

# Fundamentals of Git

PPT By Zachary Ling

CUHK

# Outline

- History of Git
- Distributed V.S Centralized Version Control
- Getting started
- Branching and Merging
- Working with remote
- Summary

# A Brief History of Git

- Linus uses BitKeeper to manage Linux code
- Ran into BitKeeper licensing issue
  - Liked functionality
  - Looked at CVS as how not to do things
- April 5, 2005 - Linus sends out email showing first version
- June 15, 2005 - Git used for Linux version control

# Git is Not an SCM

*Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable filesystem, used to track directory trees.*

*Linus Torvalds, 7 Apr 2005*

<http://lkml.org/lkml/2005/4/8/9>



# Centralized Version Control

- Traditional version control system
  - Server with database
  - Clients have a working version
- Examples
  - CVS
  - Subversion
  - Visual Source Safe
- Challenges
  - Multi-developer conflicts
  - Client/server communication

# Distributed Version Control

- Authoritative server by convention only
- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial
- Other distributed systems include
  - Mercurial
  - BitKeeper
  - Darcs
  - Bazaar

mercurial (hg)

bazaar

subversion (svn)

# version control

concurrent version system (cvs)

perforce

visual source safe





mercurial (hg)

“not bad”

SoS

bazaar

subversion (svn)

# version control

concurrent version system (cvs)

perforce

visual source safe





mercurial (hg)

bazaar

subversion (svn)

# version control

concurrent version system (cvs)

perforce

visual source safe “kill self”



**git**



mercurial (hg)

bazaar

subversion (svn)

# version control

concurrent version system (cvs)

perforce

visual source safe

# Git Advantages

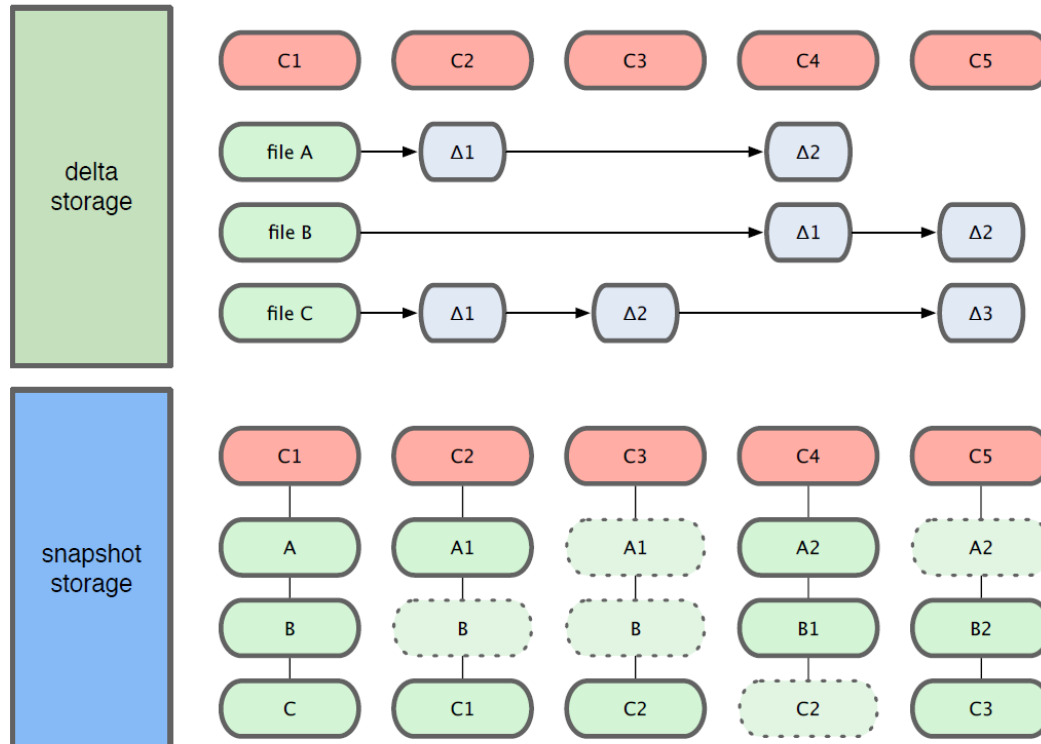
- Resilience
  - No one repository has more data than any other
- Speed
  - Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)
- Space
  - Compression can be done across repository not just per file
  - Minimizes local size as well as push/pull data transfers
- Simplicity
  - Object model is very simple
- Large userbase with robust tools

# Some GIT Disadvantages

- Definite learning curve, especially for those used to centralized systems
  - Can sometimes seem overwhelming to learn
    - Conceptual difference
    - Huge amount of commends

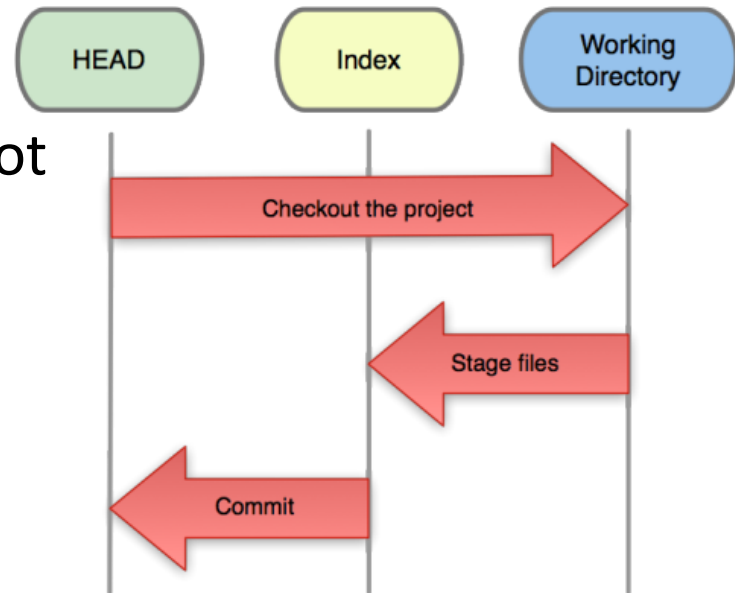
# Getting Started

- Git use snapshot storage



# Getting Started

- Three trees of Git
  - The HEAD
    - last commit snapshot, next parent
  - Index
    - Proposed next commit snapshot
  - Working directory
    - Sandbox



# Getting Started

- A basic workflow
  - (Possible init or clone) Init a repo
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

# Getting Started

- Init a repository
- Git init

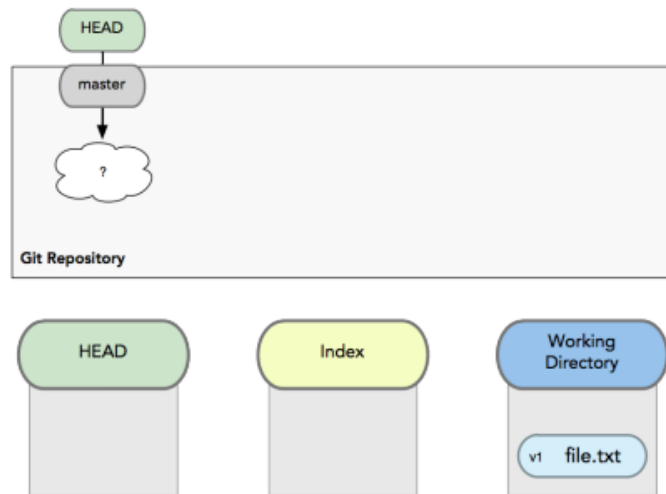
```
zachary@zachary-desktop:~/code/gitdemo$ git init
Initialized empty Git repository in /home/zachary/code/gitdemo/.git/
```

```
zachary@zachary-desktop:~/code/gitdemo$ ls -l .git/
total 32
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 branches
-rw-r--r-- 1 zachary zachary  92 2011-08-28 14:51 config
-rw-r--r-- 1 zachary zachary  73 2011-08-28 14:51 description
-rw-r--r-- 1 zachary zachary  23 2011-08-28 14:51 HEAD
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 hooks
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 info
drwxr-xr-x 4 zachary zachary 4096 2011-08-28 14:51 objects
drwxr-xr-x 4 zachary zachary 4096 2011-08-28 14:51 refs
```



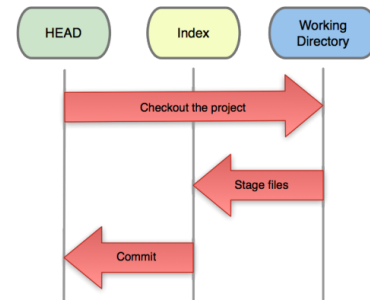
# Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes
- Use your favorite editor



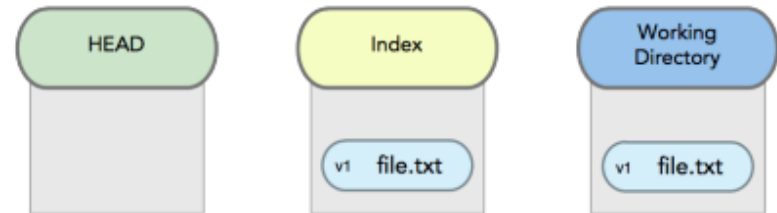
The screenshot shows a terminal window titled 'zachary@zachary-desktop: ~/code/gitdemo'. The terminal displays the contents of a file named 'hello.txt', which contains three lines: 'first line', 'second line', and 'third line'. The terminal prompt is '<y/code/gitdemo Line: 1/3 Git Branch: master <: master'. The terminal window also shows a file explorer view on the right side, displaying the file 'hello.txt'.

