Imperial College
London

Git Introduction

Matthew Citron

Why you should use git

- No change is ever irreversible - can develop without fear!
- Uses range from writing a latex report to collaboration on the linux kernel
- Using remotes makes collaboration easy and provides backup of entire history of project
- Simple and very fast to use (c.f. svn, cvs).

Git version at IC

- The version of git on the lx0n machines is out of date (git 1.5.5 from 2008….)
- Many nice (and simplifying) features added since
- Can source latest version by adding to .bashrc:

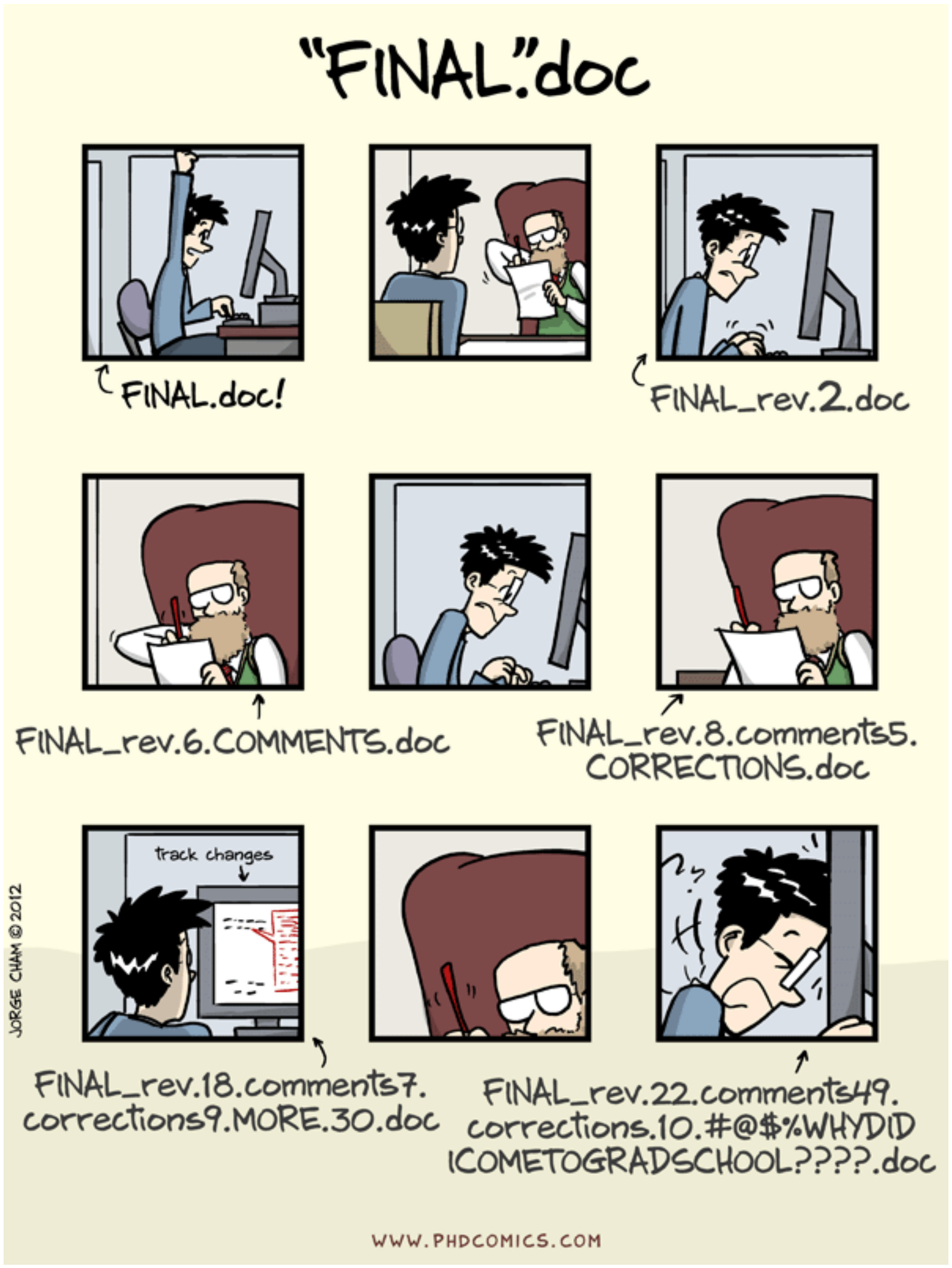
```
export PATH = home/hep/mc3909/git:$PATH
```

