Version Control with Git

Methods & Tools for Software Engineering (MTSE)
Fall 2018

Prof. Alireza Sharifi

based on https://git-scm.com/book

What is Version (Revision) Control

A system for managing changes to documents, programs, web pages,…

Maintains a revision history of changes to the document

Maintains multiple versions of a document

Enables multiple users to collaborate on a common collection of documents

There are many revision control systems available

- rcs, cvs, subversion, mercurial
- git

Git Resources

From the command line

- `git help` to get a list of commands
- `git help <cmd>`
  - where `<cmd>` is a git command (e.g., `add`, `commit`, `fetch`, `merge`)

On-line book

- https://git-scm.com/book/en/v2

Tutorial

- https://git-scm.com/docs/gittutorial

Interactive Tutorial on GitHub

- https://try.github.io/

Git History

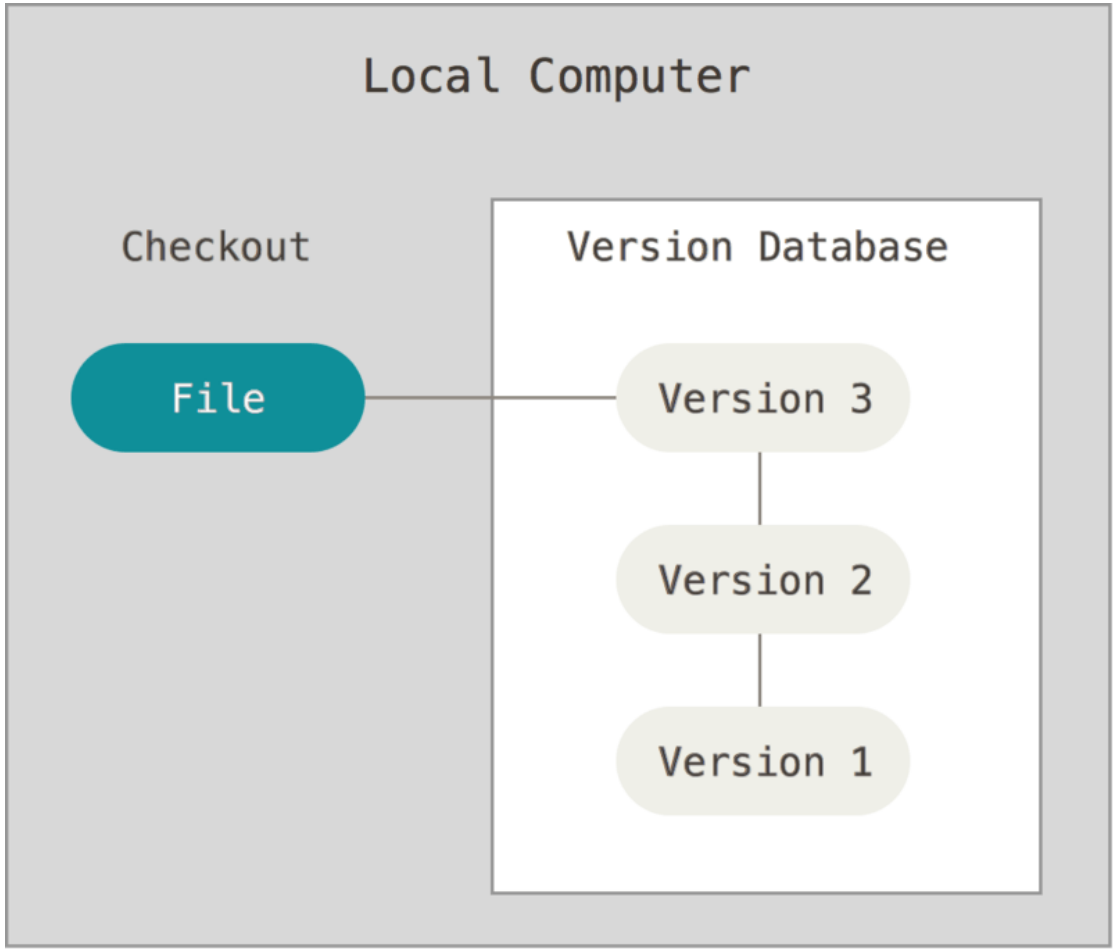
Developed by the Linux development community

