Mastering Git Basics

by Tom Preston-Werner

@mojombo

Before we start...

⚠

Forget everything you know
about version control

Installing Git

Mac

Git OSX Installer

Homebrew

MacPorts

Manually

Linux

Apt

Ports

Yum

Manually

Windows

msysgit

help.github.com

Initial Configuration

http://gist.github.com/340818

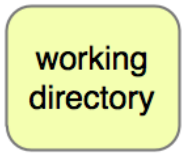
```
$ git config --global user.name "Tom Preston-Werner"
$ git config --global user.email "tom@mojombo.com"

$ git config --global color.ui true
```

Creating and Committing

Make a directory for your new project

```
$ cd path/to/repos
$ mkdir hello
$ cd hello
```

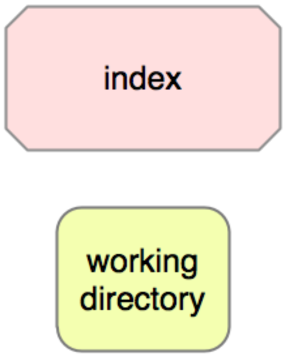
Working directory



git init

```
$ ls -al    # dir is empty
$ git init  # initialize git repo
$ ls -al    # new .git dir
```

Behold the index!



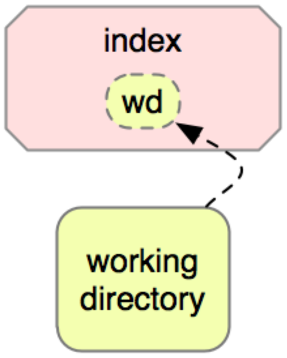
Write some code

```
$ vim hello.sh
$ vim goodbye.sh
```

git add

```
$ git add hello.sh   # add content to index
$ git add goodbye.sh # add content to index
```

Index now contains working dir content



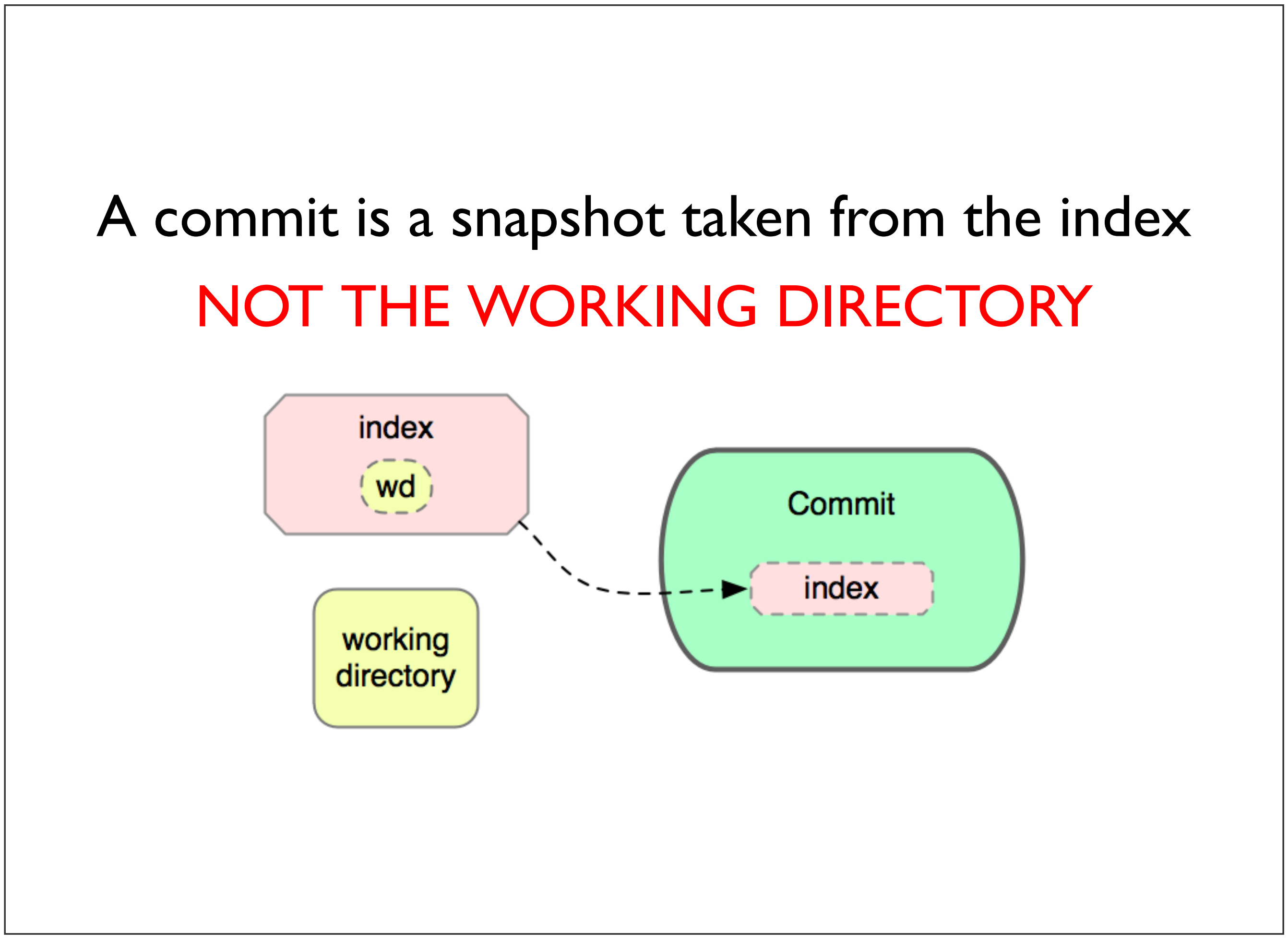
git status

Show the status of index and working dir

```
$ git status
```

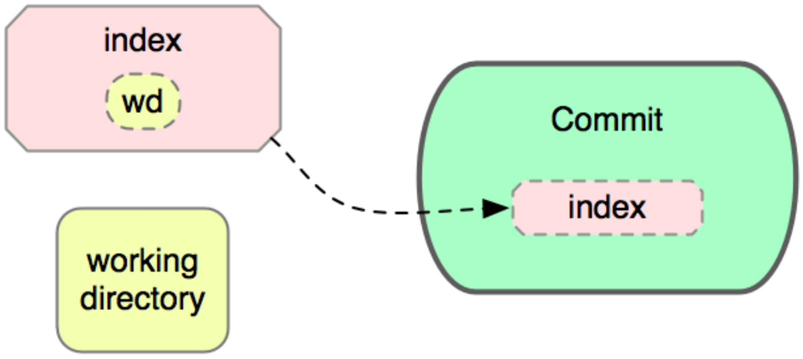
git commit

```
$ git commit  # make a commit
```



A commit is a snapshot taken from the index

NOT THE WORKING DIRECTORY



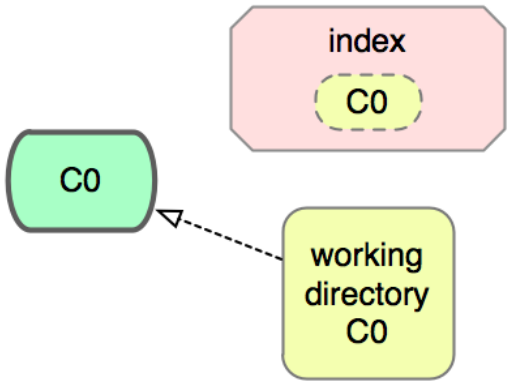
git log

Print a log of commits

```
$ git log
```

Recap

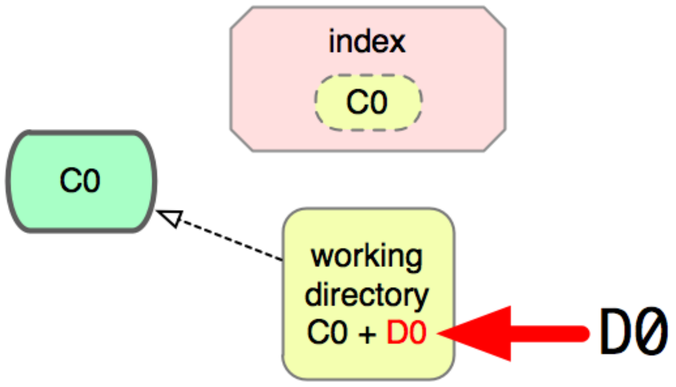
The current state of the repo



Make some ambitious changes

```
$ vim hello.sh  # modify the file
```

Working dir now contains D0: a delta from C0



Review the changed files

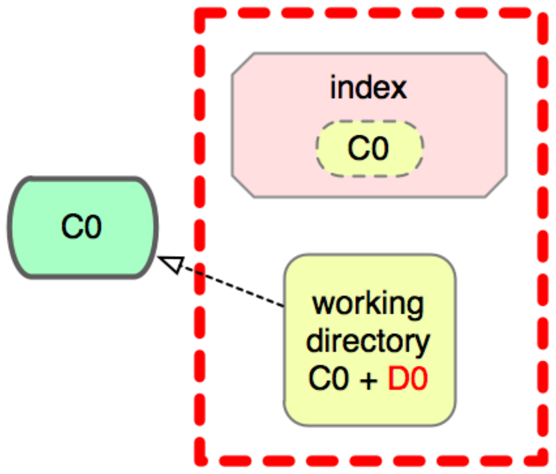
```
$ git status
```

git diff

Show diff between index and working dir

```
$ git diff
```

D0 = Diff(Index, WorkingDir)

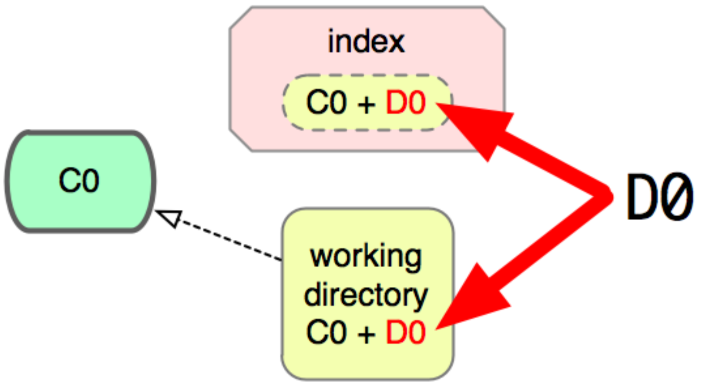


git add -p

Interactively add changed hunks

```
$ git add -p
```

D0 is now in working dir AND index



git diff now shows nothing!