# Getting Started



- A basic workflow
  - Edit files
  - **Stage the changes**
  - Review your changes
  - Commit the changes

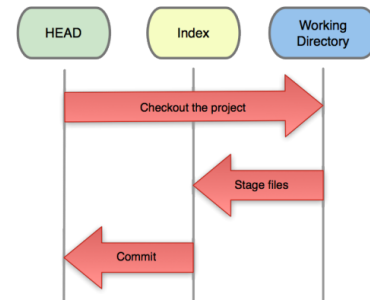
- Git add filename



**git add**

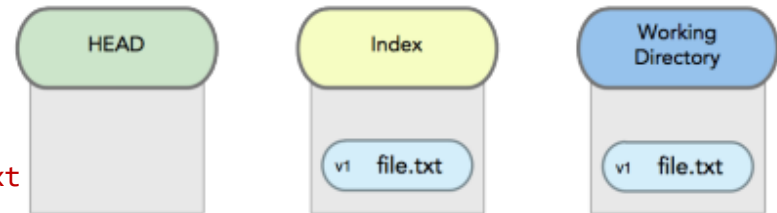
```
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

# Getting Started



- A basic workflow
  - Edit files
  - Stage the changes
  - **Review your changes**
  - Commit the changes

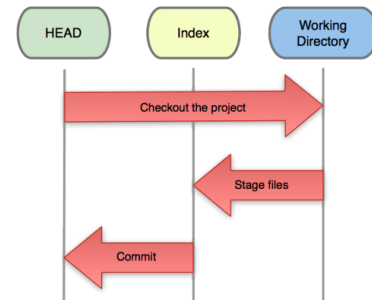
- Git status



**git add**

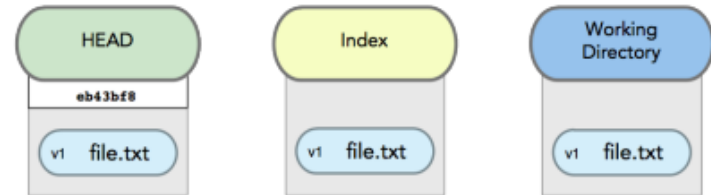
```
zachary@zachary-desktop:~/code/gitdemo$ git add hello.txt
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   hello.txt
#
```

# Getting Started



- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - **Commit the changes**

- Git commit

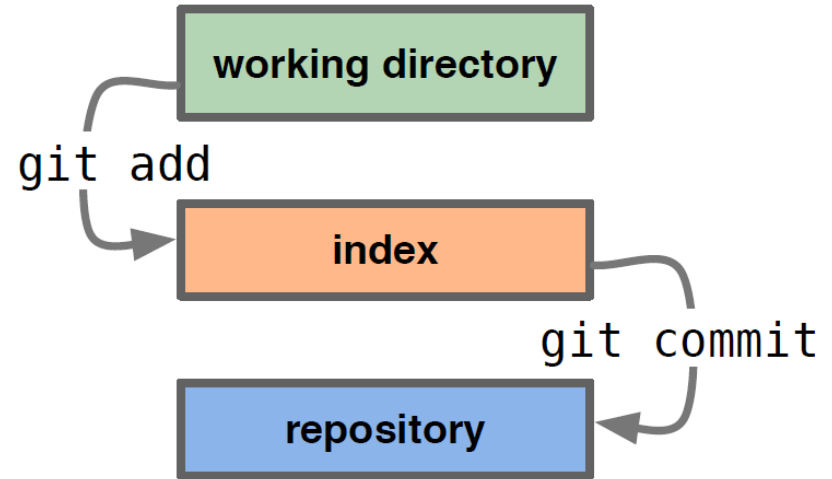


**git commit**

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.txt
#
```

# Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes



# Getting Started

- View changes
- Git diff
  - Show the difference between **working directory** and **staged**
- Git diff --cached
  - Show the difference between **staged** and **the HEAD**

- View history
- Git log

```
zachary@zachary-desktop:~/code/gitdemo$ git log
commit efb3aeae66029474e28273536a8f52969d705d04
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 15:02:08 2011 +0800
```

Add second line

```
commit 453914143eae3fc5a57b9504343e2595365a7357
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 14:59:13 2011 +0800
```

Initial commit

# Getting Started

- Revert changes (Get back to a previous version)
  - Git checkout `commit_hash`

```
zachary@zachary-desktop:~/code/gitdemo$ git log
commit efb3aeae66029474e28273536a8f52969d705d04
Author: Zachary Ling <zacling@gmail.com>
Date: Sun Aug 28 15:02:08 2011 +0800
```

Add second line

```
commit 453914143eae3fc5a57b9504343e2595365a7357
Author: Zachary Ling <zacling@gmail.com>
Date: Sun Aug 28 14:59:13 2011 +0800
```

Initial commit

```
zachary@zachary-desktop:~/code/gitdemo$ git checkout 4539
Note: checking out '4539'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

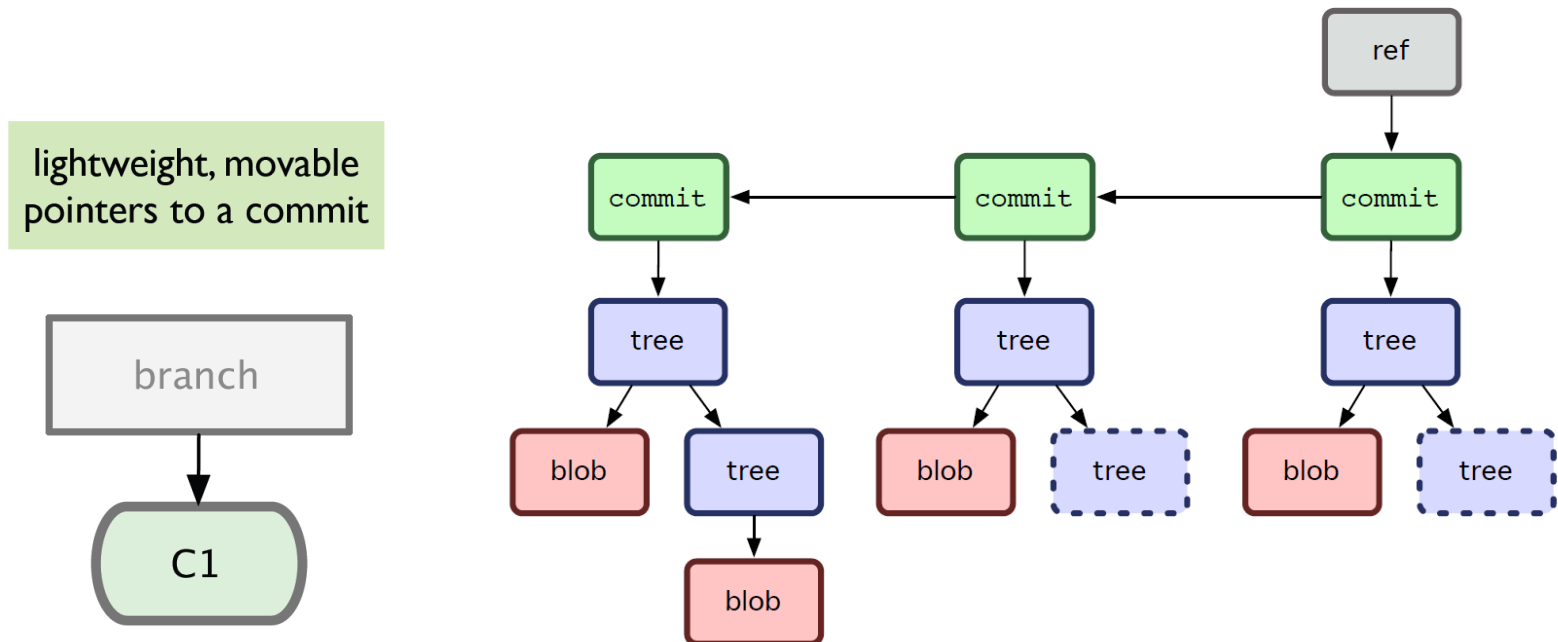
If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b new_branch_name
```

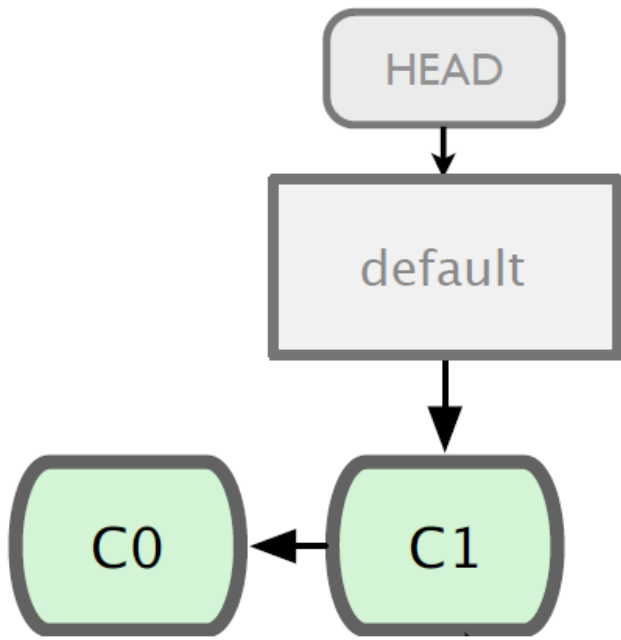
```
HEAD is now at 4539141... Initial commit
```

