Selected Topics in Computer Networks

Nguyễn Quốc Đính

Faculty of IT, Ho Chi Minh City University of Industry

Aug 2015

Version Control and Makefile

to make your coding life easier

Some parts of this presentation are copied from

- **Athula Balachandran, Wolf Richter**, lecture notes in Computer Network, CMU
- **Tom Preston-Werner**, Mastering Git Basics

Did you do this?

Myprecisecode.c	{where I started}
Myprecisecode-1.c	{little change}
Myprecisecode-2.c	{little change}
…	
Myprecisecode-n.c	{little change}

Where do I stand?

Even worse when more than
2 guys co-work

call me maybe (git)

Getting started with git

Roll your own:

- `git config --global user.name “Nguyen Quoc Dinh”`
- `git config --global user.email “nqdinh@hui.edu.vn”`
- `git init .`

Not your own:

- `git config --global user.name “Nguyen Quoc Dinh”`
- `git config --global user.email “nqdinh@hui.edu.vn”`
- (1) `git clone git://github.com/nqd/Hello-World.git`
- (2) `git remote add origin git@github.com:nqd/Hello-World.git`

Types of Repositories

Type	Meaning
rsync://	rsync client to git repo
http://	HTTP hosted repo
https://	**HTTP with SSL**
git://	**Special git server/protocol**
ssh://	git via ssh

Playing around with GIT

- Assuming you installed git
- Begin with config

```
$ git config --global user.name "NQD"
$ git config --global user.email
"nqdinh@hui.edu.vn"
$ git config --global color.ui true
$ git config -l
```