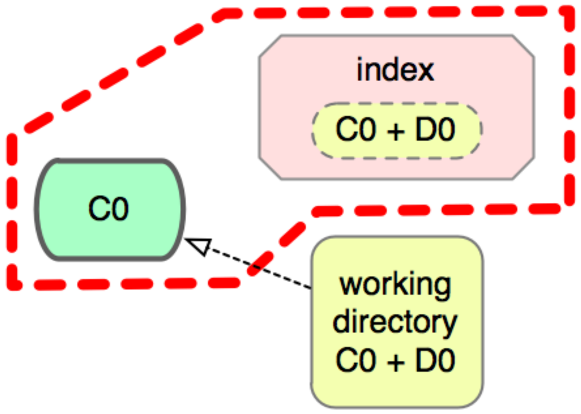
Where did it go?

git diff --staged

Show diff between commit and index

```
$ git diff --staged
```

D0 = Diff(Commit, Index)

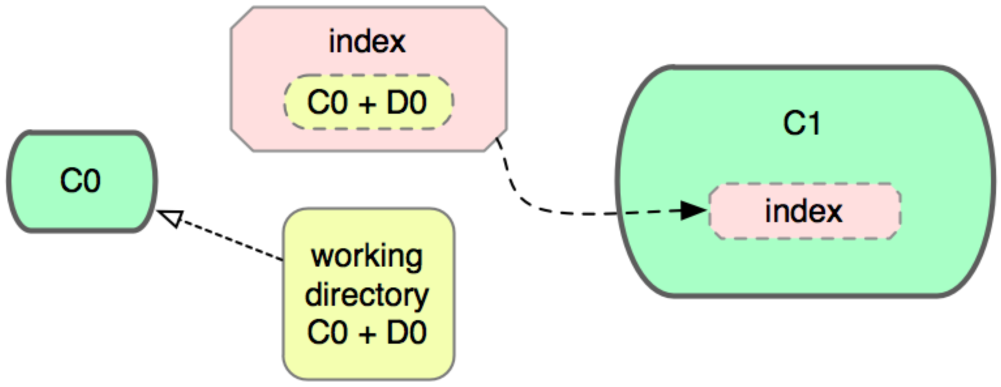


git commit -m

Create a commit with the given commit message

```
$ git commit -m "more ambition!"
```

Commit is rolled from index

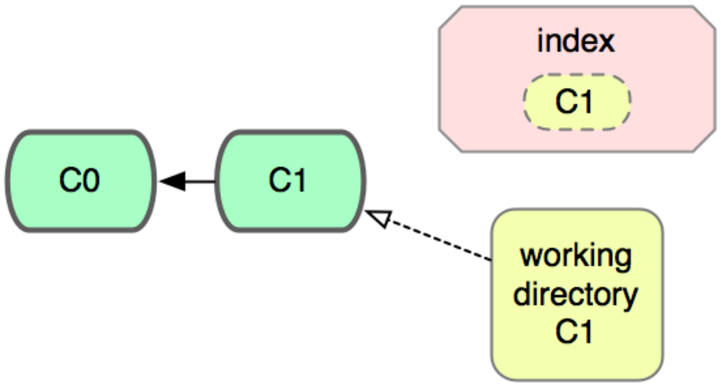


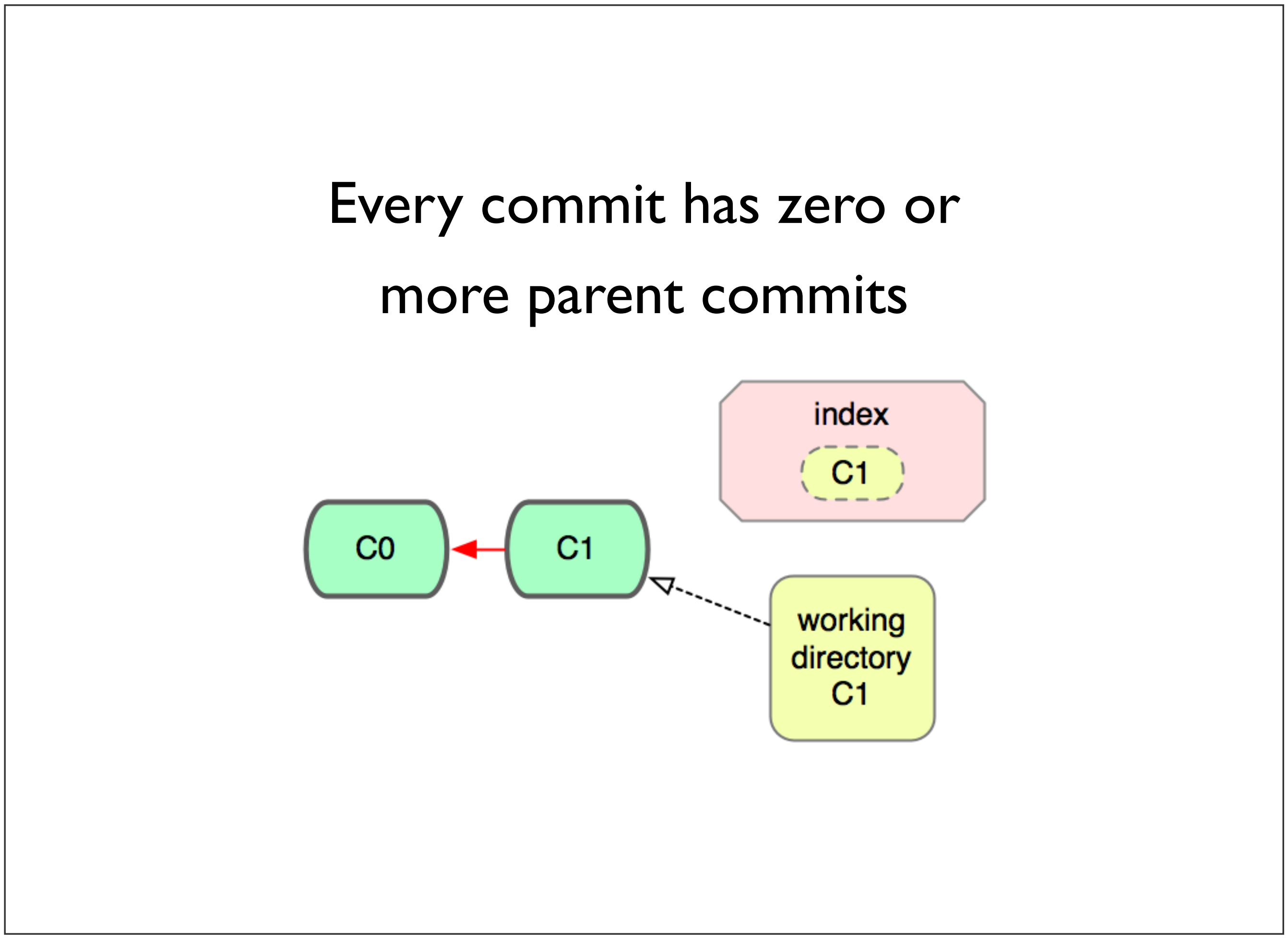
Remember:

1. Make changes to working dir
2. Stage those changes to the index
3. Commit the current state of the index

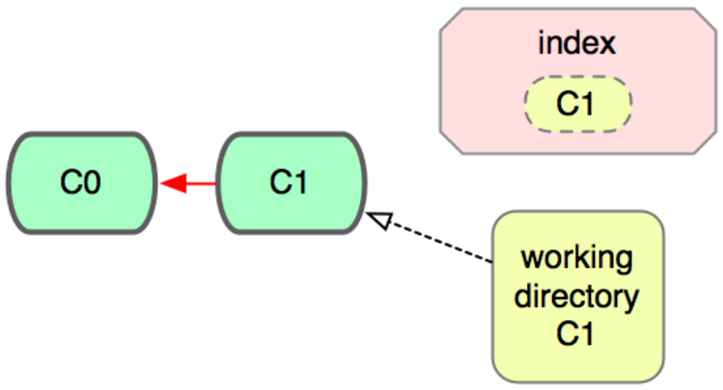
Recap

Two commits, C0 and C1





Every commit has zero or more parent commits



Branching and Merging

git branch

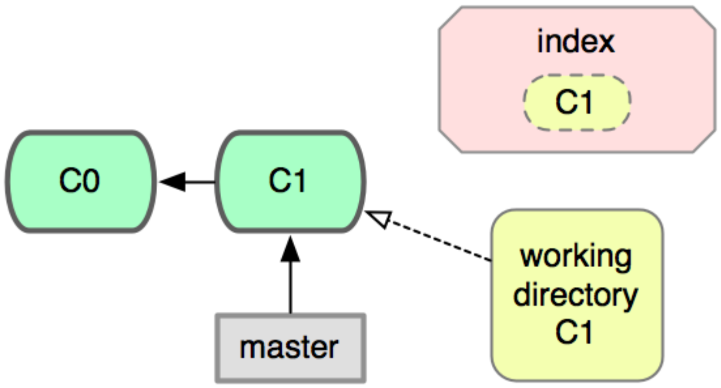
Show all local branches

```
$ git branch
```

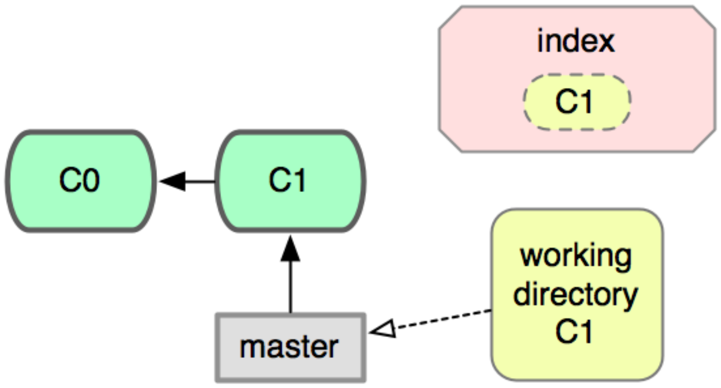
The default branch is named

master

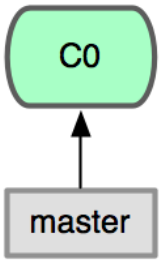
Branches are just pointers to commits



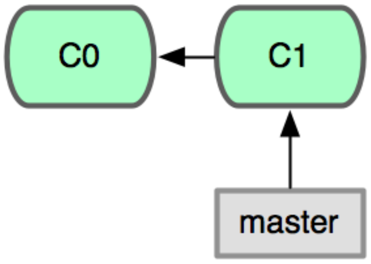
It's easiest to think of the working directory as corresponding to a branch

