

# Advanced Git

Luc Sarzyniec

Xilopix, February 2015

# About

This slides are using resources from the [Pro Git](#) book [isbn:1430218339] which is licensed under the Creative Commons v3.0 (by-nc-sa) license.

The sources of the book can be found at <https://github.com/progit/progit2> .

The sources of the slides can be found at <https://github.com/olbat/misc/tree/master/slides/advanced-git> .

# Summary

1. Overview
2. Basic usage
3. Work with branches
4. Rewrite history
5. Code introspection
6. Useful commands
7. Internals

# Overview

- Originally developed to work on the GNU/Linux kernel
- First release in 2005 (1 year after subversion 1.0)
- Free software (GPLv2)
- Main goals
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of branches)
  - Fully distributed
  - Ability to handle large projects like the Linux kernel efficiently (speed and data size)

# Finding documentation

## [Read the manual](#)

- Well written
- A lot of examples

## [Pro Git book](#)

- Very complete
- Easy to read/understand
- Available in different formats

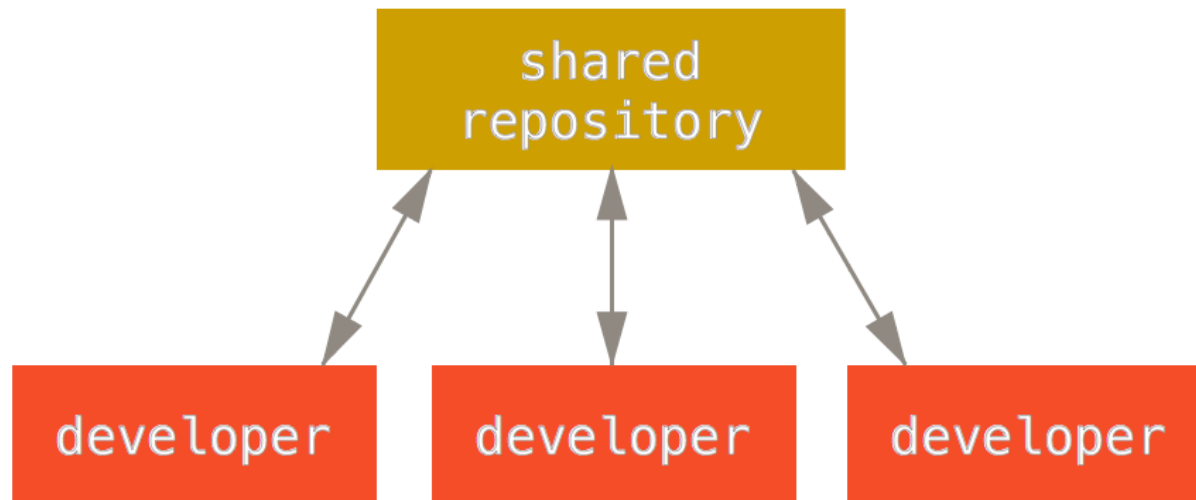
## [Other books](#)

# A distributed revision control system

[Pro Git, [chapter 5](#)]

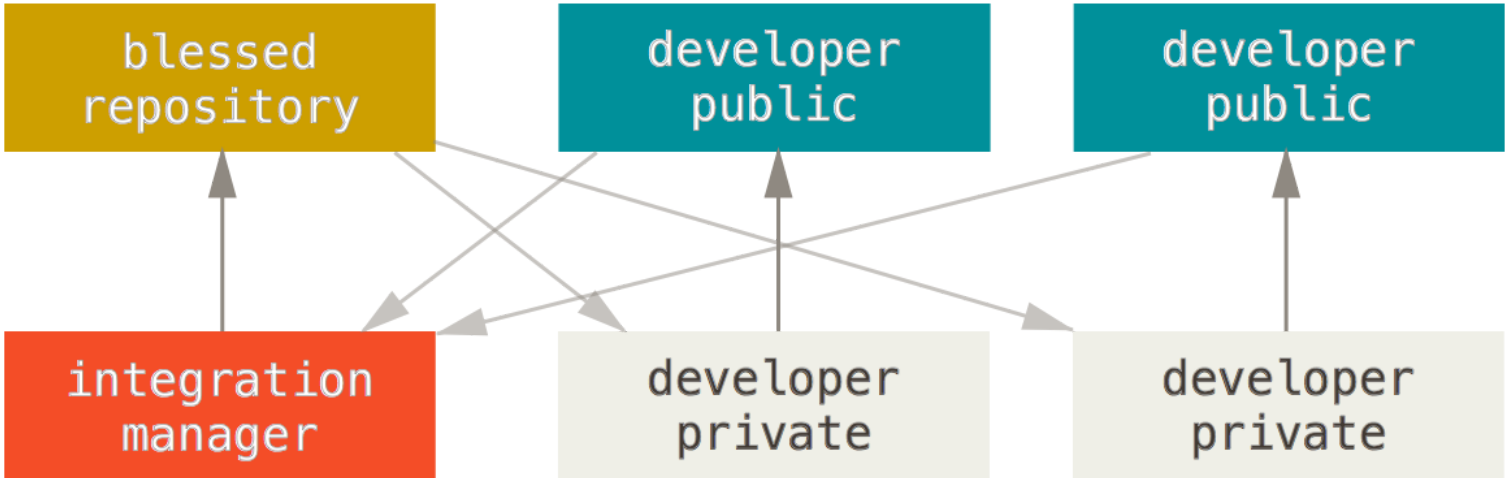
# A distributed revision control system

Centralized workflow



# A distributed revision control system

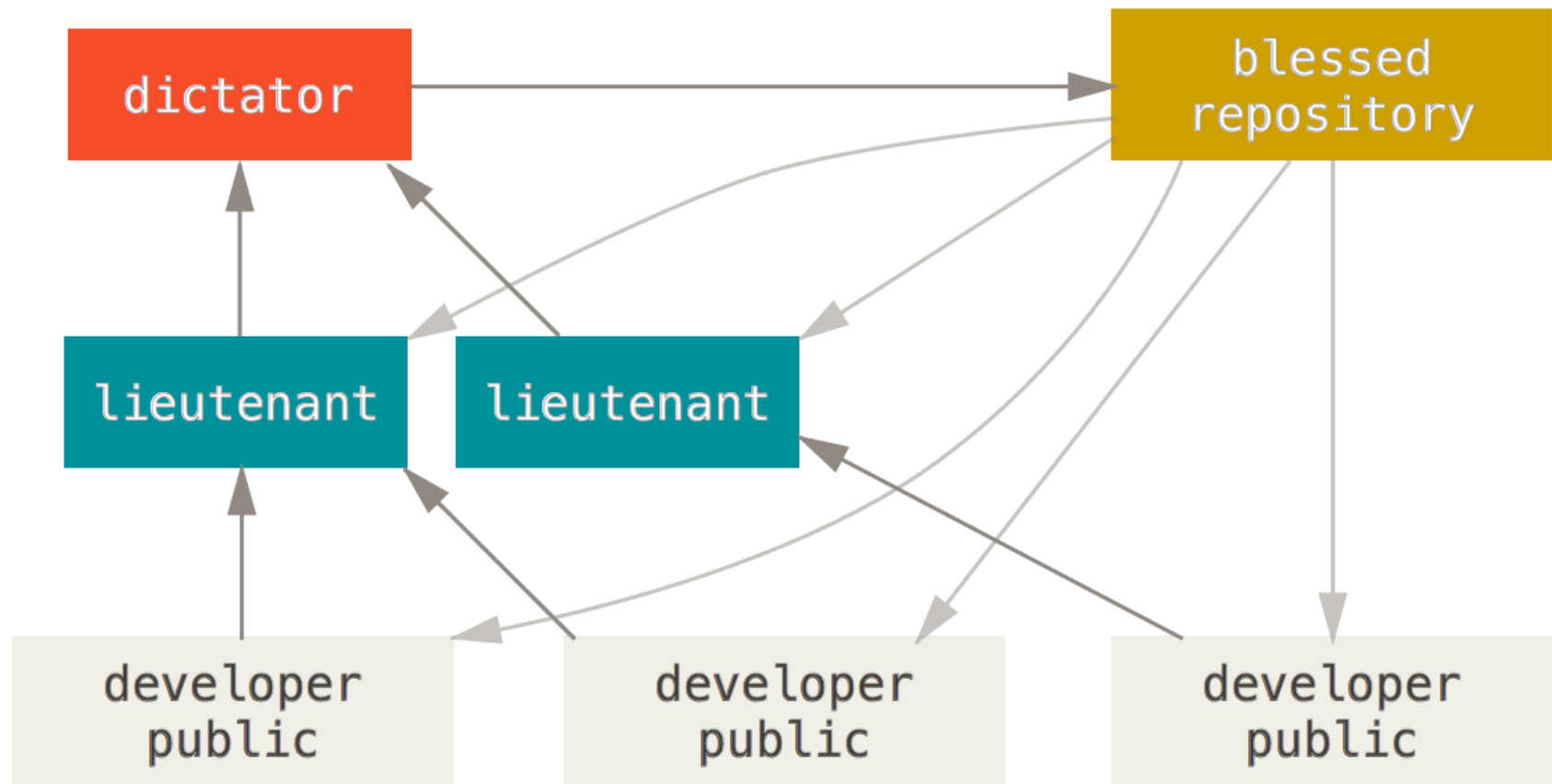
# Integration manager workflow





# A distributed revision control system

## Dictator and Lieutenants workflow



# A distributed revision control system

## Use a temporary backup repository to store WIP

- Why not rsync ?
  - Do not copy untracked/generated files, track useful files
  - Ability to push a single branch
- Create a new repository on a remote server

```
$ ssh remote "git init --bare repo.git"
```

- Register this remote in your local repository

```
$ git remote add wip ssh://remote/absolute/path/repo.git
```

- Push current work in the WIP repository

```
$ git push --all wip
```

# A distributed revision control system

Use a temporary backup repository to store WIP

- Do some work

```
$ git commit ...
```

- Push work on the WIP remote

```
$ git push wip master
```

- Continue to work

```
$ git commit --amend  
$ git rebase -i ...
```

- Once work is done push to the stable repository

```
$ git push origin master
```

# Git basics

[Pro Git, [chapter 2](#)]

# Start to work

## 1. Create an empty (sandbox) repository

```
$ git init --bare /tmp/sandbox.git
```

## 2. Clone the repository

```
$ git clone file:///tmp/sandbox.git
```

## 3. Start to work in the master branch

```
$ cd /tmp/sandbox  
$ git checkout -b master
```

# State of the repository

[Pro Git, [chapter 2.2](#)]