- Or download at https://github.com/git/git

Outline

- Version Control
- Git Basics
- Branching and merging
- Remotes
- Rebasing

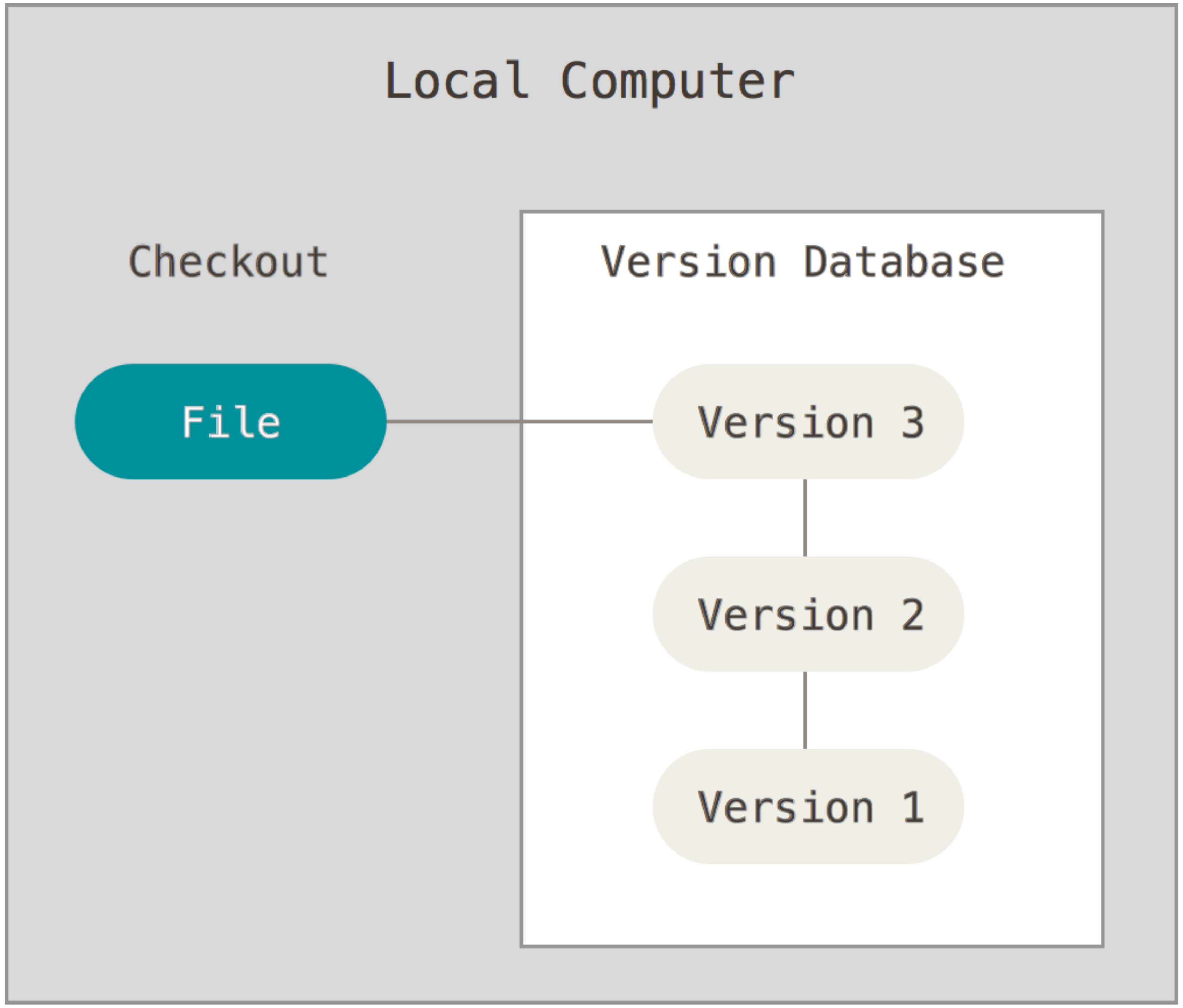
Version Control



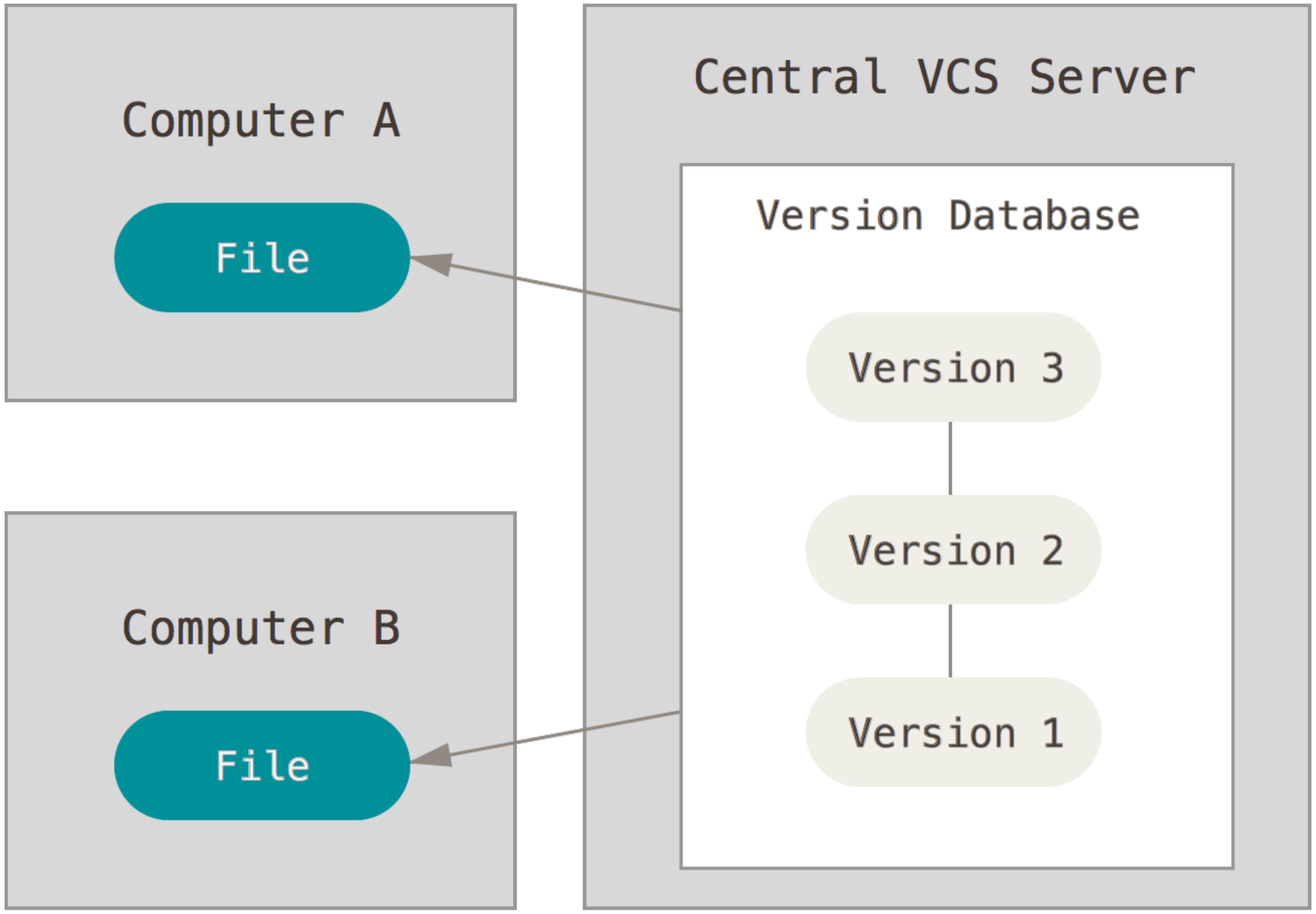
"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDIDICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Version control

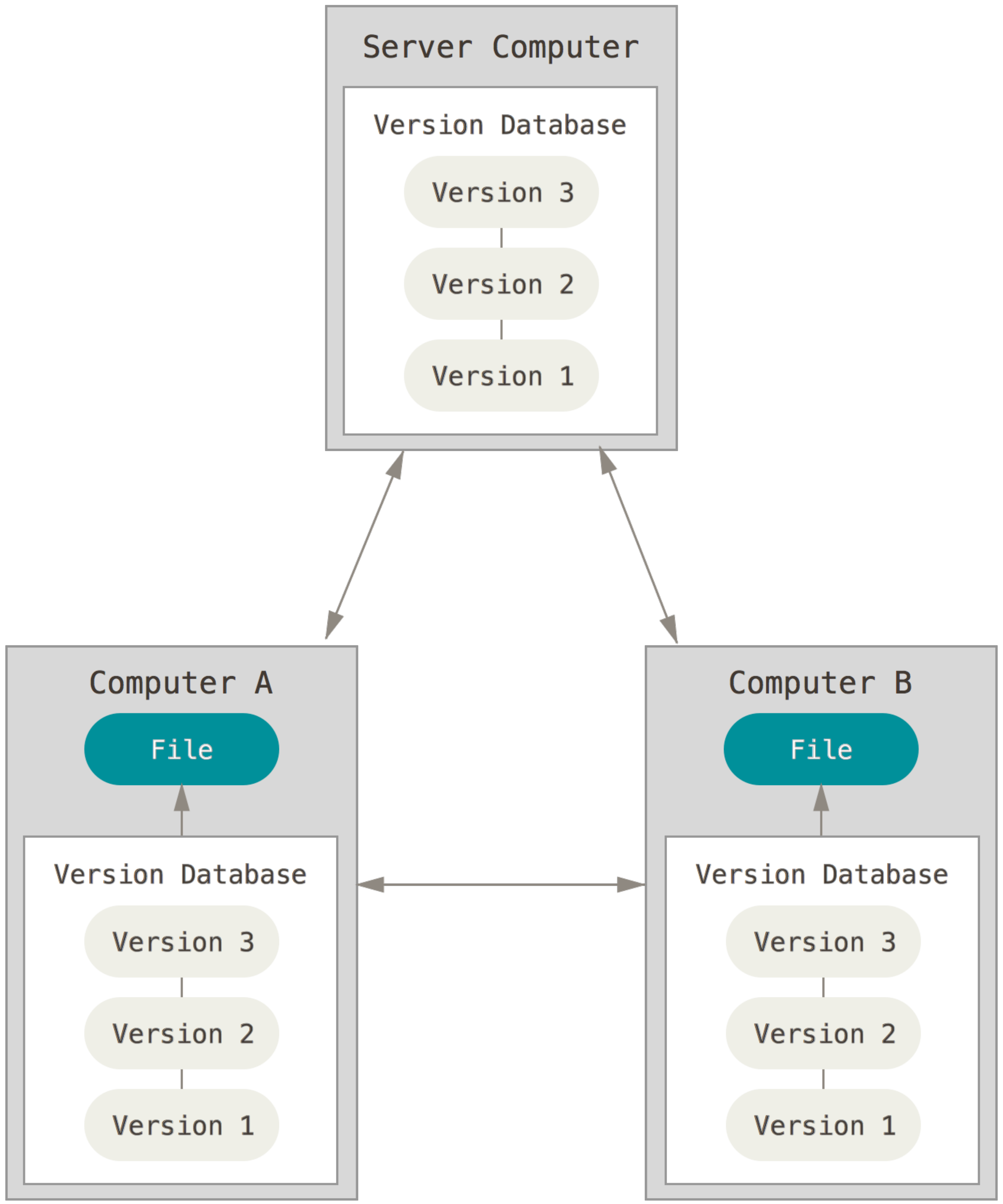
- Version control allows any changes to be reverted
- Allows stable release(s) while developing
- Can provide backup and collaboration
- Three main types
  - Local version control (RCS)
  - Centralised version control (CVS, SVN)
  - Distributed version control (git)