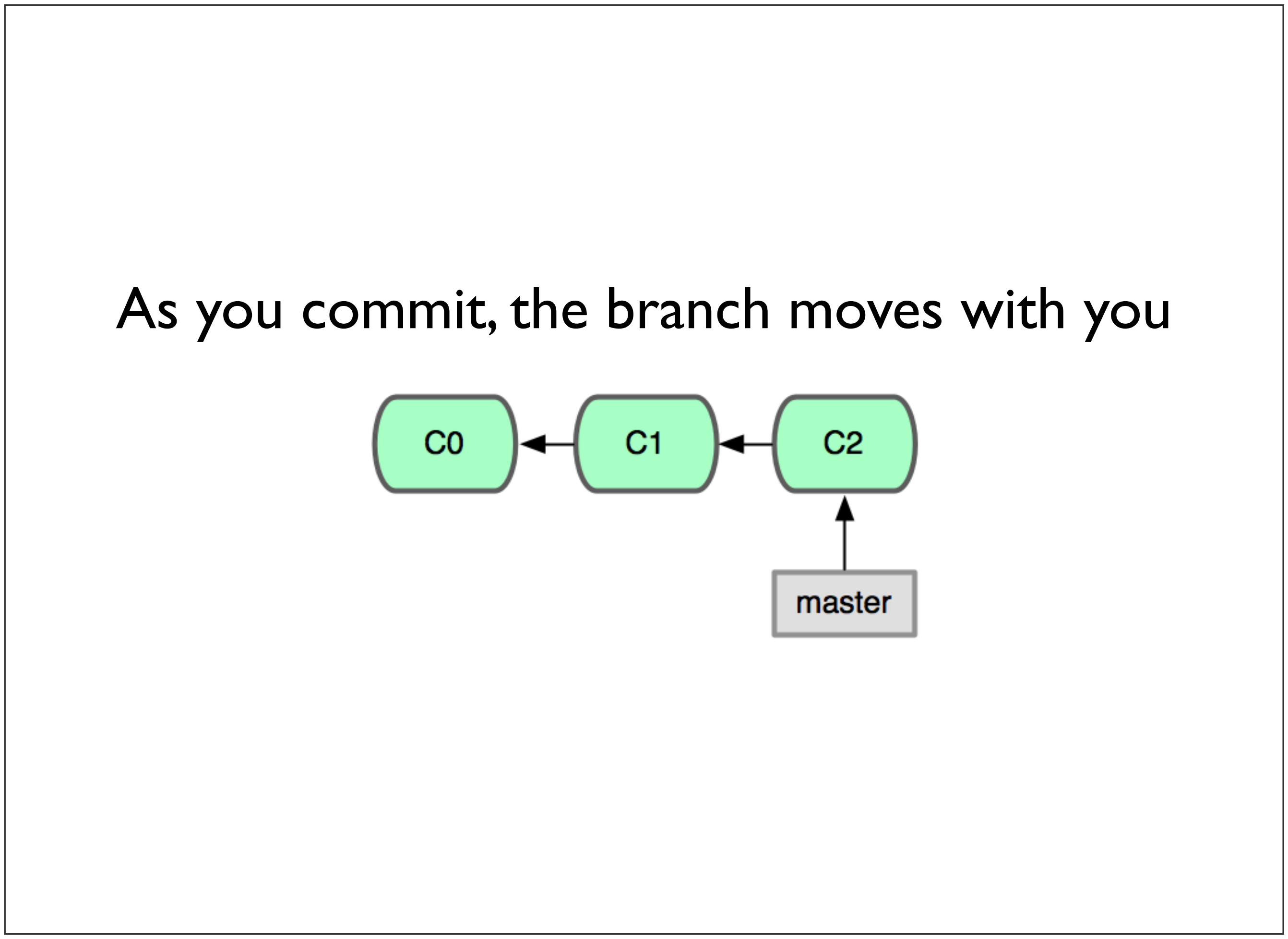


As you commit, the branch moves with you

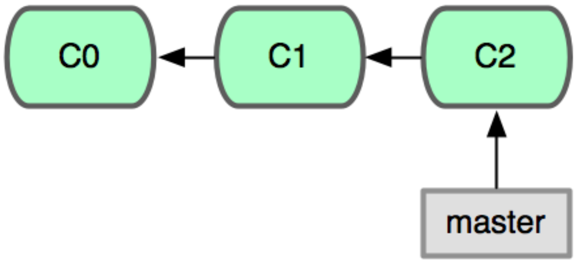


As you commit, the branch moves with you





As you commit, the branch moves with you

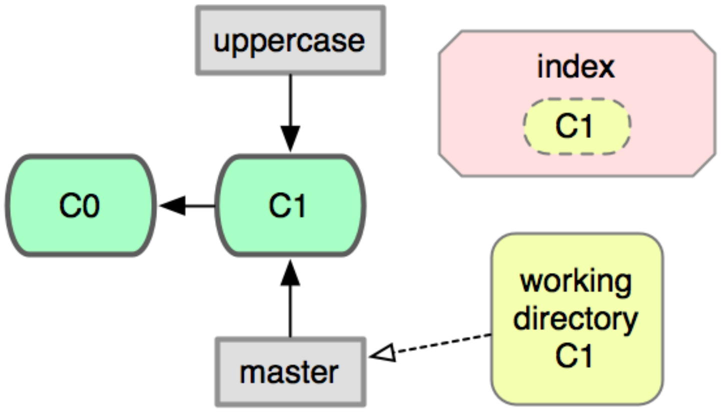


git branch

Create a new branch pointing at the current commit

```
$ git branch uppercase
```

A new branch has been created but the working dir has not changed

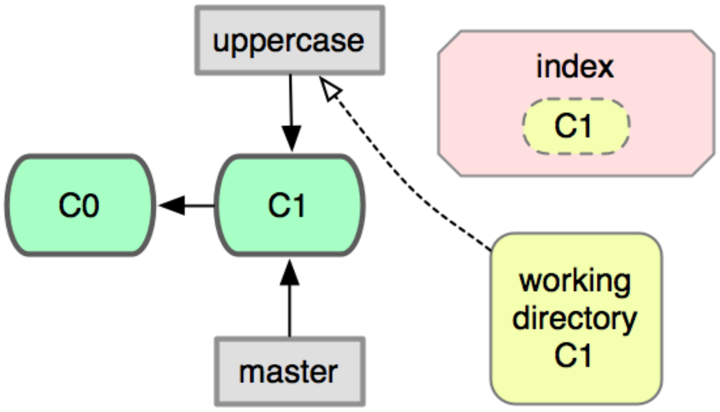


git checkout

Switch working dir to the given branch

```
$ git checkout uppercase
```

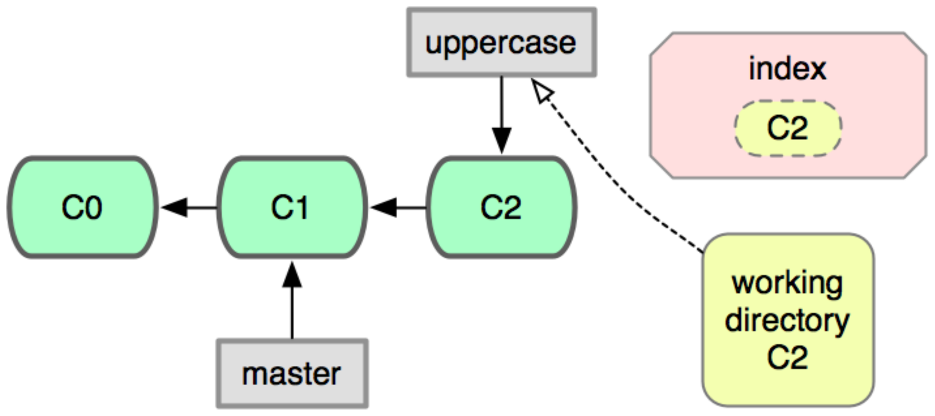
Working dir now corresponds to uppercase



Convert the string to uppercase on this branch and commit the change

```
$ vim hello.sh
$ git add -p
$ git commit -m 'convert string to uppercase'
```

Branches have now diverged!



git branch -v

Show branches and the commits they point to

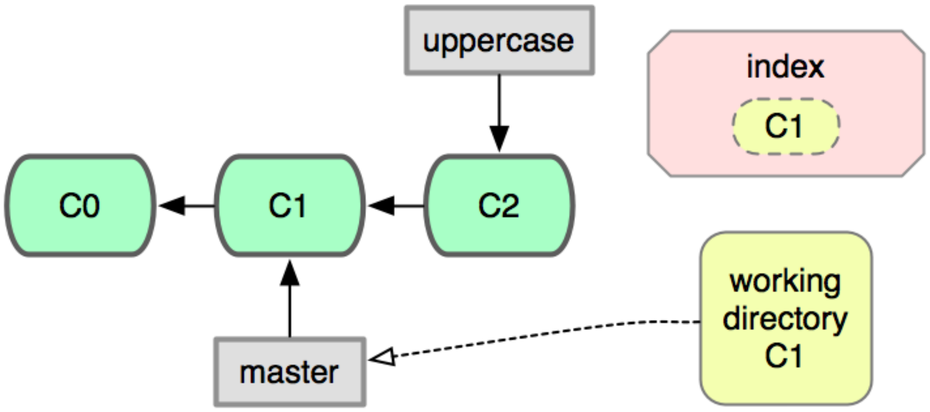
```
$ git branch -v
```

Switch back to the master branch.

Notice that working dir has been changed.

```
$ cat hello.sh         # uppercase version
$ git checkout master
$ cat hello.sh         # master version
```

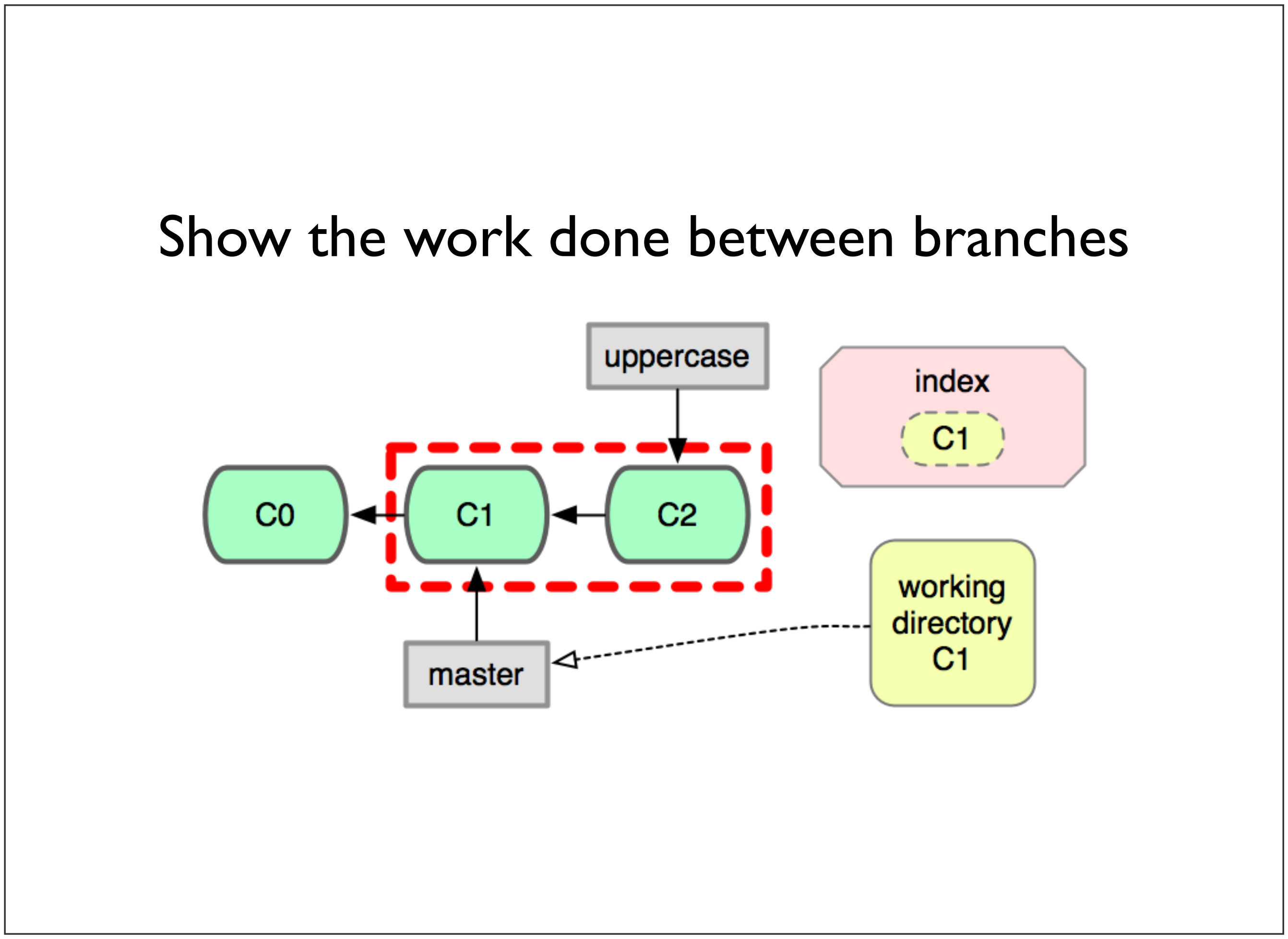
Working directory is now consistent with the master branch