# State of the repository

- State of the repository in long format

```
$ git status
Changes to be committed:
  new file:   staged_file
  deleted:    file

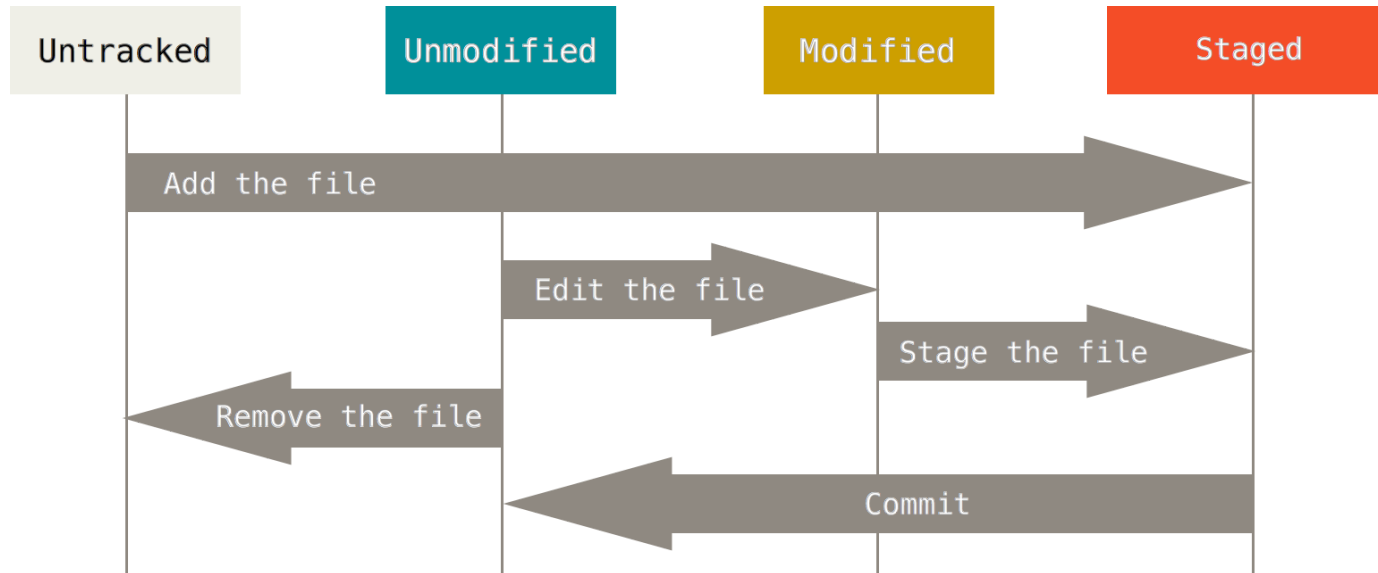
Changes not staged for commit:
  modified:   modified_file

Untracked files:
  new_file
```

- State of the repository in short format

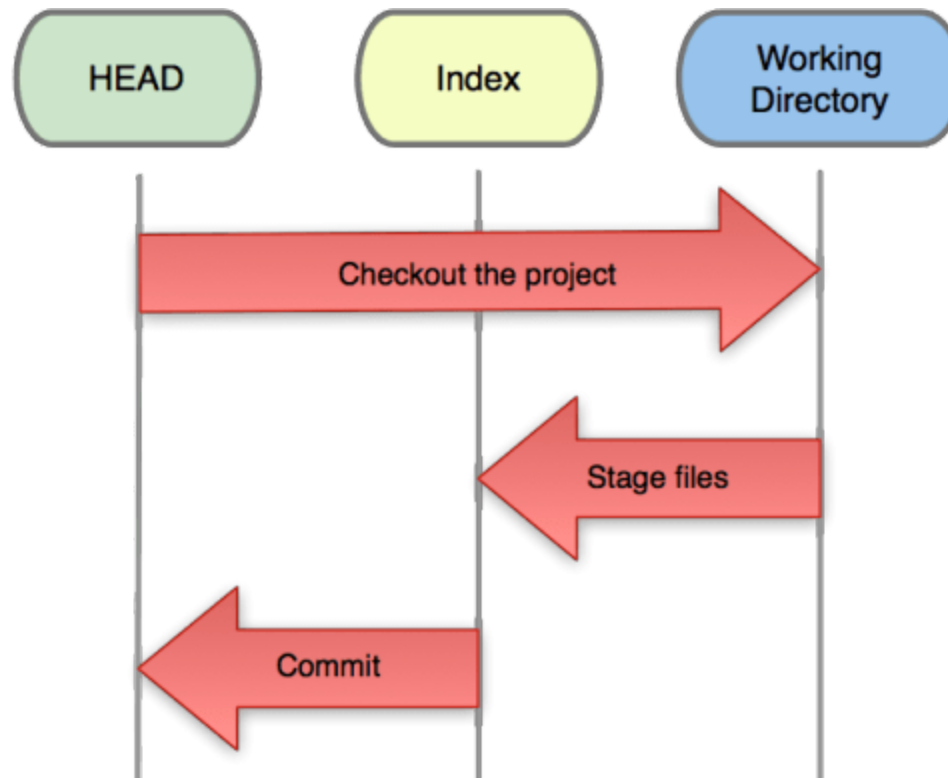
```
$ git status -s # --short
D file
M modified_file
A staged_file
?? new_file
```

# State of files





# HEAD, index and working dir.



[Git blog, [reset](#)]

# Planing modifications

[Pro Git, [chapter 2.2](#)]

# Staging modifications

- Stage only some parts of a file (interactive)

```
$ git add -p FILE # --patch
```

- Stage all indexed files that has changed

```
$ git add -u # --update
```

- Stage both modified and untracked files

```
$ git add -A # --all
```

- Unstage staged files

```
$ git reset HEAD FILE1 FILE2 .. FILEn
```

# Discard local modifications

- Discard changes in files

```
$ git checkout -- FILE1 FILE2 .. FILEn
```

- Undo commit and keep modified/new files in index

```
$ git reset --soft HEAD^
```

- Undo commit and remove modified/new files from index

```
$ git reset HEAD^
```

- Undo commit and undo changes to indexed files

```
$ git reset --hard HEAD^
```

[Pro Git, [chapter 2.4](#)]

# Save repository state w/o commit

- Stash some modifications (saves the current diff)

```
$ git status -s
A  file
M  modified_file
D  removed_file
?? untracked_file
```

```
$ git stash save
```

```
$ git status -s
?? untracked_file
```

- List current stashed changes

```
$ git stash list
HEAD is now at ce499bc commit
stash@{0}: WIP on test: ce499bc commit
stash@{1}: WIP on master: 0029594 commit2
```

[Pro Git, [chapter 7.3](#)]

# Save repository state w/o commit

- Display a specific stash

```
$ git stash show stash@{0} # -p to show in diff format
file          | 1 +
modified_file | 2 +-
removed_file  | 0
3 files changed, 2 insertions(+), 1 deletion(-)
```

- Apply stashed changes (apply diff)

```
$ git stash apply # stash@{0}
$ git status -s
A file
M modified_file
D removed_file
?? untracked_file
```

- Create a new branch and apply stashed changes in the top of it

```
git stash branch # stash@{0}
```

**Save modifications**

# Commit changes

- Commit and specify message on the CLI

```
$ git commit -m 'message'
```

- Skip the staging area

```
$ git commit -m "message" -a # ~ git add -a && commit
```

- Select what to commit (interactive)

```
$ git commit -m "message" -p # ~ git add -p && commit
```

- Rewrite (amend) the last commit (staged files will be added in the commit)

```
$ git commit --amend # --no-edit
```



# View modifications

# View modifications

- View unstaged modifications

```
$ git diff
```

- View staged modifications

```
$ git diff --cached
```

- View modifications between two branches

```
$ git diff master..develop  
$ git diff origin/develop..develop
```

- View changes of a specific file

```
$ git diff -- filename  
$ git diff master..develop -- filename
```

# View modifications

- Summary of changes

```
$ git diff --stat
```

- Show ~bitwise diff

```
$ git diff --color-words
```

- View changes of a specific commit

```
$ git show HEAD~
```

- Show the content of a file in a specified version

```
$ git show HEAD~:filename  
$ git show fa616be:filename
```

# Explore the history

[Pro Git, [chapter 2.3](#)]

# Exploring the history

- Show the history of another branch in short version

```
$ git log --oneline branchname
```

- Show the history with branch names

```
$ git log --decorate # git config --global log.decorate true
```

- Show graph version of the history

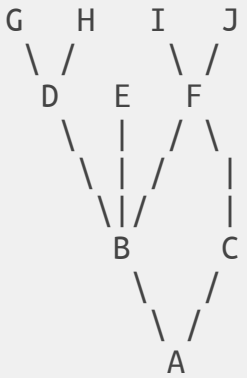
```
$ git log --graph # --all to display every branches
```

- Summary of history grouped by author

```
$ git shortlog
```

# Specifying revisions

- The previous commit: HEAD<sup>^</sup>, HEAD~, HEAD<sup>^</sup>1
- The previous commit of the *develop* branch: develop~1 or develop<sup>^</sup>1
- Two commit before *fa616be*: fa616be~2 or fa616be<sup>^^</sup>
- Three commit before this commit: HEAD~3 or HEAD<sup>^^^</sup>

Commit tree	Revisions
	$A = \quad = A^0$ $B = A^1 = A^1 = A \sim 1$ $C = A^2 = A^2$ $D = A^{11} = A^{11^1} = A \sim 2$ $E = B^2 = A^{12}$ $F = B^3 = A^{13}$ $G = A^{111} = A^{11^11^1} = A \sim 3$ $H = D^2 = B^{12} = A^{112} = A \sim 2^2$ $I = F^1 = B^{13^1} = A^{113^1}$ $J = F^2 = B^{13^2} = A^{113^2}$

[[git rev-parse manual](#), section *SPECIFYING REVISIONS*]

# Work in team

[Pro Git, [chapter 2.5](#) and [chapter 5.2](#)]

# Download and upload changes

- Push several new branches on the remote

```
$ git push origin branchname name:othername HEAD:name
```

- Delete a branch from the remote

```
$ git push origin :branchname
```

- Delete local branches that track deleted remote branches

```
$ git fetch origin -p # --prune
```

- Fetch changes from a branch in a specific branch

```
$ git fetch origin master:updated_master
```

[Pro Git, [chapter 3.5](#)]



# Working with remotes

## Local view

```
$ find .git/refs -type f
.git/refs/heads/localbranch
.git/refs/heads/master
.git/refs/remotes/origin/master
.git/refs/remotes/origin/remotebranch
```

## Classic state

```

                                C1  C2  C3
uri:///project.git/refs/heads/master ----*----*----*
(remote,read-write)

                                C1  C2
refs/remotes/origin/master -----*----*
(local,read-only)

                                C1
refs/heads/master -----*
(local,read-write)
```

# Fetch and Pull

Fetch (git fetch origin master)

```

                                C1  C2  C3
uri:///project.git/refs/heads/master ----*----*----*
(remote,read-write)
                                C1  C2  v
refs/remotes/origin/master -----*----*====*
(local,read-only)
                                C1
refs/heads/master -----*
(local,read-write)
```

Pull (git pull origin master Or git fetch origin master:master)

```

                                C1  C2  C3
uri:///project.git/refs/heads/master ----*----*----*
(remote,read-write)
                                C1  C2  v
refs/remotes/origin/master -----*----*====*
(local,read-only)
                                C1          v
refs/heads/master -----*====*====*
(local,read-write)
                                C2  C3
```

# Discard remote modifications

- Revert commits (applies the reverse diffs)

```
$ git revert COMMIT1 COMMIT2 .. COMMITn  
$ git push origin
```

- Override a remote branch with a local one (TO AVOID)

