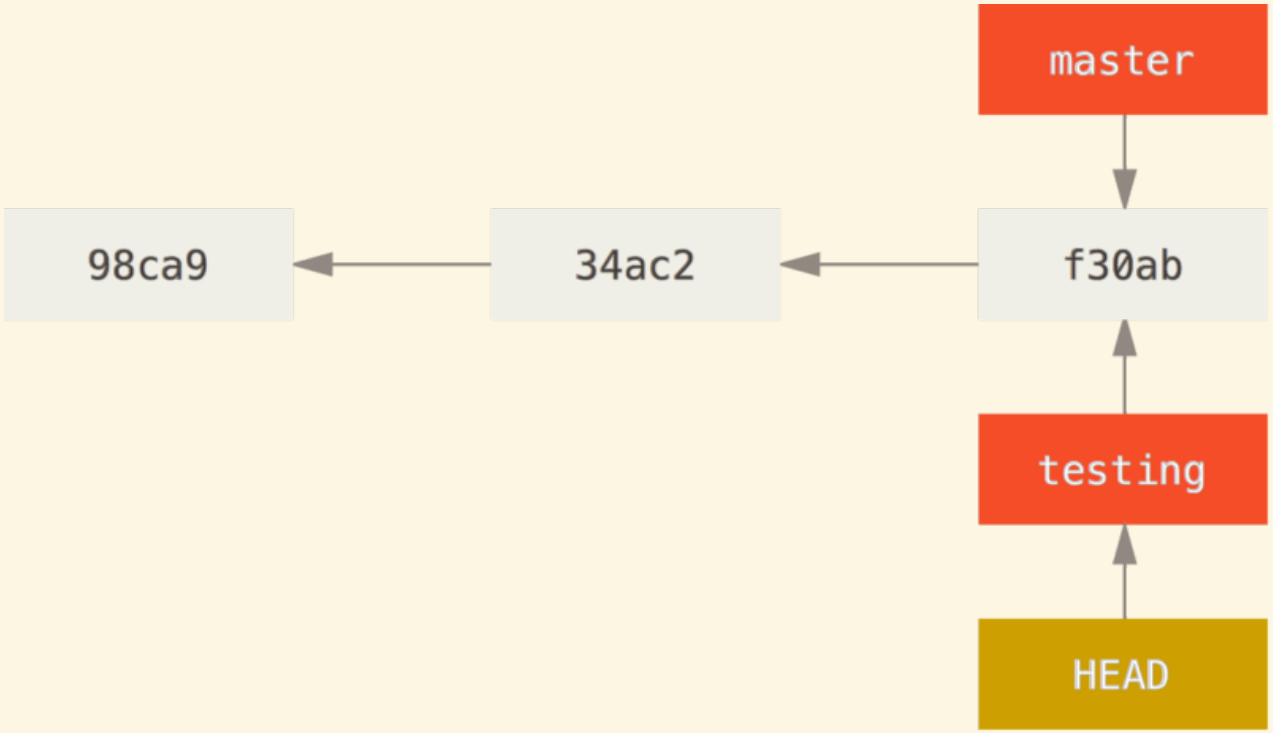
Explain what you're doing, or you won't know later.

Make a snapshot of your work!

Sharing history

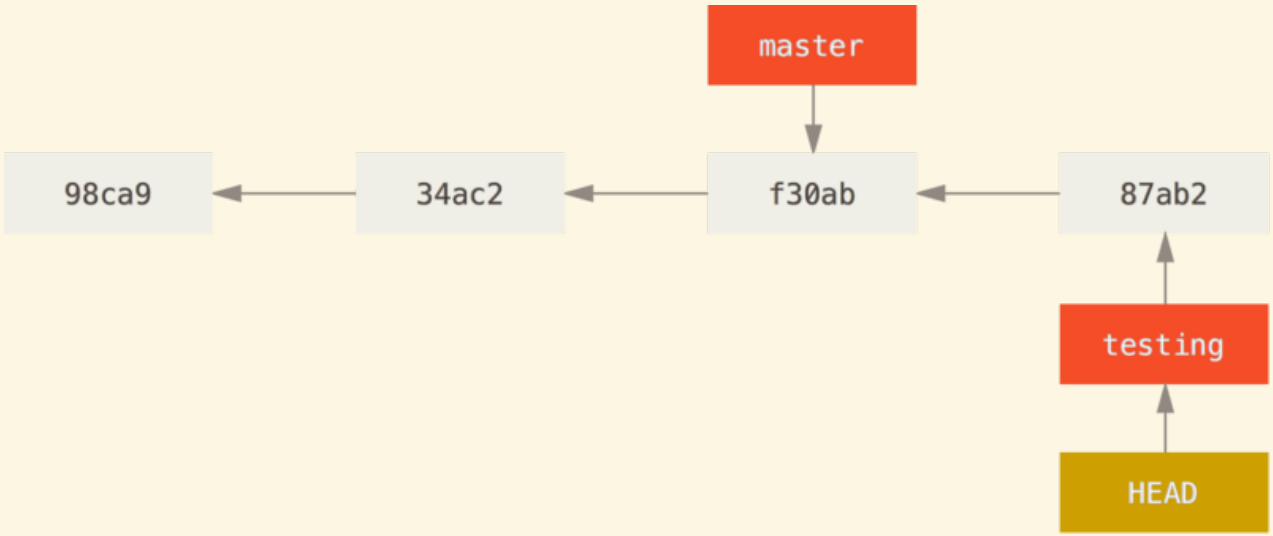
- The history is a permanent record of what happened (across copies of the repository)
- Put another way: the history is what we copy when we copy a repository

History



(from git-scm.com/book)

Branching



(from git-scm.com/book)

github / gitlab

- github used to be individual company, now owned by Microsoft
- other options (eg. gitlab) / pro's / con's
- repositories have size limitations
- huge database of source-code -> use the search function

Syllabus

1. Creating repositories
2. Adding / editing / deleting files
3. Adding and committing your work
4. Working with remote repositories
5. Making “clones”
6. Looking at history with diff and log
7. Pushing your work back to a remote
8. Pulling updates from a remote
9. Collaborating together