

# Version Control and Git and Github

How Git works and a workflow  
to support scientific  
development

Joe Hamman

UW Mountain Hydrology Research Group

July 7, 2015

# Best Practices for Scientific Computing

D.A. Aruliah <sup>\*</sup>, C. Titus Brown <sup>†</sup>, Neil P. Chue Hong <sup>‡</sup>, Matt Davis <sup>§</sup>, Richard T. Guy <sup>¶</sup>, Steven H.D. Haddock <sup>||</sup>, Katy Huff <sup>\*\*</sup>, Ian M. Mitchell <sup>††</sup>, Mark D. Plumbley <sup>‡‡</sup>, Ben Waugh <sup>§§</sup>, Ethan P. White <sup>¶¶</sup>, Greg Wilson <sup>\*\*\*</sup>, Paul Wilson <sup>†††</sup>

---

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don't repeat yourself (or others).
7. Plan for mistakes.
8. Optimize software only after it works correctly.
9. Document design and purpose, not mechanics.
10. Conduct code reviews.

# Version Control Systems

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

# Version Control Systems

<b>Local data model</b>	<b>Client-server model</b>	<b>Distributed model</b>
All developers must use the same computer system.	Each developers use a shared single repository.	Each developer works directly with his or her own local repository
<b>Examples</b>		
Revision Control System (RCS)	Concurrent Versions System (CVS)	Git
Source Code Control System (SCCS)	Subversion (svn)	Mercurial

This presentation is  
based on:

# Intro to Git

Scott Chacon

adapted by Bart Nijssen...and Joe Hamman

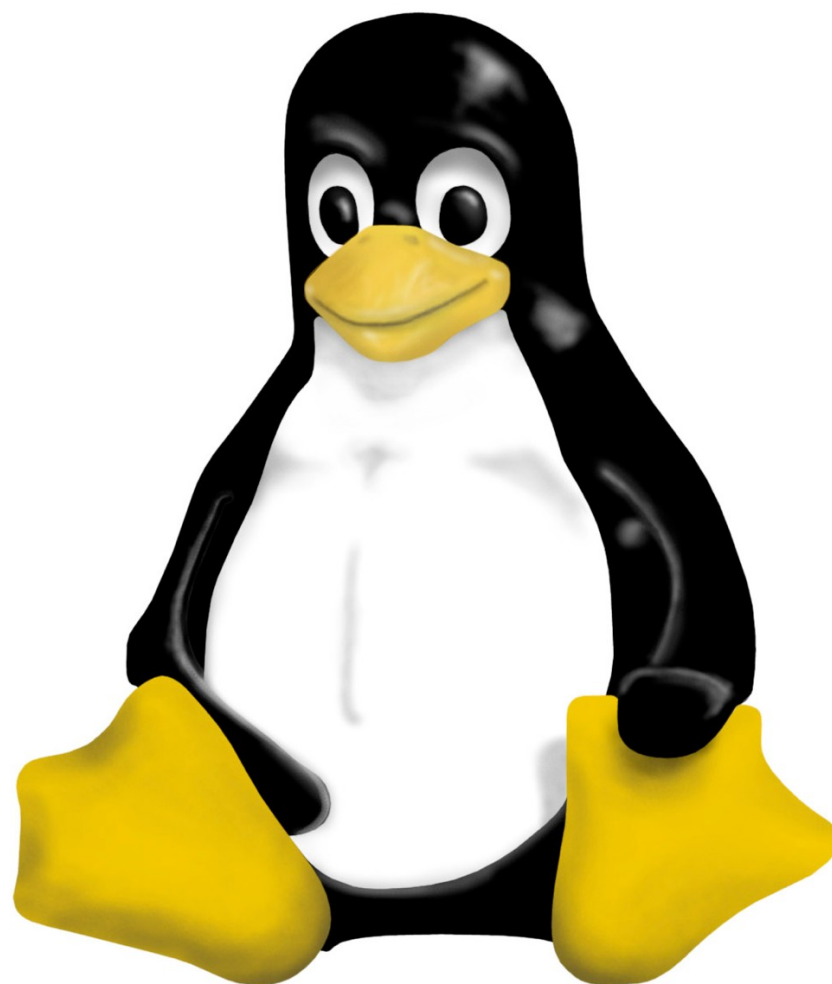
A PDF of the original presentation can be accessed at: [https://github.com/schacon/git-presentations/blob/master/basic\\_git\\_talk/BasicGitTalk.pdf?raw=true](https://github.com/schacon/git-presentations/blob/master/basic_git_talk/BasicGitTalk.pdf?raw=true)

# What is Git?

**Git is an open source,  
distributed version control  
system designed for speed  
and efficiency**

Git is an **open source**,  
distributed version control  
system designed for speed  
and efficiency





Git is an open source,  
**distributed** version control  
system designed for speed  
and efficiency

Fully Distributed

**(almost) everything is local**

# No Network Needed

Performing a diff

Viewing file history

Committing changes

Merging branches

Obtaining any other revision of a file

Switching branches

**which means**

everything is fast

every clone is a backup

work offline

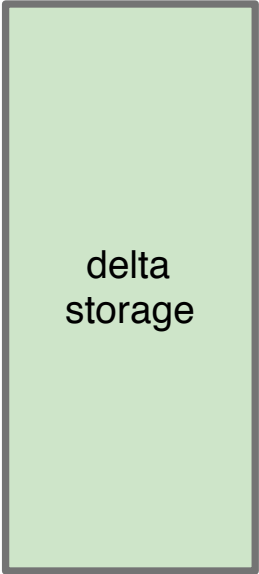
Git is an open source,  
distributed version control  
system **designed for  
speed and efficiency**

Immutable

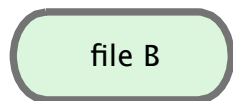
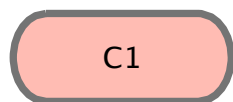
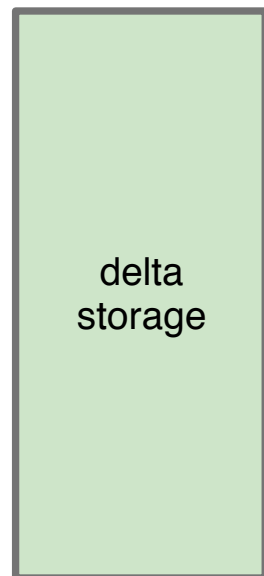


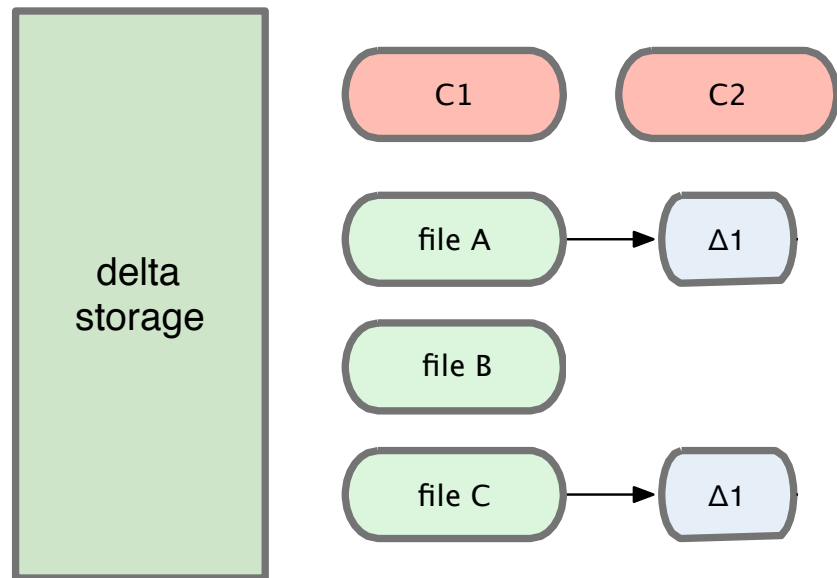
**(almost) never removes data**

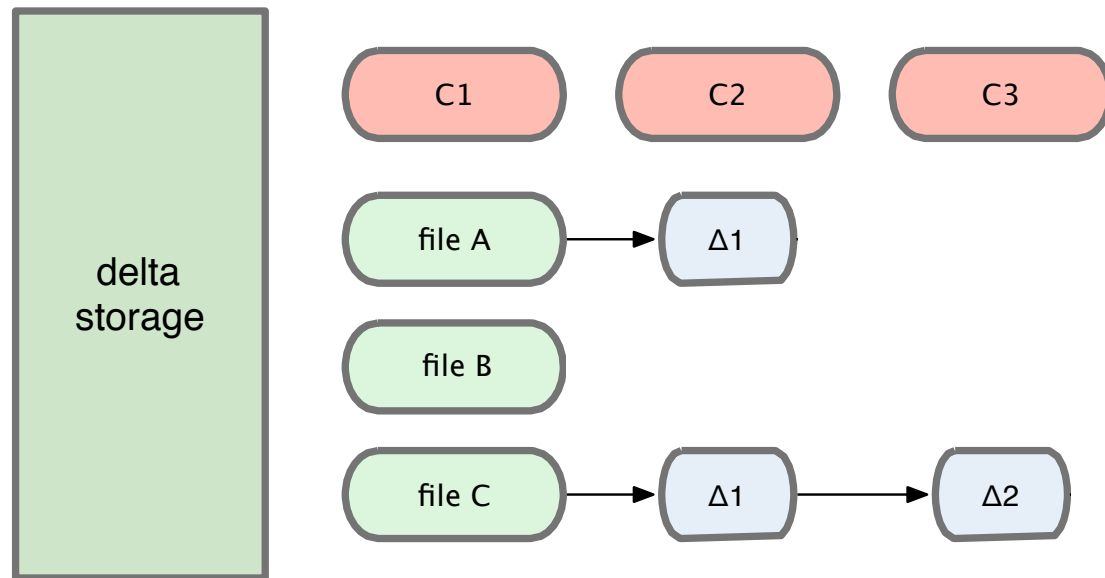
Snapshots, not Patches

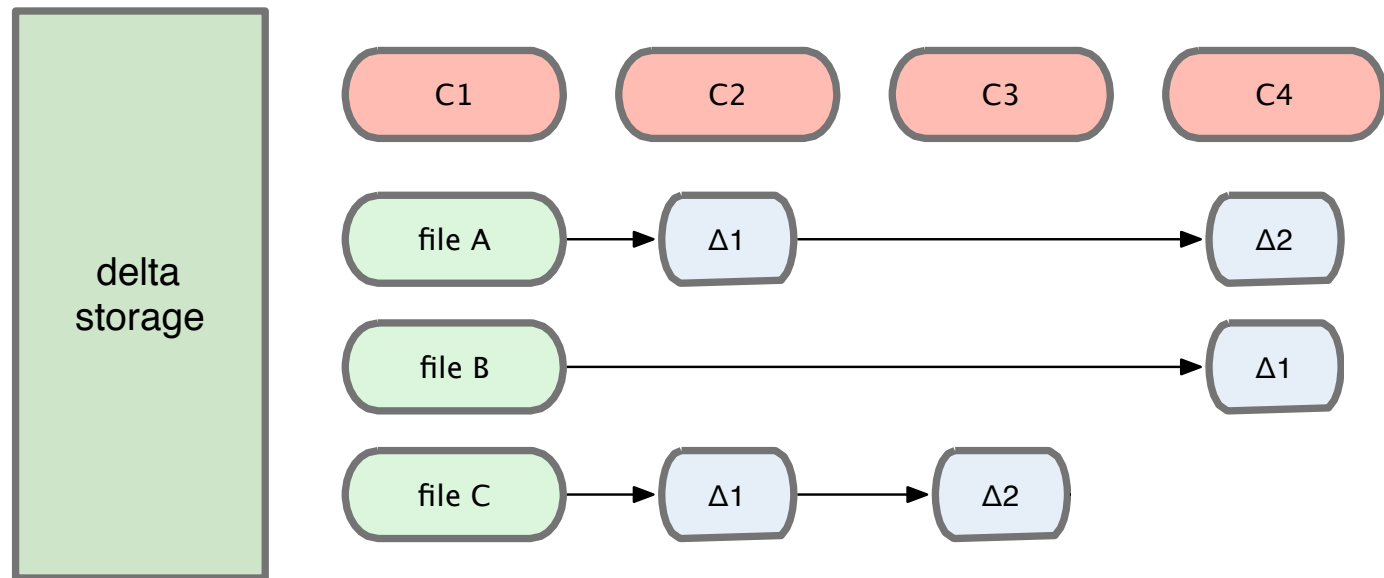


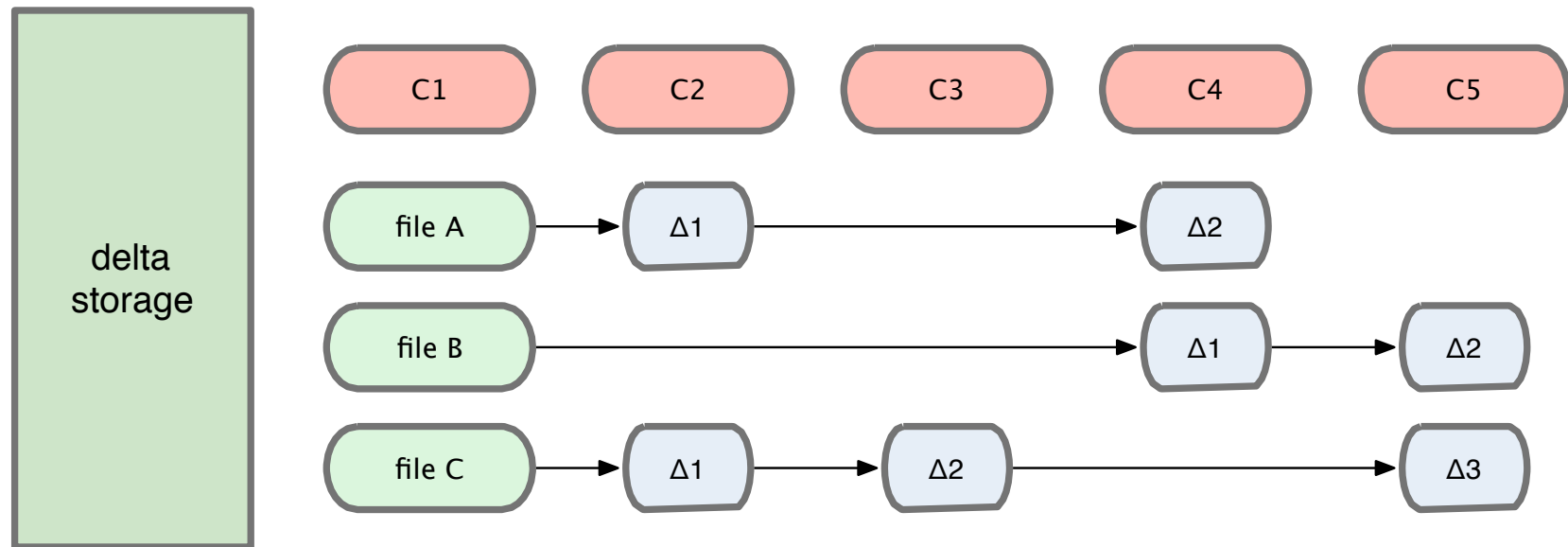
delta  
storage



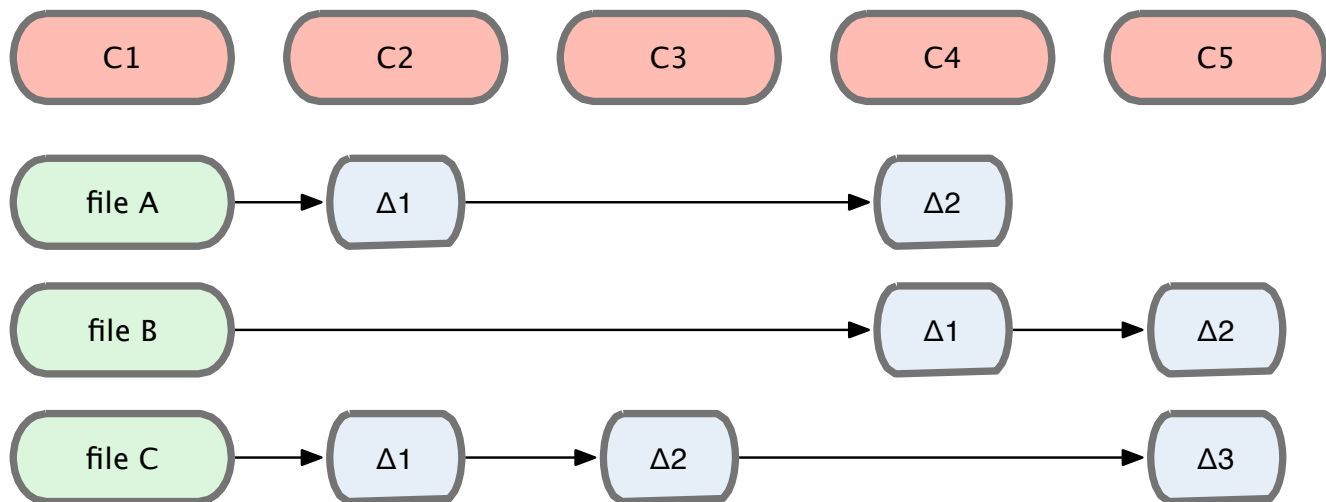
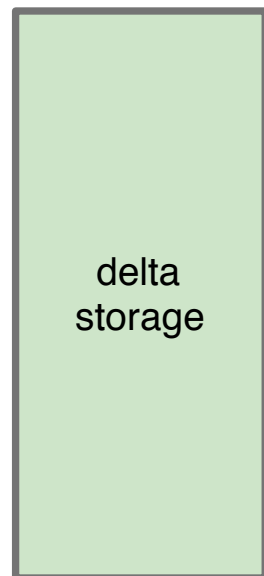


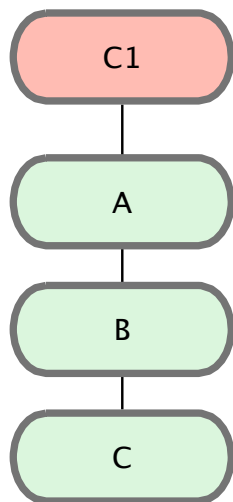
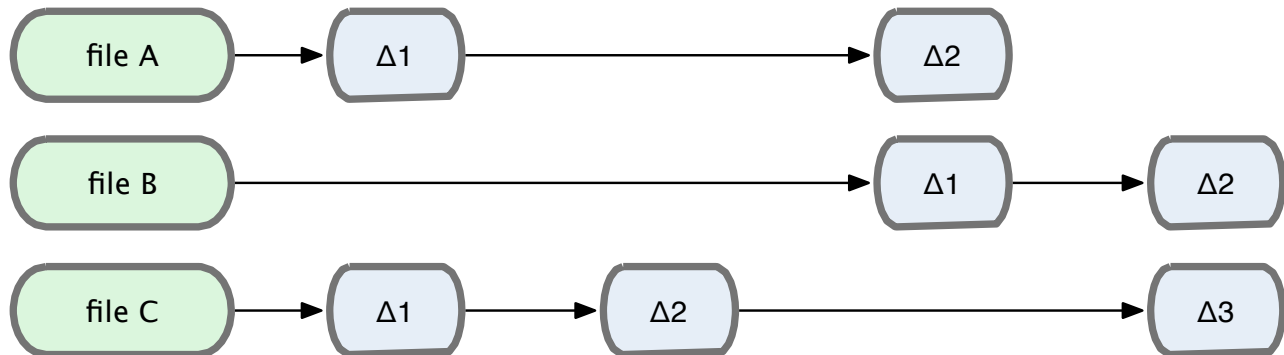
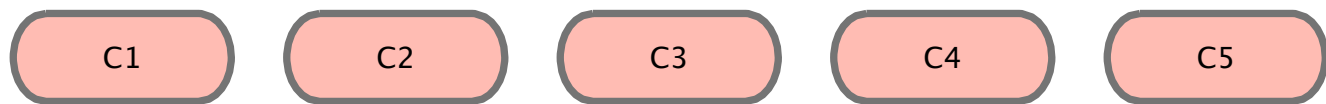
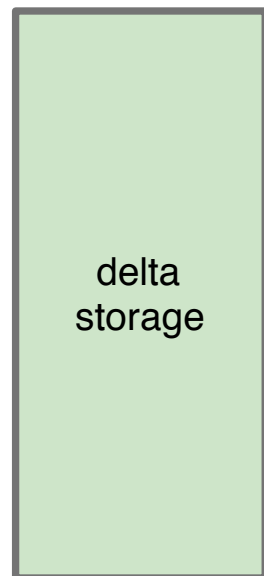


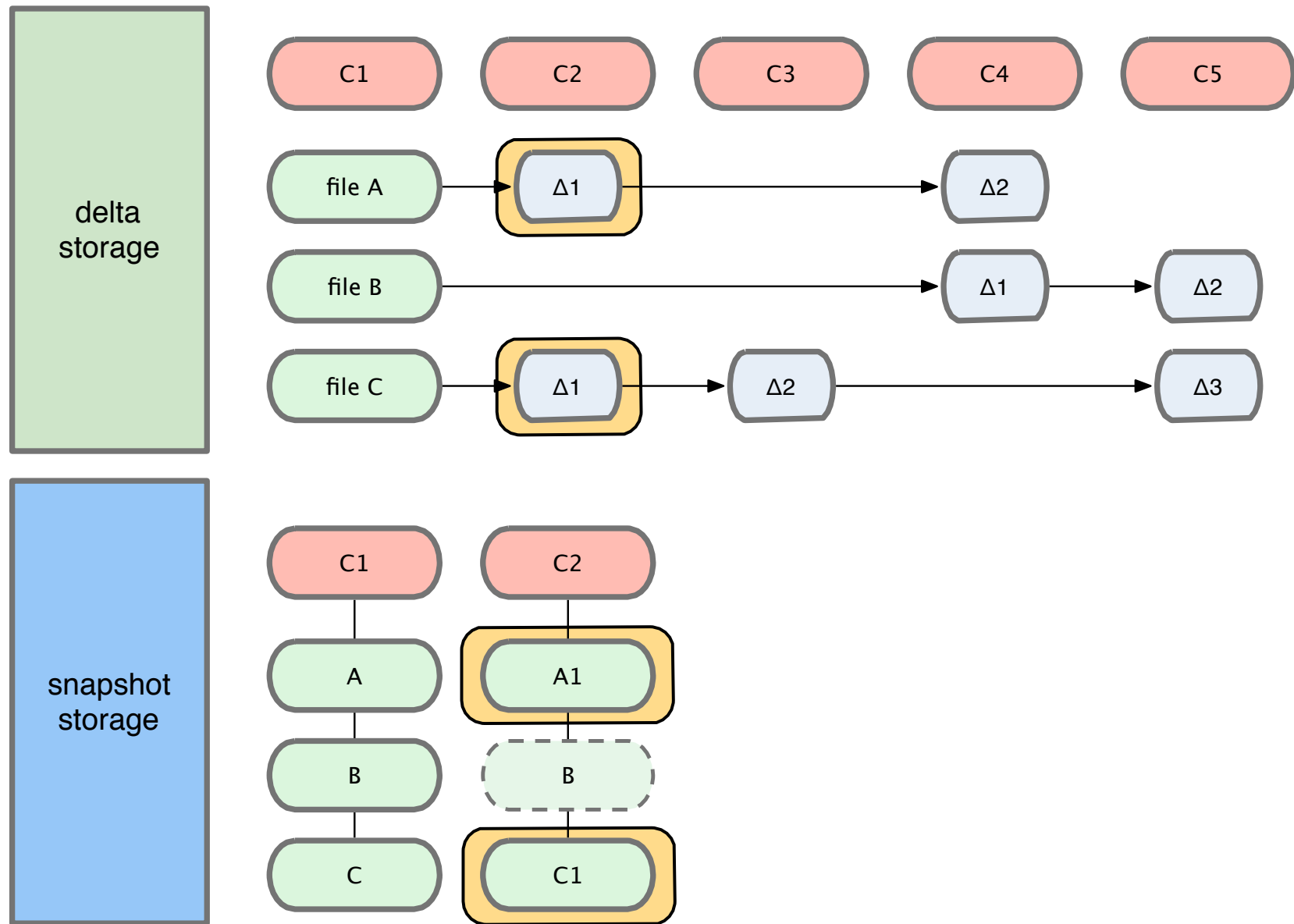


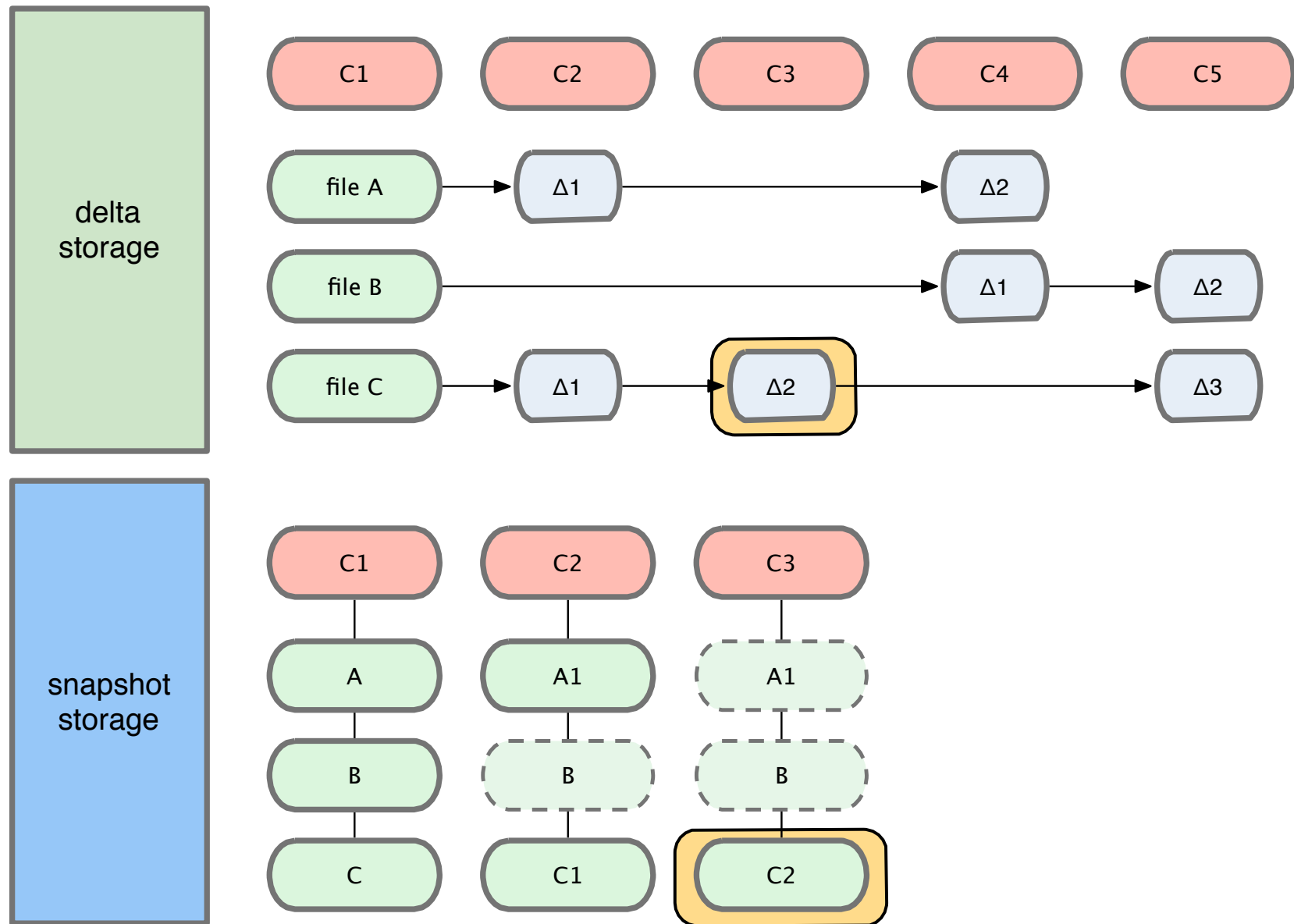


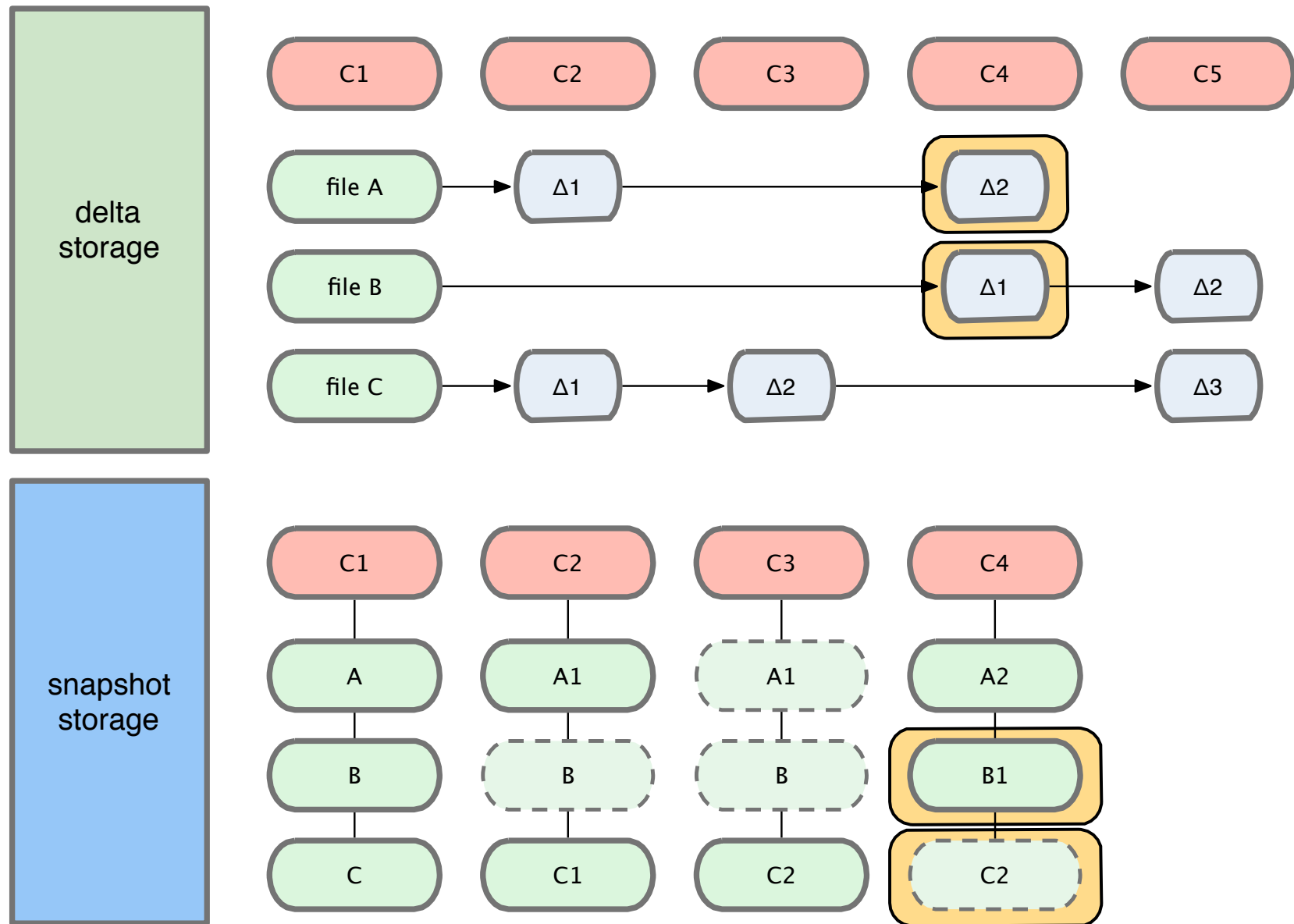




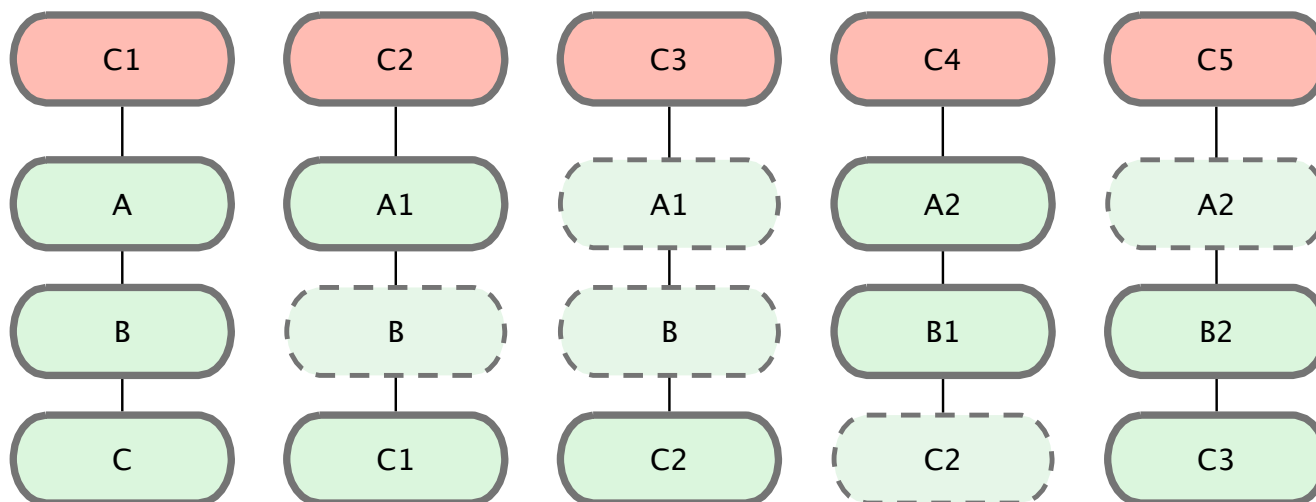
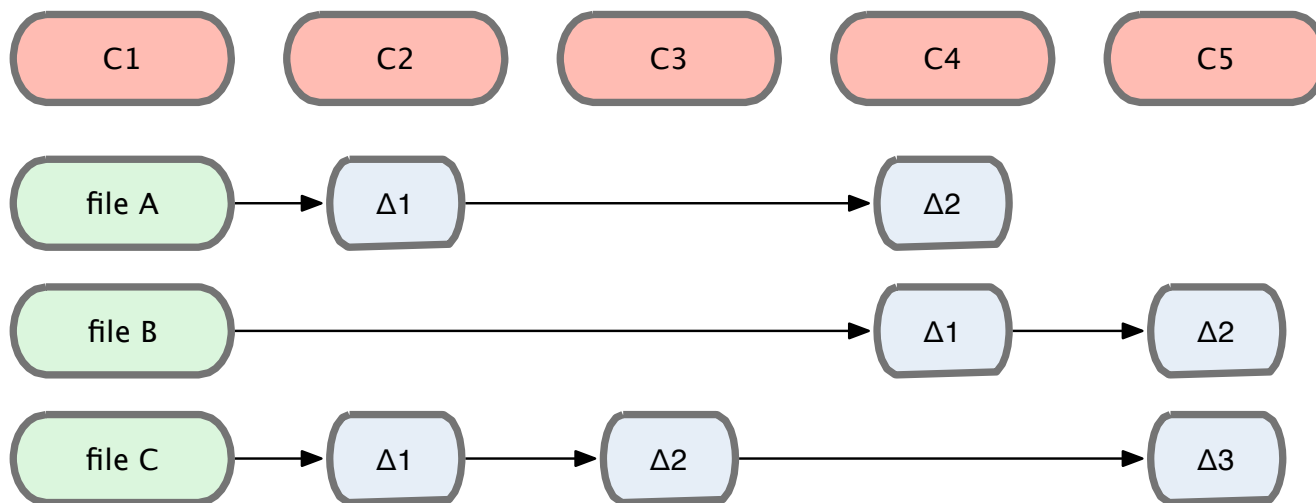
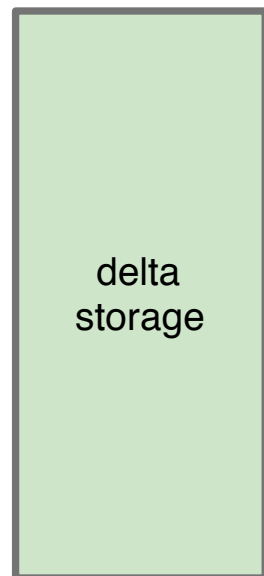


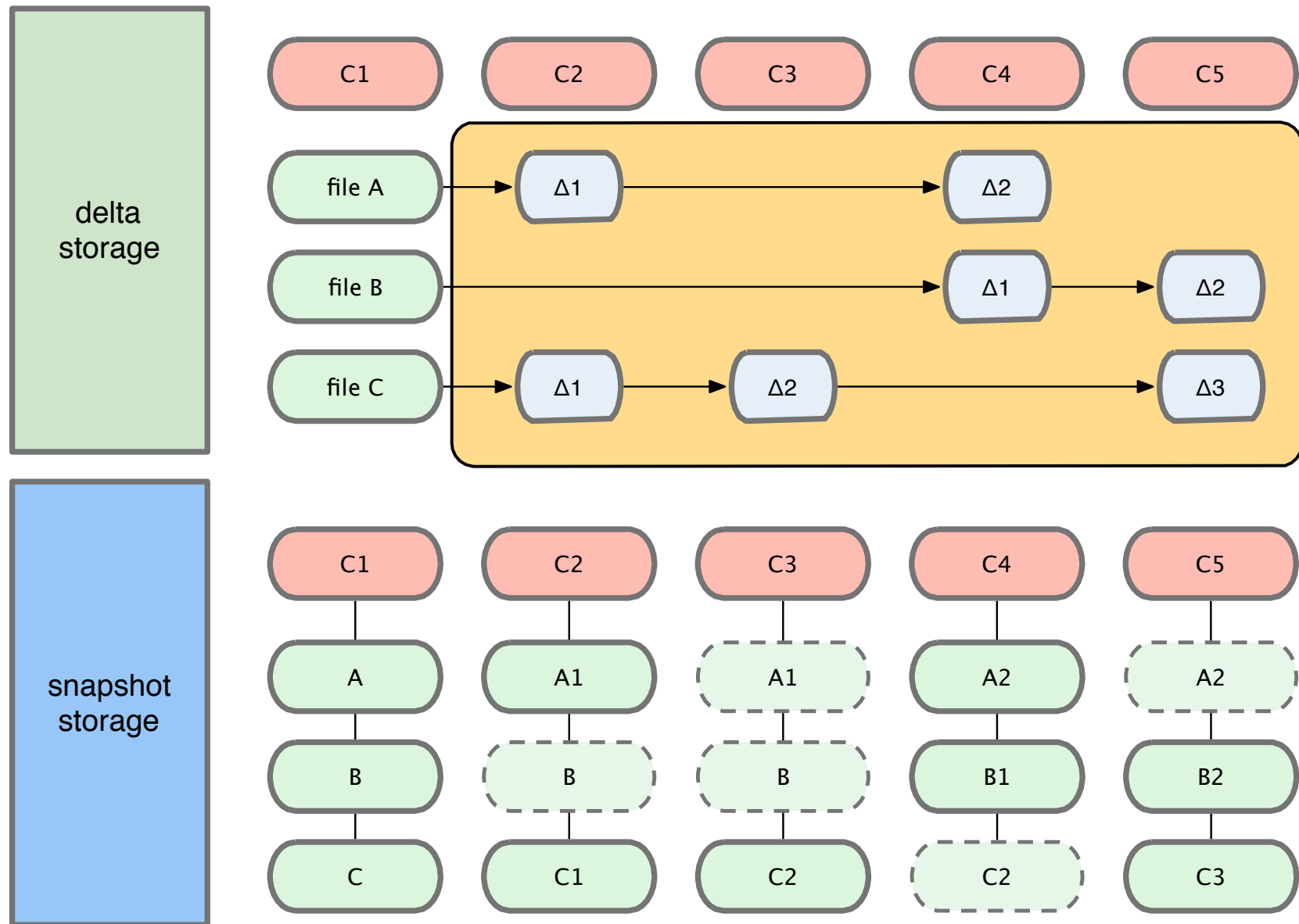




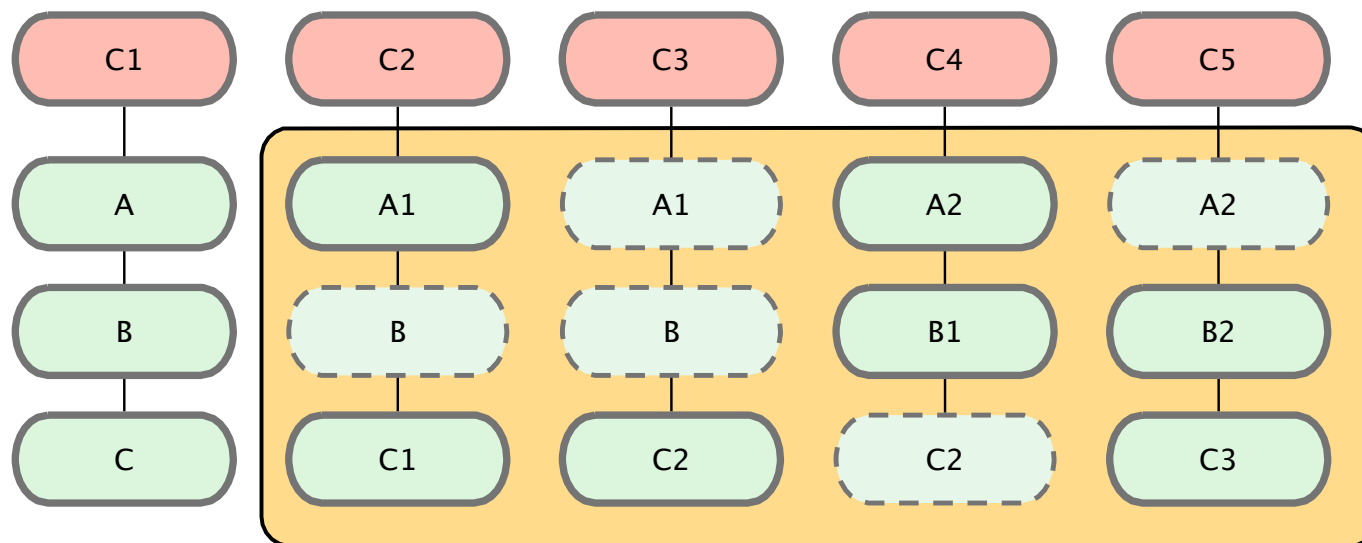
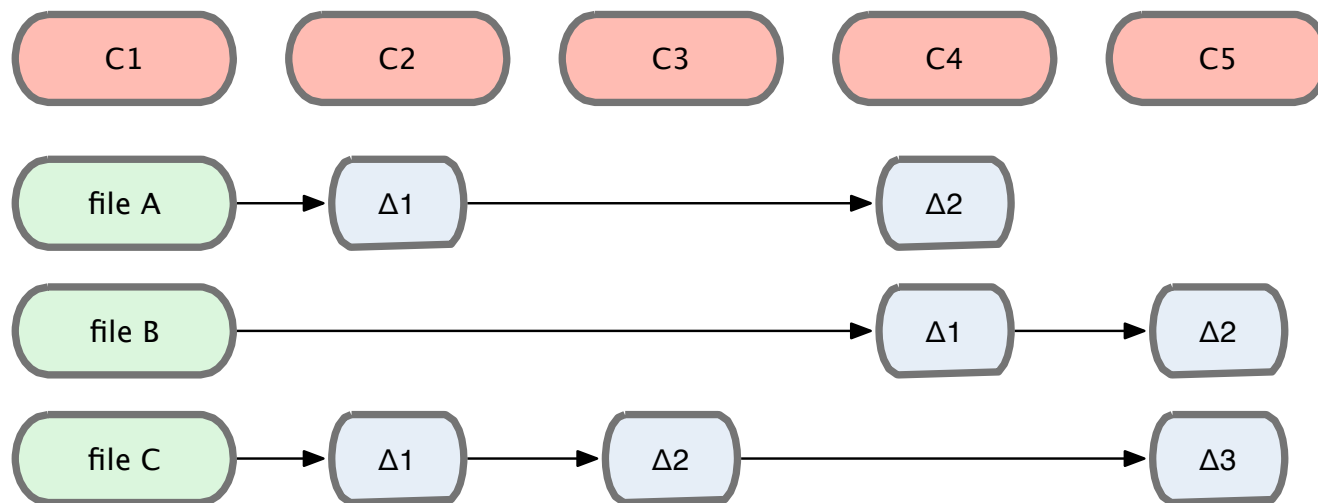
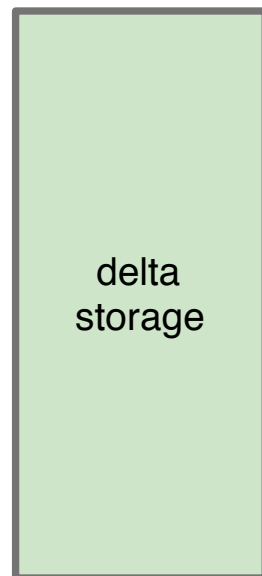












# How Do I Use Git?

# First Steps

# Getting A Repo

`git init` -- Start from scratch

`git clone` -- Start with an existing repo

git init

In the directory where you want to start a repo:

```
$ git init
```

This creates a `.git` subdirectory that contains all the behind-the-scenes info that git interacts with (and that you never edit directly).

In the directory where you want to start a repo:

```
$ git init
```

This creates a .git subdirectory that contains all the behind-the-scenes info that git interacts with (and that you never edit directly).

```
$ tree -a
.
!"" .git
  #"" HEAD
  #"" config
  #"" description
  #"" hooks
  $ #"" applypatch-msg.sample
  $ #"" commit-msg.sample
  $ #"" post-update.sample
  $ #"" pre-applypatch.sample
  $ #"" pre-commit.sample
  $ #"" pre-rebase.sample
  $ #"" prepare-commit-msg.sample
  $ !"" update.sample
  #"" info
  $ !"" exclude
  #"" objects
  $ #"" info
  $ !"" pack
  !"" refs
    #"" heads
    !"" tags
```



# Repository

```
$ tree -a
.
!"" .git
  #"" HEAD
  #"" config
  #"" description
  #"" hooks
  $ #"" applypatch-msg.sample
  $ #"" commit-msg.sample
  $ #"" post-update.sample
  $ #"" pre-applypatch.sample
  $ #"" pre-commit.sample
  $ #"" pre-rebase.sample
  $ #"" prepare-commit-msg.sample
  $ !"" update.sample
  #"" info
  $ !"" exclude
  #"" objects
  $ #"" info
  $ !"" pack
  !"" refs
    #"" heads
    !"" tags
```

Set default settings:

```
$ git config --global user.name "Joe Hamman"
```

```
$ git config --global user.email "jhamman1@uw.edu"
```

Note that you override your global settings for a specific repo:

```
$ git config --local <option> <value>
```

You can get a listing of you current settings:

```
$ git config -l
```

You can directly edit your current settings, e.g.:

```
$ git config --global -e
```

# git settings

/etc/gitconfig

# git settings

/etc/gitconfig

~/.gitconfig

# git settings

/etc/gitconfig

~/.gitconfig

.git/config

```
$ git init  
$ touch hello_world.py  
$ git add .  
$ git commit -m 'first commit'
```

```
$ tree -a
.
#"" .git
$ #"" COMMIT_EDITMSG
$ #"" HEAD
$ #"" config
$ #"" description
$ #"" hooks
$ $ #"" applypatch-msg.sample
$ $ #"" commit-msg.sample
$ $ #"" post-update.sample
$ $ #"" pre-applypatch.sample
$ $ #"" pre-commit.sample
$ $ #"" pre-rebase.sample
$ $ #"" prepare-commit-msg.sample
$ $ !"" update.sample
$ #"" index
$ #"" info
$ $ !"" exclude
$ #"" logs
$ $ #"" HEAD
$ $ !"" refs
$ $ !"" heads
$ $ !"" master
$ #"" objects
$ $ #"" 21
$ $ $ !"" 85689457a69bc7fabdc8245a7856bc733d44e4
$ $ #"" e1
$ $ $ !"" 08bfe338fecfbad4b9025092a8c8bbacac977e
$ $ #"" e6
$ $ $ !"" 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
$ $ #"" info
$ $ !"" pack
$ !"" refs
$ #"" heads
$ $ !"" master
$ !"" tags
!"" hello_world.py
```

git clone



```
$ git clone
```

```
$ git clone git://github.com/schacon/grit.git
```

```
$ git clone git://github.com/schacon/grit.git mygrit
```

```
$ git clone git://github.com/schacon/grit.git mygrit
Initialized empty Git repository in /home/schacon/mygrit/.git/
remote: Counting objects: 3220, done.
remote: Compressing objects: 100% (2093/2093), done.
remote: Total 3220 (delta 1134), reused 3149 (delta 1081)
Receiving objects: 100% (3220/3220), 1.79 MiB | 357 KiB/s, done.
Resolving deltas: 100% (1134/1134), done.
```

```
$ git clone git://github.com/schacon/grit.git mygrit
Initialized empty Git repository in /home/schacon/mygrit/.git/
remote: Counting objects: 3220, done.
remote: Compressing objects: 100% (2093/2093), done.
remote: Total 3220 (delta 1134), reused 3149 (delta 1081)
Receiving objects: 100% (3220/3220), 1.79 MiB | 357 KiB/s, done.
Resolving deltas: 100% (1134/1134), done.
```

```
$ cd mygrit
$
```

```
$ git clone git://github.com/schacon/grit.git mygrit
Initialized empty Git repository in /home/schacon/mygrit/.git/
remote: Counting objects: 3220, done.
remote: Compressing objects: 100% (2093/2093), done.
remote: Total 3220 (delta 1134), reused 3149 (delta 1081)
Receiving objects: 100% (3220/3220), 1.79 MiB | 357 KiB/s, done.
Resolving deltas: 100% (1134/1134), done.
```

```
$ cd mygrit
```

```
$ ls
```

```
API.txt      Manifest.txt README.txt   benchmarks.rb  examples lib
History.txt  PURE_TODO Rakefile   benchmarks.txt grit.gemspec test
$
```

# A Basic Workflow

# A Basic Workflow

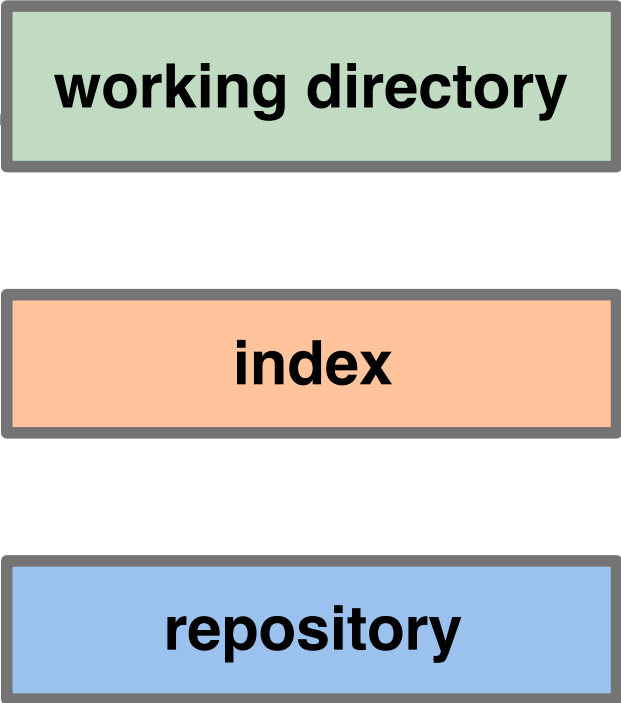
Edit files

Stage the changes

Review your changes

Commit the changes





**working directory**

**index**

**repository**

The diagram consists of three vertically stacked rectangular boxes. The top box is light green with the text 'working directory'. The middle box is light orange with the text 'index'. The bottom box is light blue with the text 'repository'. To the right of the top box is a large light gray rounded rectangle containing the text 'a working copy of your project'.