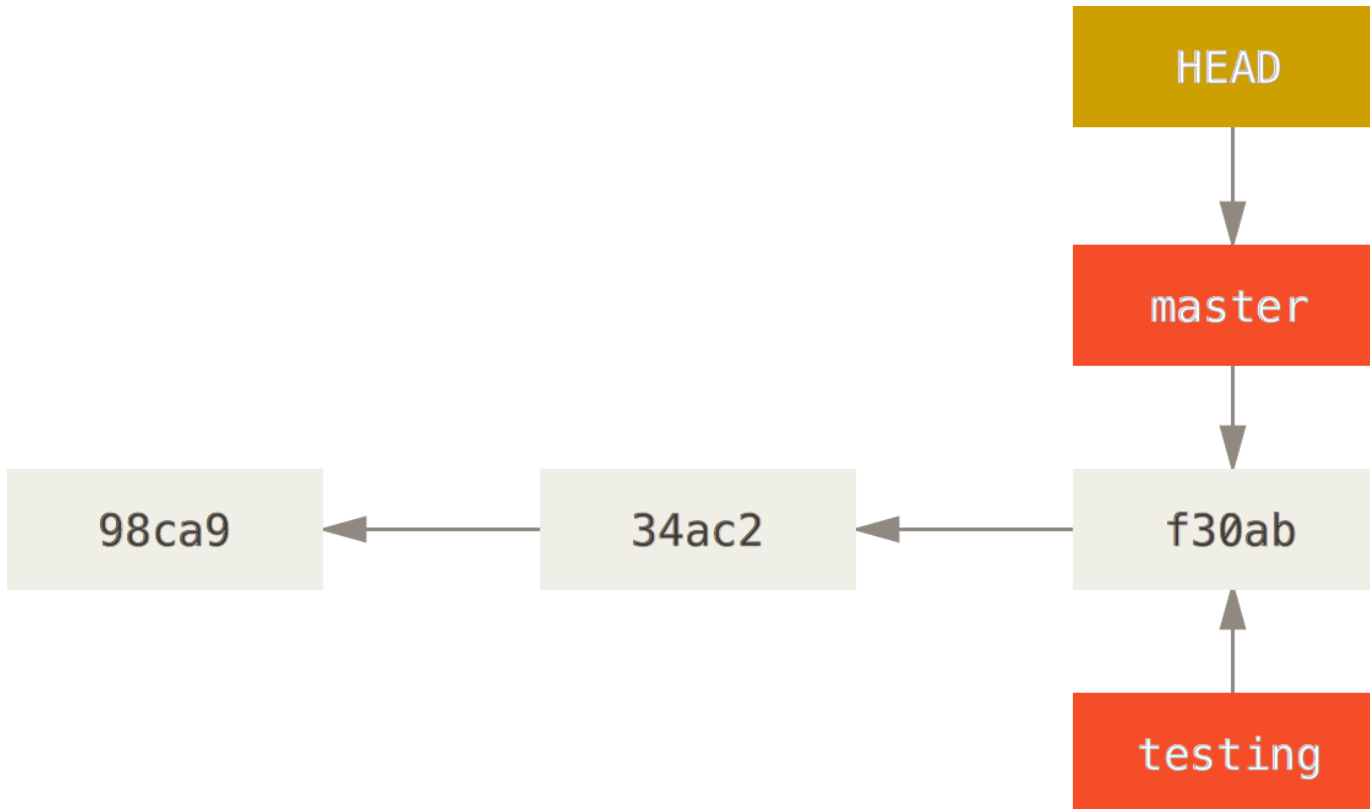
```
$ git rebase -i ... # rewrite history  
$ git push -f origin branch  
# Prepare yourself to be punched by co-workers
```

# Working with branches

[Pro Git, [chapter 3](#)]

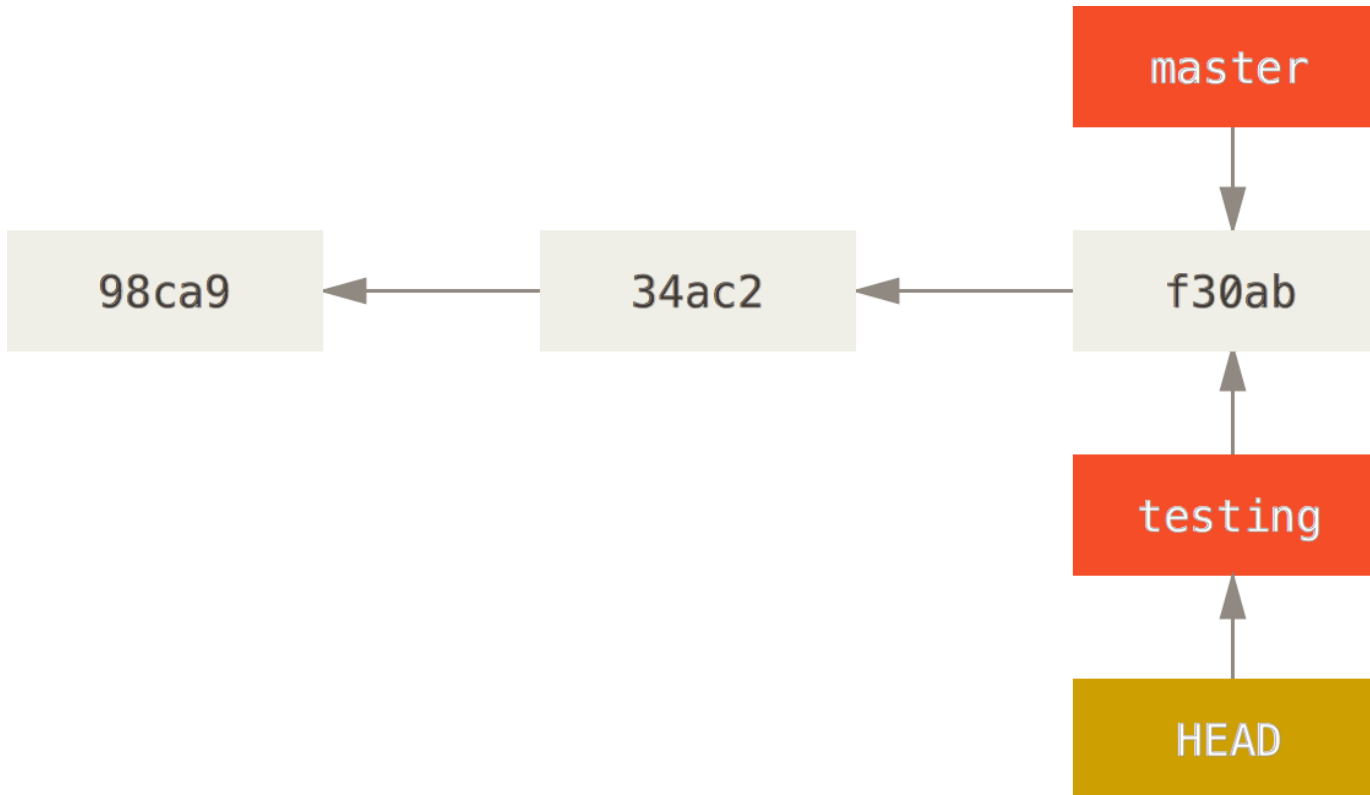
# Working in branches

```
$ git branch testing
```



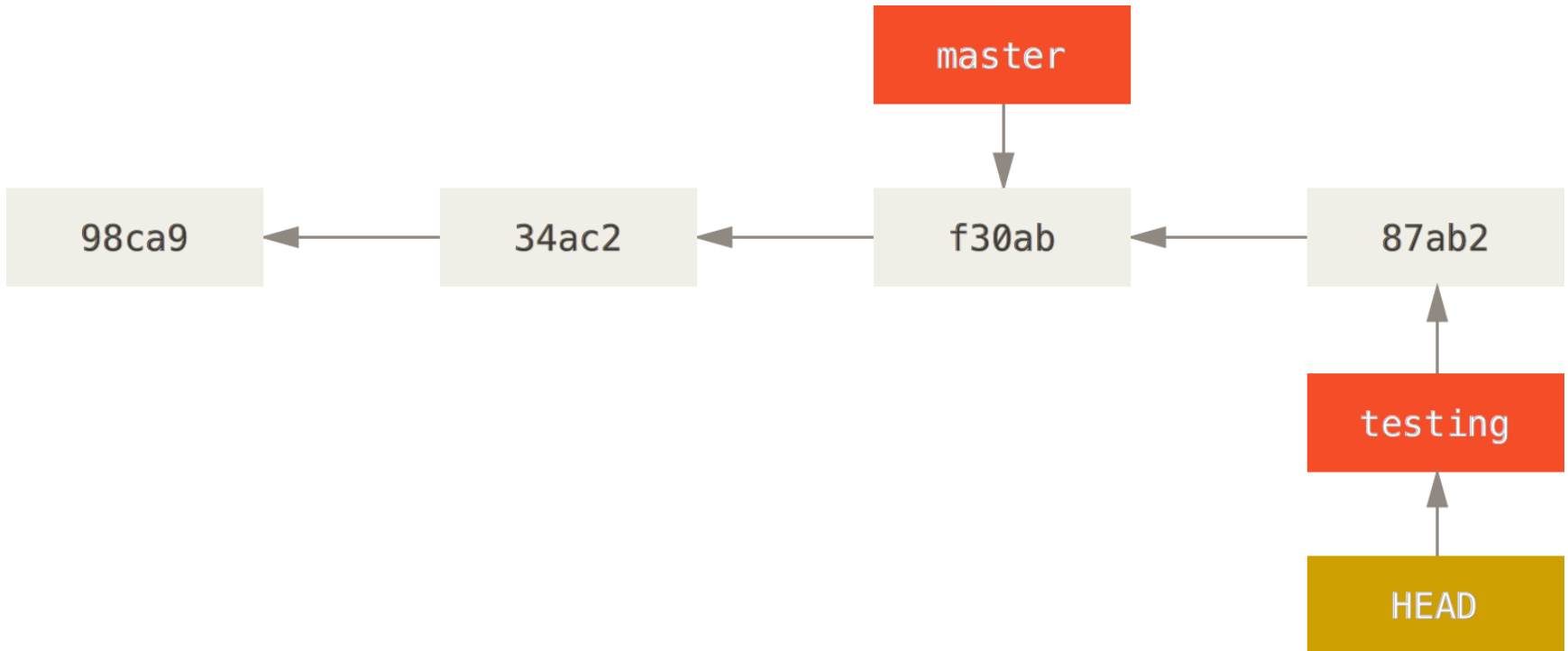
# Working in branches

```
$ git checkout testing
```



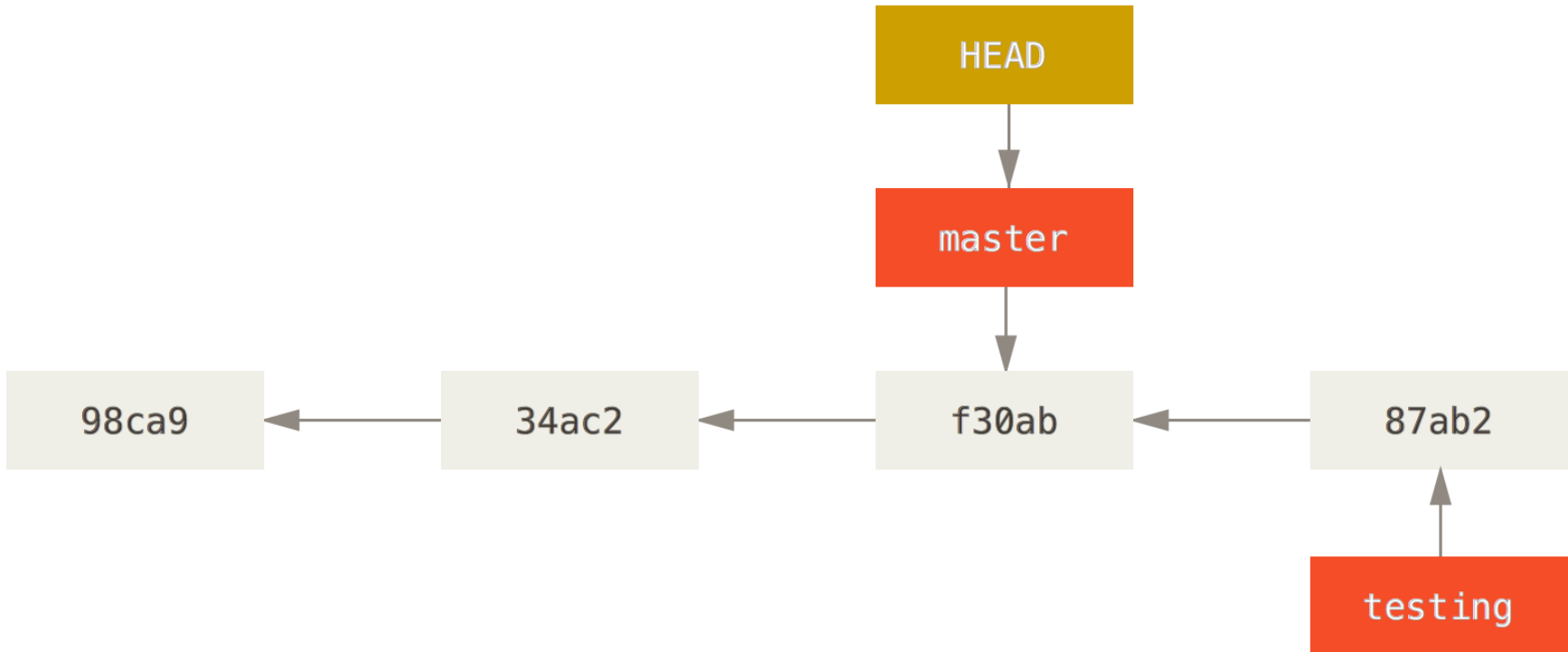
# Working in branches

```
$ git add ... && git commit ... # in testing
```