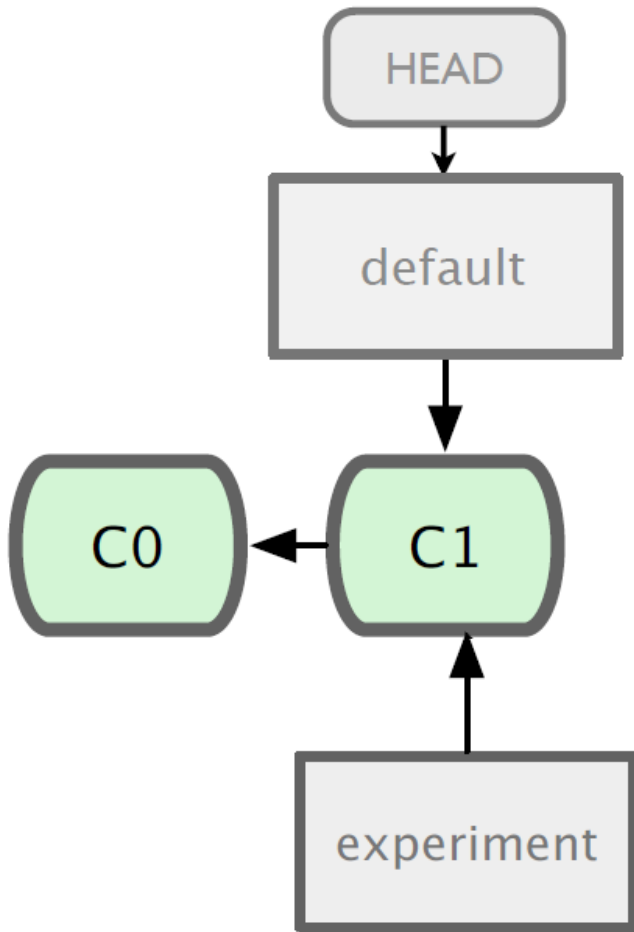
# Branching

- Git sees commit this way...
- Branch annotates which commit we are working on

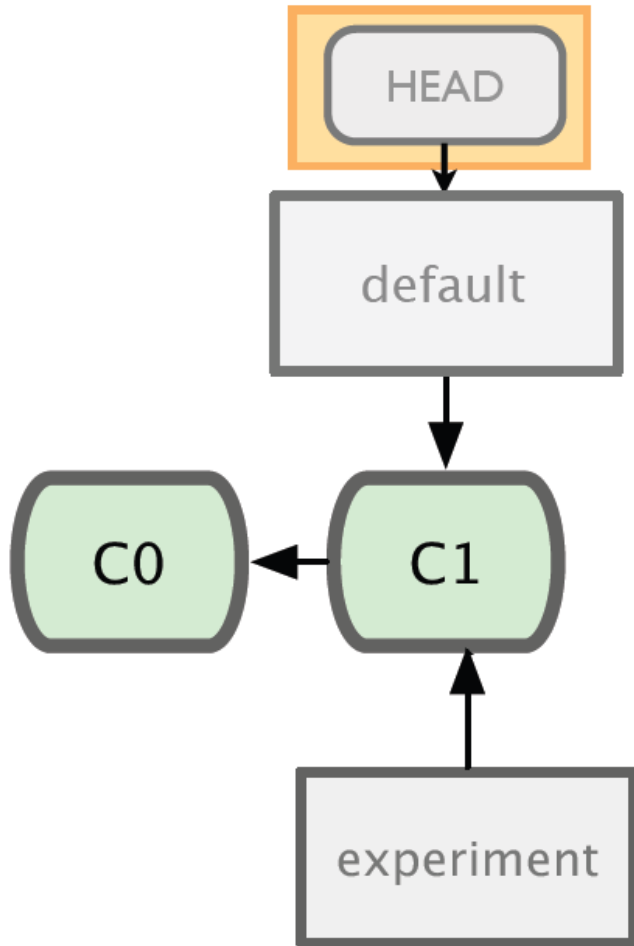




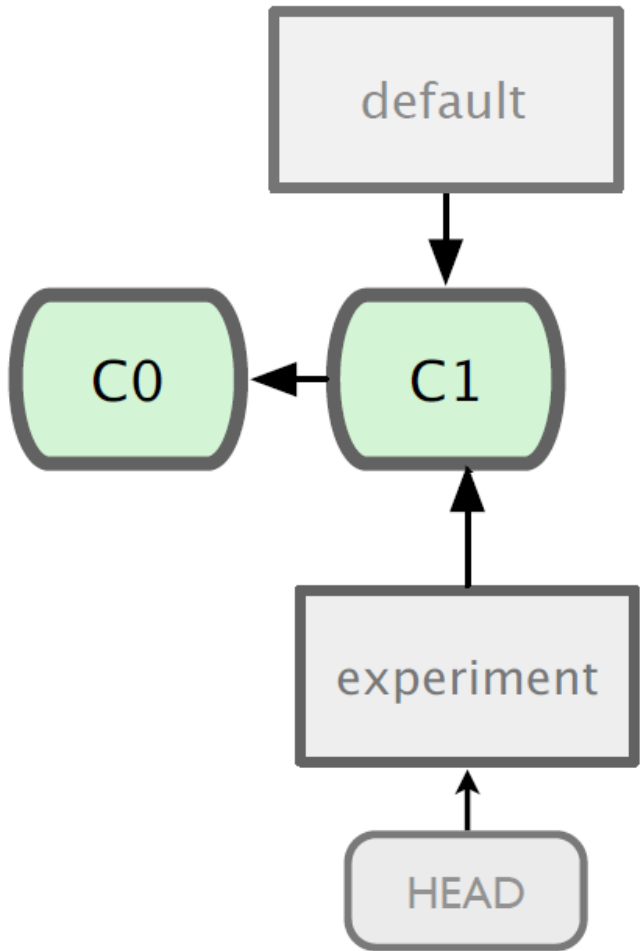




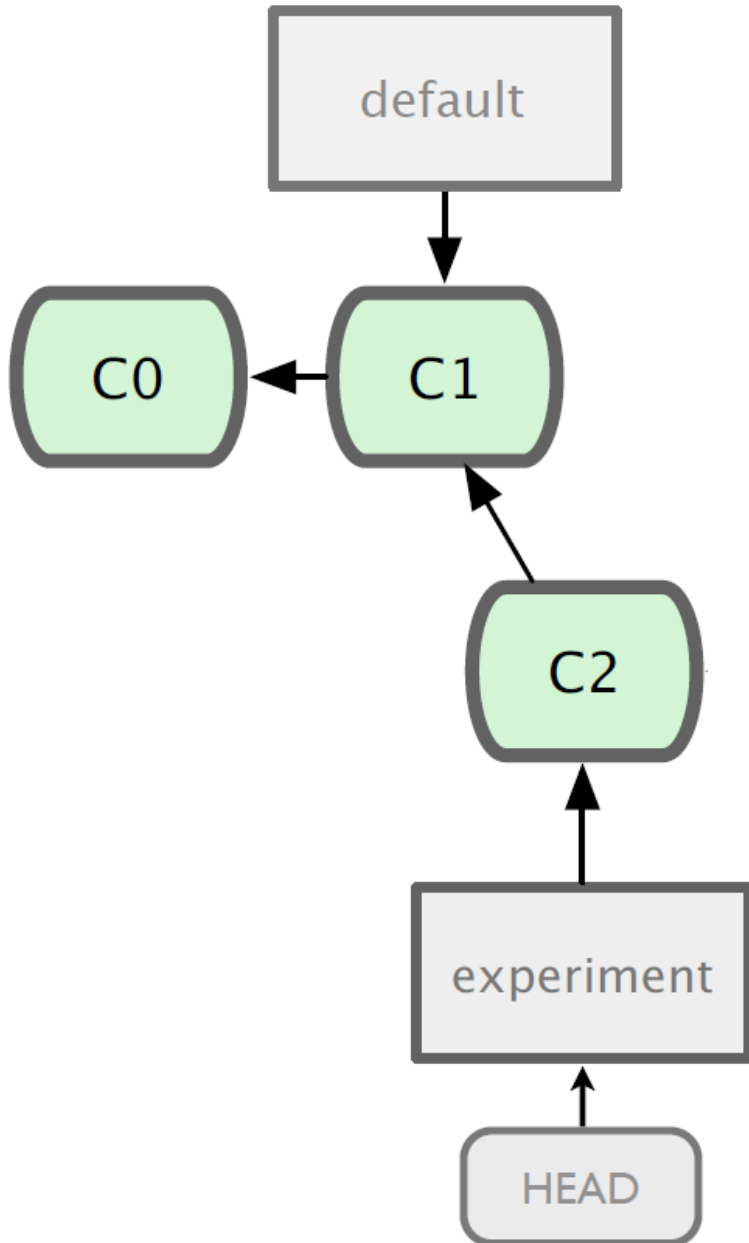
git branch experiment



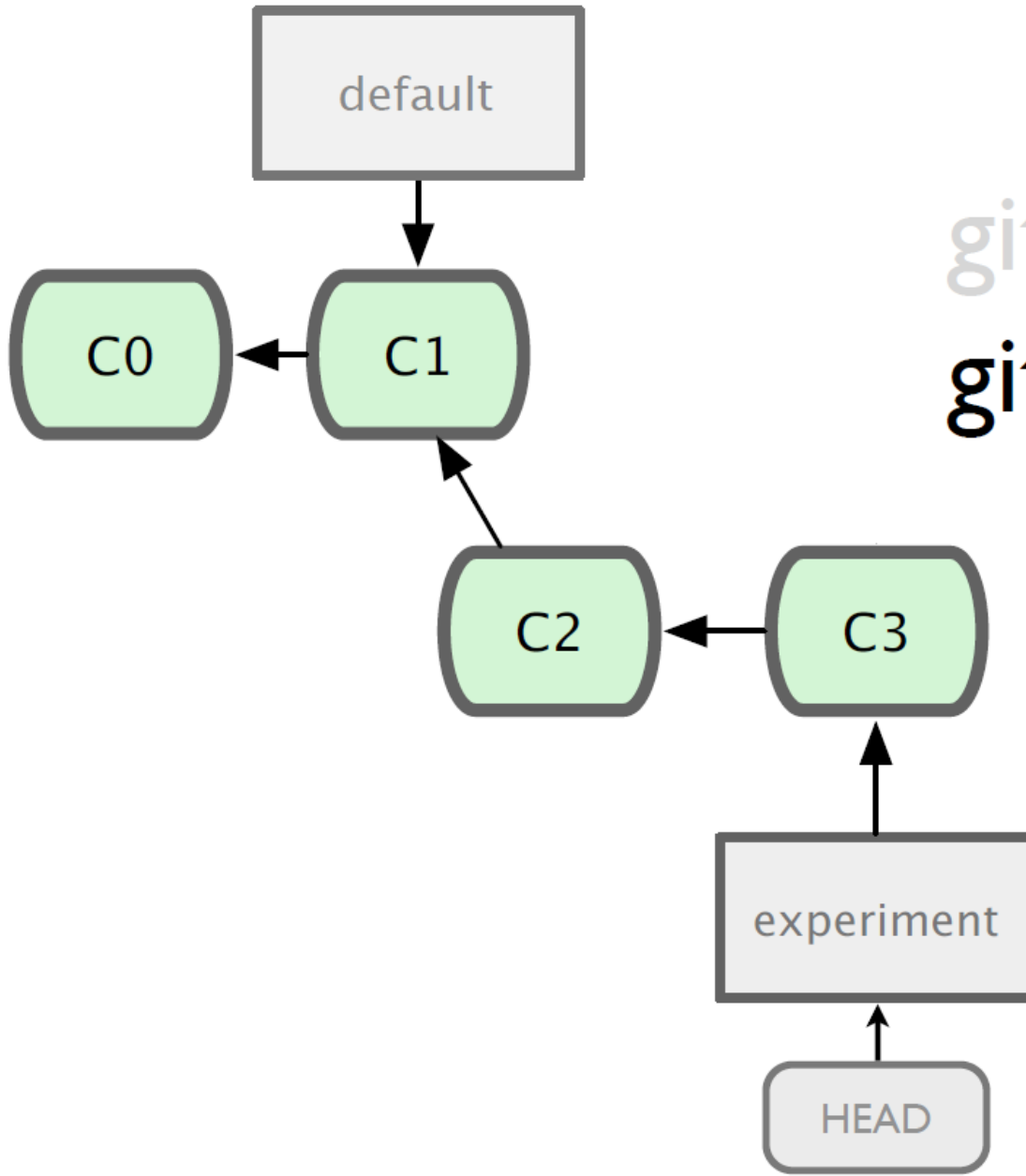
```
$ git branch  
* default  
experiment
```