Making directory for your project

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

git initialization

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

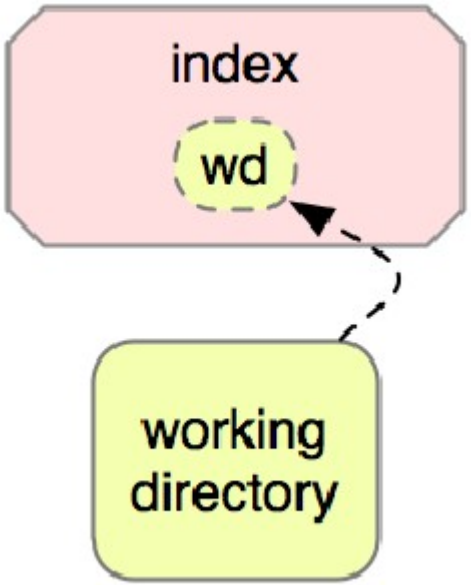
Write some code

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

Then … git add

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

Index now contains working directory



git status: show the status of index and working directory

```
$ git status
```

git commit: make a commit

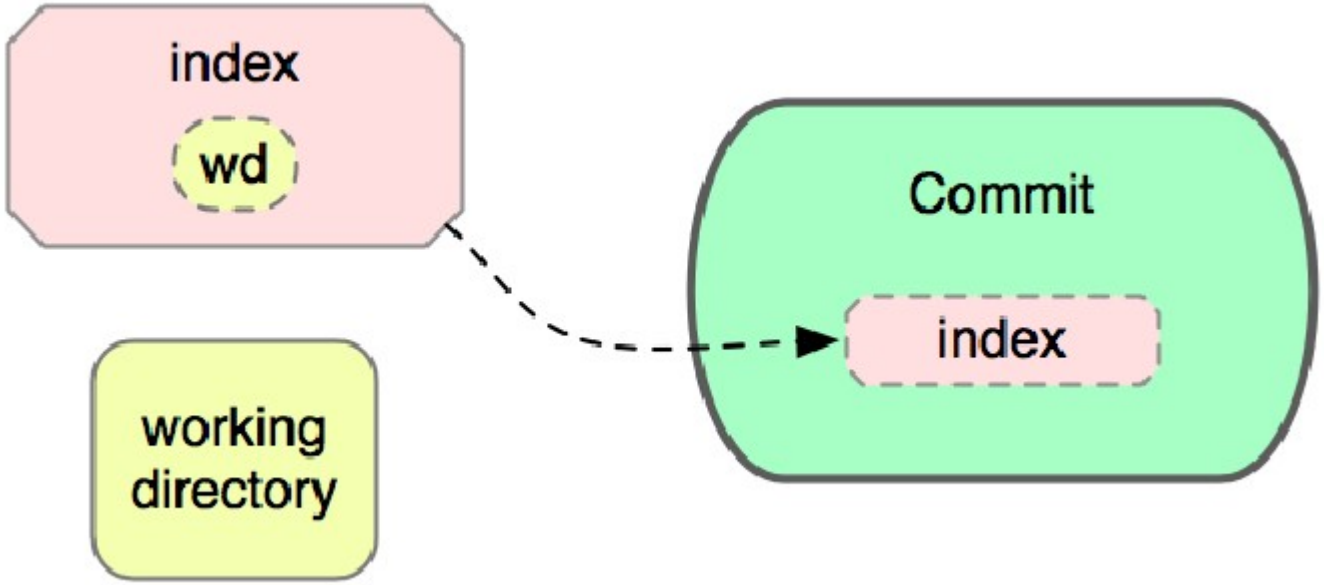
```
$ git commit  #make a commit
```

git log: print a log of commit

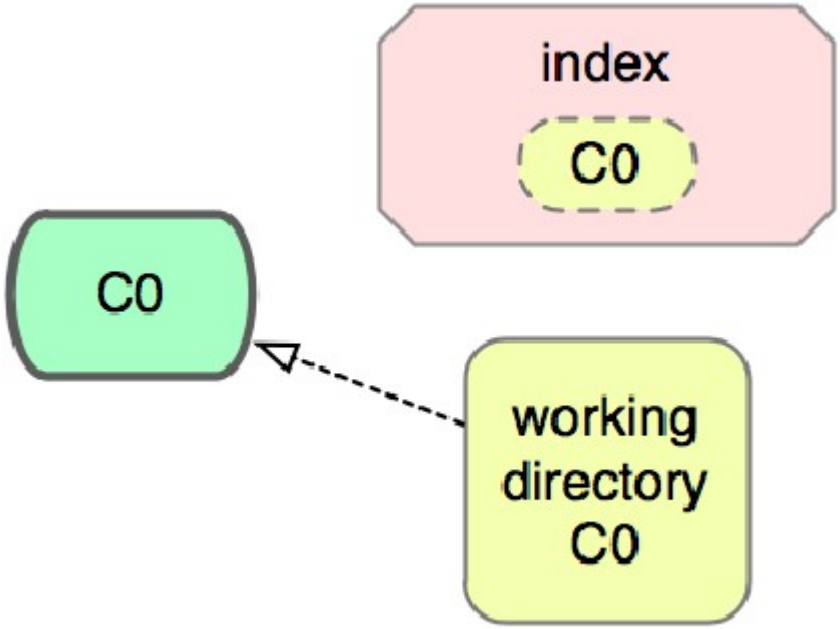
```
$ git log
```

A commit is a snapshot taken from the index

NOT THE WORKING DIRECTORY



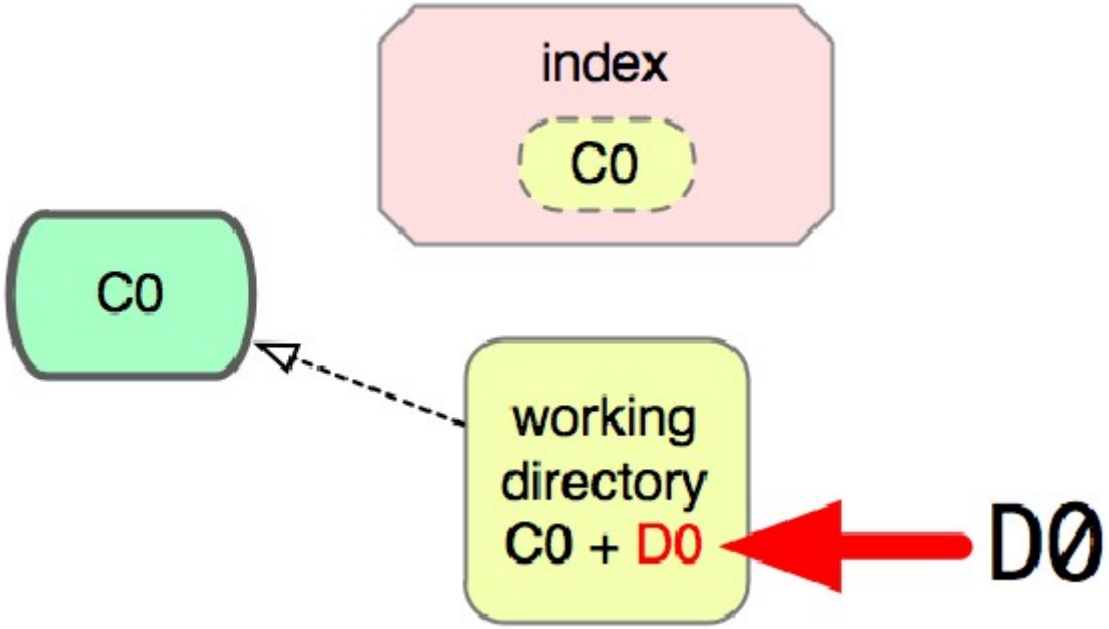
Current state of the repository



Make some ambiguous changes

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

Working directory now contains D0: a delta from C0

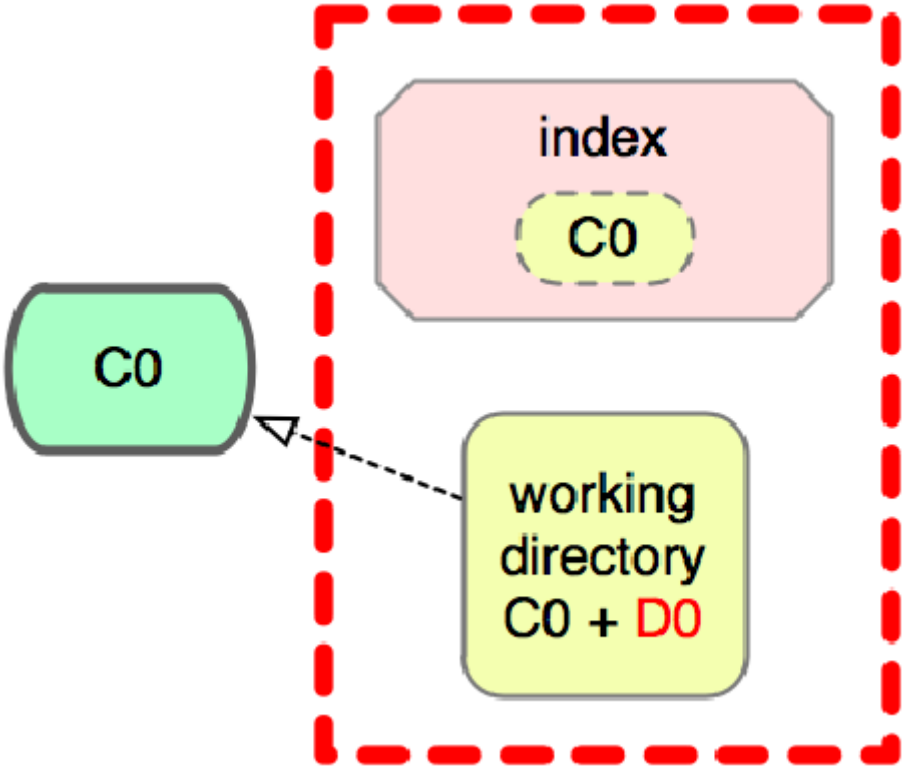


Review the change file

```
$ git status
```

Show the diff between index and working dir

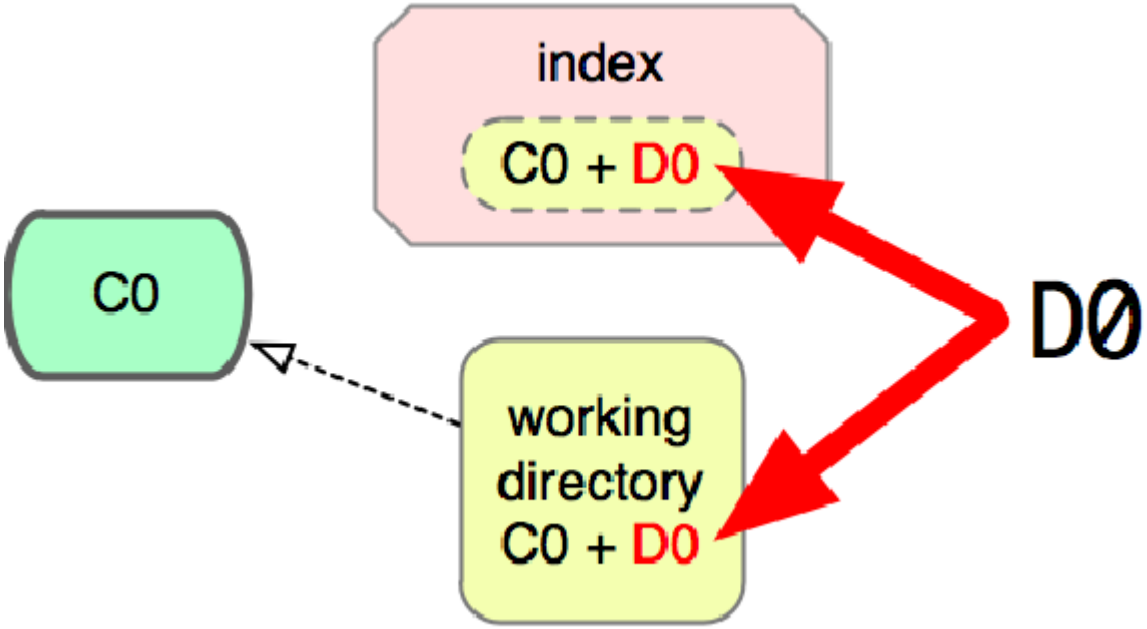
```
$ git diff
```



Interactively add changed hunks

```
$ git add -p
```

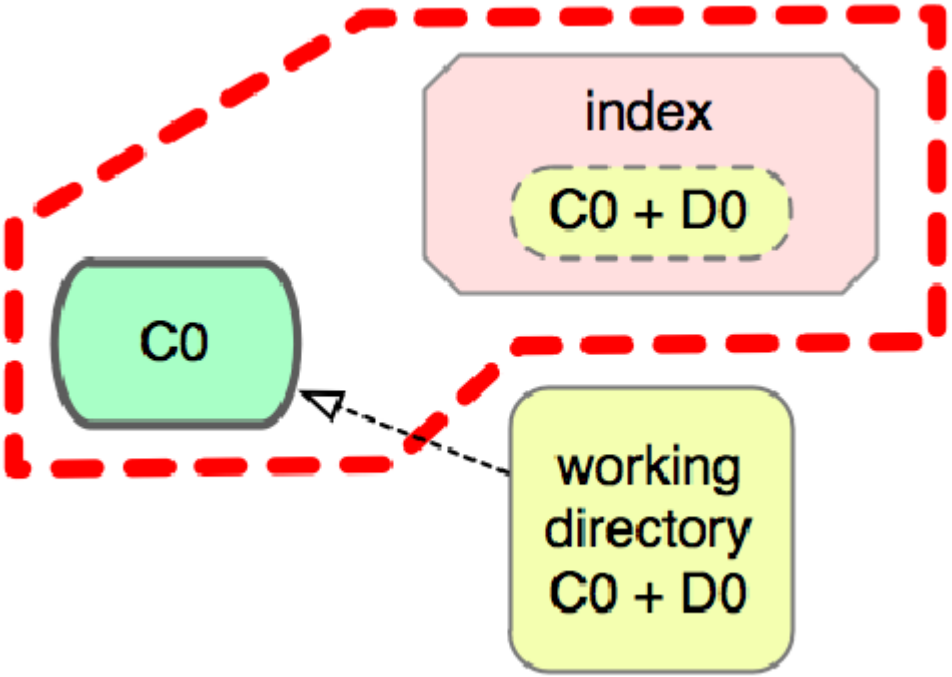
D0 is now in working directory and index



git diff now show nothing. Why?