# Working in branches

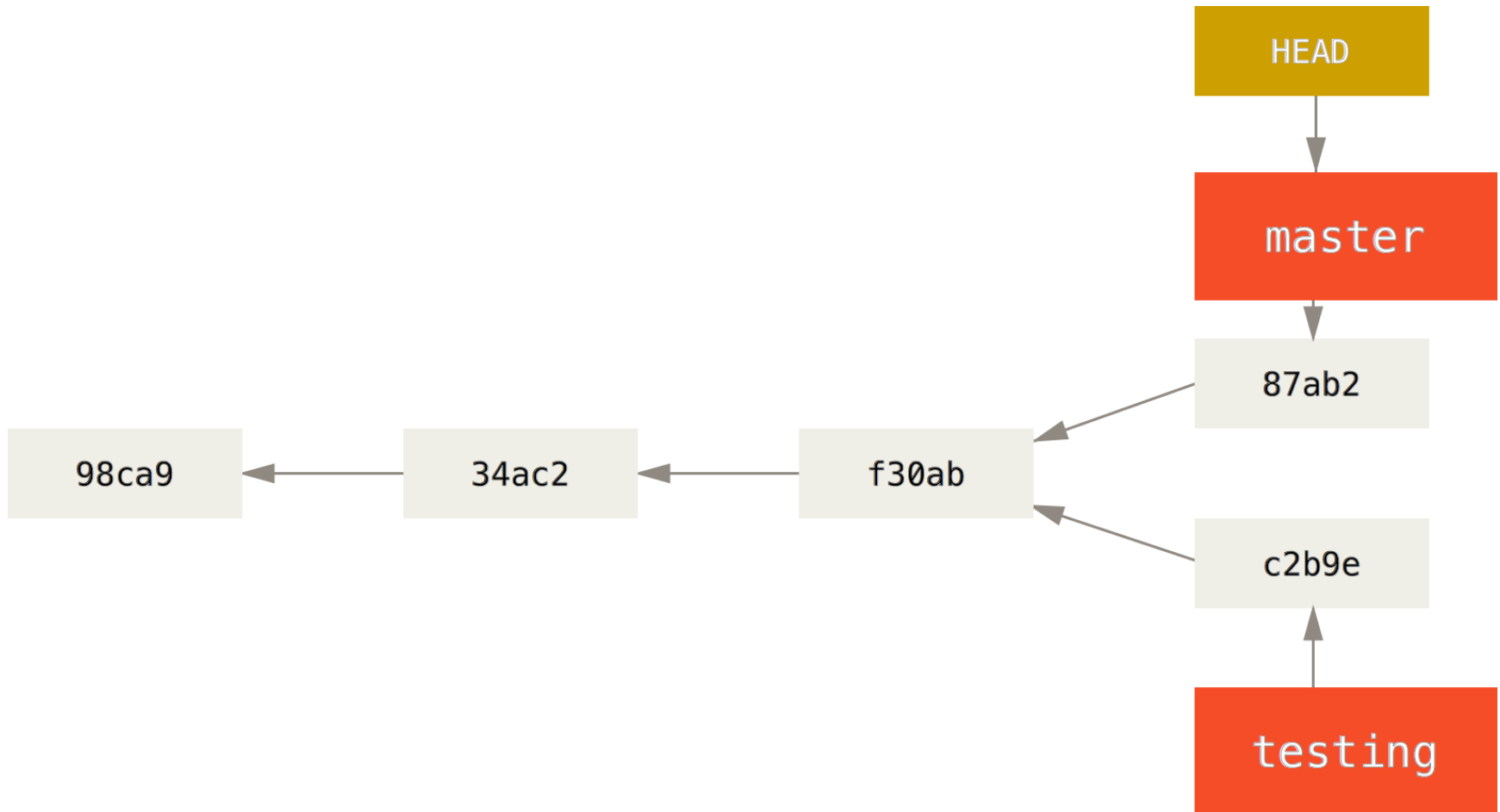
```
$ git checkout master
```





# Working in branches

```
$ git add ... && git commit ... # in master
```



# Working with branches

- Show history of HEAD's values (find deleted/reseted branch)

```
$ git reflog
```

- Create and checkout a new branch based on an existing one

```
$ git checkout -b feature origin/master
```

- Checkout a new empty branch

```
$ git checkout --orphan newbranch  
$ git rm -r --cached .
```

- Clean: remove every local branches that has been merged

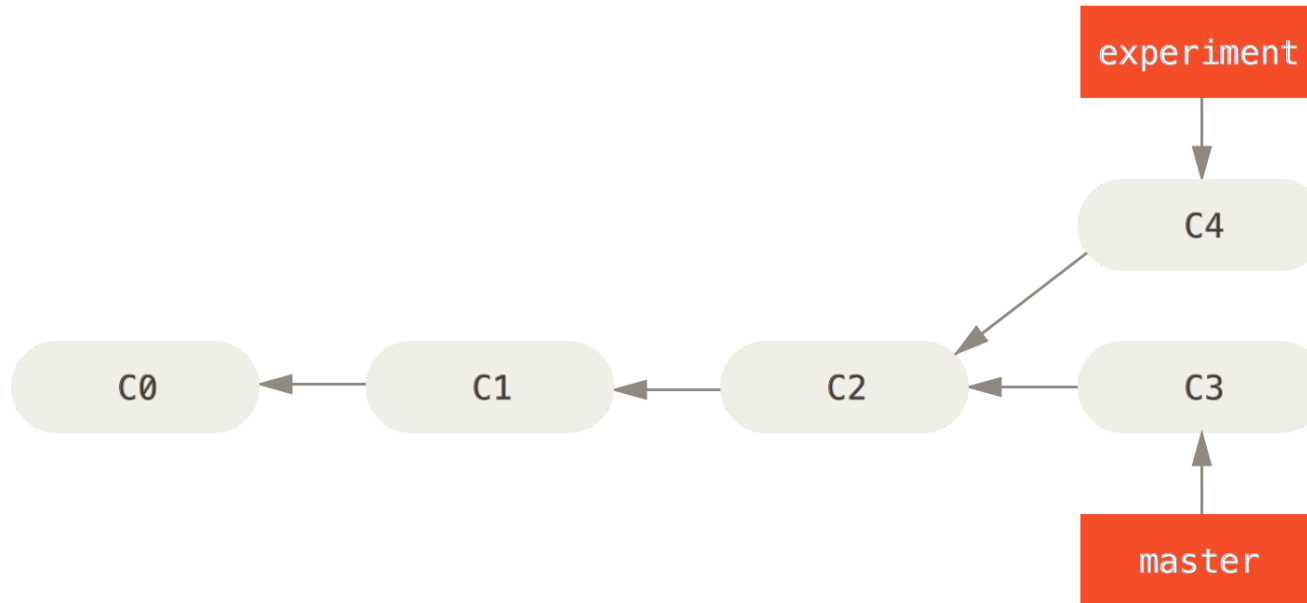
```
git branch --merged master | grep -v '^*' | xargs -n 1 git branch -d
```

# Integrate changes between branches

[Pro Git, [chapter 5.3](#)]