Local version control



Centralised version control



Distributed version control

Git Basics

Git Basics

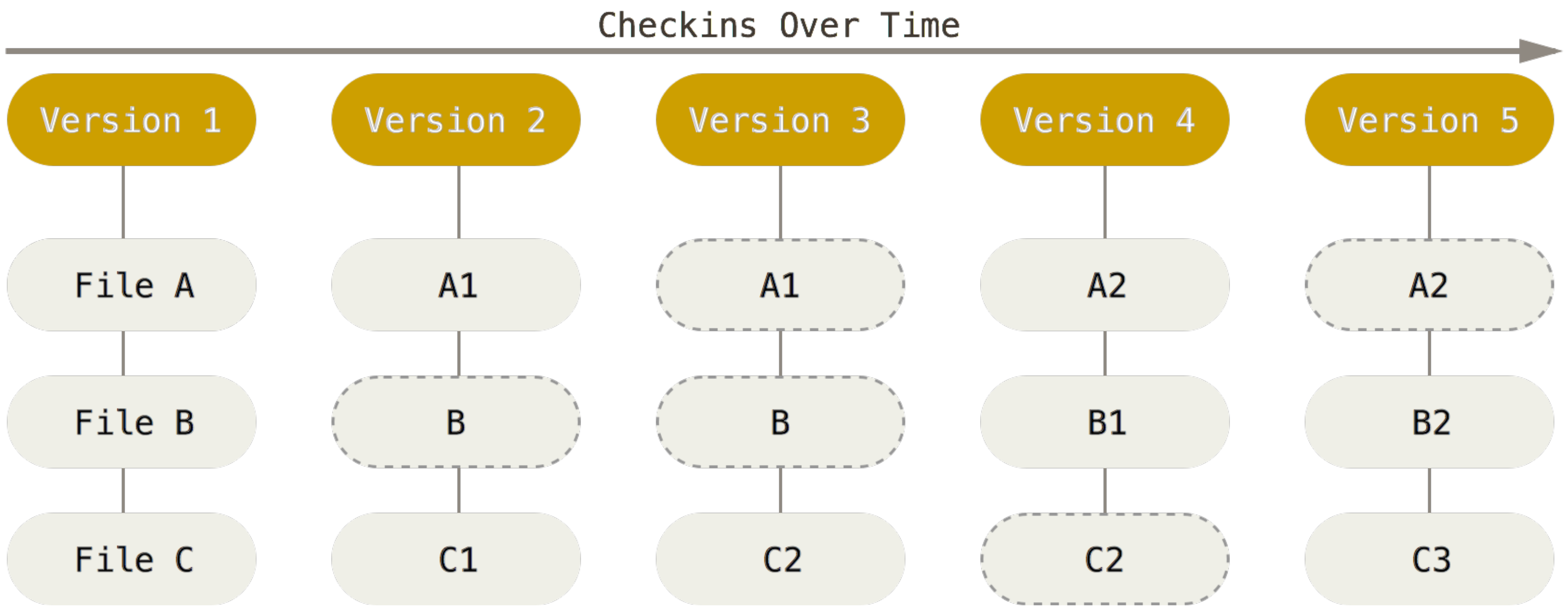
- The git repository contains the entire history of the project

- A git repository can be local or remote
  - Most operations local

- Every time you commit (save the state of the project) git stores snapshot of repository (repo)

- Git operations generally add data - (almost) everything is reversible

Git repository snapshots



Note: This and all other figures not otherwise credited taken from http://git-scm.com/book/en/v2
(Pro-git manual by Scott Chacon and Ben Straub)

The pro-git manual is an excellent resource for learning about git (especially git workflows)