**working directory**

a working copy  
of your project

**index**

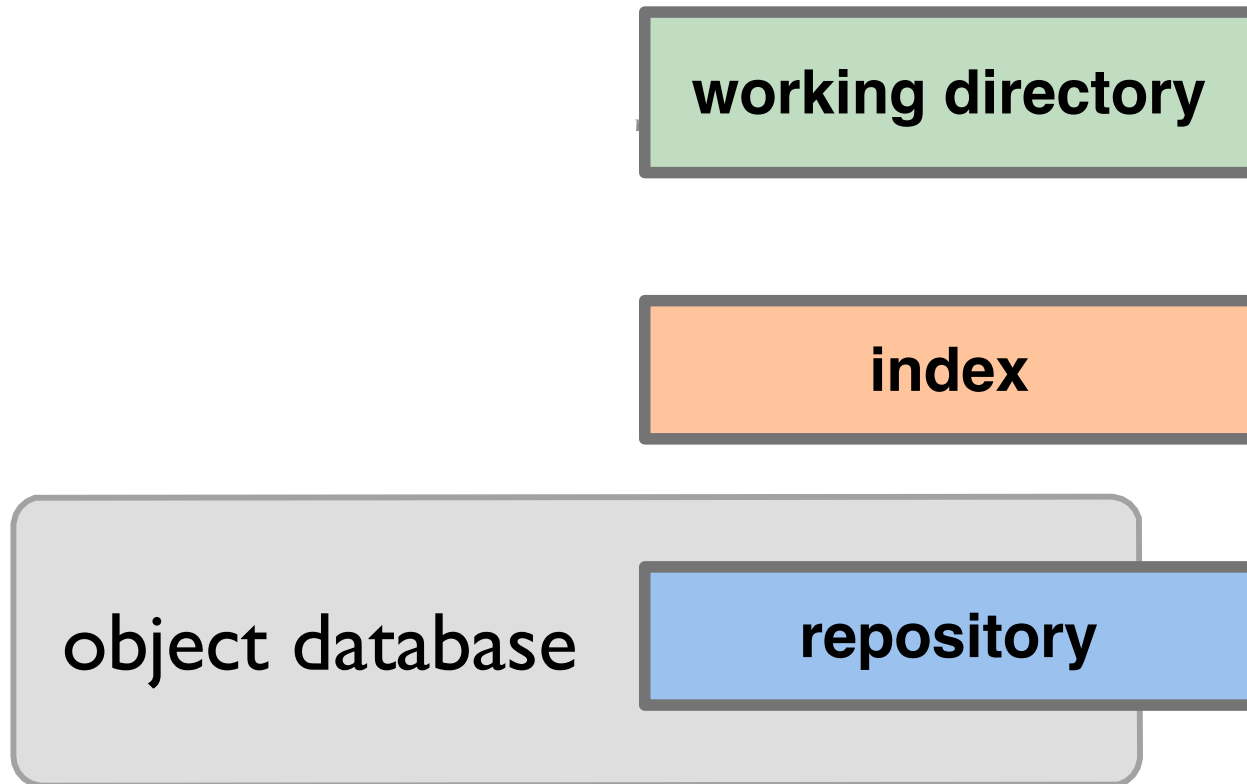
**repository**

**working directory**

**index**

**object database**

**repository**



**working directory**

**index**

“staging”

**repository**

# A Basic Workflow

**Edit files**

Stage the changes

Review your changes

Commit the changes

# \$ edit hello\_world.py

```
#!/usr/bin/env python
"""Hello world program
"""
import sys

def hello(name):
    print 'Hello {}'.format(name)

def main():
    name = 'world'
    if (len(sys.argv) > 1):
        name = sys.argv[1]
    hello(name)

if __name__ == '__main__':
    main()
```

# \$ edit hello\_world.py

```
#!/usr/bin/env python  
"""Hello world program  
"""
```

```
import sys
```

```
# this program prints out 'hello world'
```

```
def hello(name):  
    print 'Hello {}'.format(name)
```

```
def main():  
    name = 'world'  
    if (len(sys.argv) > 1):  
        name = sys.argv[1]  
    hello(name)
```

```
if __name__ == '__main__':  
    main()
```

git status



```
$ git status
```

# \$ git status

```
bash-4.2$ 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_world.py
```

```
#
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

# \$ git status

```
bash-4.2$ 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_world.py
```

```
#
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

# \$ git status

```
bash-4.2$ 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_world.py
```

```
#
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

# \$ git status

```
bash-4.2$ 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_world.py
```

```
#
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

# \$ git status

```
bash-4.2$ 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_world.py
```

```
#
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

# A Basic Workflow

Edit files

Stage the changes

Review your changes

Commit the changes

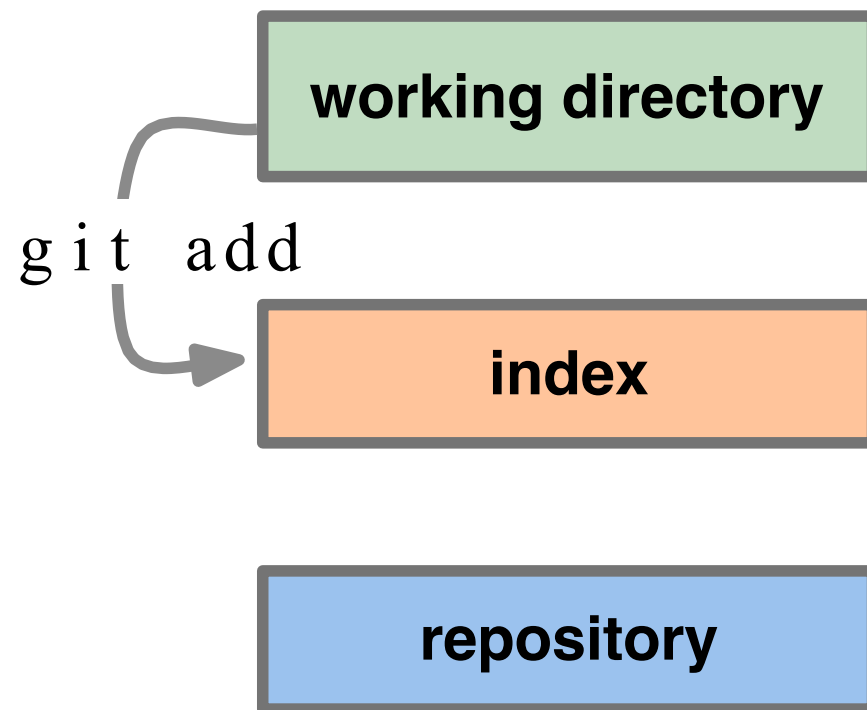
git add



**working directory**

**index**

**repository**



```
$ git add hello_world.py  
$ git status
```

```
# On branch master  
# Changes to be committed:  
#   (use "git reset HEAD <file>..." to unstage)  
#  
# modified:   hello_world.py  
#
```

# \$ git status

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified:   hello_world.py
#
```

```
$ git status
```

```
# On branch master
```

```
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
```

```
#
```

```
# modified:   hello_world.py
```

```
#
```

```
$ git status
```

staged

```
# On branch master
```

```
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
```

```
#
```

```
# modified:   hello_world.py
```

```
#
```

# \$ vim hello\_world.py

```
#!/usr/bin/env python
"""Hello world program
"""
import sys

def hello(name):
    print 'Hello {}'.format(name)
    print '... and a good day to you!'

def main():
    name = 'world'
    if (len(sys.argv) > 1):
        name = sys.argv[1]
    hello(name)

if __name__ == '__main__':
    main()
```

# \$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: hello\_world.py

#

# 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\_world.py

#



## Staged

```
#!/usr/bin/env python
"""Hello world program
"""
import sys

def hello(name):
    print 'Hello {}'.format(name)

def main():
    name = 'world'
    if (len(sys.argv) > 1):
        name = sys.argv[1]
    hello(name)

if __name__ == '__main__':
    main()
```

## InWorking Directory

```
#!/usr/bin/env python
"""Hello world program
"""
import sys

def hello(name):
    print 'Hello {}'.format(name)
    print '... and a good day to you!'

def main():
    name = 'world'
    if (len(sys.argv) > 1):
        name = sys.argv[1]
    hello(name)

if __name__ == '__main__':
    main()
```

## Staged

```
#!/usr/bin/env python
"""Hello world program
"""
import sys

def hello(name):
    print 'Hello {}'.format(name)

def main():
    name = 'world'
    if (len(sys.argv) > 1):
        name = sys.argv[1]
    hello(name)

if __name__ == '__main__':
    main()
```

## InWorking Directory

```
#!/usr/bin/env python
"""Hello world program
"""
import sys

def hello(name):
    print 'Hello {}'.format(name)
    print '... and a good day to you!'

def main():
    name = 'world'
    if (len(sys.argv) > 1):
        name = sys.argv[1]
    hello(name)

if __name__ == '__main__':
    main()
```

**You have to stage a file  
after you edit it**

You have to stage a file  
**after** you edit it

You have to stage a file  
after you edit it

```
$ git add hello_world.py  
$ git status
```

```
# On branch master  
# Changes to be committed:  
#   (use "git reset HEAD <file>..." to unstage)  
#  
#   modified:   hello_world.py  
#  
#
```

# A Basic Workflow

Edit files

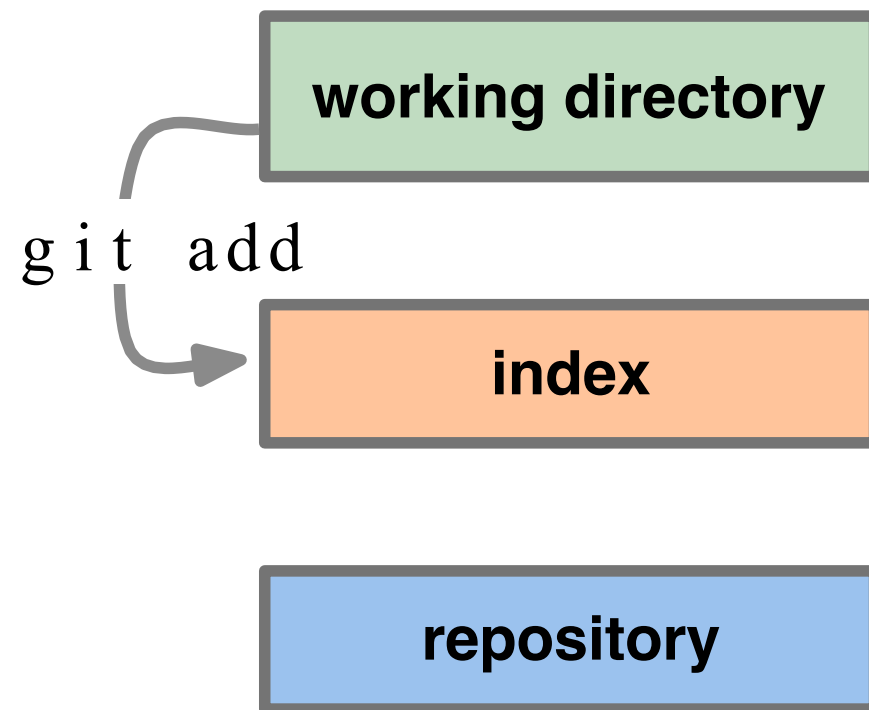
Stage the changes

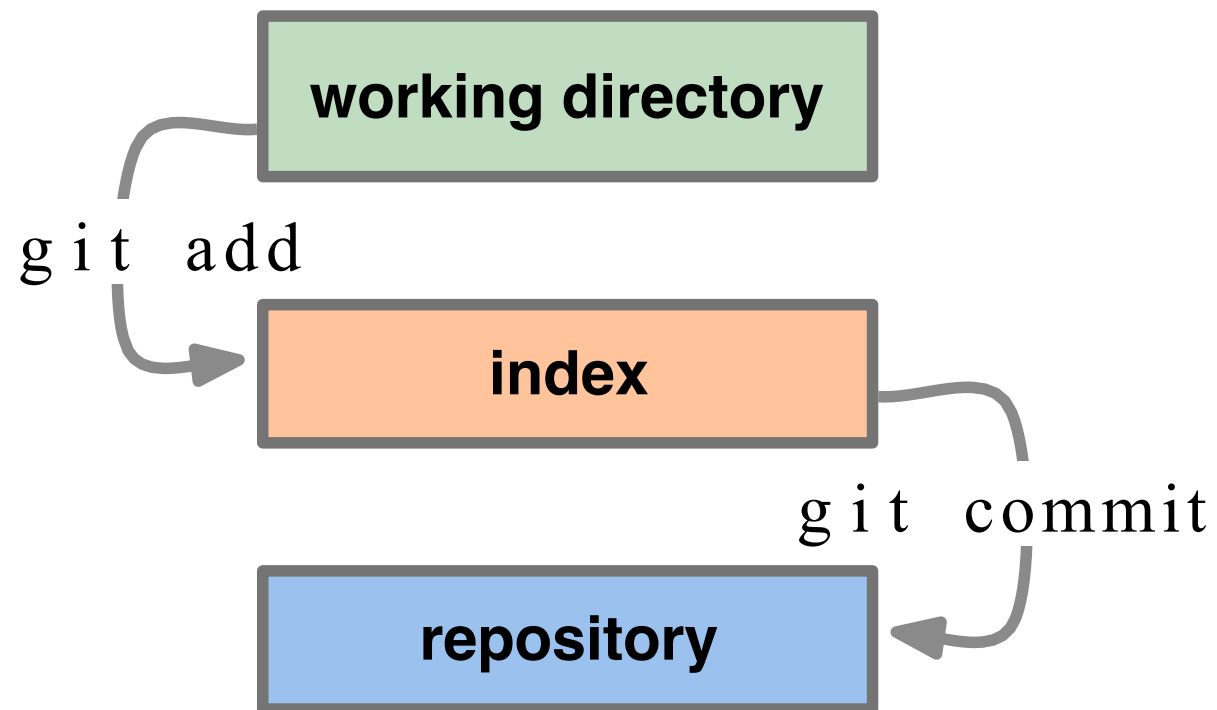
Review your changes

**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_world.py
#
```

# \$ git commit

descriptive commit message

```
# 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_world.py
#
```

# \$ git commit

descriptive commit message

```
# 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_world.py  
#
```

```
$ git commit
```

```
[master 9df86bd] A descriptive commit message  
1 file changed, 17 insertions(+)
```

# A Basic Workflow

# A Basic Workflow

Edit files

vim / emacs / etc



# A Basic Workflow

Edit files

vim / emacs / etc

Stage the changes

git add (file)

# A Basic Workflow

Edit files

vim / emacs / etc

Stage the changes

git add (file)

Review your changes

git status

# A Basic Workflow

Edit files

vim / emacs / etc

Stage the changes

git add (file)

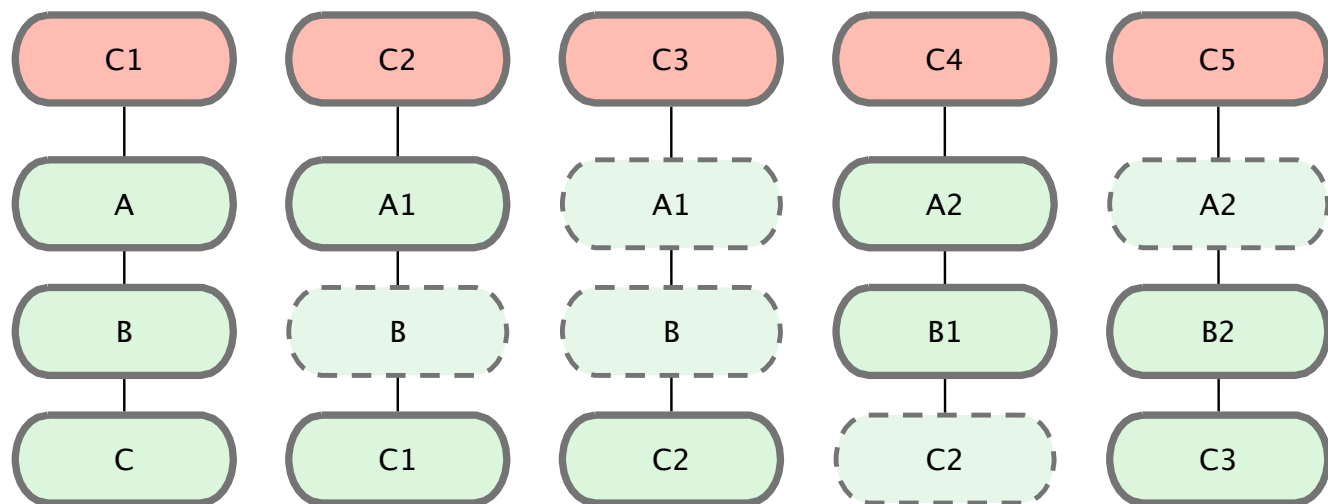
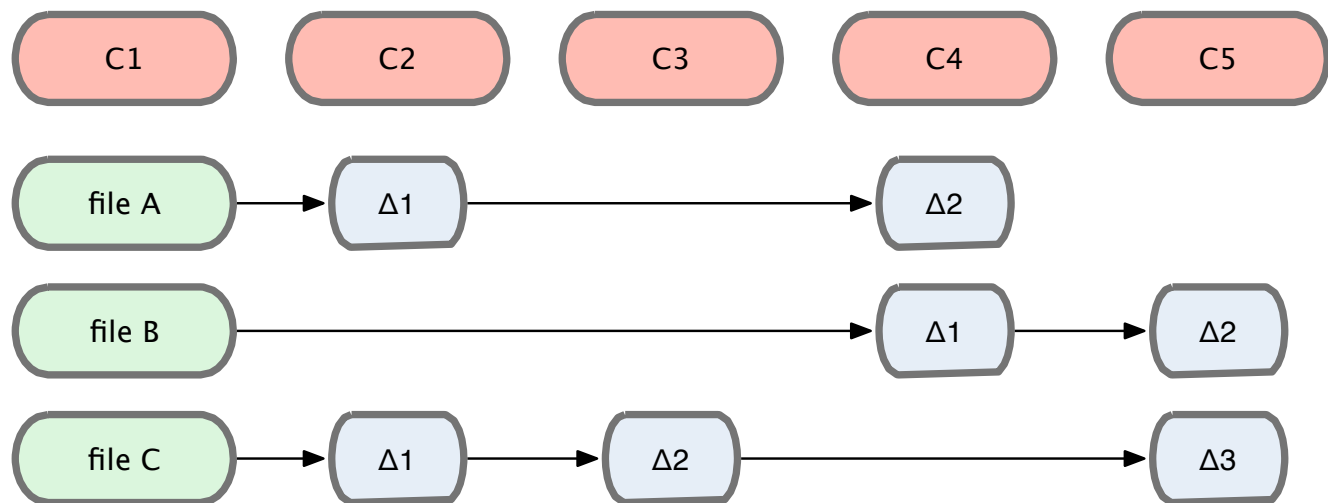
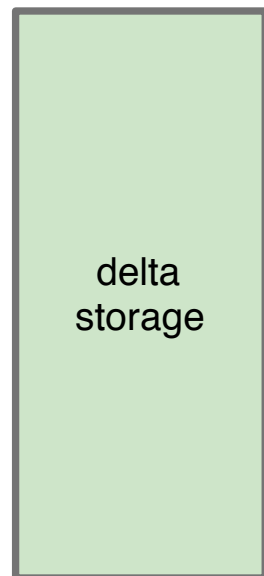
Review your changes

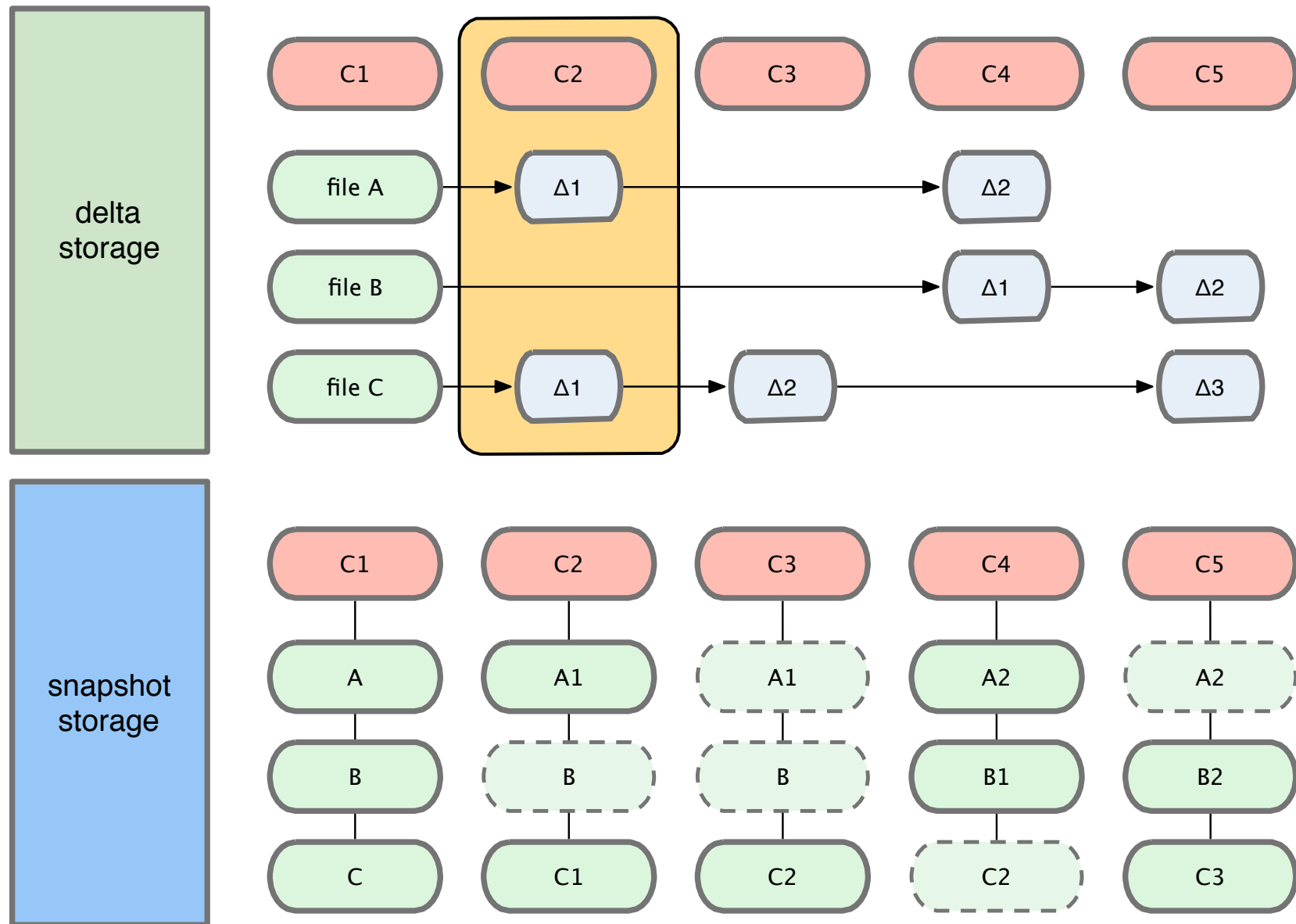
git status

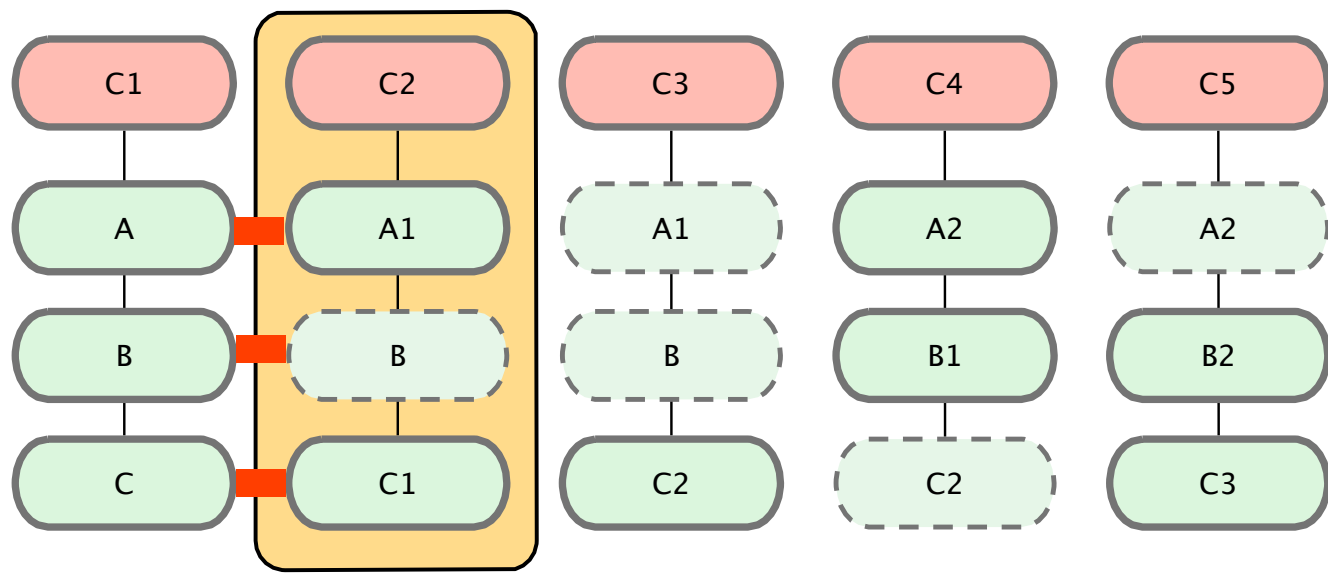
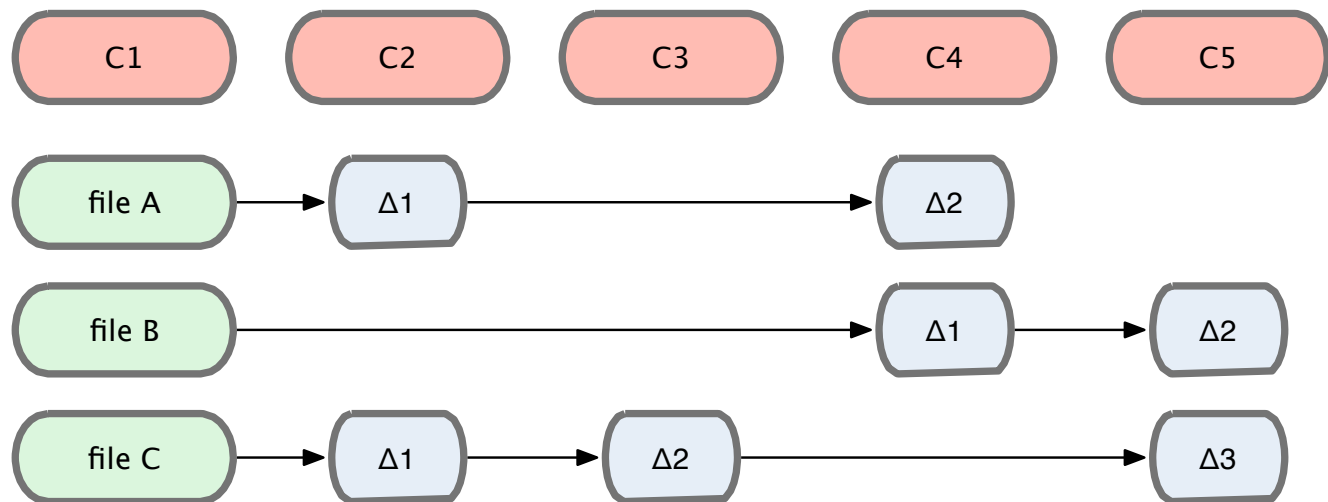
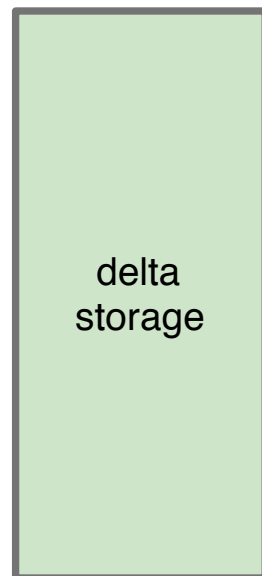
Commit the changes

git commit

# Changes







git diff



```
diff --git a/main.py b/main.py index 6db8b97..b9bcc62
199755
--- a/main.py
+++ b/main.py
@@ -2,10 +2,12 @@
import wsgiref.handlers
from google.appengine.ext import webapp

+# this program prints out 'hello world'
+
class MainHandler (webapp.RequestHandler) :

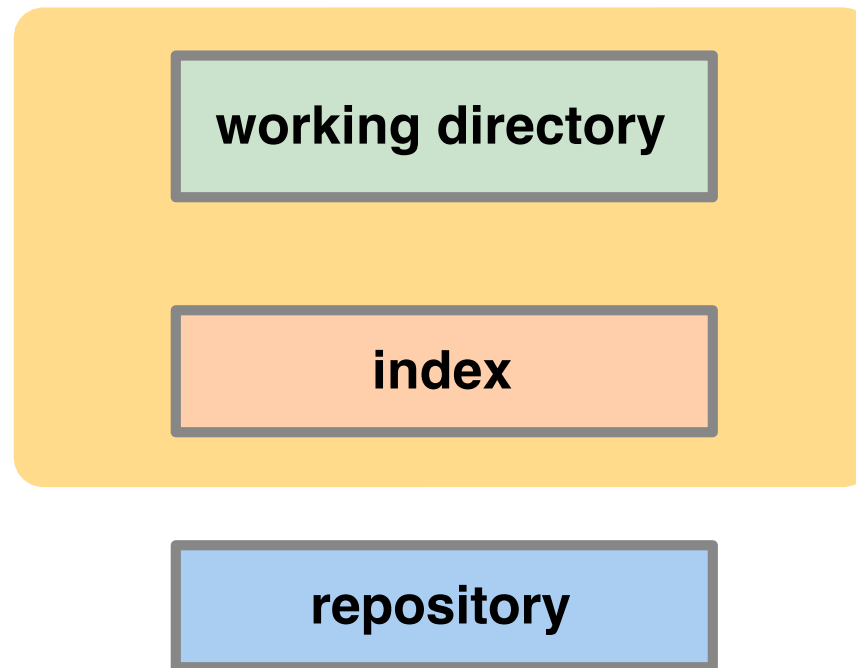
    def get (self) :
+        self.response.out.write ('Hello world!')

def main () :
    application = webapp.WSGIApplication ([('/', MainHandler) ],
```

**What is not yet staged?**

**git diff**

# git diff



# What is staged?

```
git diff --cached
```

# `git diff --cached`



A diagram illustrating the components of a Git repository. It consists of three main parts: a green box labeled 'working directory' at the top, and a larger yellow box below it. Inside the yellow box are two smaller boxes: an orange one labeled 'index' and a blue one labeled 'repository', stacked vertically. All boxes have a thin grey border.

working directory

index

repository

# History



git log

```
$>git log_
```

```
$>git log
```

```
bash-4.2$ git log
```

```
commit 431437d5760aada25d0f8794f40414216afd813c
```

```
Author: Joe Hamman <jhamman@hydro.washington.edu>
```

```
Date: Sun Oct 20 15:16:16 2013 -0700
```

```
a descriptive commit message
```

```
commit e261ed6010585c32e395b8c8cd499015fc31345b
```

```
Author: Joe Hamman <jhamman@hydro.washington.edu>
```

```
Date: Sun Oct 20 15:03:15 2013 -0700
```

```
fix typo in main()
```

```
commit 39e22eae7bc6380108c5688764fafa8e43f61eae
```

```
Author: Joe Hamman <jhamman@hydro.washington.edu>
```

```
Date: Sun Oct 20 15:02:20 2013 -0700
```

```
initial commit of hello_world
```

```
bash-4.2$ git log
```

```
commit 431437d5760aada25d0f8794f40414216afd813c
```

```
Author: Joe Hamman <jhamman@hydro.washington.edu>
```

```
Date: Sun Oct 20 15:16:16 2013 -0700
```

```
a descriptive commit message
```

```
commit e261ed6010585c32e395b8c8cd499015fc31345b
```

```
Author: Joe Hamman <jhamman@hydro.washington.edu>
```

```
Date: Sun Oct 20 15:03:15 2013 -0700
```

```
fix typo in main()
```

```
commit 39e22eae7bc6380108c5688764fafa8e43f61eae
```

```
Author: Joe Hamman <jhamman@hydro.washington.edu>
```

```
Date: Sun Oct 20 15:02:20 2013 -0700
```

```
initial commit of hello_world
```

bash-4.2\$ git log

commit 431437d5760aada25d0f8794f40414216afd813c

Author: Joe Hamman <jhamman@hydro.washington.edu>

Date: Sun Oct 20 15:16:16 2013 -0700

a descriptive commit message

commit e261ed6010585c32e395b8c8cd499015fc31345b

Author: Joe Hamman <jhamman@hydro.washington.edu>

Date: Sun Oct 20 15:03:15 2013 -0700

fix typo in main()

commit 39e22eae7bc6380108c5688764fafa8e43f61eae

Author: Joe Hamman <jhamman@hydro.washington.edu>

Date: Sun Oct 20 15:02:20 2013 -0700

initial commit of hello\_world

bash-4.2\$ git log

commit 431437d5760aada25d0f8794f40414216afd813c

Author: Joe Hamman <jhamman@hydro.washington.edu>

Date: Sun Oct 20 15:16:16 2013 -0700

a descriptive commit message

commit e261ed6010585c32e395b8c8cd499015fc31345b

Author: Joe Hamman <jhamman@hydro.washington.edu>

Date: Sun Oct 20 15:03:15 2013 -0700

fix typo in main()

commit 39e22eae7bc6380108c5688764fafa8e43f61eae

Author: Joe Hamman <jhamman@hydro.washington.edu>

Date: Sun Oct 20 15:02:20 2013 -0700

initial commit of hello\_world

# Branching and Merging

# branches



# branches

lightweight, movable  
pointers to a commit

branch



C1

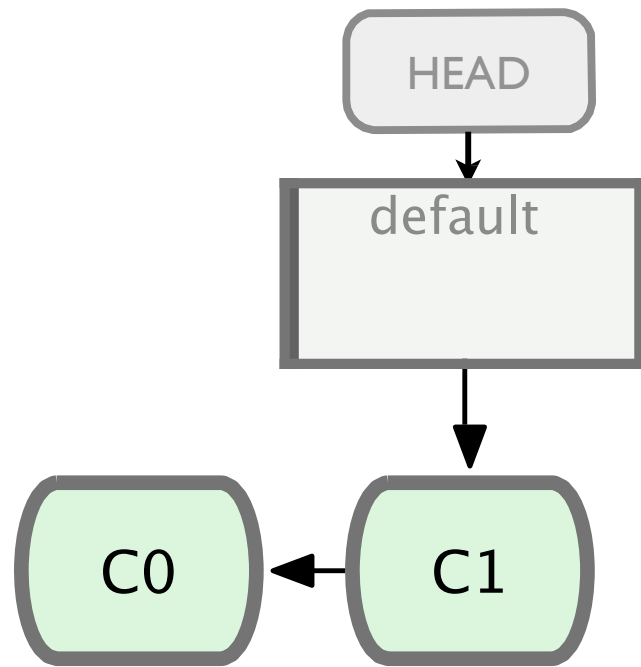
**branching**

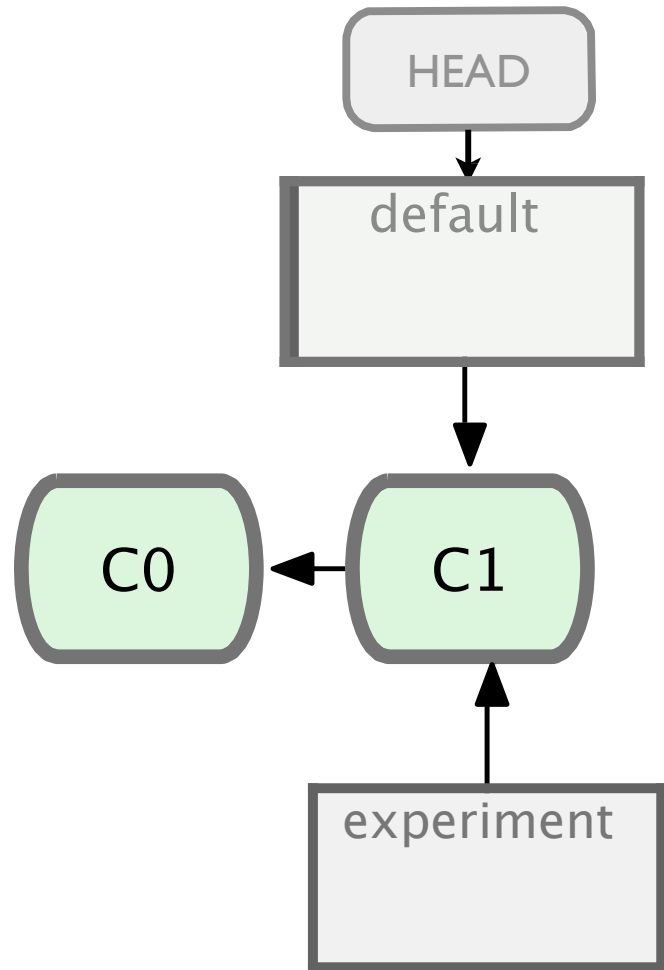
# git branch

create/delete/query a branch

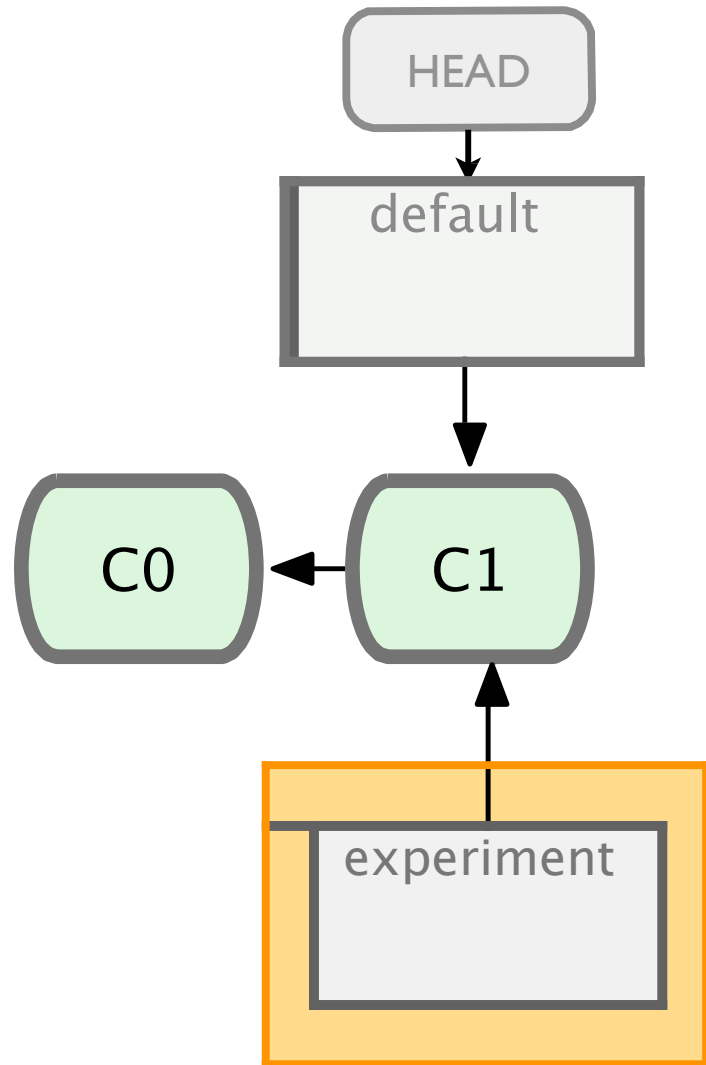
# git checkout

switch to/select a branch

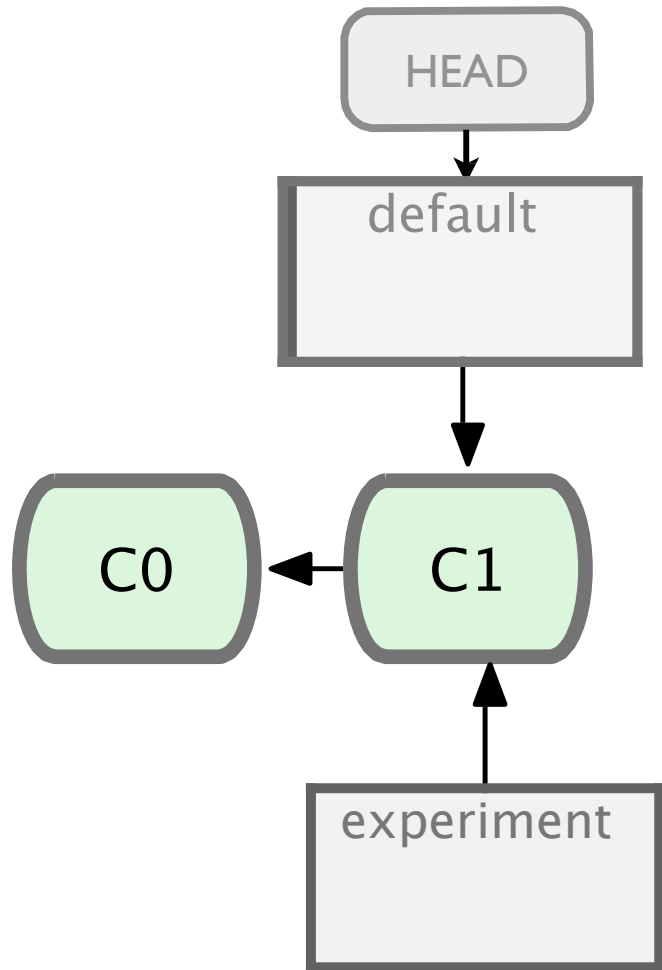




git branch experiment

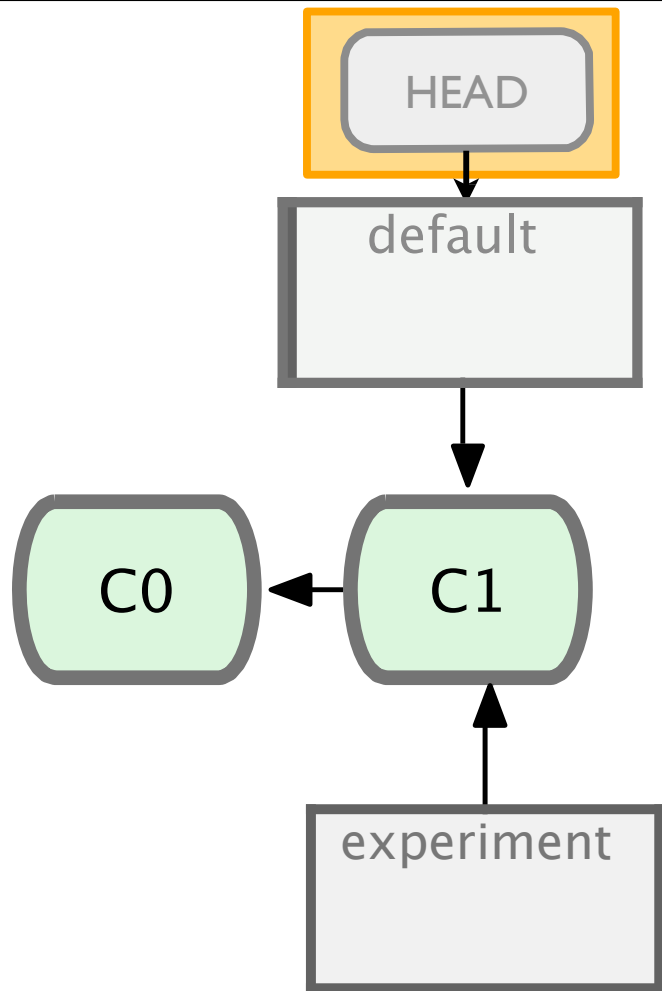


git branch experiment

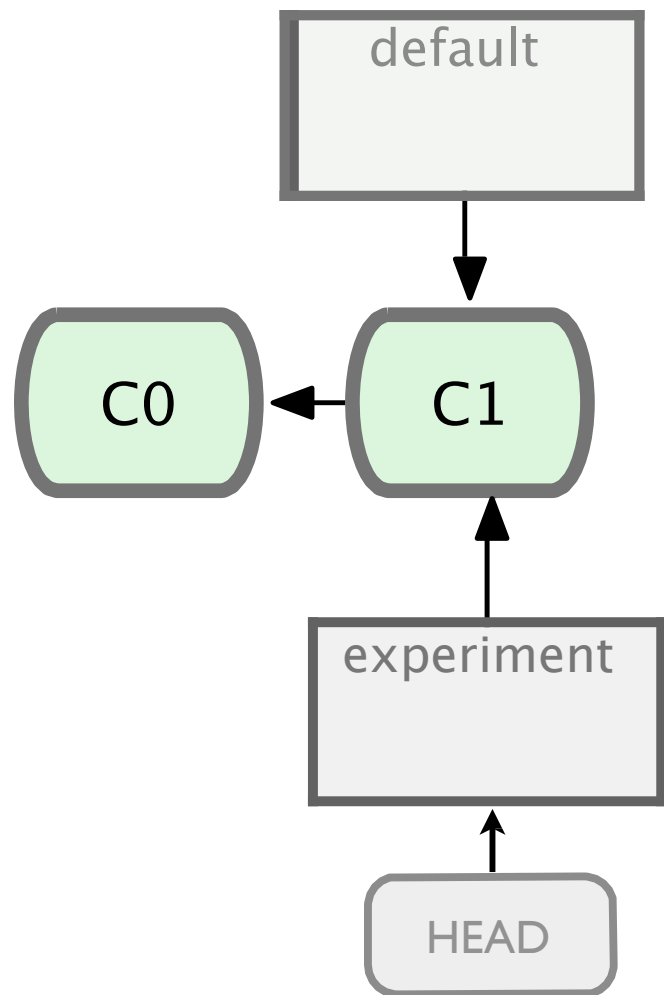