- Linux Torvalds, 2005

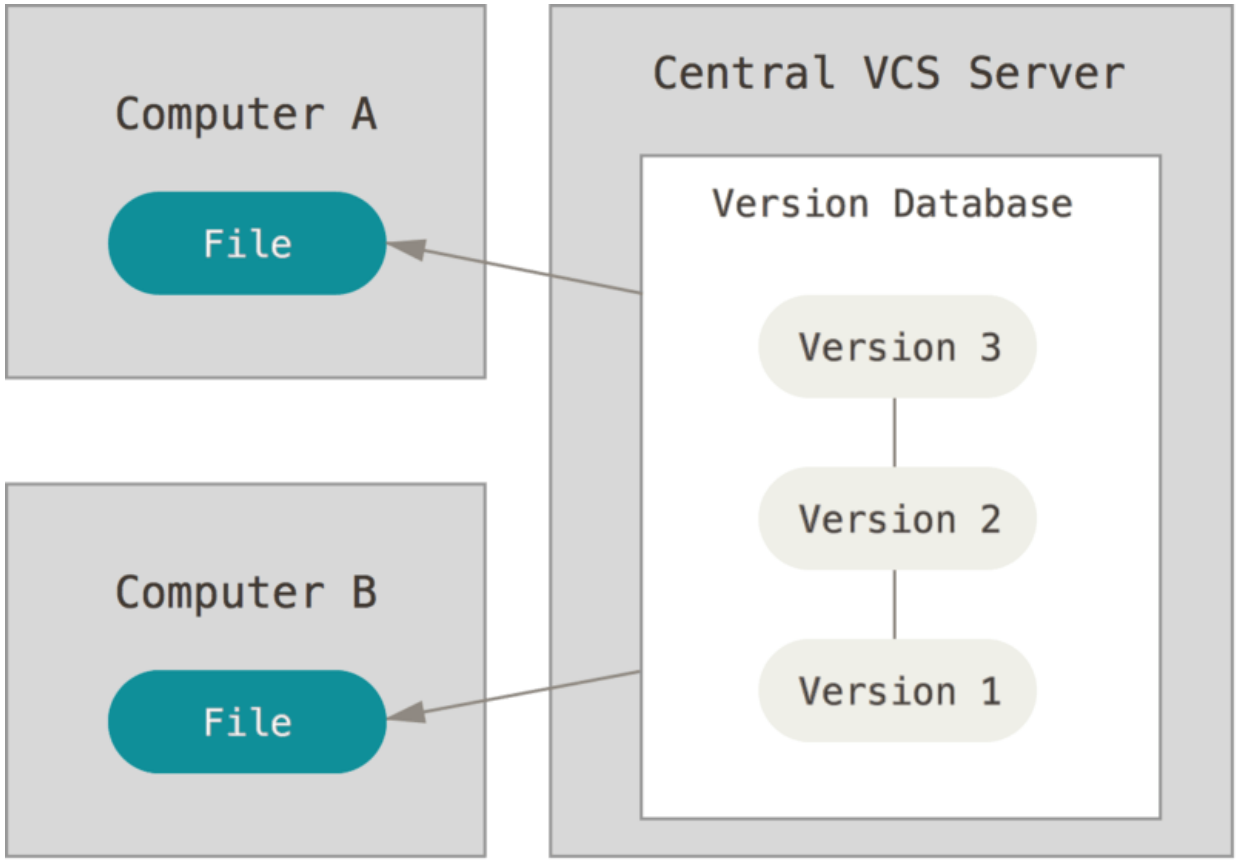
Initial goals

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

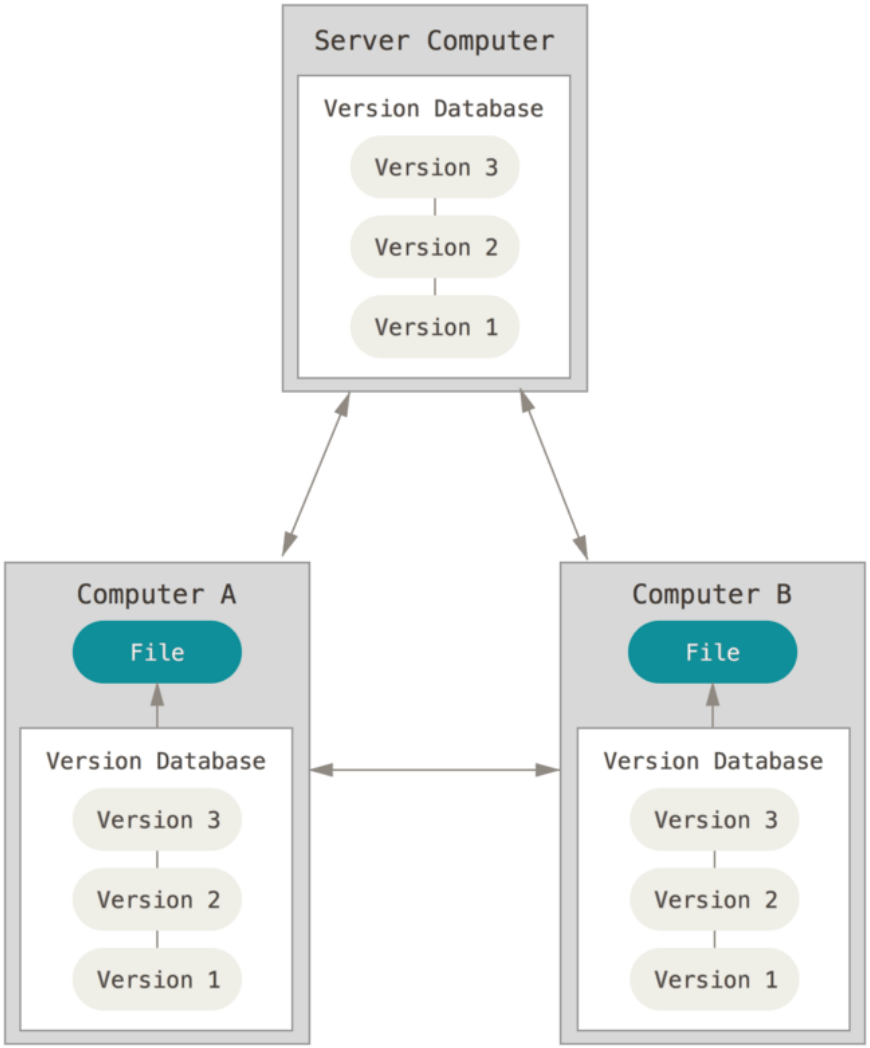
Local Version Control



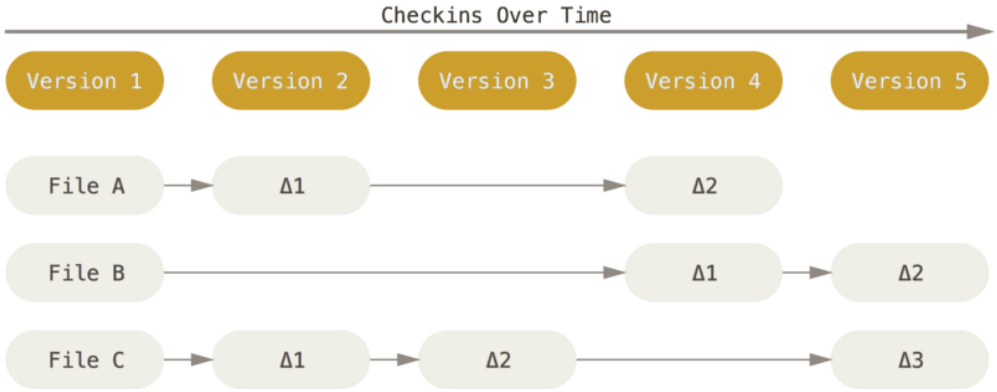
Centralized Version Control (subversion)



Distributed Version Control (git)

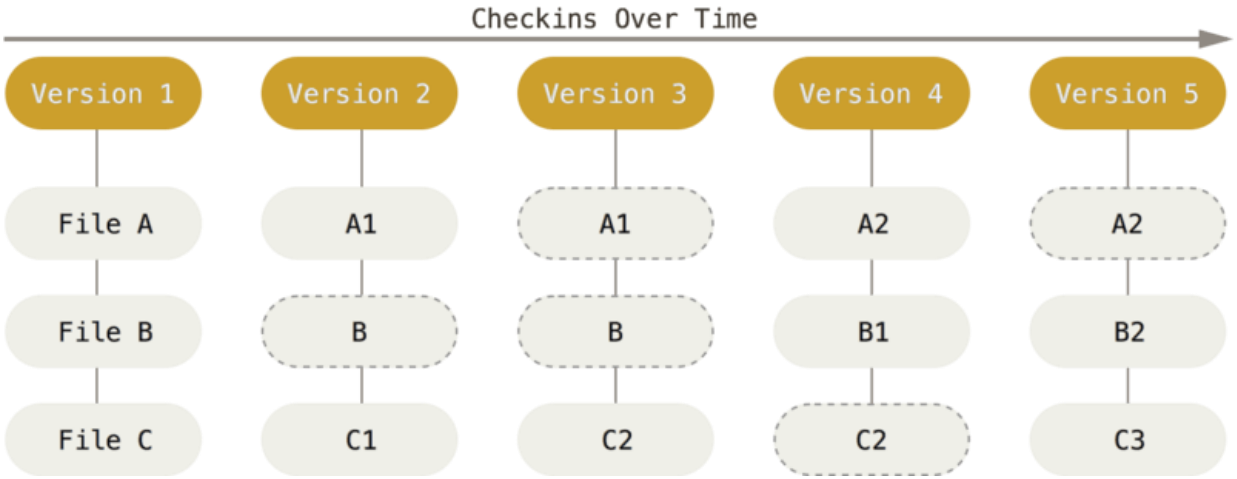


Snapshots, Not Differences



differences

snapshots



Common GIT workflow

init or **clone**

- create an empty repo or make a local copy of a remote repo

edit some files, create and modify content

add (or stage)

- mark changes to be combined into a commit
- a commit is a unit of change, a new version
- each commit has a globally unique name (i.e., `029389678201859fd6838c8b6c059edd0f17efcf`)