git checkout experiment

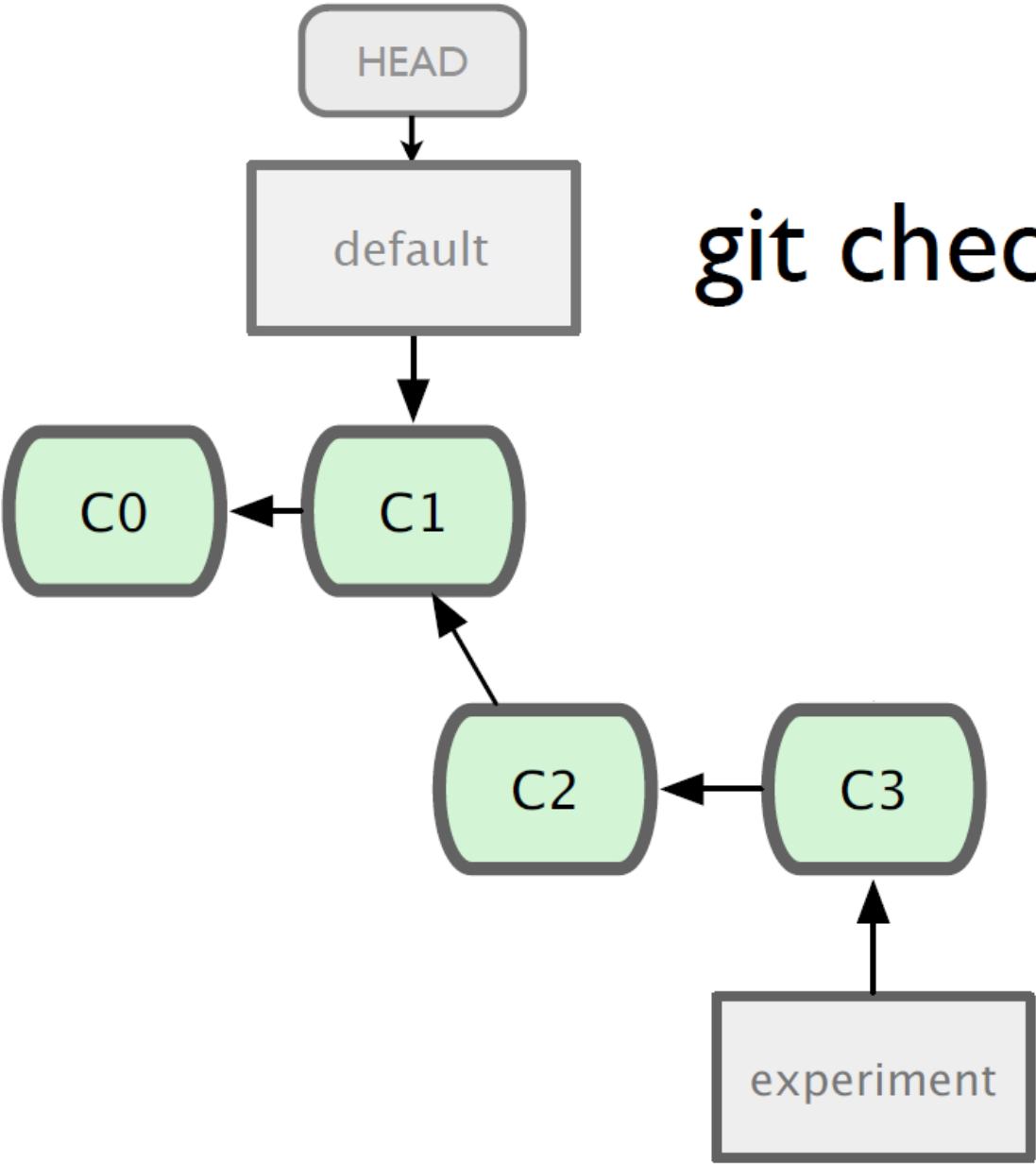


`git commit`

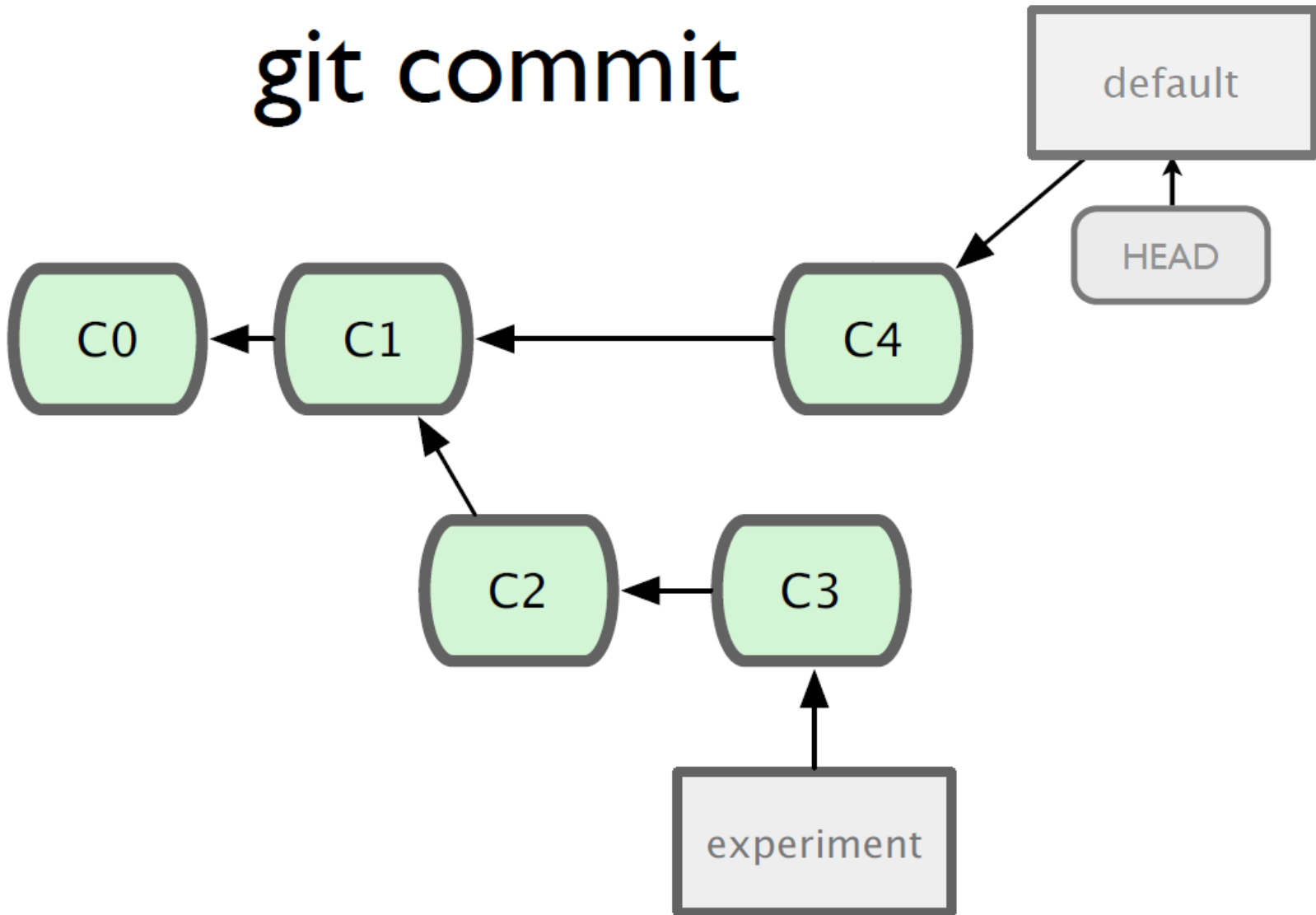


git commit  
git commit

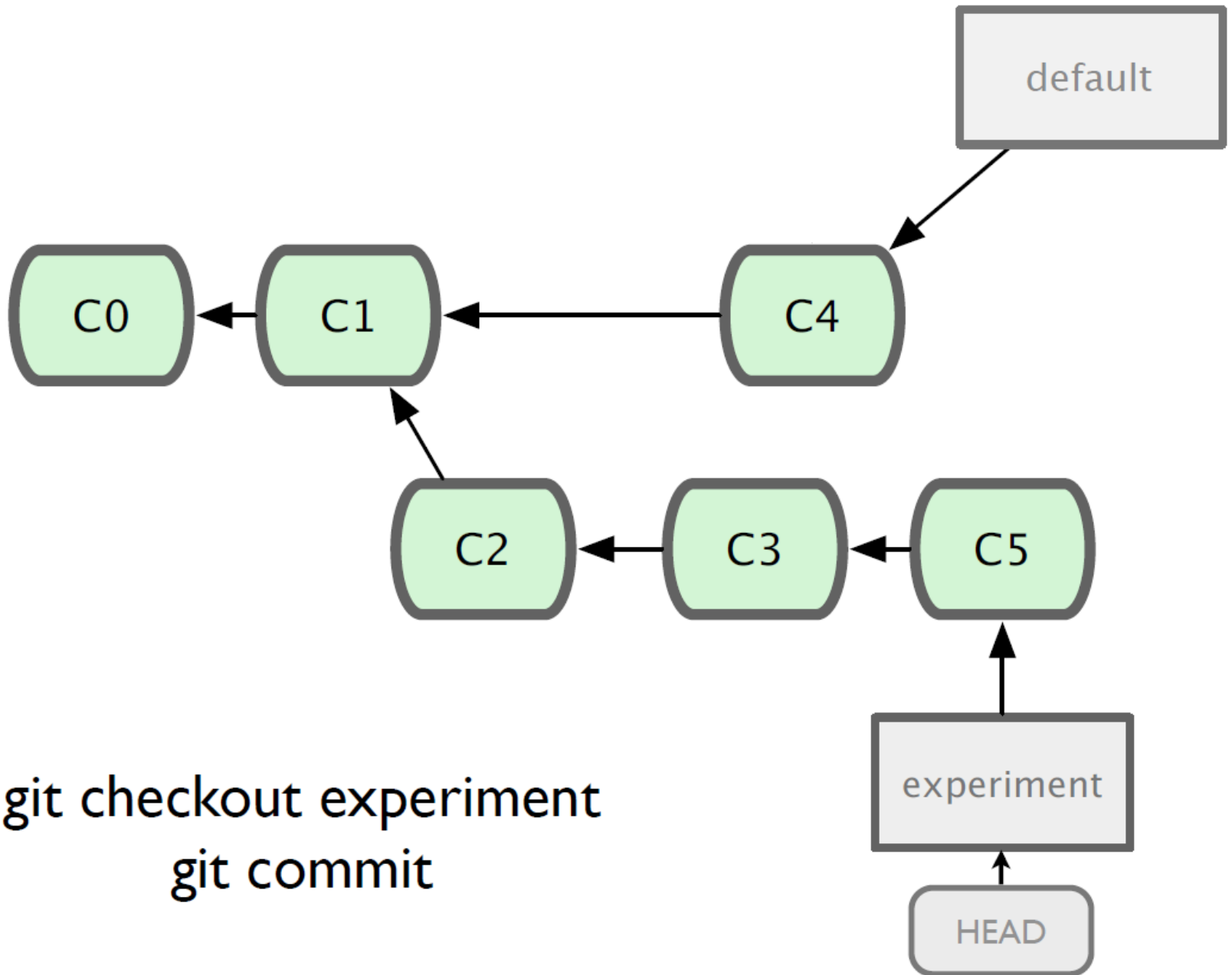