```
$ git branch  
* default  
experiment
```

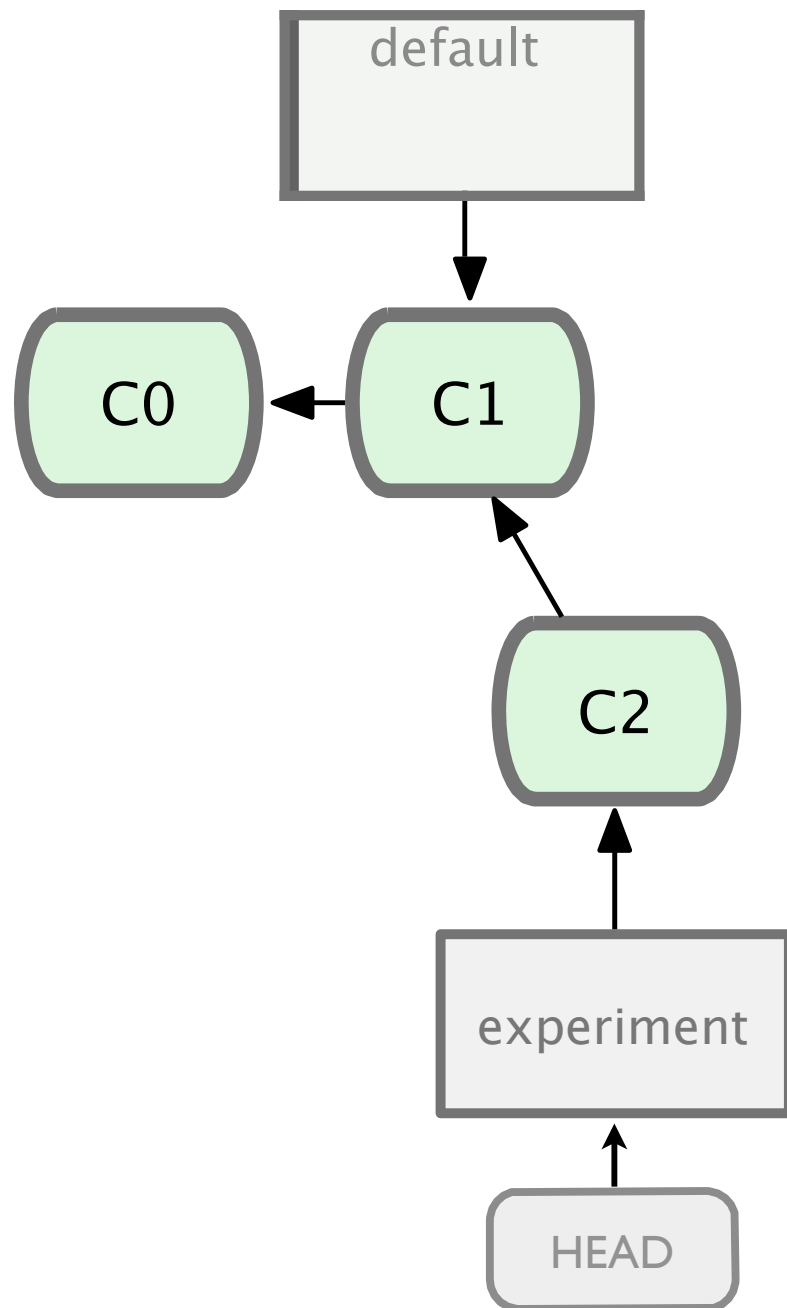




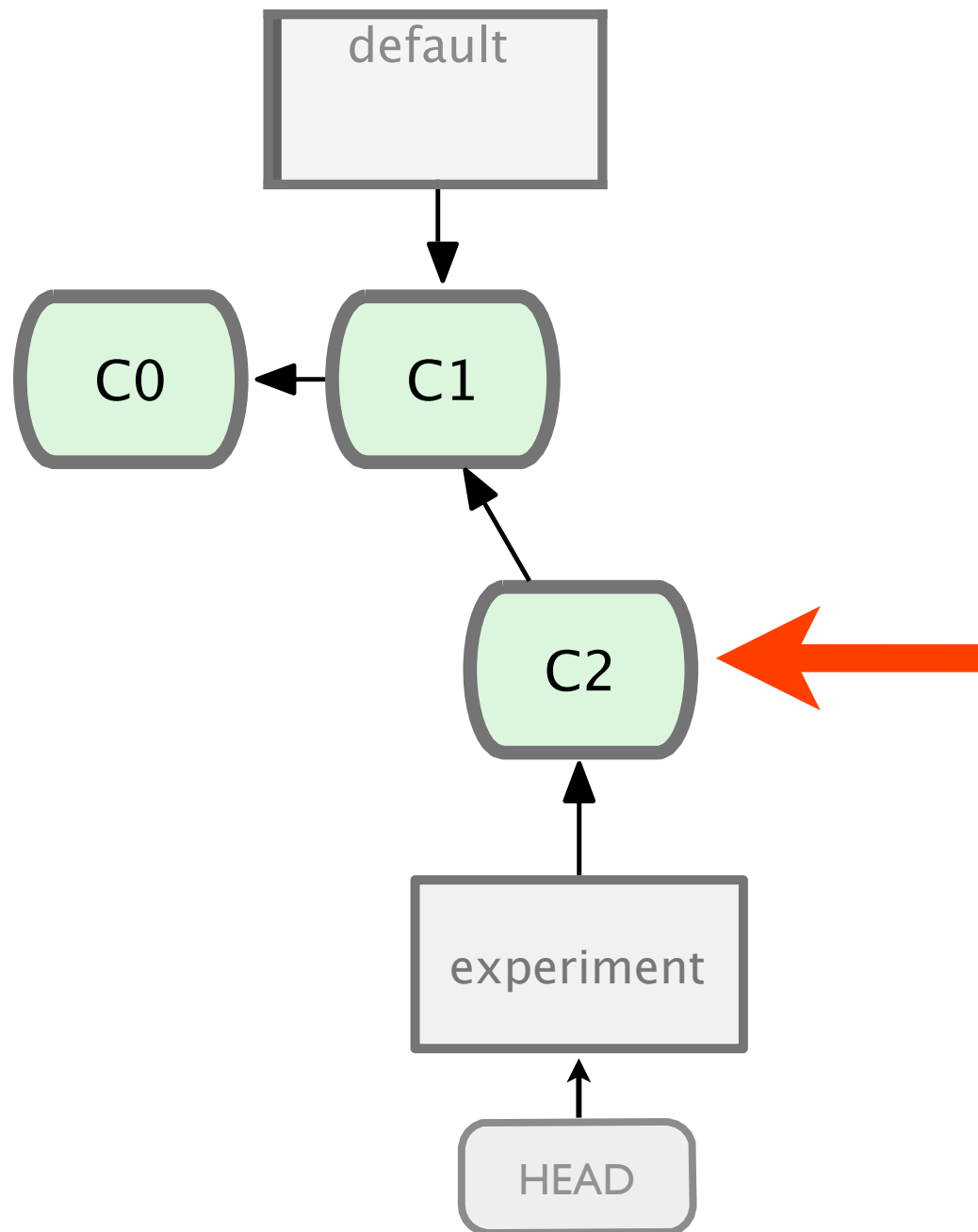
```
$ git branch  
* default  
experiment
```



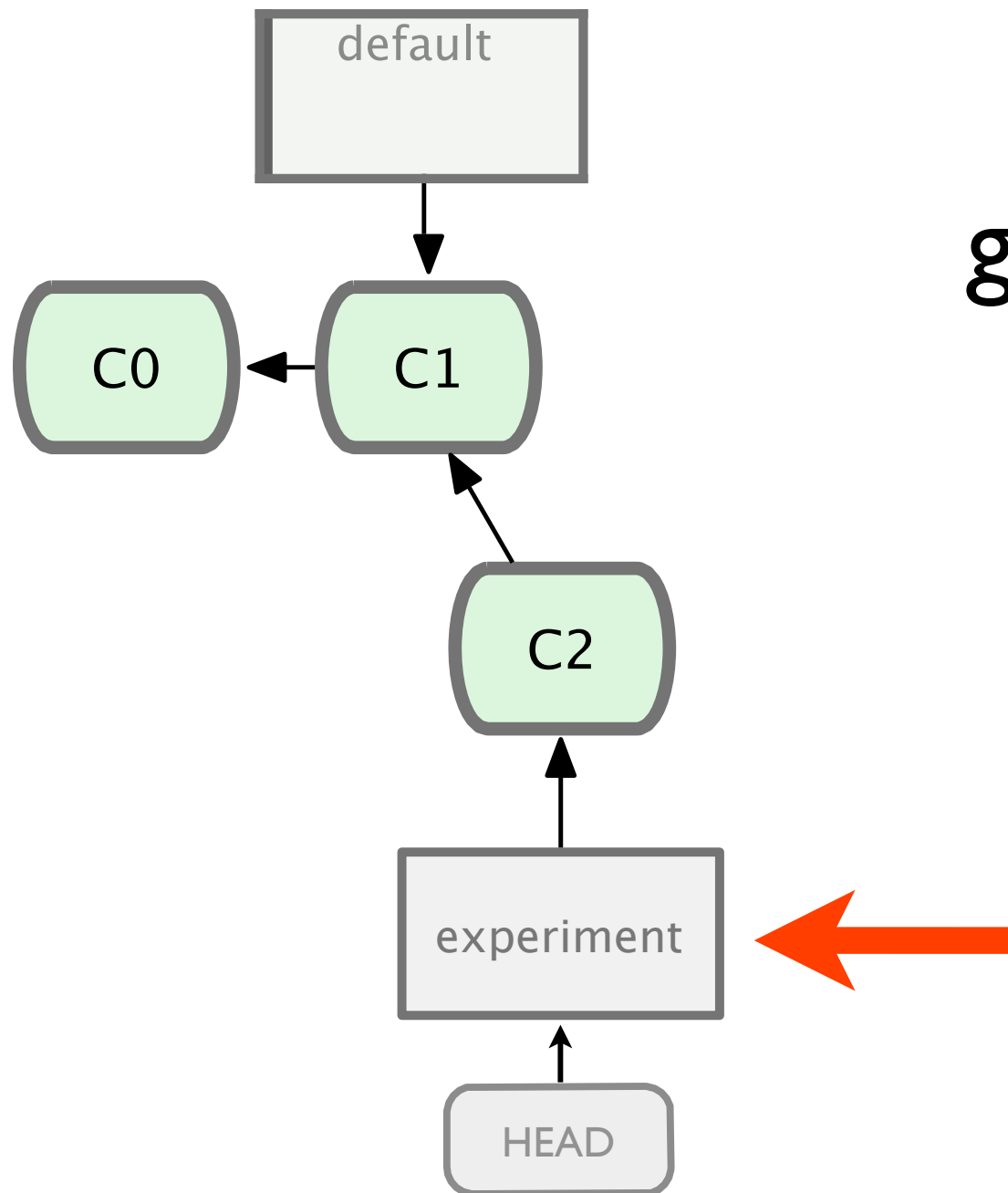
git checkout experiment



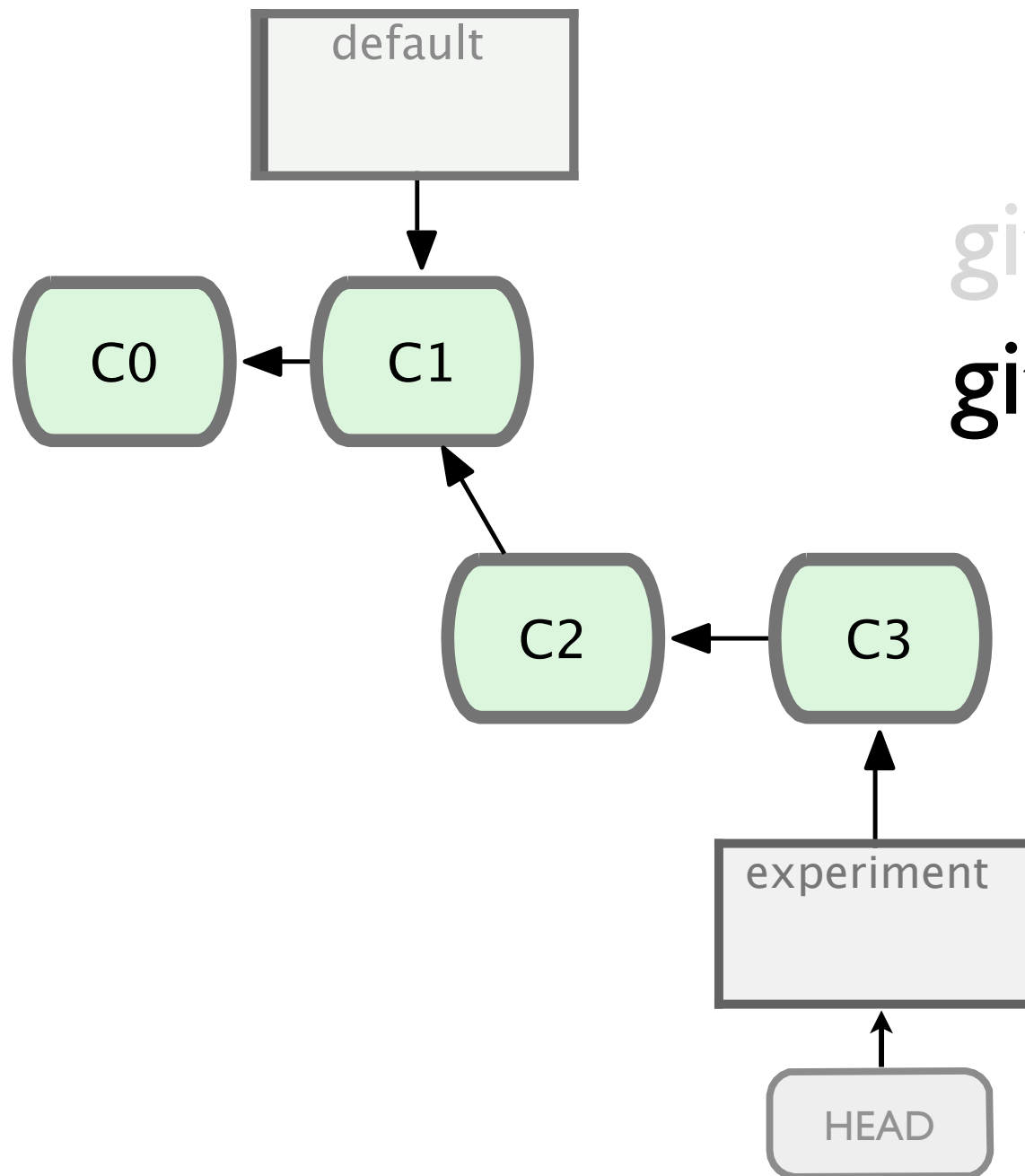
git commit



git commit

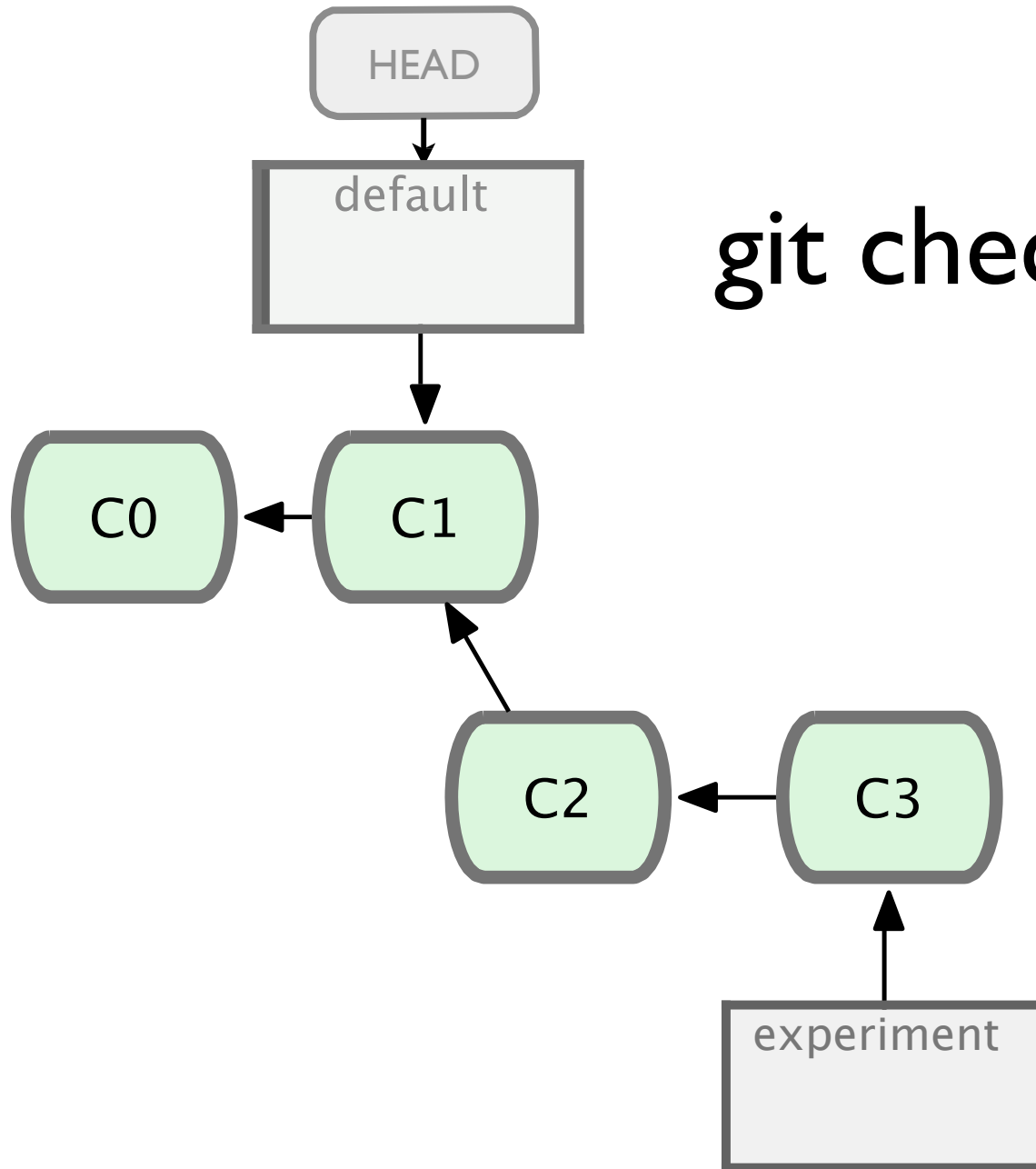


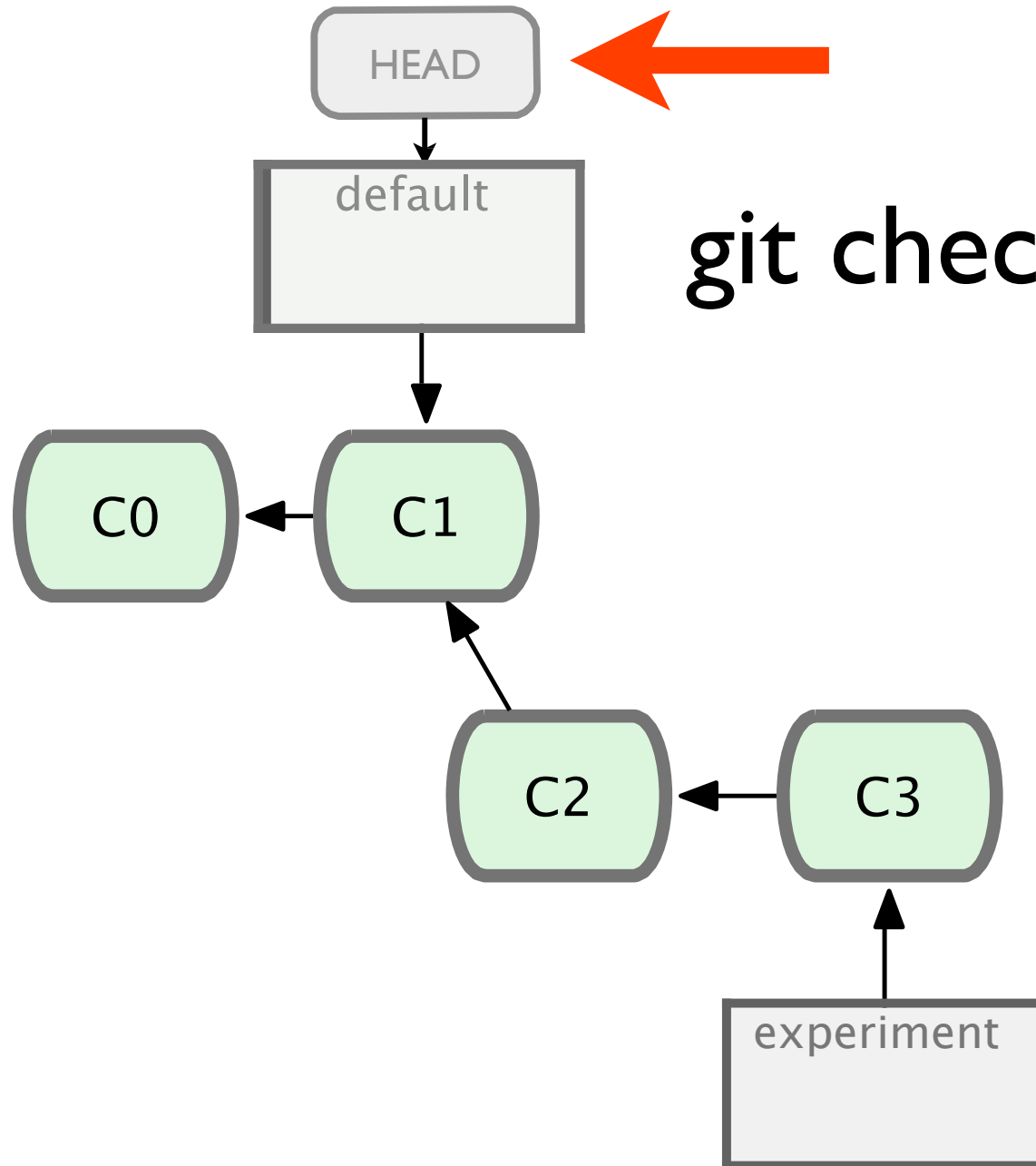
git commit



git commit  
git commit

# git checkout default

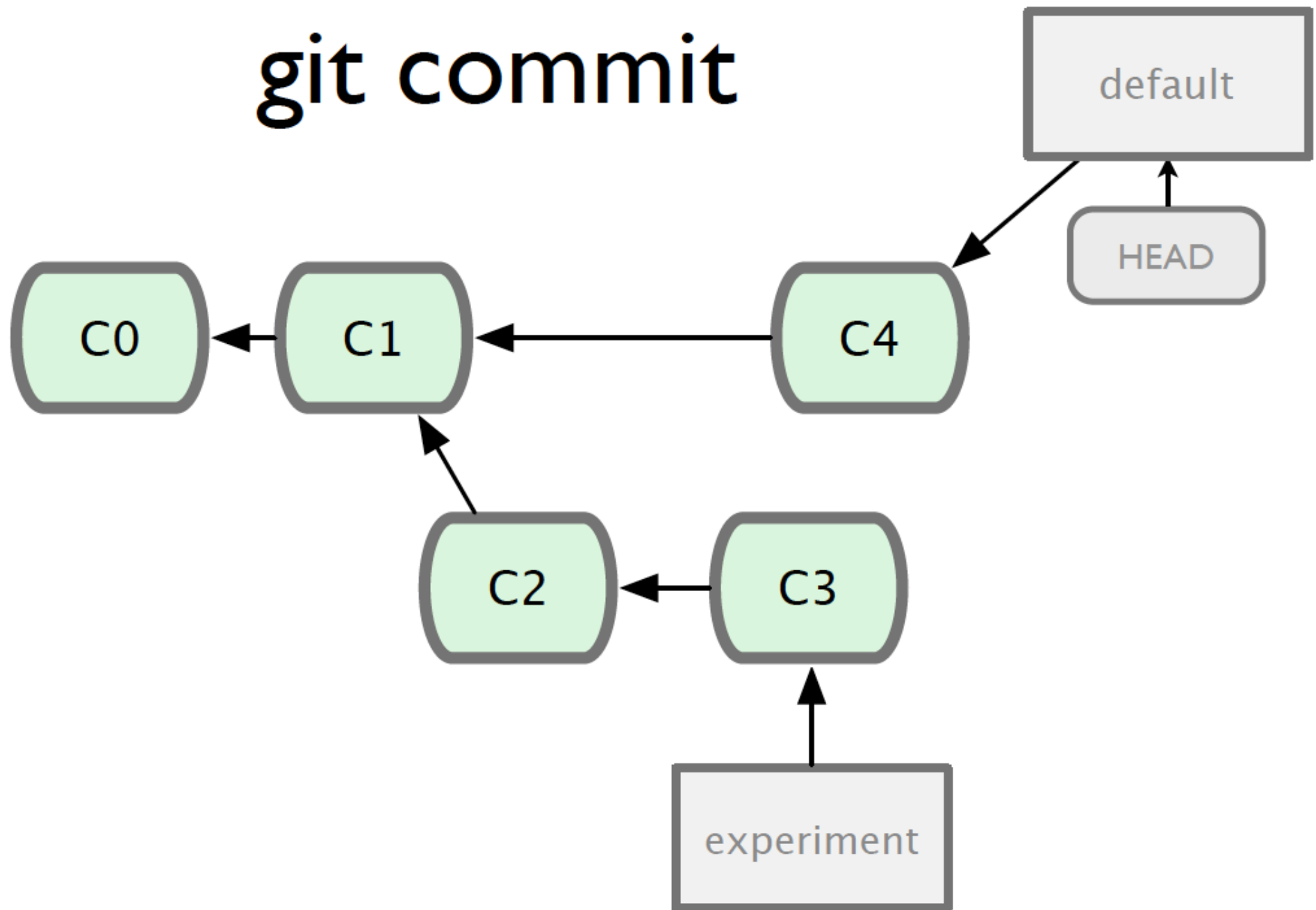




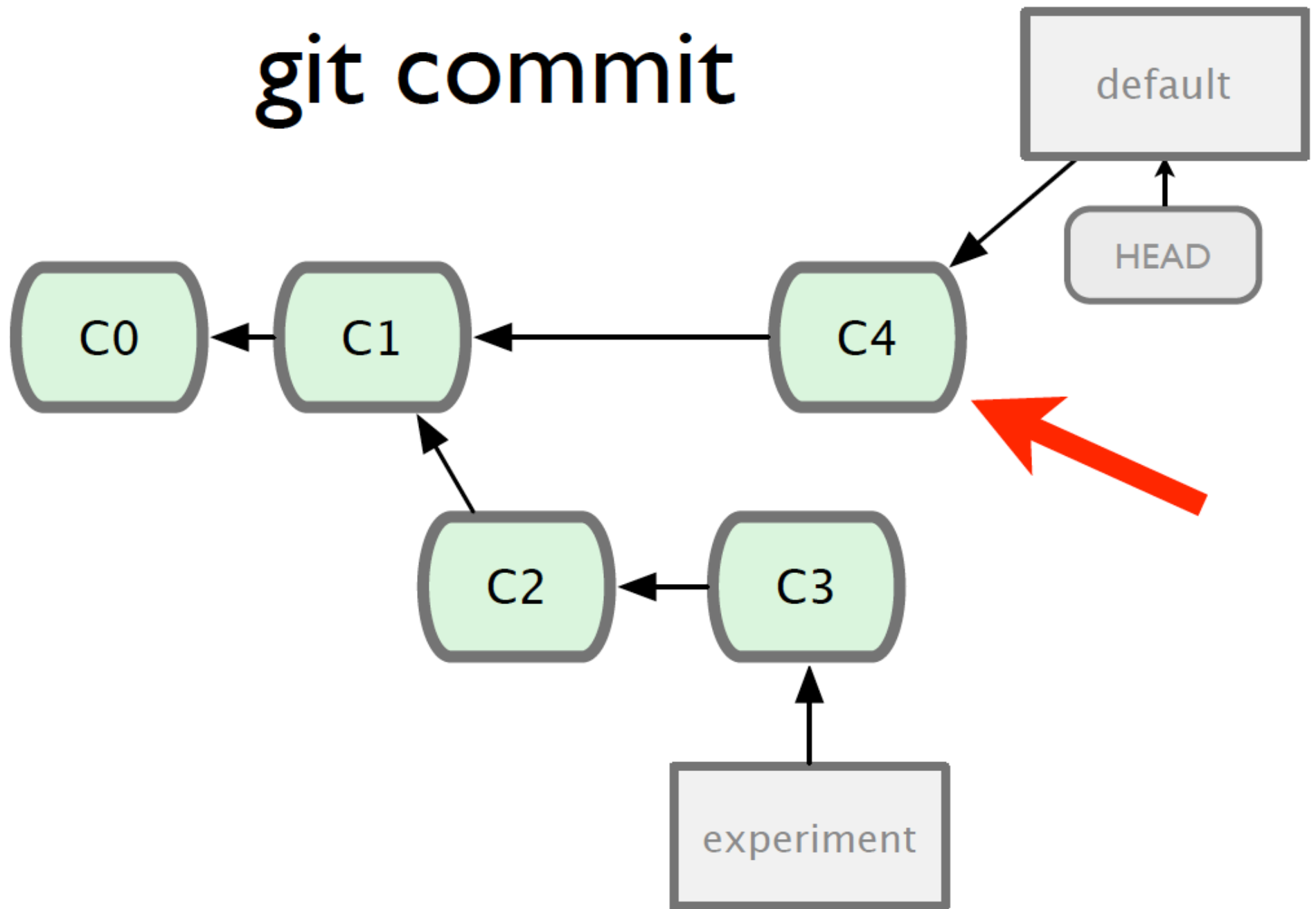
git checkout default



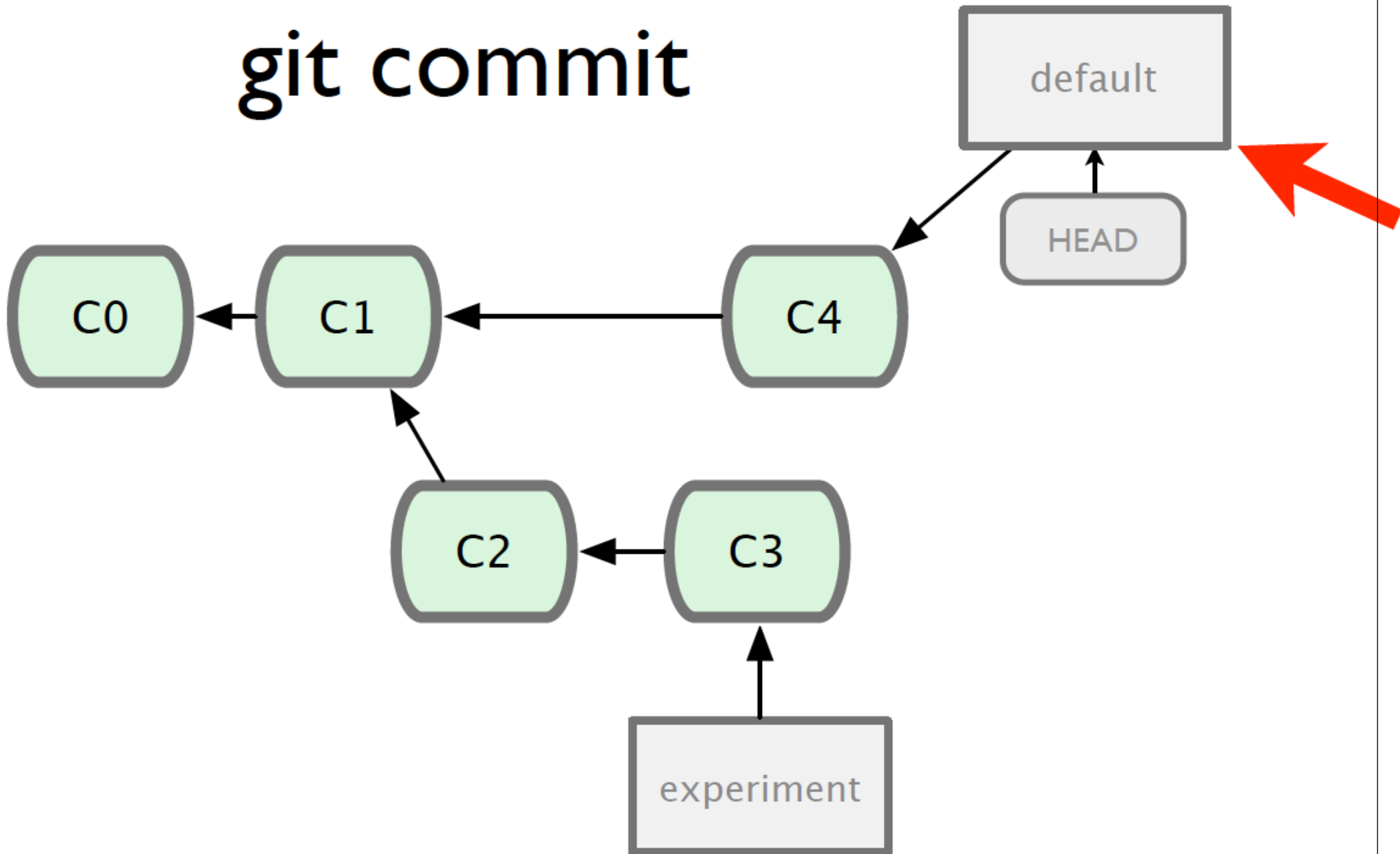
# git commit

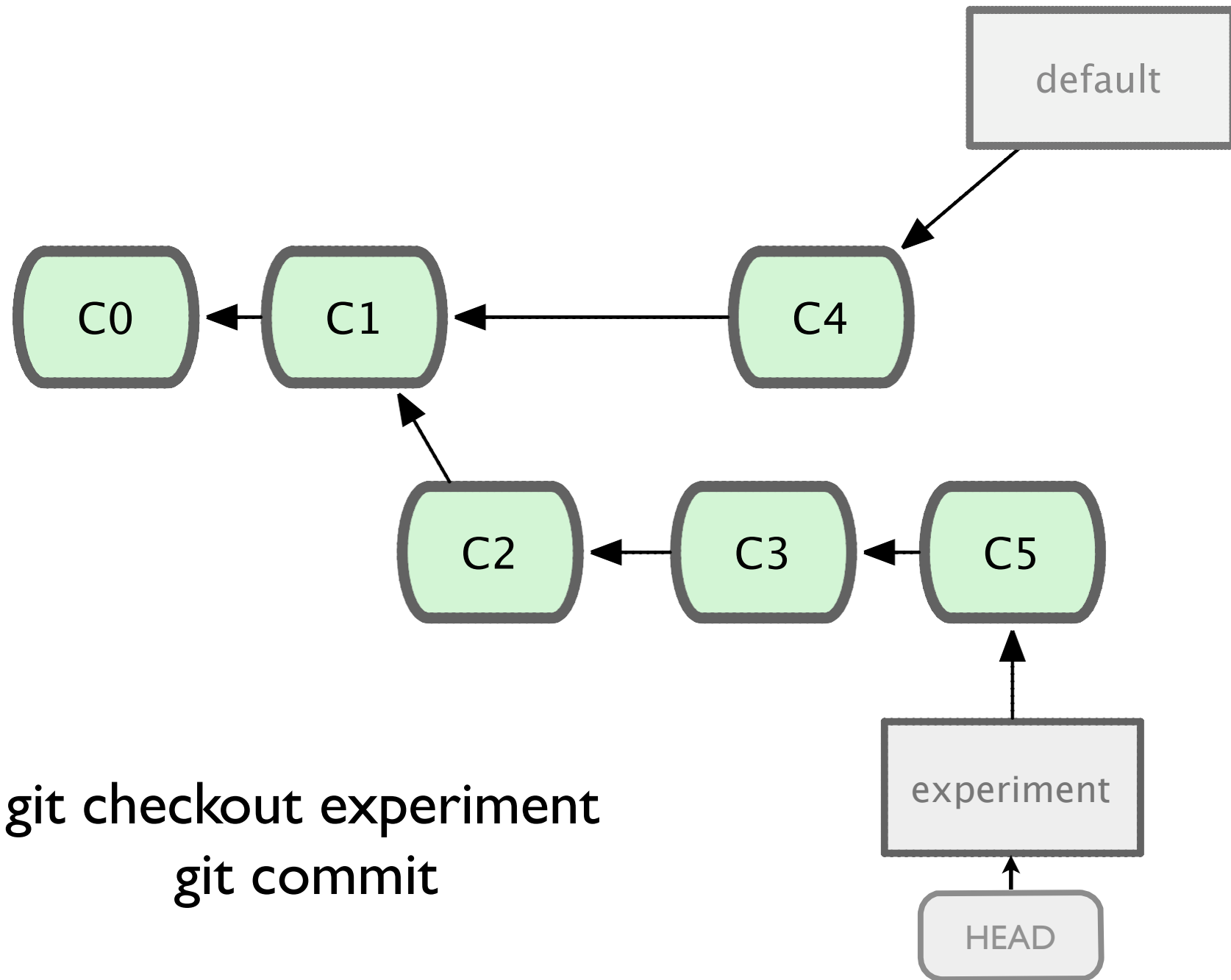


# git commit



# git commit

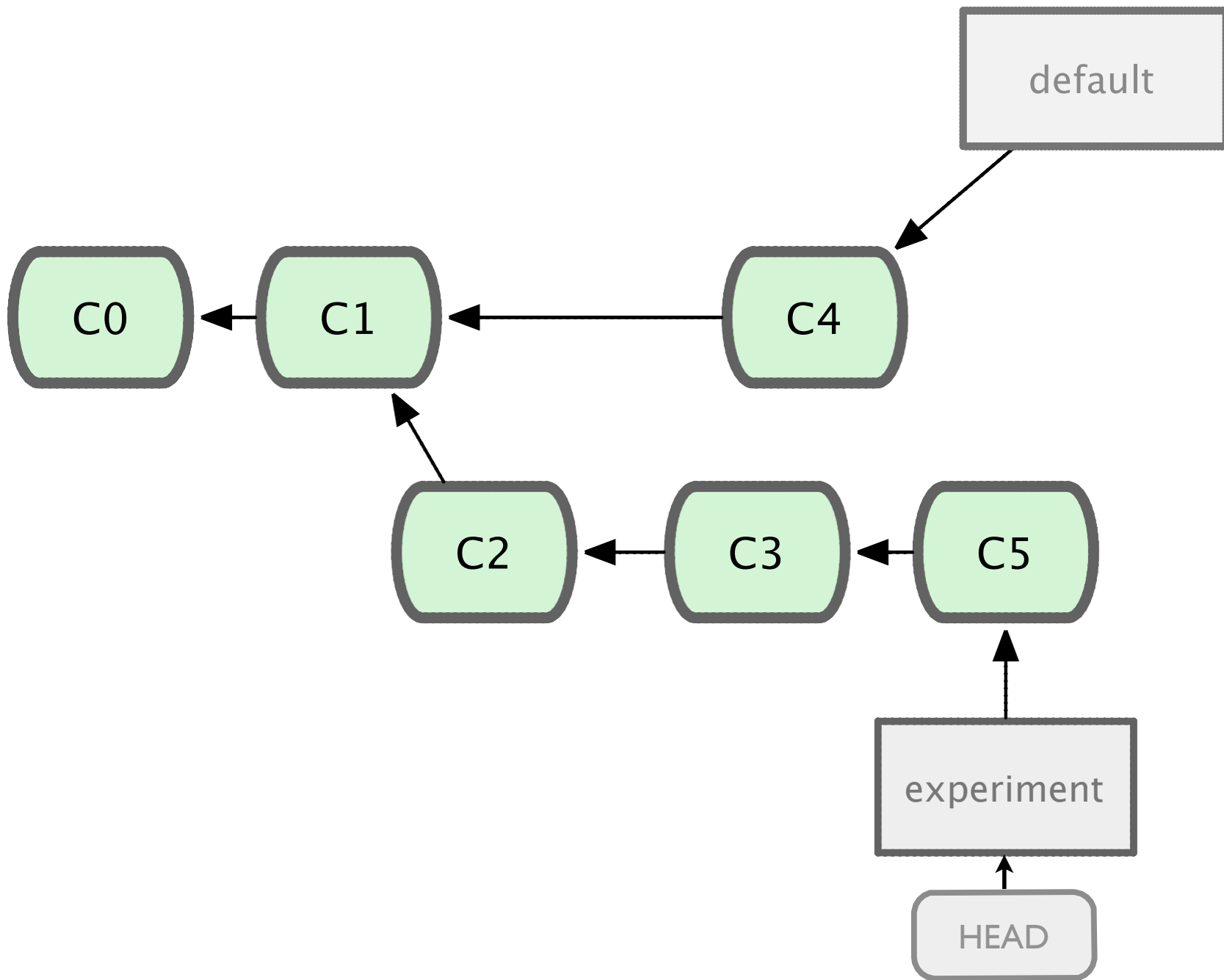


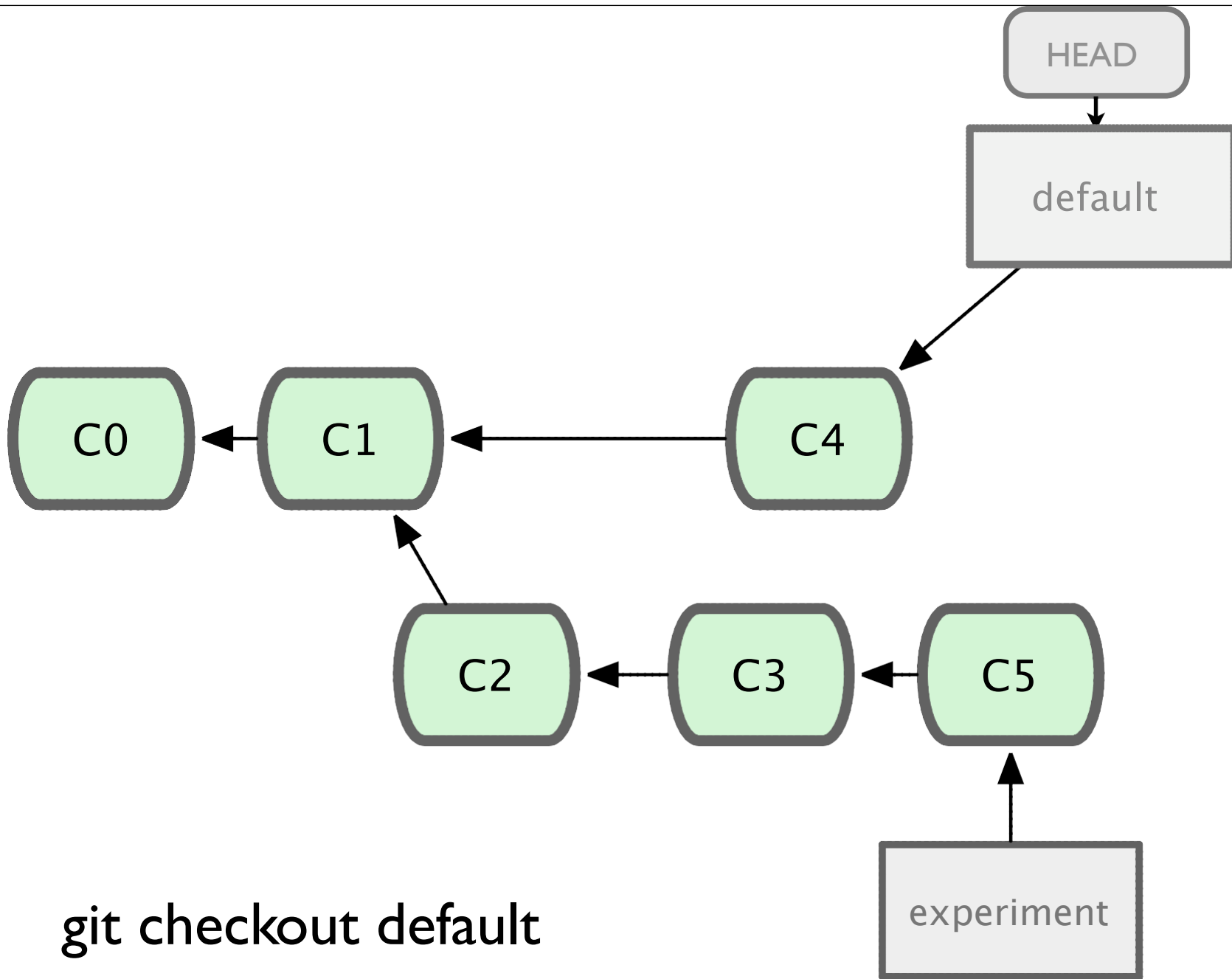


merging

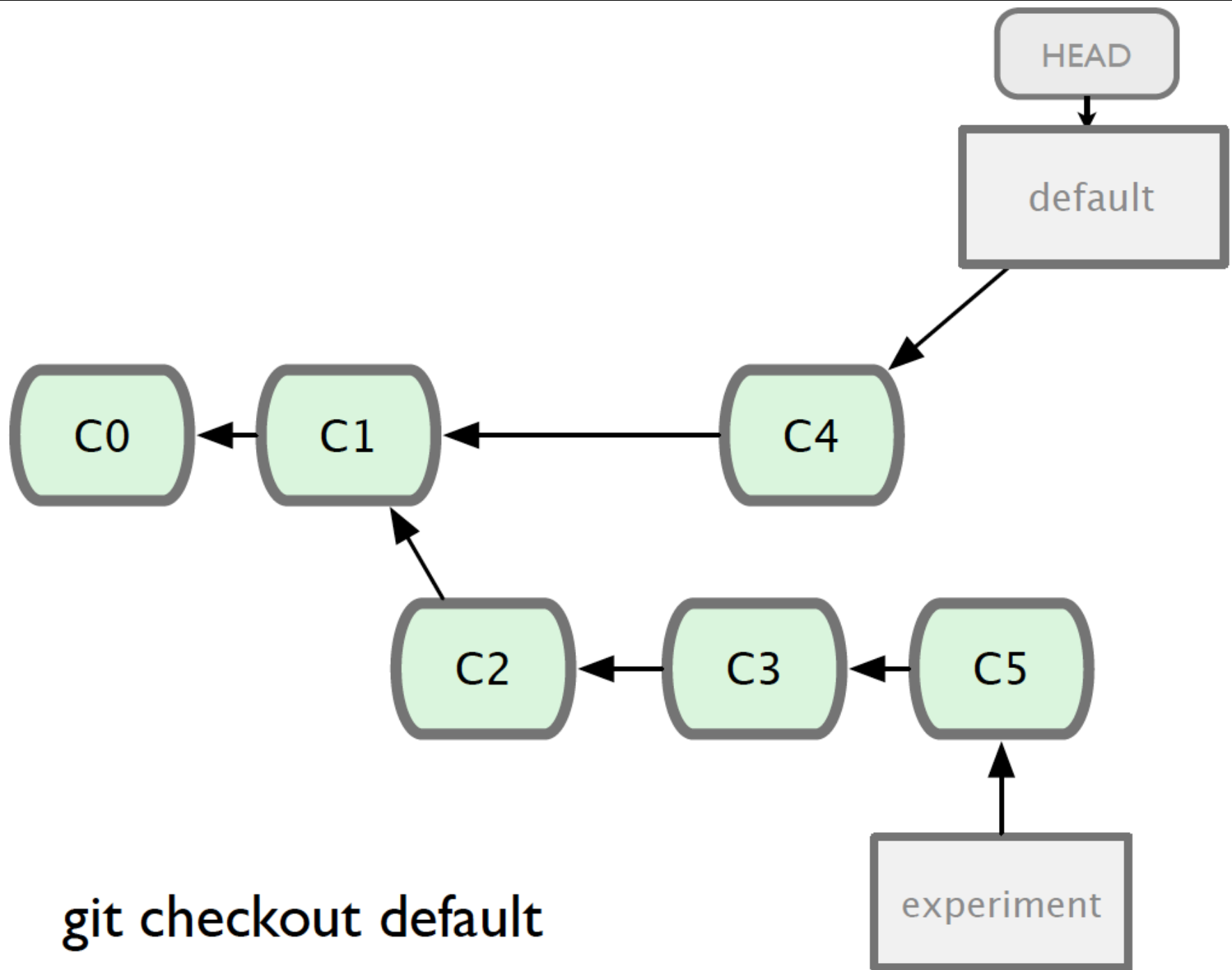
# git merge

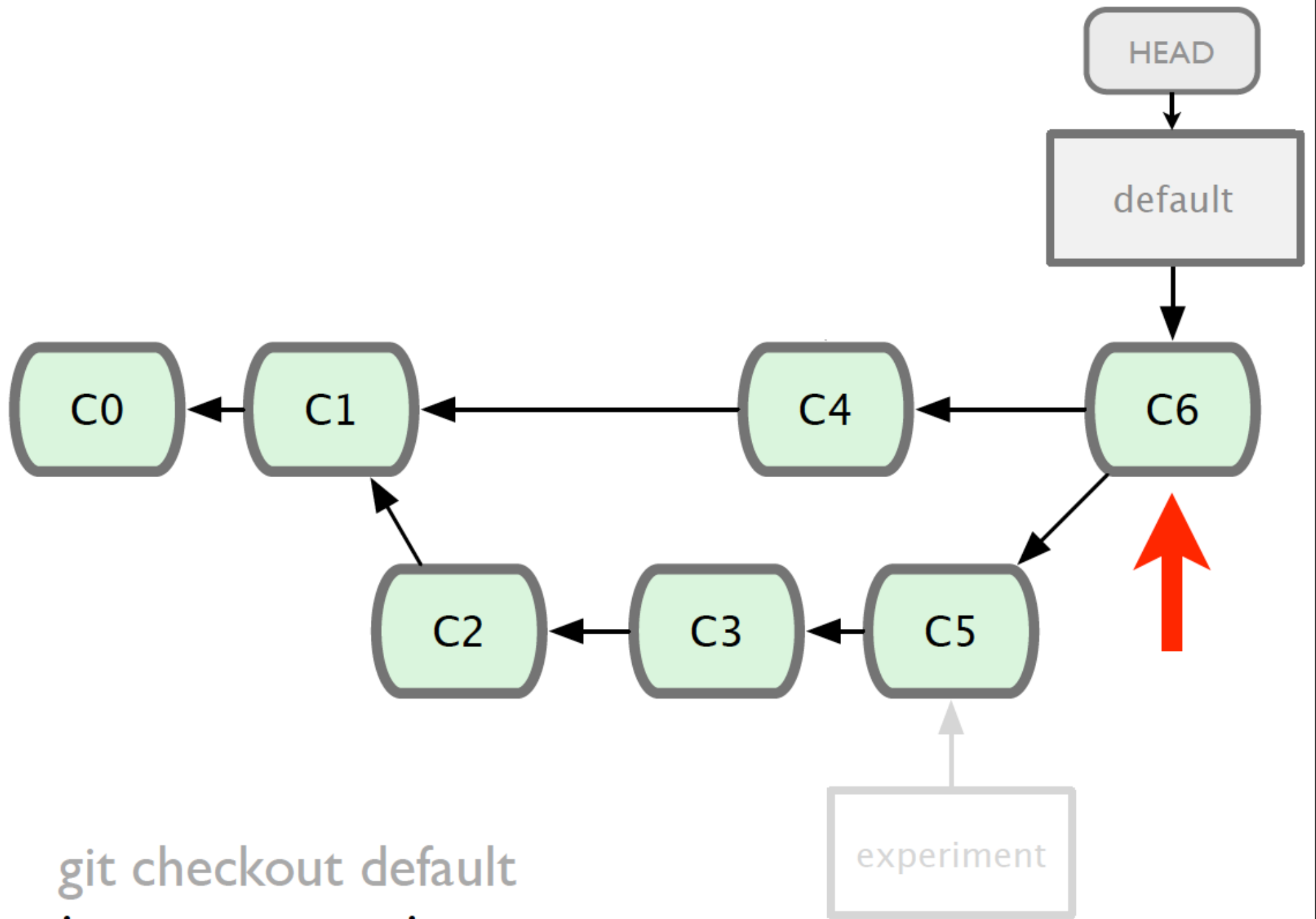
apply edits on one branch to another  
branch

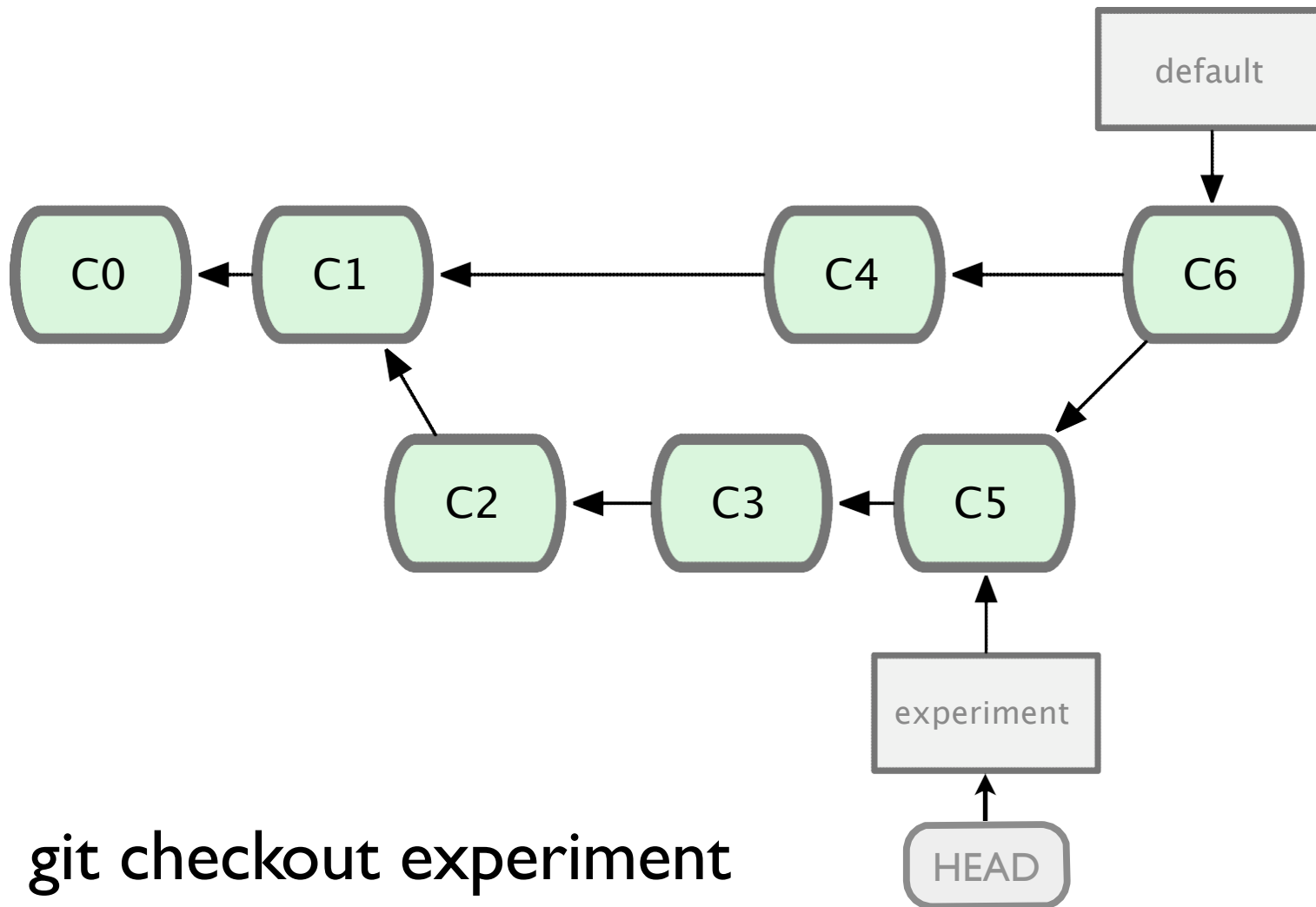


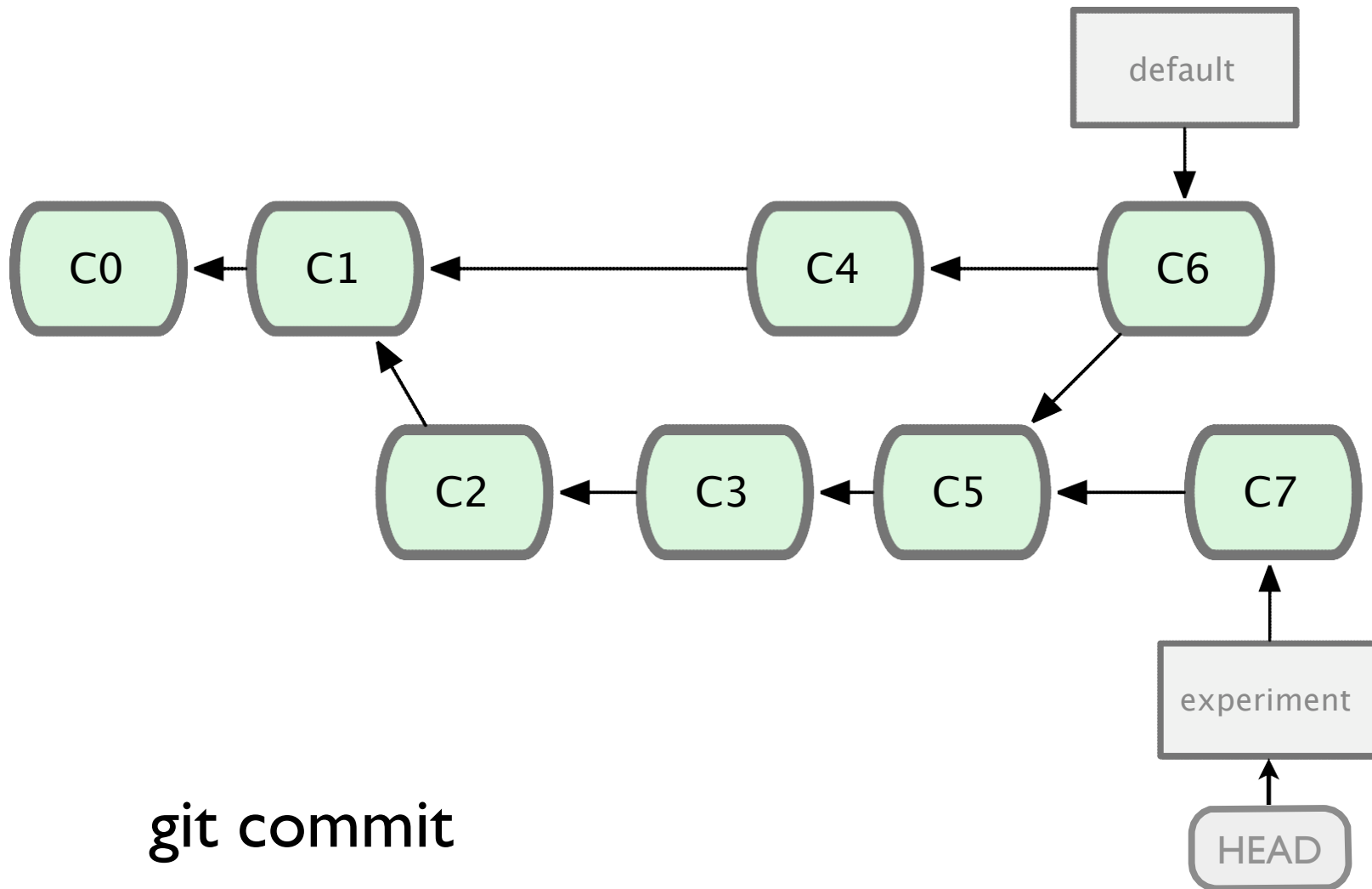


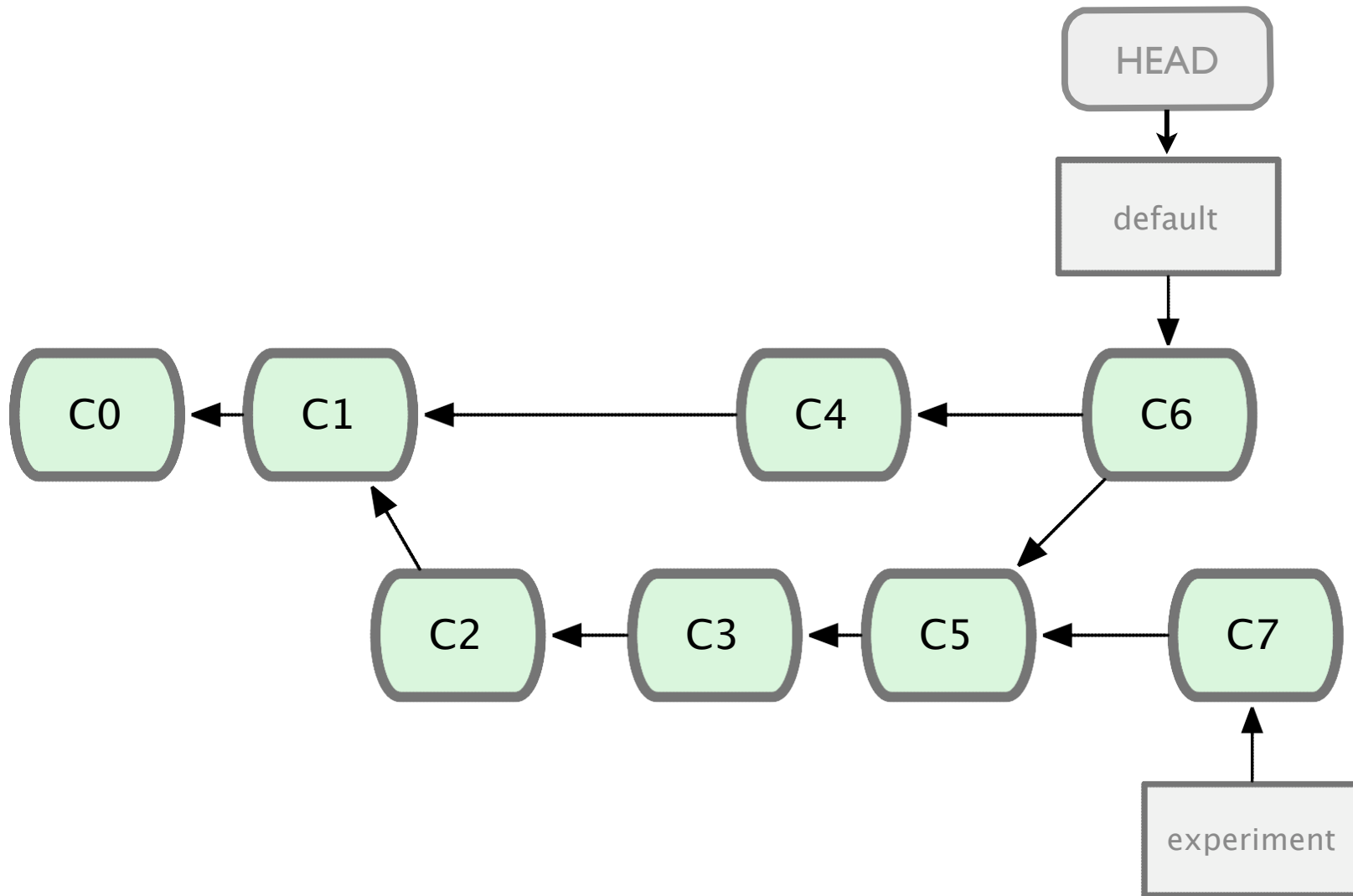




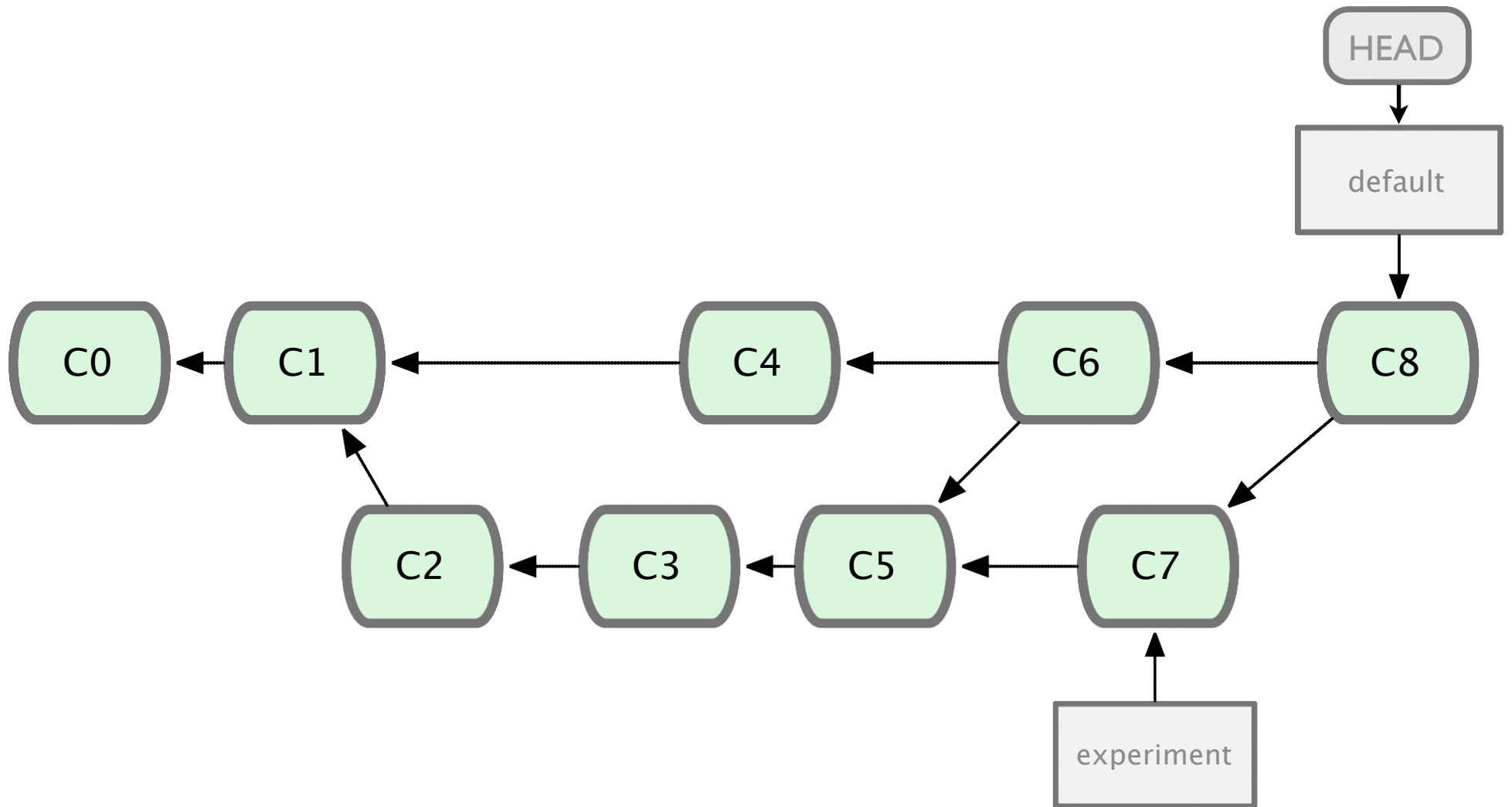








git checkout default



git merge experiment

**Lots more to merging**

**AND...**

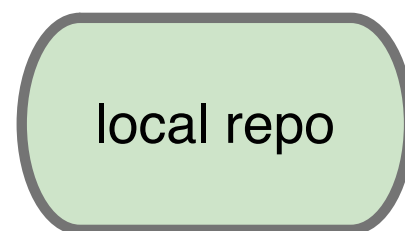
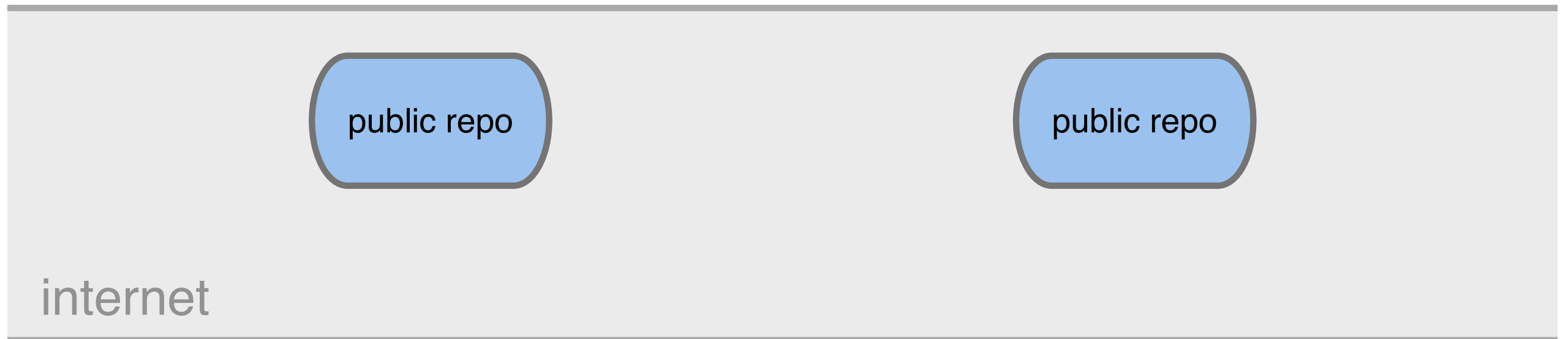
**Merging cannot always  
be done cleanly and  
may need your  
interaction to succeed.**

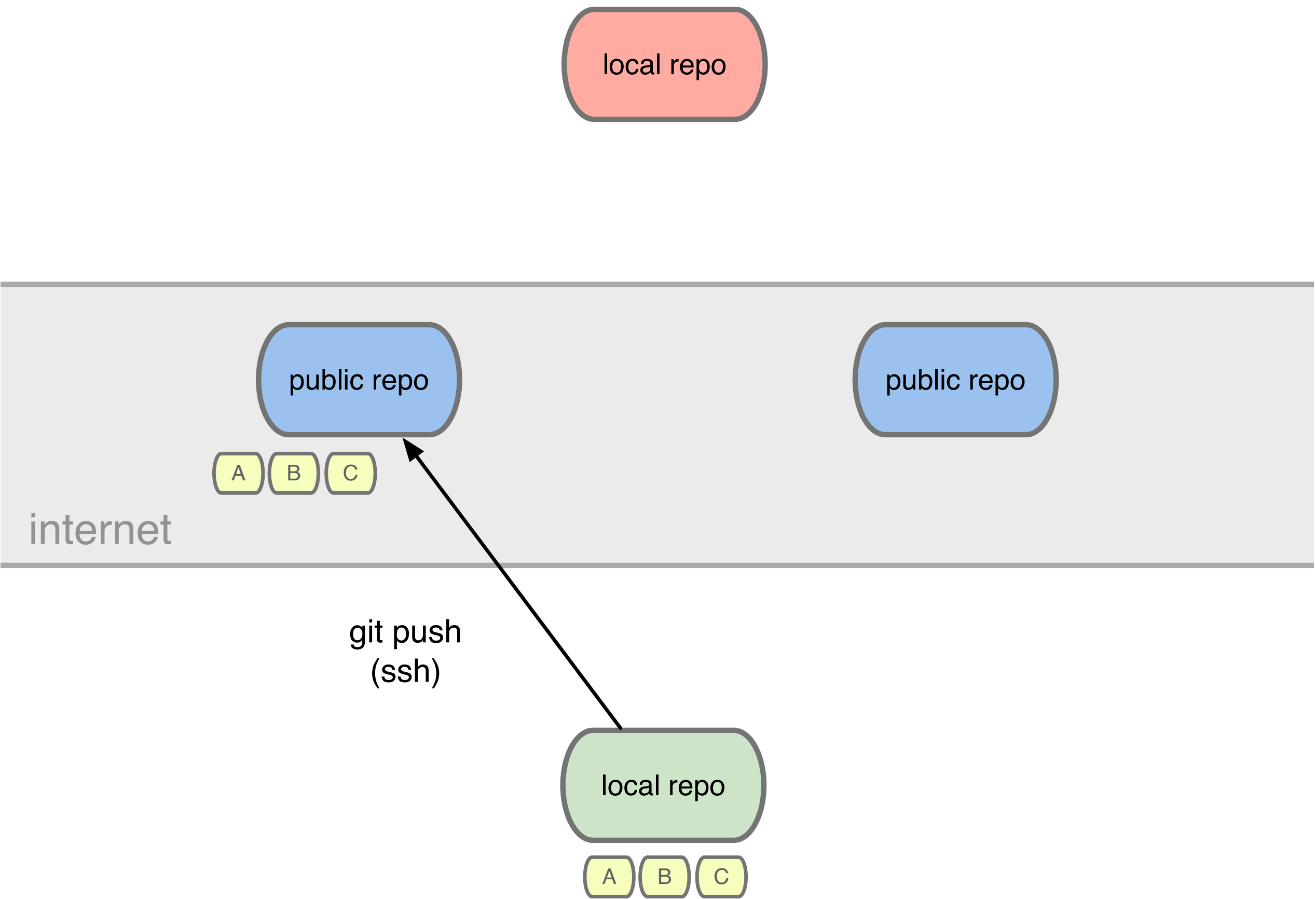
See the git resources for details.

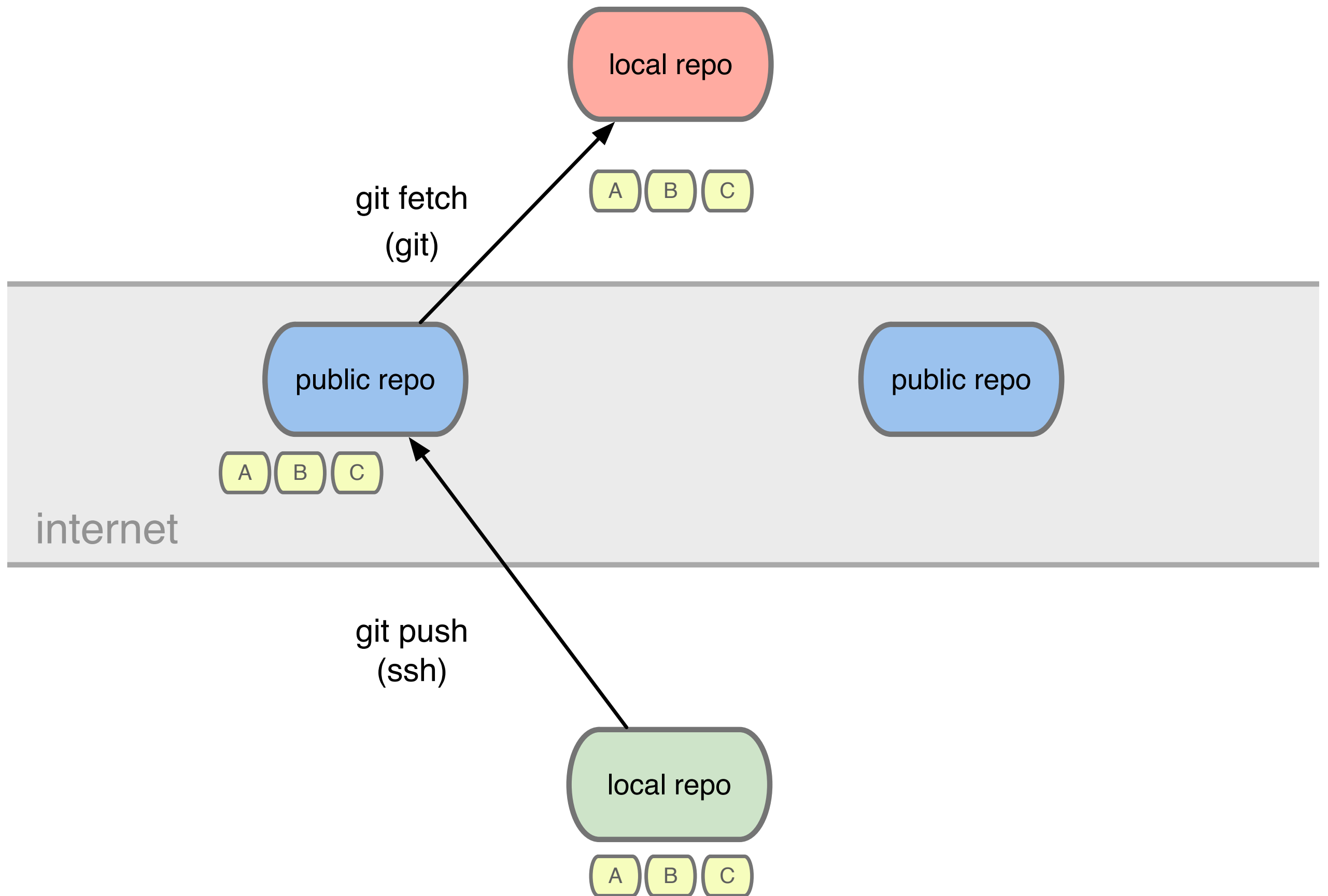
# Remotes

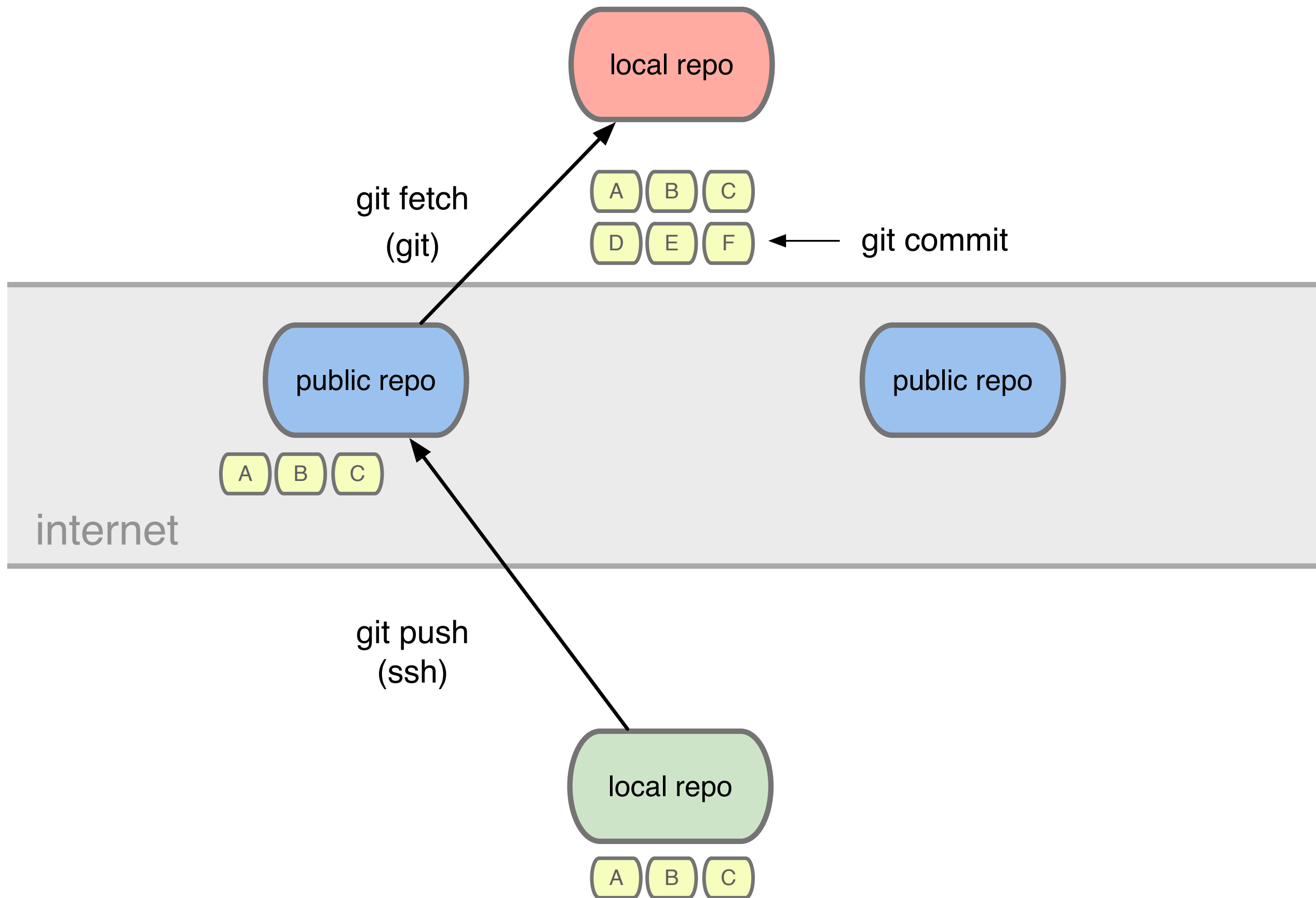


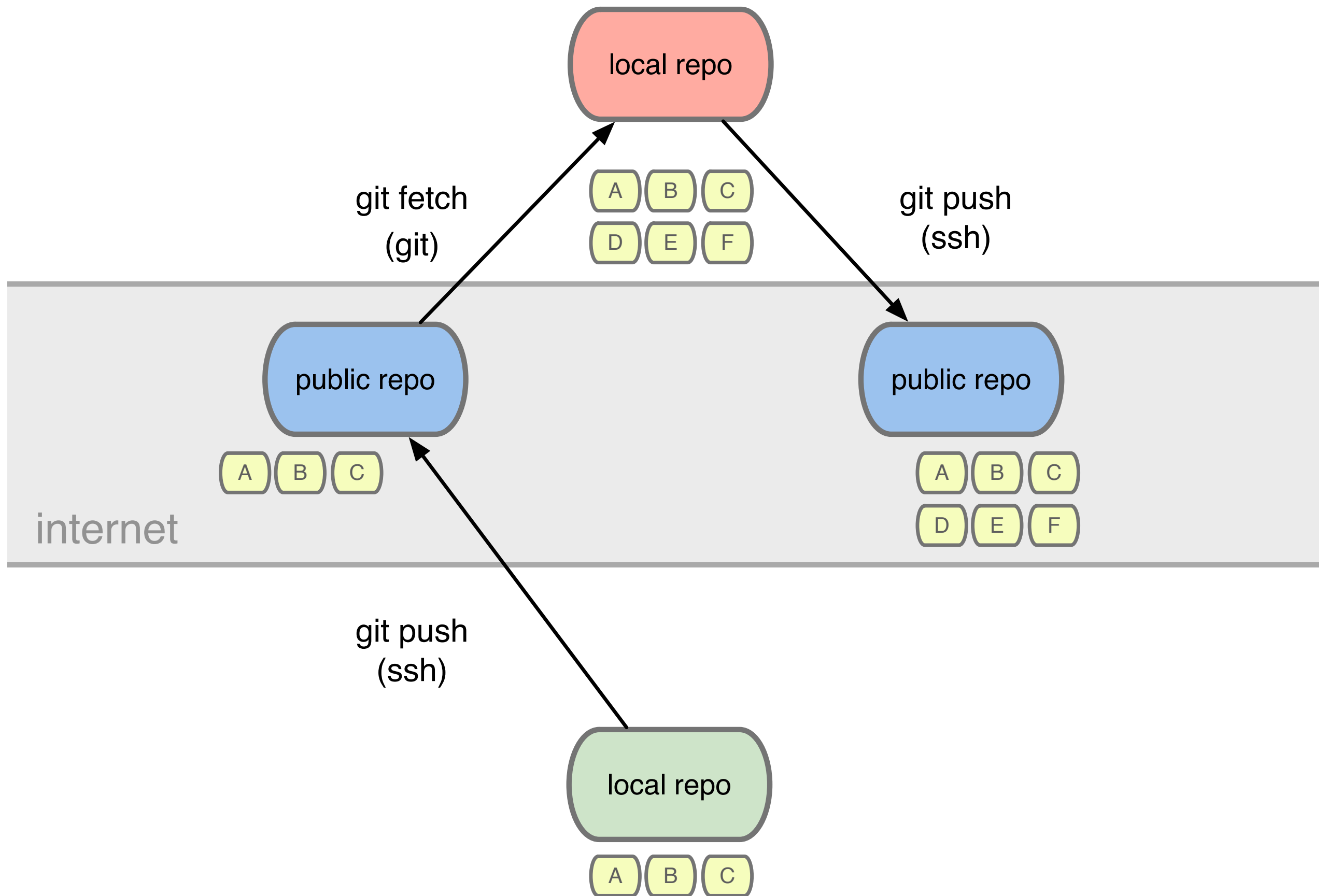
# DistributedWorkflow

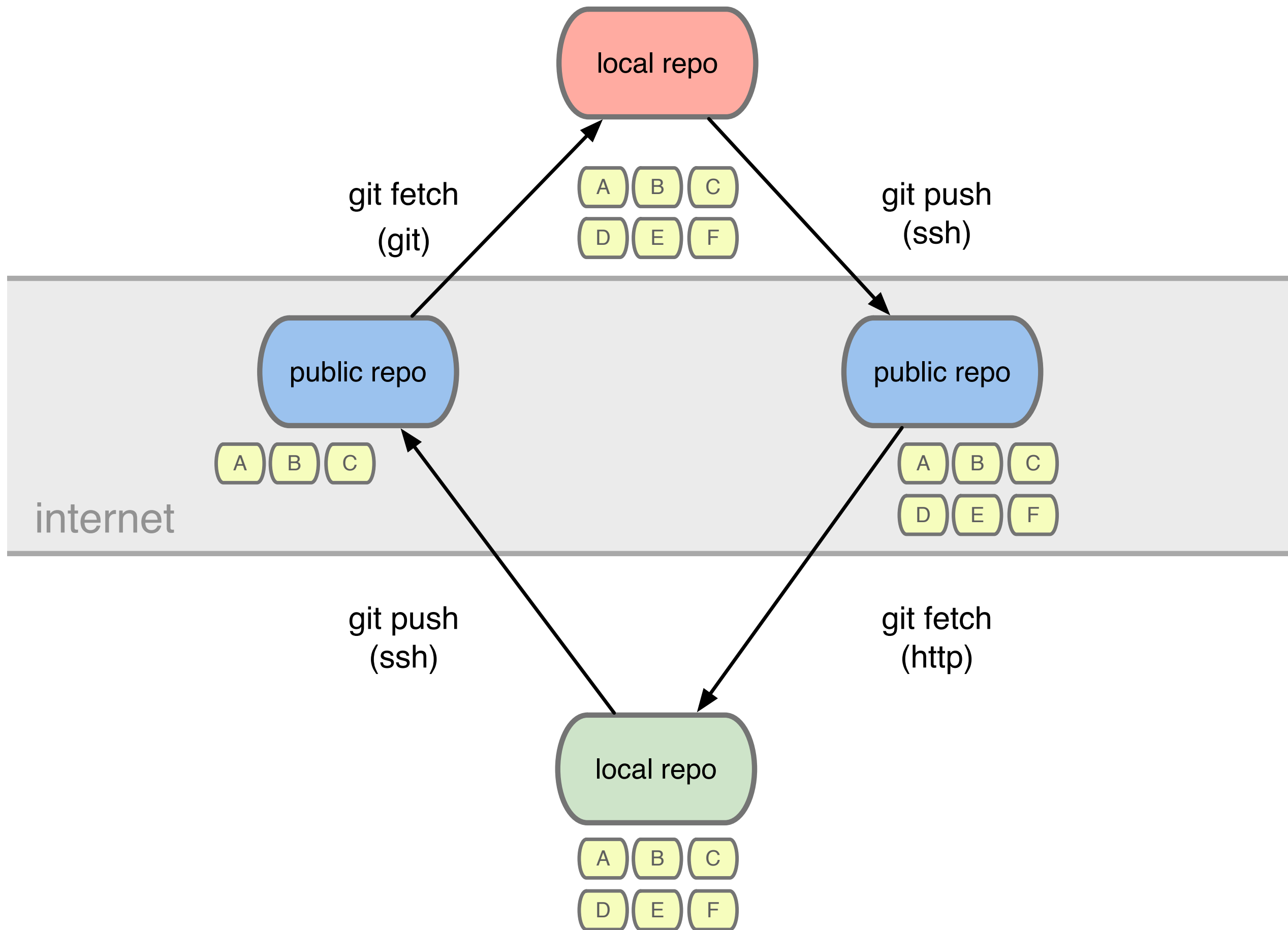


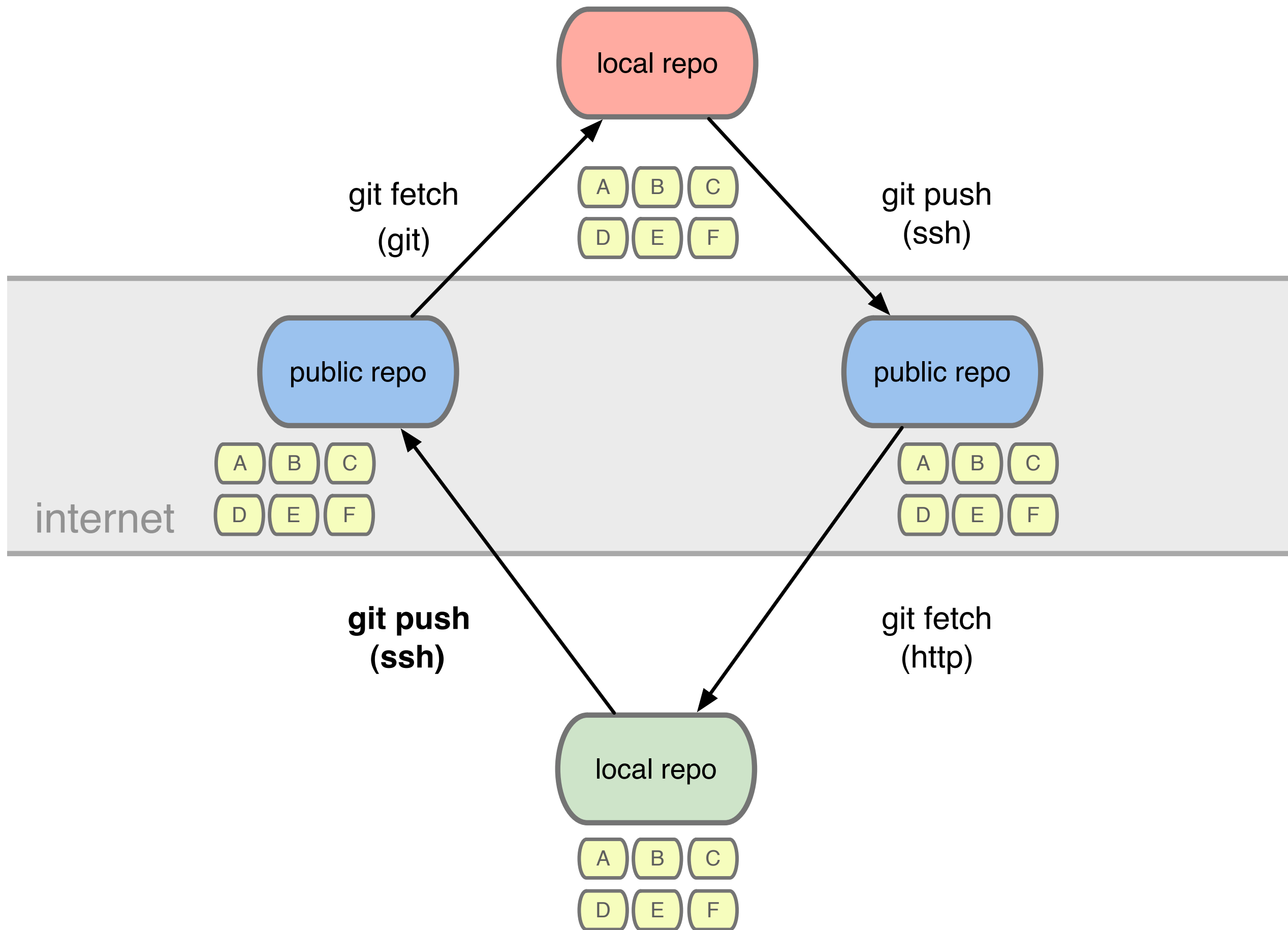










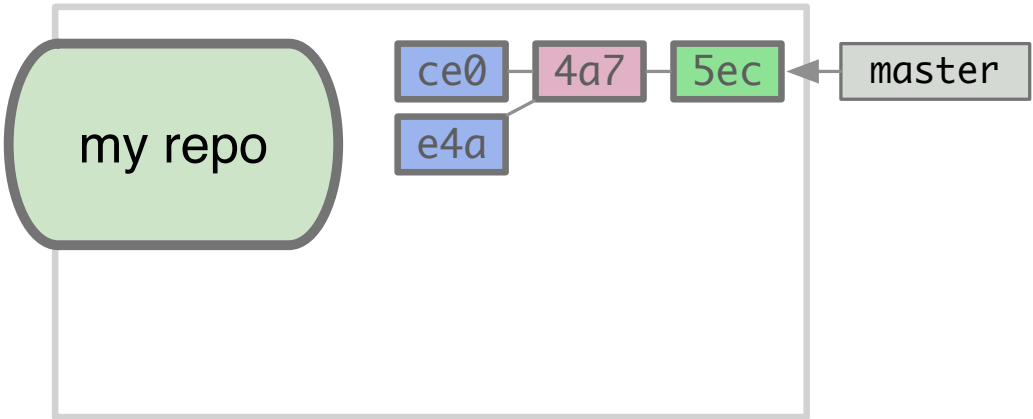
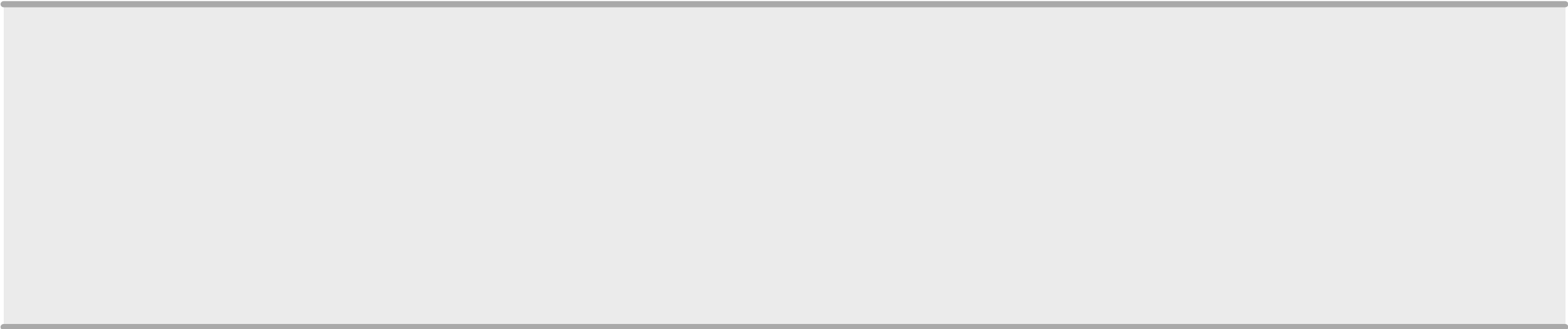




# Multiple Remotes

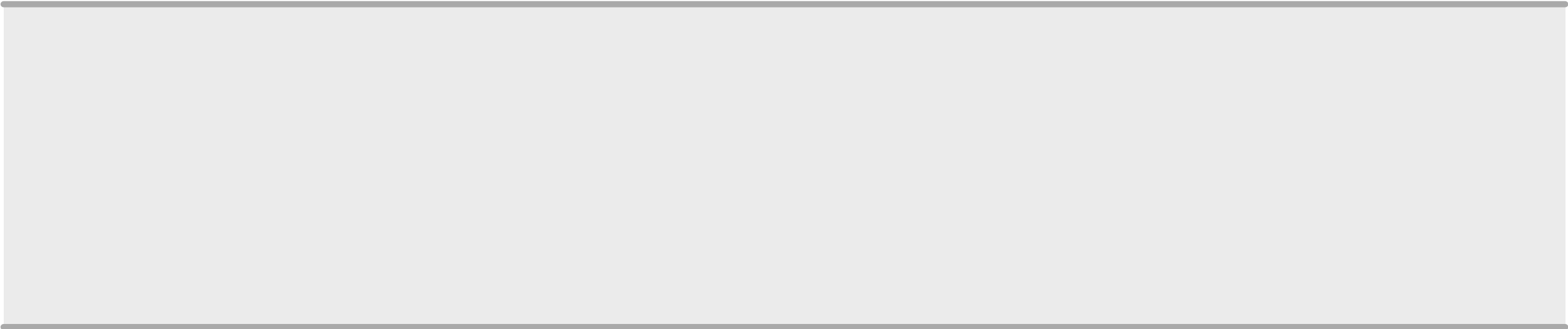
developer  
nick

developer  
jessica

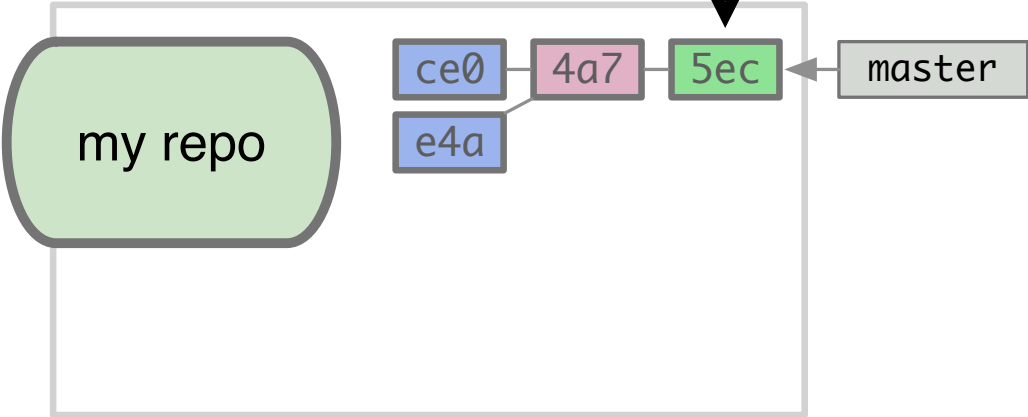


developer  
nick

developer  
jessica

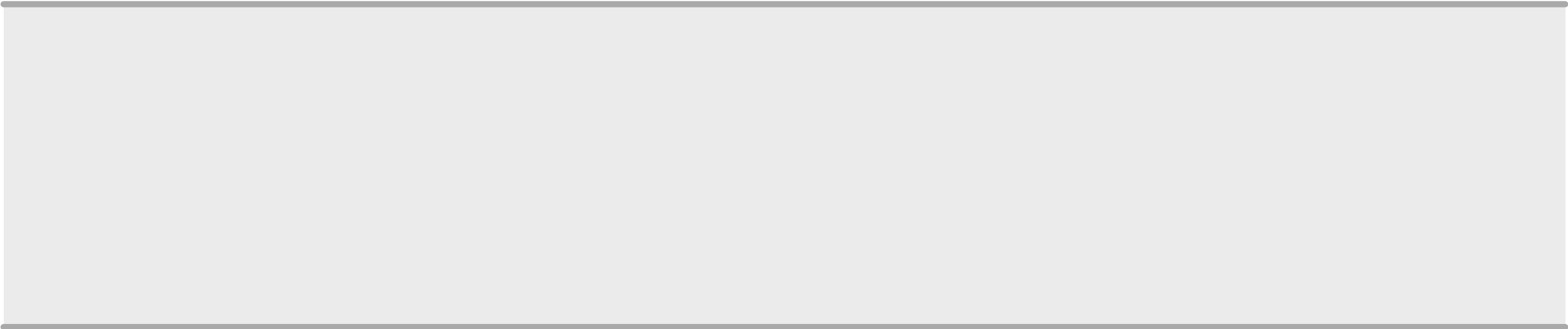


commit

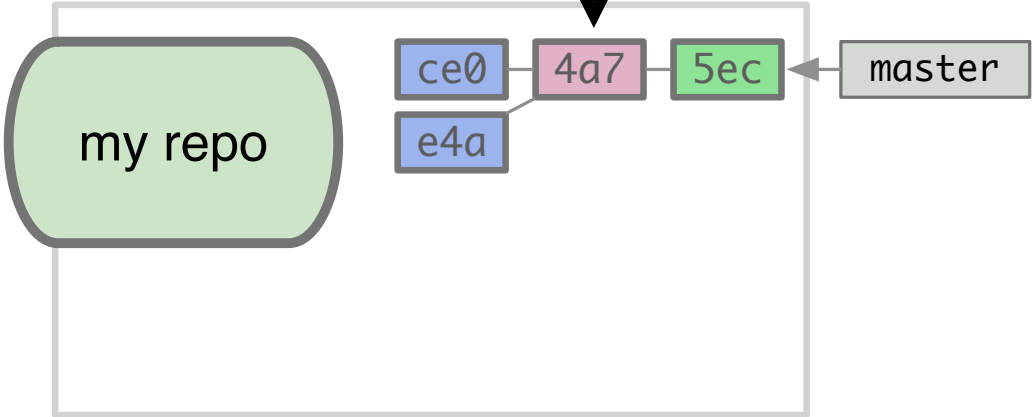


developer  
nick

developer  
jessica

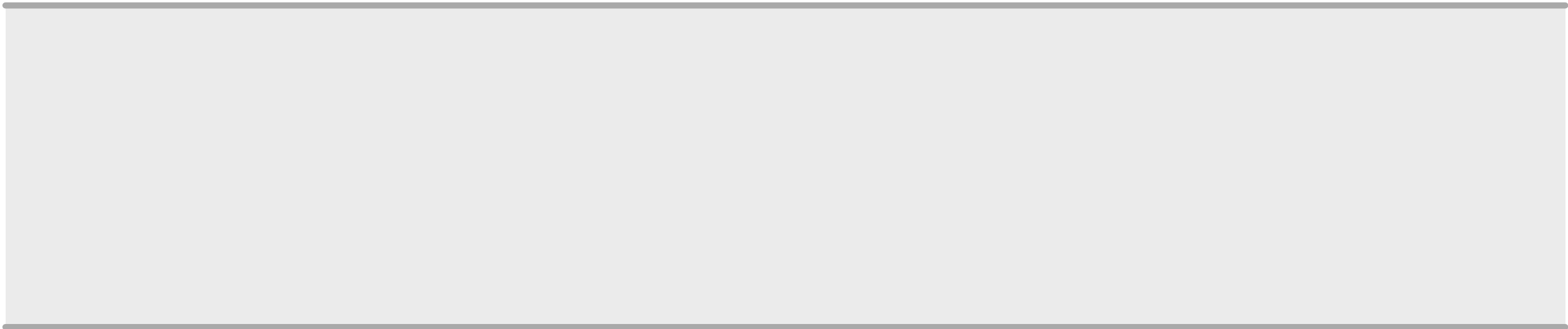


tree

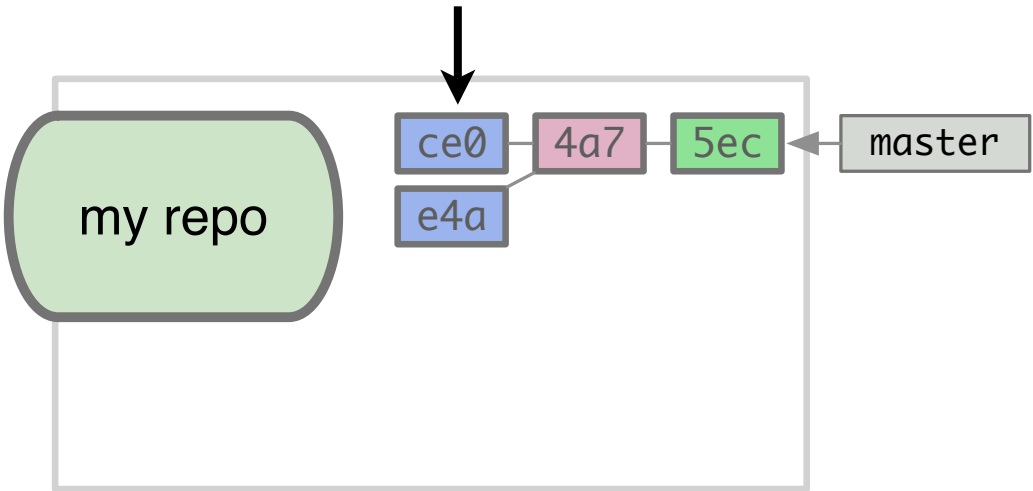


developer  
nick

developer  
jessica

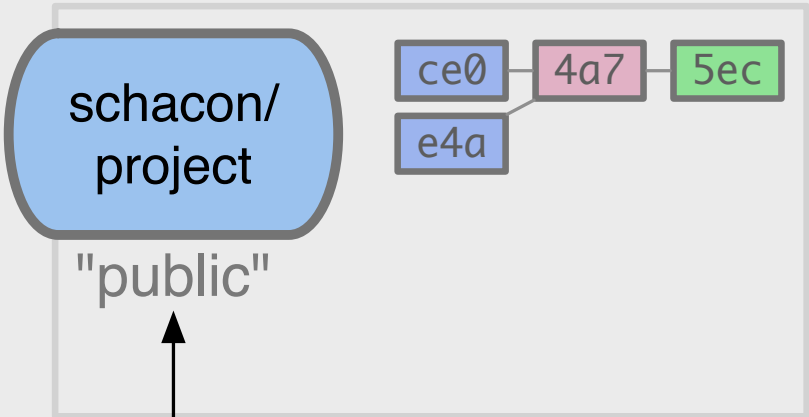


blobs

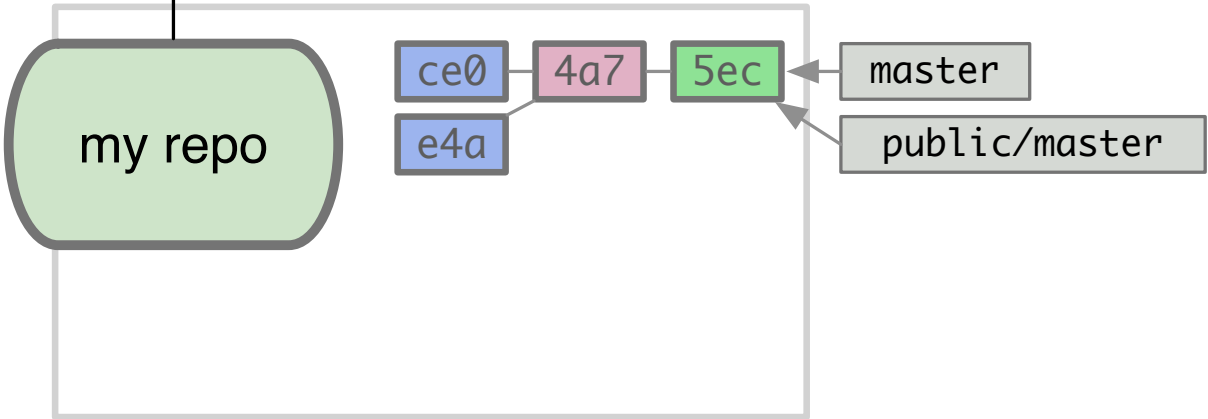