commit

- create a commit based on identified changes

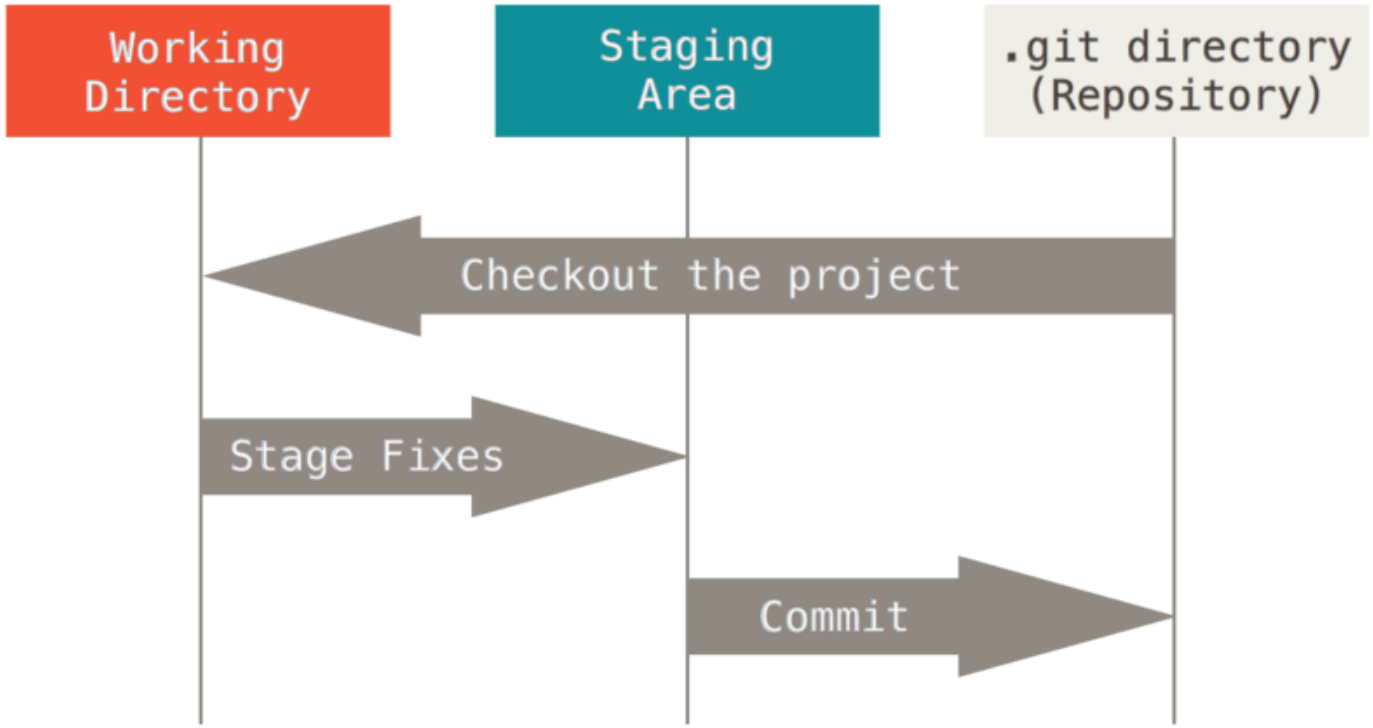
push

- propagate changes to remote repo

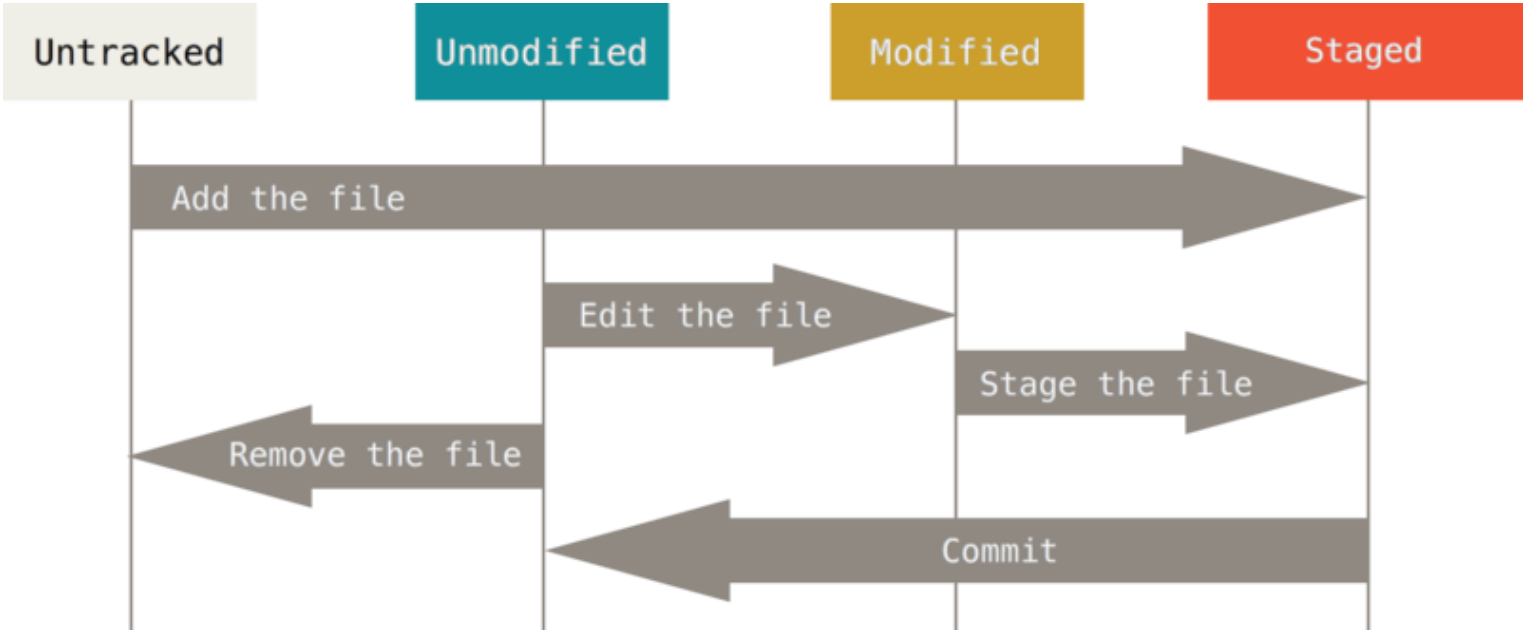
fetch or **pull**

- download changes from report repo

The Three States



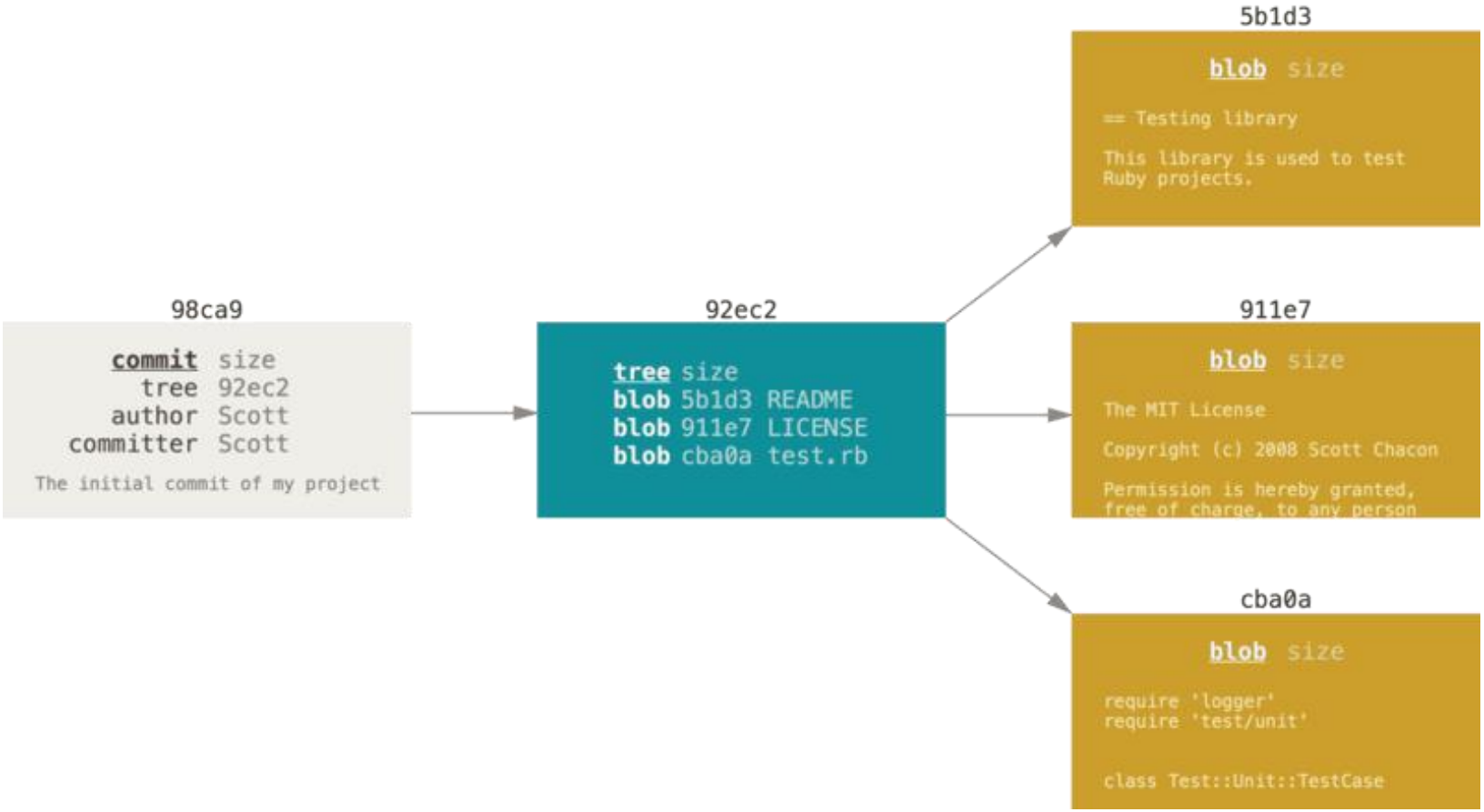
File Life Cycle



https://try.github.com

TO THE CONSOLE!

Git Internals: Blobs, trees, and commits



SHA: Secure Hash Algorithm

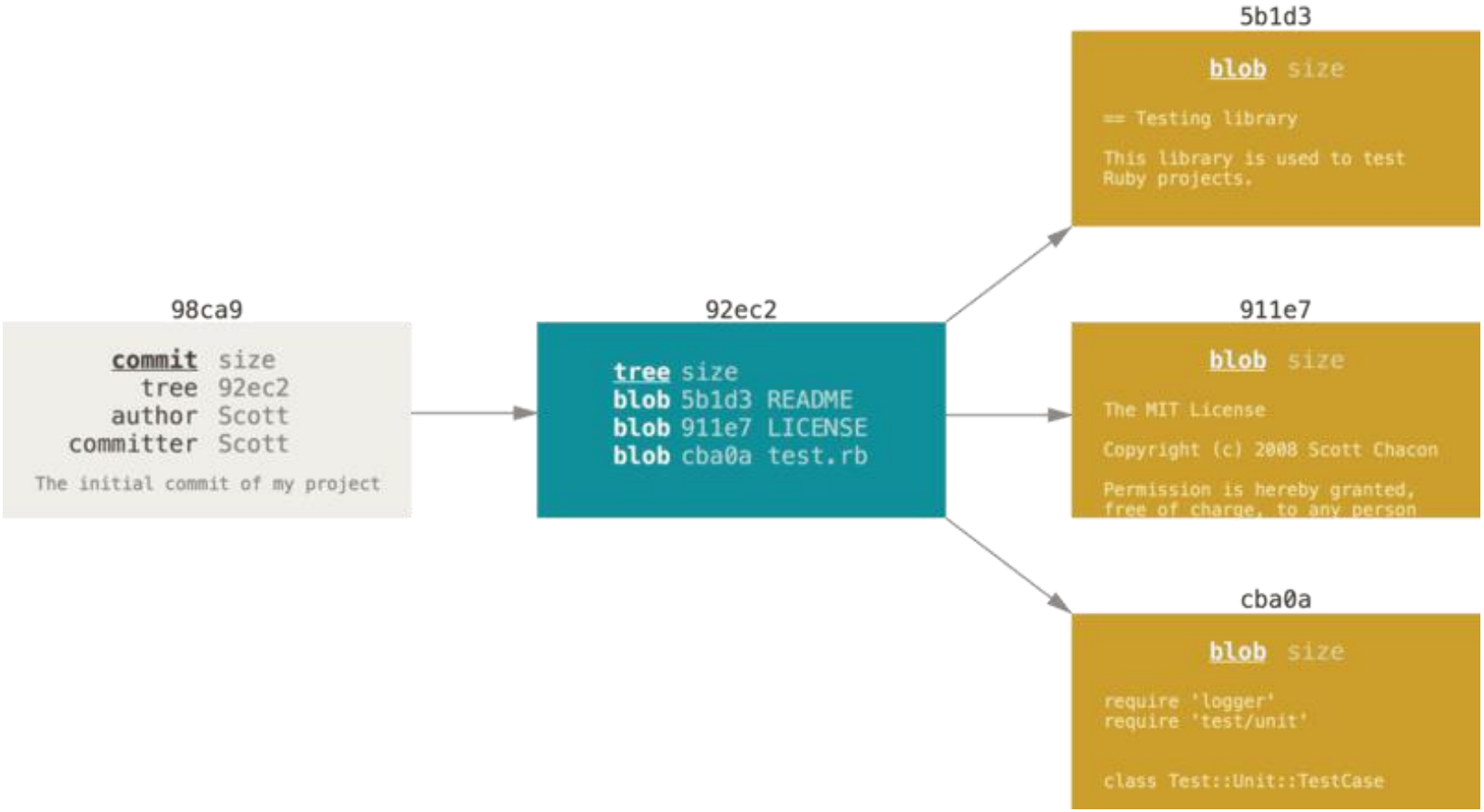
SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST.[3] SHA-1 produces a 160-bit (20-byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long. [Wikipedia]

For any message (sequence of characters) computes a 40 digit hex number (a digest) such that the probability that two different messages are assigned the same digest (collision) is very low.