Show diff between commit and index

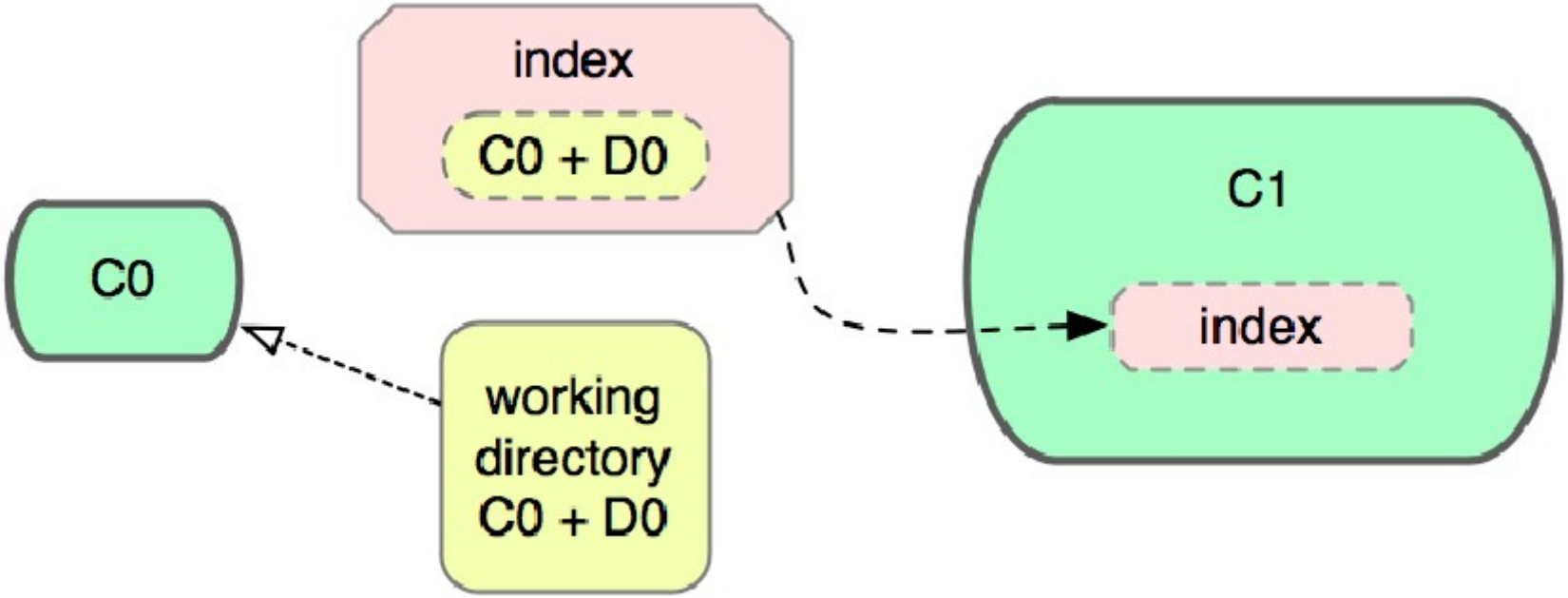
```
$ git diff --staged
```



Create a commit

```
$ git commit -m “more ambition”
```

Again, commit is rolled from index

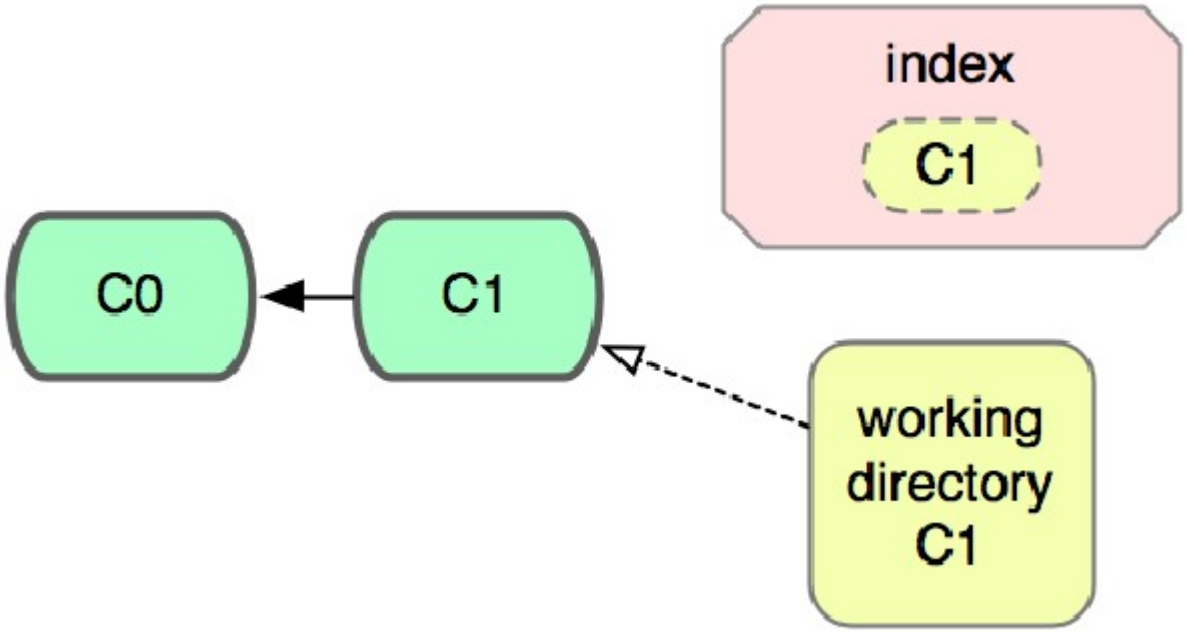


Remember

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

Recap

- Two commit, C0 and C1
- Every commit has zero or more parent commits



Branching and Merging

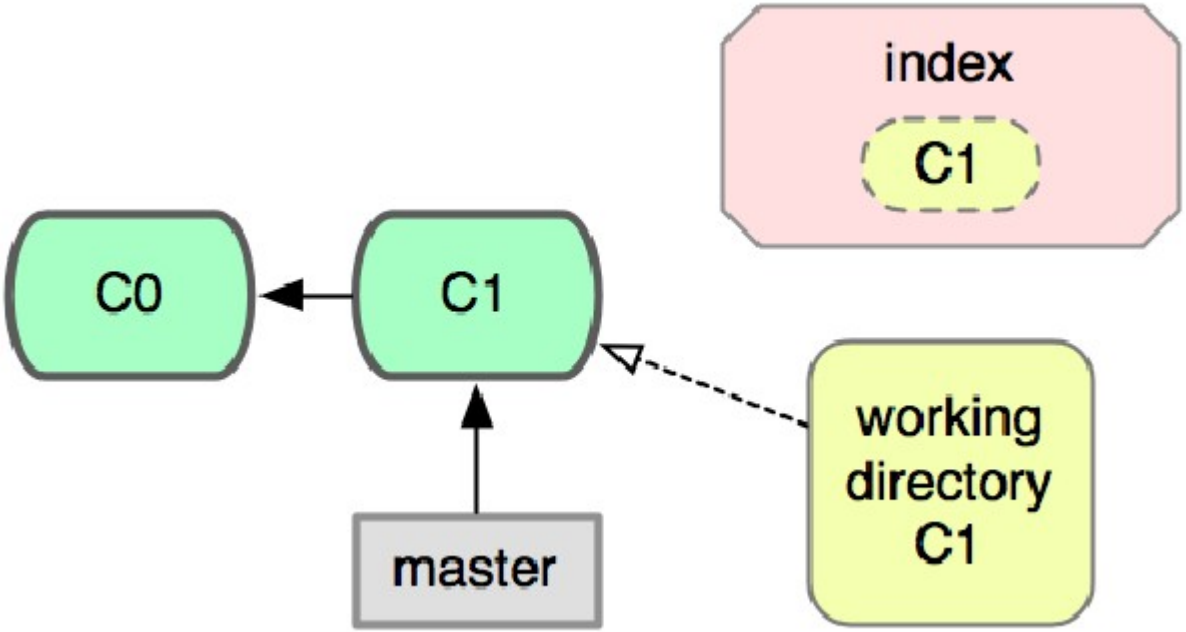
Show all local branch

```
$ git branch
```

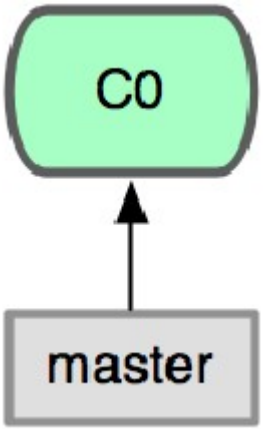
The default branch is named

master

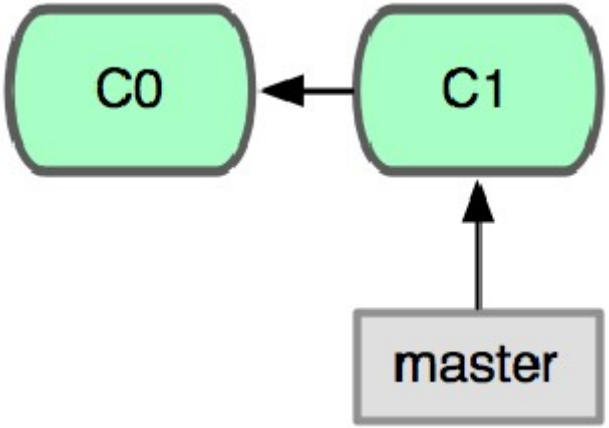
Branches are just pointers to commits



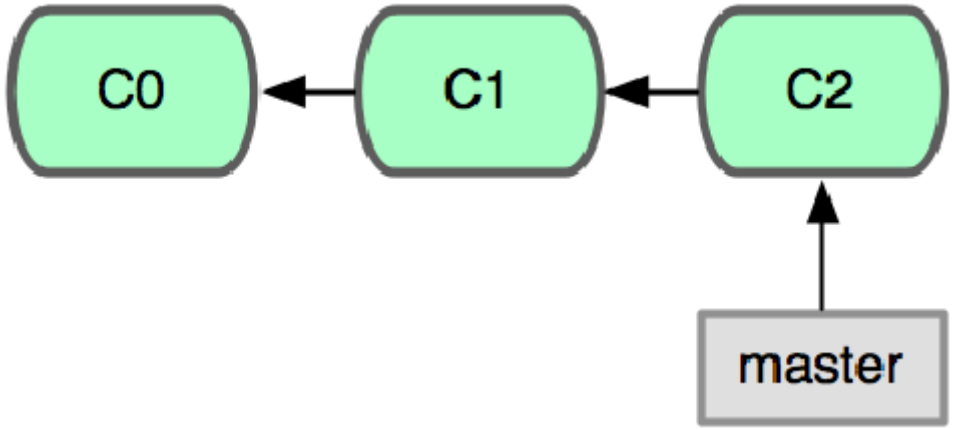
As you commit, the branch moves with you