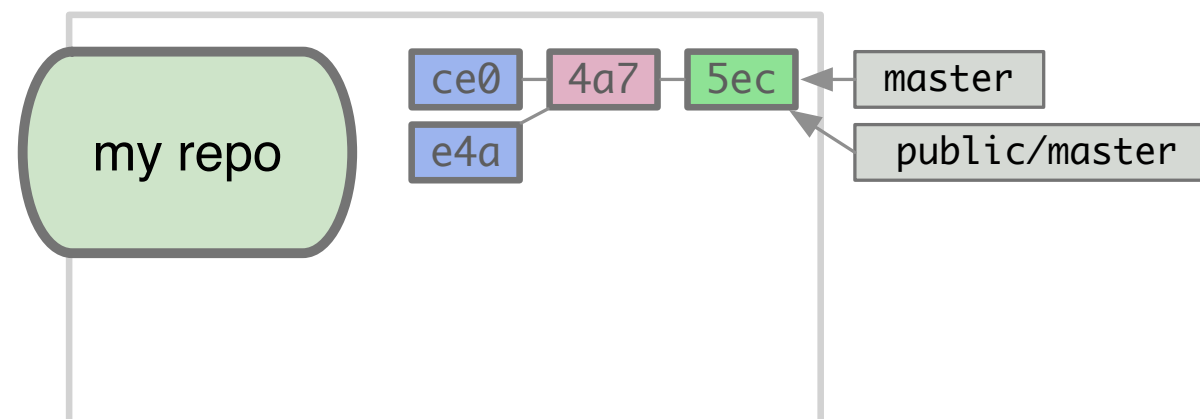
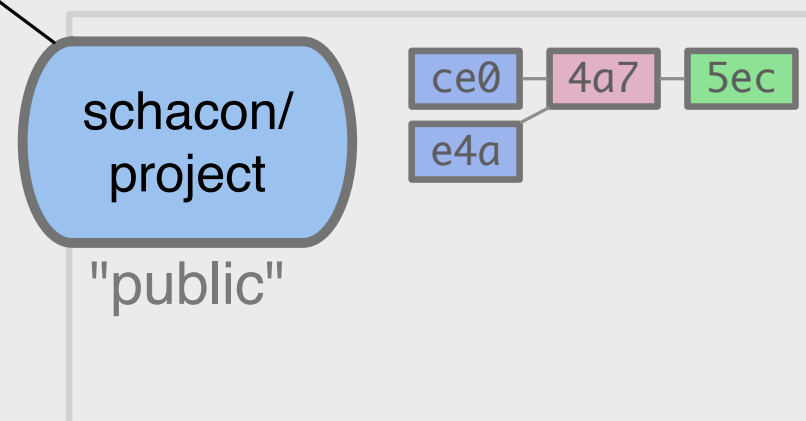
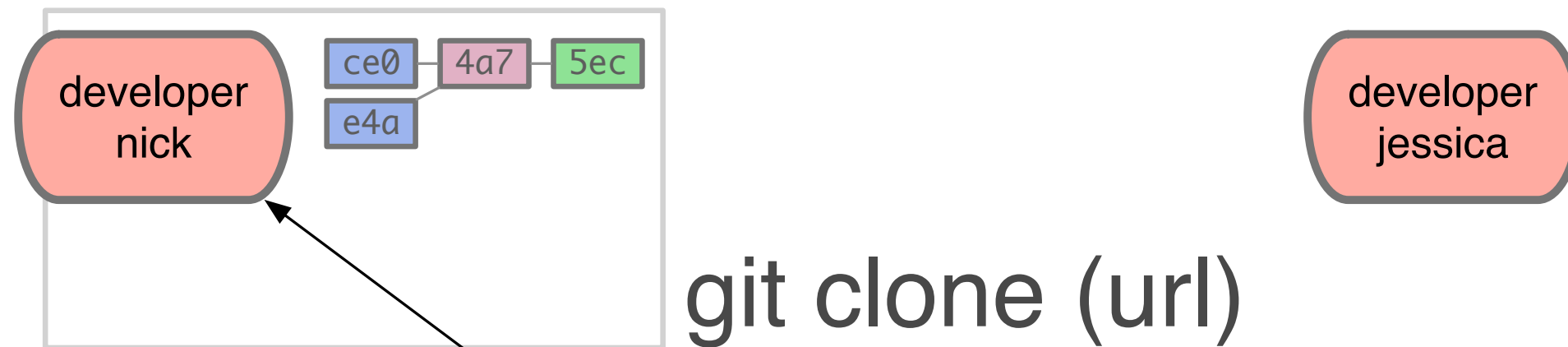
developer  
nick

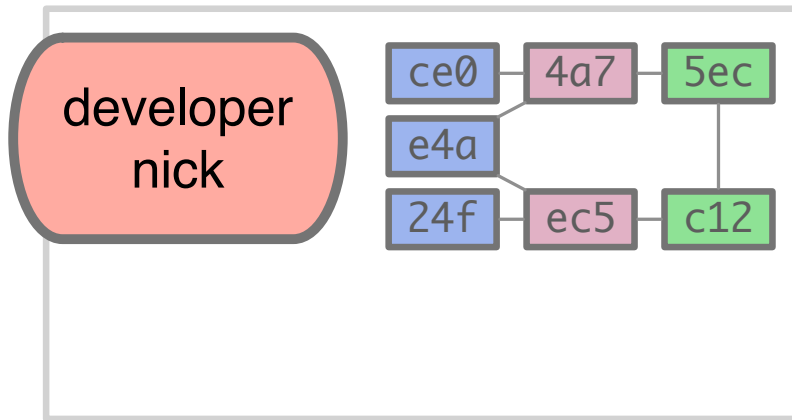
developer  
jessica



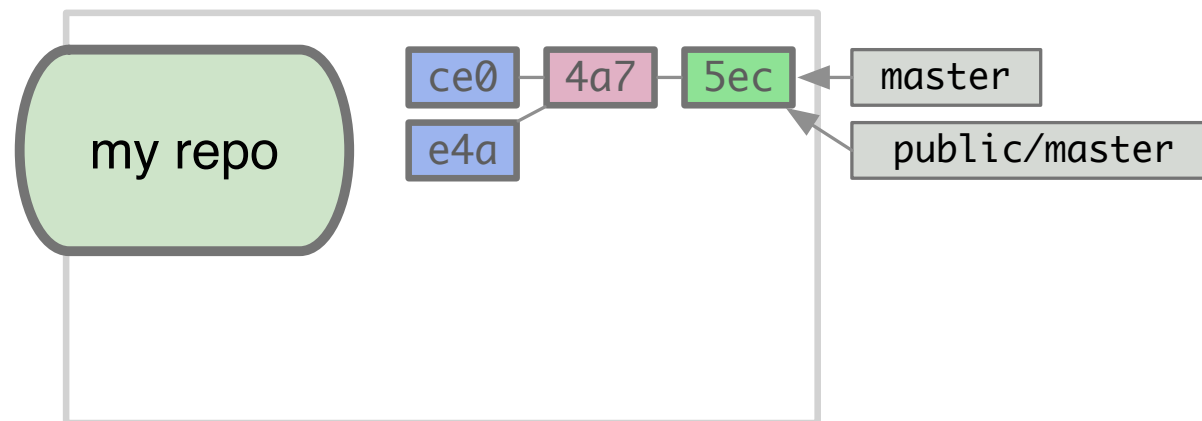
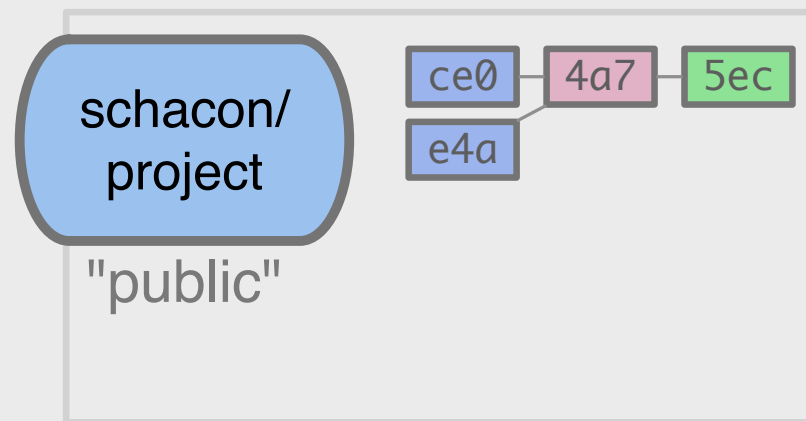
git push public



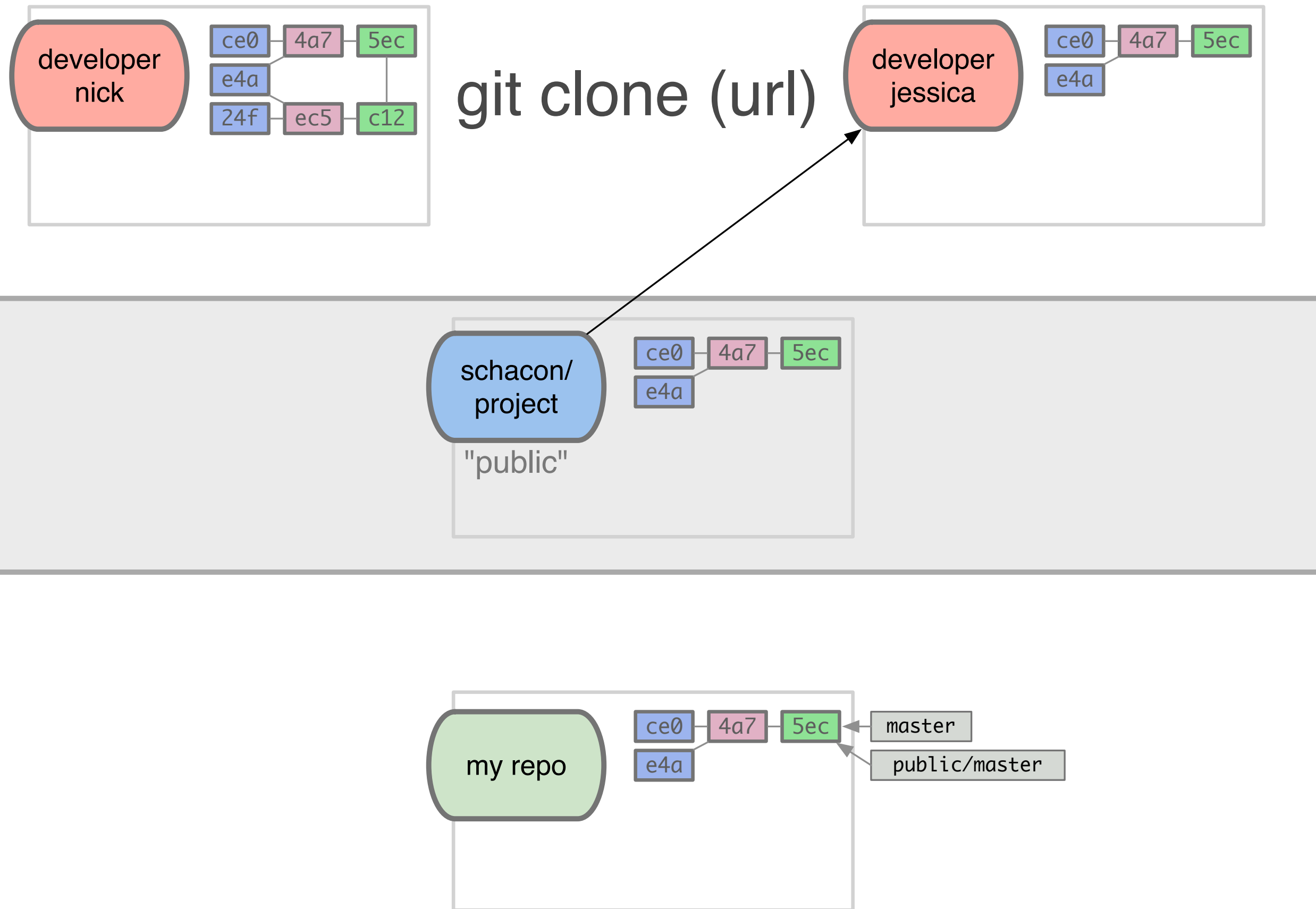


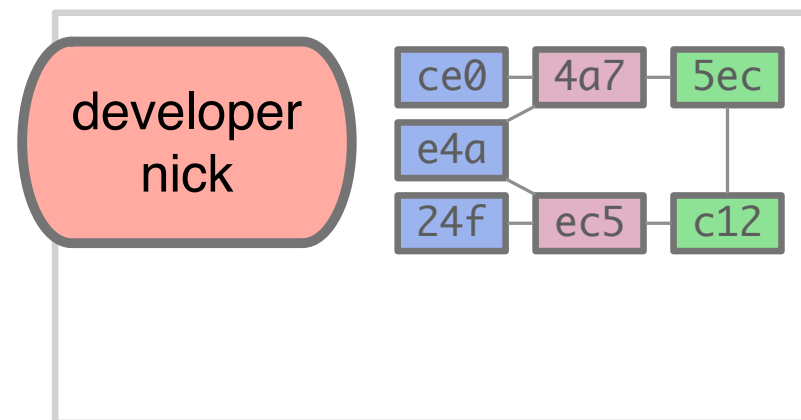


git commit

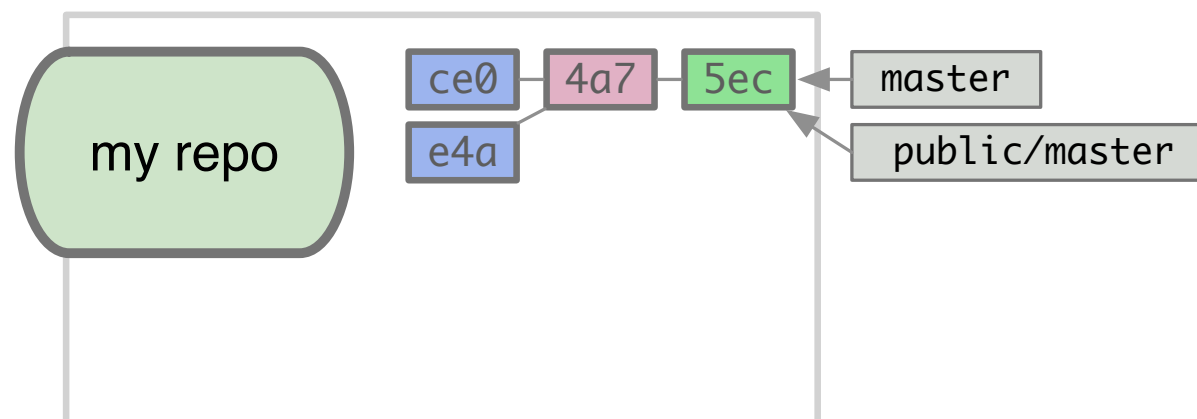
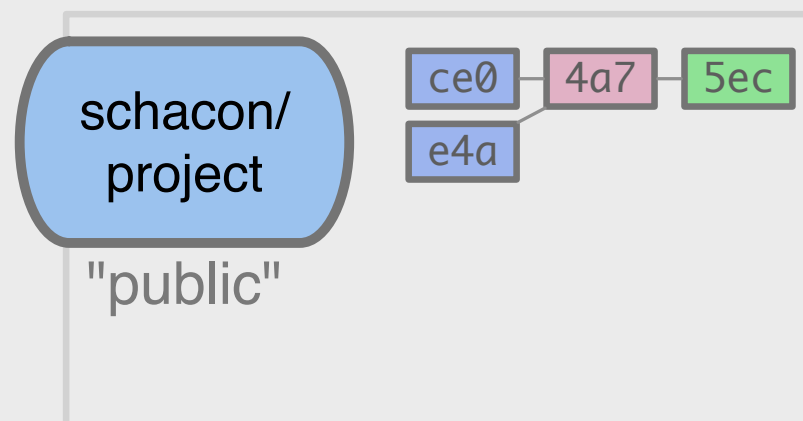
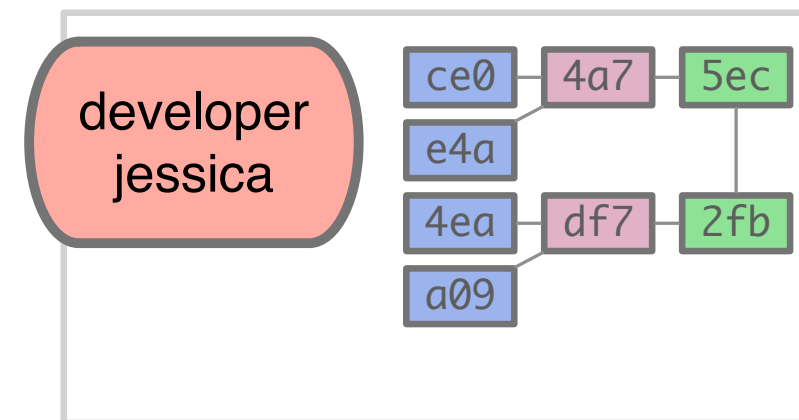


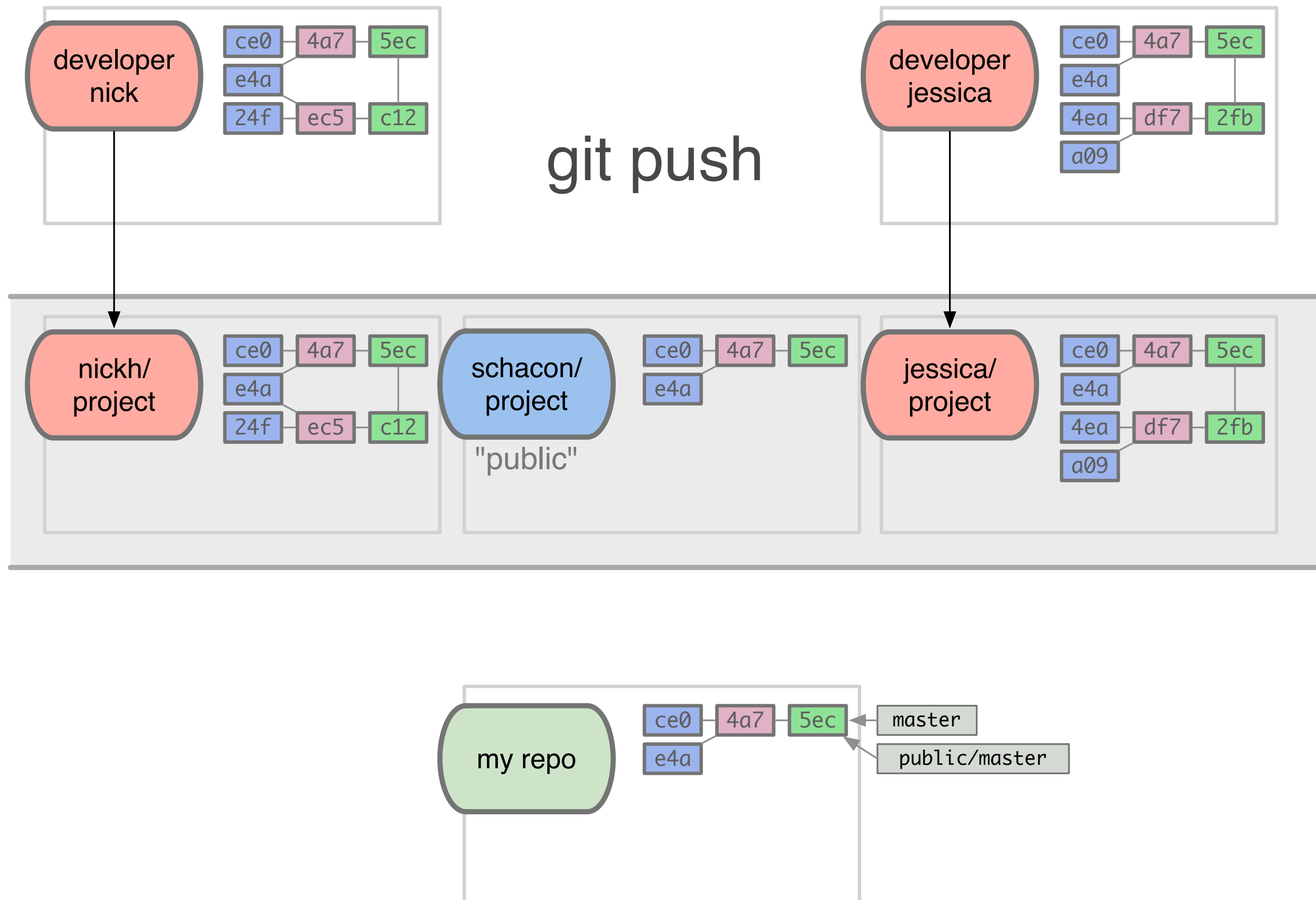


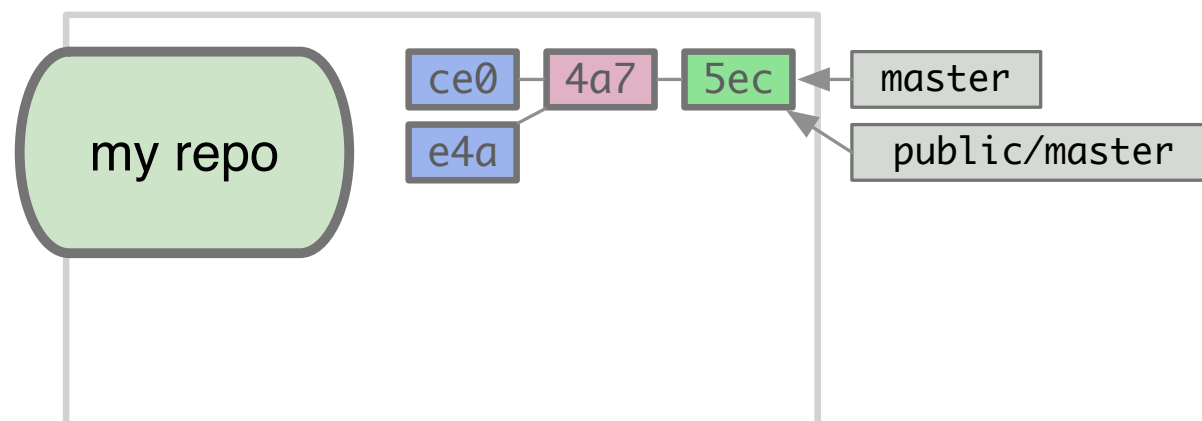
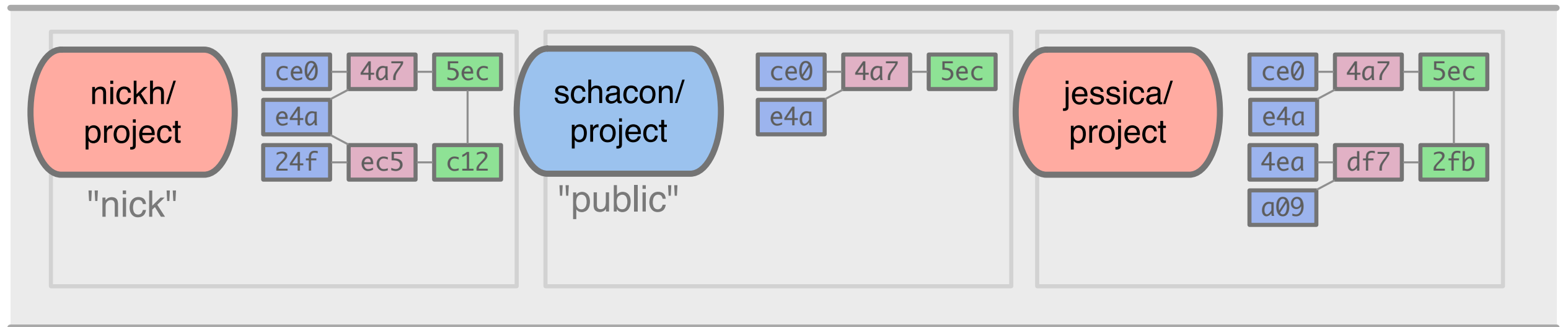
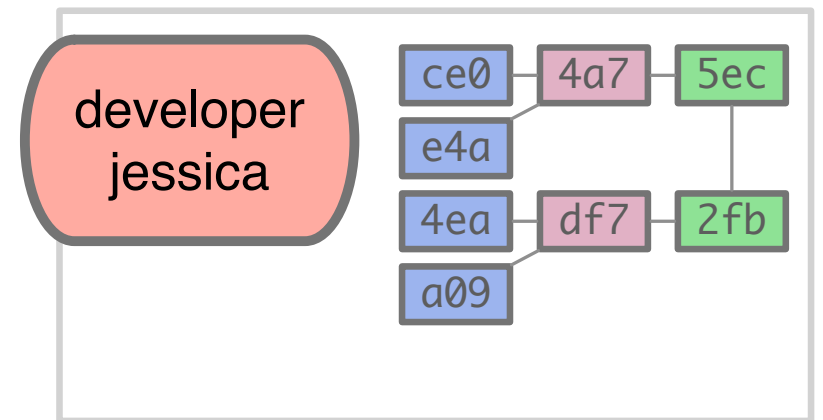
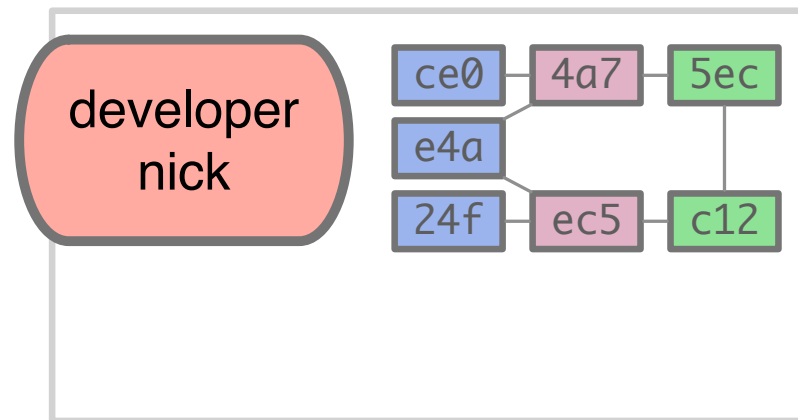




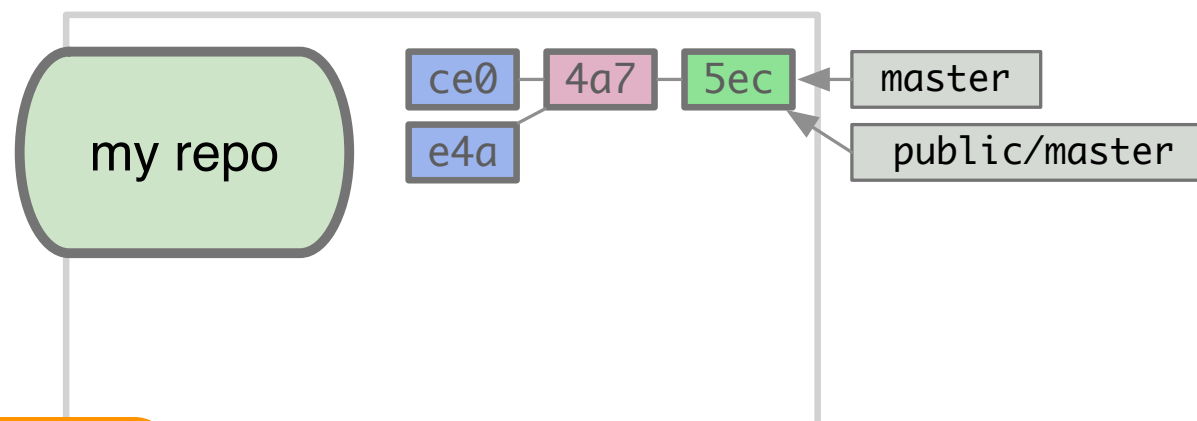
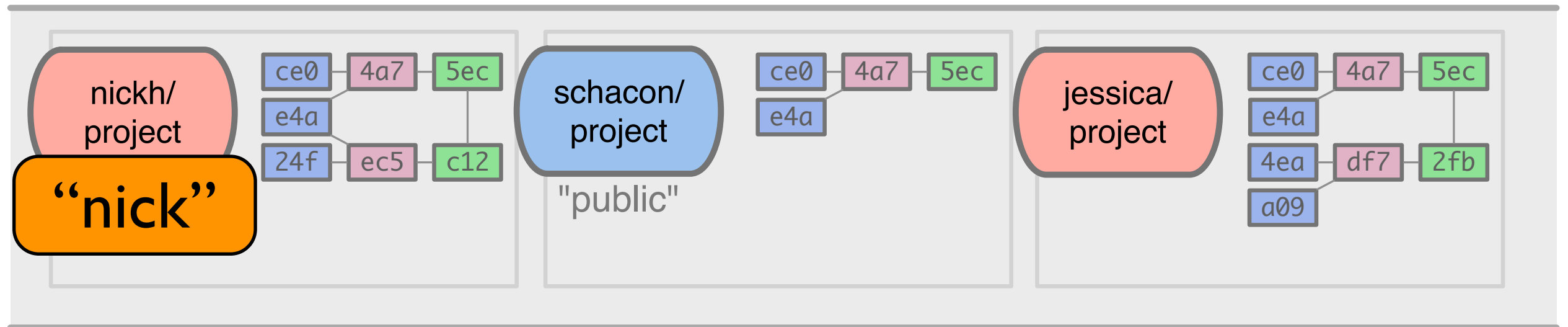
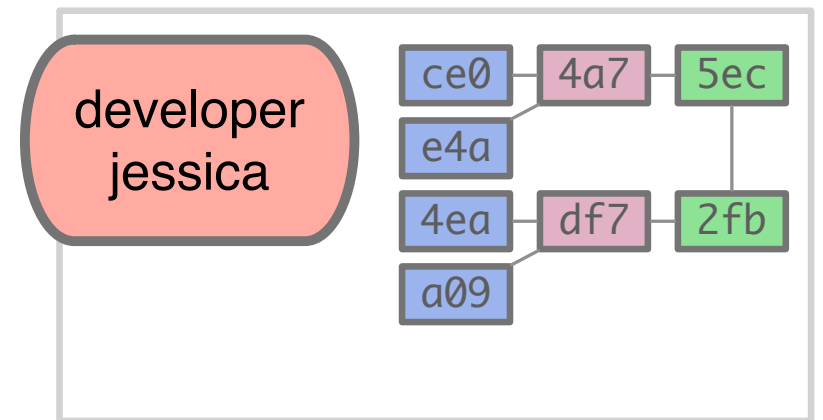
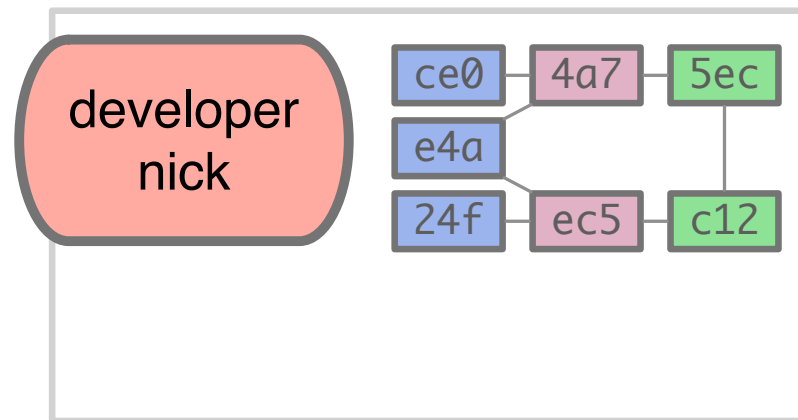
git commit



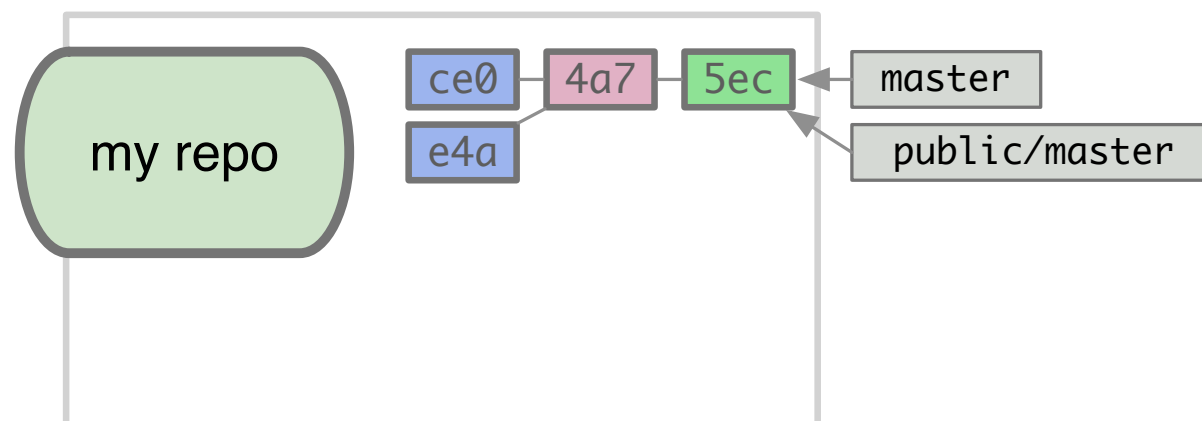
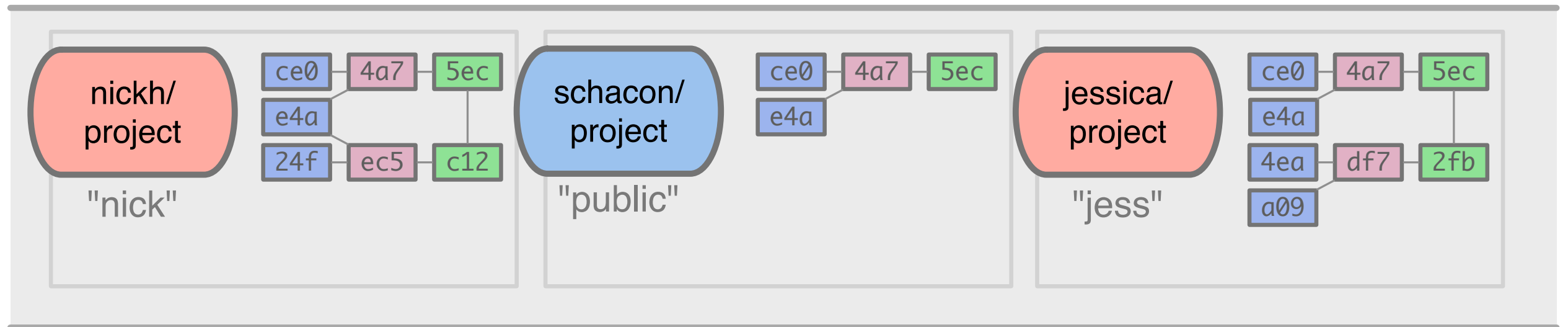
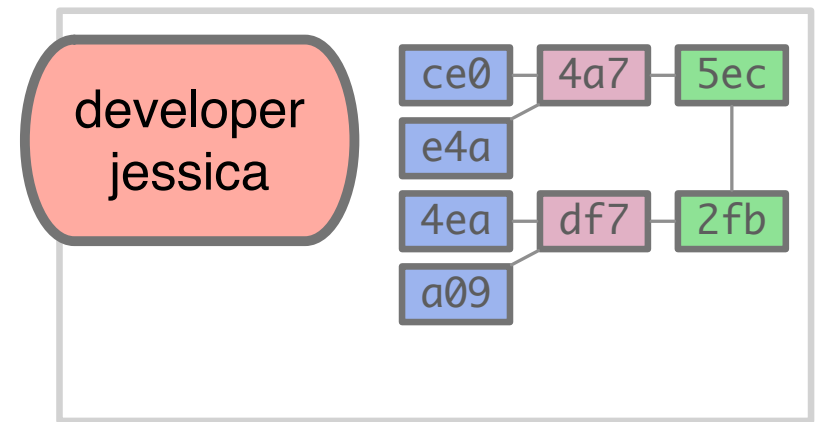
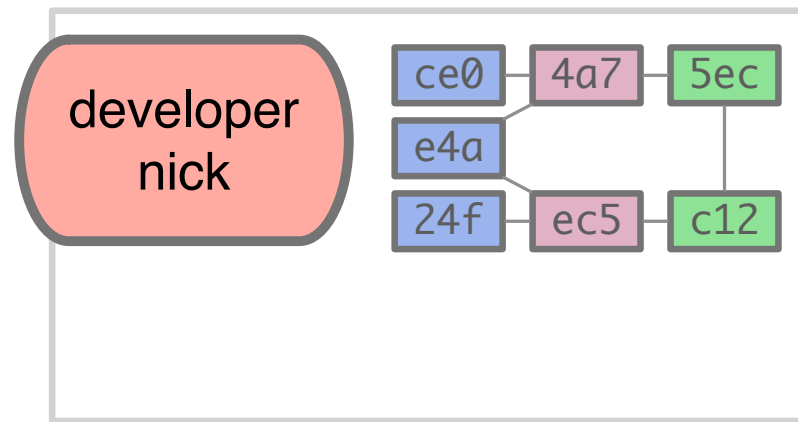




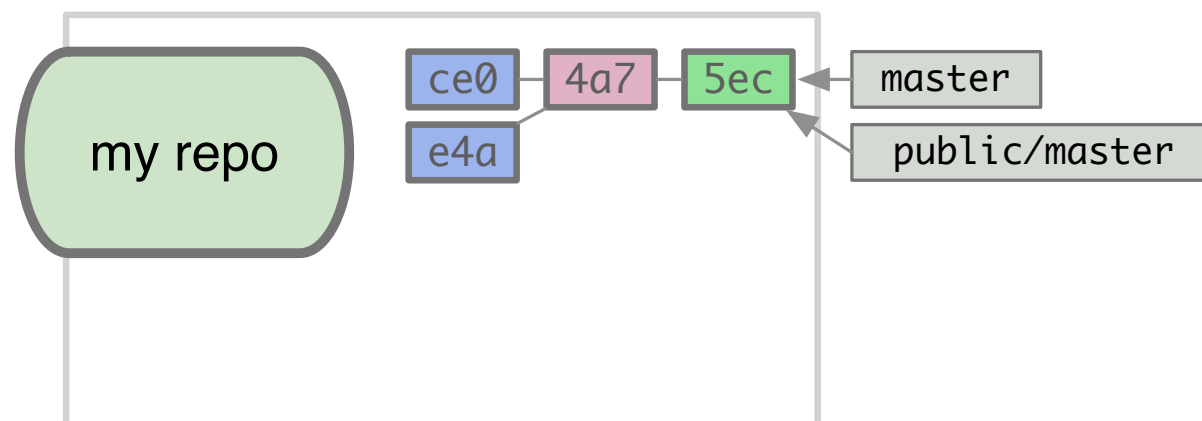
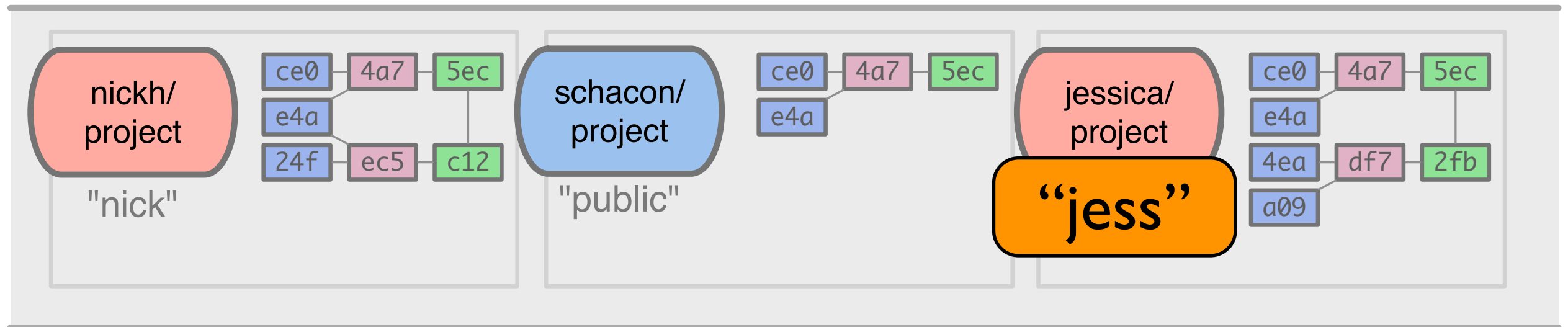
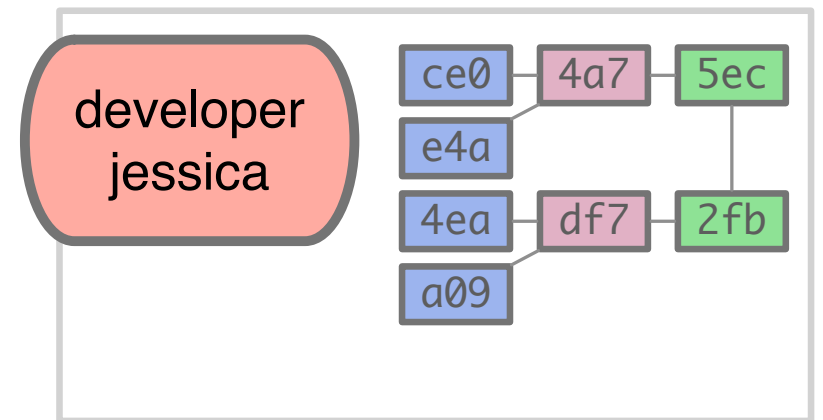
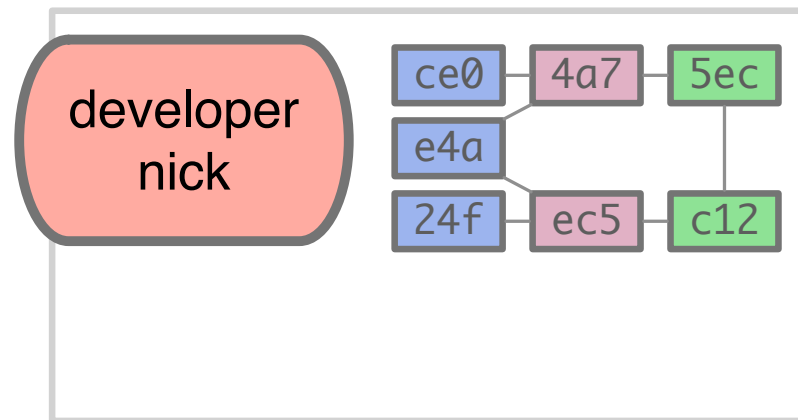
git remote add nick git://github.com/nickh/project.git



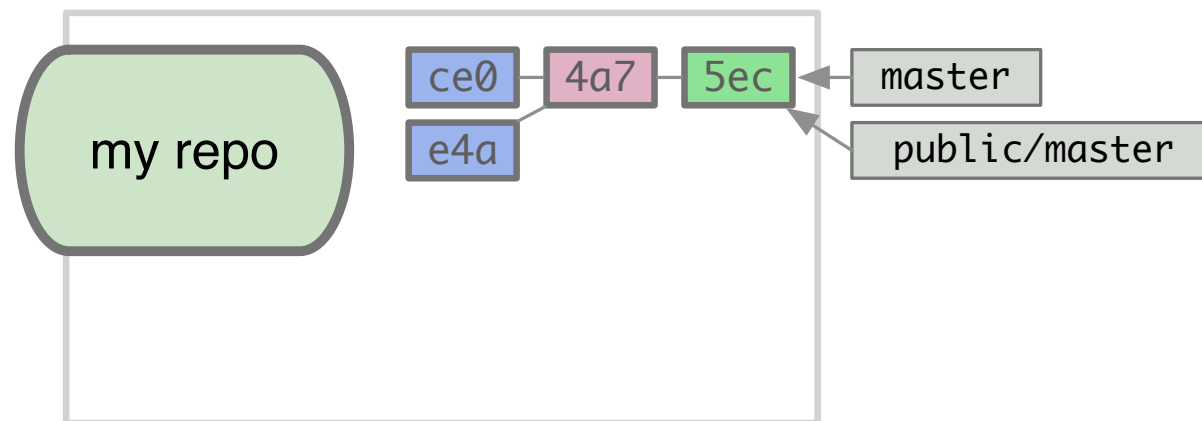
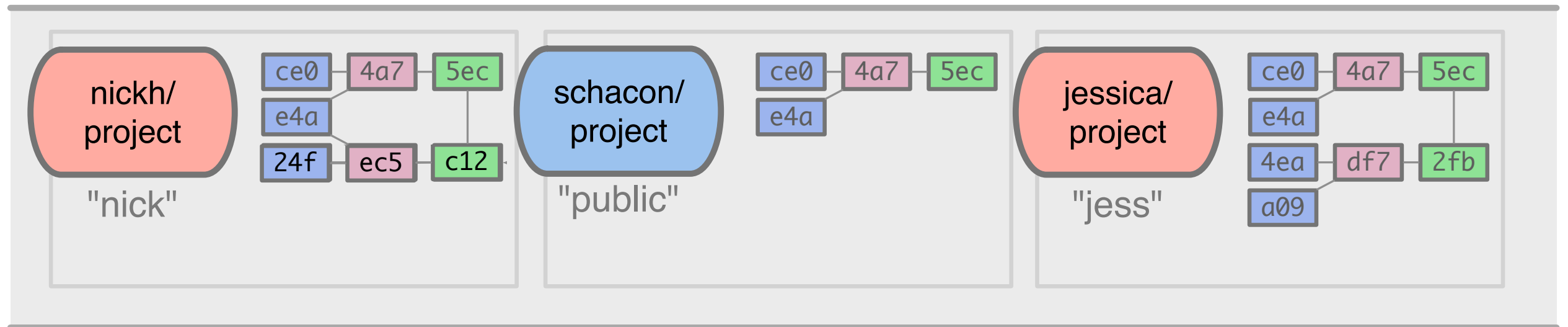
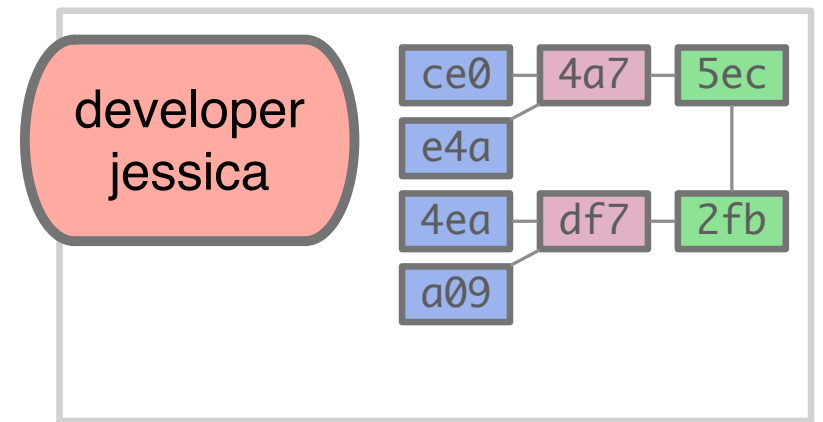
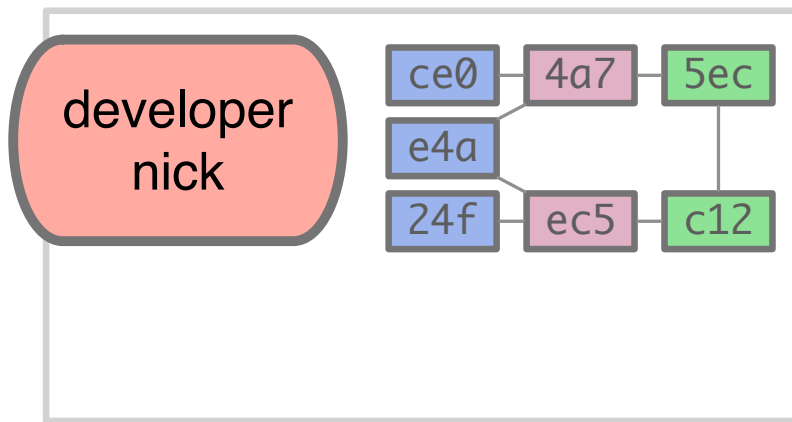
git remote add **nick** git://github.com/nickh/project.git



`git remote add jess git://github.com/jessica/project.git`

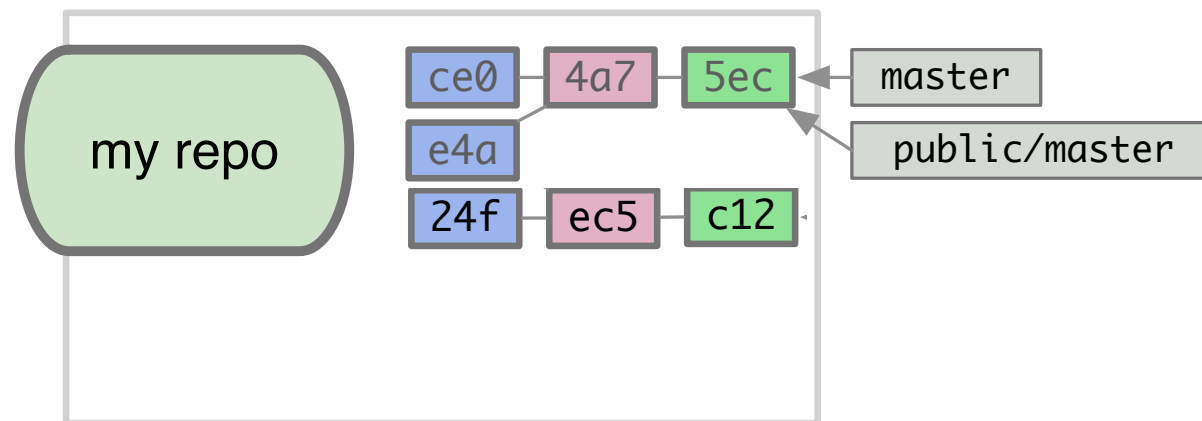
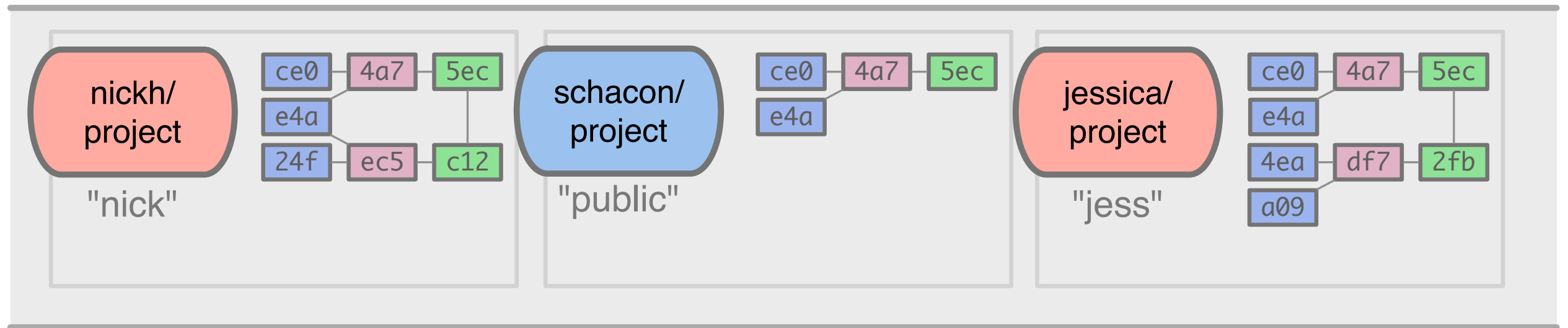
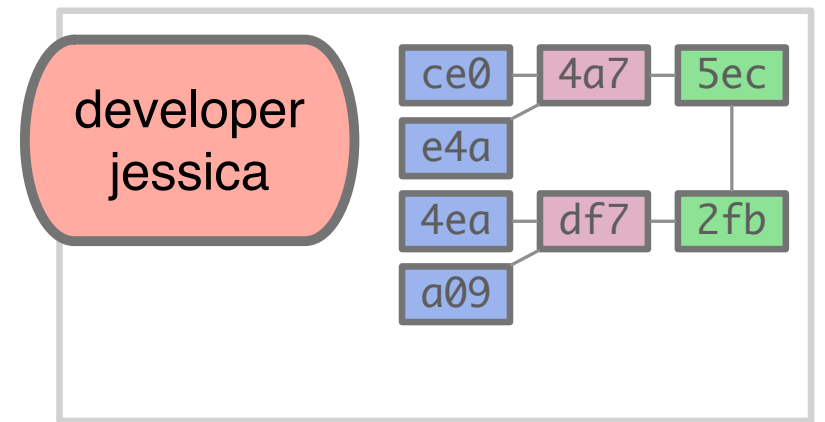
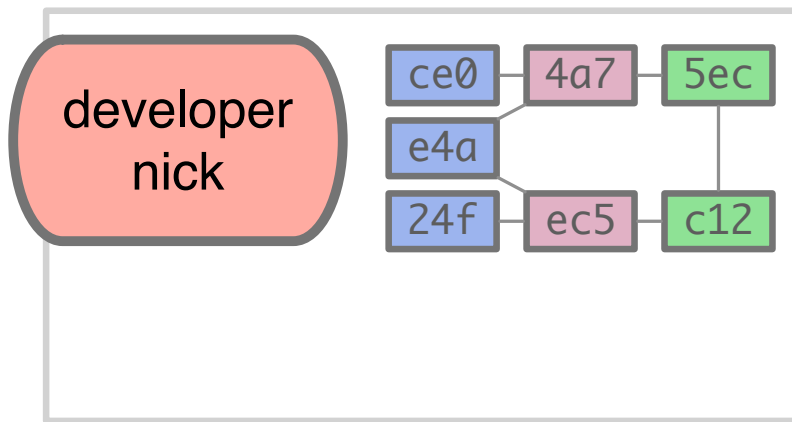


git remote add **jess** git://github.com/jessica/project.git

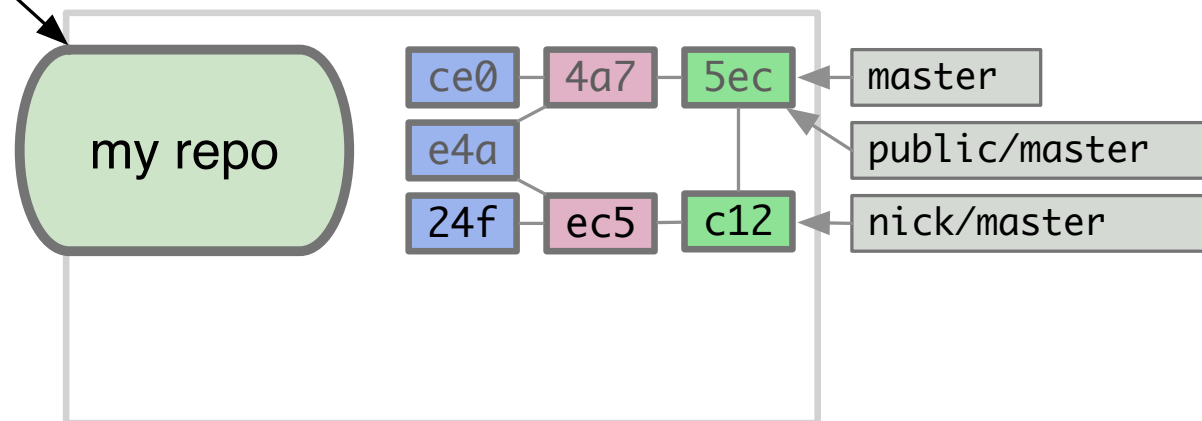
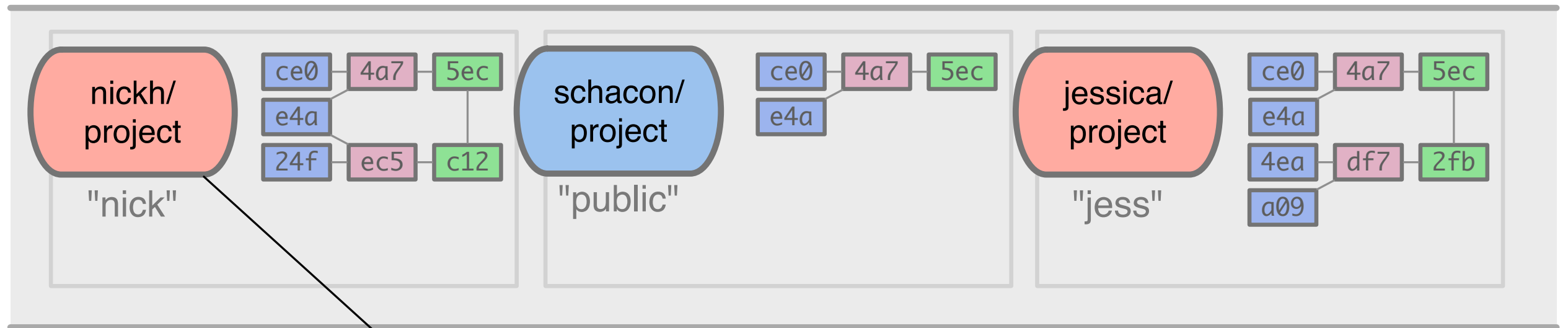
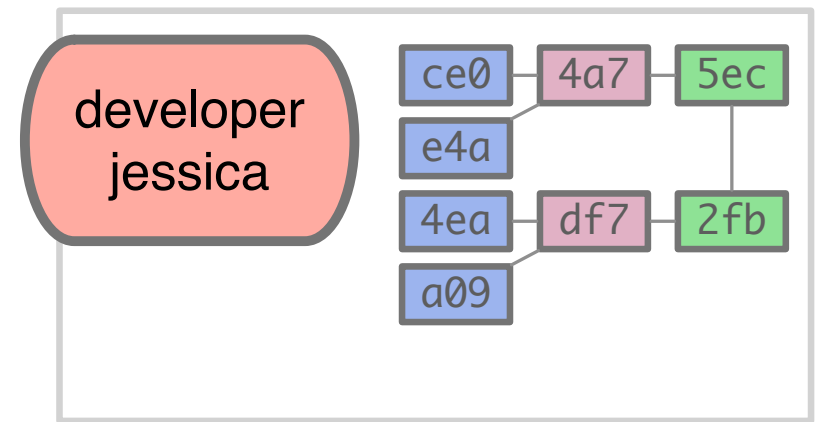
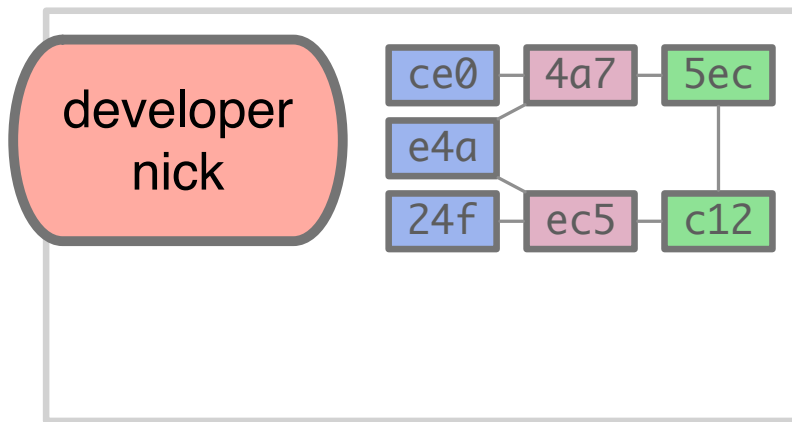


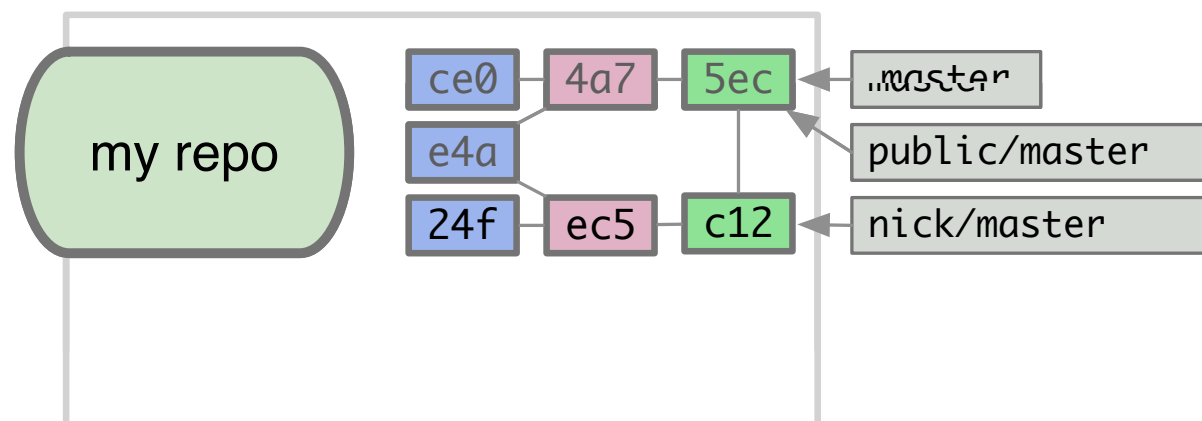
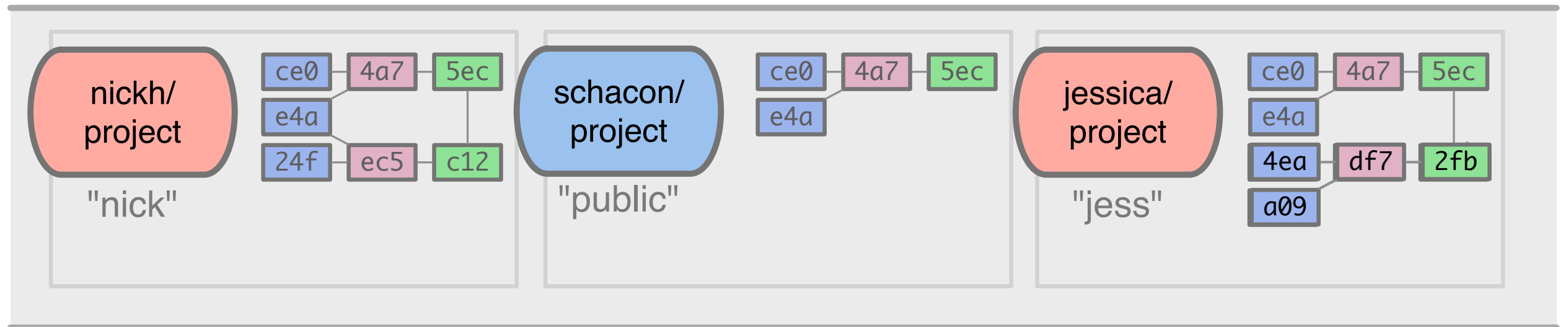
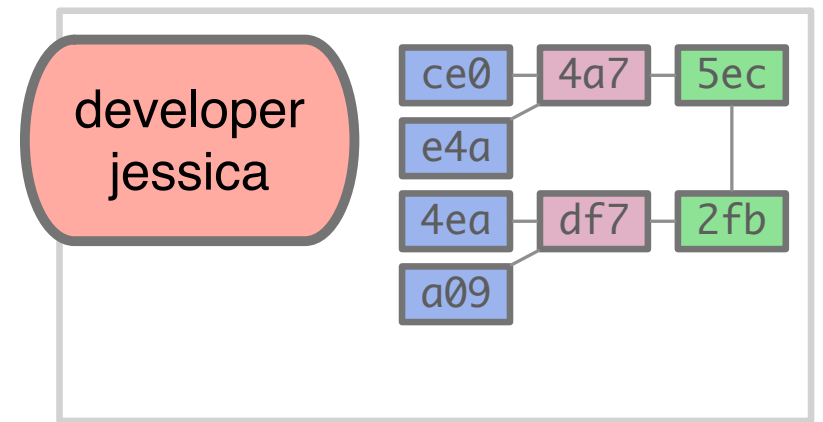
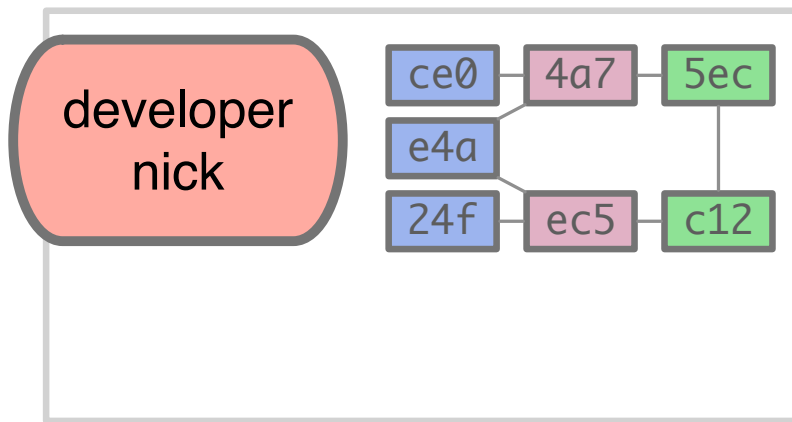
git fetch nick



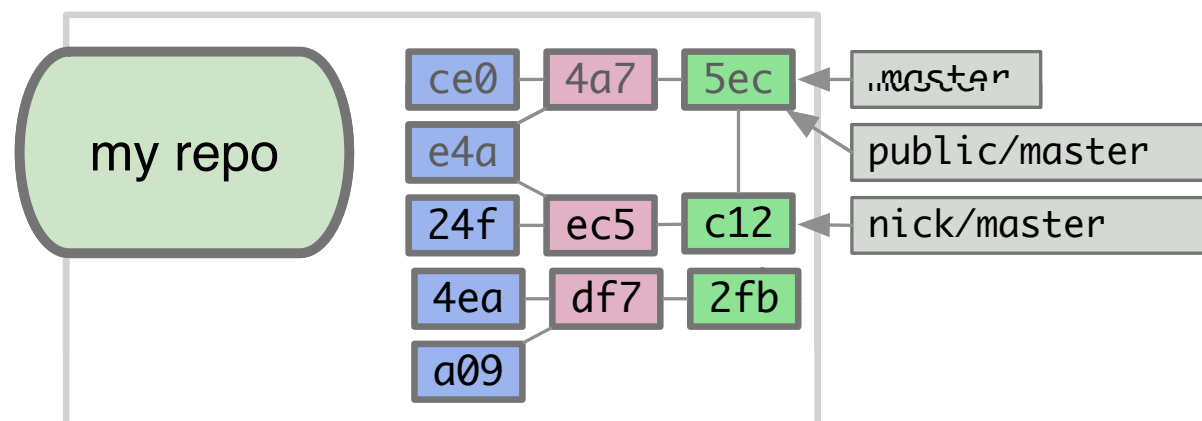
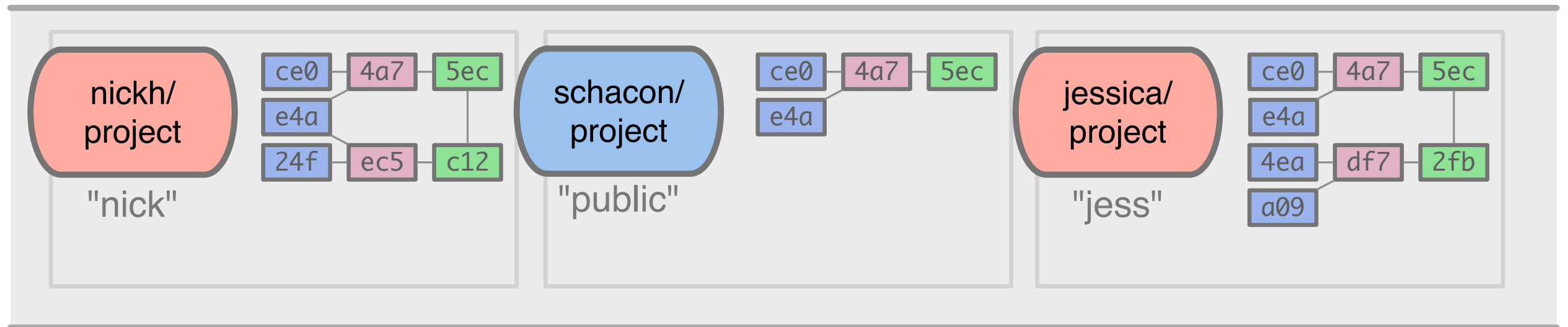
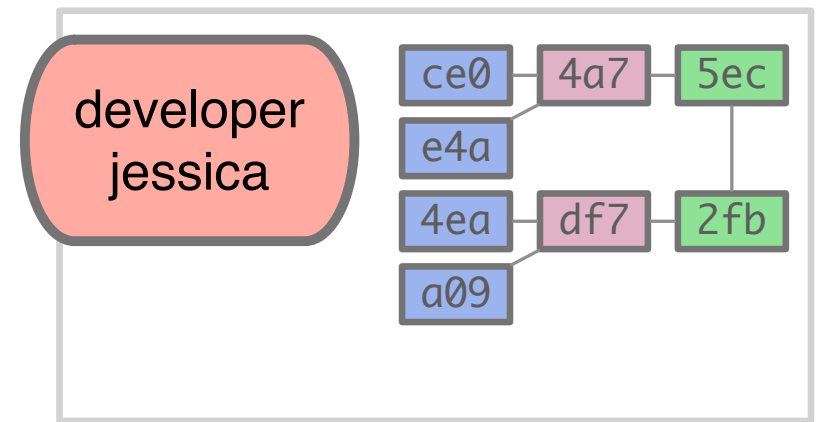
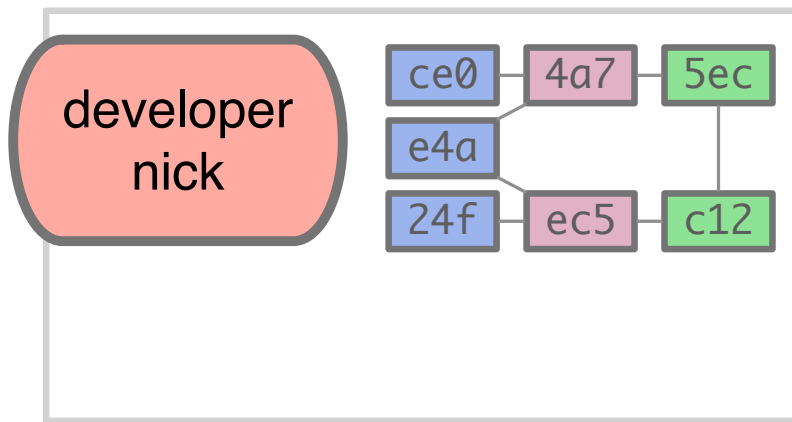