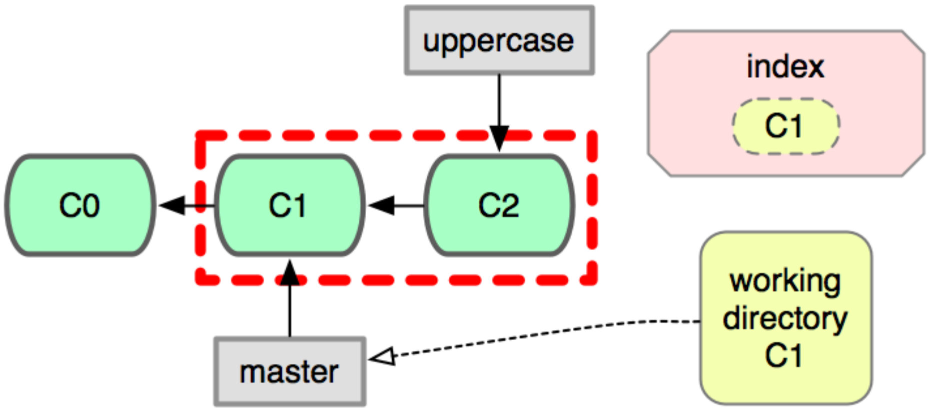
git diff R1 R2

Diff between two arbitrary commits

```
$ git diff master uppercase
```



Show the work done between branches

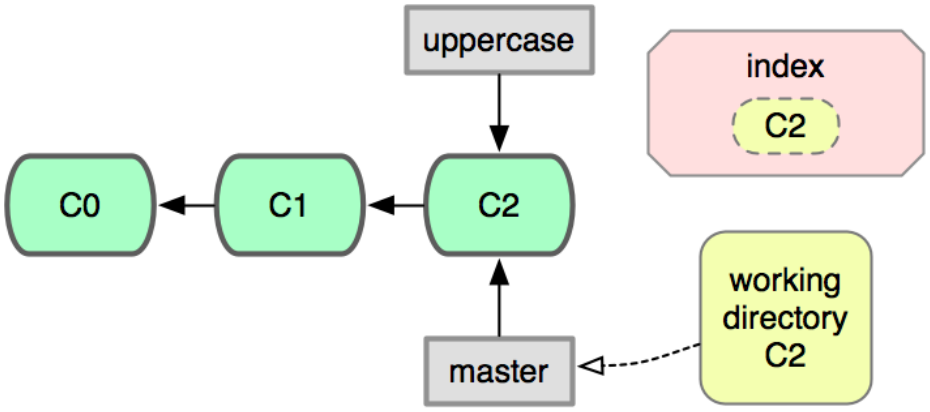


git merge

Merge the given commit into the current branch

```
$ git merge uppercase
```

Both branches now point at the same commit



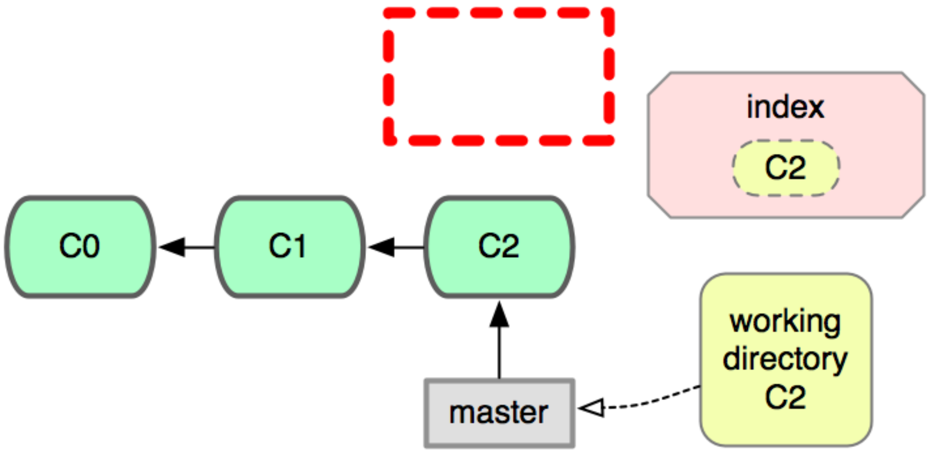
This kind of merge is known as a fast-forward merge because the merged branch was a direct descendent

git branch -d

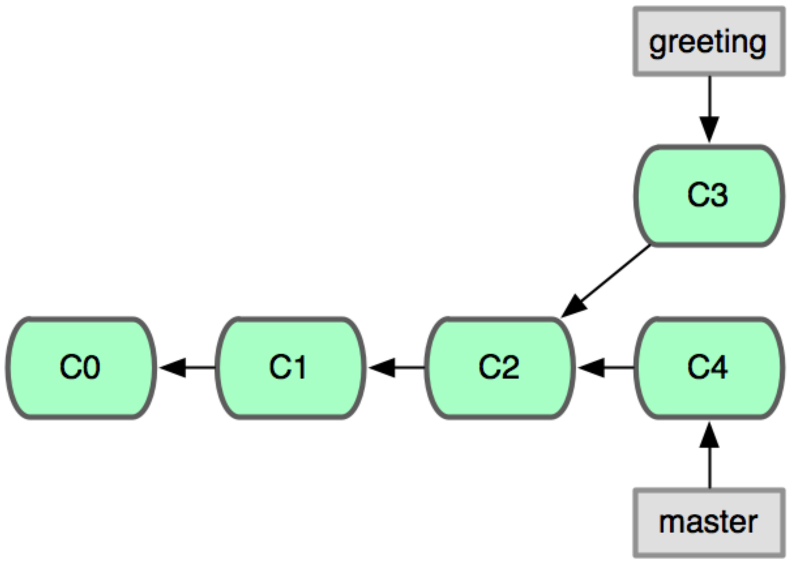
Delete the given branch

```
$ git branch -d uppercase
```

Only the pointer has been deleted



What if both branches have commits?



git checkout -b

Create a new branch and switch to it

```
$ git checkout -b greeting
```

Modify the greetings; commit; and switch back to master

```
$ vim hello.sh
$ vim goodbye.sh
$ git add -p
$ git commit -m 'new greetings'
$ git checkout master
```

Create a new file on master and attempt to add it

```
$ vim README
$ git add -p         # nothing to review!
$ git status         # find out why
```

File tracking

Git remembers what files have been added.

New files must be explicitly added.

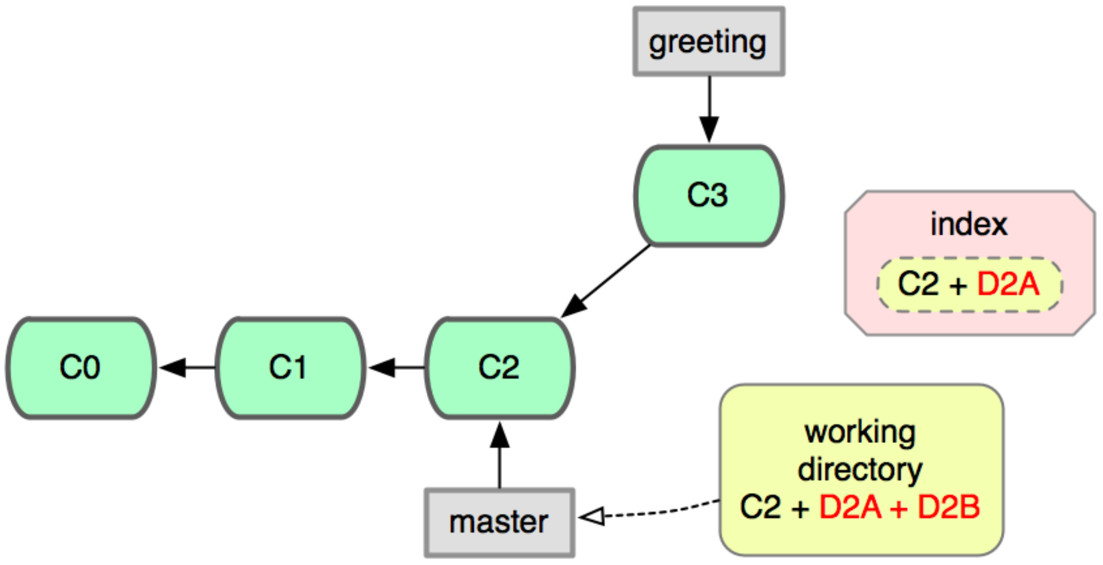
Add the contents of the new file

```
$ git add .     # add everything!
$ git status    # see that it worked
```

Oops, we forgot something...

```
$ vim README
$ git status        # make sure
```

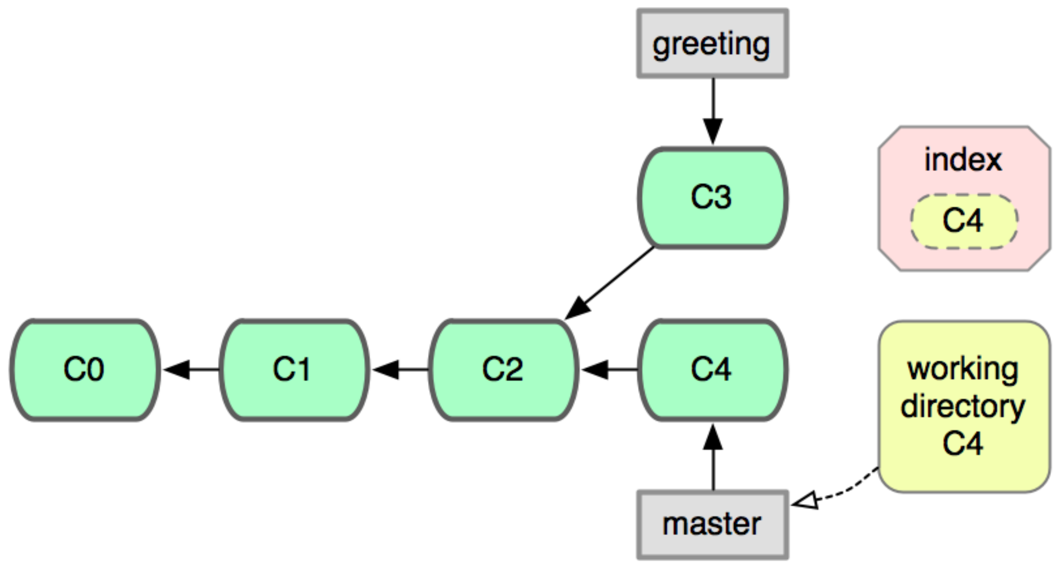
Two deltas have been introduced



Inspect the deltas, and continue

```
$ git diff --staged # old change
$ git diff          # new change
$ git add -p        # add again
$ git commit -m "add a readme"
```