As you commit, the branch moves with you

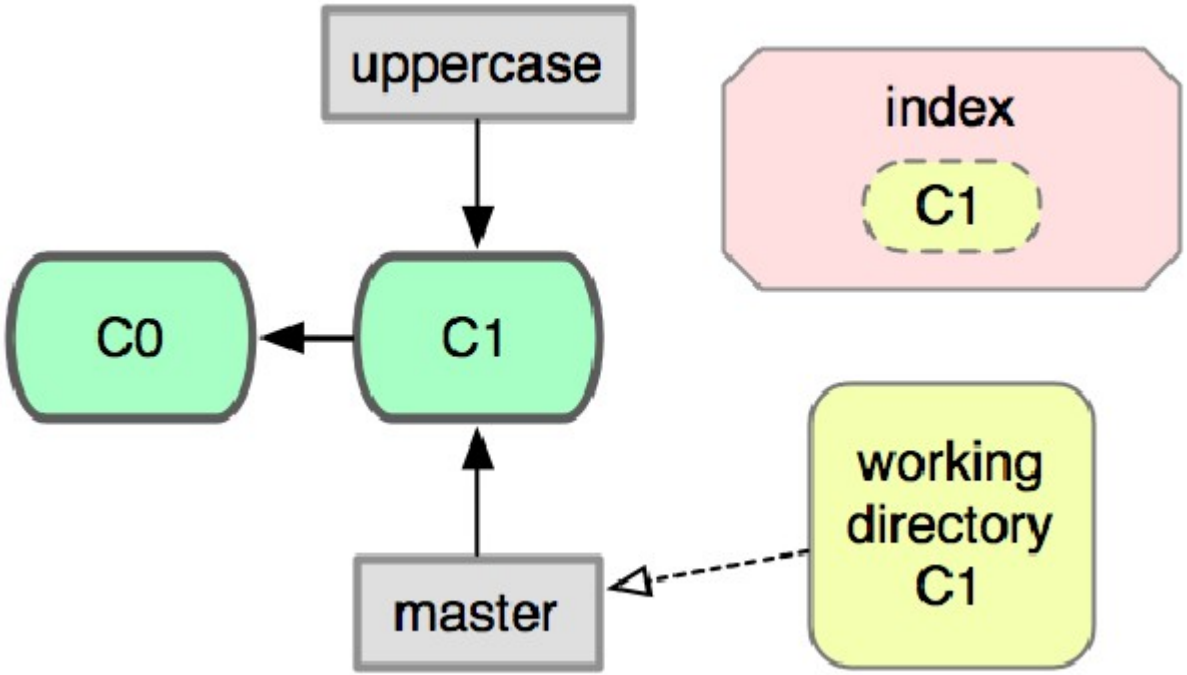


As you commit, the branch moves with you



Create new branch pointing at current commit

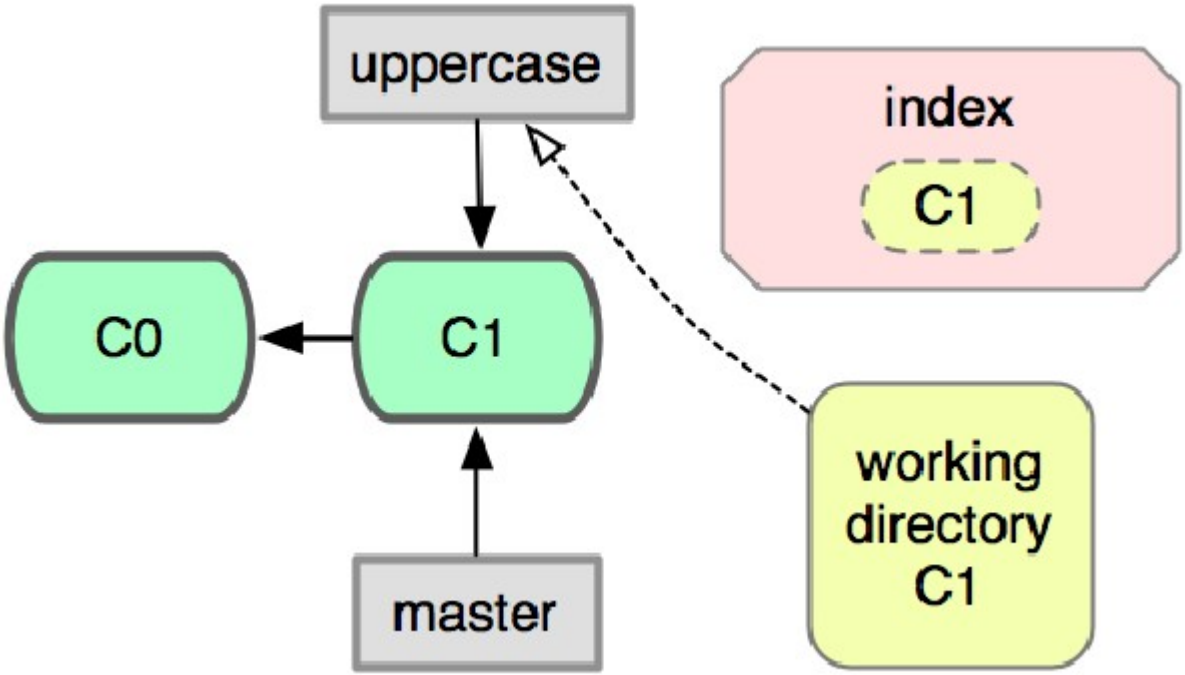
```
$ git branch uppercase
```



Git checkout: switching director to the given branch

```
$ git checkout uppercase
```

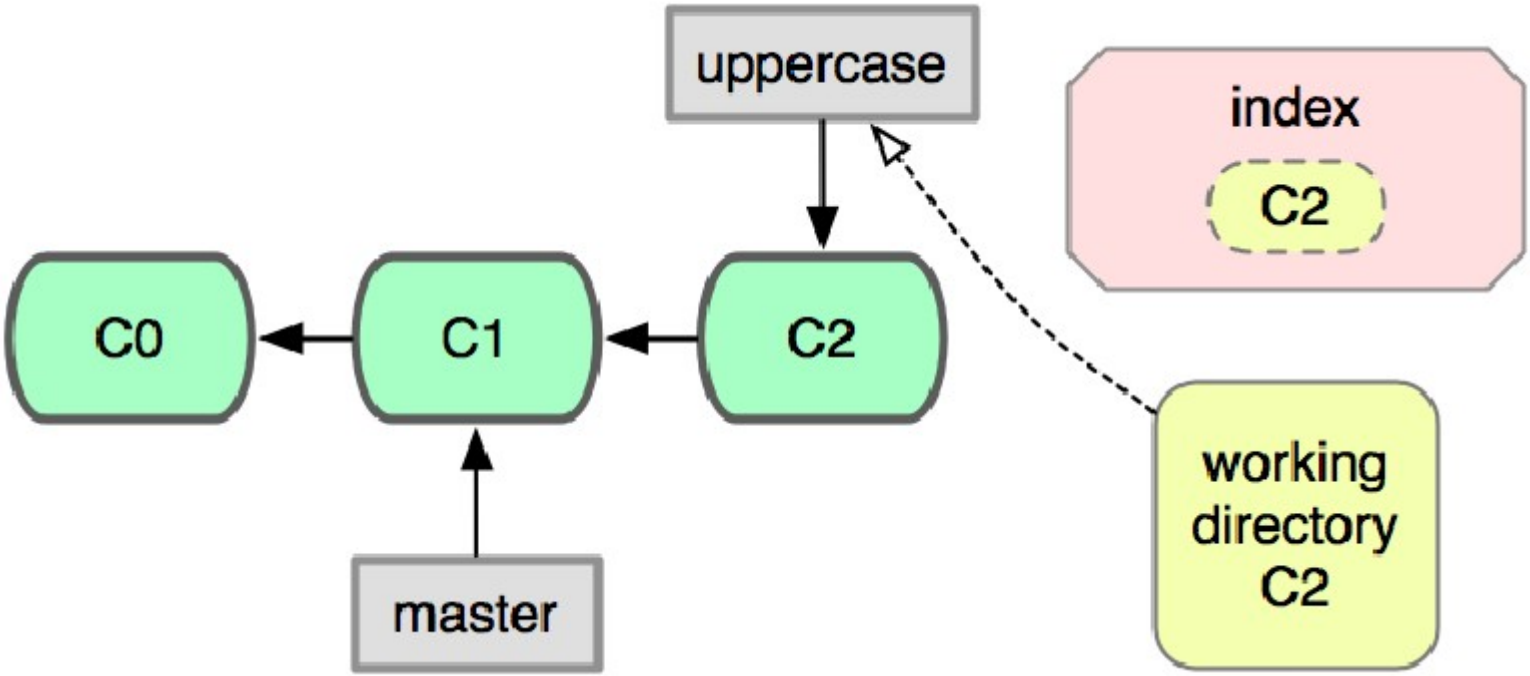
Working dir now corresponds to uppercase