# Integrate changes between branches

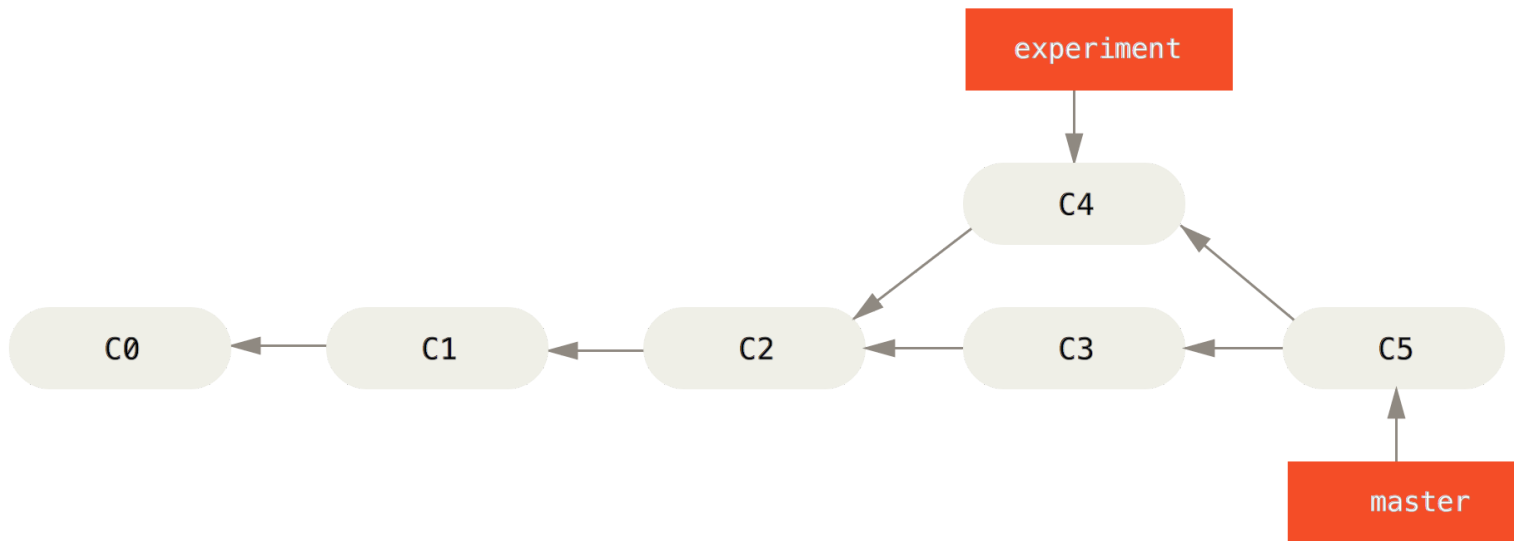
- Simple divergent history



# Integrate changes between branches

- Merging

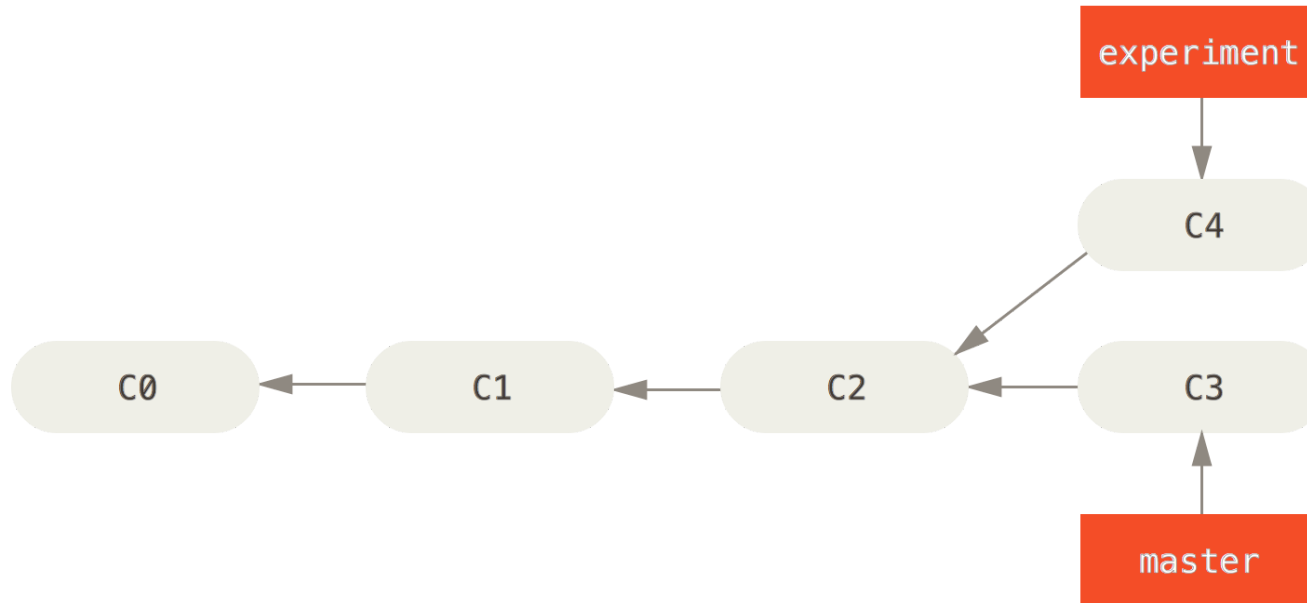
```
$ git checkout master  
$ git merge experiment
```



[Pro Git, [chapter 3.2](#)]

# Integrate changes between branches

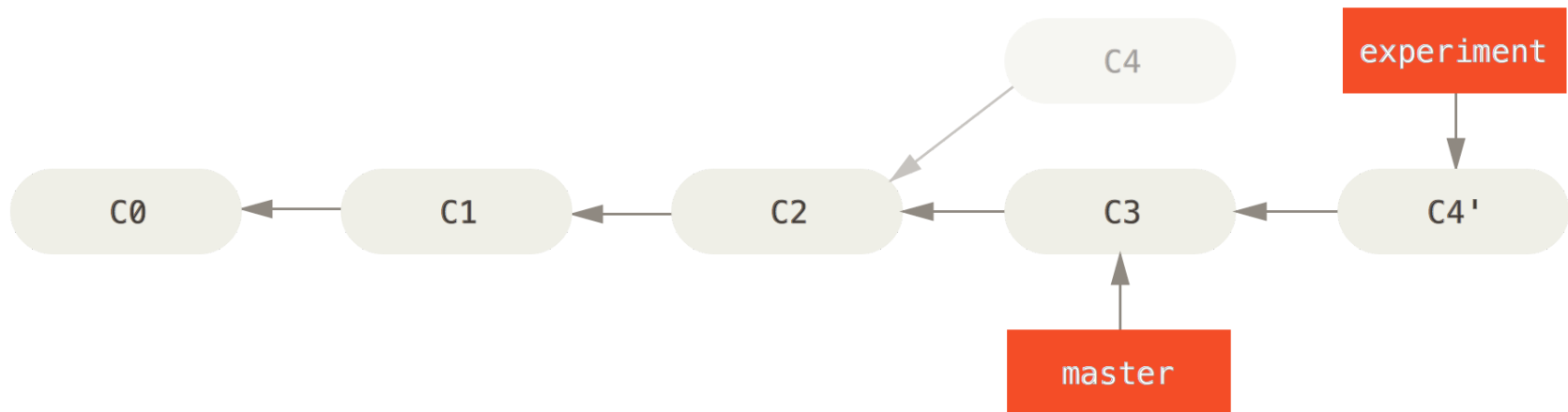
- Simple divergent history



# Integrate changes between branches

- Rebasing

```
$ git checkout experiment  
$ git rebase master
```

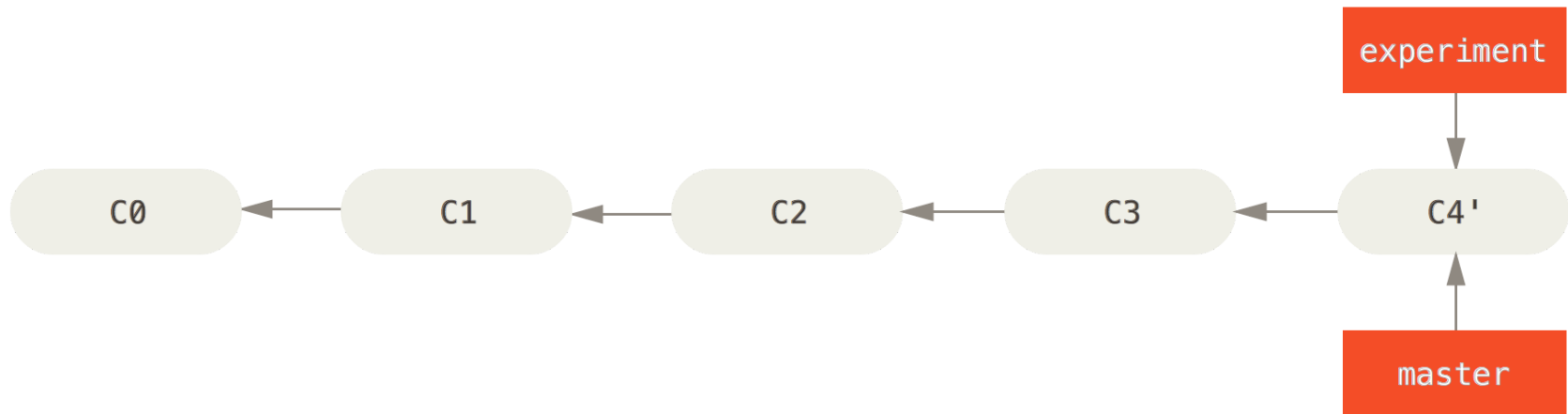


[Pro Git, [chapter 3.6](#)]

# Integrate changes between branches

- Rebasing

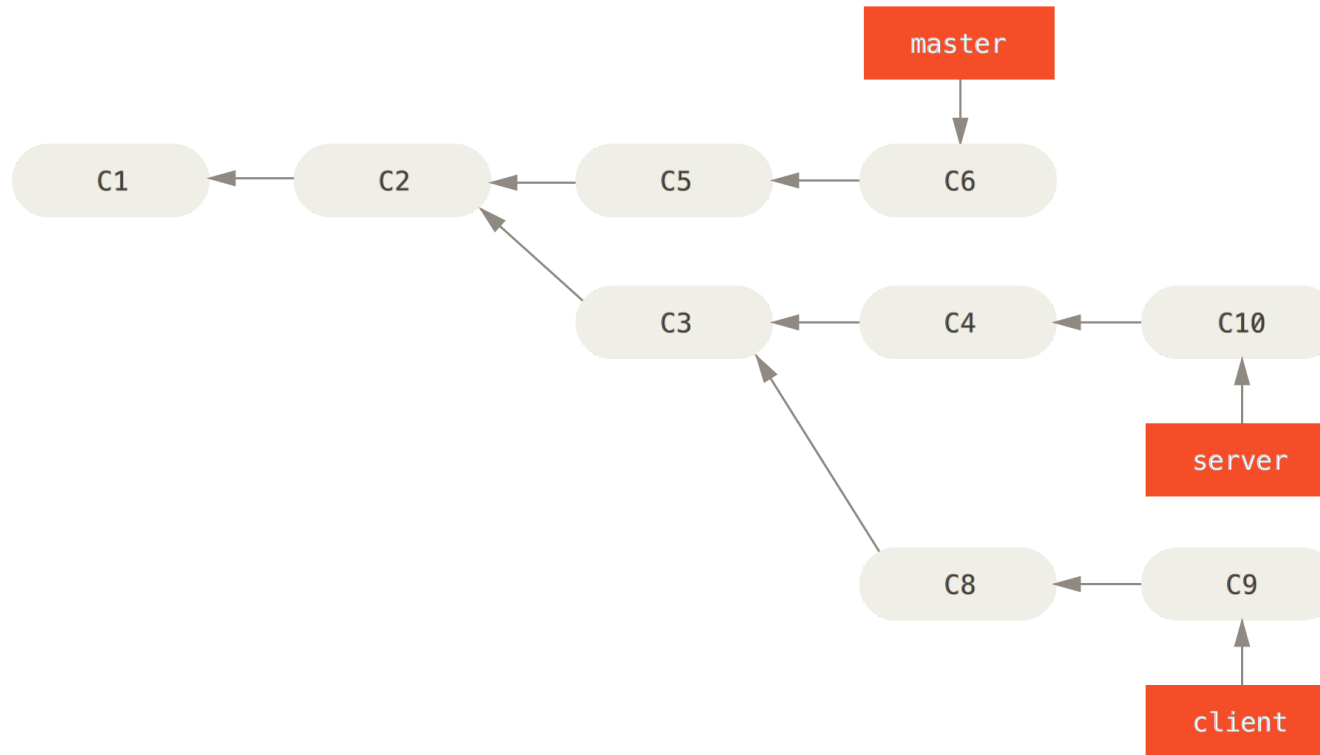
```
$ git checkout master  
$ git merge --ff experiment
```