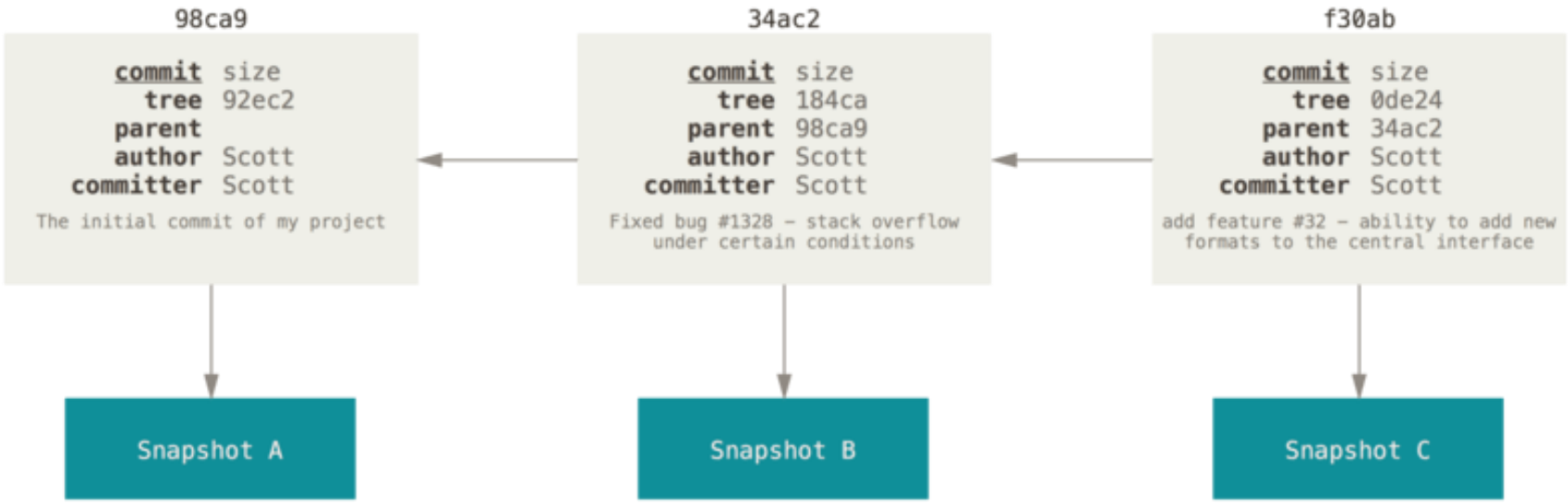
Git assigns every piece of content a unique name using its SHA-1 digest. In practice, the first 8 digest are sufficient for uniqueness.

You can use `sha1sum` to compute a digest on command line

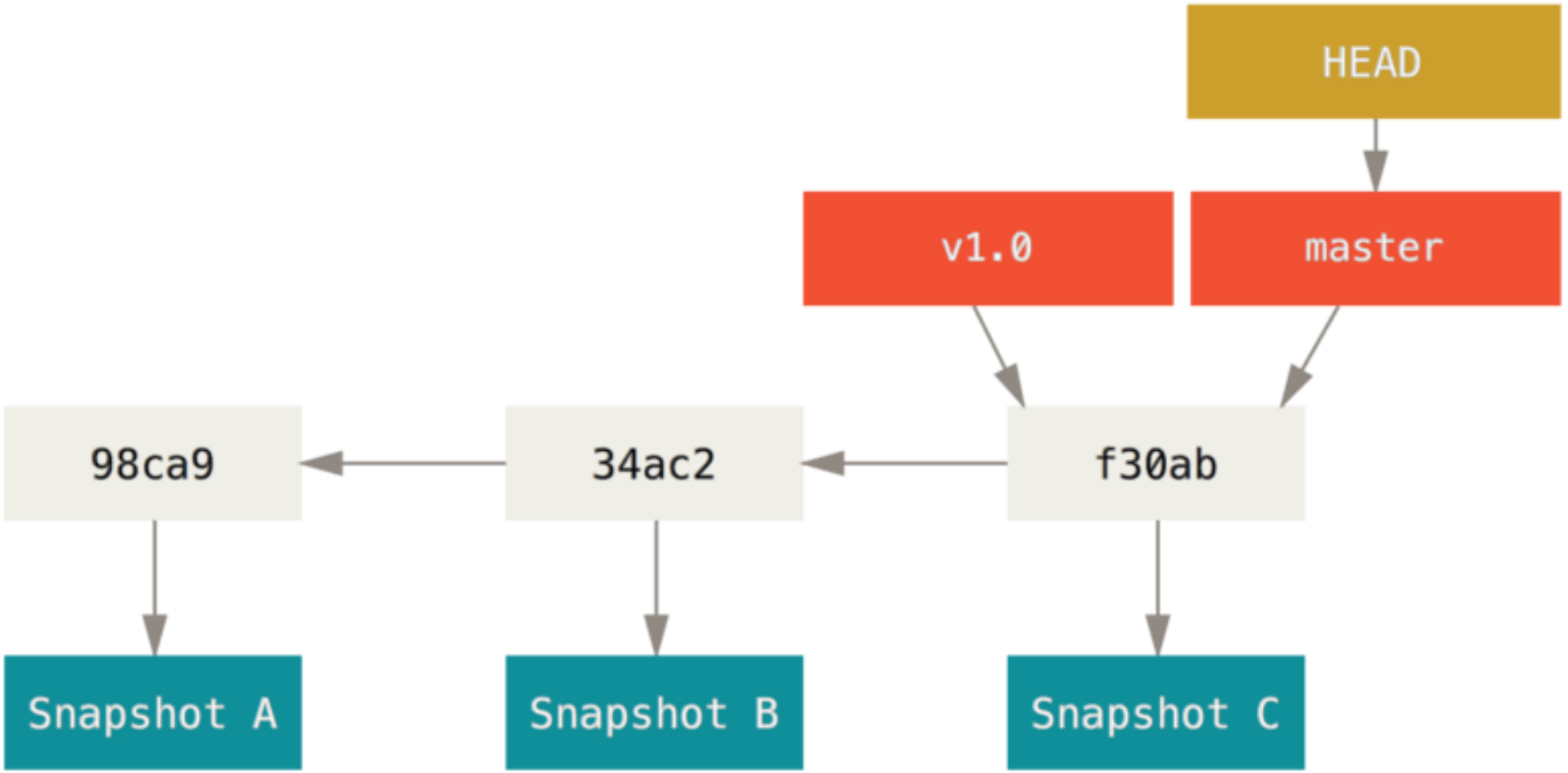
Git Internals: Blobs, trees, and commits