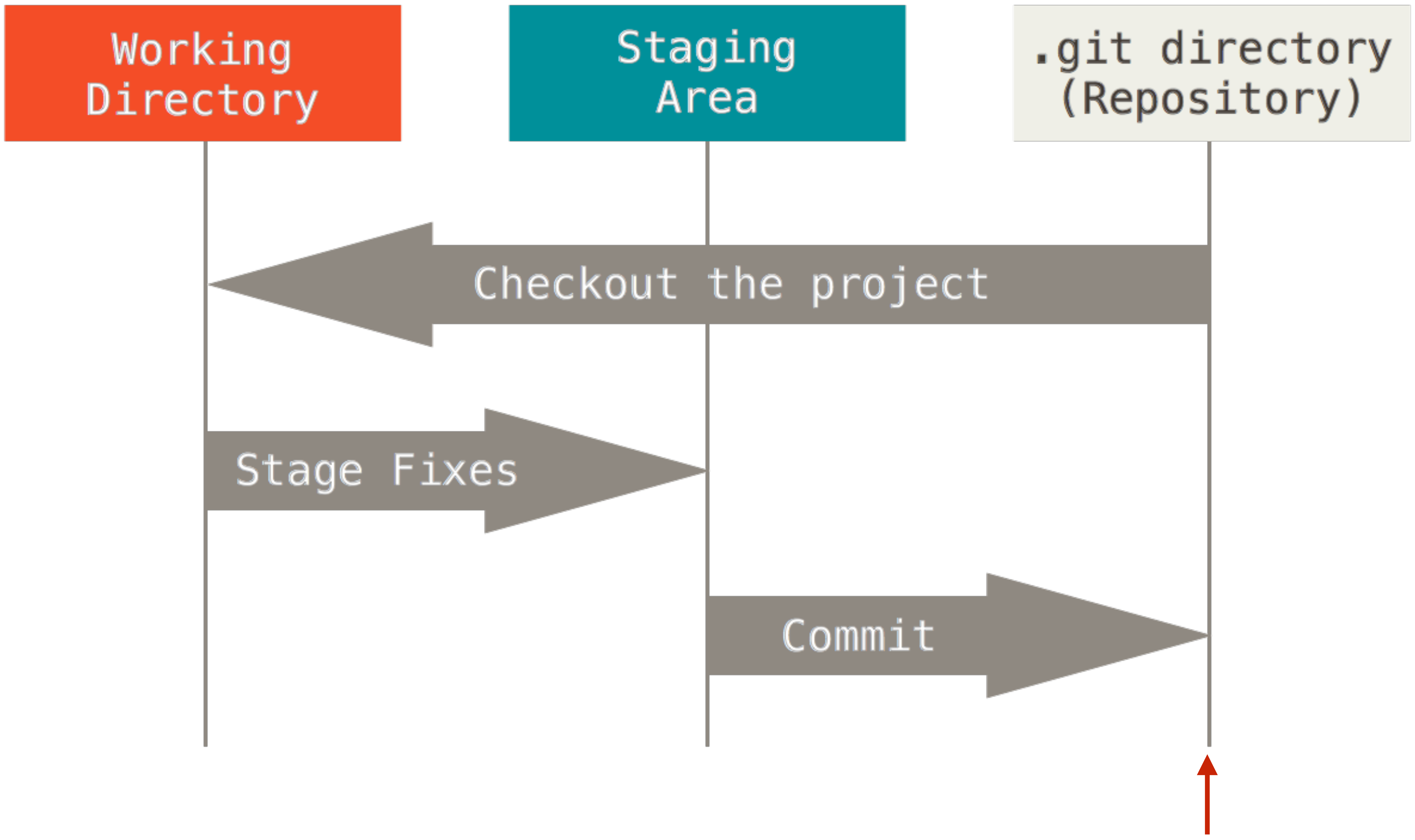
Basic git repo commands

- git init
  - Makes a skeleton git repository
  - Should be run in the top folder of your project
  - Initially no files will be tracked (see later)
- git clone <url of git repo>
  - Makes a copy of an existing git repository
  - Will include entire history by default
- git grep <string>
  - Search repository (very fast and can even search entire history)

Git Workflow

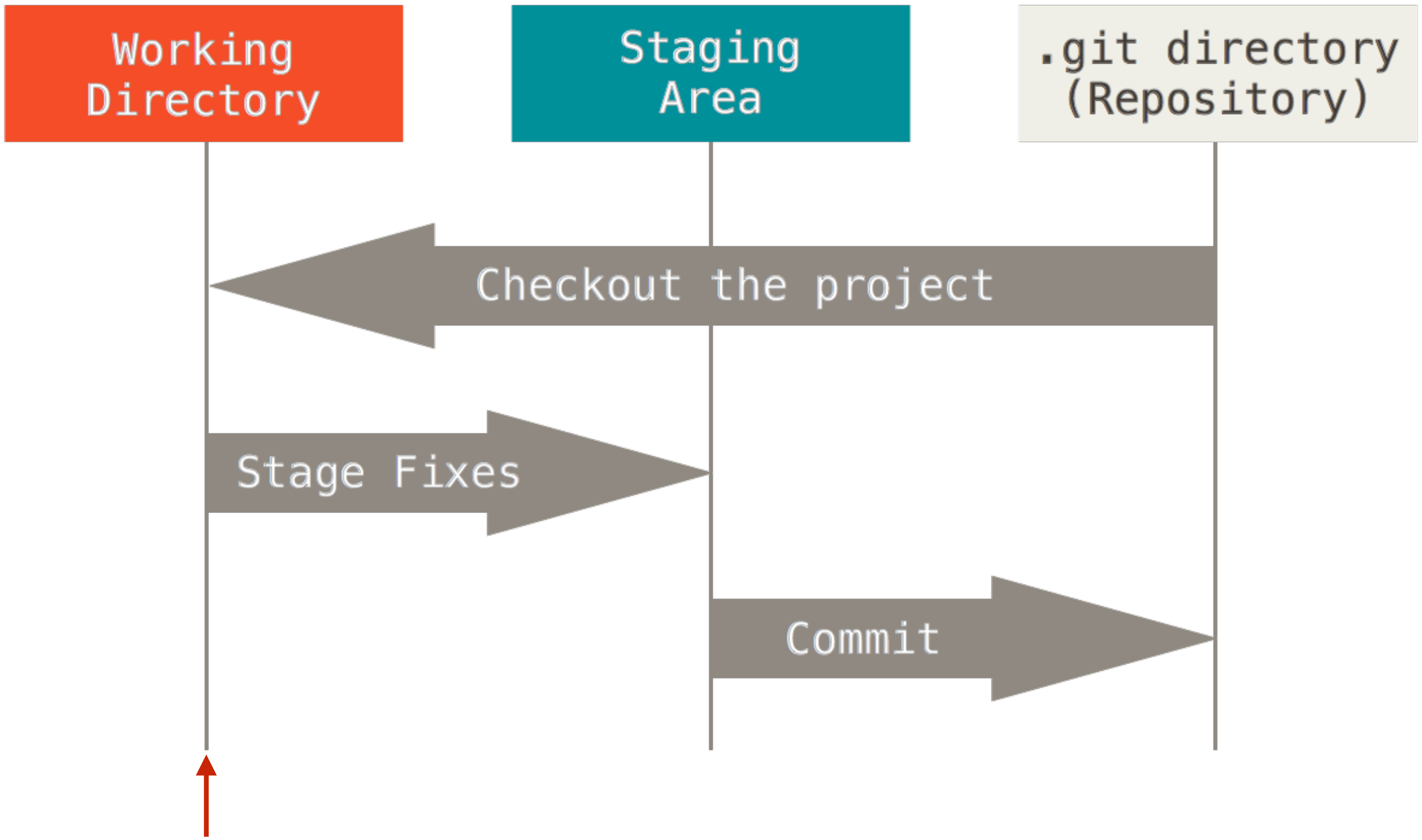
- Three main states tracked files can be in: committed, modified or staged
- **Committed** - stored in git’s database
- **Modified** - files with changes not yet committed
- **Staged** - modified files marked to be committed
- Untracked files are those not included in the previous snapshot