# Integrate changes between branches

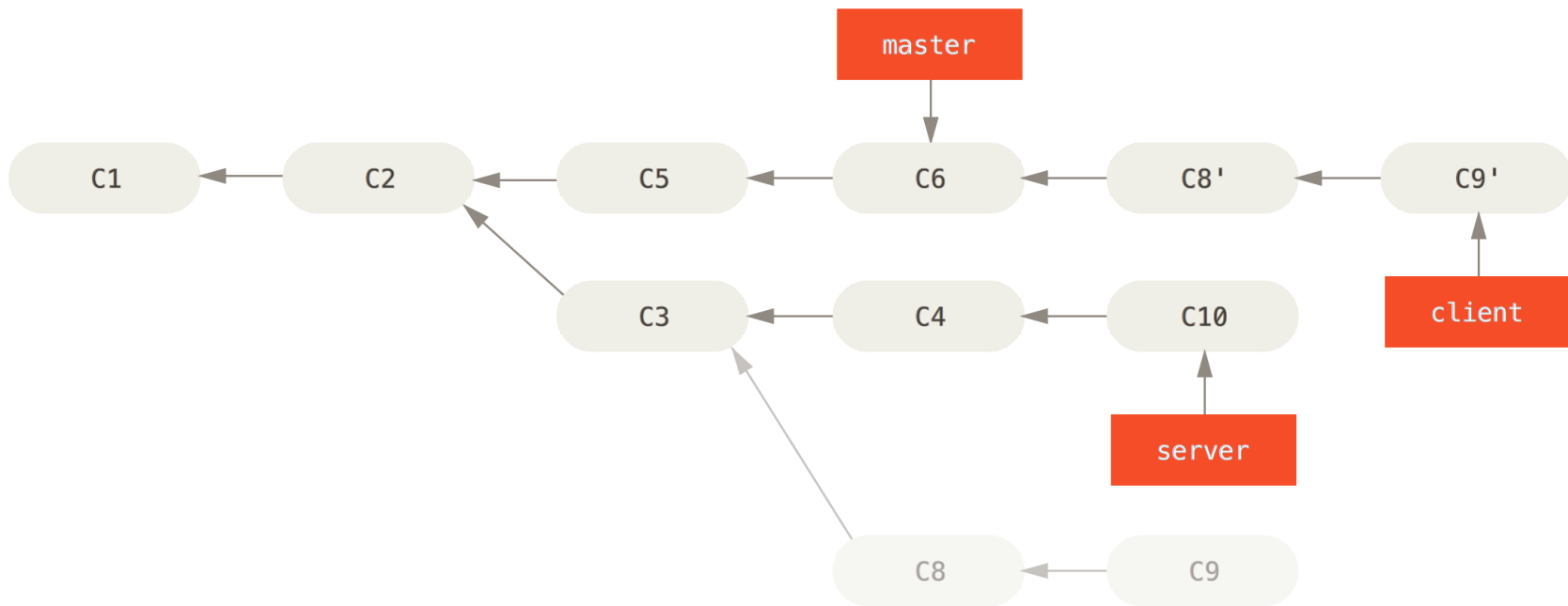
- Complex divergent history



# Integrate changes between branches

- Rebase a branch onto another

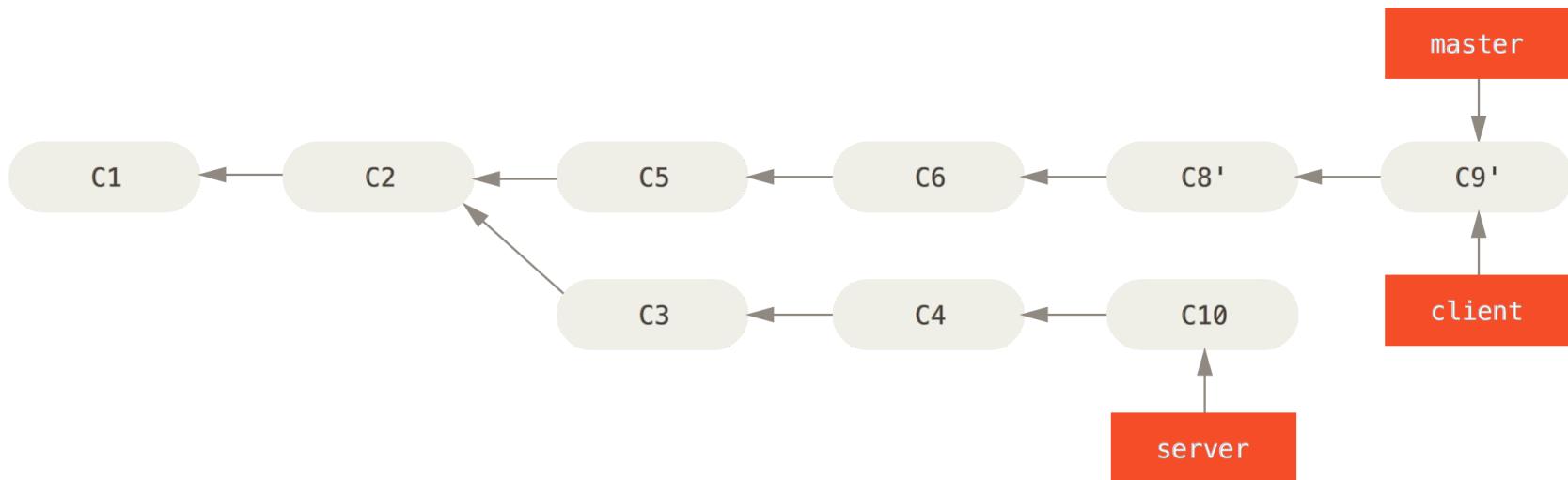
```
$ git rebase --onto master server client
```



# Integrate changes between branches

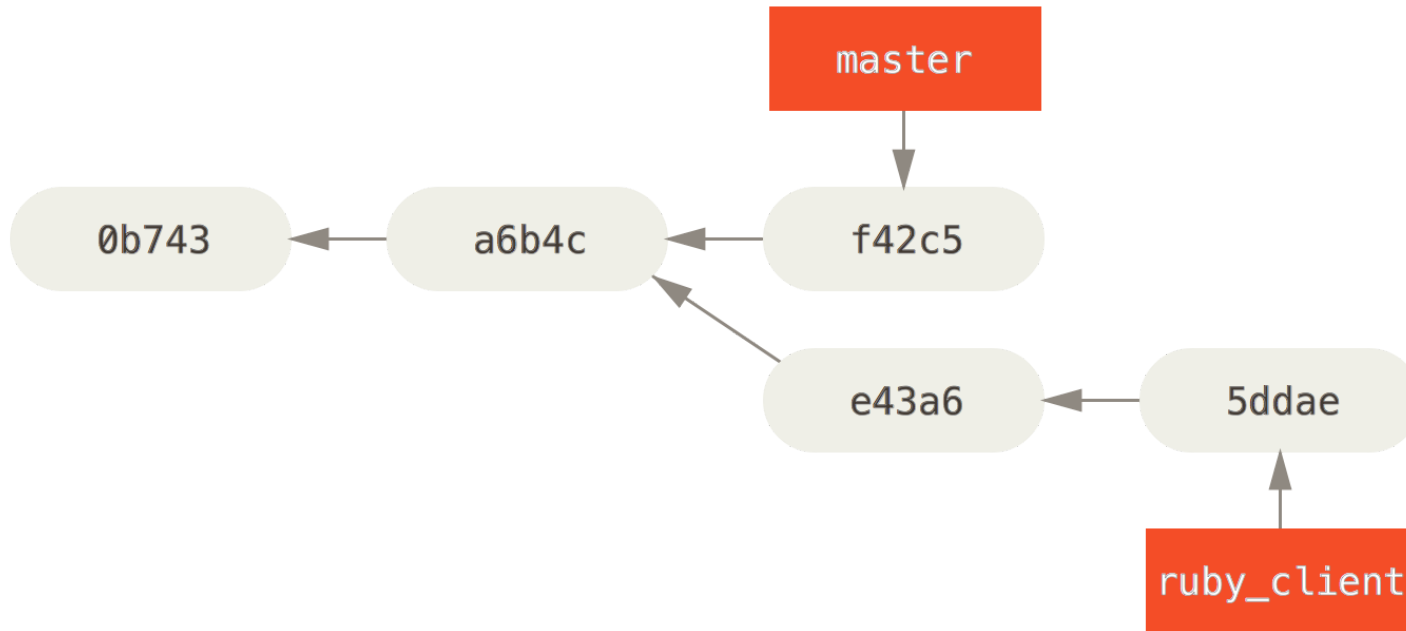
- Rebase a branch onto another

```
$ git checkout master  
$ git merge --ff client
```



# Integrate changes between branches

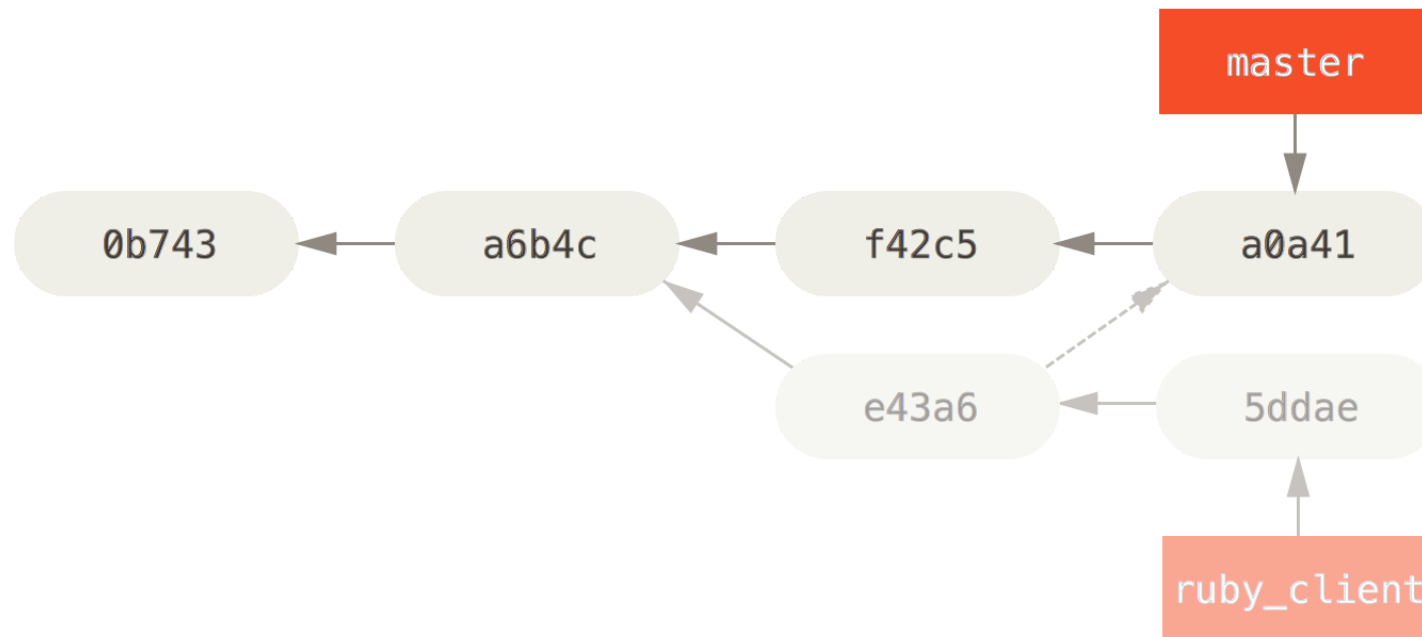
- Another simple divergent history



# Integrate changes between branches

- Cherry-Picking (applies the diff of a commit on another branch)

```
$ git checkout master  
$ git cherry-pick e43a6
```



[Pro Git, [chapter 5.3](#)]

# Integrate changes between branches

- Cherry-Picking and keep track of the original commit

```
$ git checkout master

$ git cherry-pick -x db3e256ed4a23c92077aa2f136edab95970e8597

$ git show HEAD
commit 841a4e2375b5dc586c283fd4fb6f1f0a9ee443d3 (HEAD, master)
Author: Luc Sarzyniec <luc.sarzyniec@xilopix.com>
Date:   Tue Feb 24 08:27:00 2015 +0100

    commit4

    (cherry picked from commit db3e256ed4a23c92077aa2f136edab95970e8597)
```

# Rewrite history

[Pro Git, [chapter 7.6](#)]

# Rewrite history

- Rewrite (amend) the last commit

```
$ # git add ...; git rm ...  
$ git commit --amend # --no-edit
```

- Rewrite several commits

```
$ git rebase -i HEAD~3  
pick f7f3f6d commit 4  
pick 310154e commit 5  
pick a5f4a0d commit 6  
  
# Rebase 710f0f8..a5f4a0d onto 710f0f8  
#  
# Commands:  
# p, pick = use commit  
# r, reword = use commit, but edit the commit message  
# e, edit = use commit, but stop for amending  
# s, squash = use commit, but meld into previous commit  
# f, fixup = like "squash", but discard this commit's log message  
# x, exec = run command (the rest of the line) using shell
```



# Rewrite history

- Rewrite commit messages only

```
$ git rebase -i HEAD~3  
pick f7f3f6d commit 4  
reword 310154e commit 5  
pick a5f4a0d commit 6
```

- Re-order commits

```
$ git rebase -i HEAD~3  
pick 310154e commit 5 # <-  
pick f7f3f6d commit 4 # ->  
pick a5f4a0d commit 6
```

- Delete commits

```
$ git rebase -i HEAD~3  
pick f7f3f6d commit 4  
pick a5f4a0d commit 6
```