git fetch nick

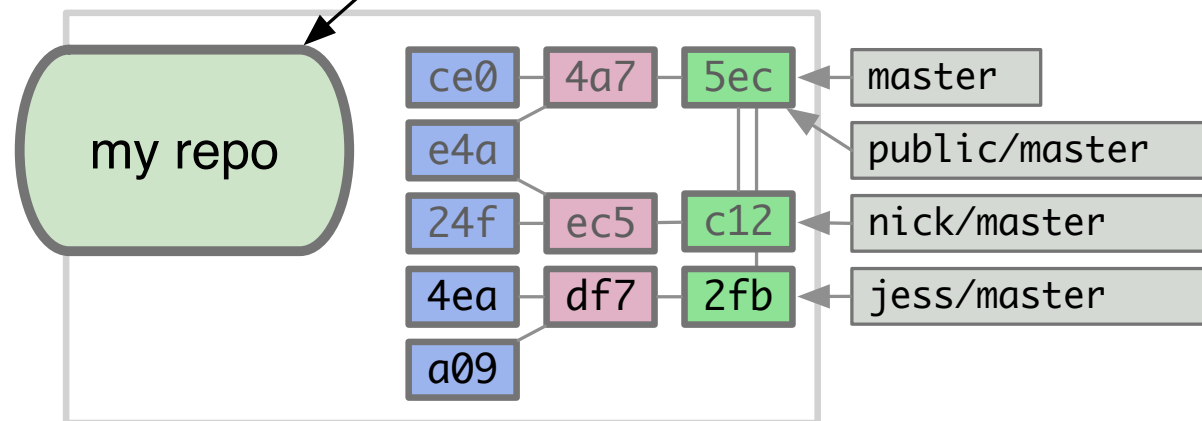
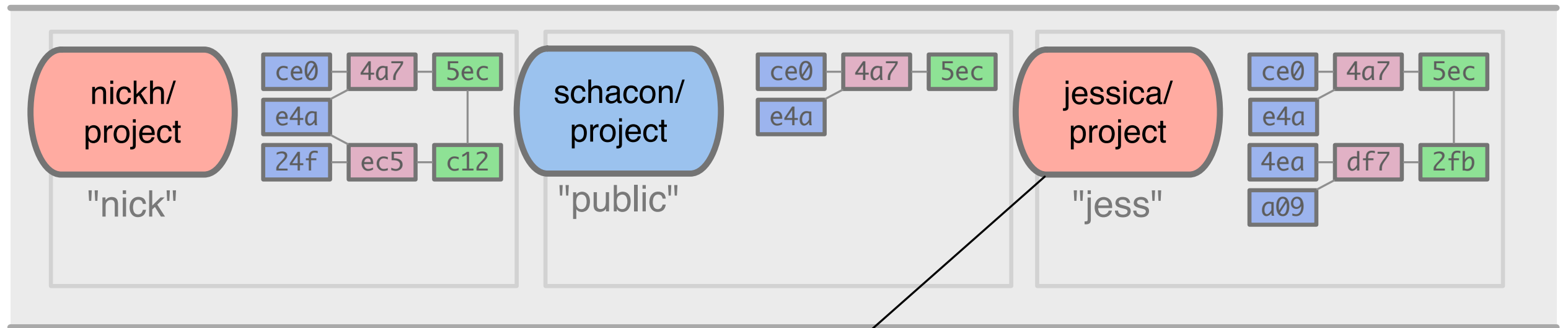
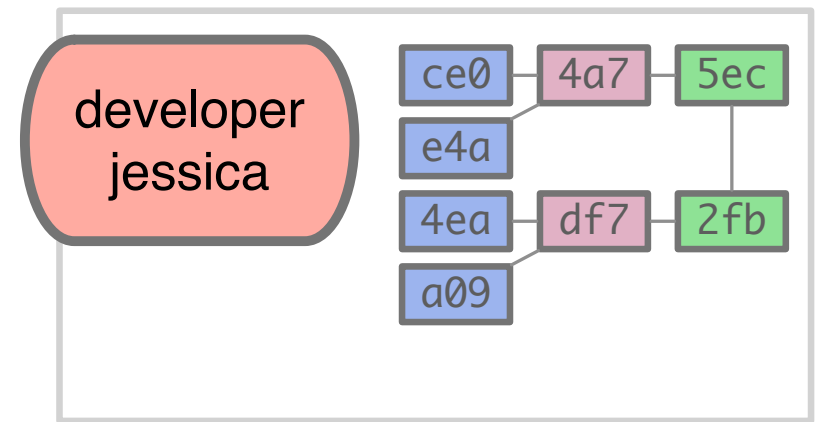
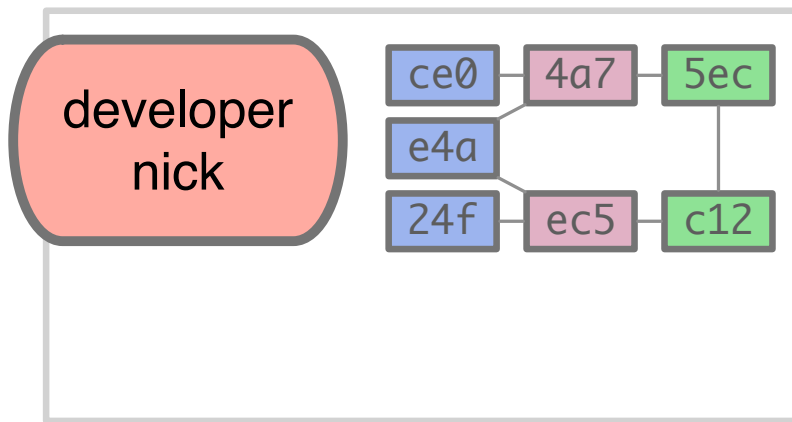


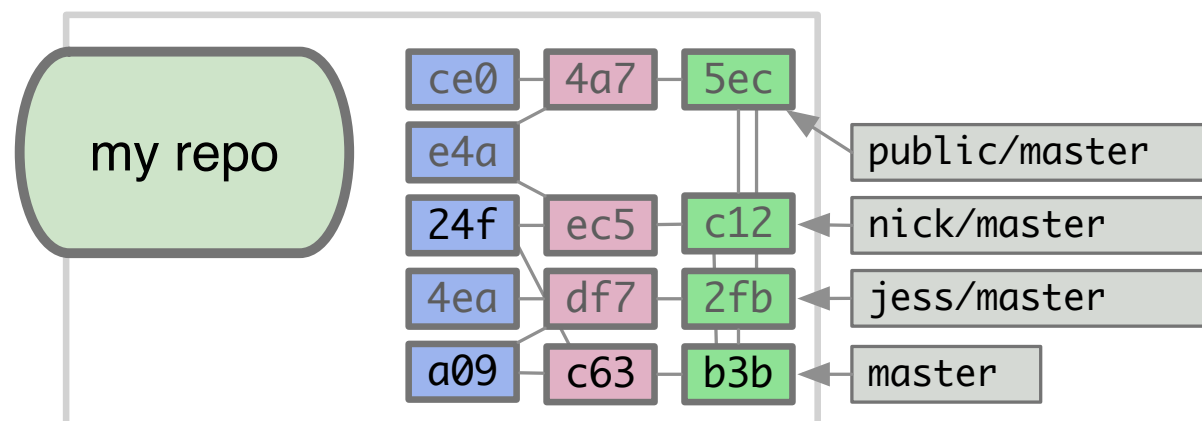
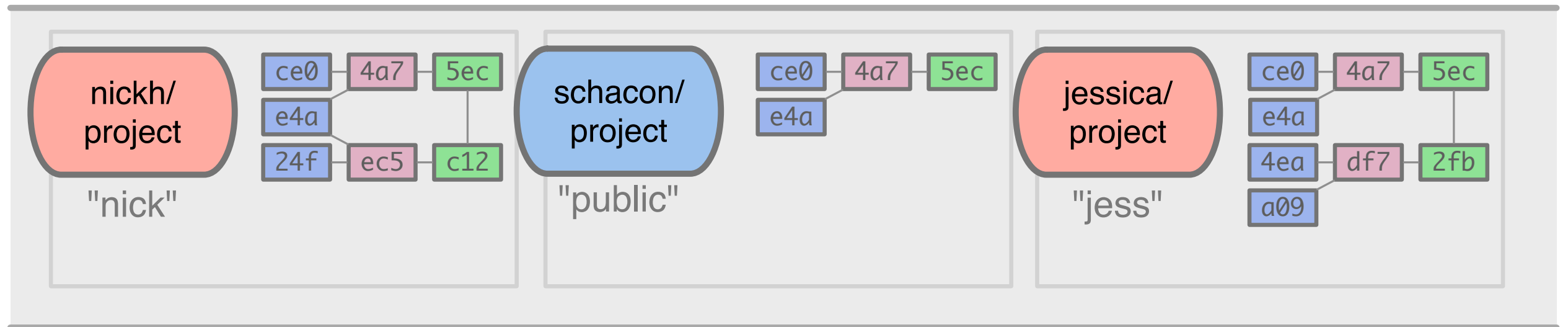
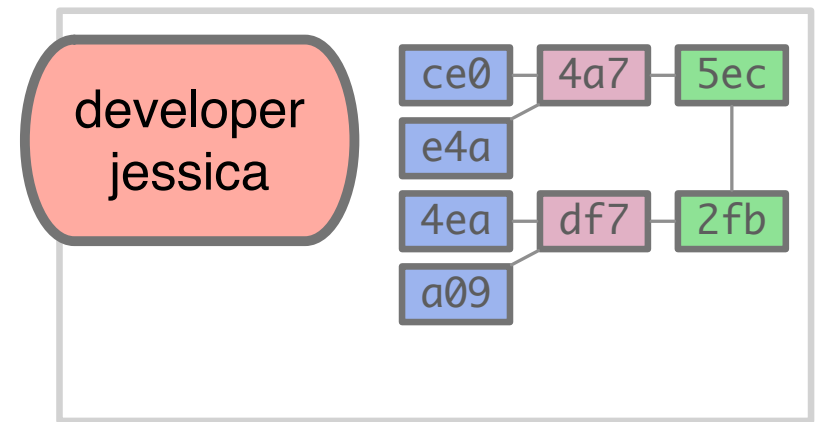
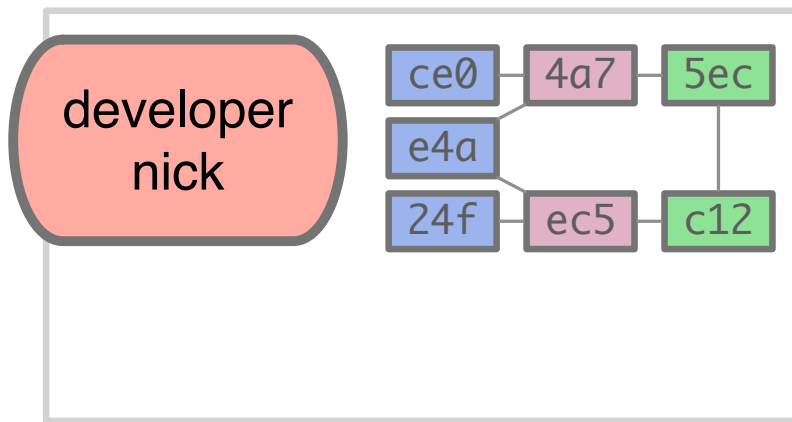


git fetch jess

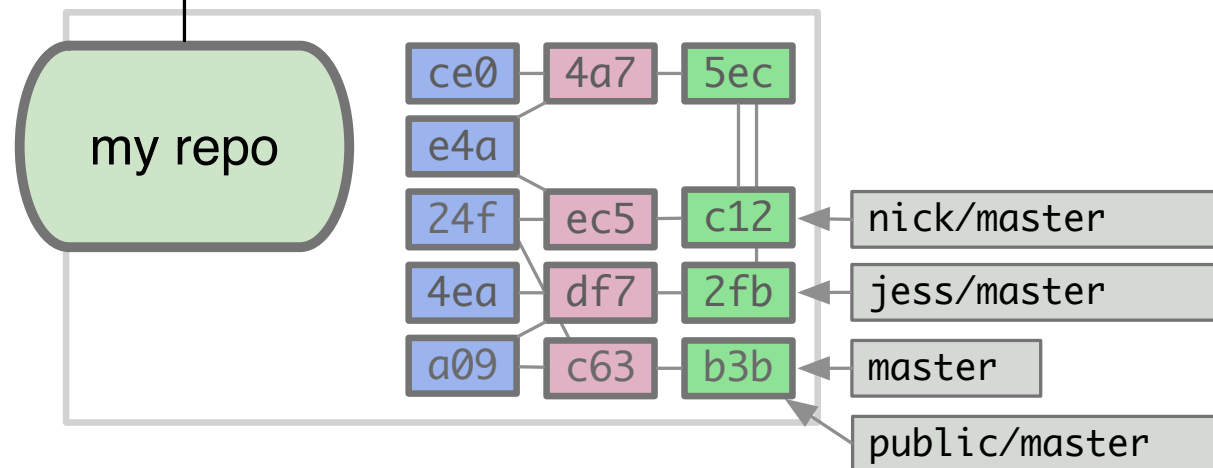
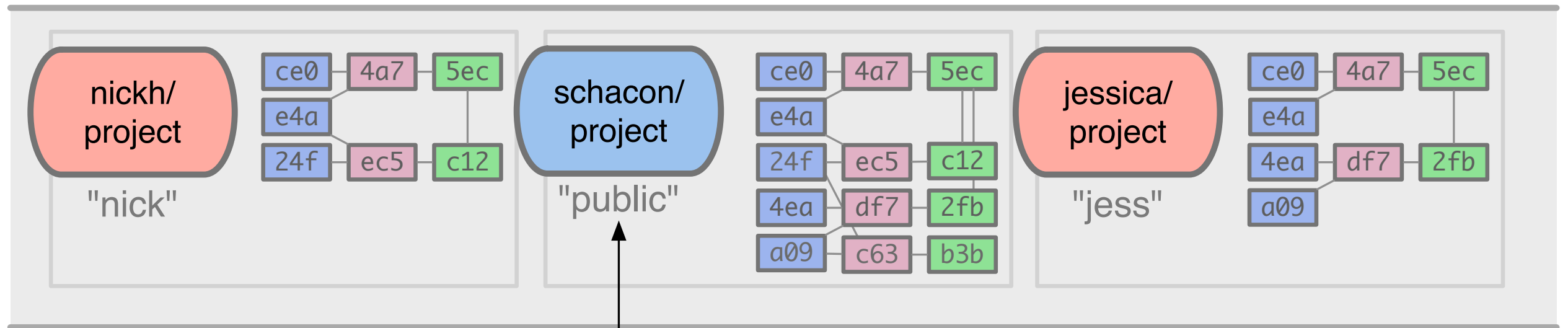
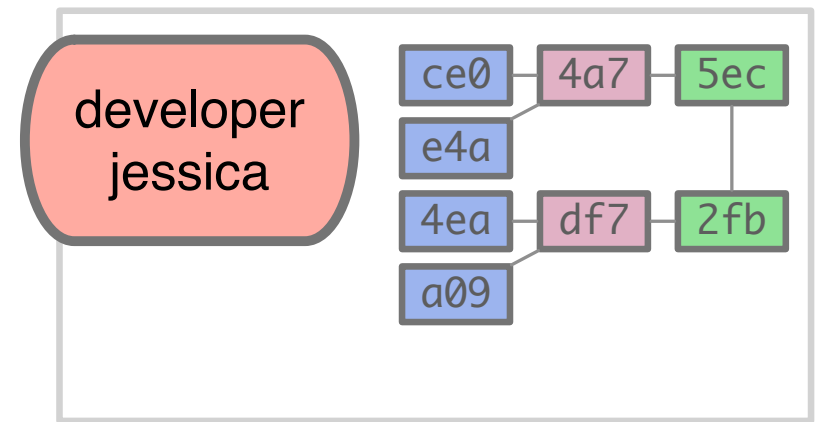
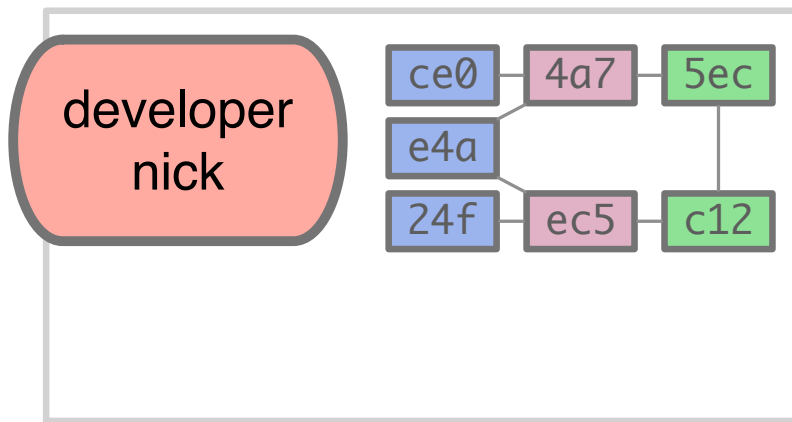


git fetch jess

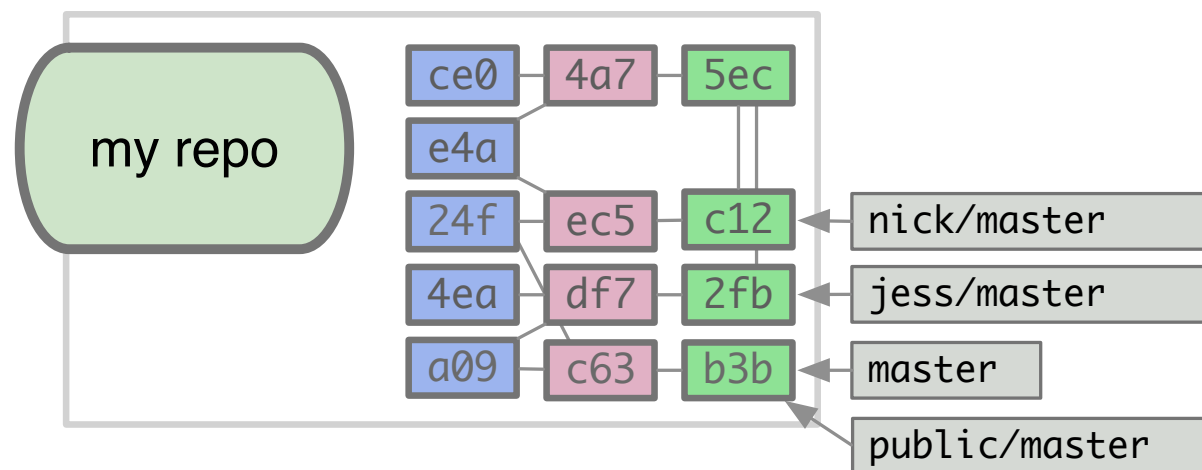
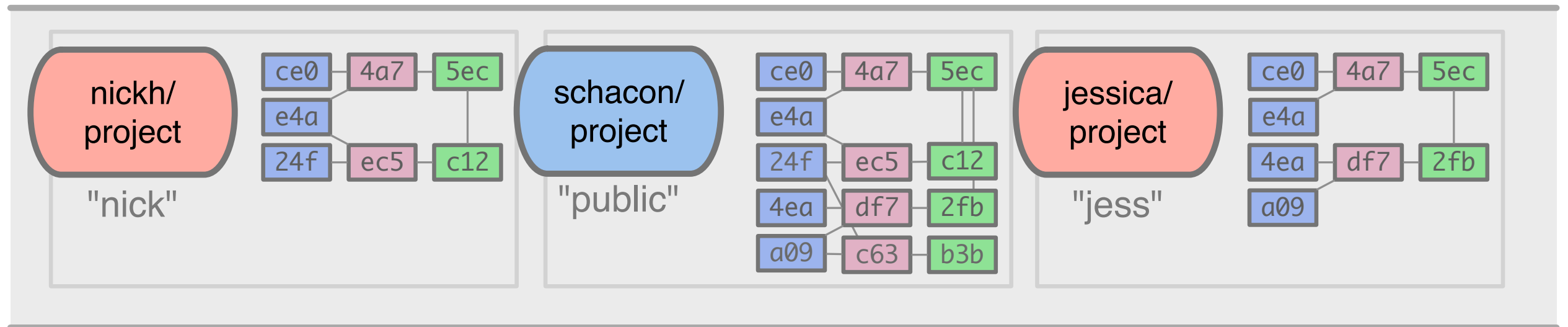
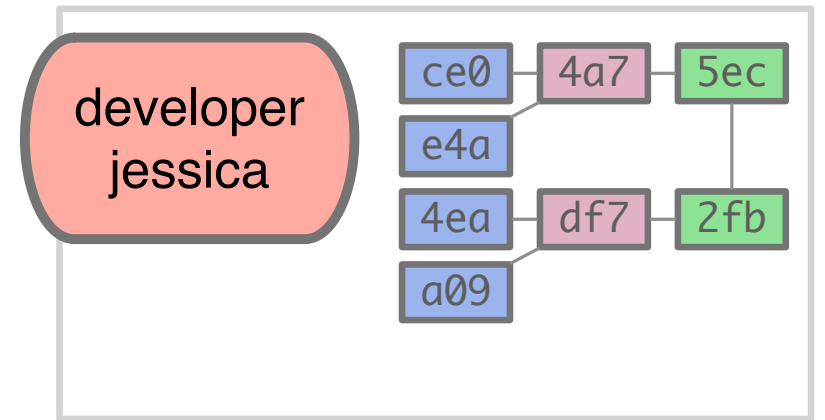
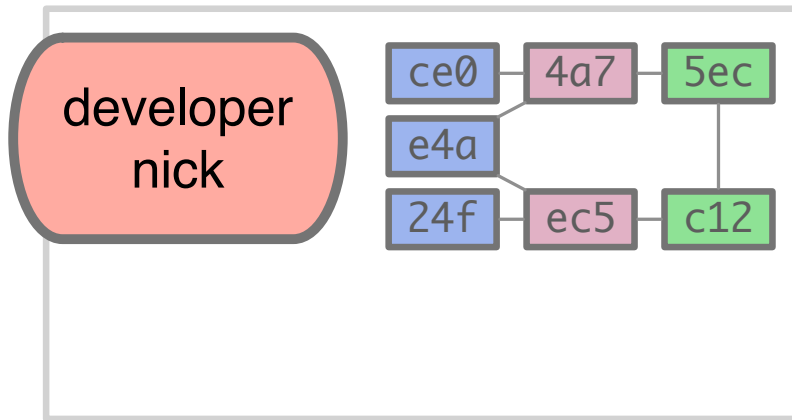




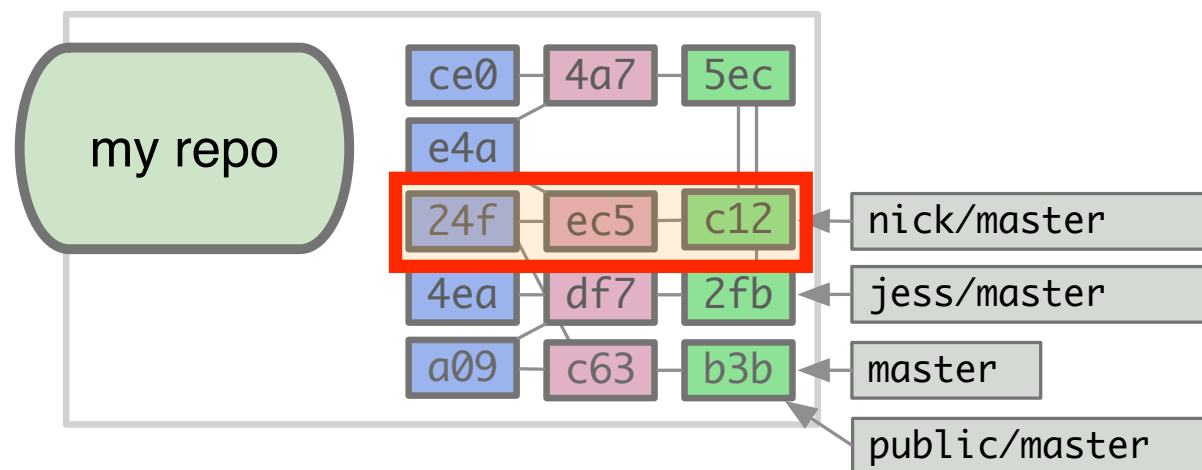
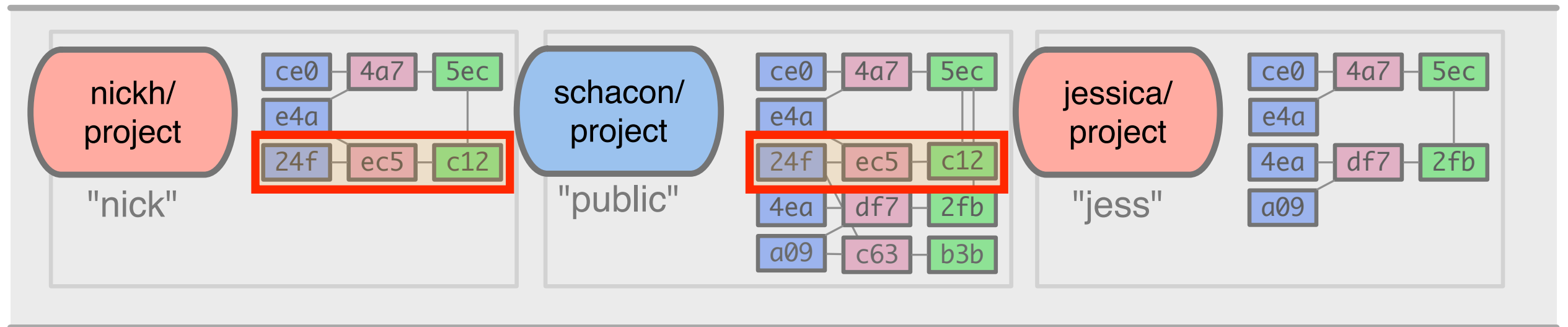
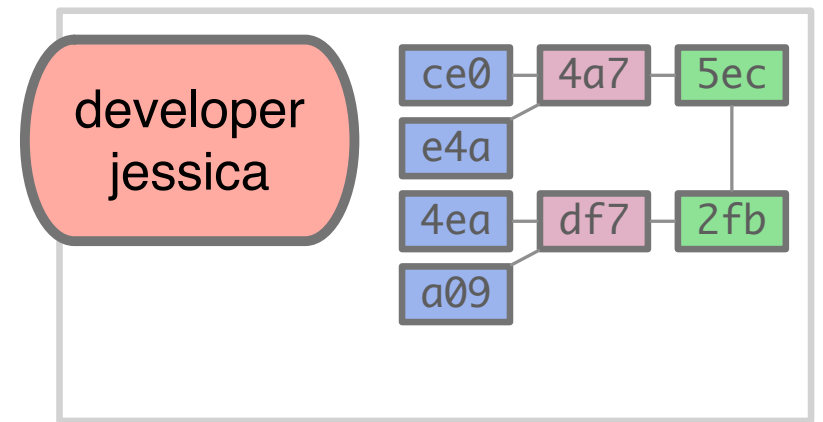
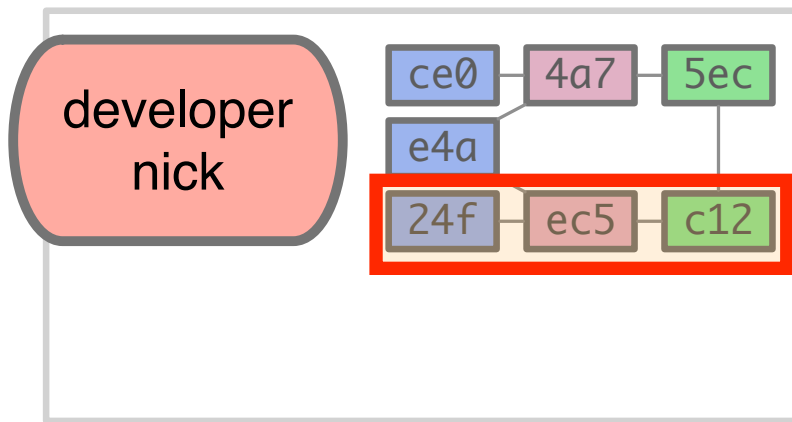
git merge nick jess

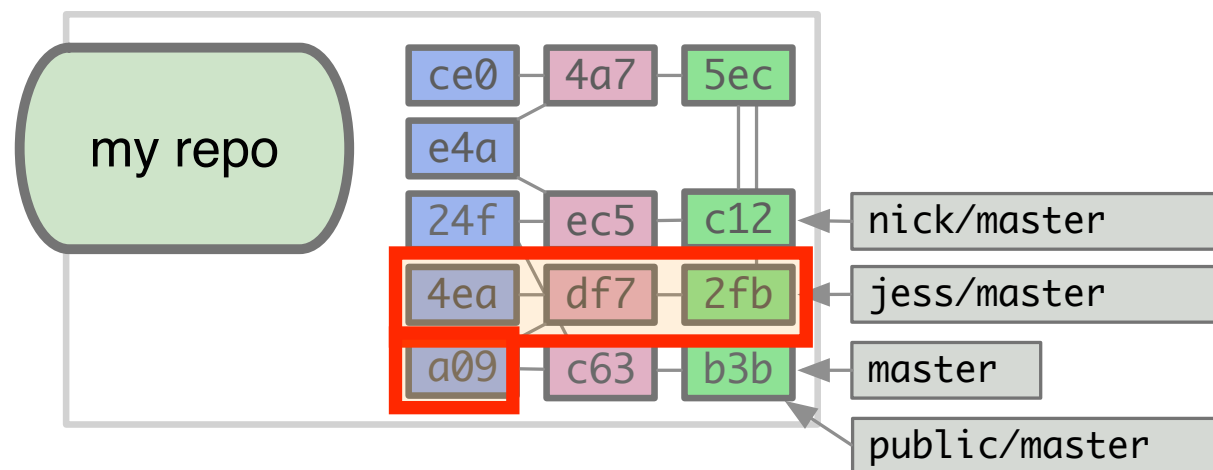
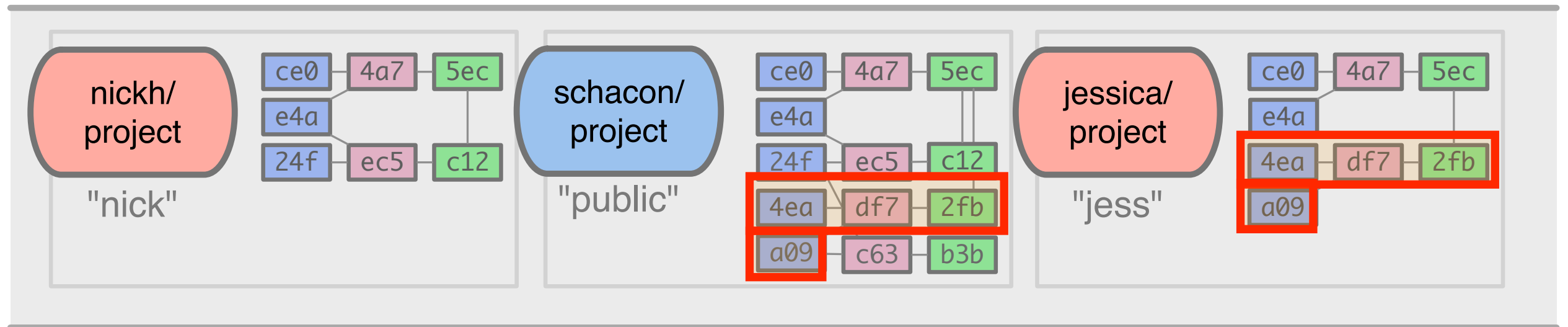
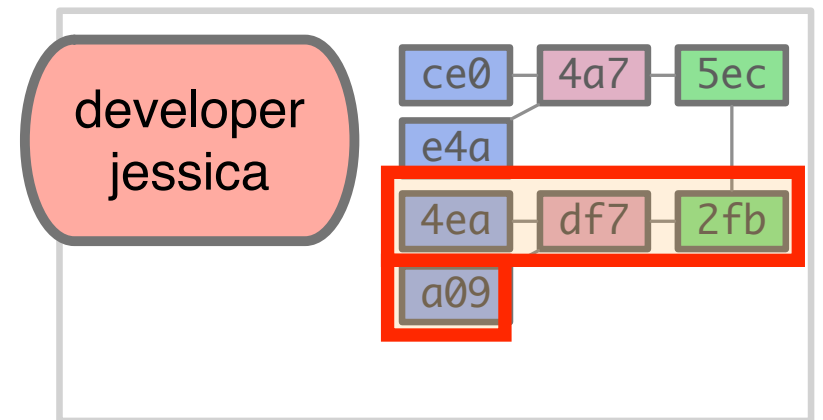
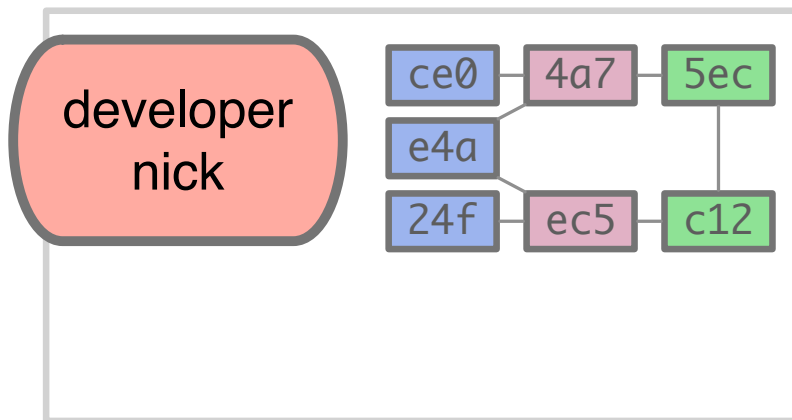


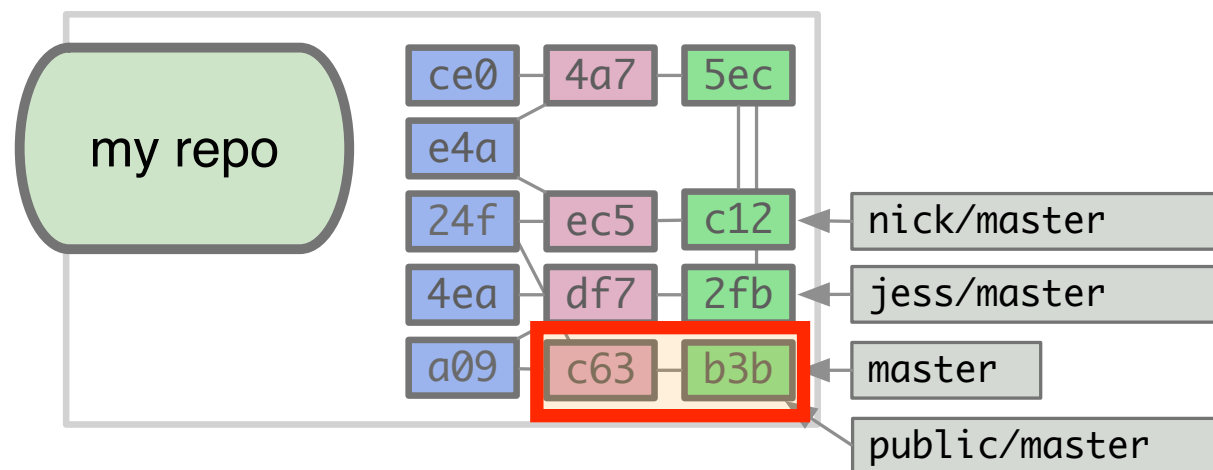
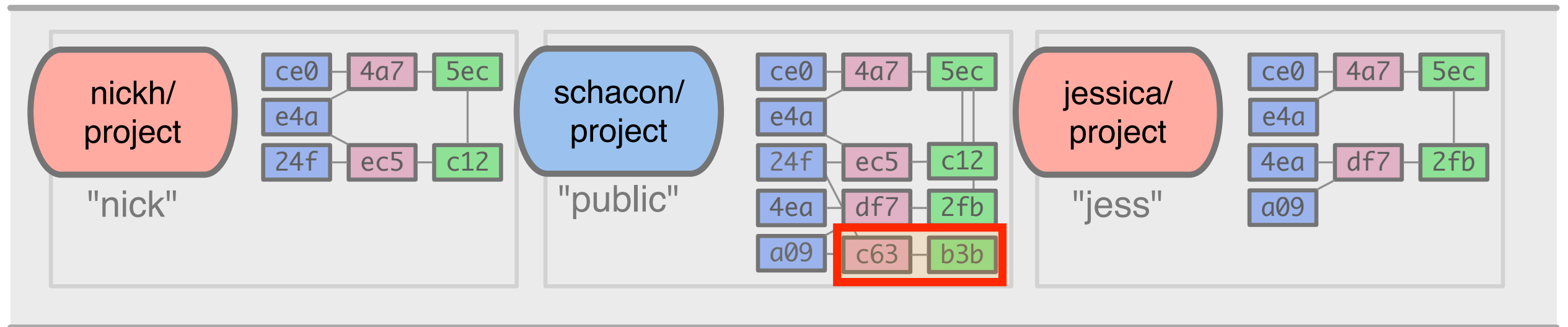
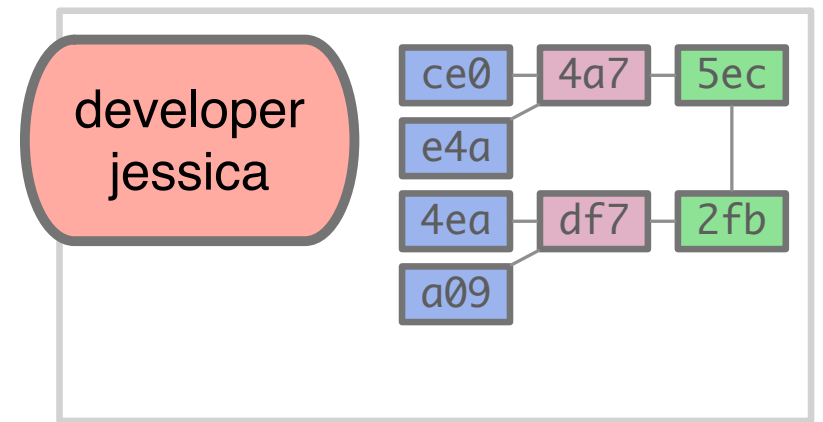
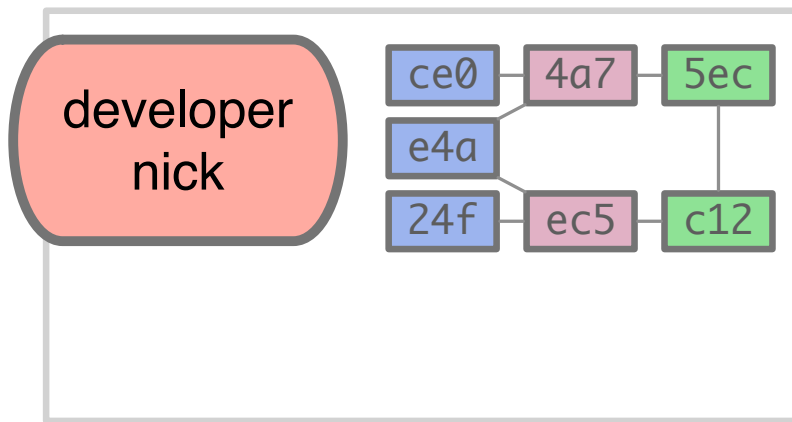
git push public







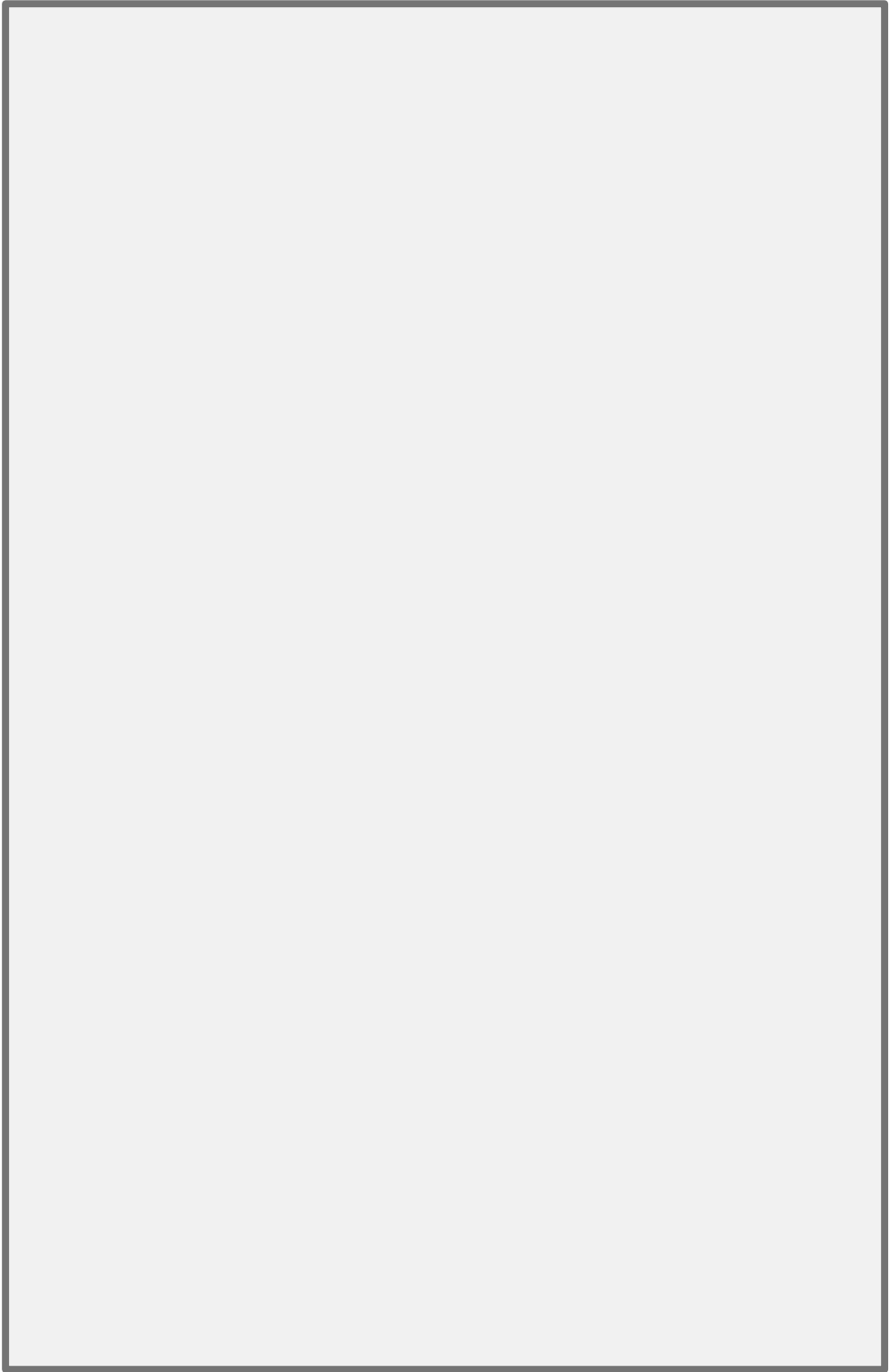
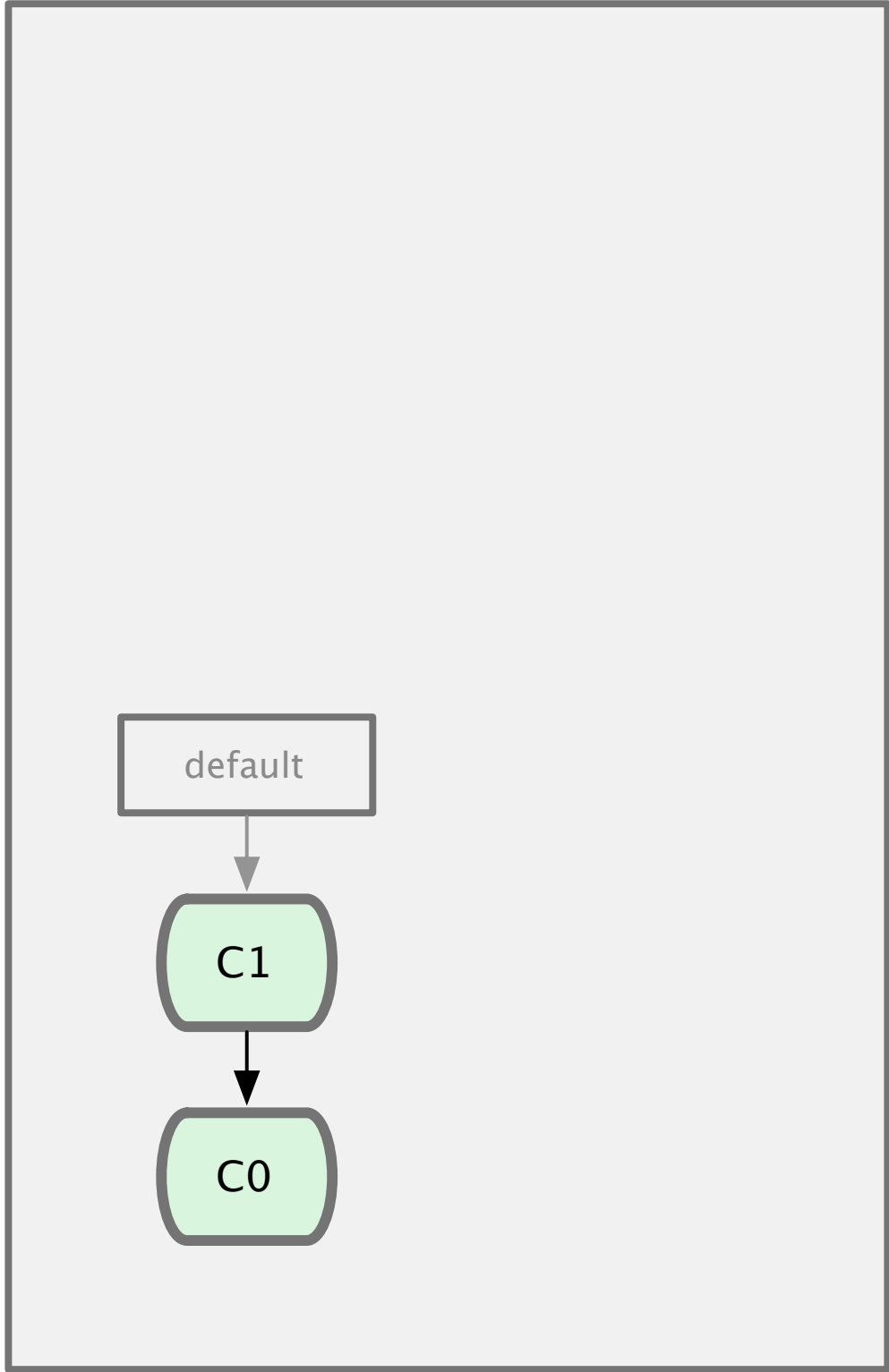




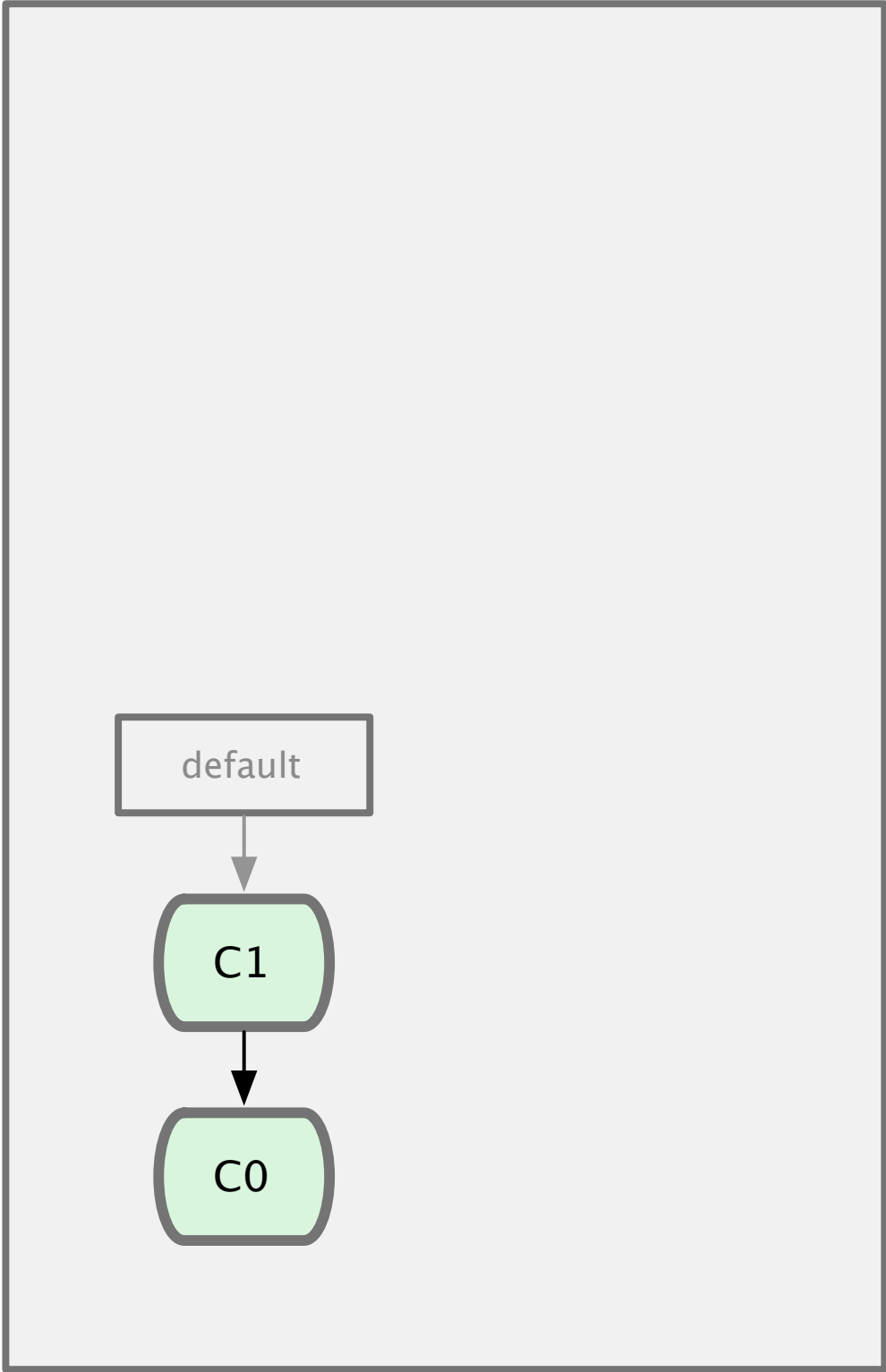
# Remotes Are Branches

scott

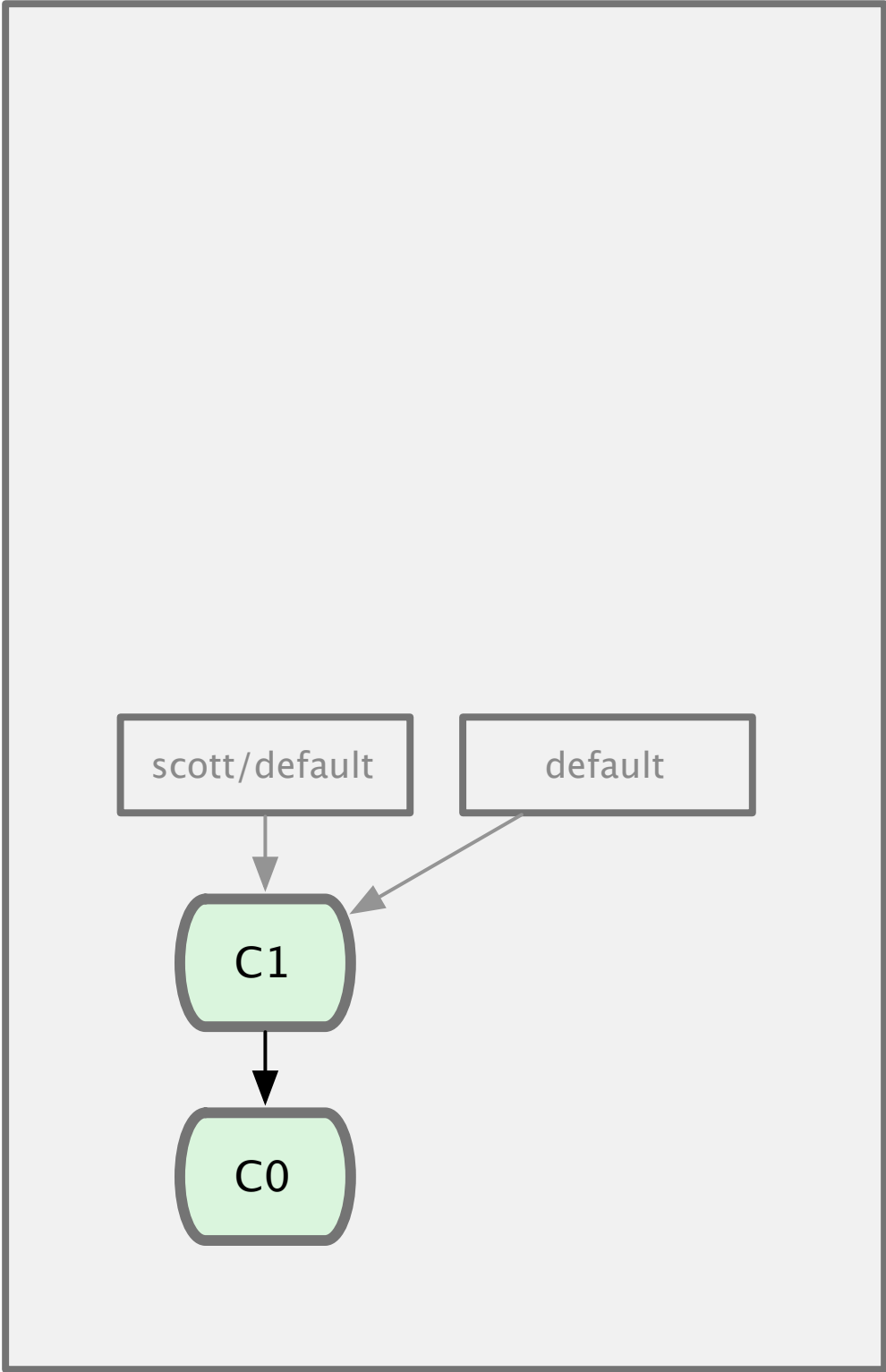
jessica



scott

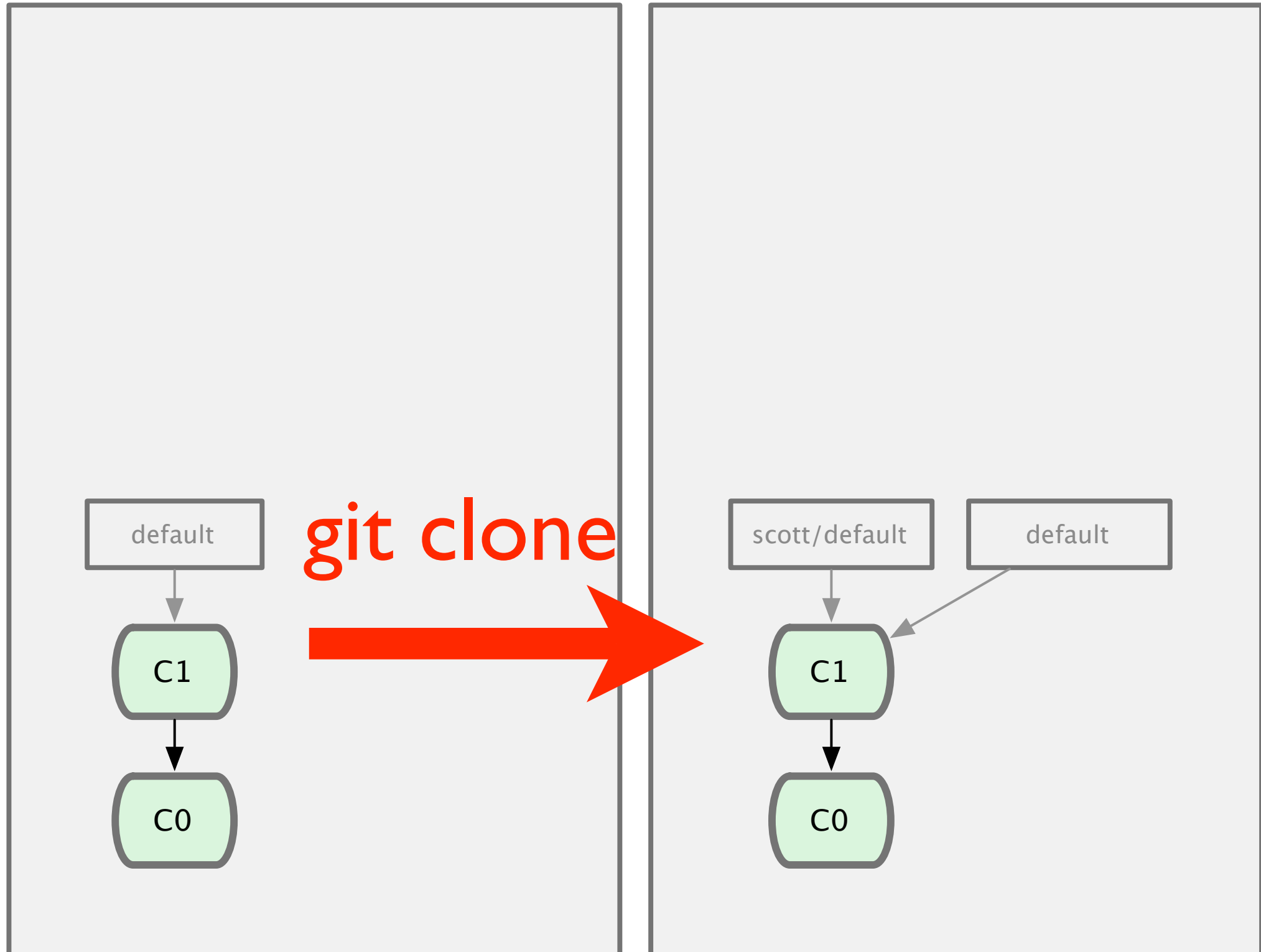


jessica

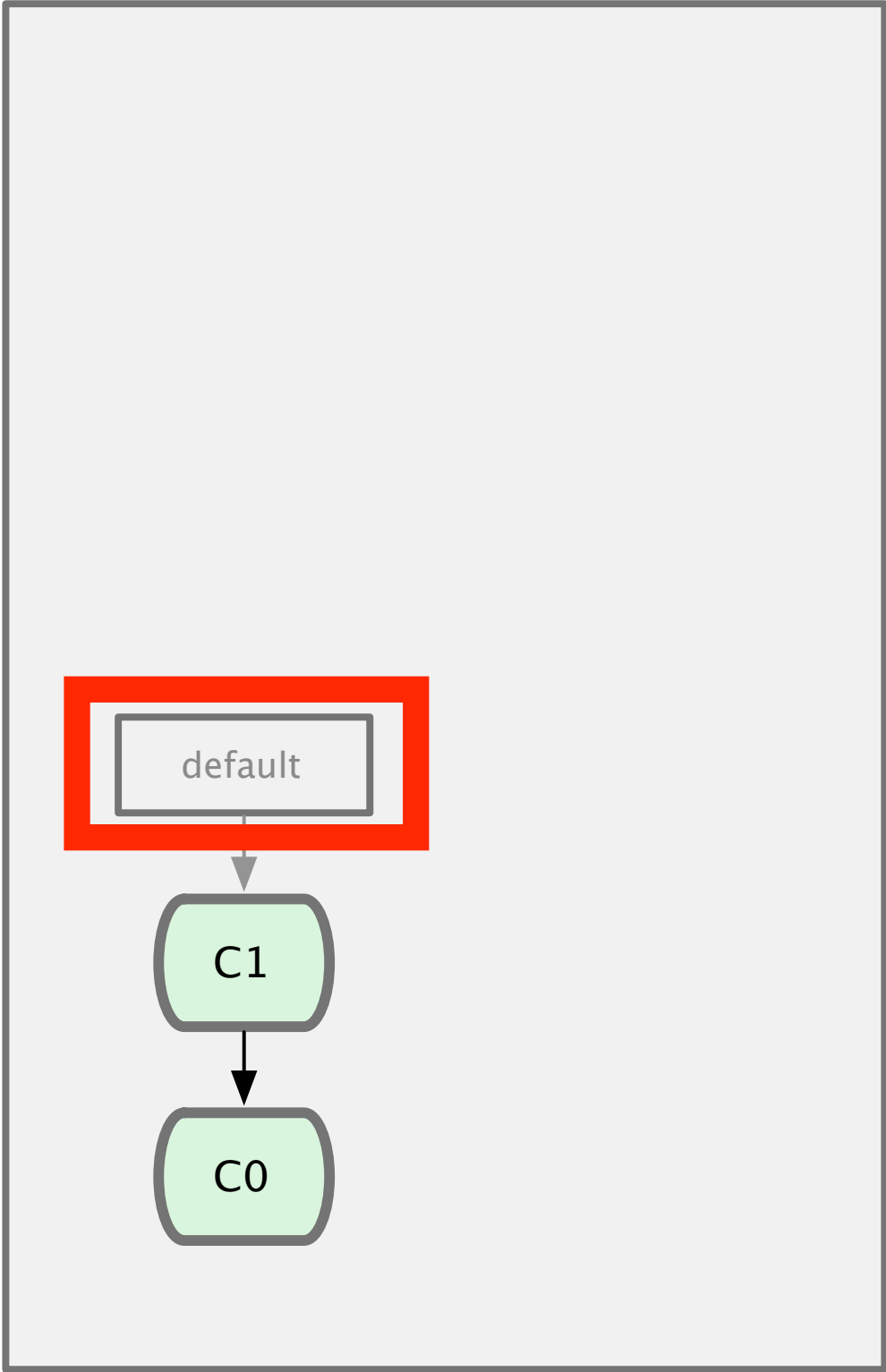


scott

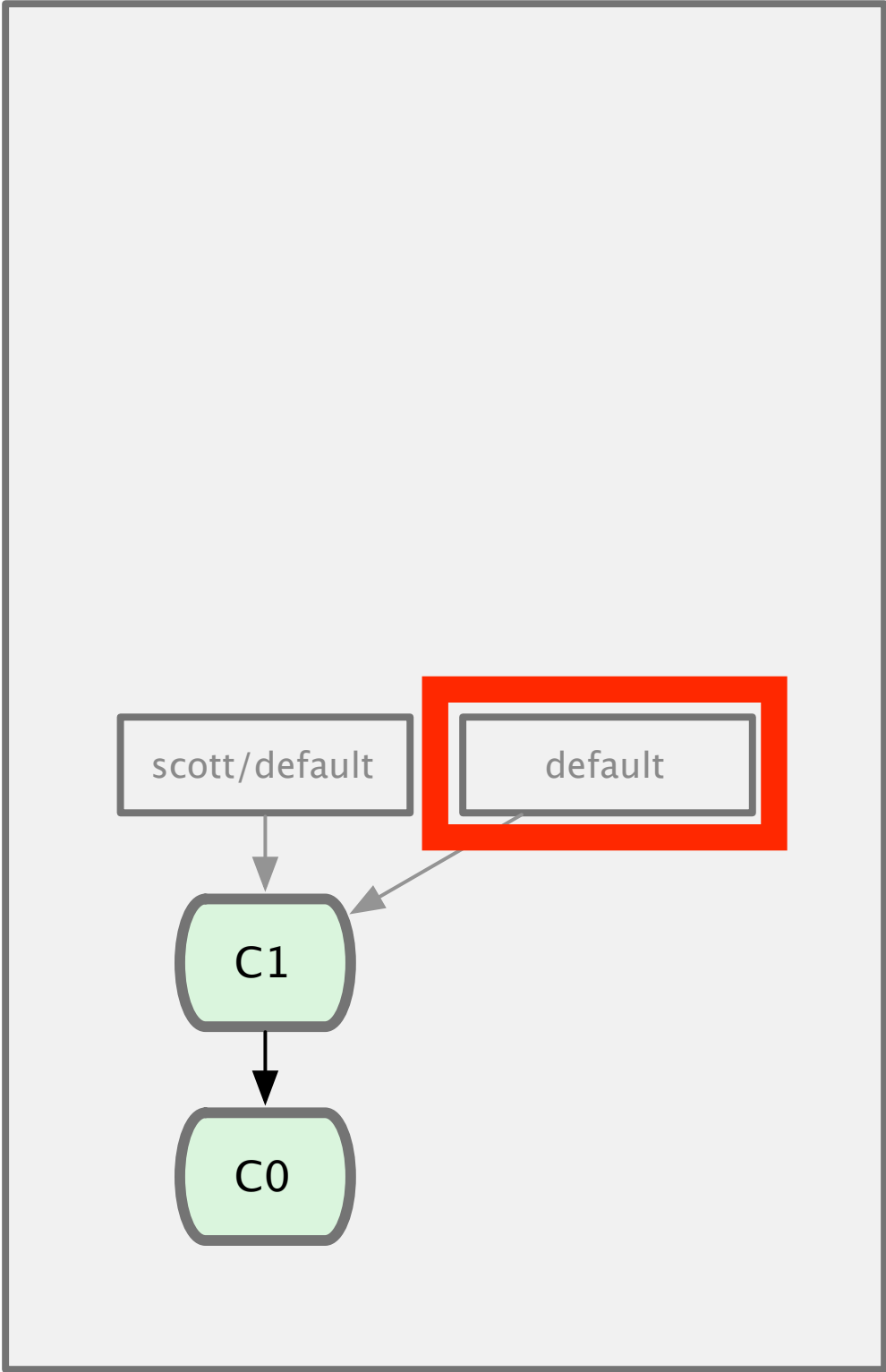
jessica



scott



jessica





scott

default

C1

C0

jessica

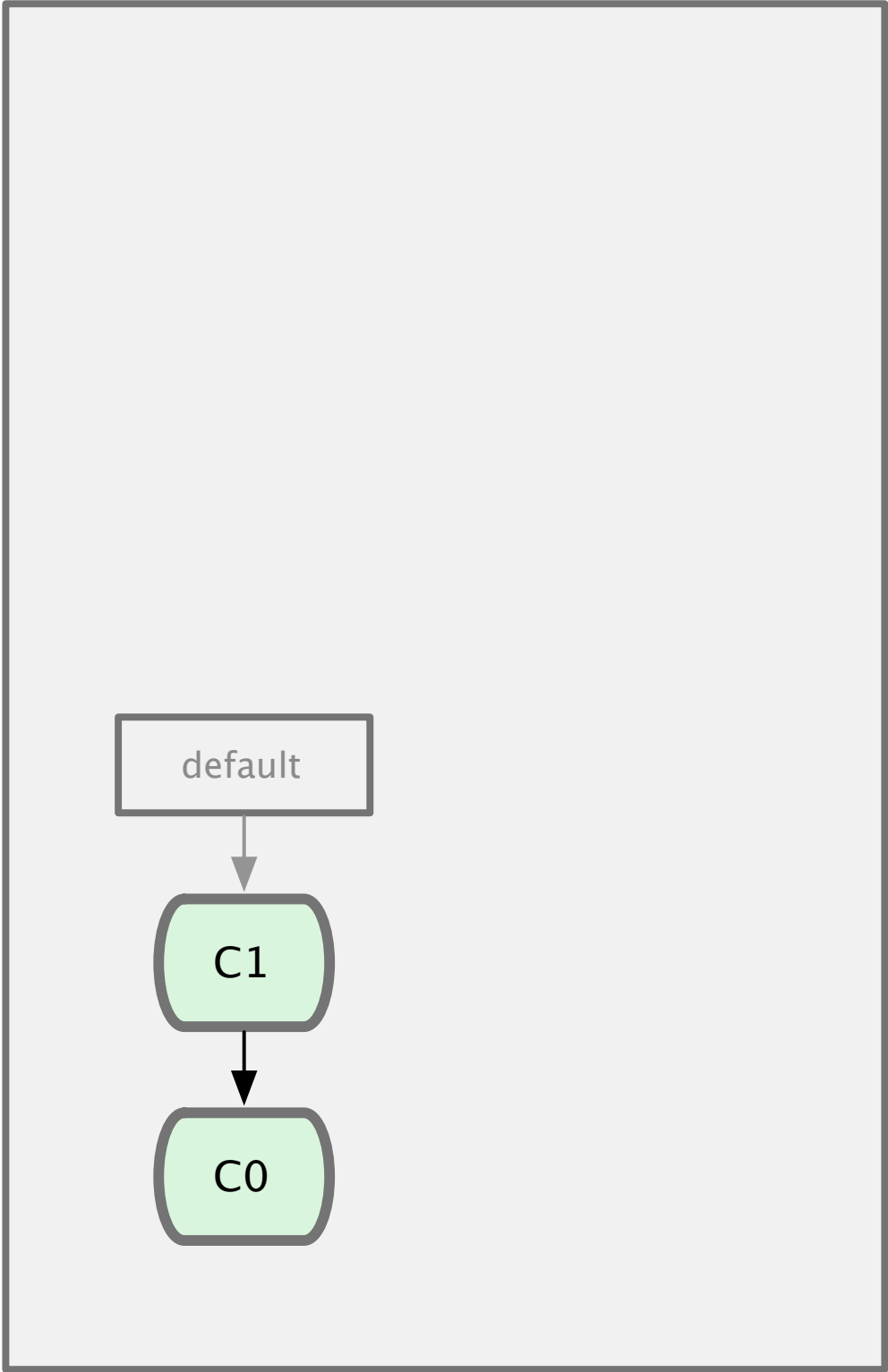
scott/default

default

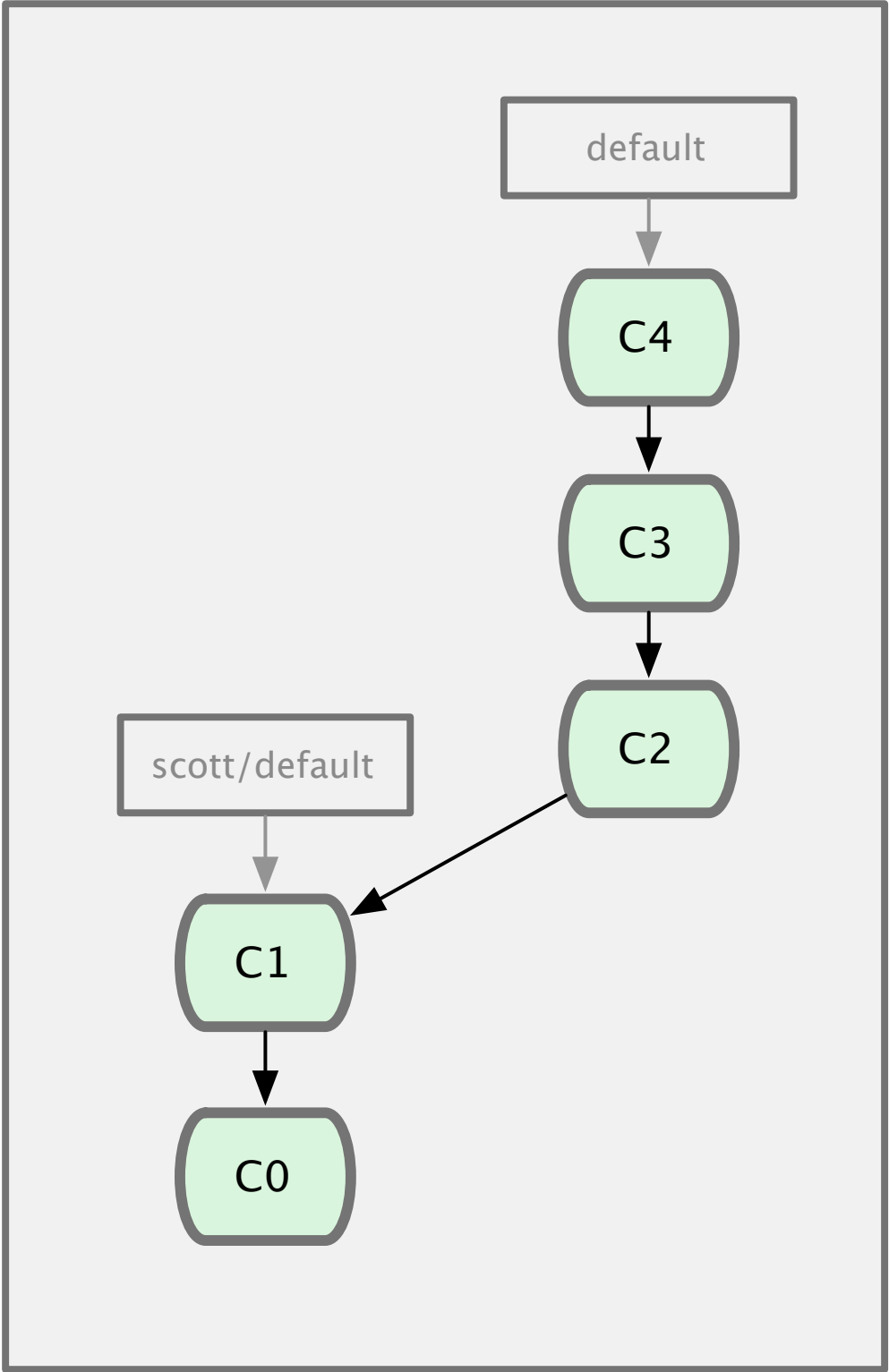
C1

C0

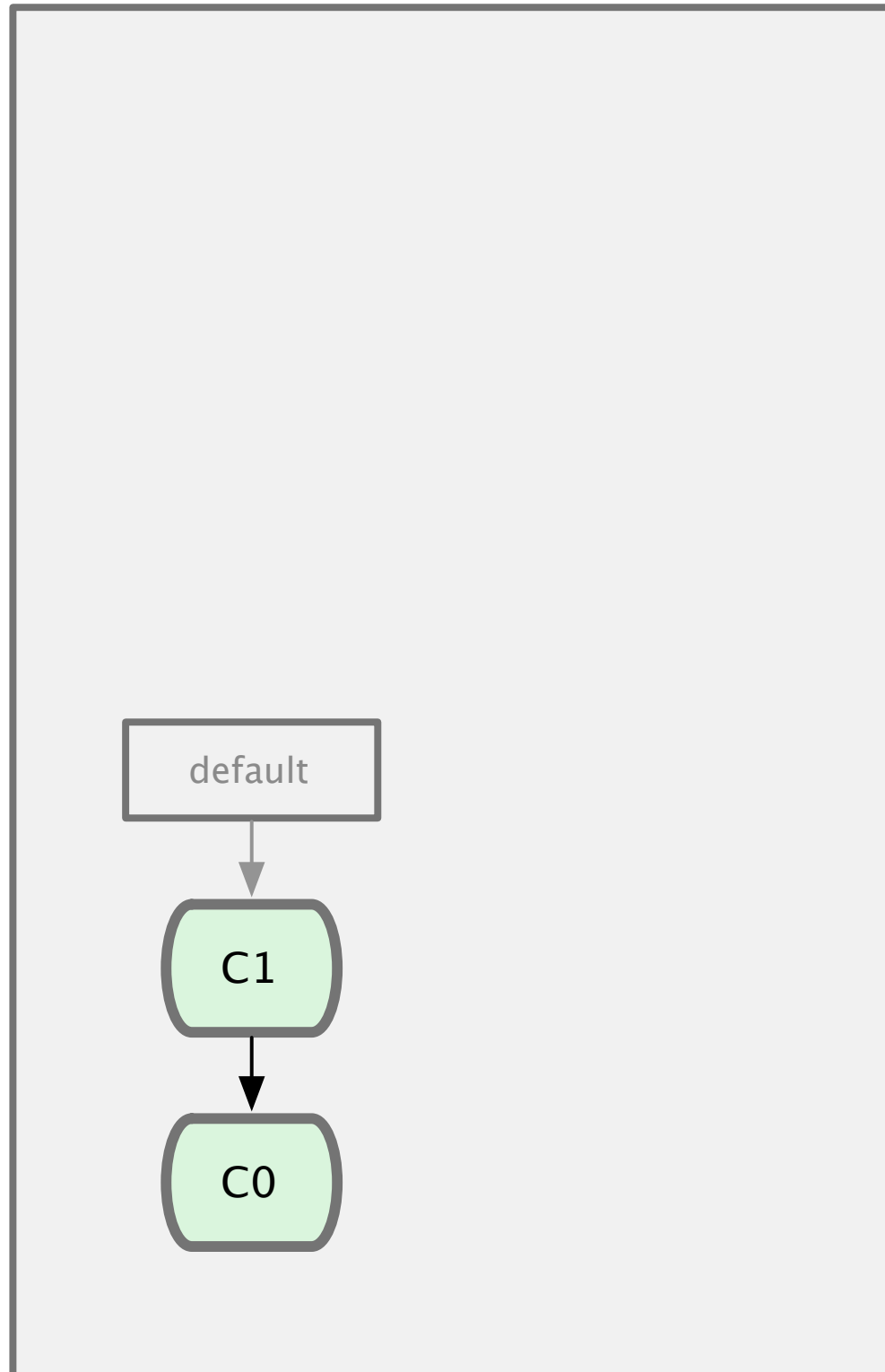
scott



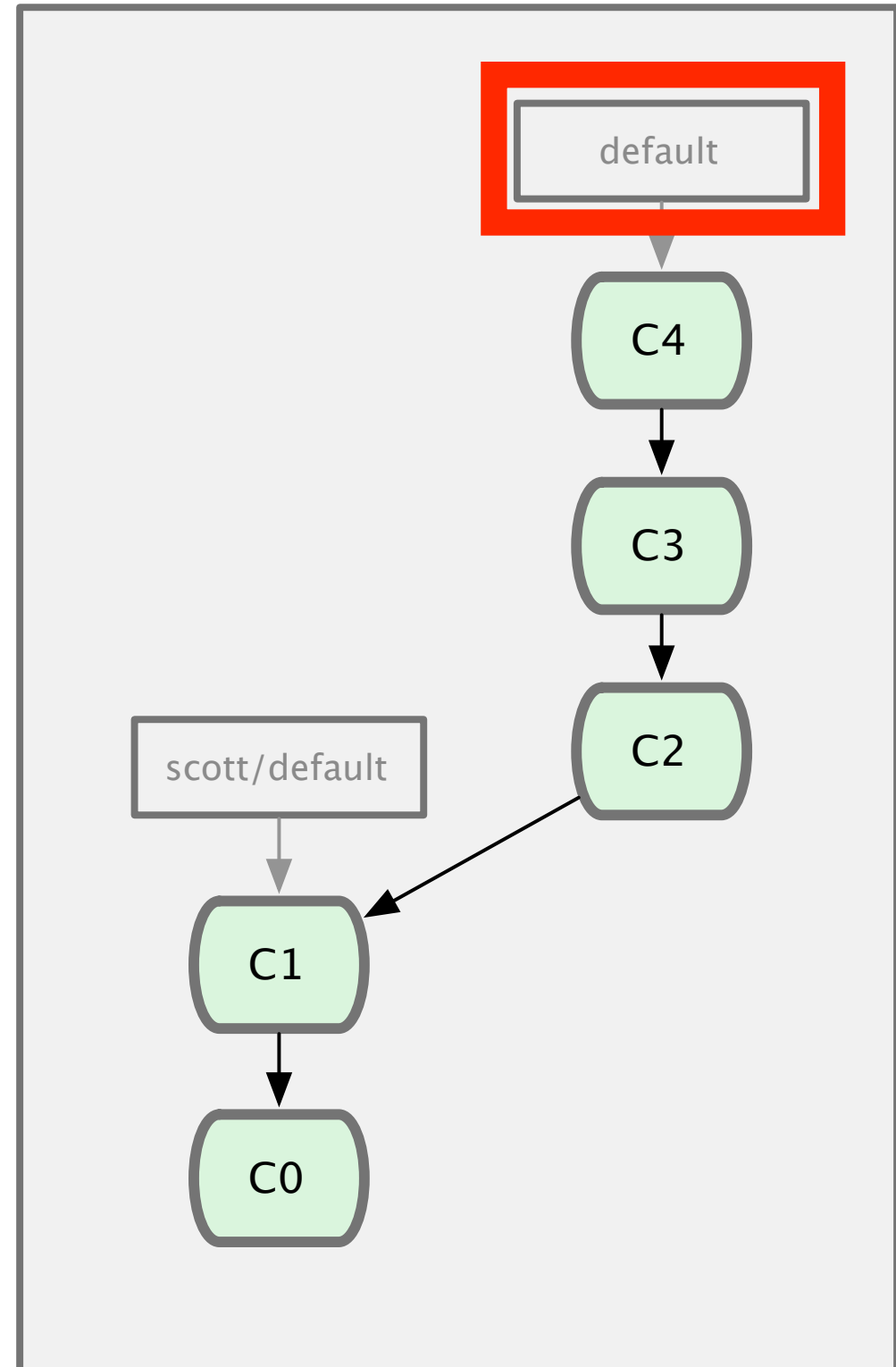
jessica



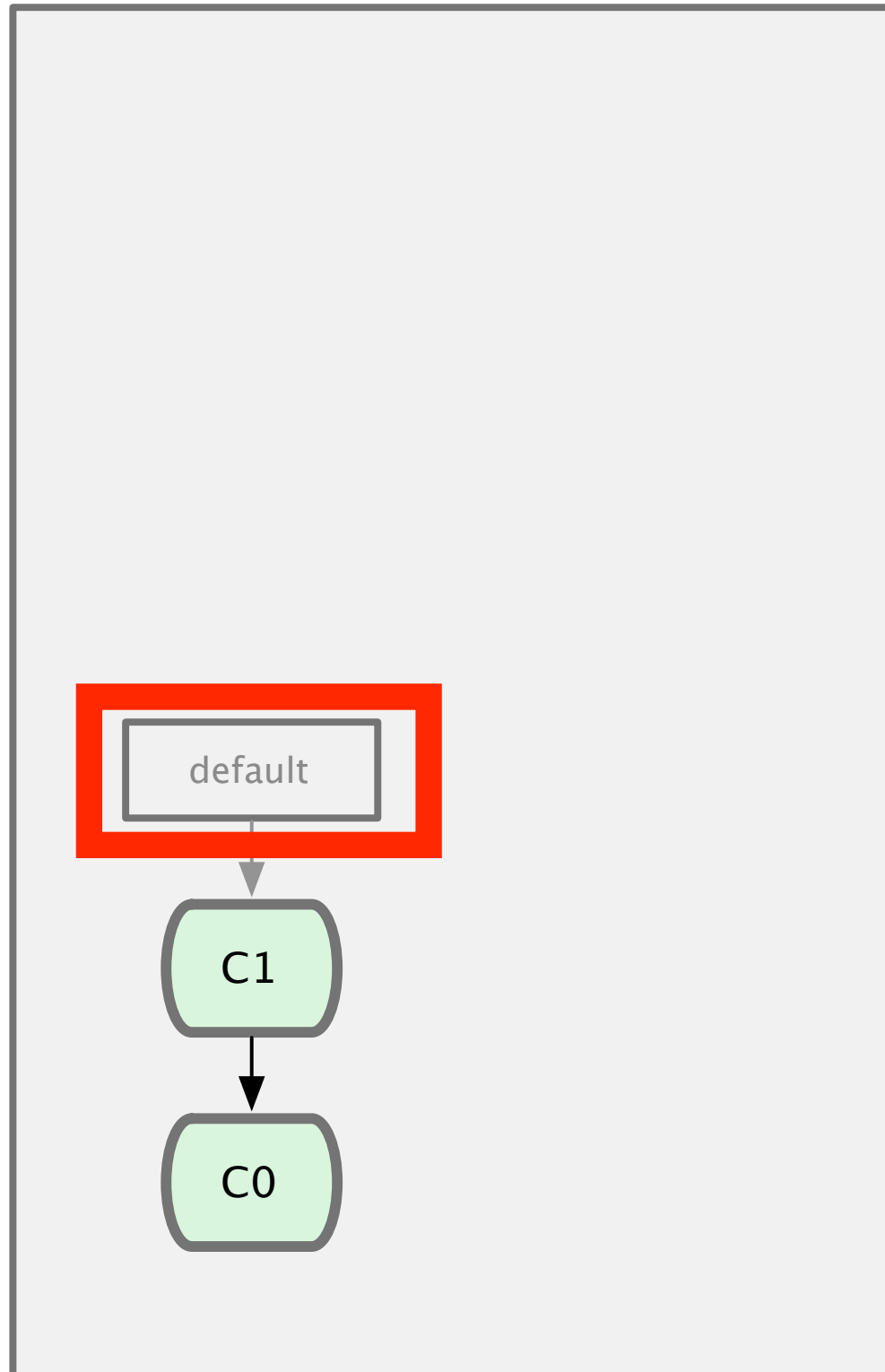
scott



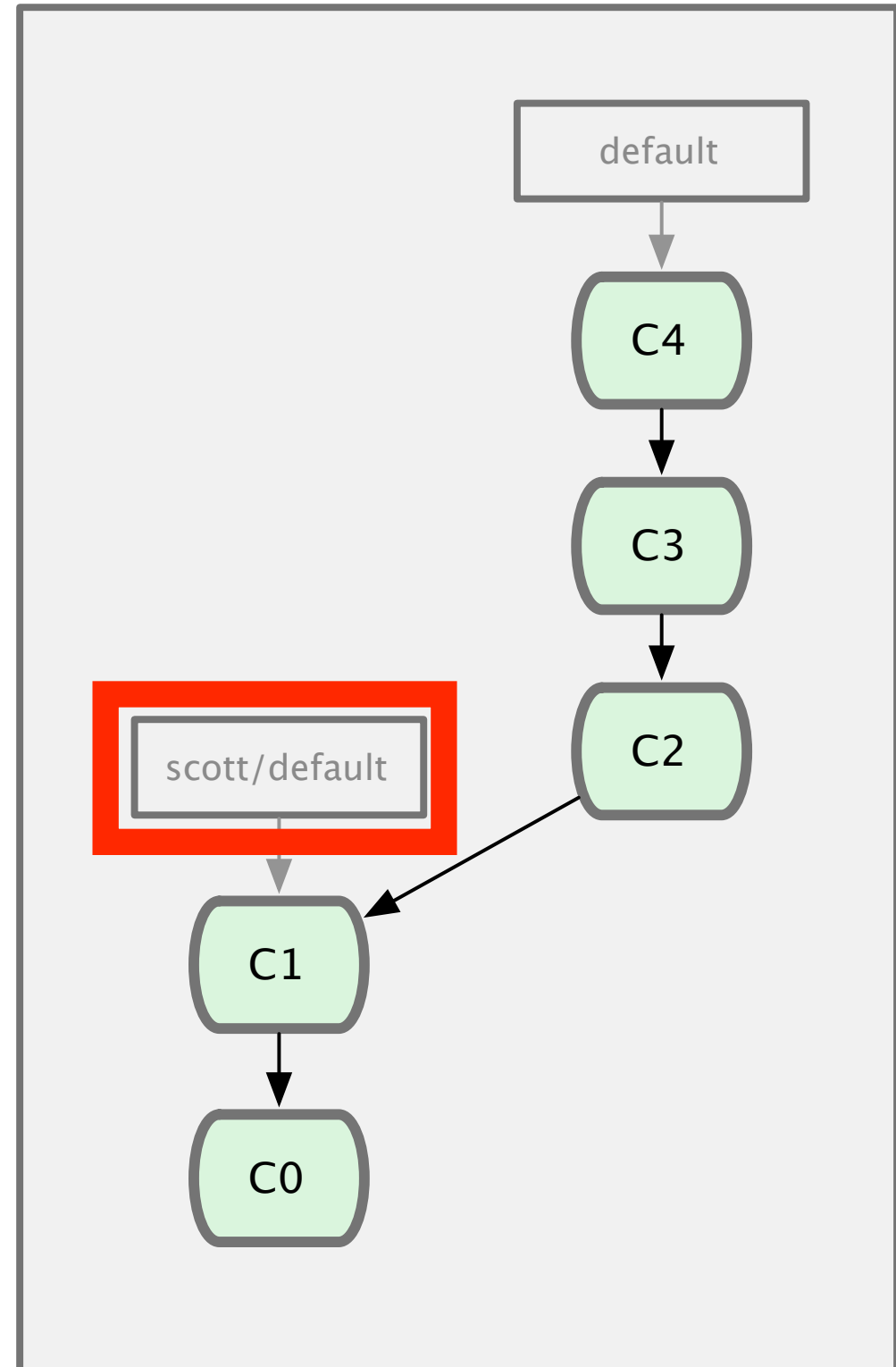
jessica



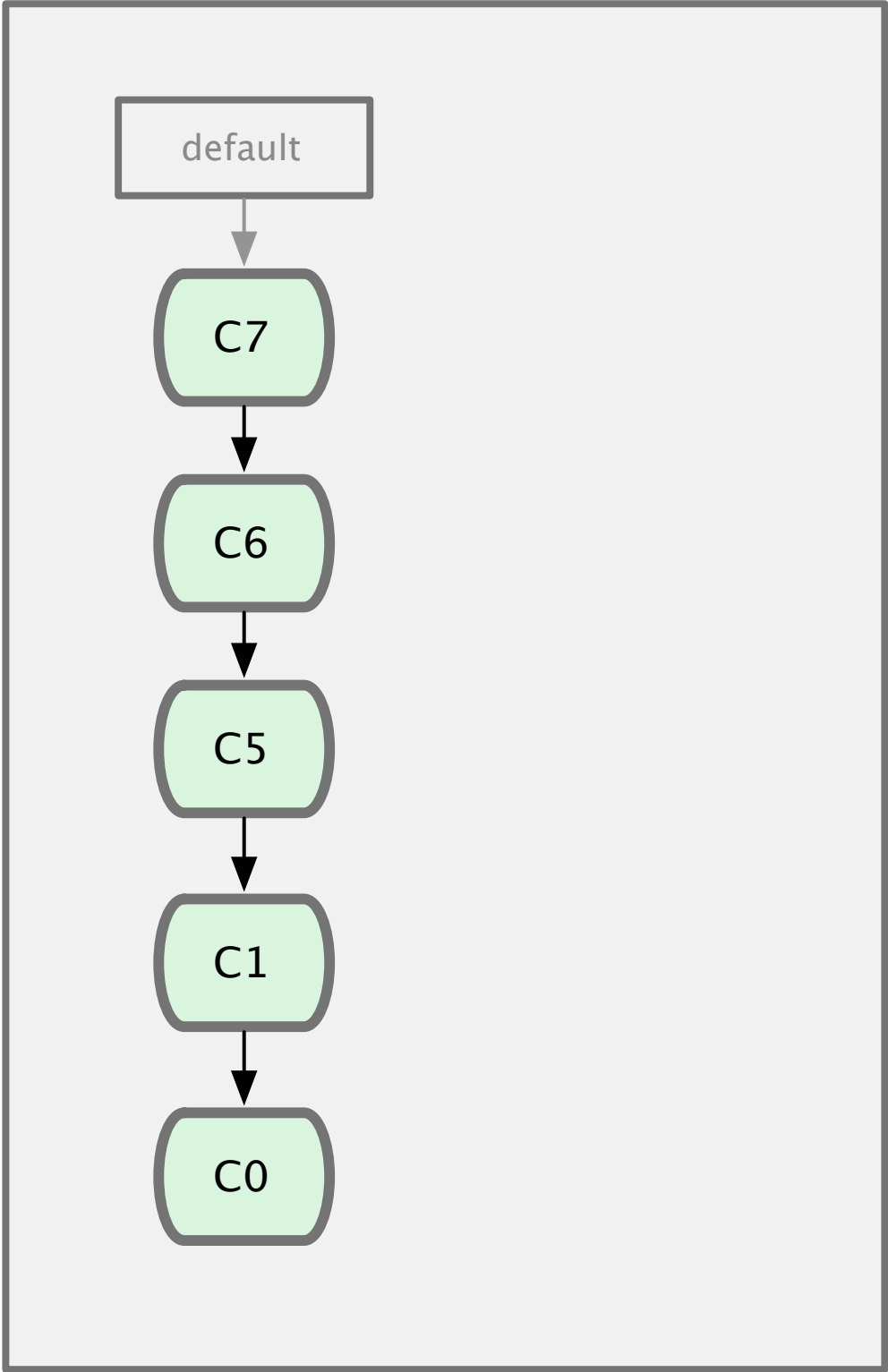
scott



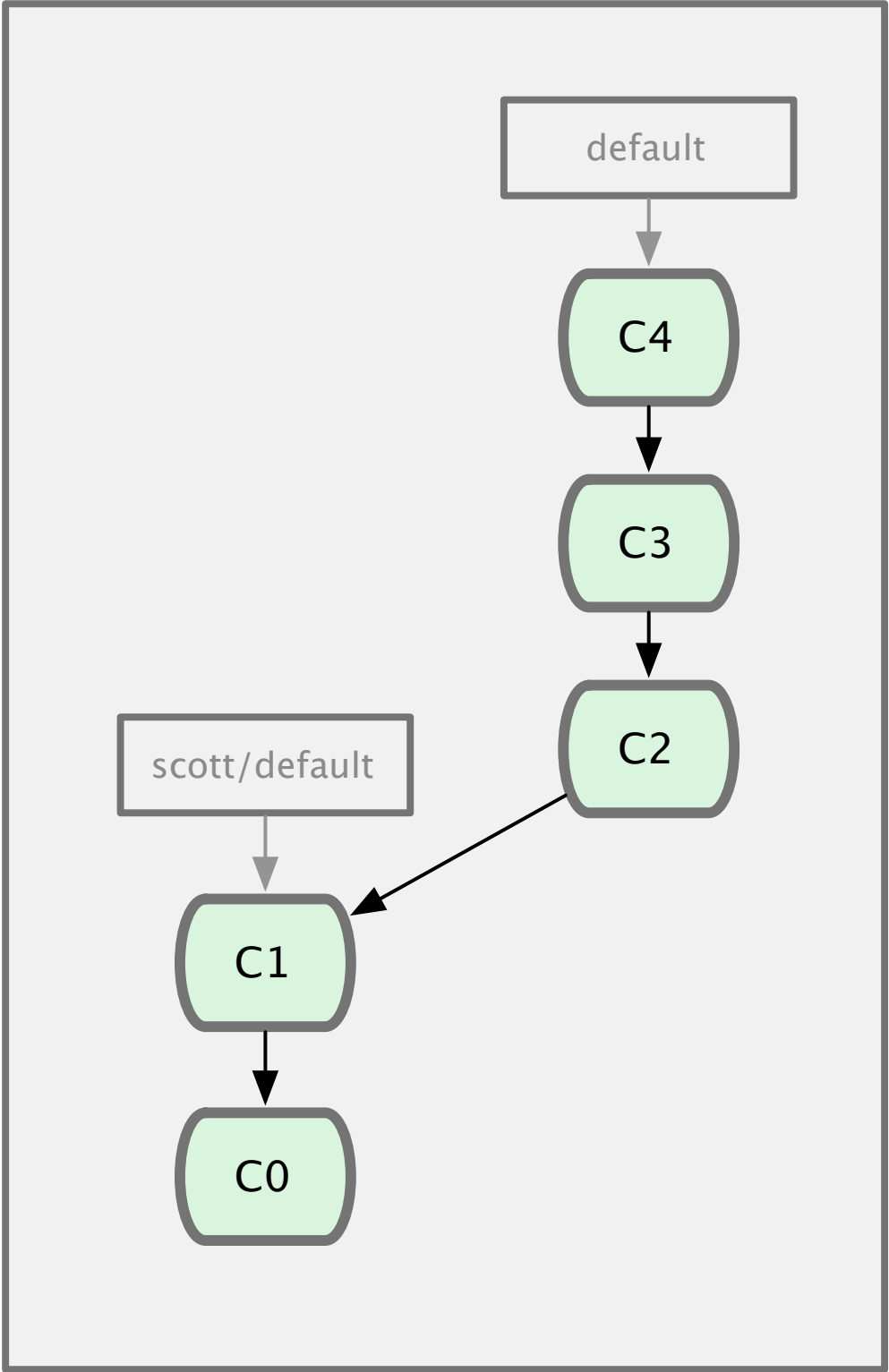
jessica



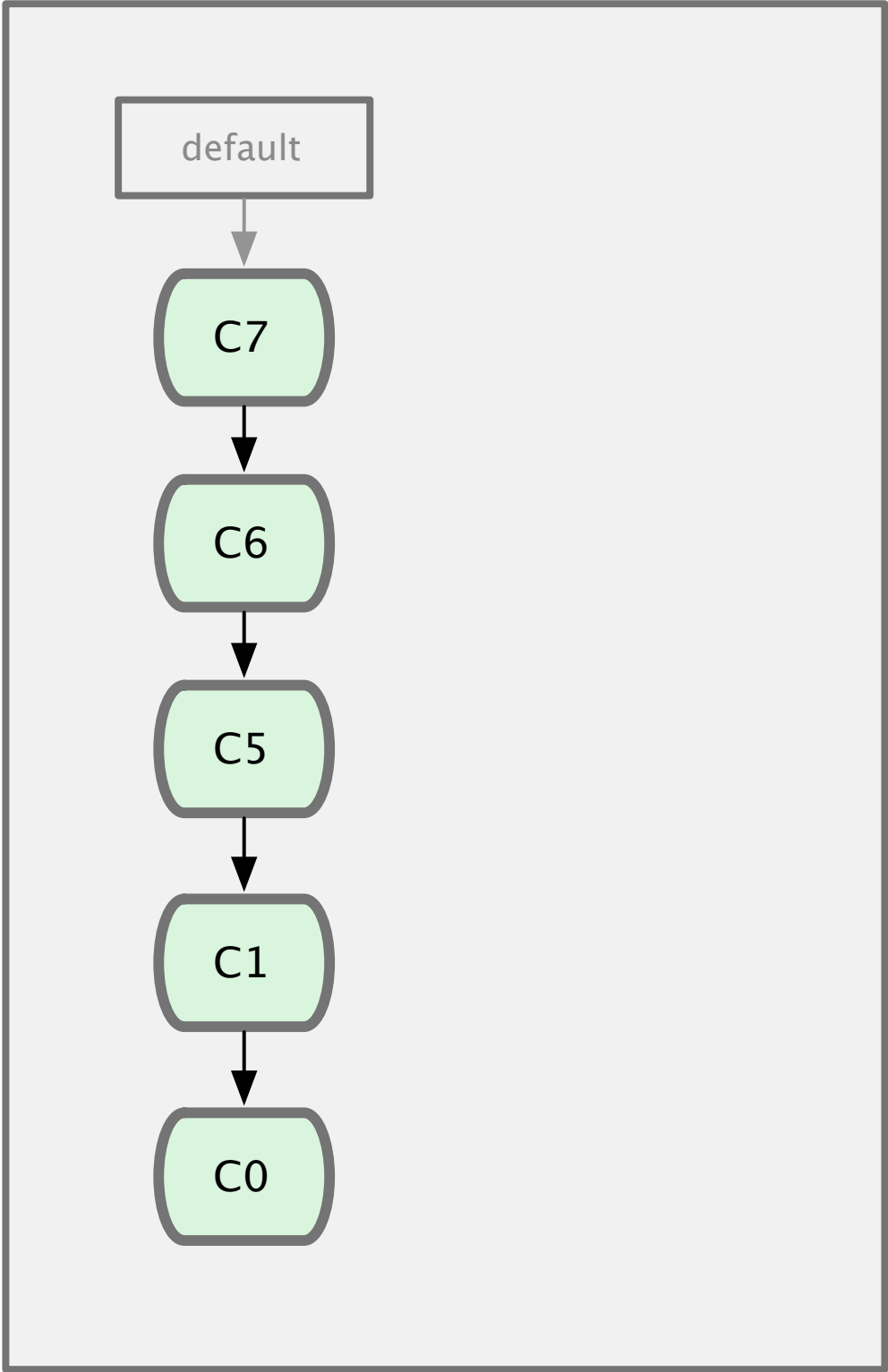
scott



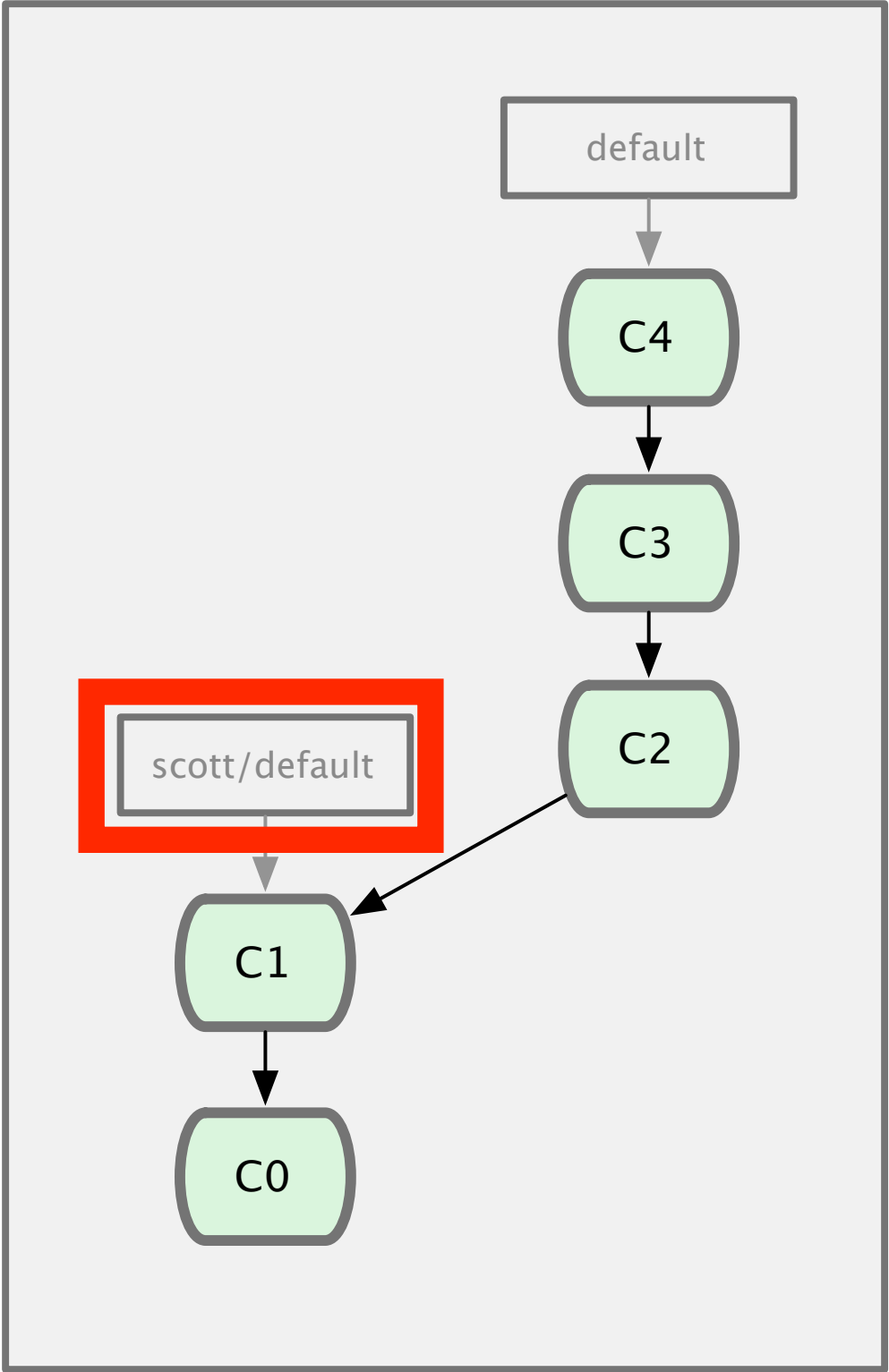
jessica



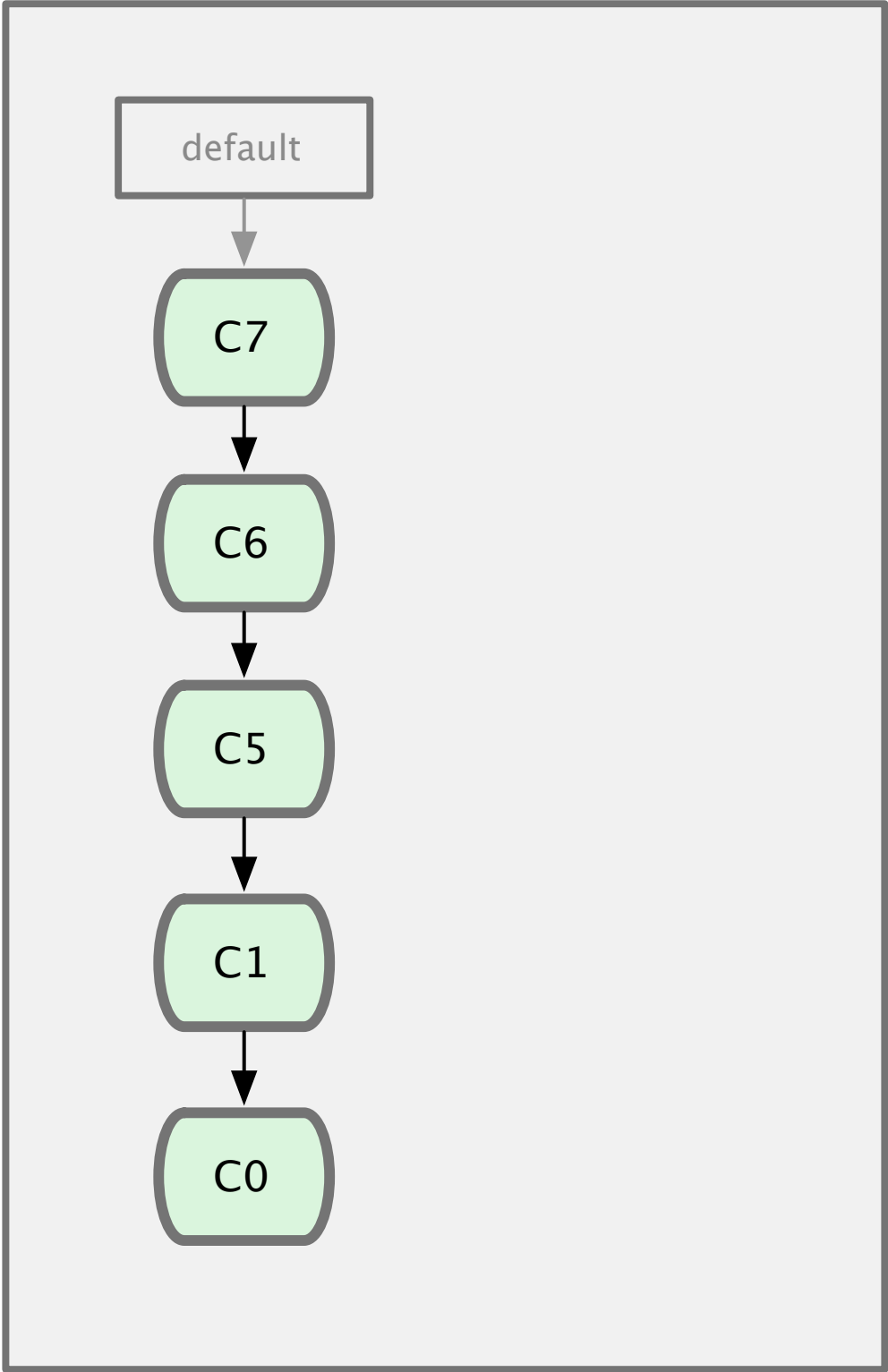
scott



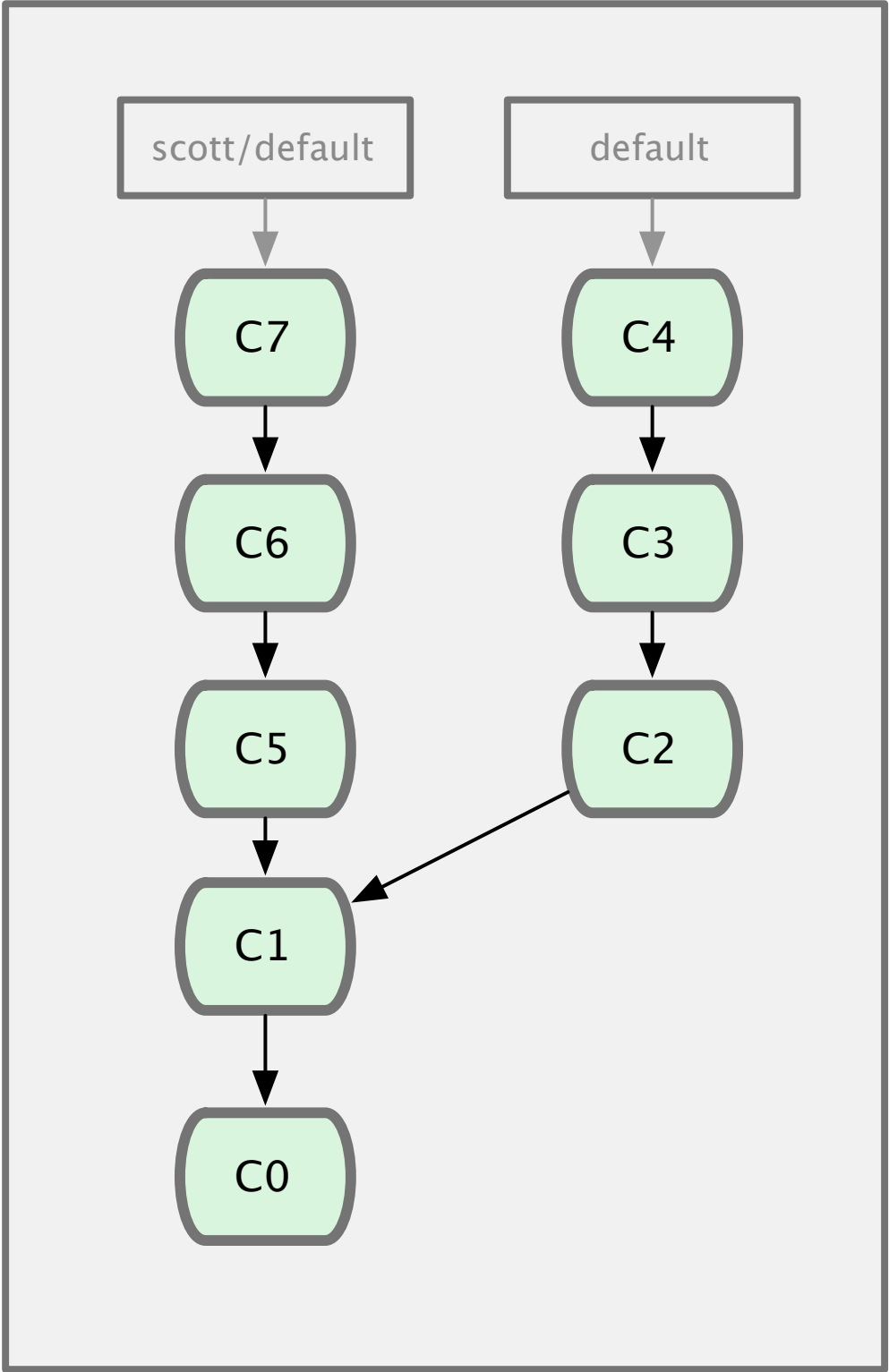
jessica