# git checkout default



# git commit

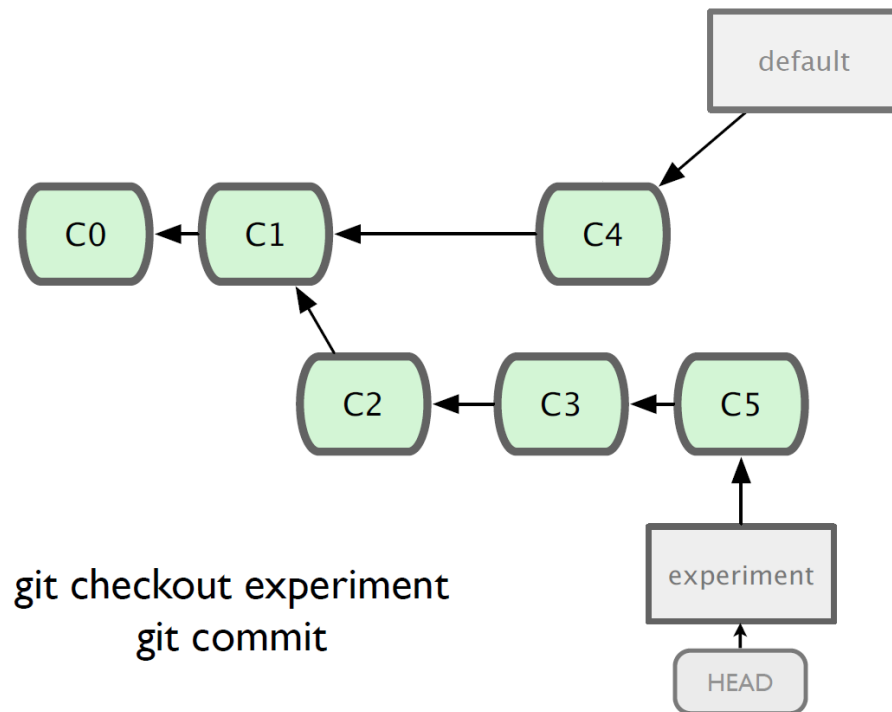






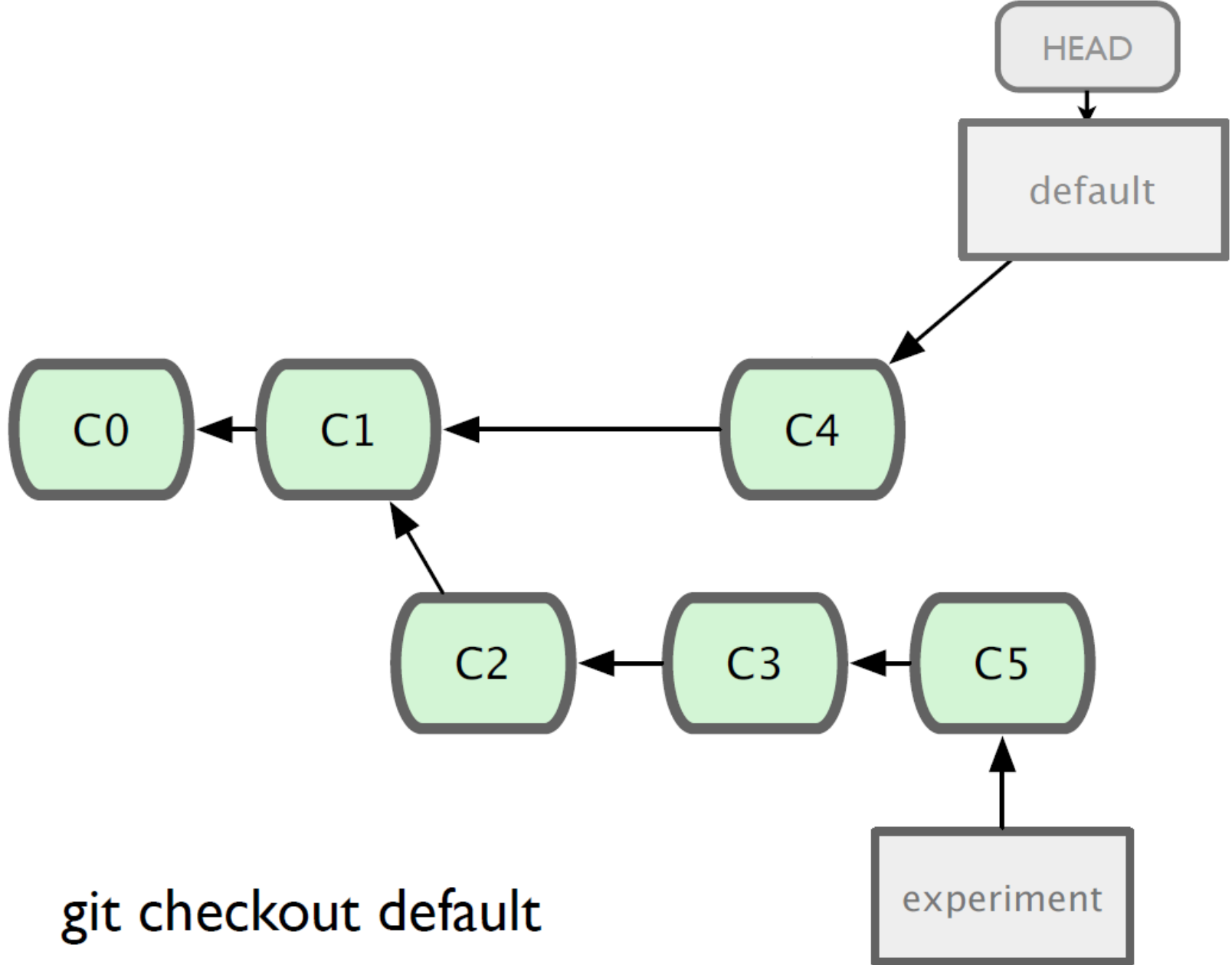
# Merging

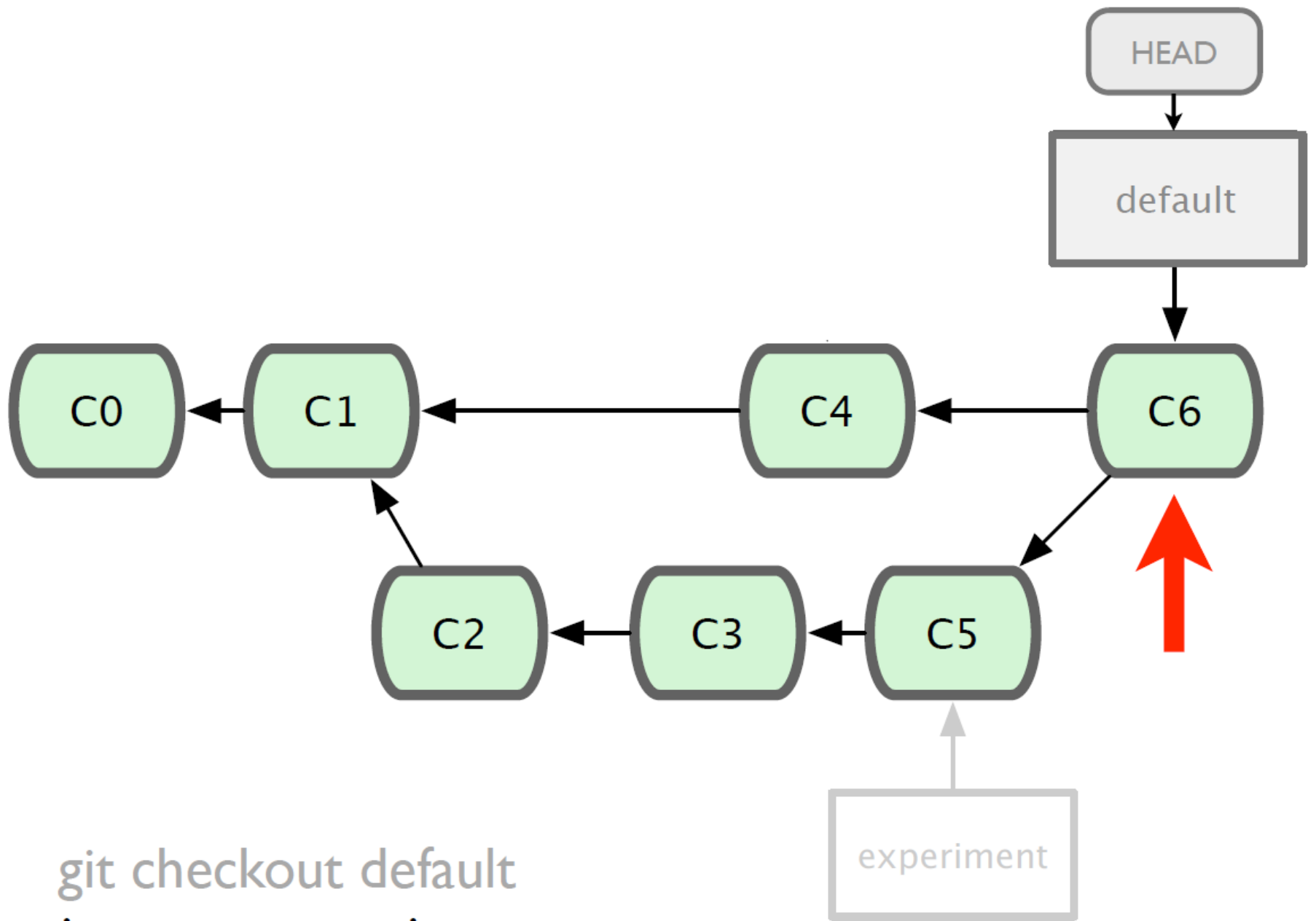
- What do we do with this mess?