# Rewrite history

Edit several commits

# Edit several commits

- Select which commit to edit

```
$ git rebase -i HEAD~3
edit f7f3f6d commit 4
edit 310154e commit 5
pick a5f4a0d commit 6
# Save and quit
```

Stopped at f7f3f6d ... commit 4  
You can amend the commit now, with

```
git commit --amend
```

Once you are satisfied with your changes, run

```
git rebase --continue
```

- Rewrite the first commit

```
# edit files
$ git add ... # git rm ...
$ git commit --amend
```

# Edit several commits

- Continue with the second commit

```
$ git rebase --continue
Stopped at 310154e ... commit 5

# edit files
$ git add ... # git rm ...
$ git commit --amend

$ git rebase --continue
Successfully rebased and updated refs/heads/master.
```

- Check that everything was done as expected

```
$ git log --oneline -3
53bb260 commit 4 # SHA1 has changed since files were modified
f8765fa new commit 5 # SHA1 has changed since files and message were modified
4fc3652 commit 6 # SHA1 has changed since parents were modified
```

# Rewrite history

Mix commits

# Mix commits

- Select the commits to be mixed (with the previous commit)

```
$ git rebase -i HEAD~3  
pick f7f3f6d commit 4  
squash 310154e commit 5  
pick a5f4a0d commit 6
```

- Create a new commit message

```
# This is a combination of 2 commits.  
# The first commit's message is:  
commit 4  
  
# This is the 2nd commit message:  
commit 5
```

- Check that everything was done as expected

```
$ git log --oneline -2  
pick f7f3f6d commit 4 and 5  
pick a5f4a0d commit 6
```

# Rewrite history

Insert new commits

# Insert new commits

- Select where to insert the commit (after witch existing commit)

```
$ git rebase -i HEAD~3
edit f7f3f6d commit 4
edit 310154e commit 5
pick a5f4a0d commit 6
```

- Add files and create new commits

```
$ git add ... && git commit -m "commit 4-1"
$ git rebase --continue
$ git add ... && git commit -m "commit 5-1"
$ git add ... && git commit -m "commit 5-2"
$ git rebase --continue
```

- Check that everything was done as expected

```
$ git log --oneline -6
f7f3f6d commit 4
0737964 commit 4-1
310154e commit 5
fa96cb9 commit 5-1
26cd81d commit 5-2
cc4ad9a commit 6
```



# Rewrite history

Split commits

# Split commits

- Select the commits to split

```
$ git rebase -i HEAD~3  
pick f7f3f6d commit 4  
edit 310154e commit 5  
pick a5f4a0d commit 6
```

- Reset the current commit

```
$ git reset HEAD~
```

- Create several new commits

```
$ git add ...  
$ git commit -m 'first'  
  
$ git add ...  
$ git commit -m 'second'  
  
$ git add ...  
$ git commit -m 'third'
```

# Split commits

- Continue once it's done

```
$ git rebase --continue  
Successfully rebased and updated refs/heads/master.
```

- Check that everything was done as expected

```
$ git log --oneline -5  
f7f3f6d commit 4  
66b1120 first  
afcd336 second  
4fc3652 third  
a5f4a0d commit 6
```

# Automatically rewrite history

- Automatically rewrite **all** the history

```
git filter-branch --tree-filter 'rm -f passwords.txt' HEAD
```

- Change your email address

```
git filter-branch --commit-filter '  
if [ "$GIT_AUTHOR_EMAIL" = "schacon@localhost" ];  
then  
    GIT_AUTHOR_NAME="Scott Chacon";  
    GIT_AUTHOR_EMAIL="schacon@example.com";  
    git commit-tree "$@";  
else  
    git commit-tree "$@";  
fi' HEAD
```

# Debugging

[Pro Git, [chapter 7.10](#)]

# Code introspection

- Read the code annotated with commit/line

```
$ git blame -L 1,10 zlib.c
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 1) /*
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 2)  * zlib wrappers to make sure we don't
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 3)  * at init time.
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 4)  */
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 5) #include "cache.h"
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 6)
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 7) static const char *zerr_to_string(int s
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 8) {
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 9)     switch (status) {
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 10)         case Z_MEM_ERROR:
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 11)             return "out of memory";
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 12)         case Z_VERSION_ERROR:
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 13)             return "wrong version";
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 14)         case Z_NEED_DICT:
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 15)             return "needs dictionary";
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 16)         case Z_DATA_ERROR:
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 17)             return "data stream error";
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 18)         case Z_STREAM_ERROR:
1a507fc1 (Junio C Hamano 2011-06-10 10:31:34 -0700 19)             return "stream consistency error";
b0613ce0 (Jonathan Nieder 2010-11-06 06:47:34 -0500 20)         default:
```

# Code introspection

- In short format

```
$ git blame -L 1,10 -s zlib.c
b0613ce0 1) /*
b0613ce0 2)  * zlib wrappers to make sure we don't silently miss errors
b0613ce0 3)  * at init time.
b0613ce0 4)  */
b0613ce0 5) #include "cache.h"
b0613ce0 6)
1a507fc1 7) static const char *zerr_to_string(int status)
b0613ce0 8) {
1a507fc1 9)     switch (status) {
b0613ce0 10)         case Z_MEM_ERROR:
1a507fc1 11)             return "out of memory";
b0613ce0 12)         case Z_VERSION_ERROR:
1a507fc1 13)             return "wrong version";
1a507fc1 14)         case Z_NEED_DICT:
1a507fc1 15)             return "needs dictionary";
1a507fc1 16)         case Z_DATA_ERROR:
1a507fc1 17)             return "data stream error";
1a507fc1 18)         case Z_STREAM_ERROR:
1a507fc1 19)             return "stream consistency error";
b0613ce0 20)         default:
```

# Code introspection

- See where sections of code originally came from

```
$ git blame -s -C -L 1,20 zlib.c
b0613ce0 zlib.c      1) /*
39c68542 wrapper.c  2)  * zlib wrappers to make sure we don't silently miss errors
39c68542 wrapper.c  3)  * at init time.
39c68542 wrapper.c  4)  */
b0613ce0 zlib.c      5) #include "cache.h"
b0613ce0 zlib.c      6)
1a507fc1 zlib.c      7) static const char *zerr_to_string(int status)
b0613ce0 zlib.c      8) {
1a507fc1 zlib.c      9)     switch (status) {
b0613ce0 zlib.c     10)     case Z_MEM_ERROR:
1a507fc1 zlib.c     11)         return "out of memory";
b0613ce0 zlib.c     12)     case Z_VERSION_ERROR:
1a507fc1 zlib.c     13)         return "wrong version";
1a507fc1 zlib.c     14)     case Z_NEED_DICT:
1a507fc1 zlib.c     15)         return "needs dictionary";
1a507fc1 zlib.c     16)     case Z_DATA_ERROR:
1a507fc1 zlib.c     17)         return "data stream error";
1a507fc1 zlib.c     18)     case Z_STREAM_ERROR:
1a507fc1 zlib.c     19)         return "stream consistency error";
b0613ce0 zlib.c     20)     default:
```



# Track a bug using binary search

- Start to search, specify the commit of the last working version

```
$ git bisect start HEAD v2.2.0  
Bisecting: 150 revisions left to test after this (roughly 7 steps)
```

- At each step specify if the current snapshot is working or not

```
# Do some tests  
$ git bisect good  
Bisecting: 75 revisions left to test after this (roughly 6 steps)  
  
# Do some tests  
$ git bisect bad  
Bisecting: 37 revisions left to test after this (roughly 5 steps)  
  
# ...
```

- Find the version that introduced the bug (-> read the diff to understand)

```
# ...  
bcbdeb1a1256f777e52192fa7da0f7dbad680162 is the first bad commit  
  
$ git show -p bcbdeb1a1256f777e52192fa7da0f7dbad680162
```

# Track a bug automating binary search

- Find a command or create a script to reproduce the bug

```
rake test # ?
```

- Start the binary search

```
$ git bisect start HEAD v2.2.0
```

- Use the script to automatically run the binary search

```
$ git bisect run rake test
```

- Stop the binary search procedure

```
$ git bisect reset
```

[See <http://lwn.net/Articles/317154/>]

# Other useful commands

[Pro Git, [chapter 7](#)]

# Other useful commands

- Grep in a specific commit

```
git grep test 49e4c29
49e4c29:lib/disco/common/service.rb:      test_connect()
49e4c29:lib/disco/common/service.rb:  def test_connect()
```

- Find in which tag a commit was included

```
$ git describe --tag 49e4c299dc390698724da5d21de853c44737c65c
0.1.0
```

- Remove untracked files from the working directory

```
$ git clean # -d to remove directories too
```

- Create an archive of the repository (a commit/tag can be specified)

```
$ git archive -o soft-2.2.0.tar.gz v2.2.0
```

# Other useful commands

- Resolve conflicts using an external (GUI?) tool

```
$ git mergetool
```

[Pro Git, [chapter 3.2](#)]

- Share changes saving commits in a bundle file (can be sent by mail, ...)
  - Create the bundle file

```
$ git bundle create repo.bundle HEAD master
```

- Load the downloaded bundle file

```
$ git clone repo.bundle repo
```

```
$ git fetch ../commits.bundle master:other-master
```

[Pro Git, [chapter 7.12](#)]

# Memo

```
$ git add -p
$ git checkout -- FILE
$ git reset REV # --soft/--hard
$ git stash
$ git commit --amend
$ git diff REV -- FILE
$ git diff --color-words
$ git show REV:FILE
$ git log --decorate --graph
$ git fetch origin BRANCH:OTHER_BRANCH
$ git revert REV
$ git rebase -i REV
$ git cherry-pick -x REV
$ git blame FILE
$ git bisect REV_END REV_START
$ git grep STR REV
$ git clean
$ git archive -o FILE.tar.gz REV
```

# Internals

[Pro Git, [chapter 10](#)]

# Git: content-addressable filesystem

- Object database, index = SHA1 hash
- Objects are stored in the filesystem under the `.git/objects` directory
- Several kind of objects: commit, tree, blob, ...
- Objects linking each-other (commits, trees)
- Compression on demand or when files are too big



# Git objects: blobs

- Create and store a new blob (file) object:

```
$ echo "Awesome!" | git hash-object --stdin -w  
6d4ed2c98c4fbe835280634af0cbddefffaf7ee6  
  
$ touch file && git hash-object -w file  
e69de29bb2d1d6434b8b29ae775ad8c2e48c5391
```

- Find this object in the filesystem

```
$ find .git/objects/  
.git/objects/6d/  
.git/objects/6d/4ed2c98c4fbe835280634af0cbddefffaf7ee6
```

- Get information about the object

```
$ git cat-file -t 6d4ed2c98c4fbe835280634af0cbddefffaf7ee6  
blob  
  
$ git cat-file -p 6d4ed2c98c4fbe835280634af0cbddefffaf7ee6  
Awesome!
```

# Git objects representation

- Content of the file associated to the object

```
$ cat .git/objects/6d/4ed2c98c4fbe835280634af0cbddefffaf7ee6  
xKÊÉÓR°dp,O-ÎÏMUä,S
```

- *deflate* (zip,gzip,zlib,...) decompressed content

```
$ cat .git/objects/6d/4ed2c98c4fbe835280634af0cbddefffaf7ee6 | \  
ruby -r zlib -e "p Zlib::Inflate.inflate(STDIN.read)"  
"blob 9\x00Awesome!\n"
```

- Calculation of the SHA1 hash associated of the object

```
$ ruby -r digest/sha1 -e 'p Digest::SHA1.hexdigest("blob 9\x00Awesome!\n")'  
"6d4ed2c98c4fbe835280634af0cbddefffaf7ee6"
```