Convert the string to uppercase and commit the change

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

Branches now have diverged



```
$ git branch -v
```

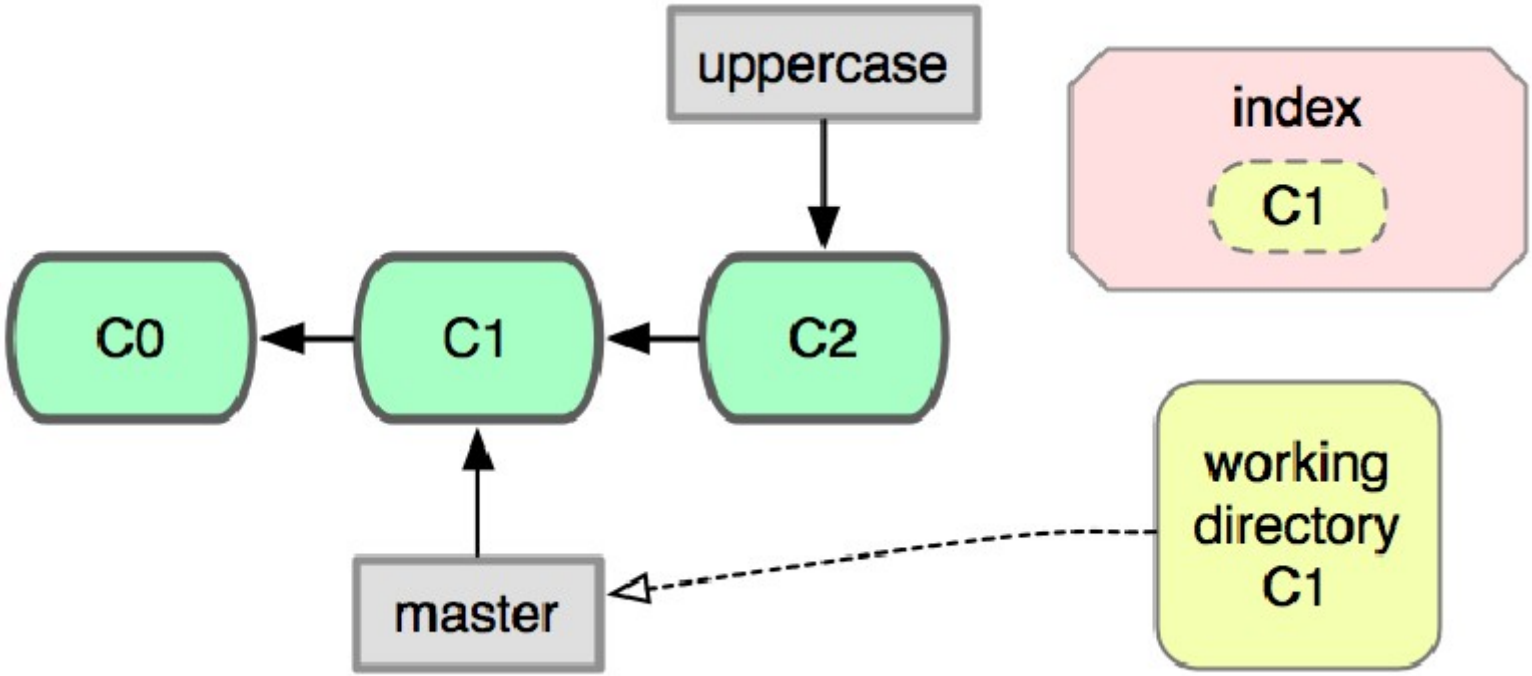
Show branches and the commits they point to

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
```

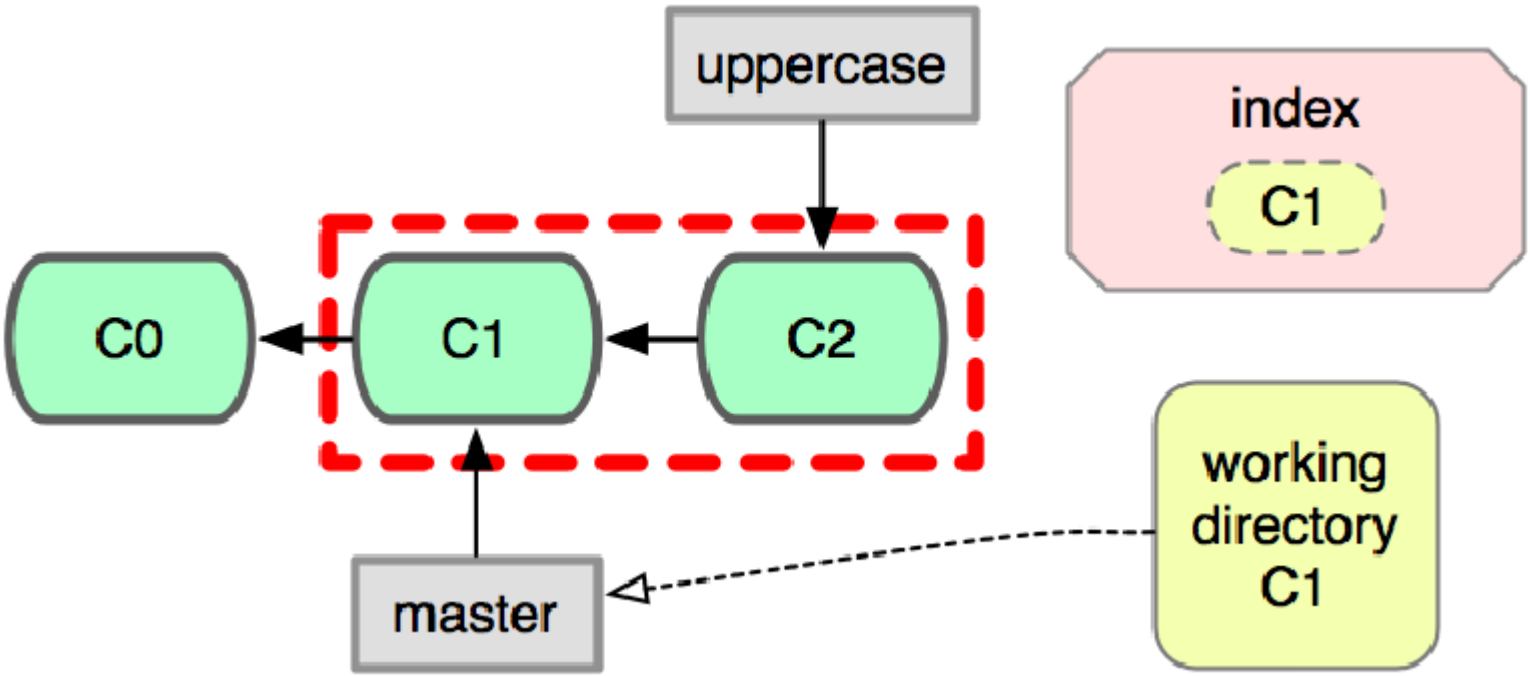
Working dir is now consistent with the master branch



`git diff B1, B2`

Diff between two arbitrary commits

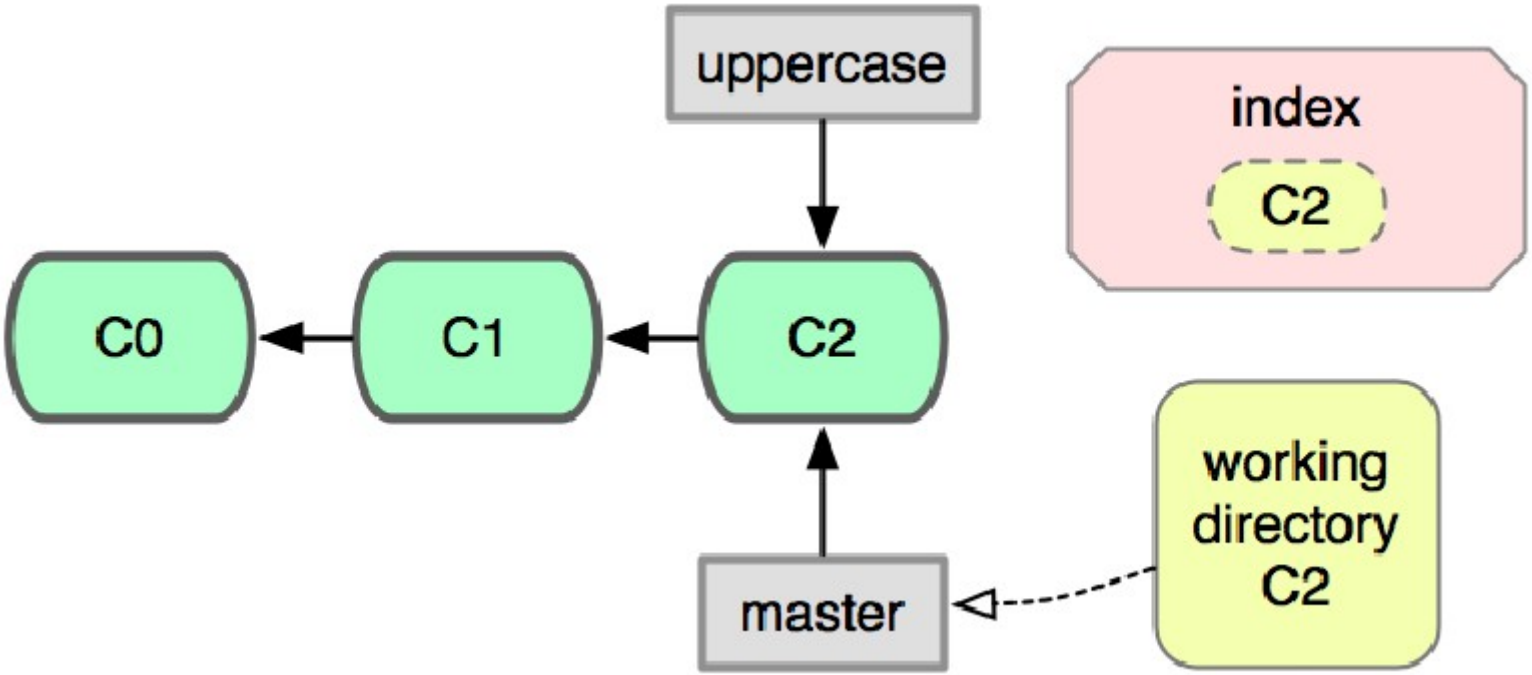
```
$ git diff master uppercase
```



Merge the given commit into current branch