Three main areas of the repo



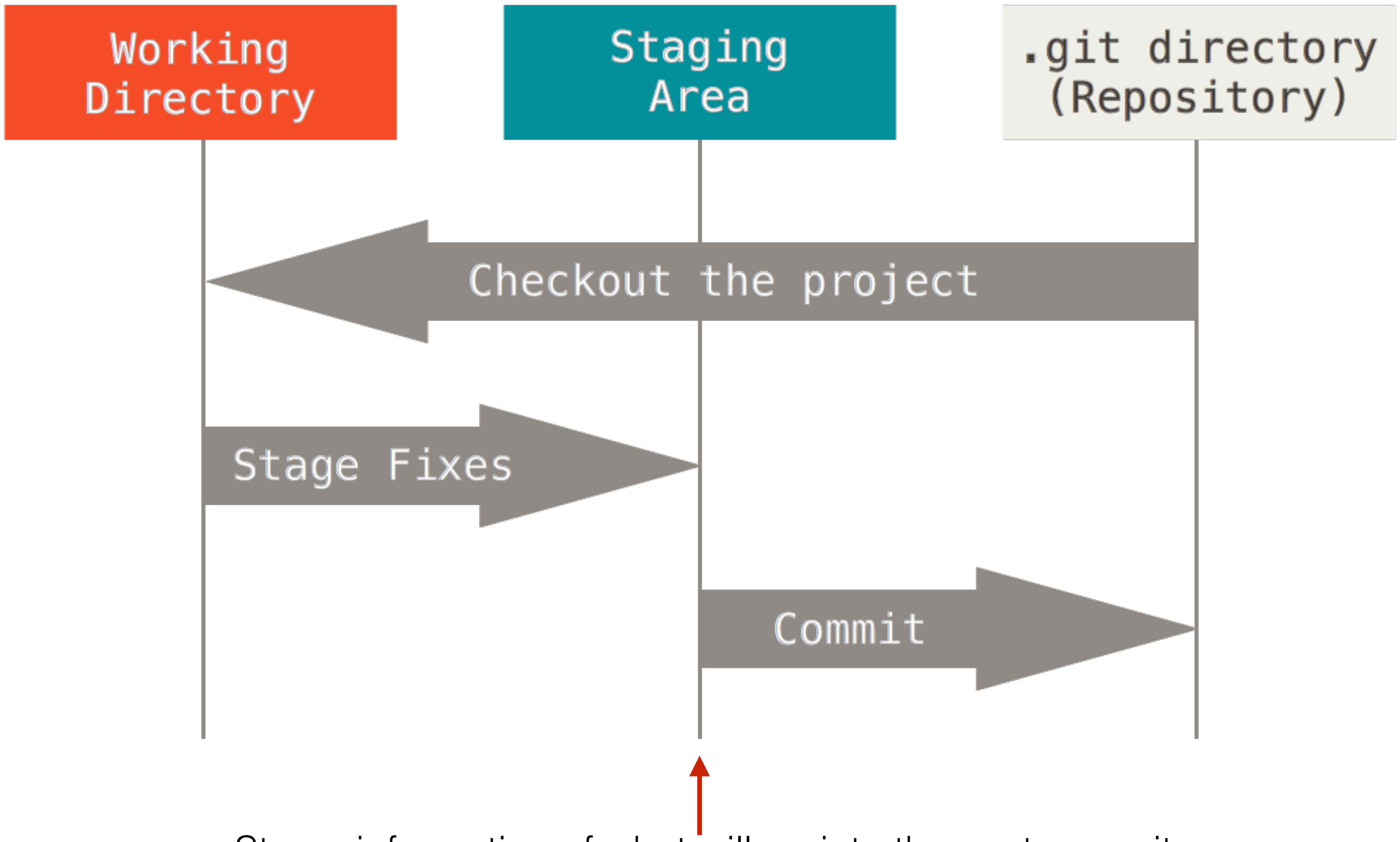
Where git stores the metadata and object database for the project

Three main areas of the repo



One version of the project read from the .git directory (modifiable)

Three main areas of the repo



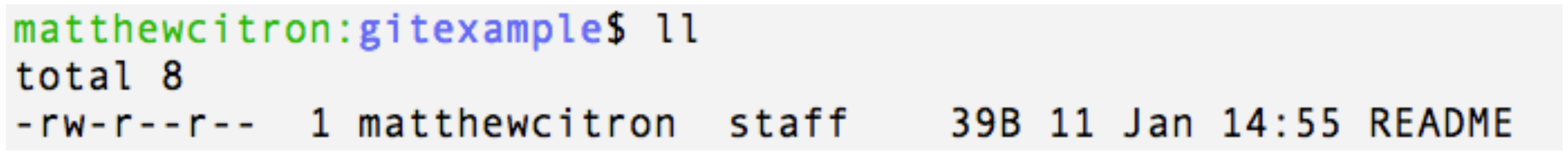
Stores information of what will go into the next commit

Basic git file commands

- git status
  - To find the status of all the files in the repo (untracked, unmodified, modified or staged)
- git add <filename>
  - Adds an untracked or modified file to the staging area
- git commit -m “<Message>”
  - Takes a snapshot of all files in the staging area
- git log
  - git commit history
  - Many useful options (http://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History)

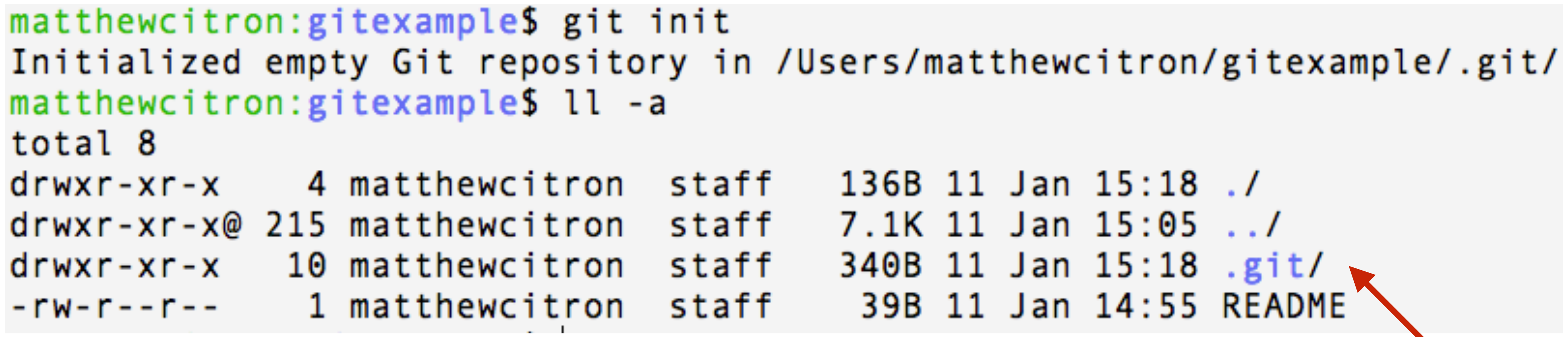
Basic example

Have folder containing project (just README file)



```
matthewcitron:gitexample$ ll
total 8
-rw-r--r--  1 matthewcitron  staff    39B 11 Jan 14:55 README
```

Run git init in project folder

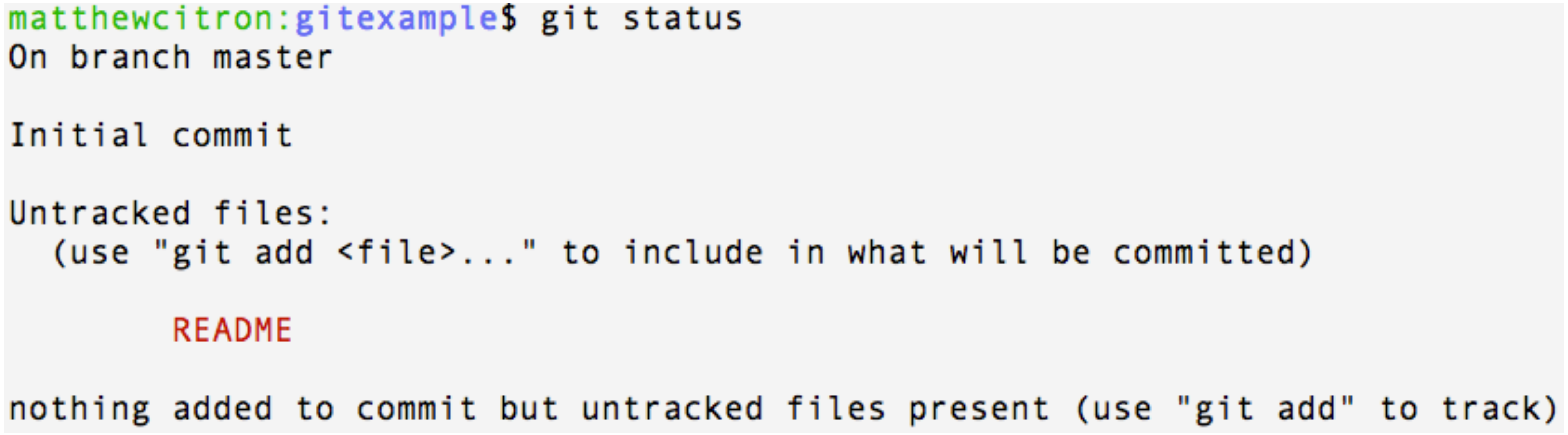


```
matthewcitron:gitexample$ git init
Initialized empty Git repository in /Users/matthewcitron/gitexample/.git/
matthewcitron:gitexample$ ll -a
total 8
drwxr-xr-x    4 matthewcitron  staff   136B 11 Jan 15:18 ./
drwxr-xr-x@ 215 matthewcitron  staff   7.1K 11 Jan 15:05 ../
drwxr-xr-x   10 matthewcitron  staff   340B 11 Jan 15:18 .git/
-rw-r--r--    1 matthewcitron  staff    39B 11 Jan 14:55 README
```