  - Merge them



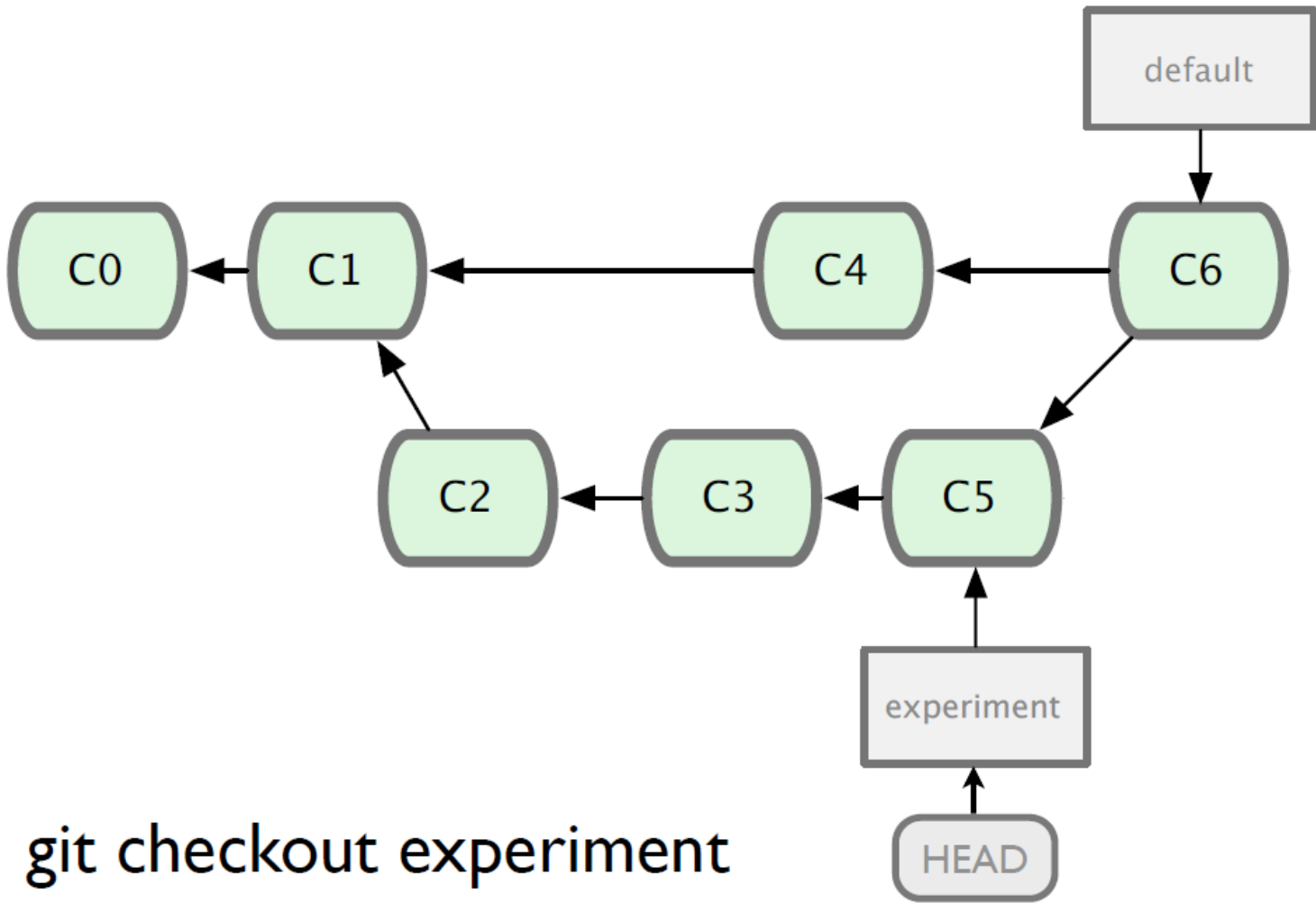
# Merging

- Steps to merge two branch
  - Checkout the branch you want to merge **onto**
  - Merge the branch you want to merge