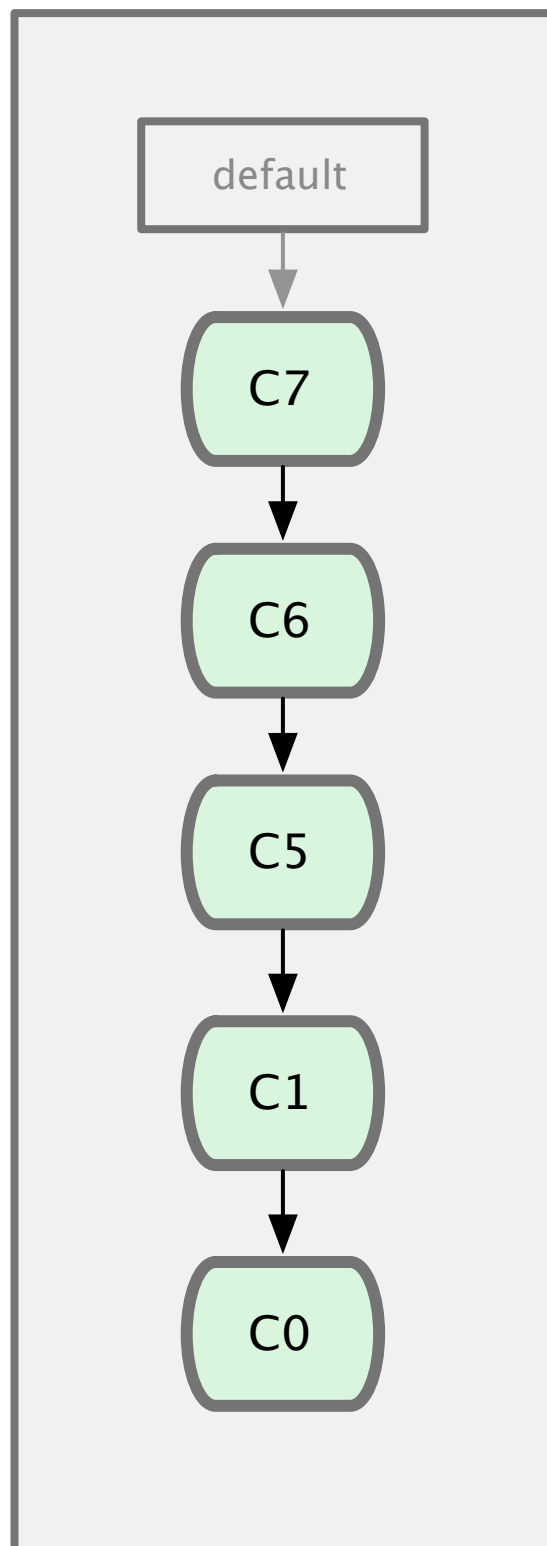
scott



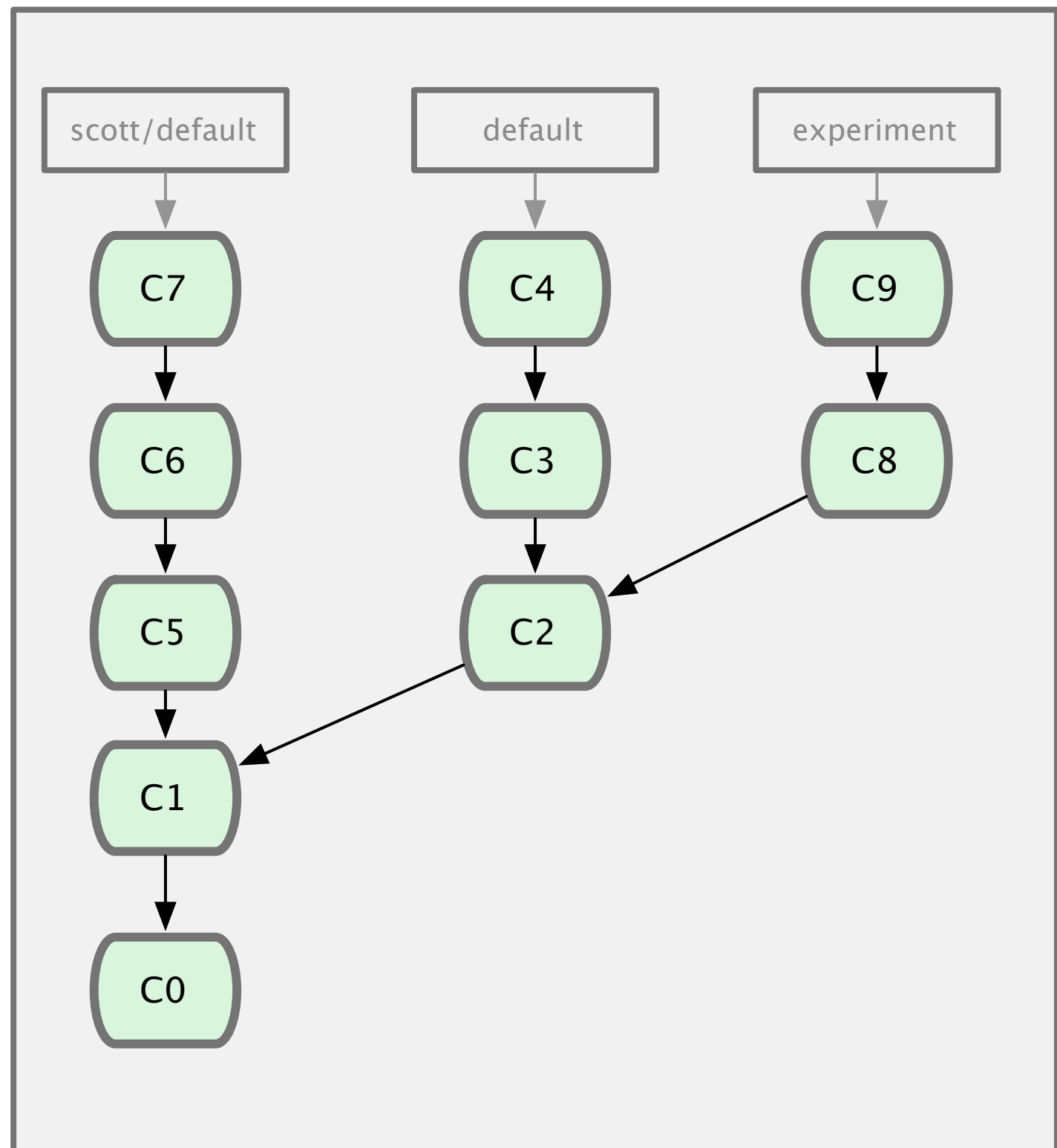
jessica



scott

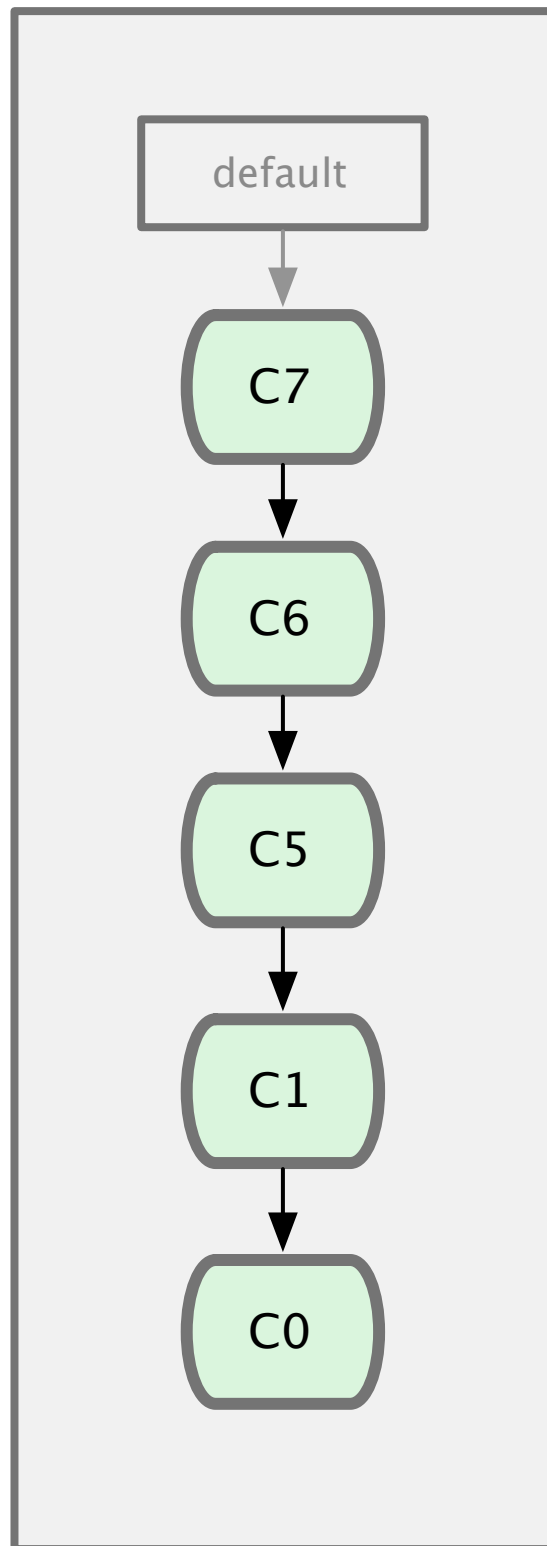


jessica

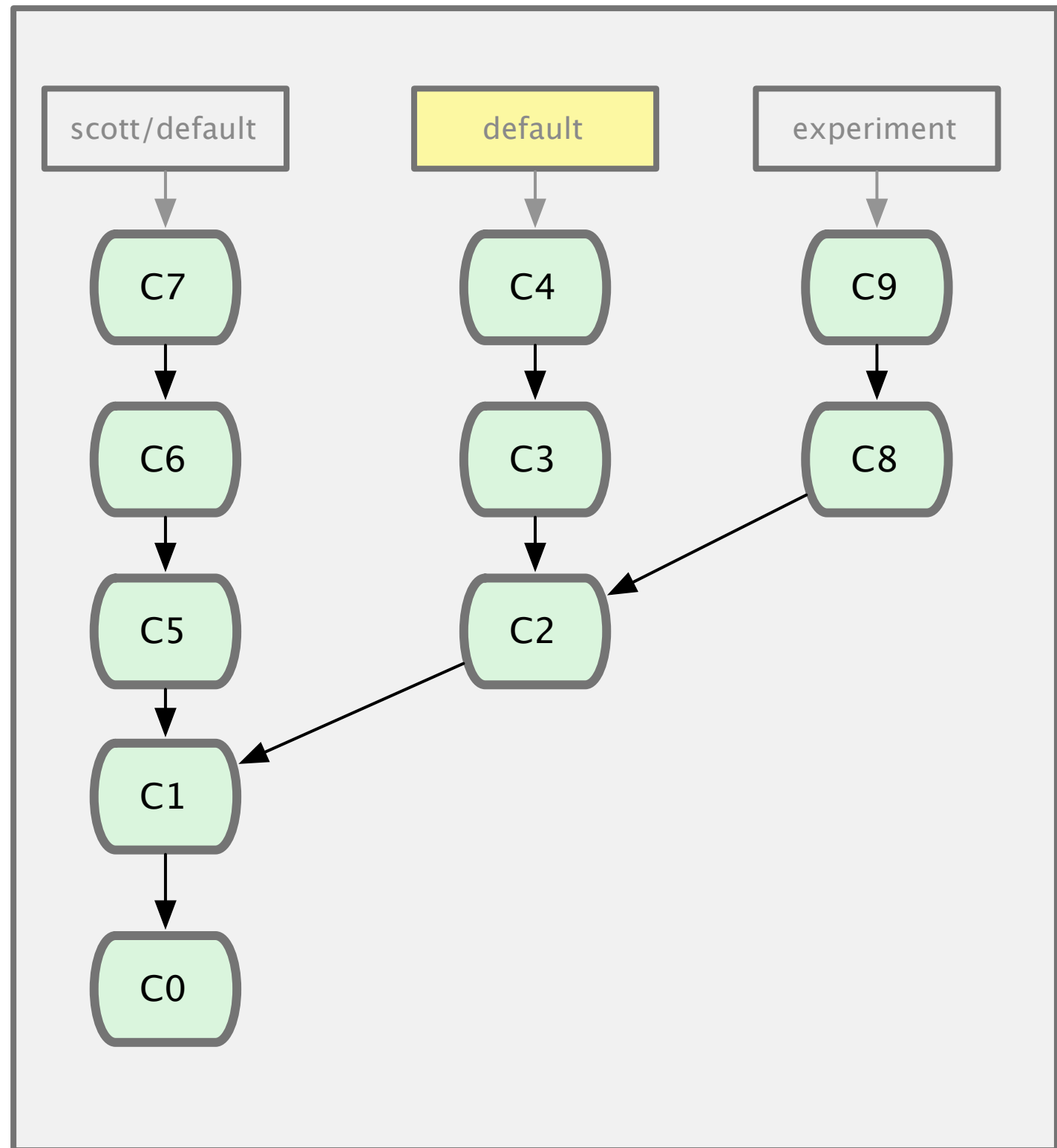




scott



jessica



scott

jessica

default

scott/default

default

experiment

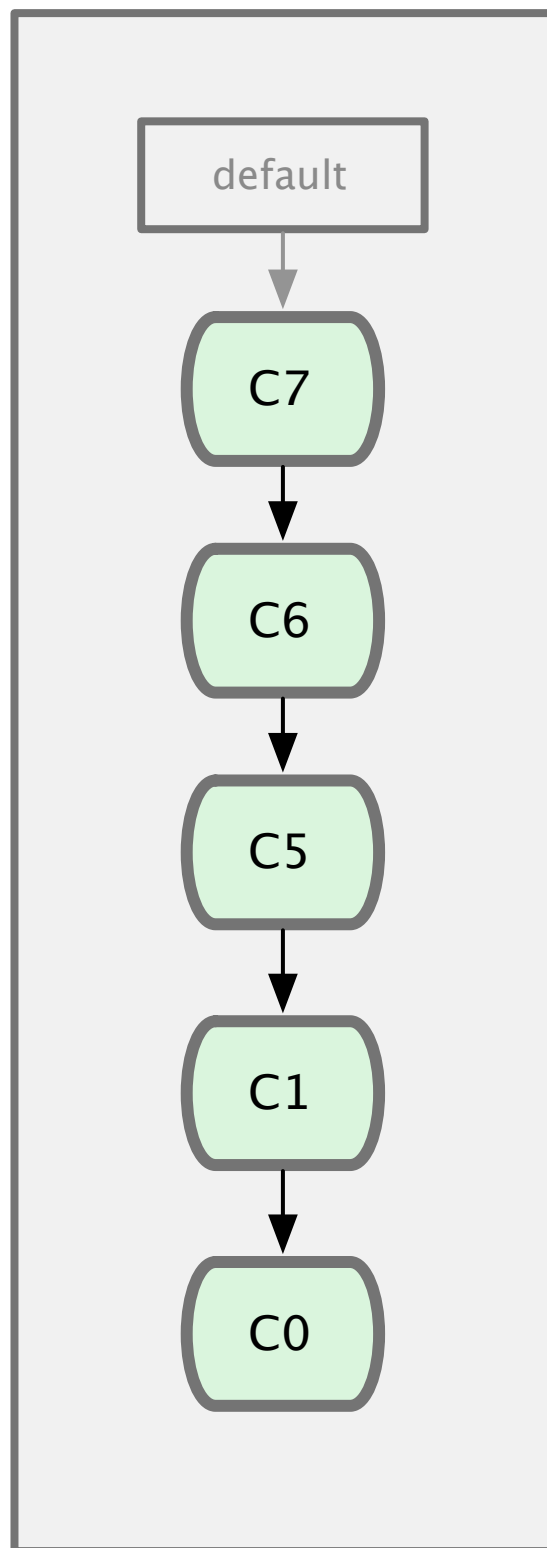
git merge experiment

git merge scott/default

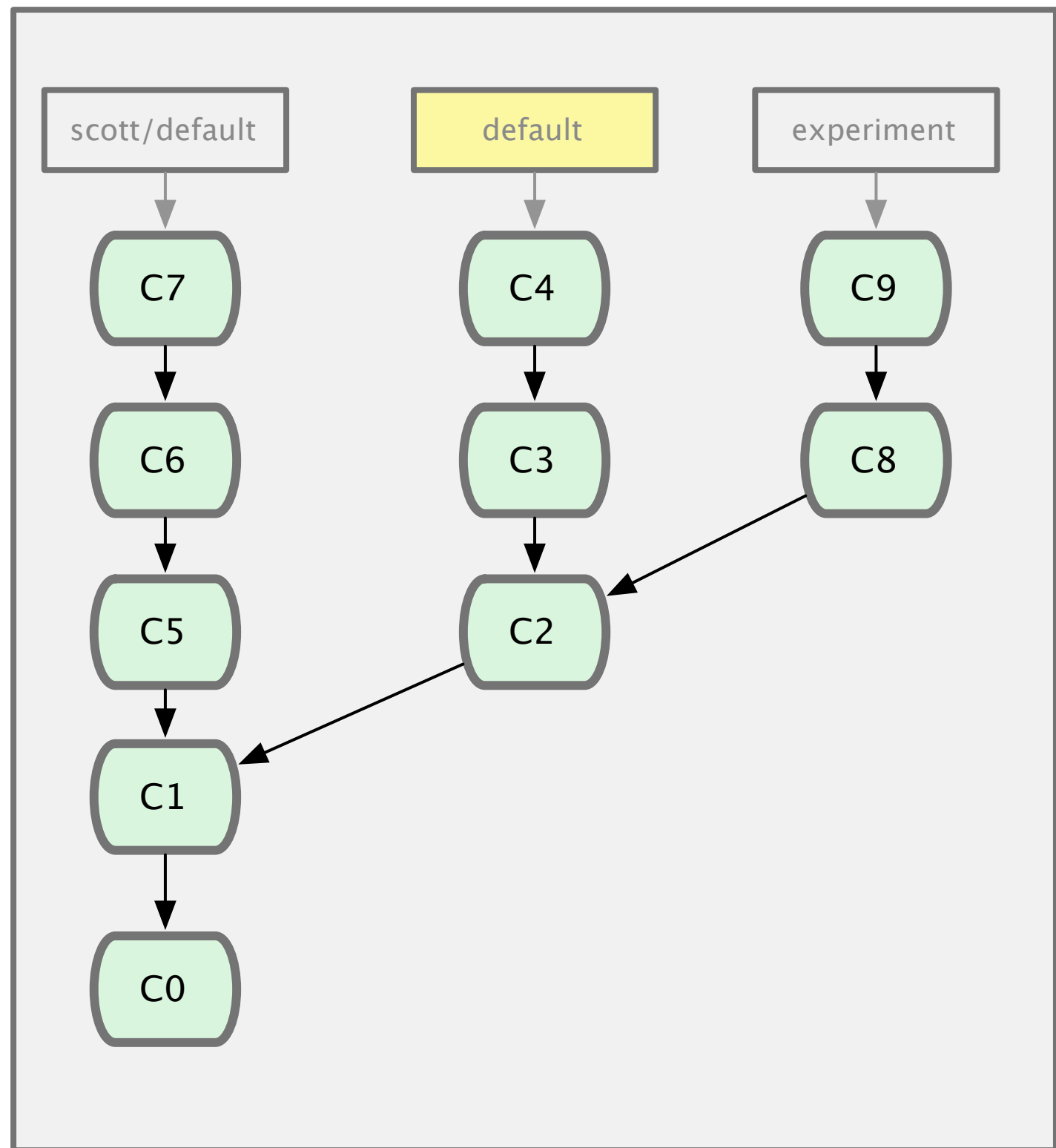
C0

C0

scott

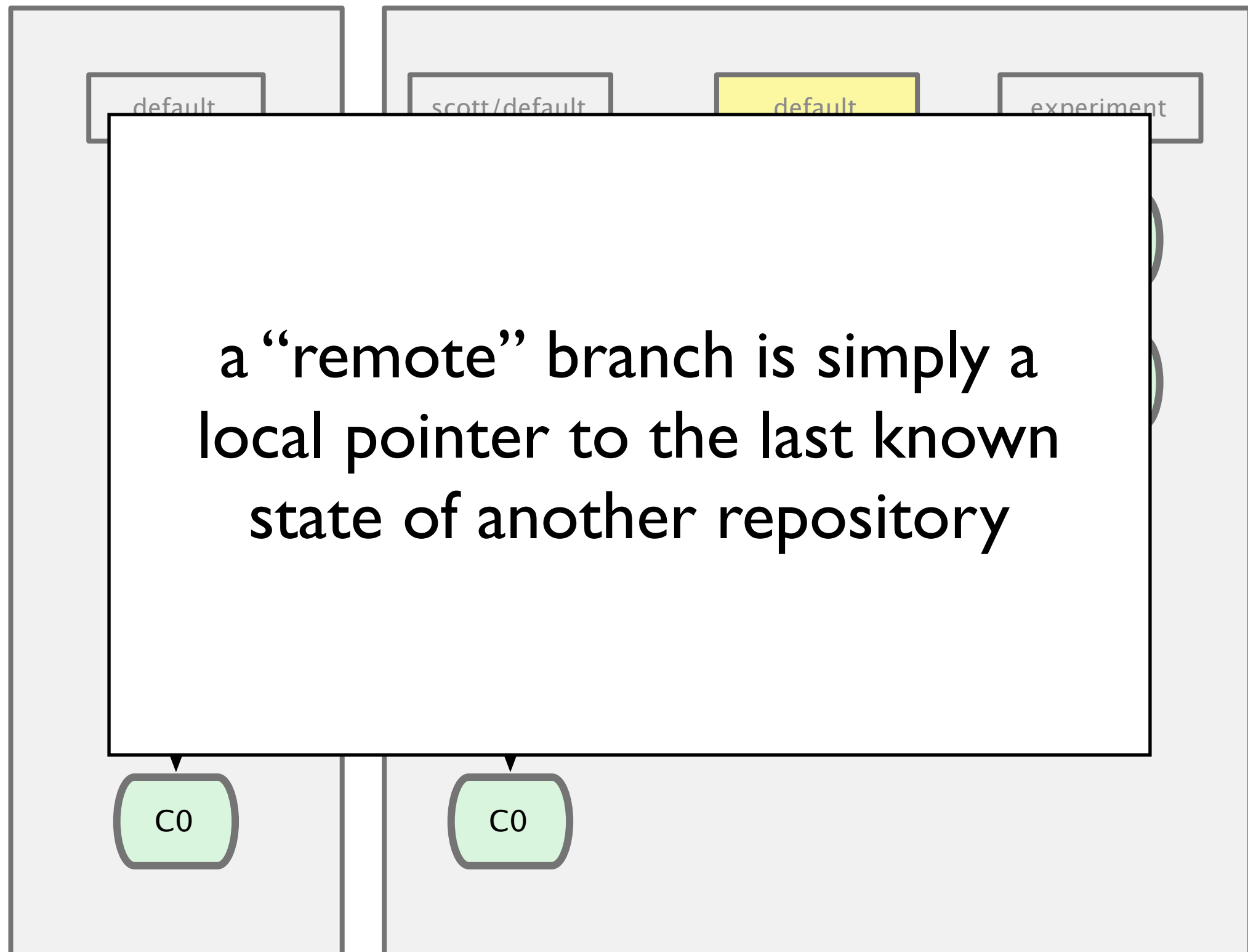


jessica



scott

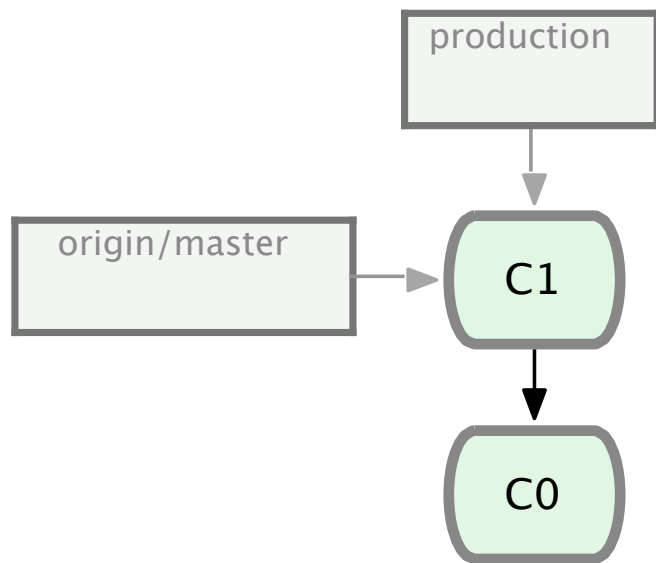
jessica



**Why is this cool?**

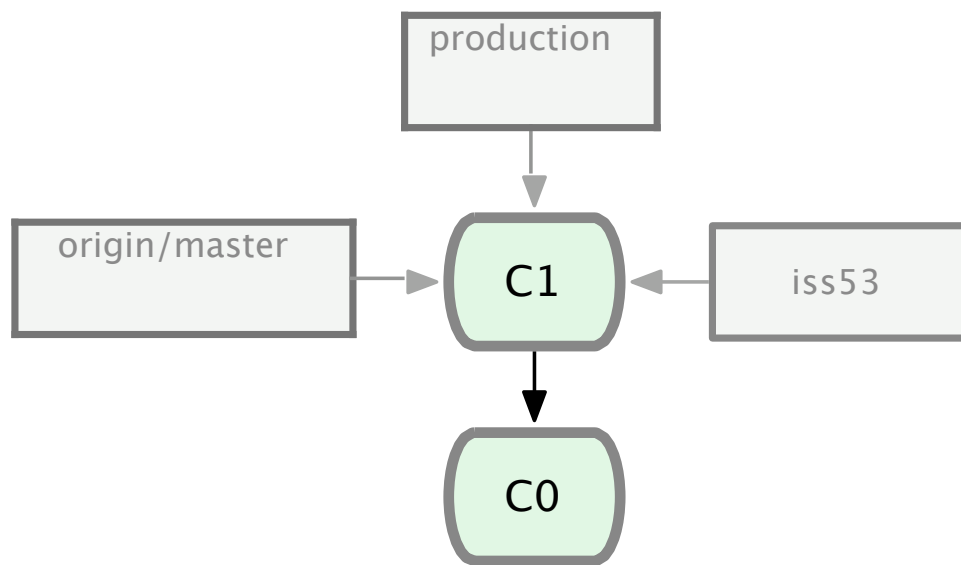
**non-linear development**

- 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'
- etc

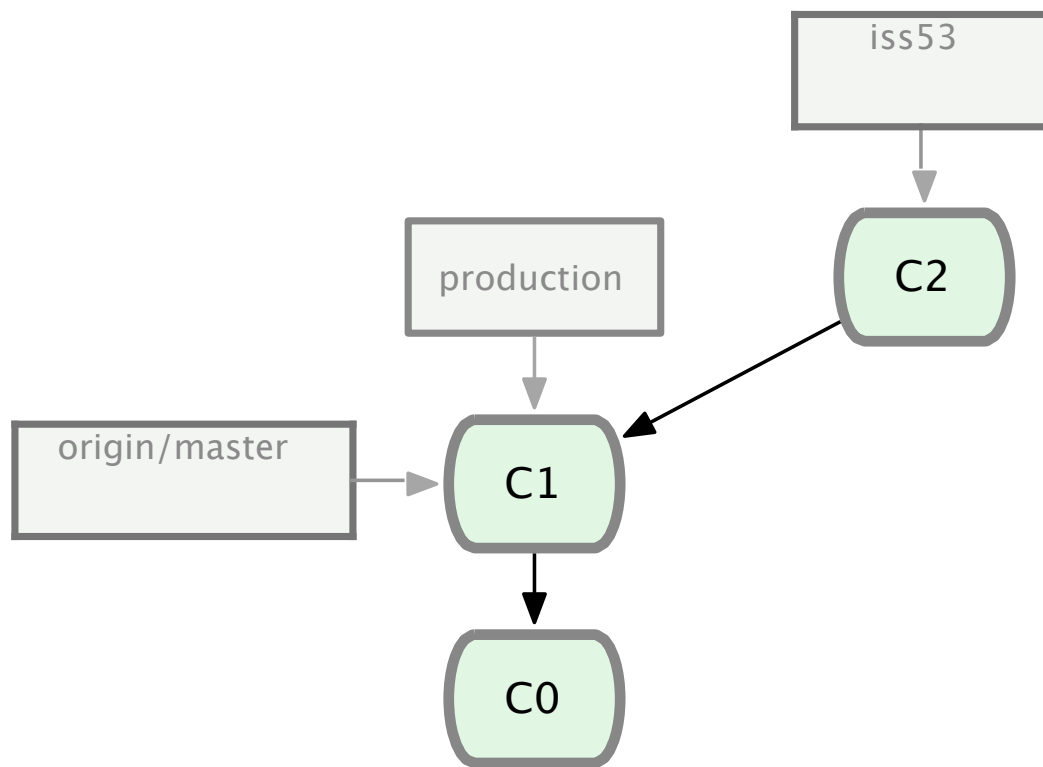


git clone

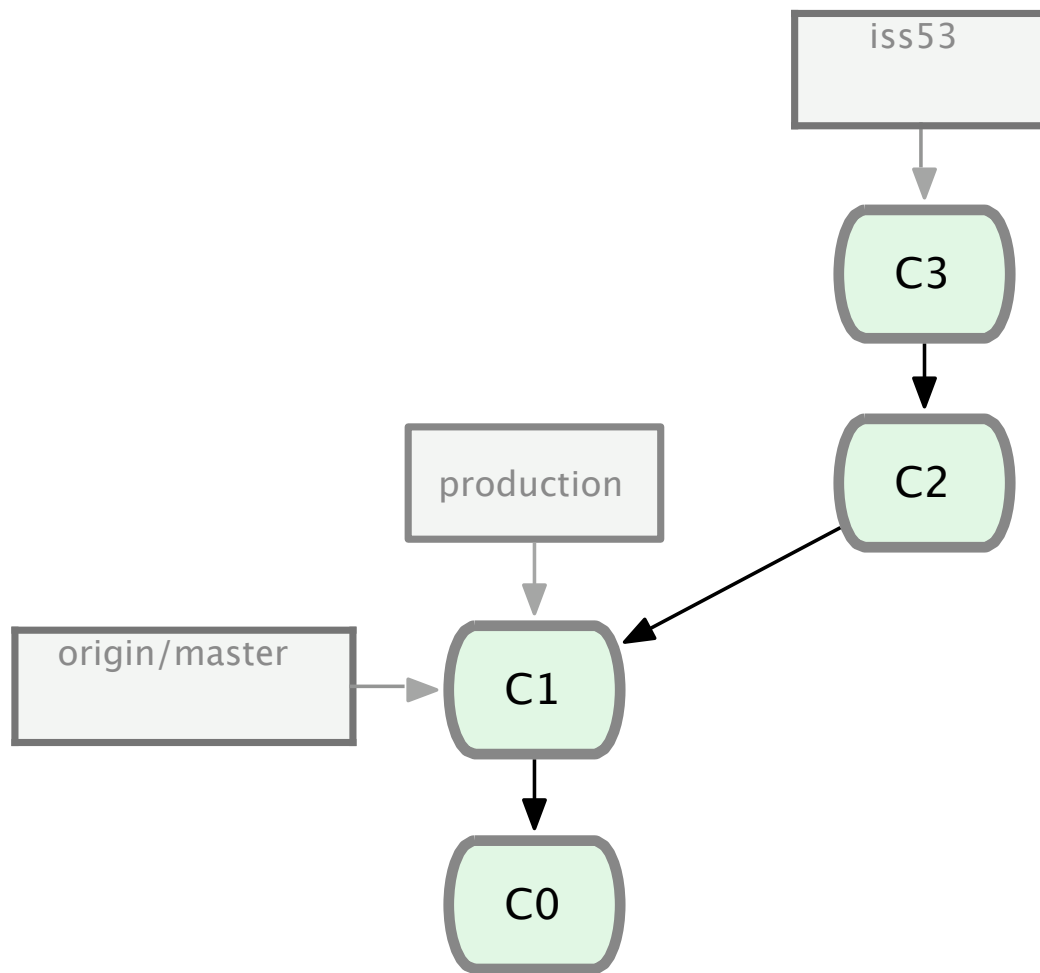




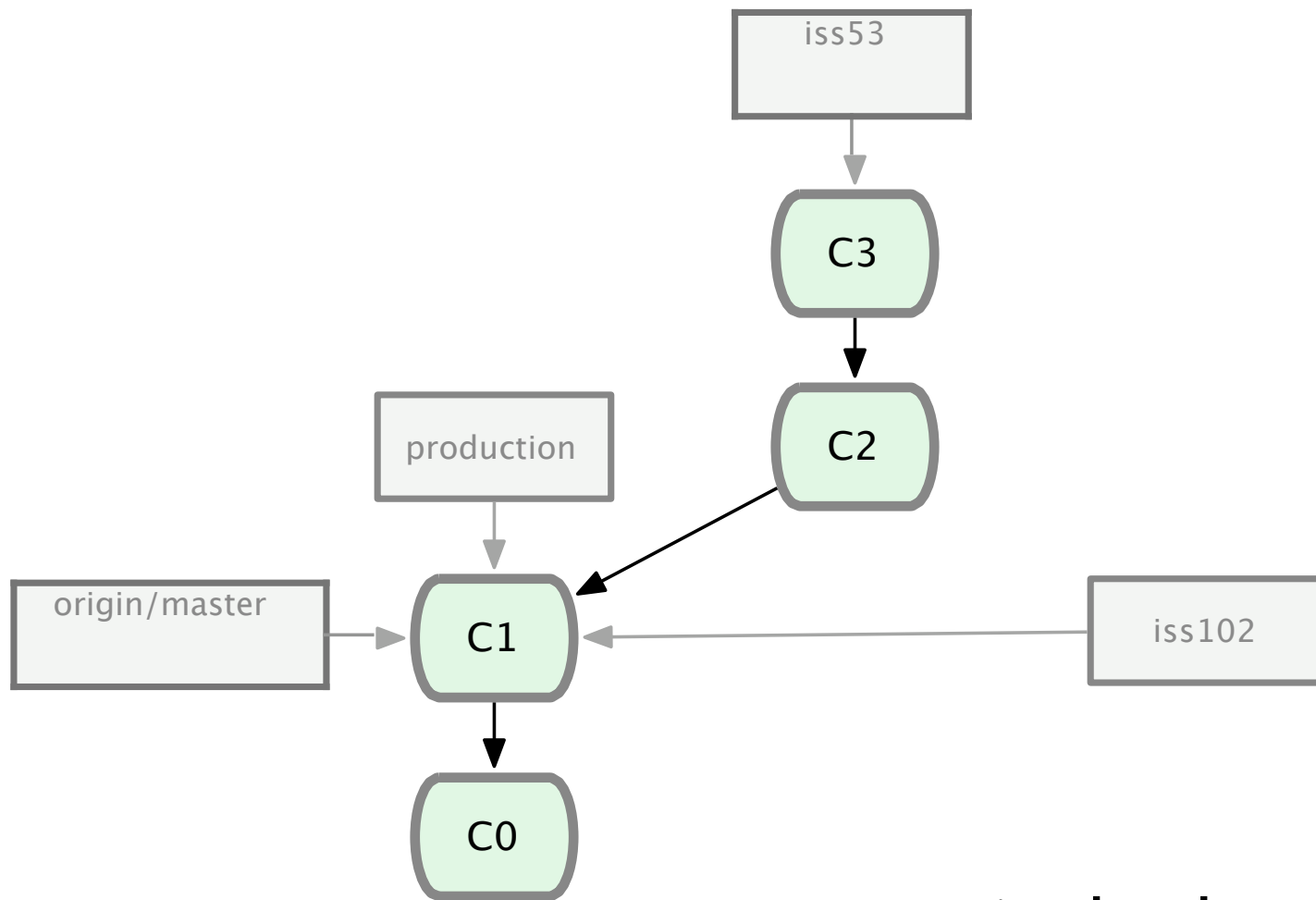
**git checkout -b iss53**



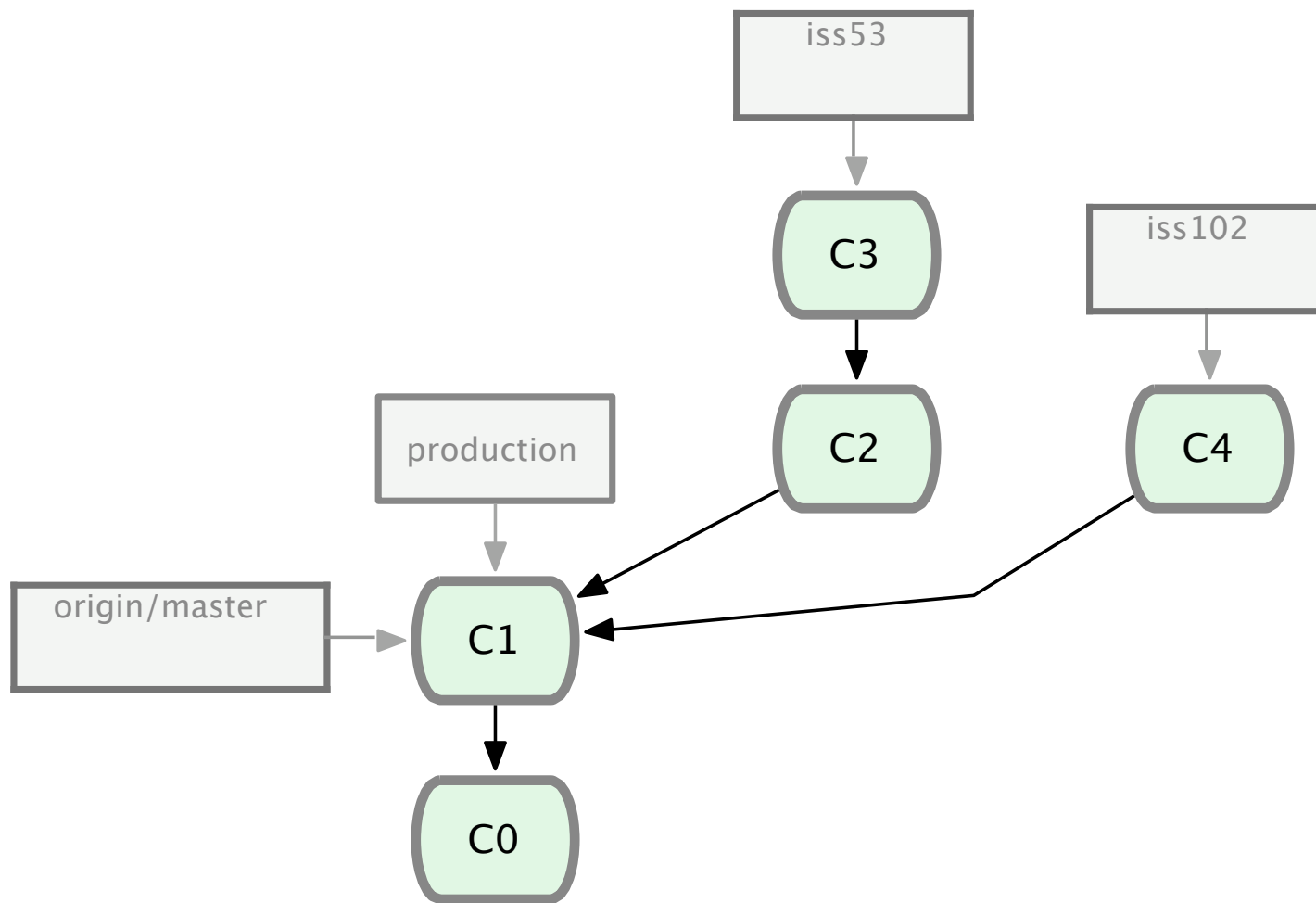
git commit



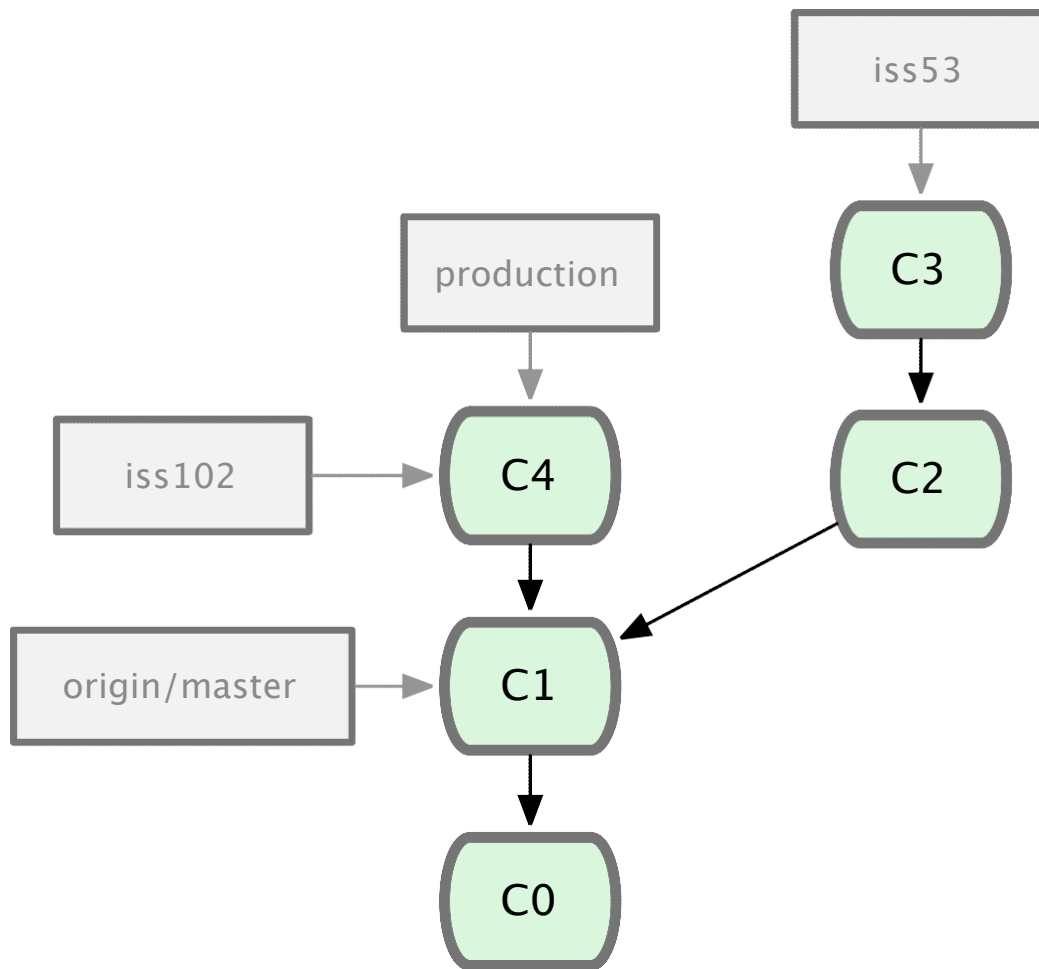
git commit



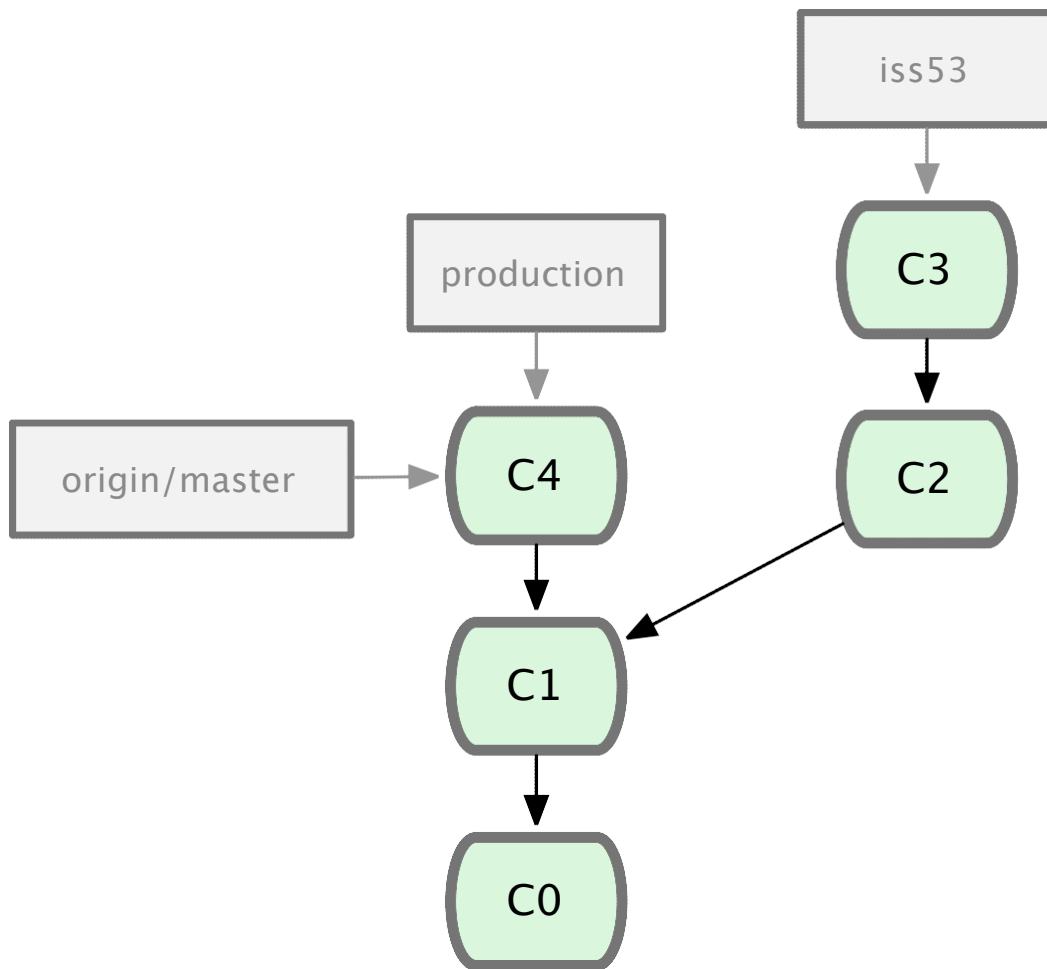
git checkout production  
git checkout -b iss102



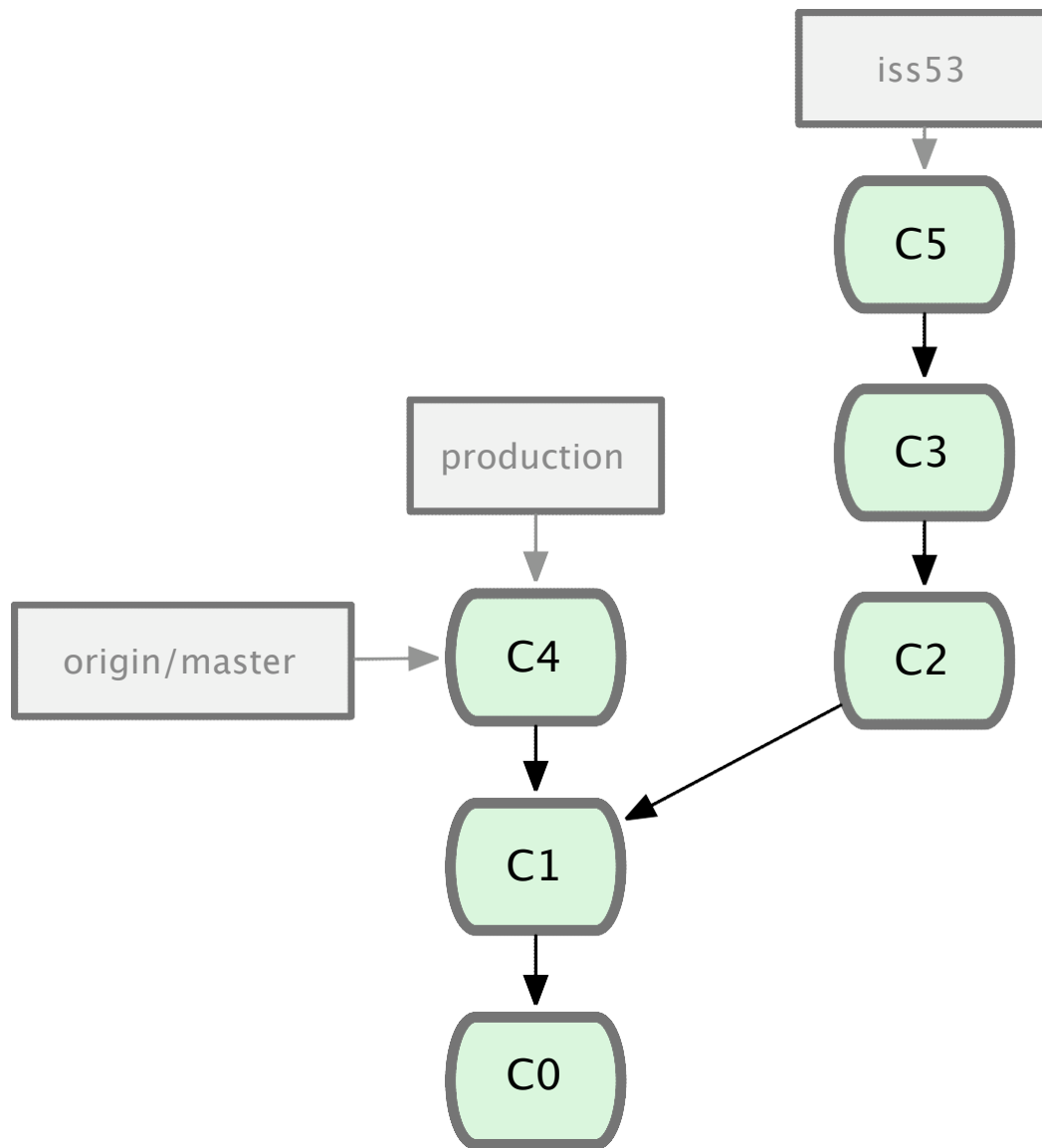
git commit



git checkout production  
git merge iss102

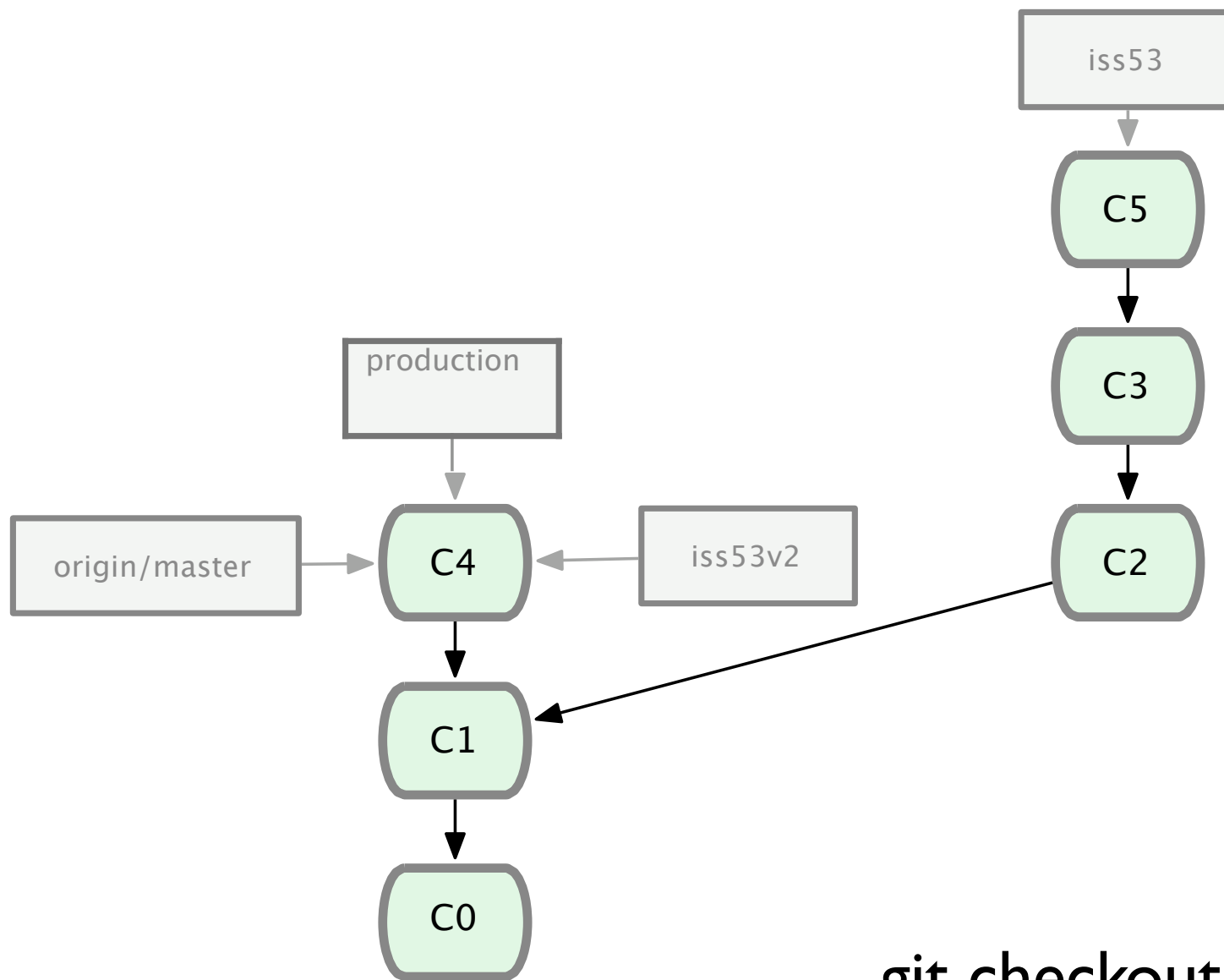


git push

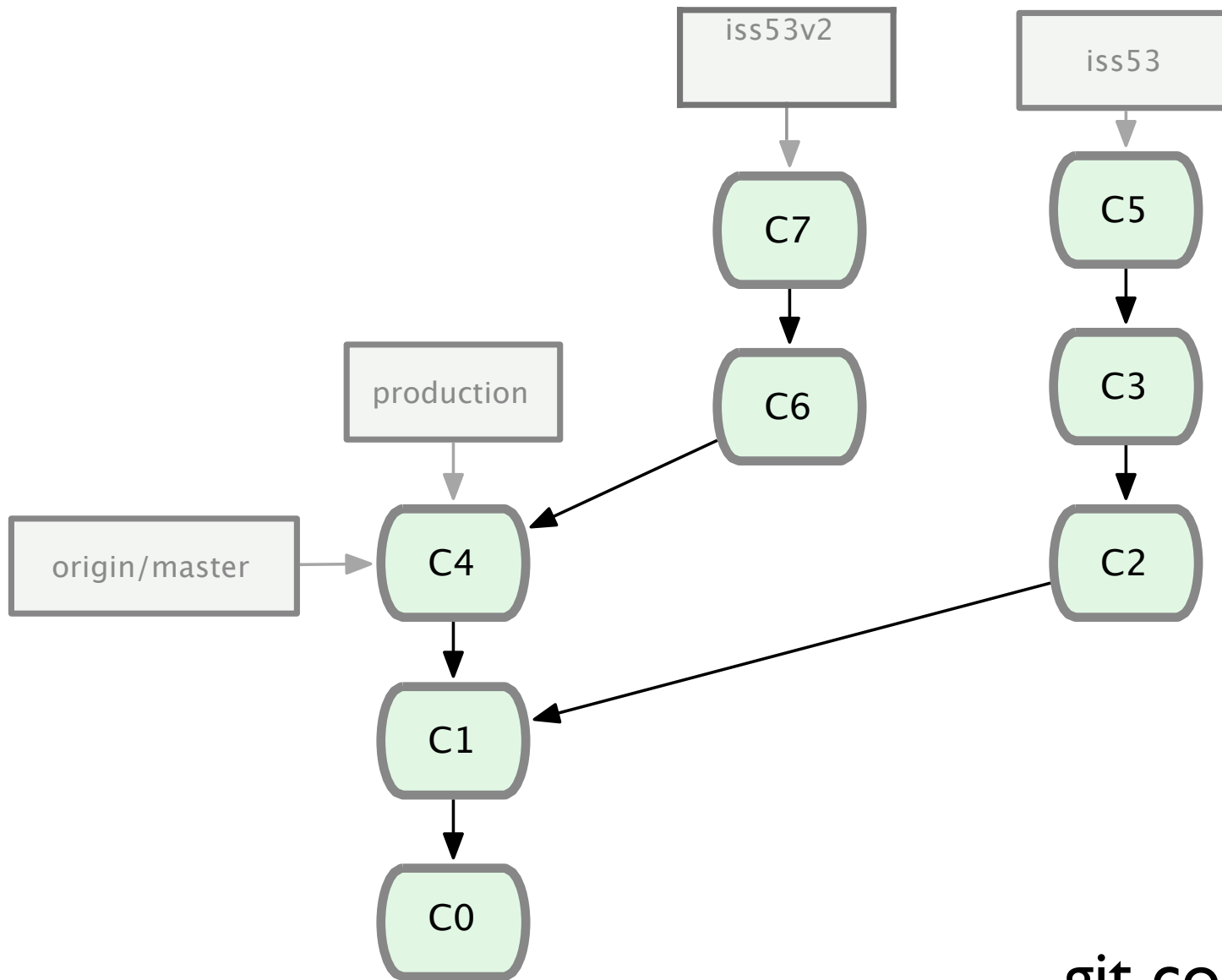


git checkout iss53  
git commit





`git checkout production`  
`git checkout -b iss53v2`



git commit  
git commit

**try out an idea**

**isolate work units**

**long running topics**

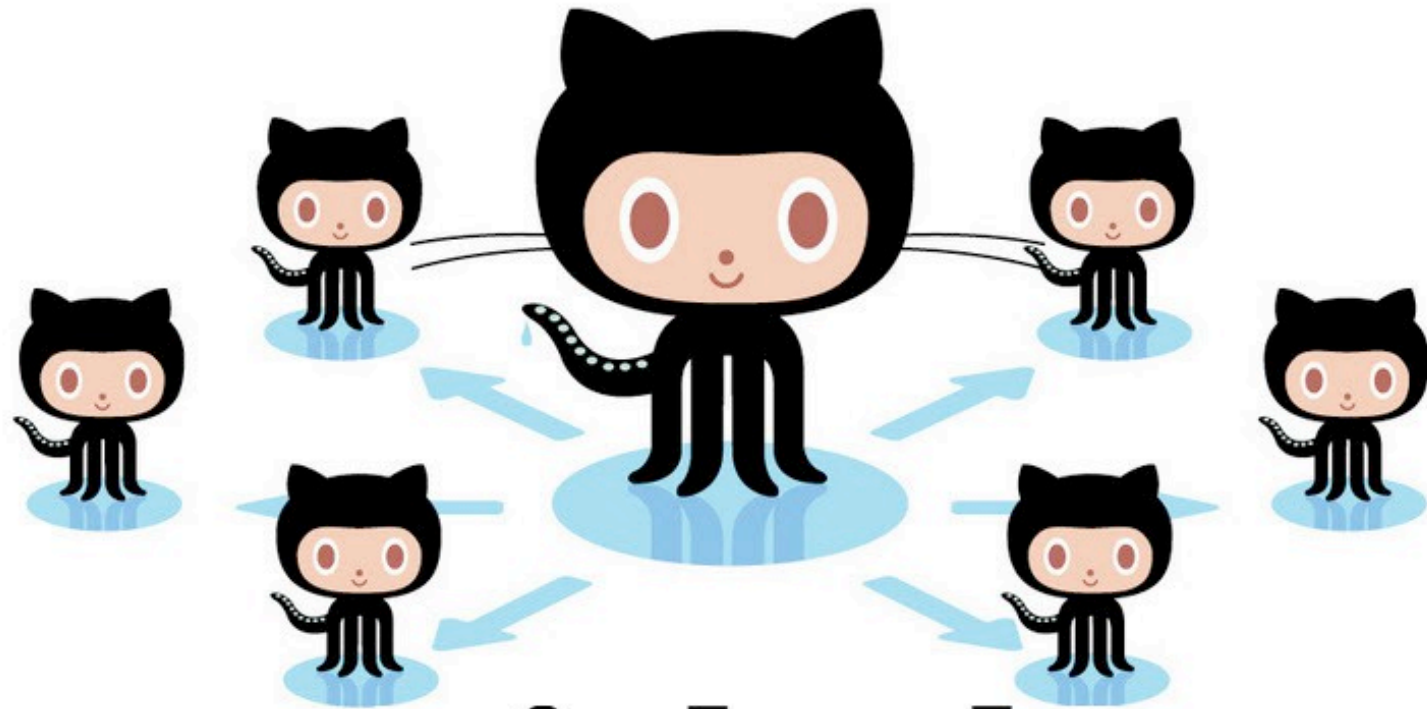
**pain free  
context switching**

# GitHub



**GitHub** is a web-based Git repository hosting service.





**github**  
SOCIAL CODING

## **GitHub Features:**

- Fork
- Pull Request
- Merge
- Issues
- Open Source / Closed Source Collaboration

fork

*A fork* is a copy of a repository.  
Forking a repository allows you  
to freely experiment with  
changes without affecting the  
original project.



This repository Search

Pull requests Issues Gist



UW-Hydro / VIC

Unwatch 20

Unstar 22

Fork 85

The Variable Infiltration Capacity (VIC) Macroscale Hydrologic Model  
<http://www.hydro.washington.edu/Lettenmaier/Models/VIC/index.shtml> — Edit

841 commits

4 branches

90 releases

5 contributors



branch: master

VIC / +



Update readme.md



jhamman authored on May 29

latest commit a0343954b5