# Git objects: commits and trees

- Content of a commit object

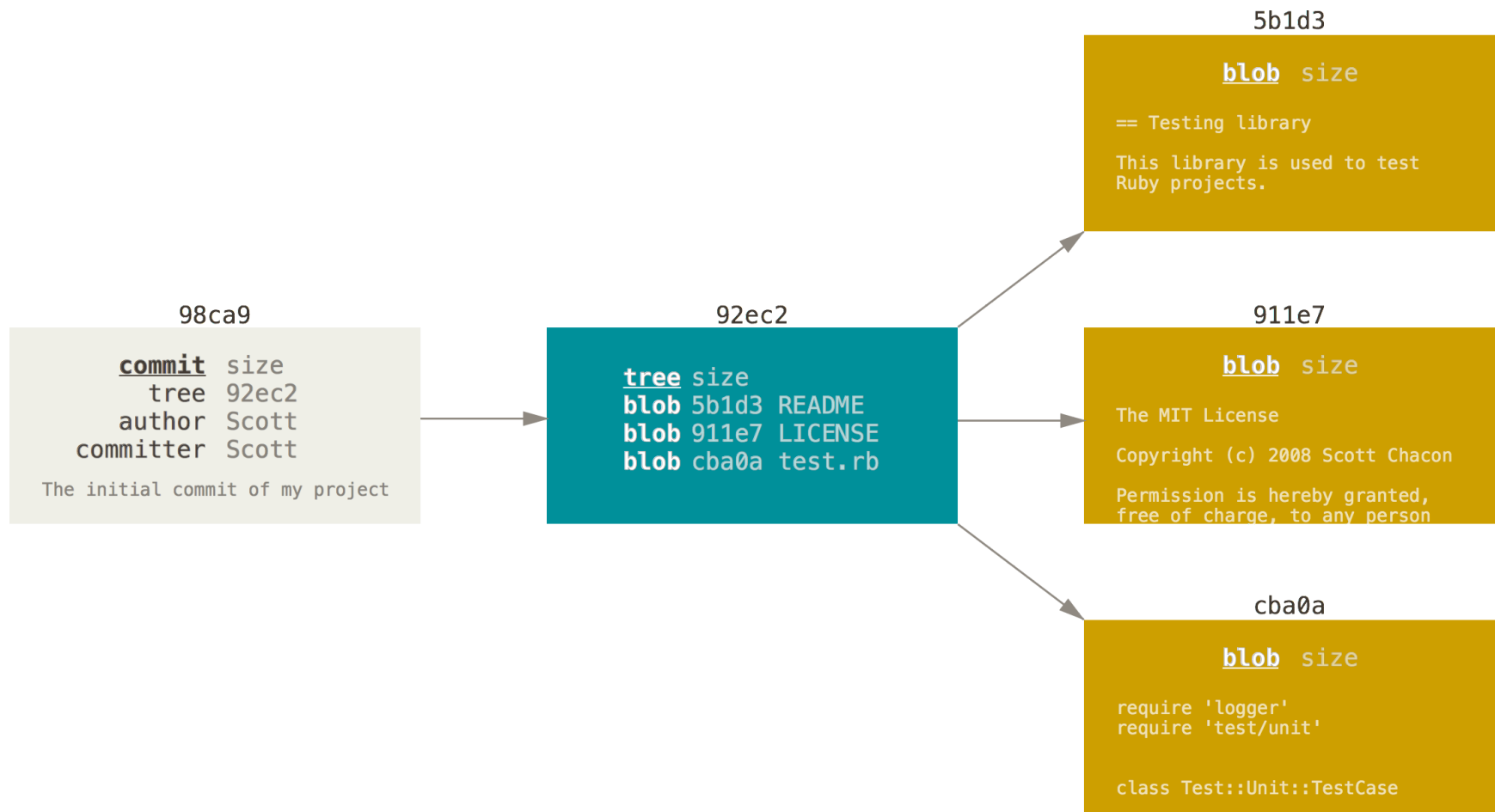
```
$ git cat-file -p $(git rev-parse HEAD) # 00c4dfce3c28787870d2574a50c5de3725d5fcfb
tree 4814e377c18f2da9cce56631f24e0d09181b0feb
parent e8a0d201e0b701d7c2de28cb33fa03ef59b22506
author Luc Sarzyniec <luc.sarzyniec@xilopix.com> 1424853891 +0100
committer Luc Sarzyniec <luc.sarzyniec@xilopix.com> 1424853895 +0100
```

Commit message

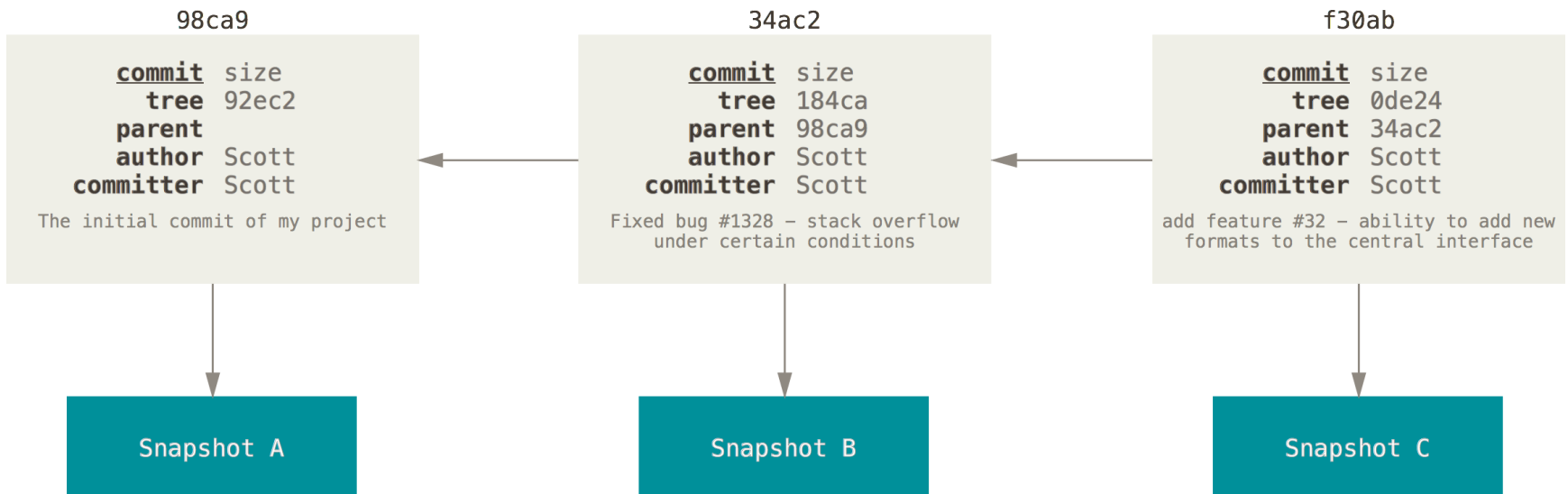
- Content of a tree object

```
$ git cat-file -p $(git rev-parse HEAD^{tree}) # 4814e377c18f2da9cce56631f24e0d09181b0feb
040000 tree e4af7700f8c091d18cc15f39c184490125fb0d17    dir
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    file1
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    file3
$ git cat-file -p e4af7700f8c091d18cc15f39c184490125fb0d17
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    file2
```

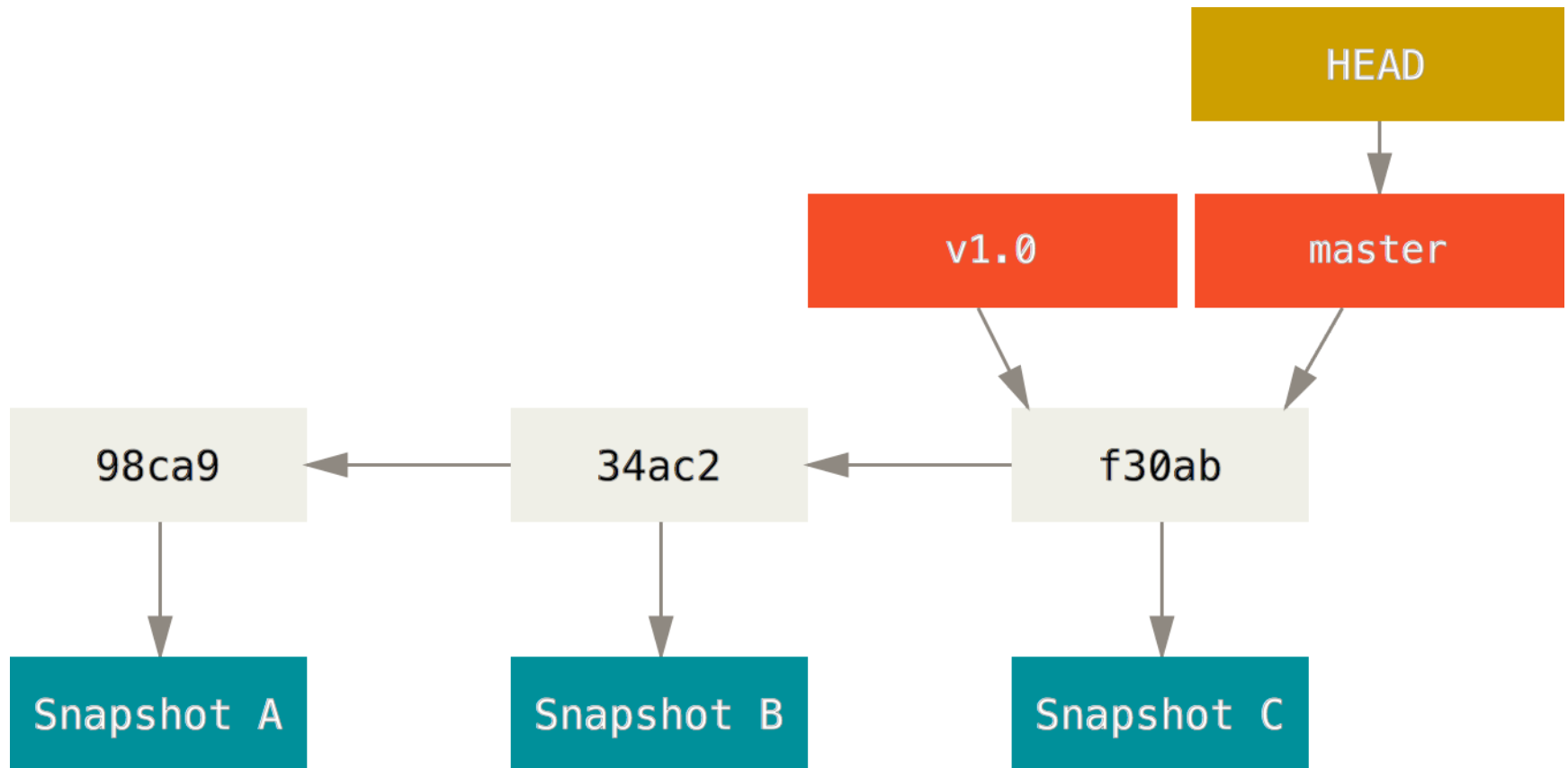
# A commit



# History



# Branches



# Branches

- Branch = pointer on a commit object

```
$ cat .git/refs/heads/master  
7f4ba4b6e3ba7075ca6b379ba23fd3088cbe69a8
```

- HEAD = pointer on the current branch

```
$ cat .git/HEAD  
ref: refs/heads/master
```

- Create a branch

```
$ echo 7f4ba4b6e3ba7075ca6b379ba23fd3088cbe69a8 > .git/refs/heads/test
```

- Local and remote branches

```
$ find .git/refs -type f  
.git/refs/remotes/origin  
.git/refs/remotes/origin/HEAD  
.git/refs/remotes/origin/master  
.git/refs/heads/master
```

**Thank you !**