Commit history

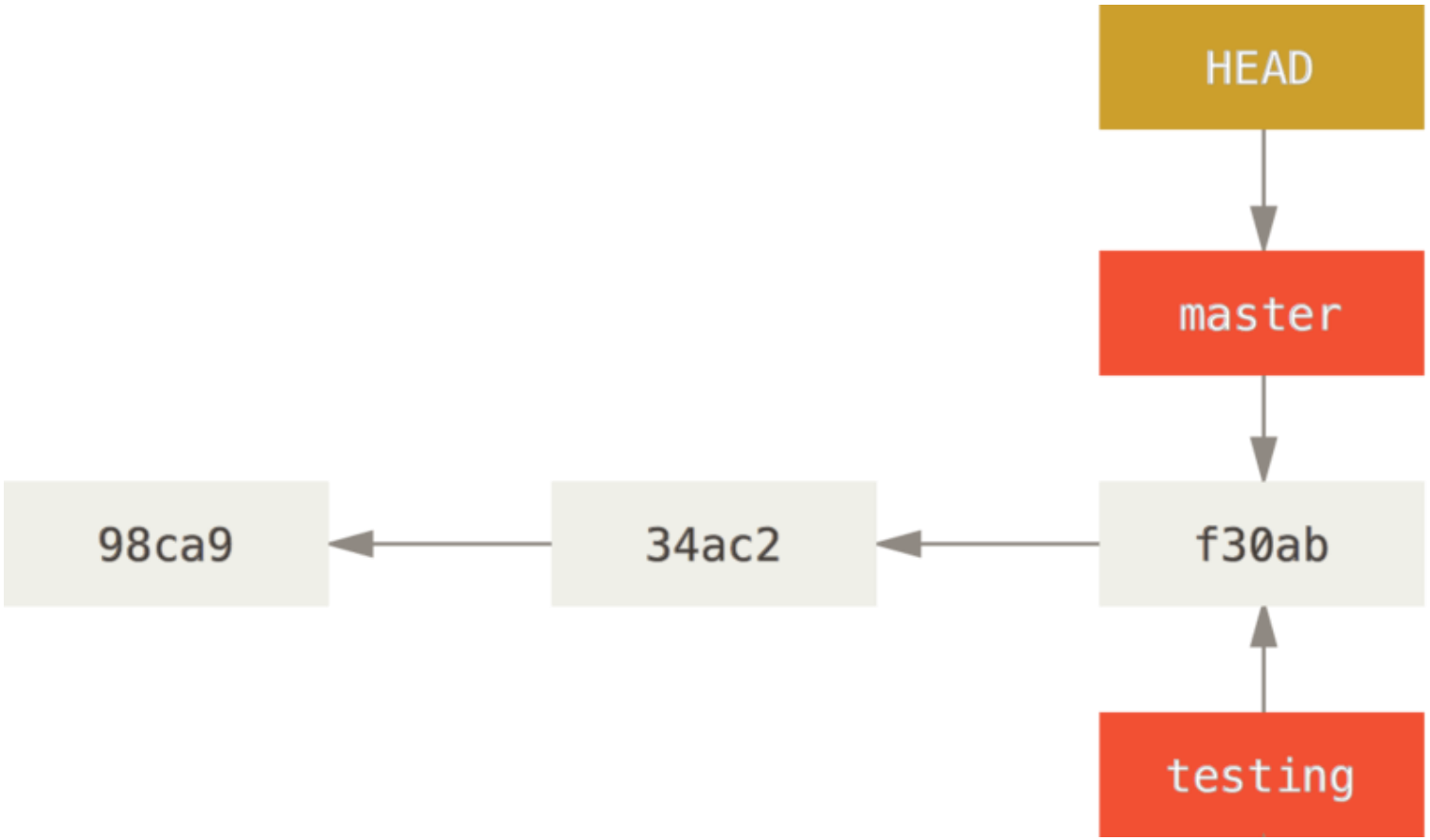


Commit history and branches



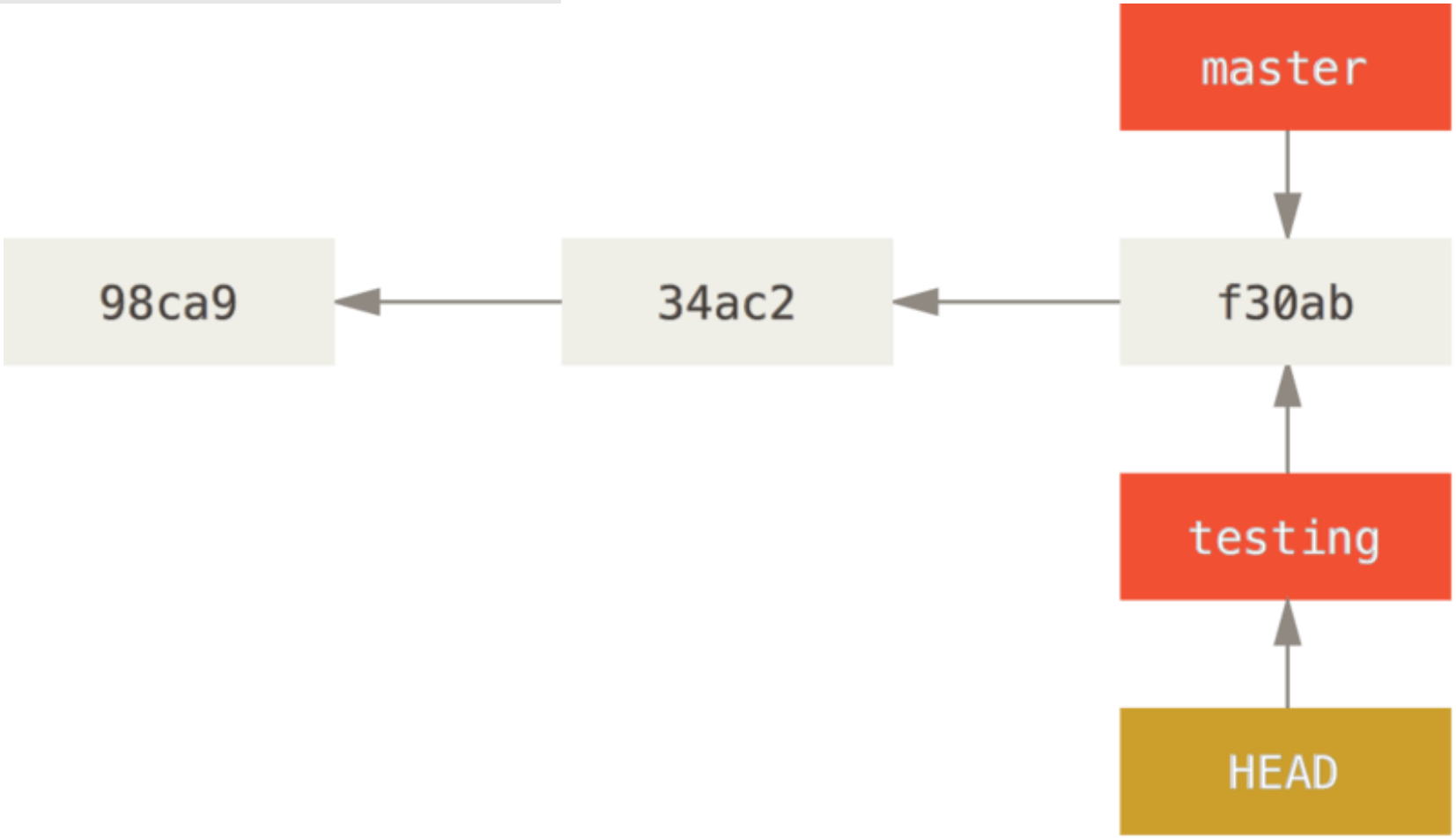
Creating a branch

```
$ git branch testing
```



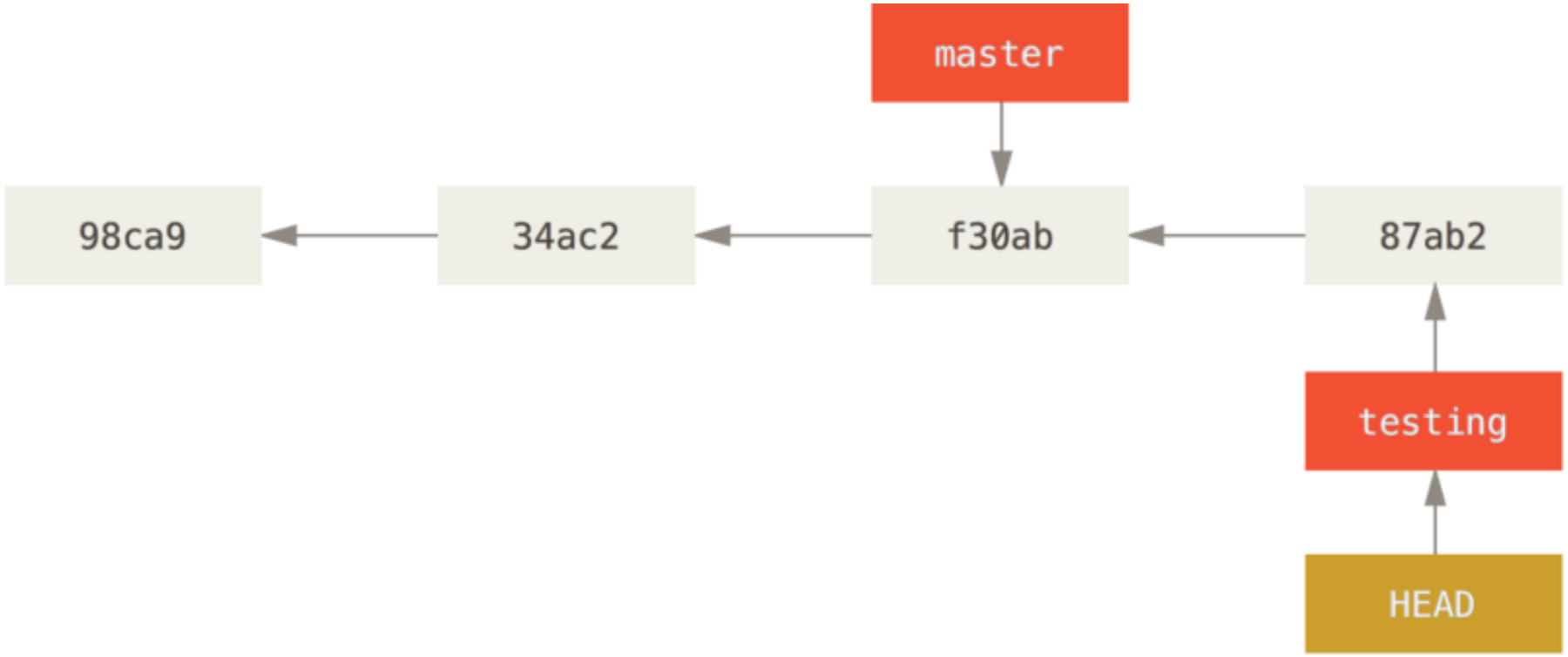
Switching a branch

```
$ git checkout testing
```



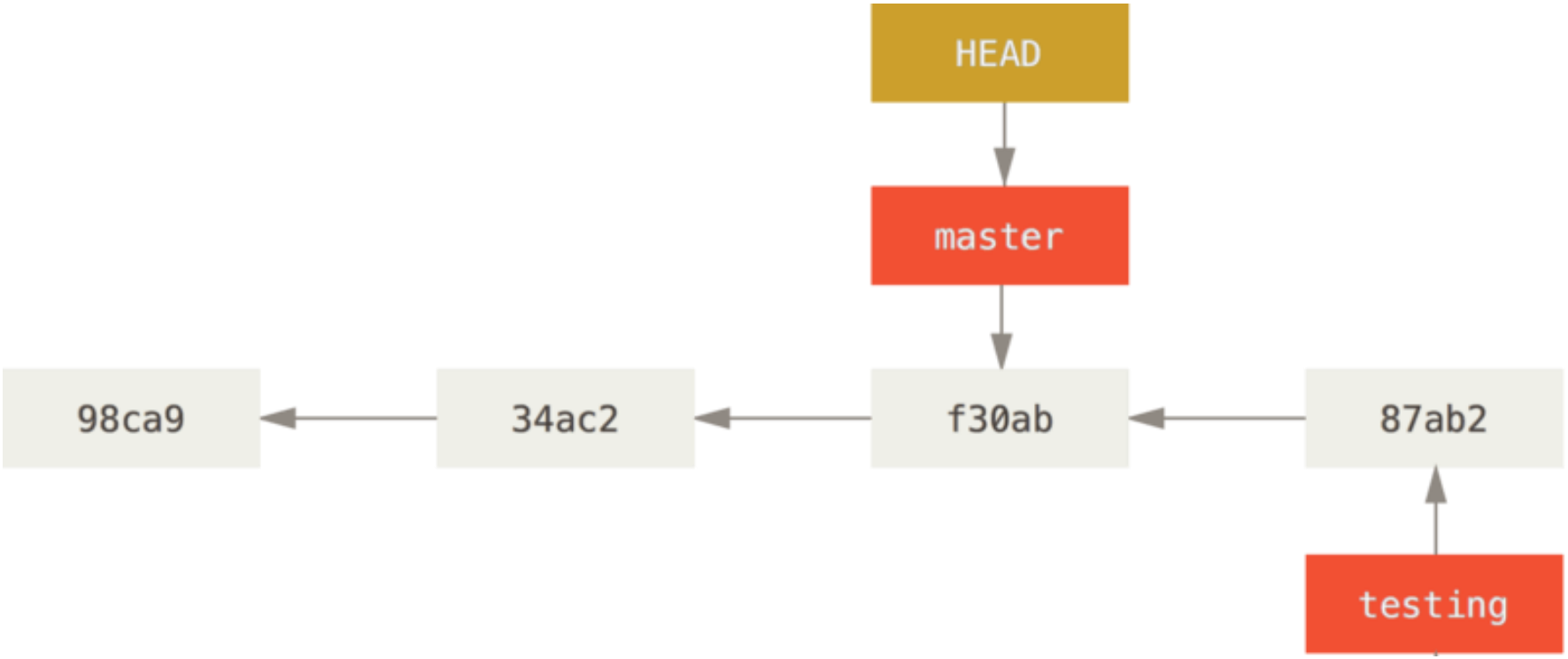
Change a file on testing

```
$ <make a change to foo.txt>
$ git commit –a –m ‘a change’
```