samples	remove LOG_MATRIC option from sample global parameter file	8 months ago
src	Update ChageLog for new 4.2 version numbering.	6 months ago
tools	Merge branch 'hotfix/4.1.2.1' into develop	a year ago
.gitignore	Revert "Added notes subdirectory to gitignore"	a year ago
.travis.yml	fix to travis make	2 years ago
LICENSE.txt	Create LICENSE.txt	9 months ago
readme.md	Update readme.md	a month ago

Code

Issues 33

Pull requests 4

Wiki

Pulse

Graphs

Settings

SSH clone URL

git@github.com:UW-Hy



You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).



This repository Search

Pull requests Issues Gist



jhamman / VIC  
forked from UW-Hydro/VIC

Unwatch 1

Star 1

Fork

85

## The Variable Infiltration Capacity (VIC) Macroscale Hydrologic Model — Edit

1,127 commits

18 branches

91 releases

4 contributors



branch: develop

VIC / +



This branch is 3 commits behind UW-Hydro:develop

Pull Request Compare

Merge pull request #219 from UW-Hydro/feature/mpi



jhamman authored on May 19

latest commit 60f860e3fc

drivers	State file saving commented out as this is not yet working in MPI	2 months ago
samples	add vic_log.h macros and optional logfile initialization	7 months ago
tests	add tests/README.md	6 months ago
tools	add code_format readme and helper script to run uncrustify for all vi...	6 months ago
vic_run	Merge branch 'develop' into feature/mpi	2 months ago
.gitignore	Add .depend to .gitignore	2 months ago
.travis.yml	Merge branch 'develop' into feature/mpi	6 months ago

Code

Issues 0

Pull requests 2

Wiki

Pulse

Graphs

Settings

SSH clone URL

git@github.com:jhamm



You can clone with HTTPS, SSH, or Subversion.


**pull request**

Pull requests let you tell others about changes you've pushed to a repository on GitHub.



## Feature/docs #207

Edit

 Open jhamman wants to merge 2 commits into UW-Hydro:develop from jhamman:feature/docs Conversation 0 Commits 2 Files changed 10+265 -1 

jhamman commented on Feb 23

Owner



This pull request includes a proof on concept for how we could include the VIC website/documentation in the repository. The documentation is in [Markdown](#) and uses [mkdocs](#) to build the site. A sample of the site can be viewed on read-the-docs at: <http://vic.readthedocs.org/en/latest/>. No changes were made to the VIC source code in this PR.

related: [#155](#)