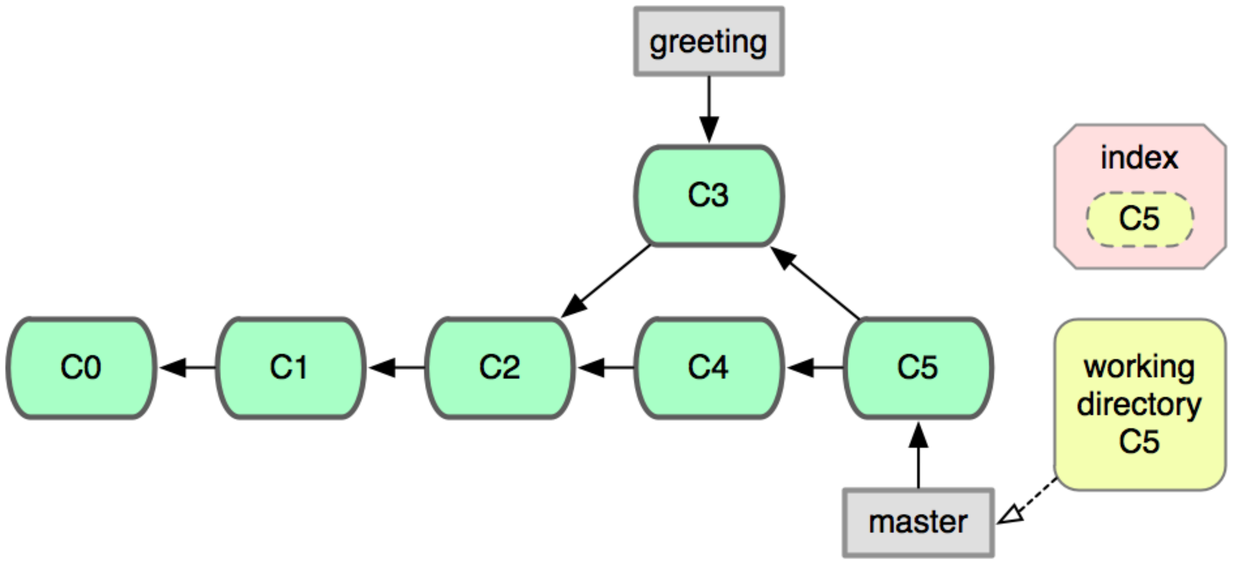
Recap of forked lineage



Merge greeting into master

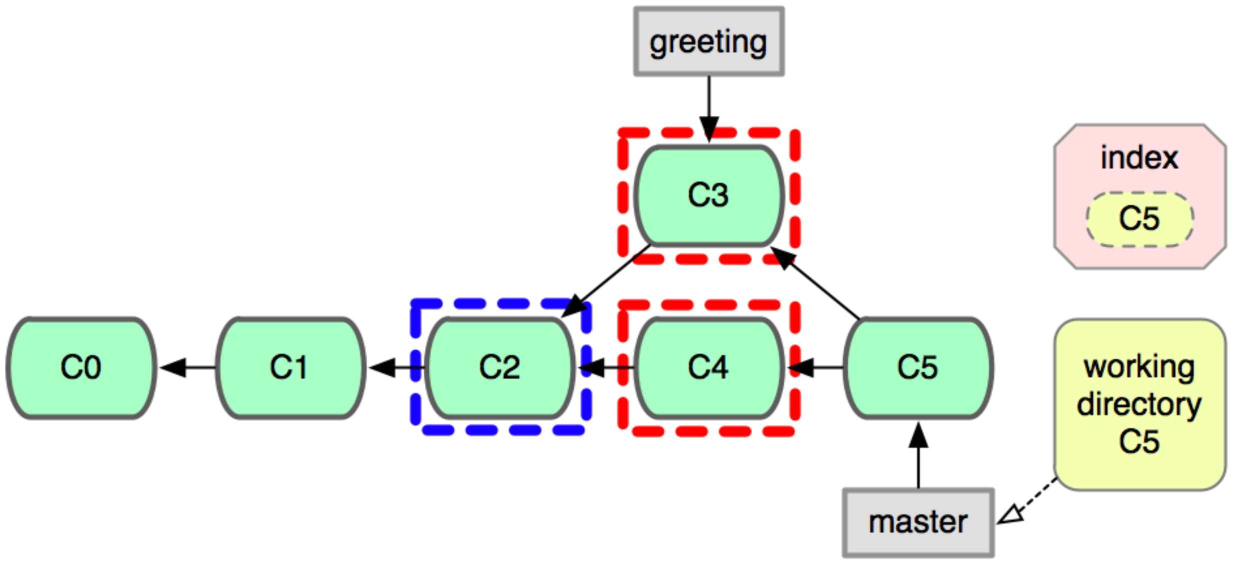
```
$ git merge greeting
```

A new merge commit (C5) is created



This kind of merge is known as
a recursive merge and
uses a 3-way merge strategy

Three way (recursive) merge strategy



git log --graph

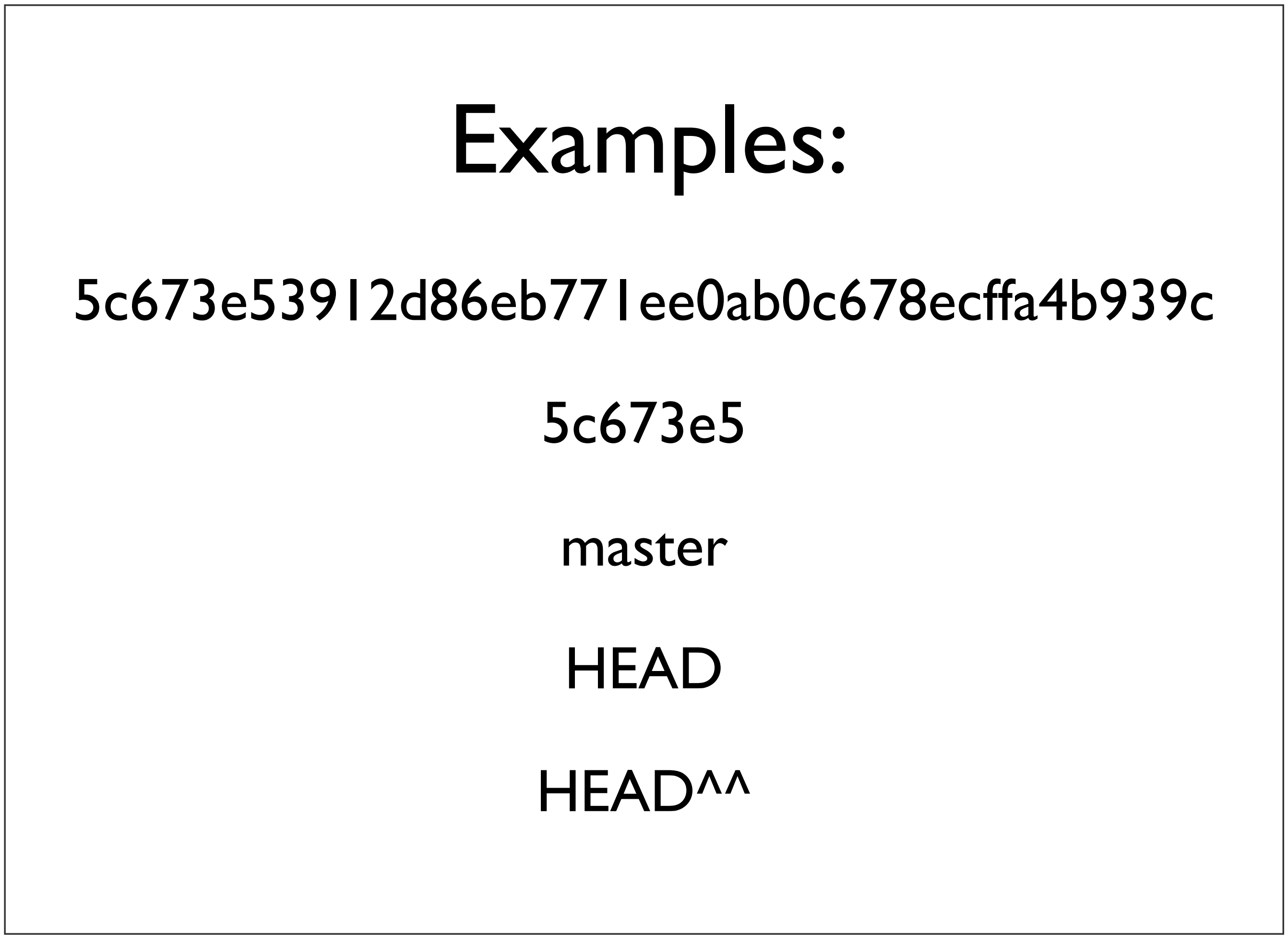
Show the commit log with graph structure

```
$ git log --graph
```

The power of Undo

First, a word about references

A reference is a way to refer to a commit



Examples:

5c673e53912d86eb771ee0ab0c678ecffa4b939c

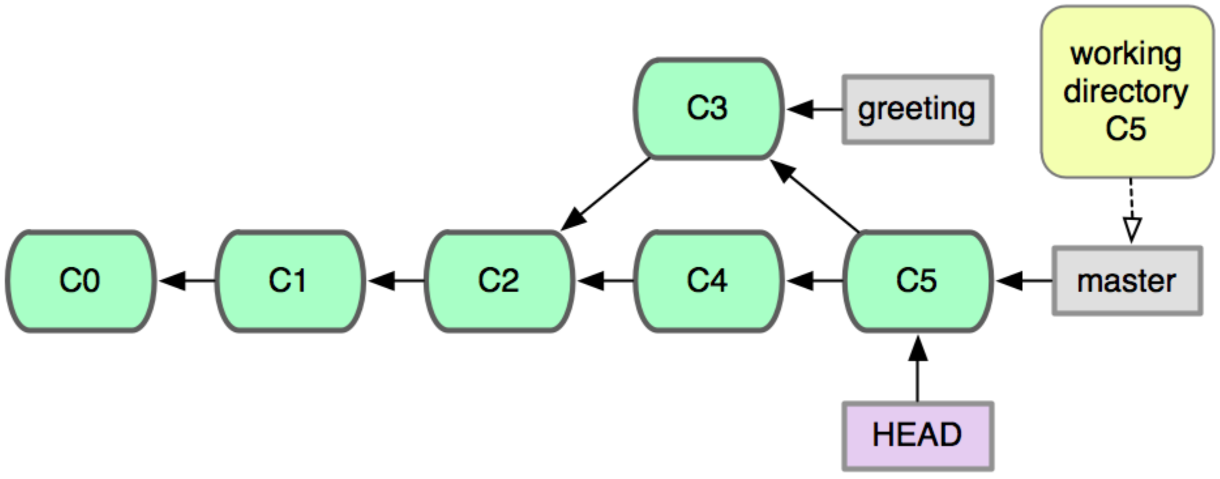
5c673e5

master

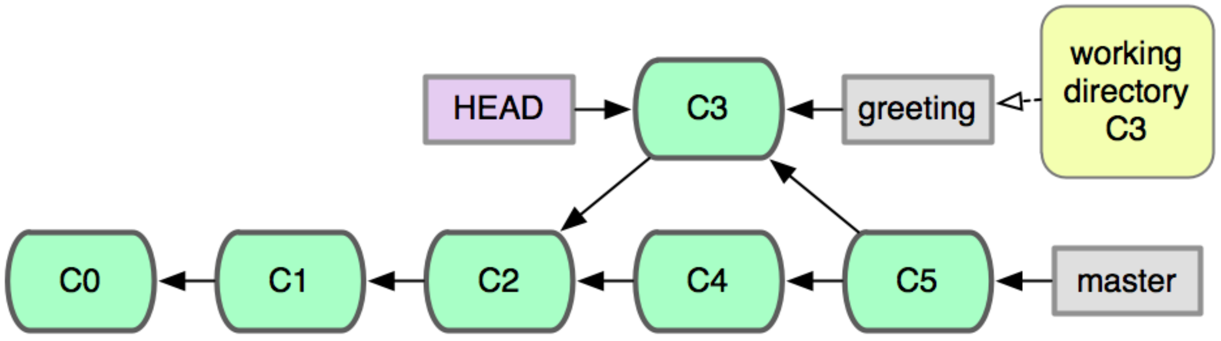
HEAD

HEAD^^

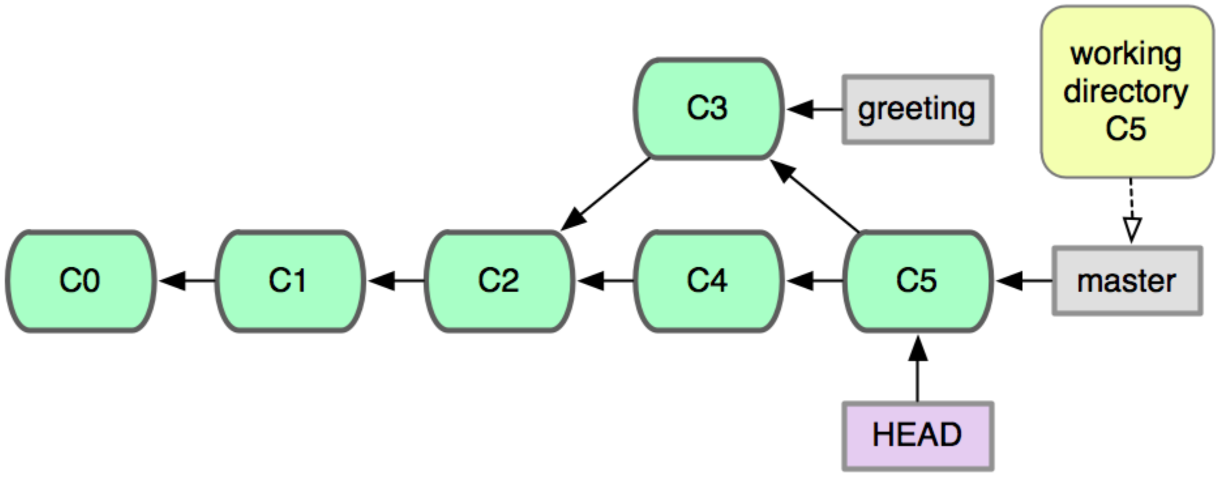
HEAD is a dynamic reference that follows your current checkout

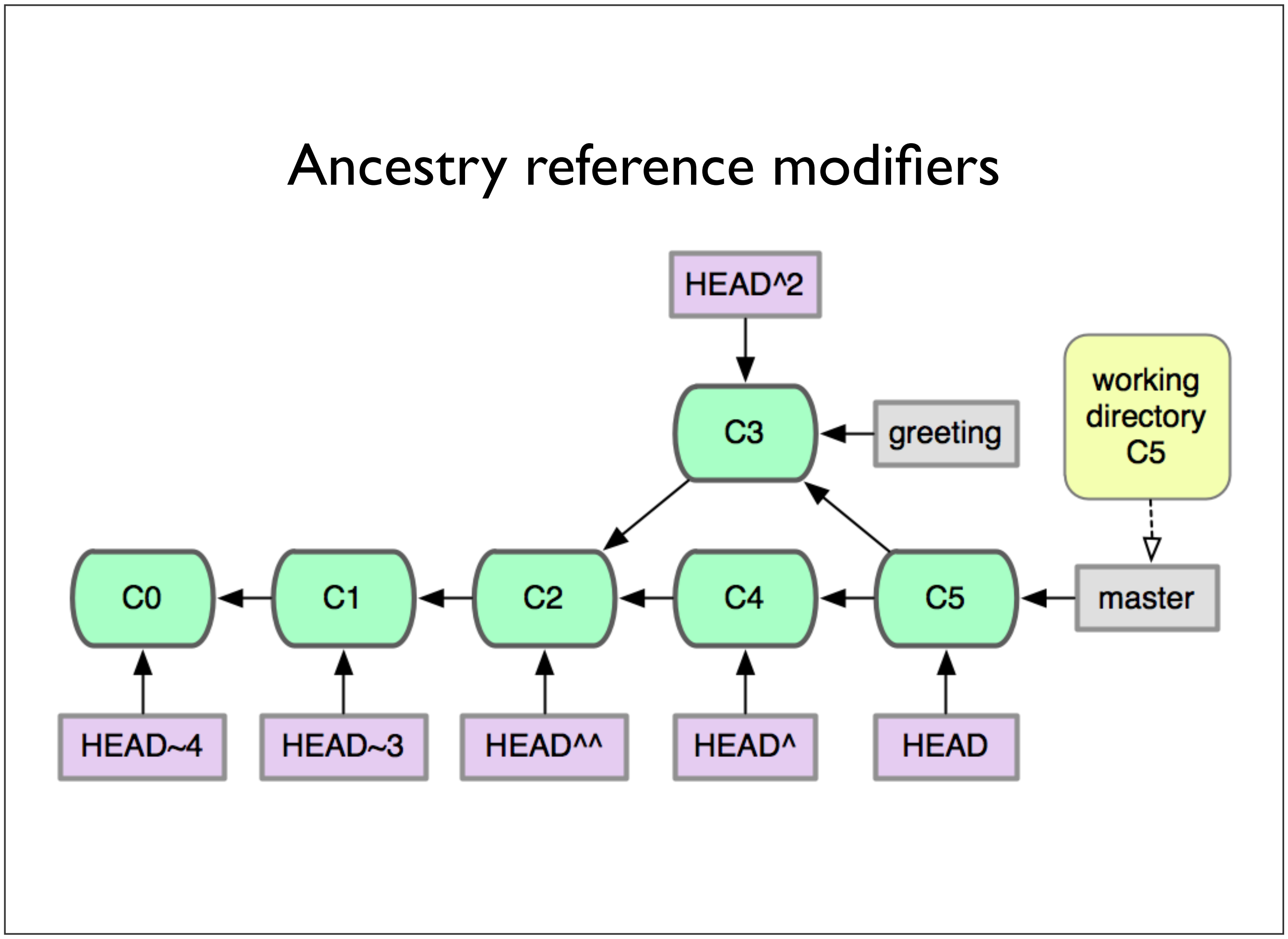


HEAD is a dynamic reference that follows your current checkout

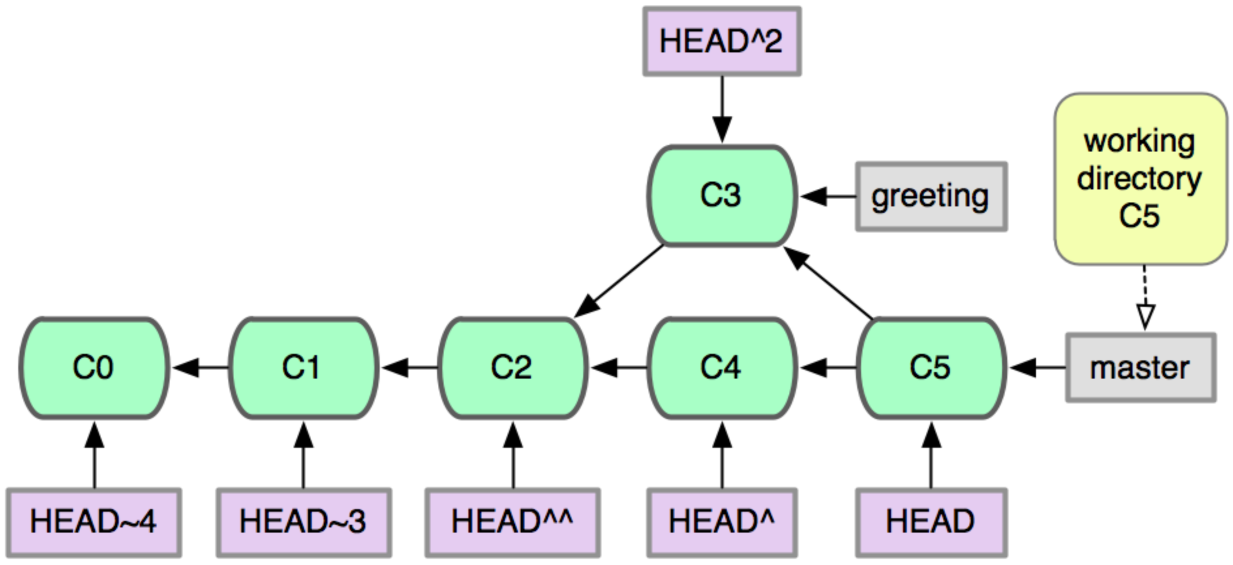


HEAD is a dynamic reference that follows your current checkout





Ancestry reference modifiers

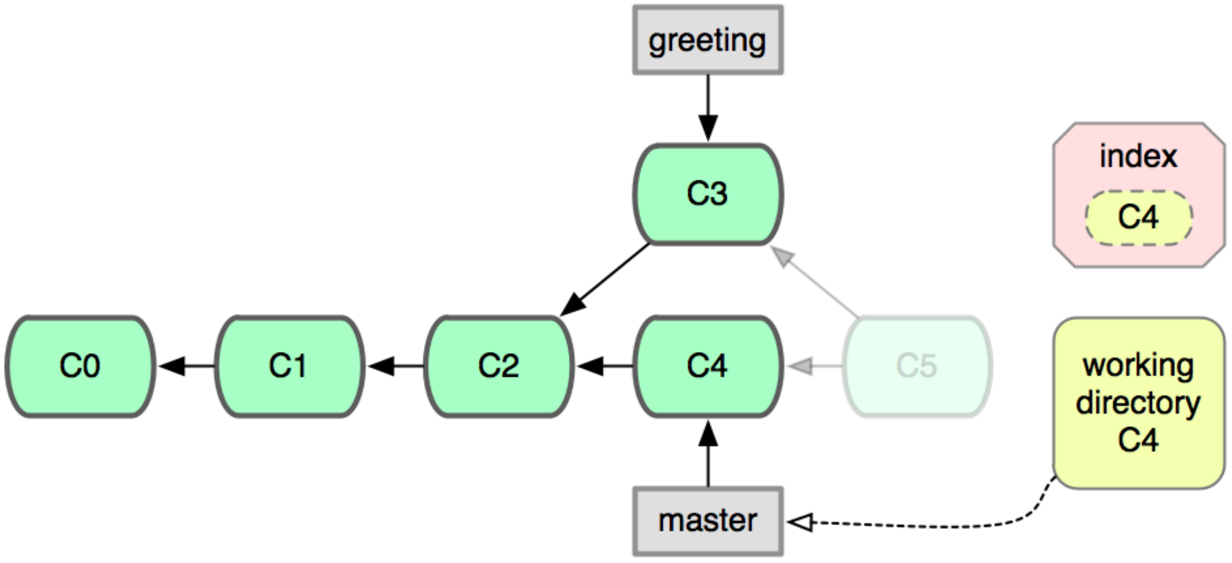


git reset --hard

Reset a branch and working dir

```
$ git reset --hard head^
```

master is now pointing to C4.
C5 still exists, but is dangling.



git reflog

Show previous values of HEAD

```
$ git reflog
```

What about merge conflicts?