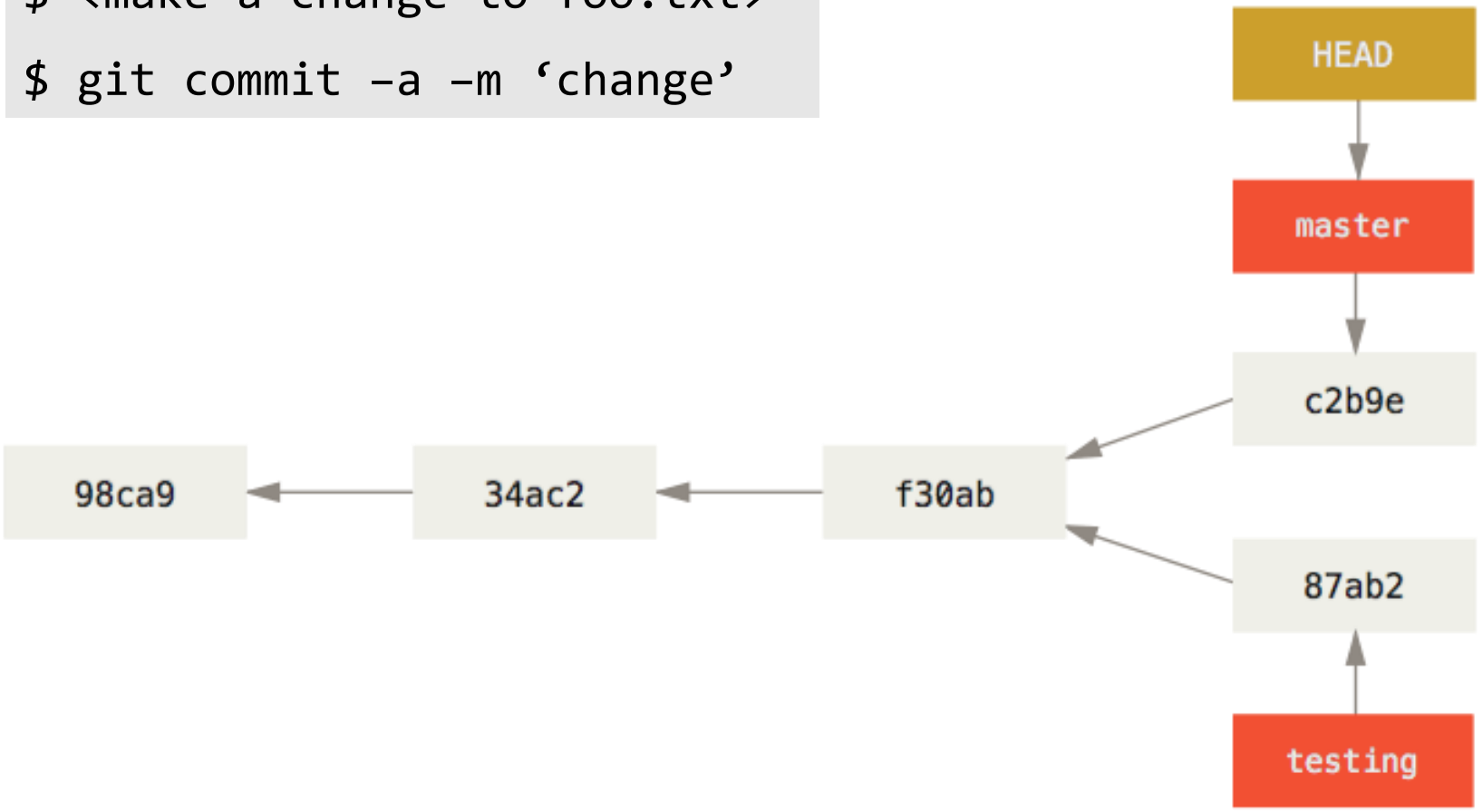
Switch back to master

```
$ git checkout master
```

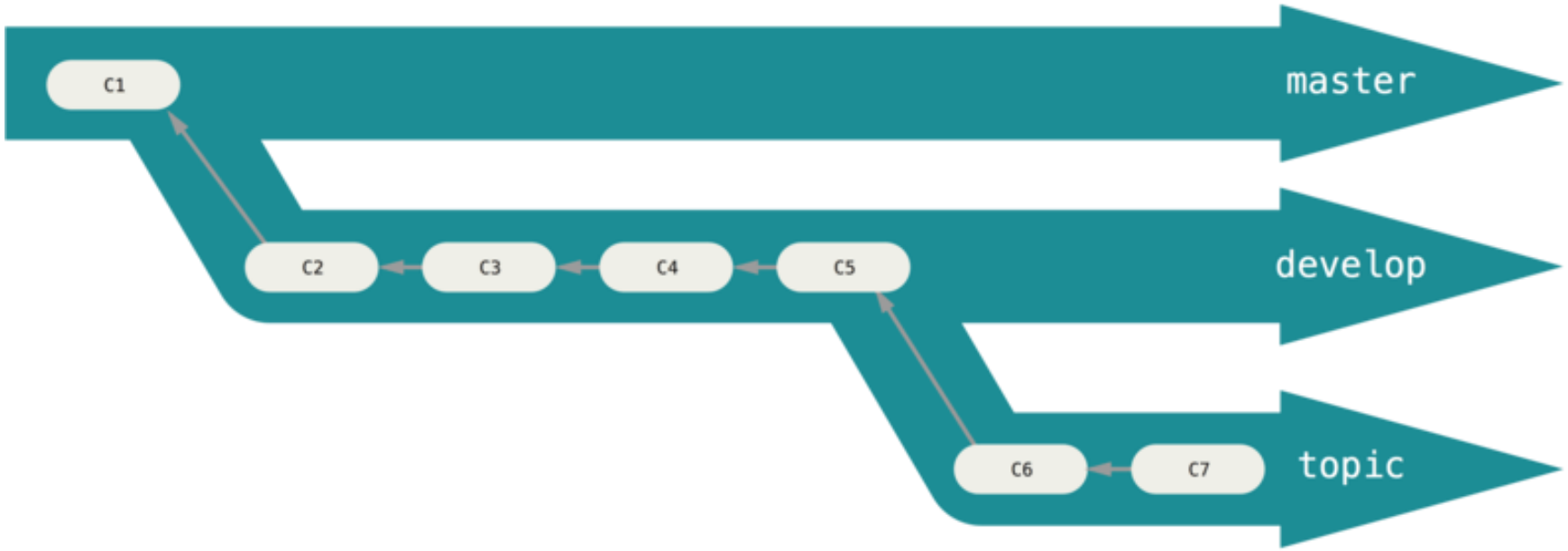


Change a file on master

```
$ <make a change to foo.txt>
$ git commit –a –m ‘change’
```



Master, develop, topic



master – stable version

- only stable, well-tested commits

develop – development version

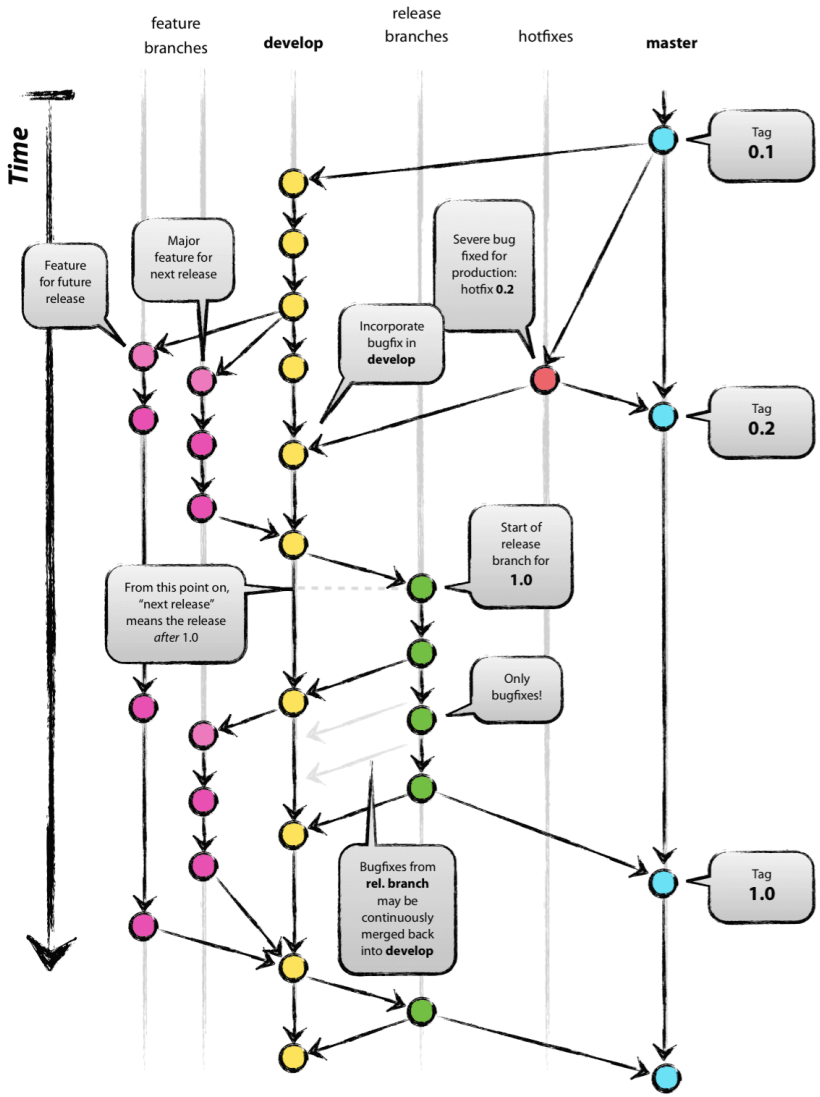
- next version to replace (merge into) master