```
$ git merge uppercase
```

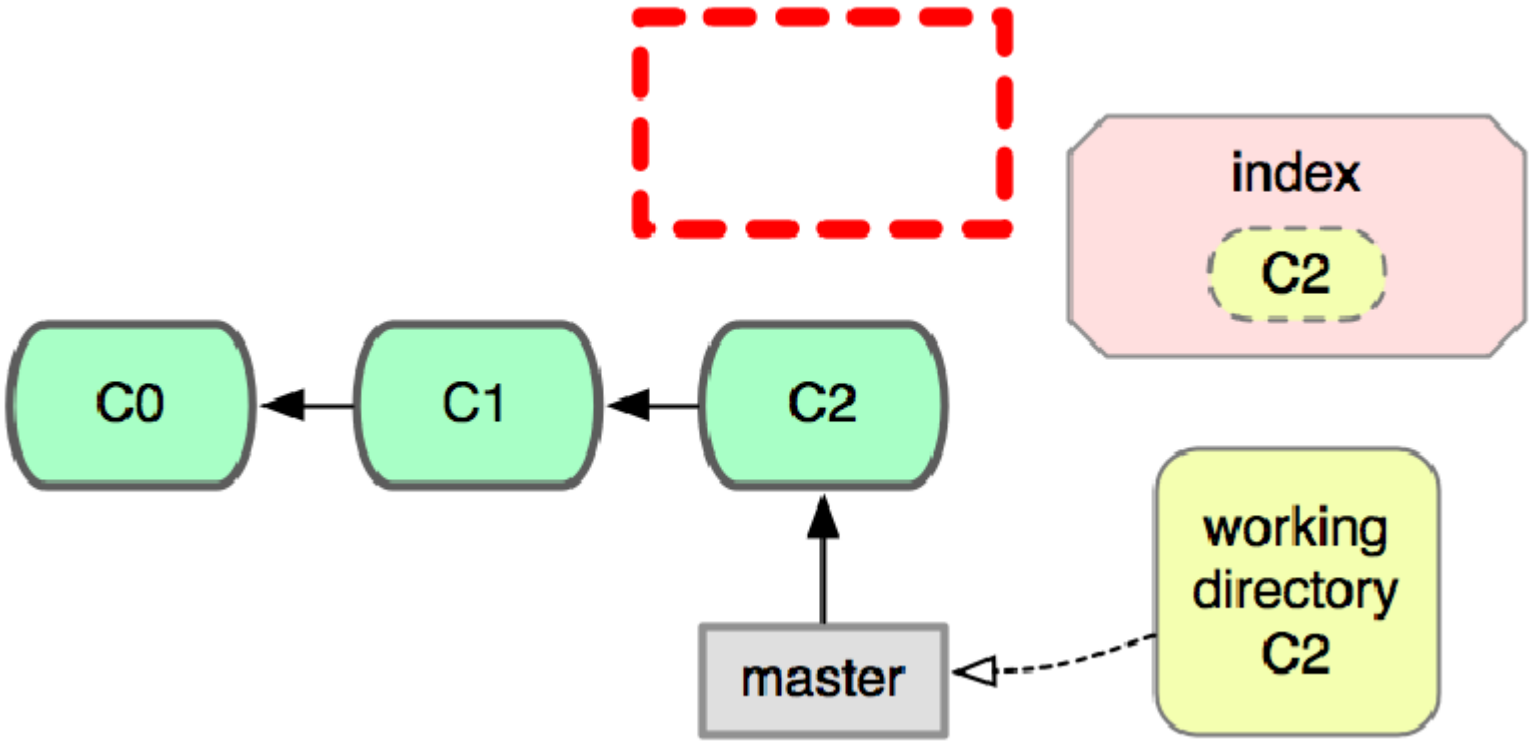
Both branches now point at the same commit



Delete a given branch

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

Only pointer has been deleted



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

Show the commit log with graph structure

The Power of Undo

First, a word about references:

A reference is a way to refer to a commit

Examples:

5c673e53912d86eb771ee0ab0c678ecffa4b939c

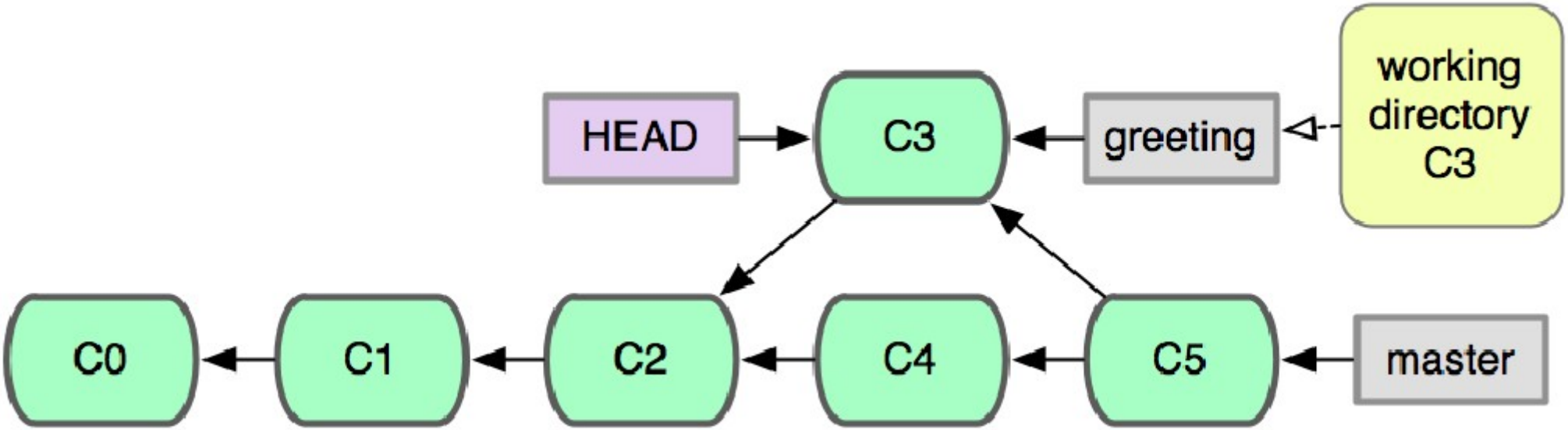
5c673e5

master

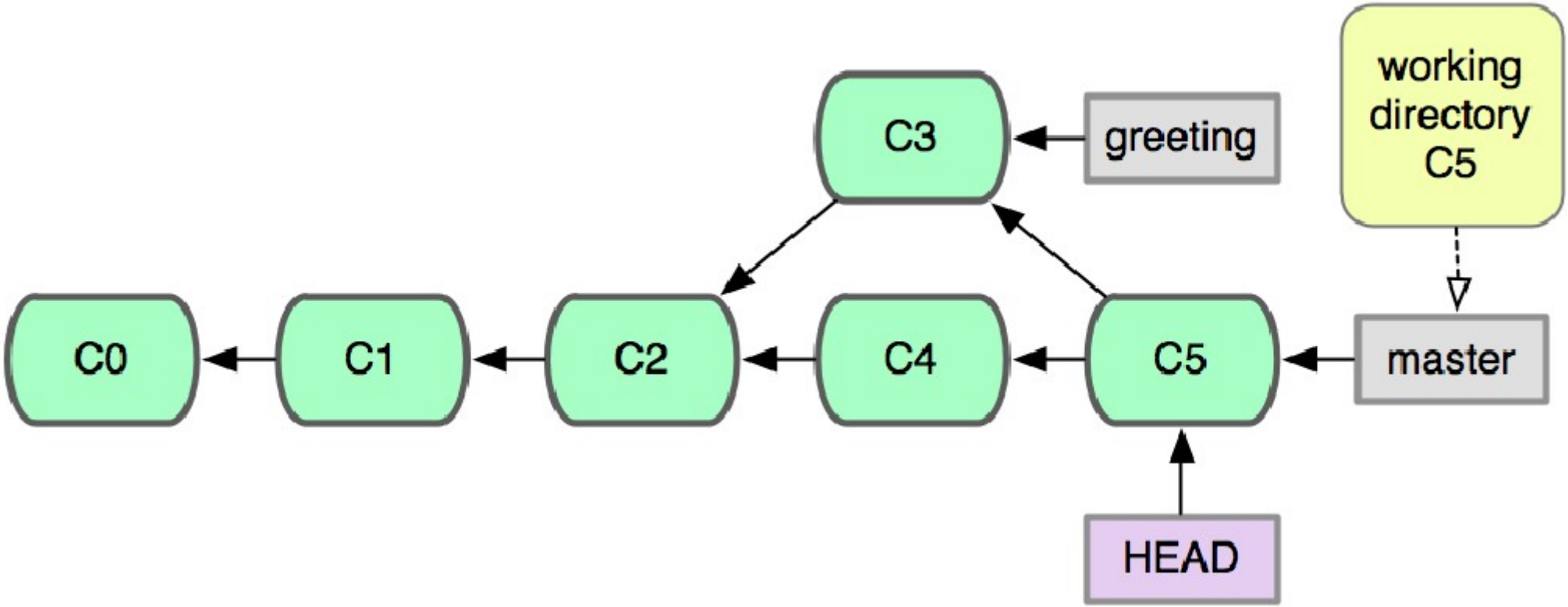
HEAD

HEAD^^

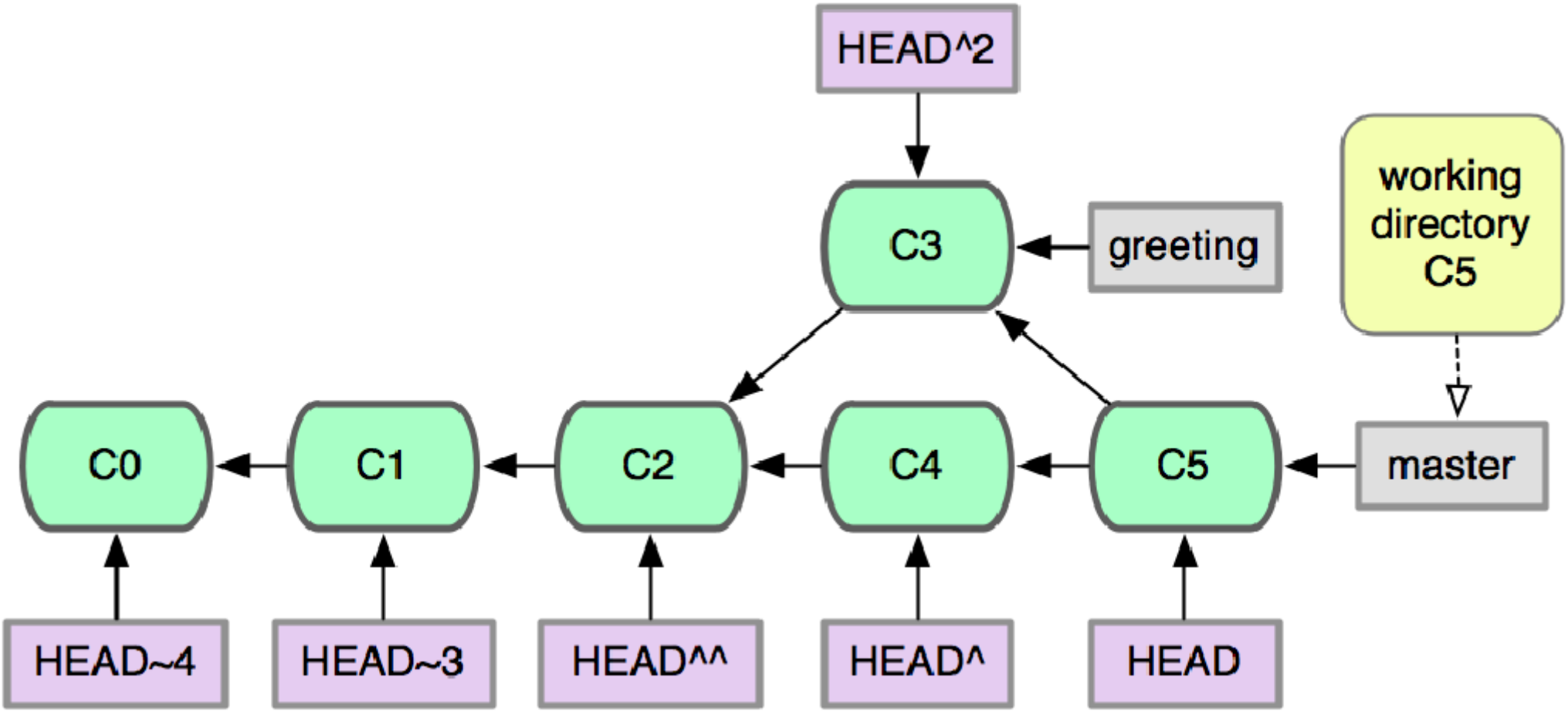
HEAD is a dynamic reference that follows your current checkout



HEAD is a dynamic reference that follows your current checkout



Ancestry reference modifiers



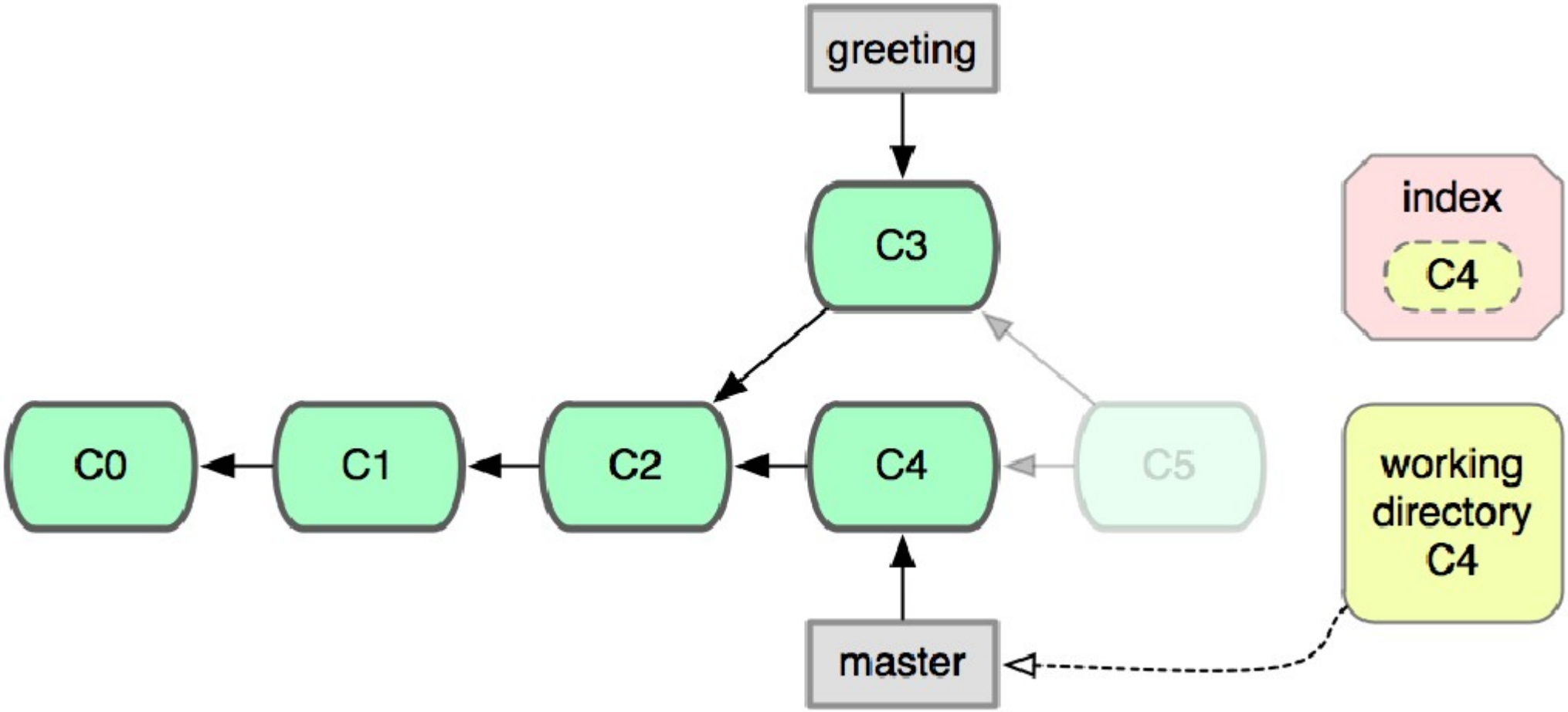
```
git reset --hard
```

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 value of HEAD