Adds .git directory

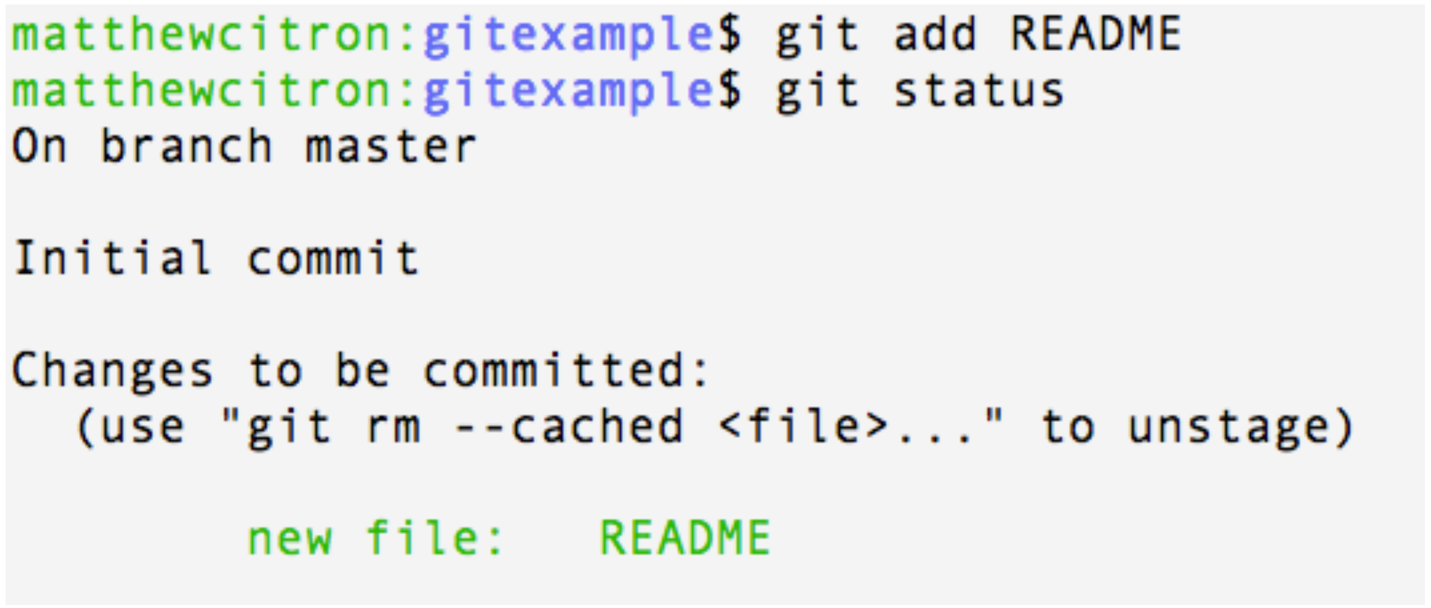
Basic example

Run git status - one untracked file



```
matthewcitron:gitexample$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README

nothing added to commit but untracked files present (use "git add" to track)
```

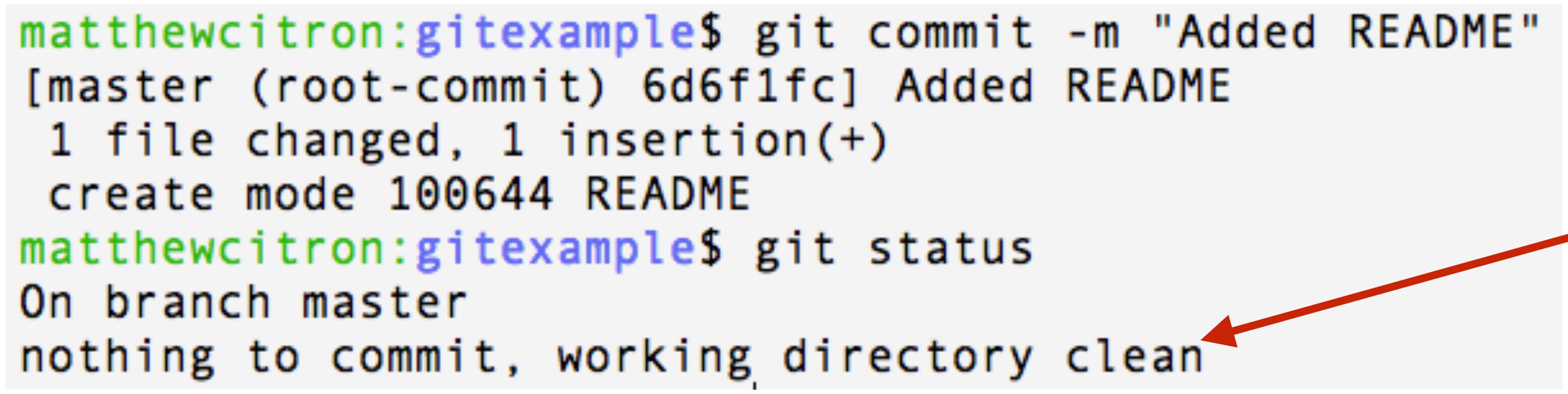
Add to staging area with git add



```
matthewcitron:gitexample$ git add README
matthewcitron:gitexample$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README
```

Basic example

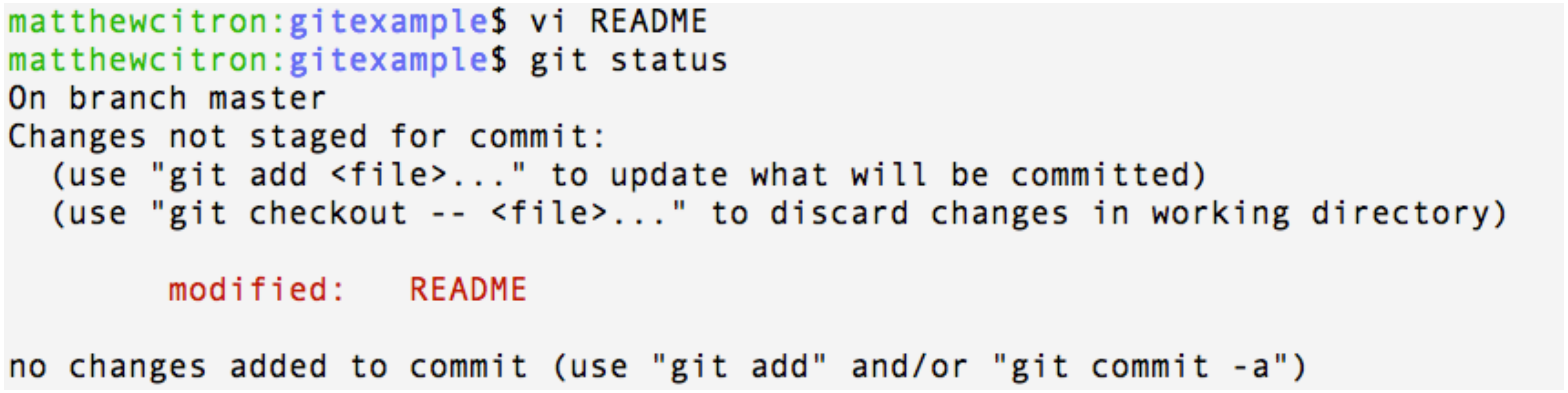
Take snapshot of repo (git commit)



```
matthewcitron:gitexample$ git commit -m "Added README"
[master (root-commit) 6d6f1fc] Added README
 1 file changed, 1 insertion(+)
 create mode 100644 README
matthewcitron:gitexample$ git status
On branch master
nothing to commit, working directory clean
```