topic – experimental bleeding edge

- test stuff out before merging into develop

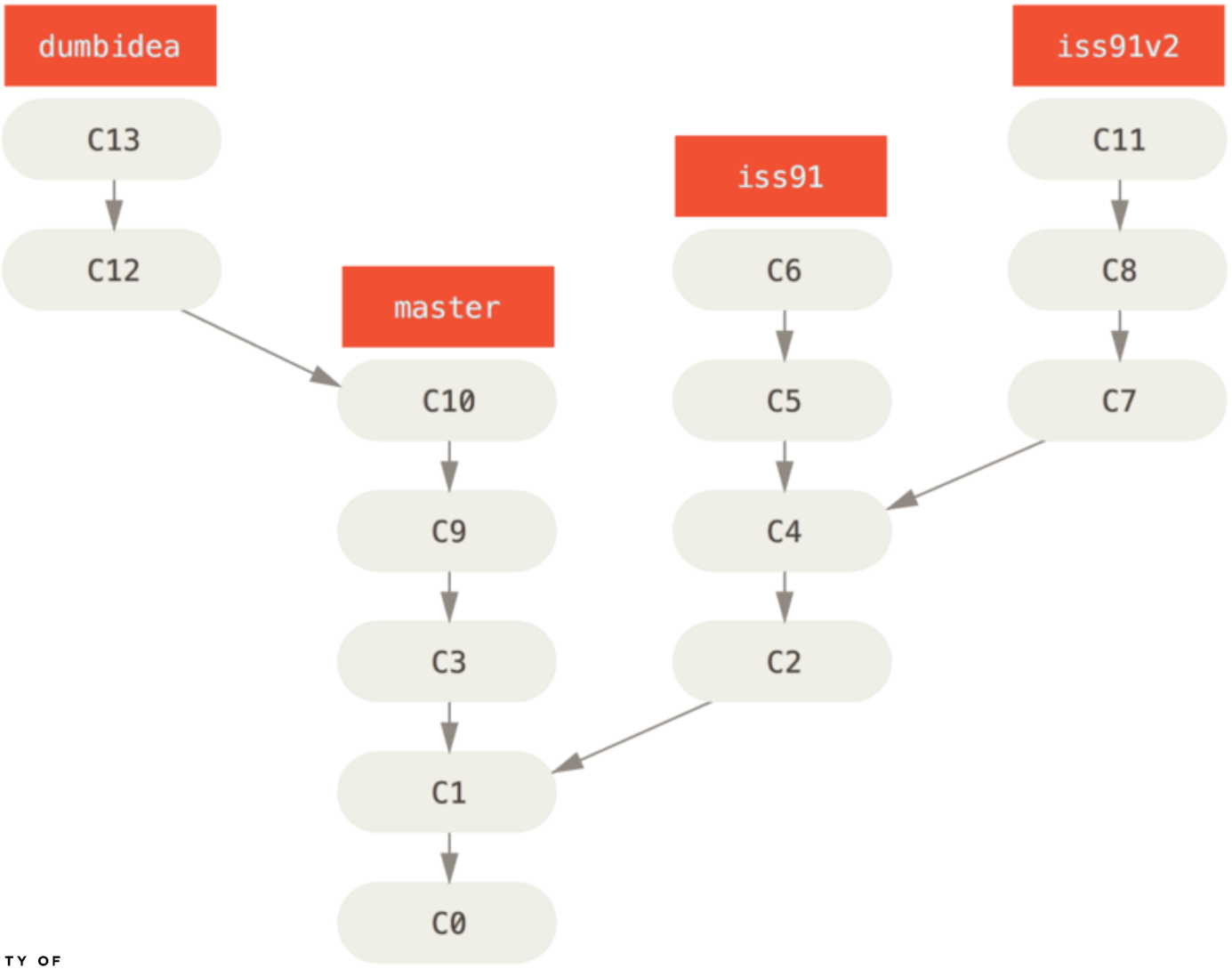
http://nvie.com/posts/a-successful-git-branching-model/

A successful Git branching model



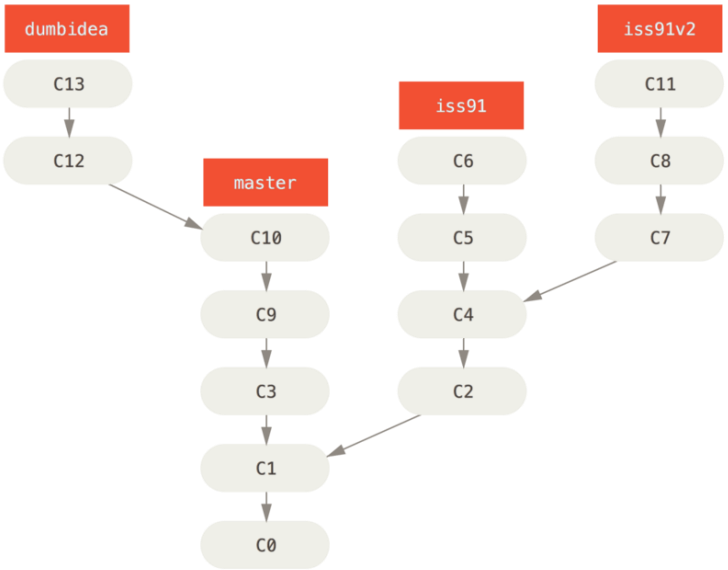
http://nvie.com/posts/a-successful-git-branching-model/

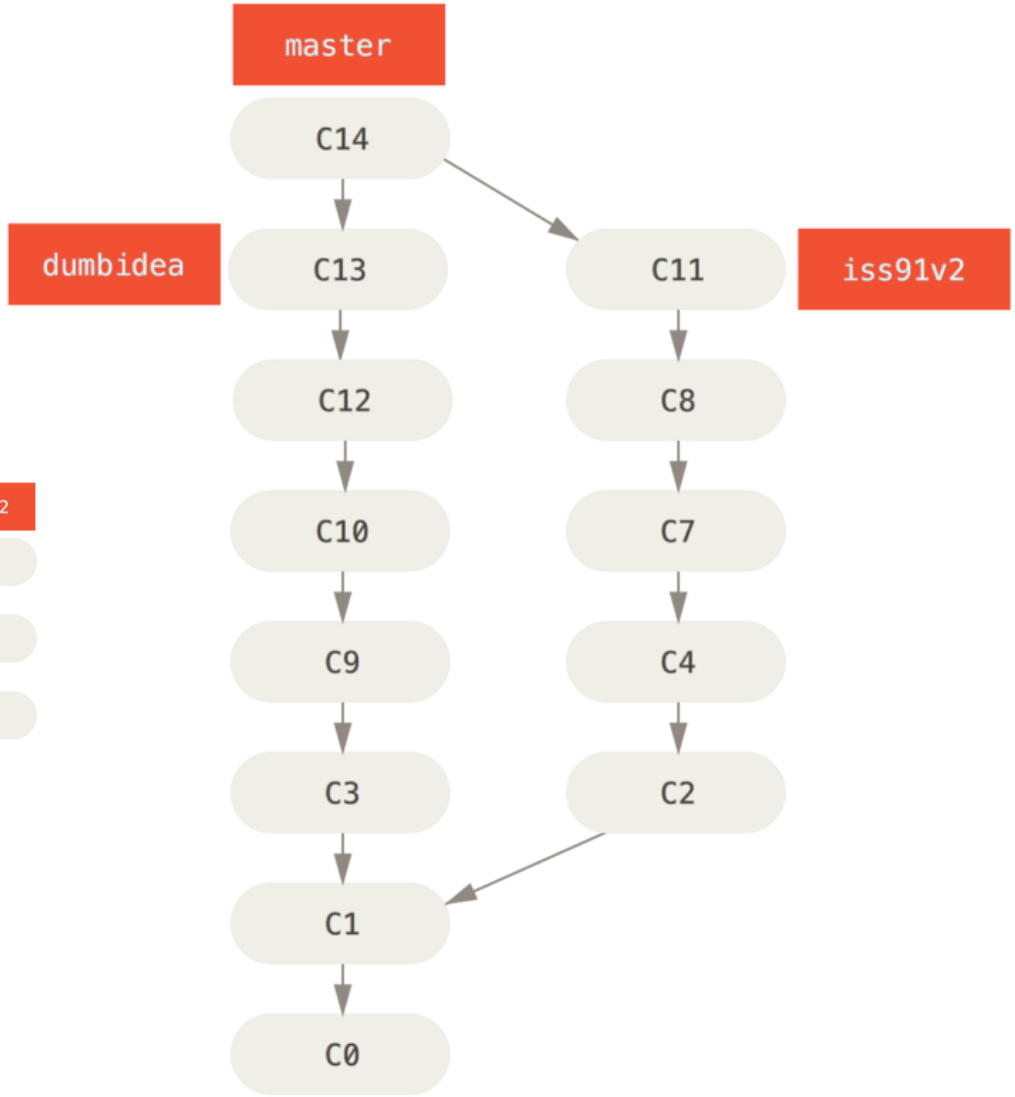
Example: many issues one solution



Example: merging things together

```
$ git checkout master
$ git merge dumbidea
$ git merge iss91v2
```