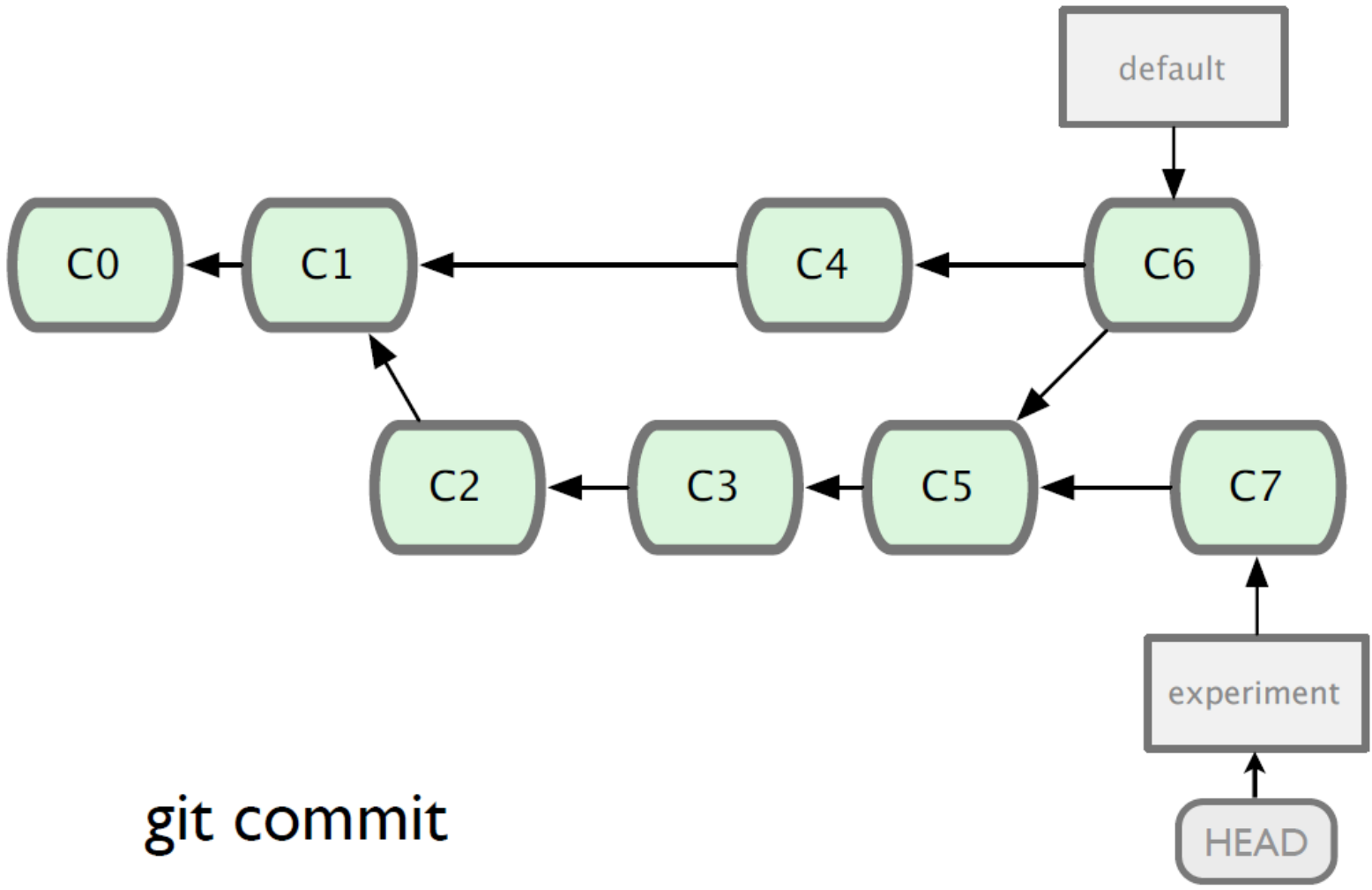




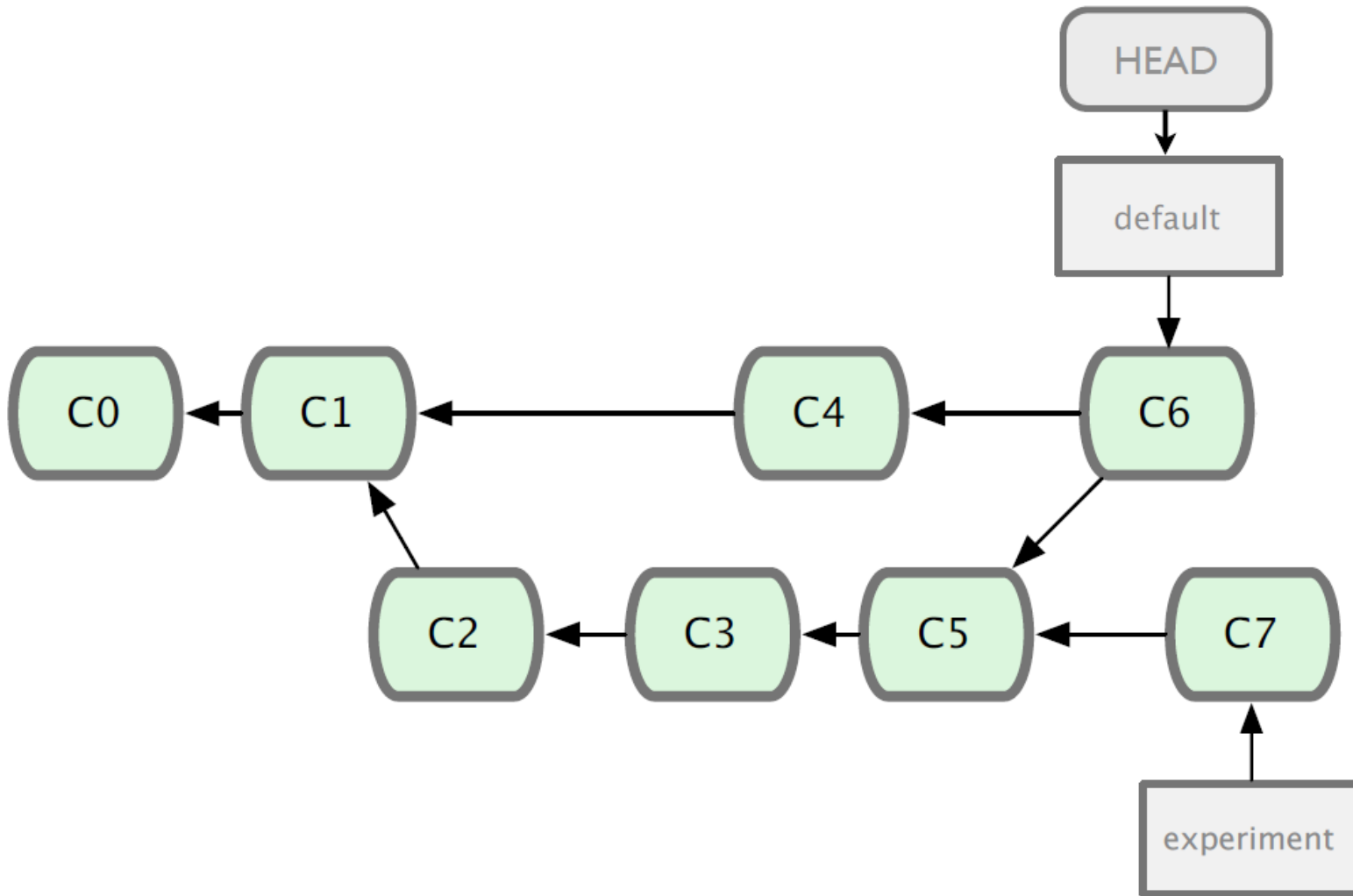
git checkout default  
git merge experiment



git checkout experiment

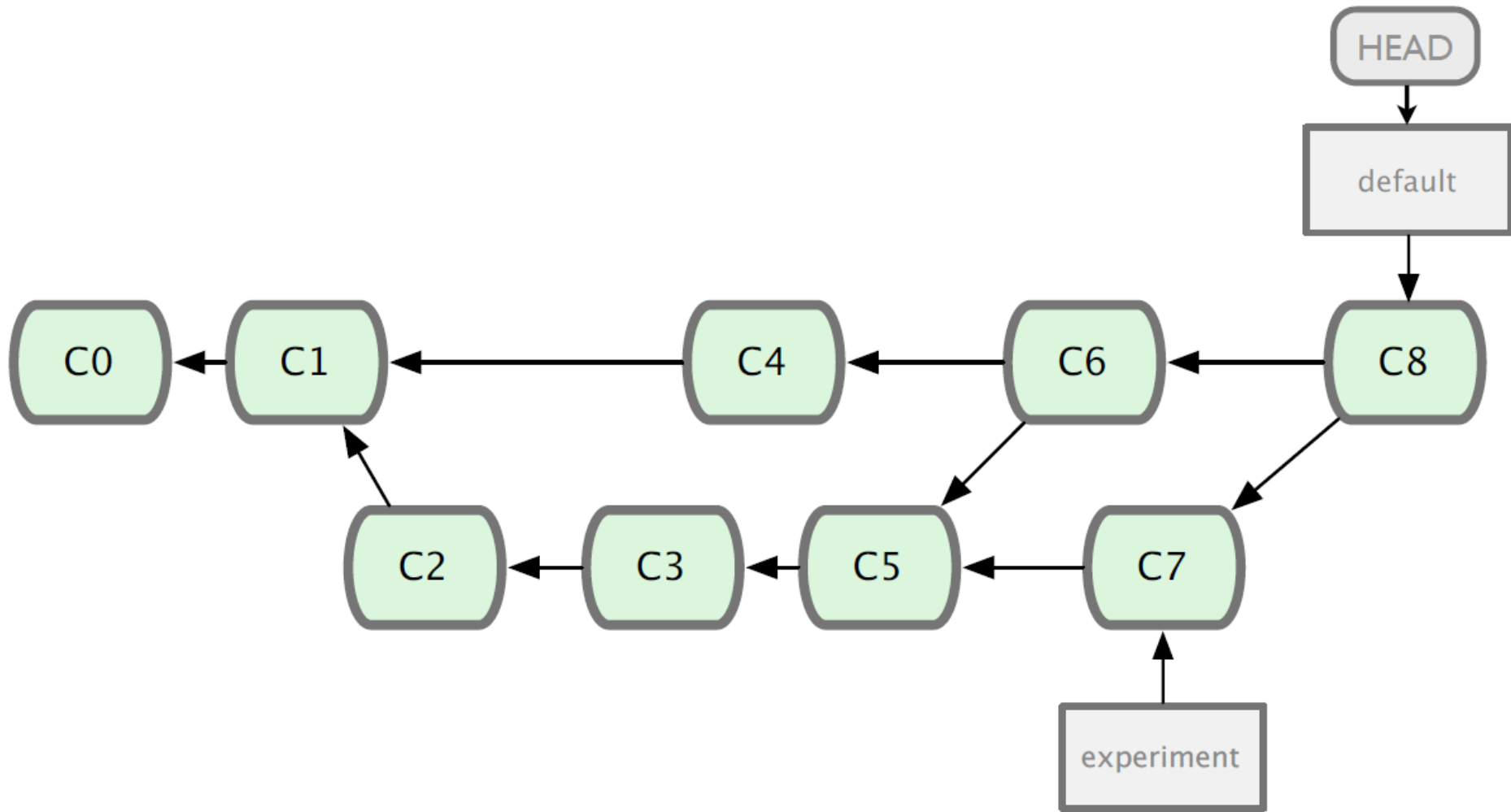


git commit



git checkout default





git merge experiment

# Branching and Merging

- Why this is cool?
  - Non-linear development

```
clone the code that is in production
create a branch for issue #53 (iss53)
work for 10 minutes
someone asks for a hotfix for issue #102
checkout 'production'
create a branch (iss102)
fix the issue
checkout 'production', merge 'iss102'
push 'production'
checkout 'iss53' and keep working
```

# Working with remote

- Use git clone to replicate repository
- Get changes with
  - git fetch
  - git pull (fetches and merges)
- Propagate changes with
  - git push
- Protocols
  - Local filesystem (file://)
  - SSH (ssh://)
  - HTTP (http:// https://)
  - Git protocol (git://)

# Working with remote Local filesystem

- Pros
  - Simple
  - Support existing access control
  - NFS enabled
- Cons
  - Public share is difficult to set up
  - Slow on top of NFS

# Working with remote SSH

- Pros
  - Support authenticated write access
  - Easy to set up as most system provide ssh toolsets
  - Fast
    - Compression before transfer
- Cons
  - No anonymous access
    - Not even for read access

# Working with remote GIT

- Pros

- Fastest protocol
- Allow public anonymous access

- Cons

- Lack of authentication
- Difficult to set up
- Use port 9418
  - Not standard port
  - Can be blocked

# Working with remote HTTP/HTTPS

- Pros
  - Very easy to set up
  - Unlikely to be blocked
    - Using standard port
- Cons
  - Inefficient