```
$ git reflog
```

Oh dear, I should redo my precious code!

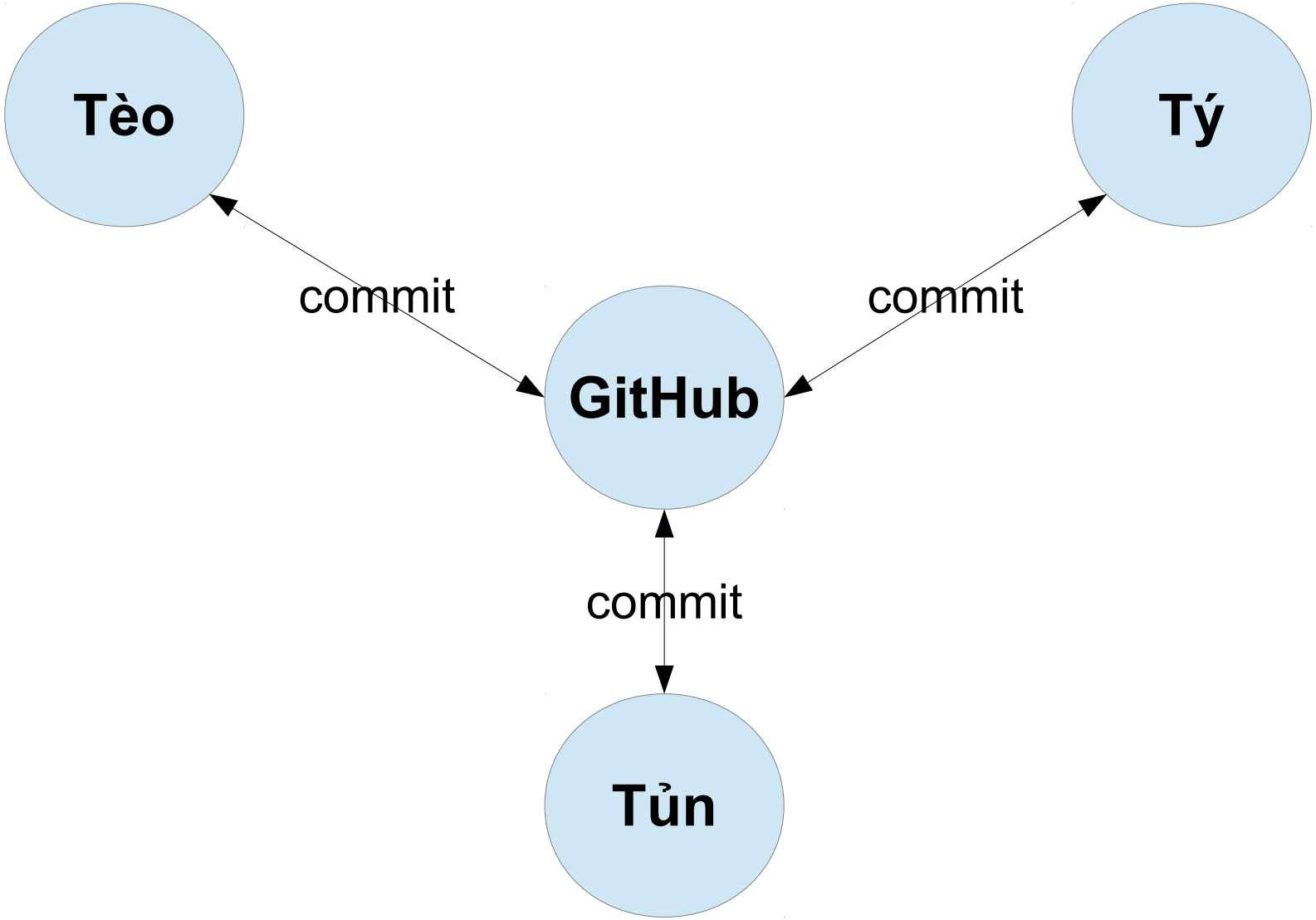
Get commit ref that we move around with reflog

Then … move back in a new branch