How to Write Git Commit Messages

https://xkcd.com/1296/

7 Rules of Great Git Commit Messages

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and why vs. how

https://chris.beams.io/posts/git-commit/

Crash course on UNIX

Methods & Tools for Software Engineering (MTSE)
Fall 2018

Prof. Alireza Sharifi

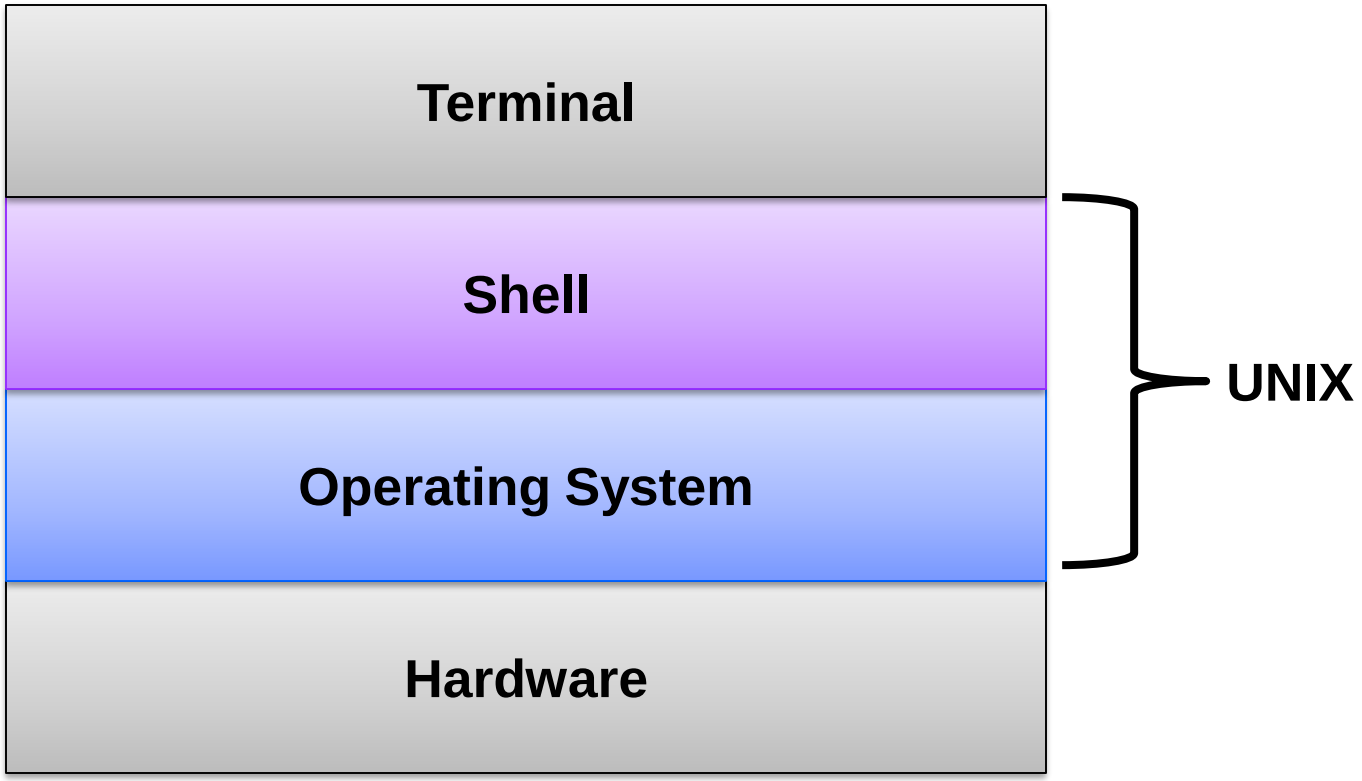
UNIX

Unix (/ˈjuː.nɪks/; trademarked as **UNIX**) is a family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix, development starting in the 1970s at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others.

- 1970s -- developed at Bell Labs research center
- 1980s -- popular on many platforms BUT too many forks / extensions (System V, AT&T, BSD, Xenix, …)
- 1990s – fragmented market, niche player
- 2000s – Linux is taking over, Apple is using Unix-based Darwin OS
- 2010s – Linux server market exceeds that of the rest of Unix market

https://en.wikipedia.org/wiki/Unix

Shell – Command Line Interface



ecelinux.uwaterloo.ca

All assignments have to work on `ecelinux`

Use Secure Shell (SSH) client to login from home

- built-in on Linux/Mac
- PuTTY on Windows (https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html)

First, connect to `ecelinux4.uwaterloo.ca` using your Waterloo username and password

Second, move to use `ssh` to hop to one of `ecelinux[1-3]` work machines

```
$ ssh -X ecelinux1
```

Third, once connected start `bash` shell (unless bash is your def shell)

```
$ bash -l
```

If in doubt about any questions, answer “Yes” ☺

Basic Shell Commands

Command	Description
`pwd`	display current working directory
`cd` *folder*	change working directory to folder
`ls`	list files in the current directory
`ls -la`	list files including hidden files and display lots of information
`ls` *folder*	list files in a given folder
`mkdir -p` *folder*	create a folder (and necessary sub-folders)
`rm -rf` *folder*	recursively delete a given folder
`touch` *filename*	create a blank file with a given name
`rm` *filename*	delete a file with a given name
`cat` *filename*	prints a content of a file onto standard output
`less` *filename*	display a content of a file
`man` *command*	display a manual page on command

there are many more commands and many online tutorials

Fun with shell and pipes

List a first/last few entries of a file

```
$ cat file | head -n 20
$ cat file | tail
```

Find all unique words in a file and their occurrences

```
$ cat file | sort | uniq -c | sort
```

Save output of a command to a file

```
$ ls > output.txt
```

Save output (stdout and stderr) and display it

```
$ ls 2>&1 | tee output.txt
```

Searching (grepping) for a string in the output

```
$ cat file | grep MYSTRING
```

Searching for a file by name

```
$ find [folder] -name '*filename*'
```

`man command` or `command -help` or google for help

Transferring files between ecelinux

Use Secure Copy (scp/sftp) to transfer files to `ecelinux4`

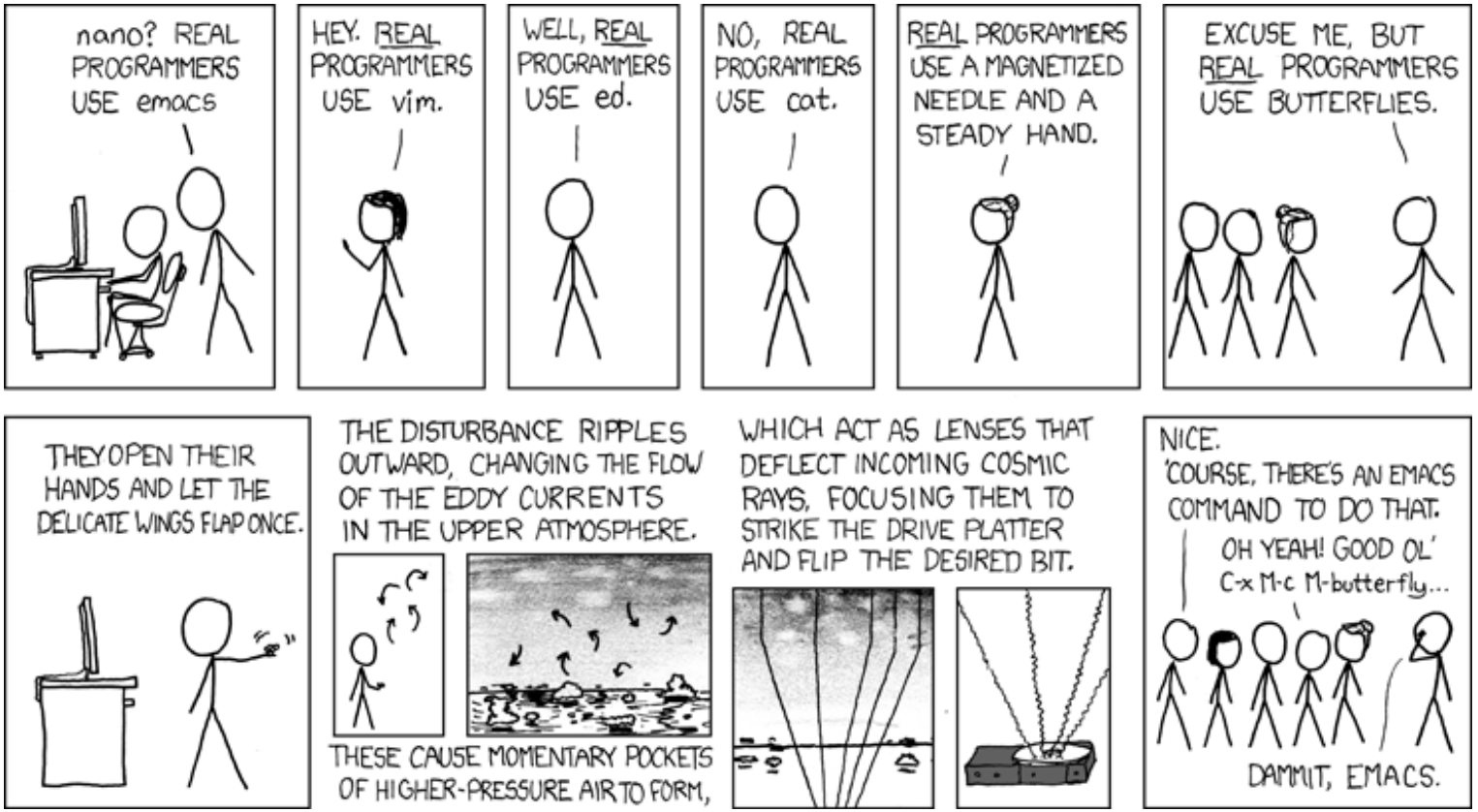
- they are immediately available on all ecelinux machines

Much better way – use Git!!!

- commit and push from your working machine to github
- fetch or pull from ecelinux

If you don’t have a local Linux environment, use a local virtual machine. Only use ecelinux for final testing and initial exploration

Which editor to use?



https://xkcd.com/378/

https://xkcd.com/1823/

So what editor to use?

Use your favorite editor or IDE on your machine

- atom, sublime, notepad++, visual studio, xcode, eclipse, etc.

Use a simple text editor on ecelinux (or another terminal)

- pico, nano

Use vi or vim (Vi Improved)

- there are gui versions available for virtually every platform