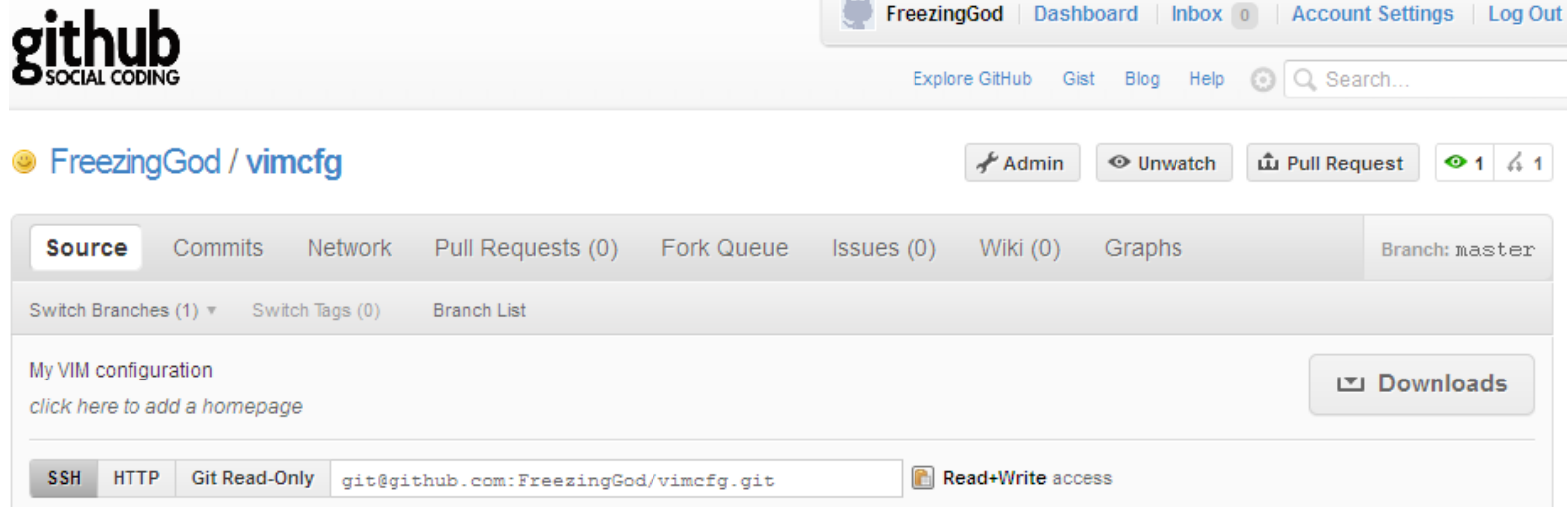
# Working with remote

- One person project
  - Local repo is enough
  - No need to bother with remote
- Small team project
  - SSH write access for a few core developers
  - GIT public read access



# Working with remote

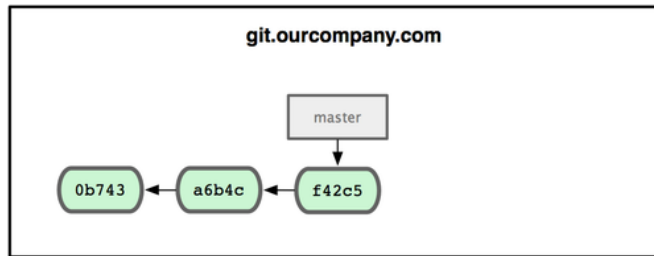
- Use git remote add to add an remote repository



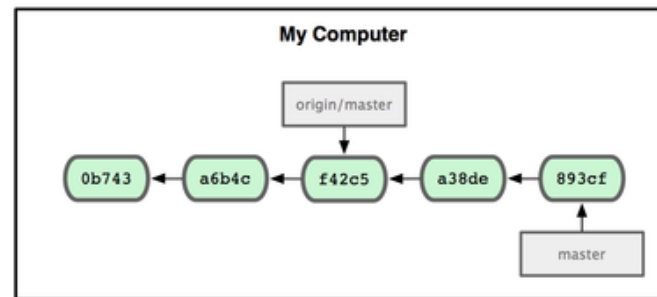
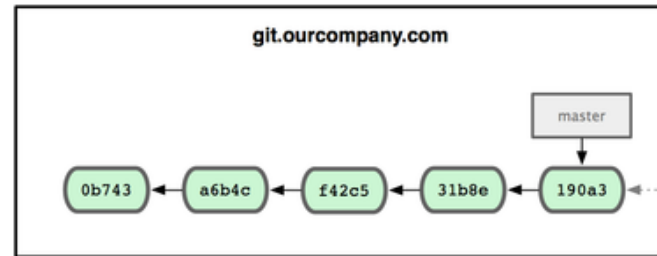
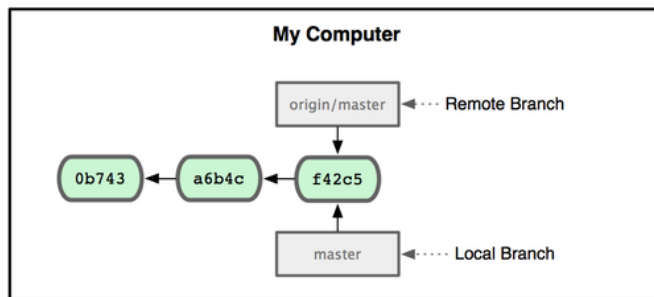
```
Git remote add origin git@github.com:FreezingGod/vimcfg.git  
zachary@zachary-desktop:~/vim_runtime$ git remote  
origin
```

# Working with remote

- Remote branching
  - Branch on remote are different from local branch

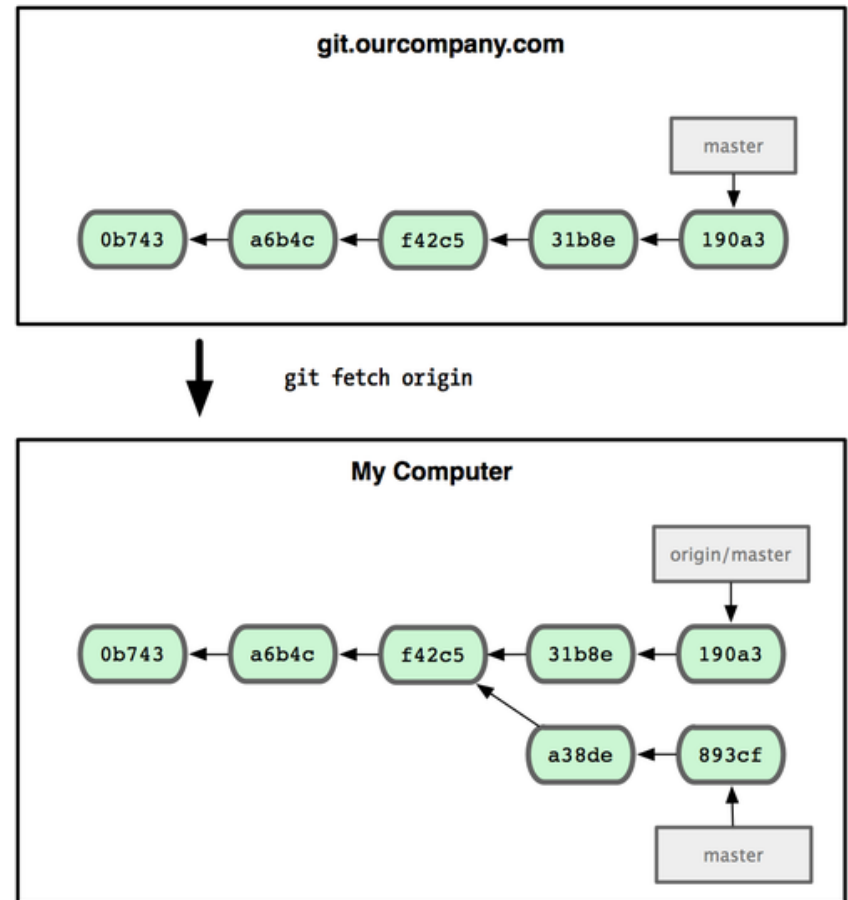


↓ `git clone schacon@git.ourcompany.com:project.git`



# Working with remote

- Remote branching
  - Branch on remote are different from local branch
  - Git fetch origin to get remote changes
  - Git pull origin try to fetch remote changes and merge it onto current branch



# Working with remote

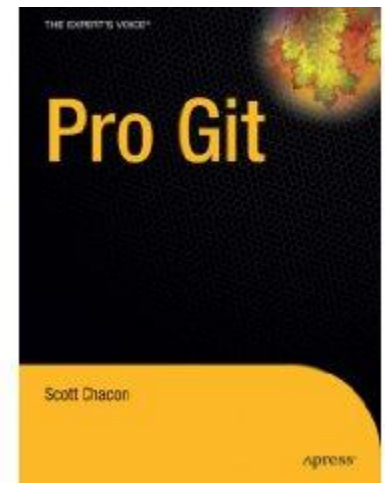
- Git push remote\_name branch\_name
  - Share your work done on branch\_name to remote\_name

# Summary

- We covered fundamentals of Git
  - Three trees of git
    - HEAD, INDEX and working directory
  - Basic work flow
    - Modify, stage and commit cycle
  - Branching and merging
    - Branch and merge
  - Remote
    - Add remote, push, pull, fetch
  - Other commands
    - Revert change, history view

# Summary

- However, this is by no means a complete portrayal of git, some advanced topics are skipped:
  - Rebasing
  - Commit amend
  - Distributed workflow
- For more information, consult
  - Official document
  - Pro Git
    - Free book available at <http://progit.org/book/>



# Q&A

- Any questions?

# References

- Some of the slides are adopted from “Introduction to Git” available at <http://innovationontherun.com/presentation-files/Introduction%20To%20GIT.ppt>
- Some of the figure are adopted from Pro GIT by Chacon, which is available at <http://progit.org/book/>
- Some of the slides are adopted from “Git 101” available at <http://assets.en.oreilly.com/1/event/45/Git%20101%20Tutorial%20Presentation.pdf>