Modify a line that was also changed in the greeting branch

```
$ vim hello.sh
$ git add -p
```

git commit --amend

Modify the content of the last commit

```
$ git commit --amend
```

Attempt to merge greeting

```
$ git merge greeting
```

Cleanly merged files are staged.

Conflicts are left in working dir.

```
$ git diff --staged  # clean
$ git diff           # dirty
$ git status         # summary
```

Clean up the mess; use git add to mark a conflicted file as properly merged. Commit to seal the deal.

```
$ git add hello.sh
$ git commit
```

Collaboration

Back to your repos directory

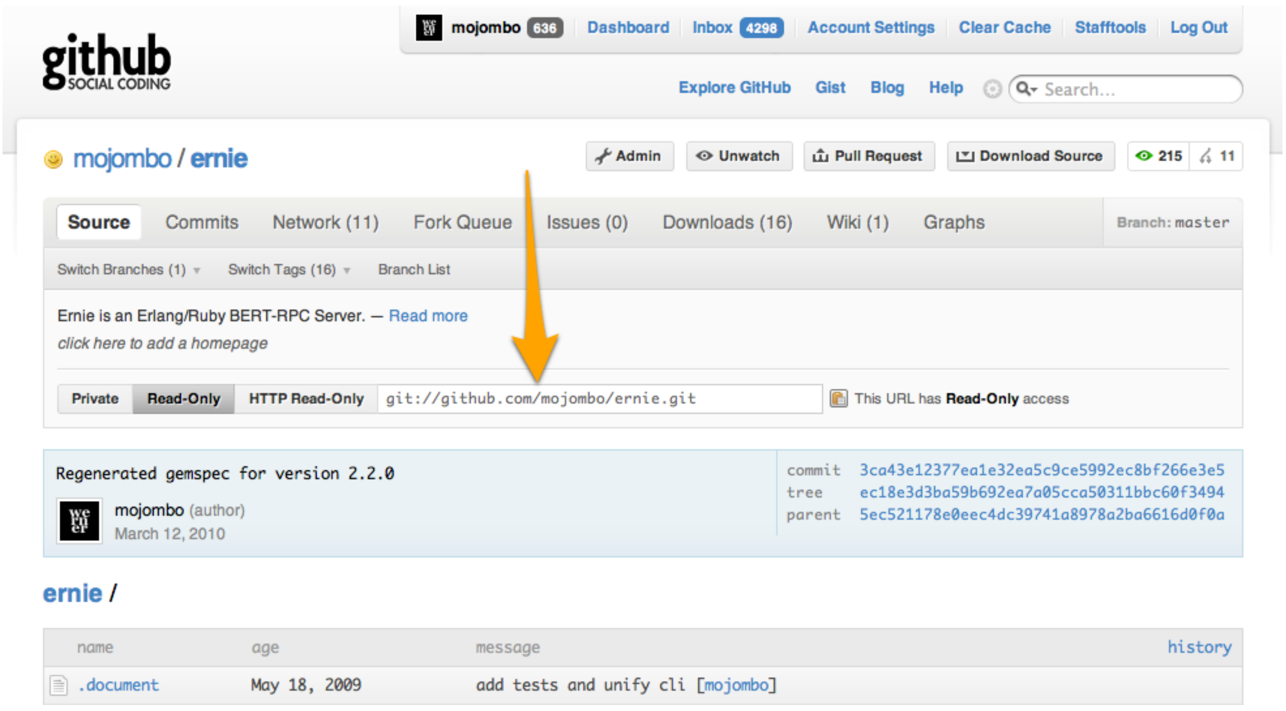
```
$ cd ..
```

git clone

Clone a repository

```
$ git clone hello helloclone
$ cd helloclone
```

You can also clone from remote repositories



Ernie is an Erlang/Ruby BERT-RPC Server. — Read more
click here to add a homepage

git://github.com/mojombo/ernie.git This URL has **Read-Only** access

Regenerated gemspec for version 2.2.0
mojombo (author)
March 12, 2010

```
commit  3ca43e12377ea1e32ea5c9ce5992ec8bf266e3e5
tree    ec18e3d3ba59b692ea7a05cca50311bbc60f3494
parent  5ec521178e0eec4dc39741a8978a2ba6616d0f0a
```

ernie /

name	age	message	history
.document	May 18, 2009	add tests and unify cli [mojombo]	