```
$ git checkout -b someNewBranchName sha1-of-commit
```

Collaboration

Collaboration in Git is all about moving commits around between repositories



Daily workflow with git

- Check for any remote updates
- Do your work
- Test your work
- Check differences, try to isolate changes
- Commit your work; repeat as needed
- Check for any remote updates
- Push changes, or submit pull request

Translated to git commands

- git pull
  - Checks for new commits in remote repository
- vim, emacs, make, create, magic, etc.
- make test (run your changes!)
- git status
  - See all changed files
- git diff
  - Understand differences line by line (like diff util)
- git add
  - Stage changes, potentially line by line
- git commit -m 'Isolated changes x and y'
- git push

Dealing with collaboration

- Branching – Cheap and effective
  - Make development or feature branches
  - Rebase and merge when features complete
  - `git branch https`
  - `git checkout https`
  - …
- Pull/Push with Merge
  - Standard model: pull requests
  - Or push into “central” repository

Merging in Practice

```
> git pull
Auto-merging test
CONFLICT (content): Merge conflict in test
Automatic merge failed; fix conflicts and then commit the result.
> cat test
<<<<<<< HEAD
helloX world
=======
helloY world
>>>>>>> 29a240d5017c73ca4f78466afcf1fd5b8f46808f
```

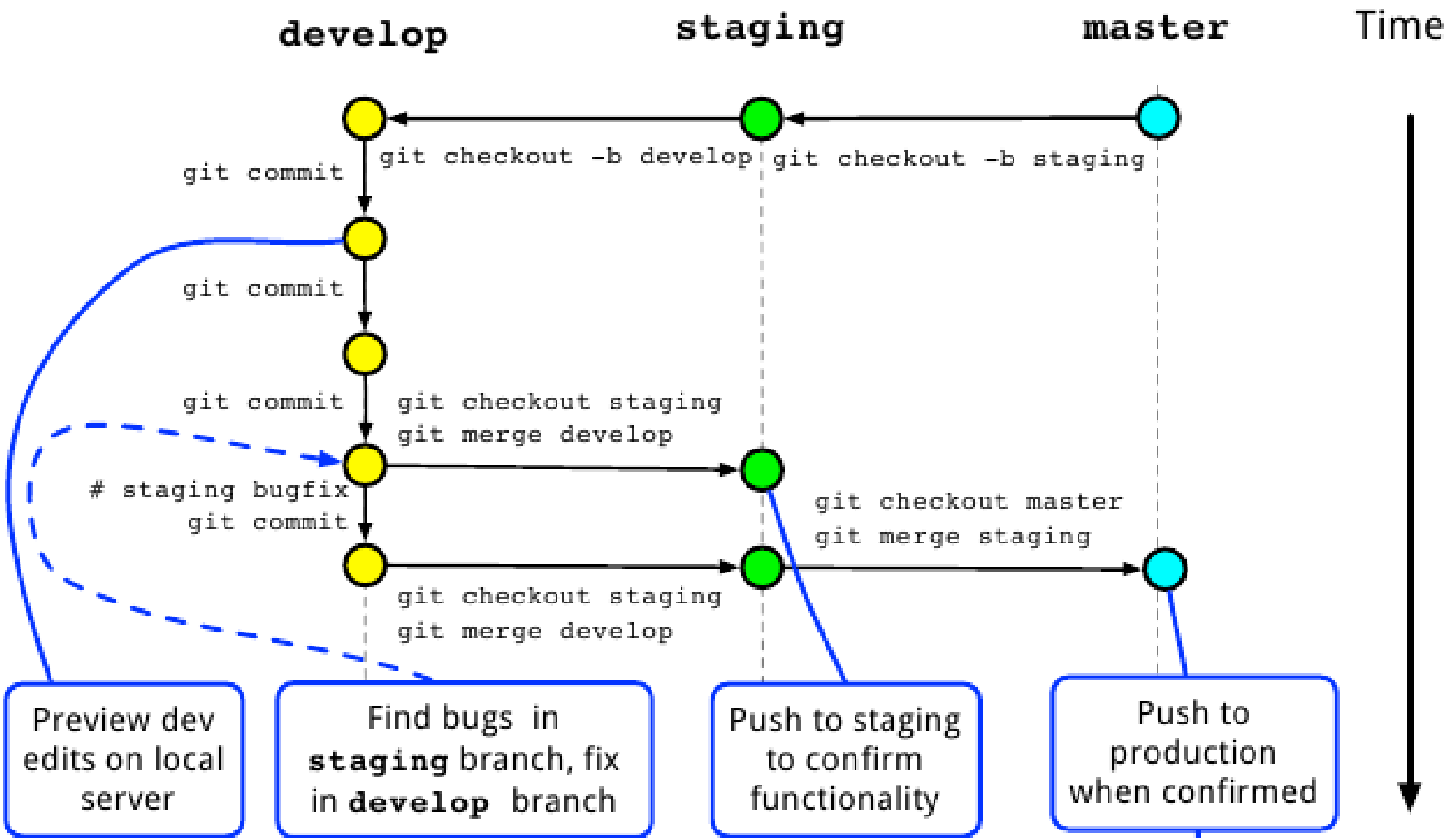
Choose how to merge—yours or other author's.
Finalize, commit, then push, or request a pull.

Git Branching Model: Dev, Staging, Production (*)

- You will do primary edits of code in the **develop branch**.
- When you have a release candidate, git merge these edits into the **staging branch** and push to the staging server to preview
- Finally, when that looks good, we git merge the changes into the **master branch** and push to the production server.

(*) Balaji S. Srinivasan, Startup Engineering, Stanford University
See nvie.com for a more complex one.

Git Branching Model:
Dev, Staging, Production



git Resources

1) man gittutorial

2) Git User's Manual
http://www.kernel.org/pub/software/scm/git/docs/user-manual.html

3) Git Immersion
http://gitimmersion.com/

4) Git Cheatsheet
https://github.com/AlexZeitler/gitcheatsheet

Makefile

Code Structure: GNU make

- Recipes for your code
  - Compilation
  - Installation
  - Cleanup
  - Testing
- Composed of a series of
  - targets [the recipes]
  - which have dependencies
  - and commands
  - also, variables...

Makefile for your first Exercise

```
/* makefile file */

SERVER_SRC = no1_server.c
CLIENT_SRC = no1_client.c
OPTIONS = -Wall

default: no1-server no1_client

no1_server:
   @gcc $(SERVER_SRC) -o no1_server $(OPTIONS)

no1_client:
   @gcc $(CLIENT_SRC) -o no1_client $(OPTIONS)

clean:
   @rm no1_server no1_client *.o
```

Makefile Resources

A short makefile tutorial

http://www.cs.umd.edu/class/spring2002/cmsc214/Tutorial/makefile.html

And an overkill GNU Makefile manual

http://www.gnu.org/software/make/manual/make.html