File unmodified after commit

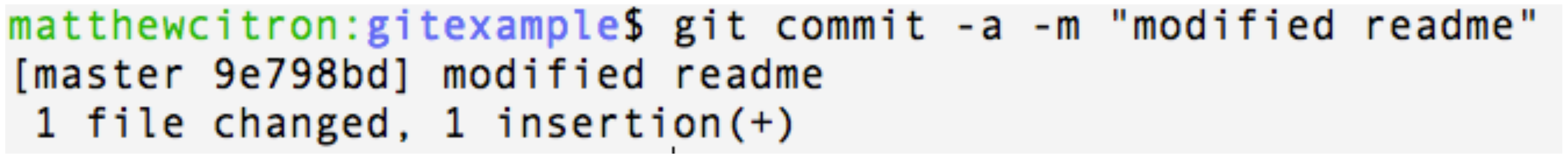
Editing file will change status to modified



```
matthewcitron:gitexample$ vi README
matthewcitron:gitexample$ 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:   README

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

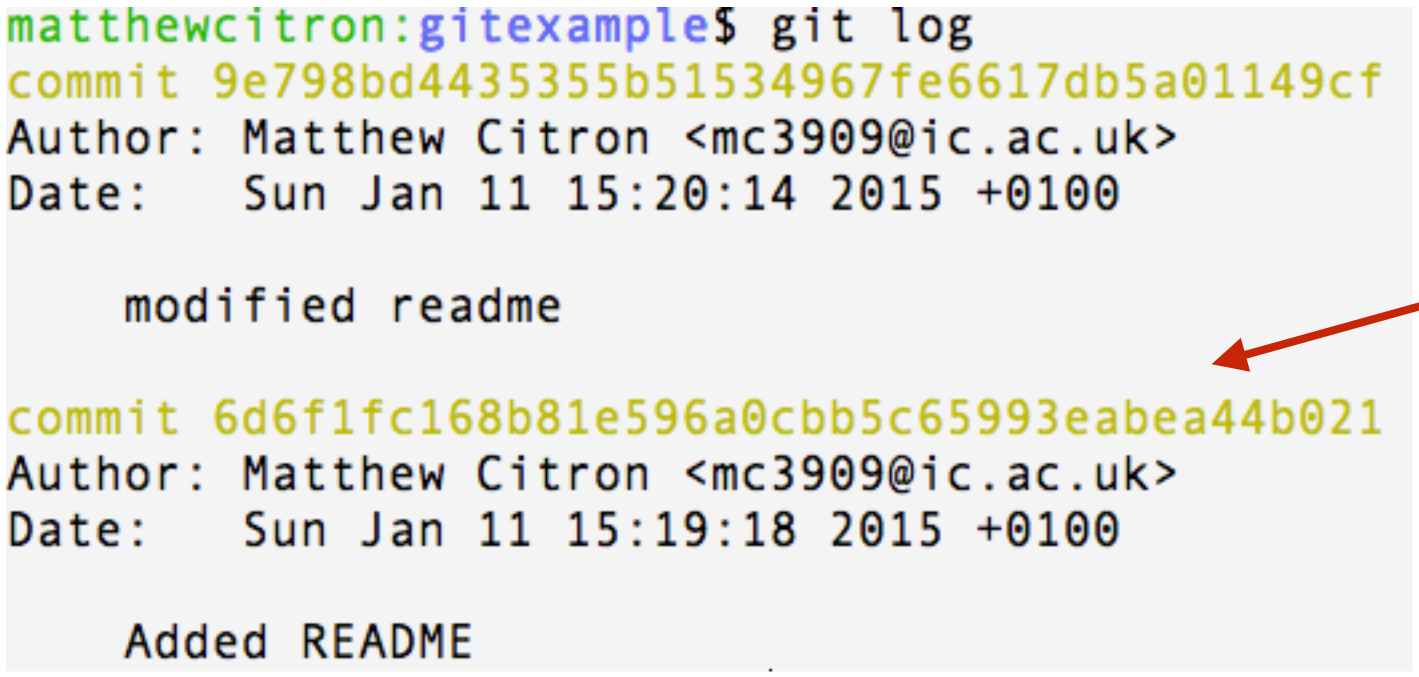
Basic example

Can directly commit changes (skip staging area) using flag -a



```
matthewcitron:gitexample$ git commit -a -m "modified readme"
[master 9e798bd] modified readme
 1 file changed, 1 insertion(+)
```

See commit history with git log



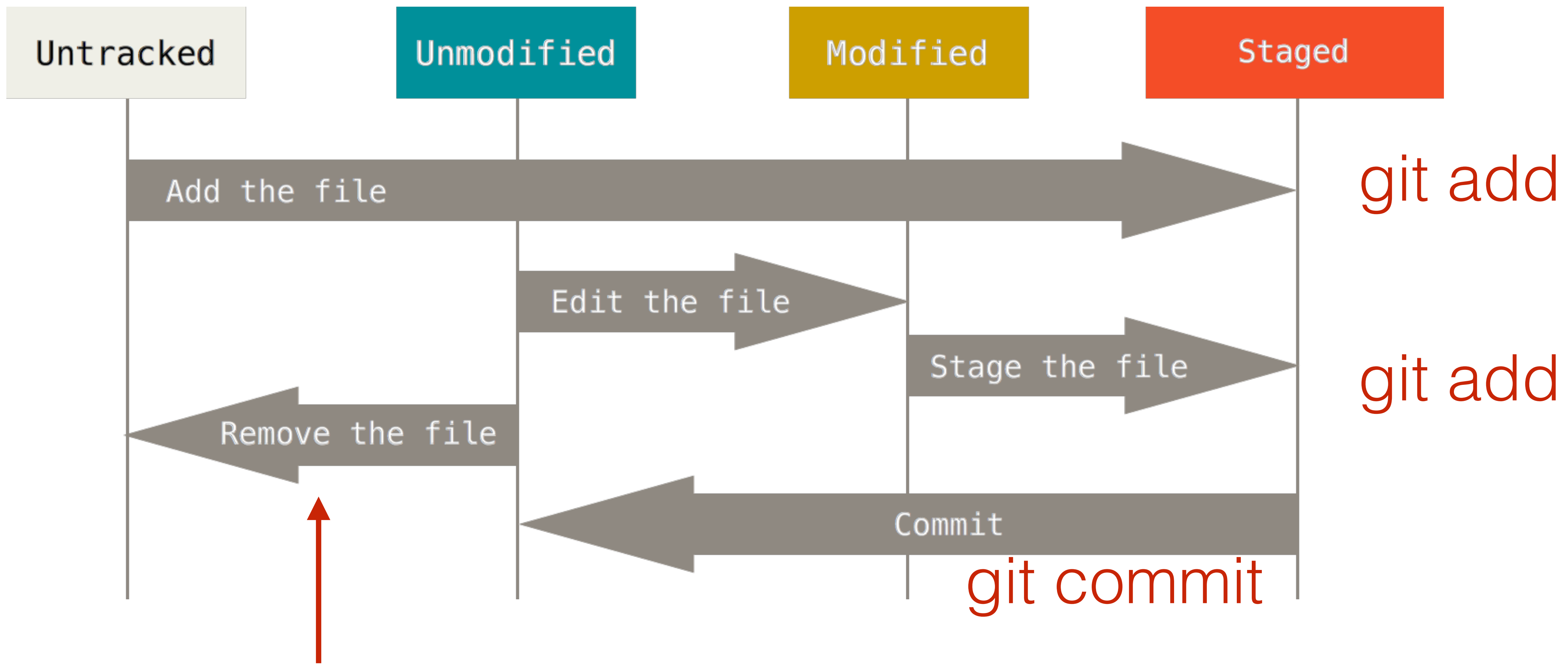
```
matthewcitron:gitexample$ git log
commit 9e798bd4435355b51534967fe6617db5a01149cf
Author: Matthew Citron <mc3909@ic.ac.uk>
Date:   Sun Jan 11 15:20:14 2015 +0100