git remote

Display a list of remotes

```
$ git remote -v
```

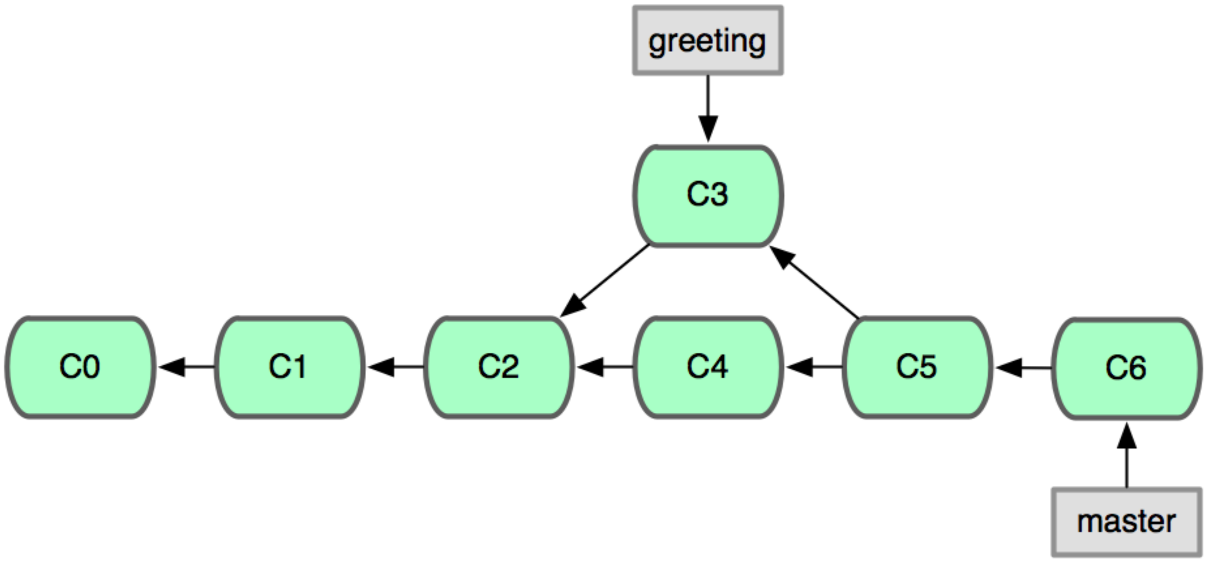
git branch -a

Show all branches (local and remote)

```
$ git branch -a
```

Assume that upstream makes a commit on the master branch

Current state of upstream

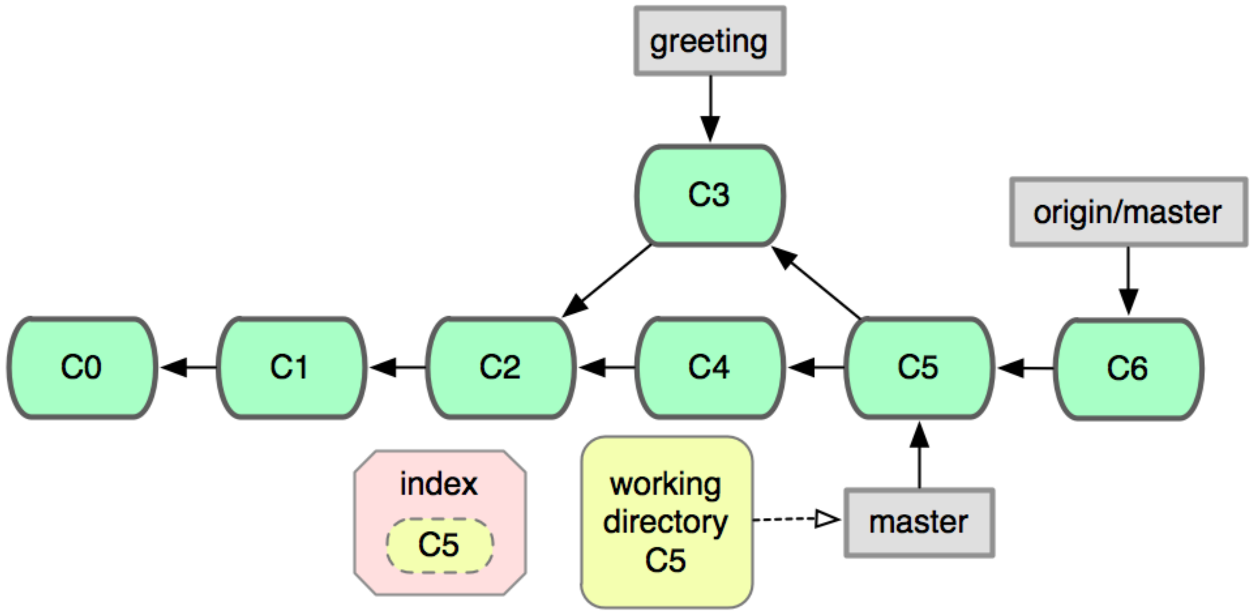


git fetch

Fetch commits from the given remote

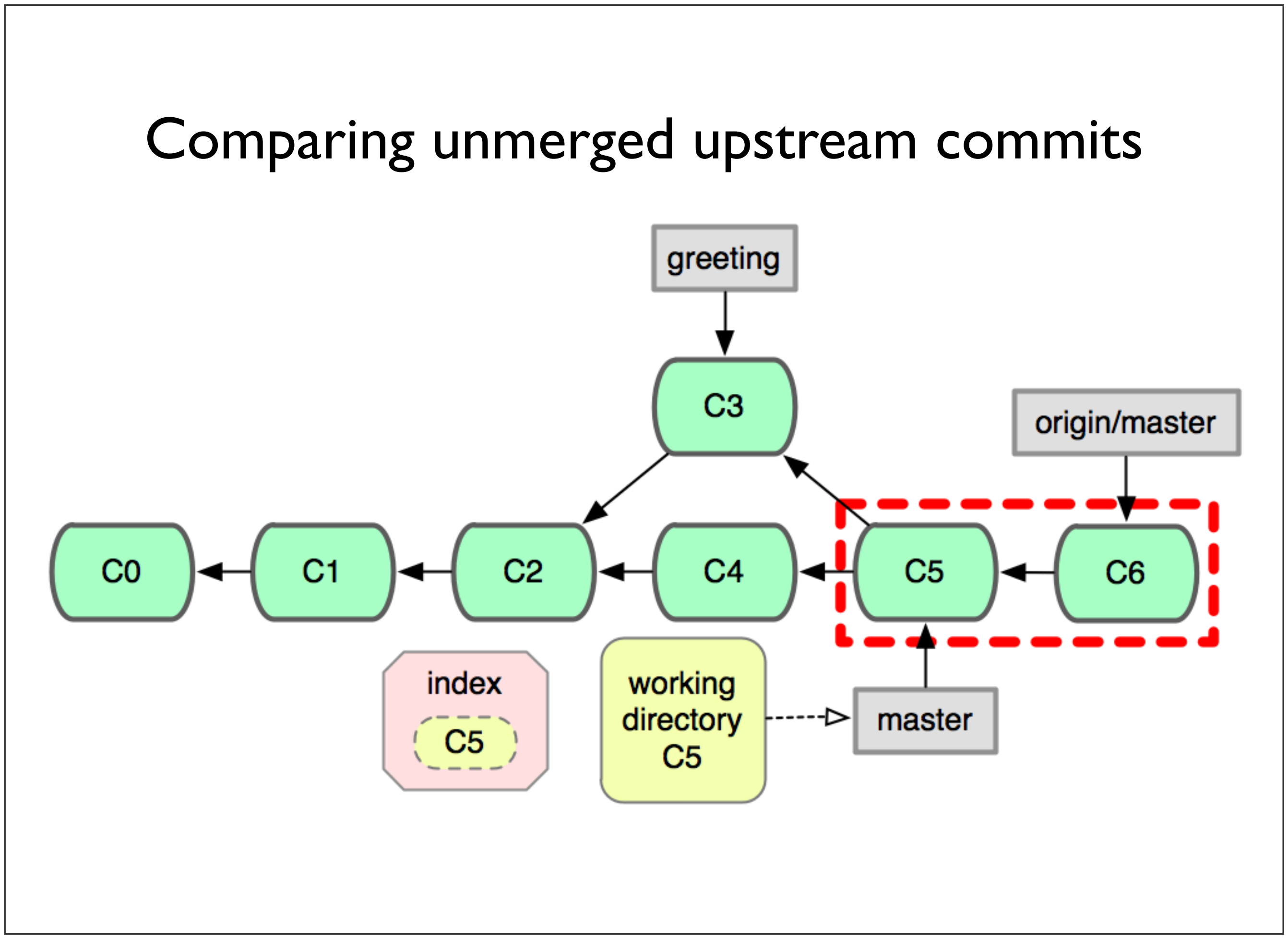
```
$ git fetch origin
```

Remote commits have been downloaded, but have not affected your local branches

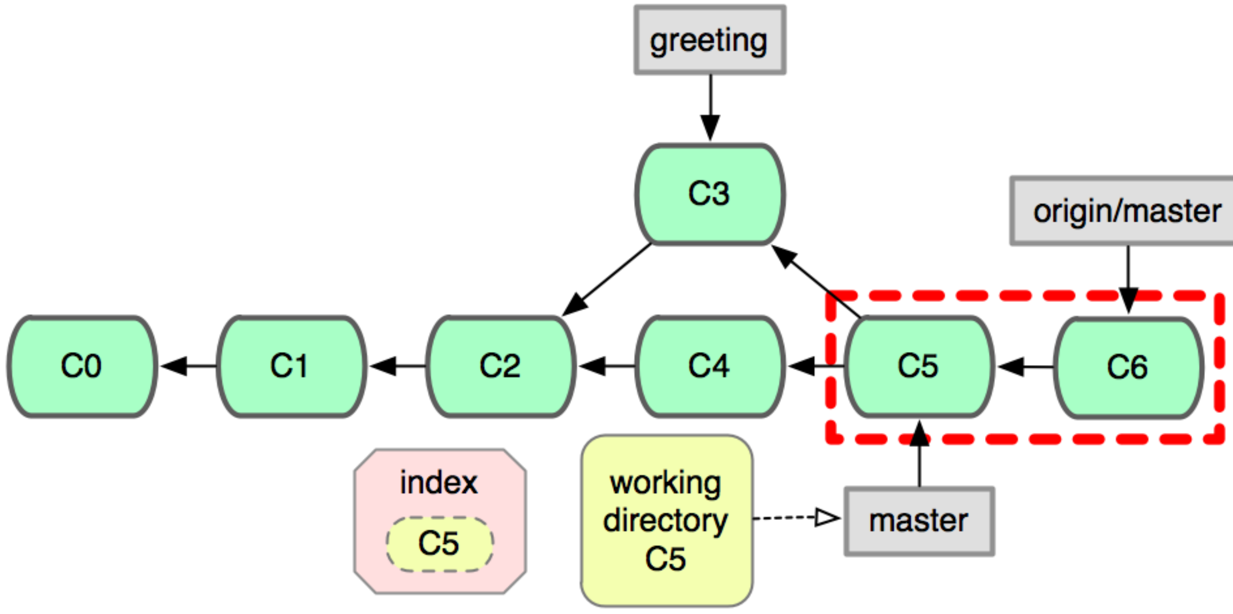


View the changes in the upstream origin/master branch

```
$ git diff head origin/master
```



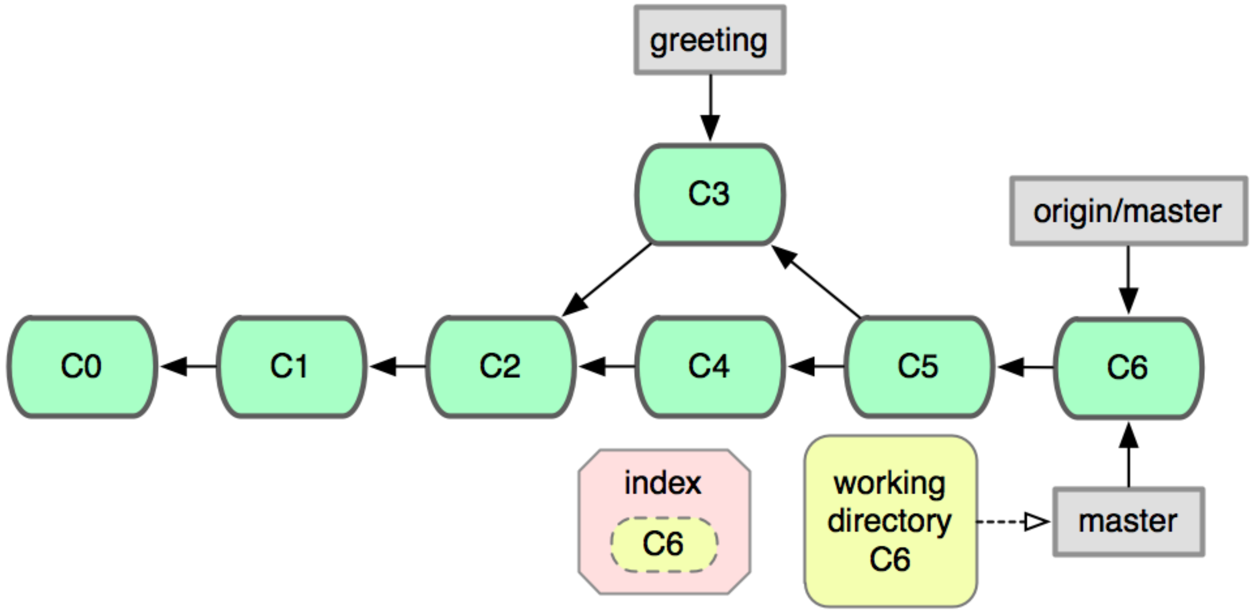
Comparing unmerged upstream commits



Merge the upstream origin/master branch into your local master branch

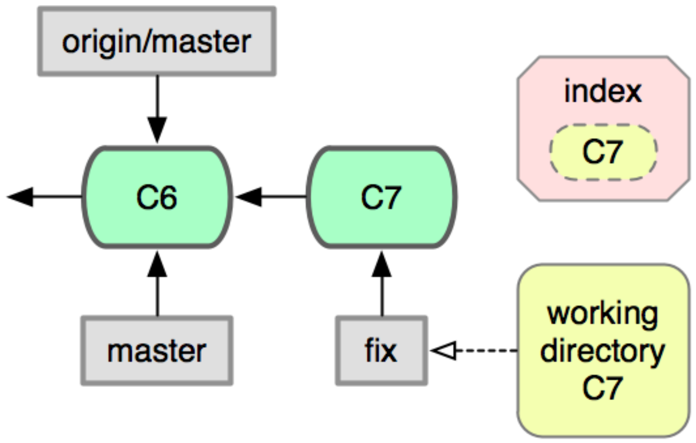
```
$ git merge origin/master
```

The upstream commits have been merged



What if you want to share your changes?

Assume a new branch fi x with a commit

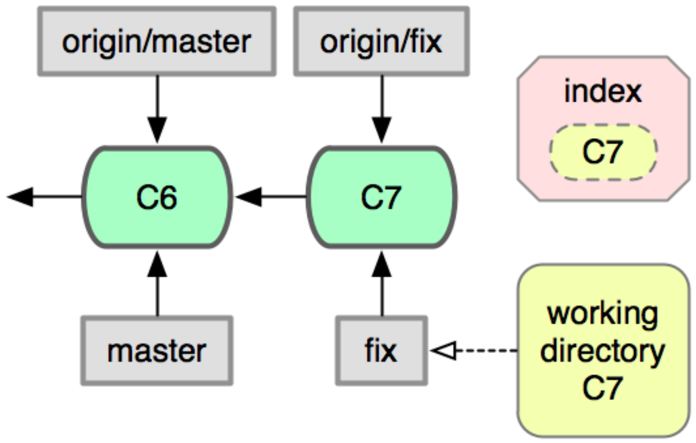


git push

Push some commits to a remote

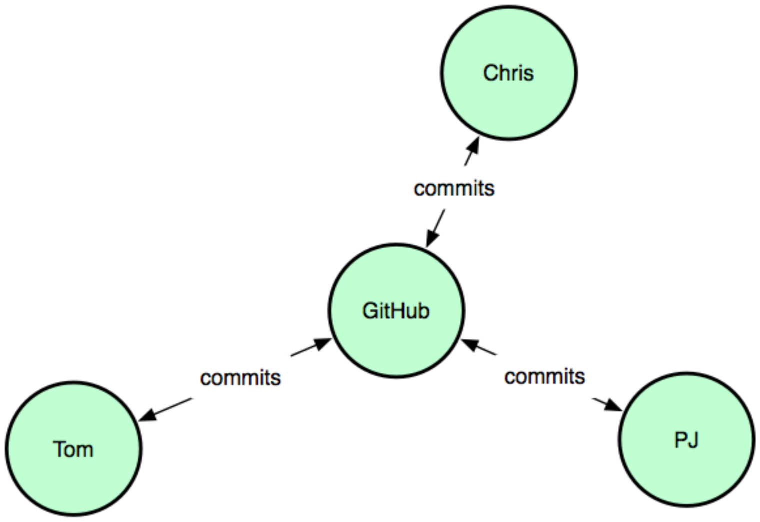
```
$ git push origin fix
```

Remote now contains the fi x branch!



Now others that have access to the remote can fetch and work on the fix branch

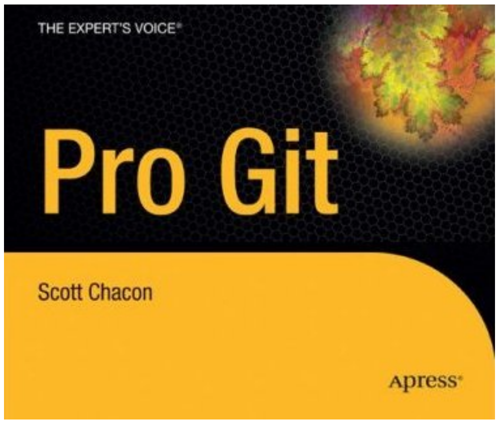
Collaboration in Git is all about moving commits around between repositories



Where do I go from here?

Pro Git by Scott Chacon

http://progit.org



Thanks!