Joe Hamman added some commits on Feb 23



add mkdocs structure to repo

cae4a2a



add docs badge to readme

✓ 9ba4f23

Add more commits by pushing to the **feature/docs** branch on **jhamman/VIC**.

✓ All checks have passed — The Travis CI build passed

[Details](#)**We can't automatically merge this pull request.**[Use the command line](#) to resolve conflicts before continuing.


Merge pull request

1 participant

 Lock pull request

Write

Preview

 Markdown supported  Edit in fullscreen

Leave a comment

merge

**issues**

Issues

Pull requests

Labels

Milestones

Filters ▾

🔍 is:issue is:open

New Issue

&lt;&gt;

☐ ⓘ 33 Open ✓ 86 Closed

Author ▾

Labels ▾

Milestones ▾

Assignee ▾

Sort ▾

- ☐ ⓘ
- Set logging level**
enhancement
- #218 opened on May 19 by jhamman
 0
- ☐ ⓘ
- How should we handle input netCDF variable names?**
driver enhancement
- #211 opened on May 5 by jhamman
 1
- ☐ ⓘ
- Lack of helpful error message when crashing due to global file OUTVAR option error**
bug documentation enhancement
- #209 opened on Apr 17 by orianac
 5.0
2
- ☐ ⓘ
- Reworking the VIC users group**
question
- #208 opened on Apr 10 by jhamman
 5.0
2
- ☐ ⓘ
- Add simulation identifier to output files.**
driver feature
- #205 opened on Jan 22 by jhamman
 5.0
3
- ☐ ⓘ
- Cleanup input and output file headers**
cleanup enhancement feature
- #204 opened on Jan 22 by jhamman
 5.0
6
- ☐ ⓘ
- Where should we put submodules and submodels**
cleanup question
- #199 opened on Jan 19 by jhamman
 5.0
5
- ☐ ⓘ
- Image Mode Driver: History Module**
driver enhancement feature
- #198 opened on Jan 17 by jhamman
 5.0
0
- ☐ ⓘ
- Image Mode Driver: Time Handling**
driver enhancement
- #197 opened on Jan 17 by jhamman
 5.0
0

ⓘ

🔗

📖

📊

📈

🔧

**collaboration**

private and public  
repositories

**groups and teams**

**code review**



# github workflow

- Open an issue (e.g. feature proposal)
- Fork a repo
- Clone my fork to my local machine
- Create a branch on my local repository
- Make changes (test, etc.)
- Commit changes
- Push changes to my fork
- Issue a pull request from my fork to the original repo

# questions?

**Lots of good resources on the web:**

- Git: <https://git-scm.com/>
- Github: <https://github.com/>
- Others:
  - <https://www.atlassian.com/git/>
  - <http://stackoverflow.com/questions/tagged/git>