    modified readme

commit 6d6f1fc168b81e596a0cbb5c65993eabea44b021
Author: Matthew Citron <mc3909@ic.ac.uk>
Date:   Sun Jan 11 15:19:18 2015 +0100

    Added README
```

SHA-1 checksum to identify commit (Can be used to directly access commit)

Undoing things

- git commit --amend
    - commit amends previous commit
    - **Don't** do this to pushed commits
- git reset HEAD <file>
    - Unstages file
- git checkout -- [file]
    - Undoes all changes since last commit
    - Dangerous! All changes will be lost.

Summary



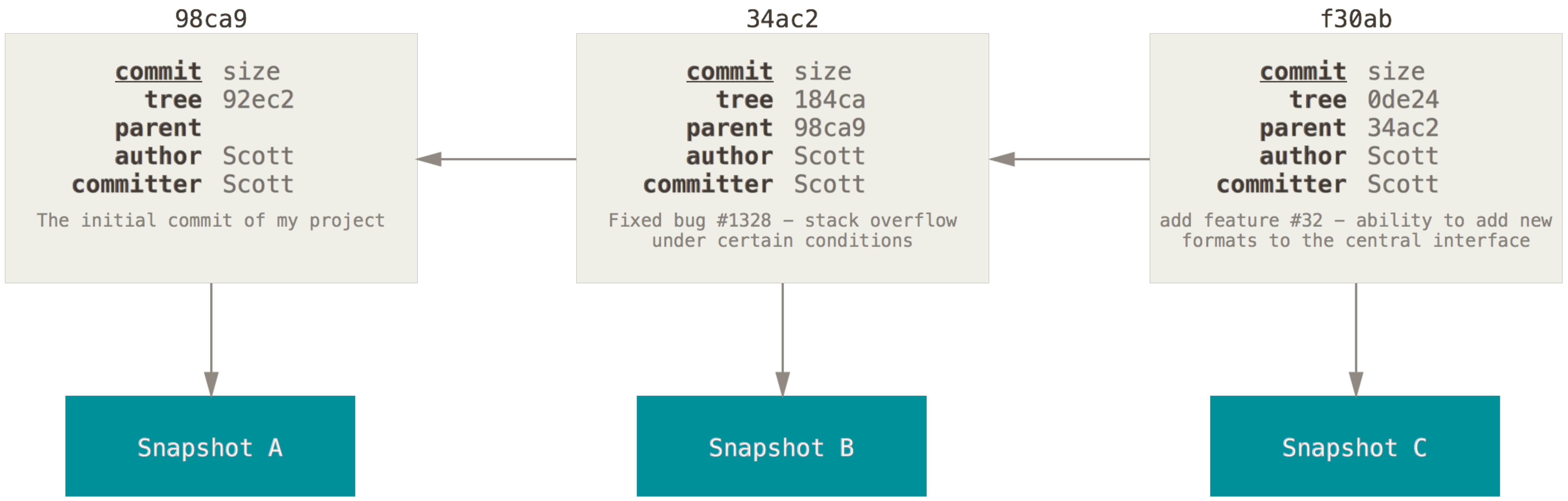
git rm (also deletes file)
git rm --cached (only unstages)

Commit early, commit often! - easy to see where/when things changed

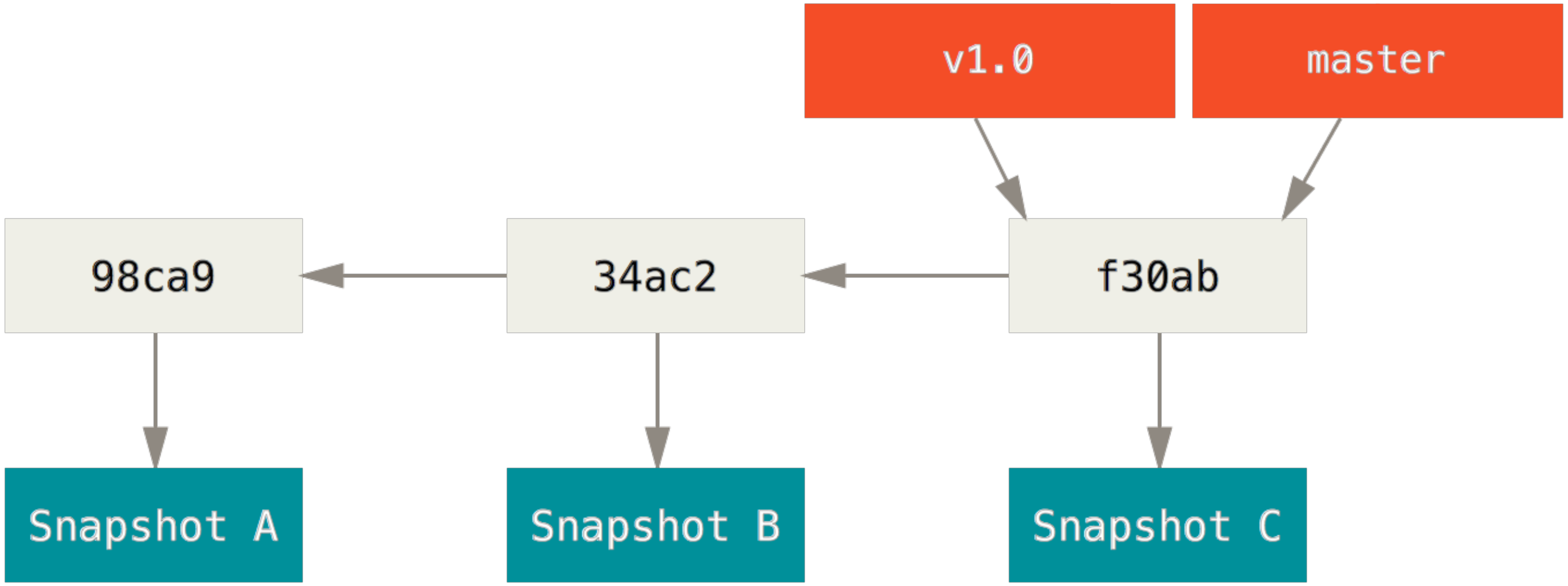
Branches and merging

What is a branch?

- Every commit stores a pointer to its snapshot as well as a pointer to the commit that came before
- A branch is just a pointer to one of the commits

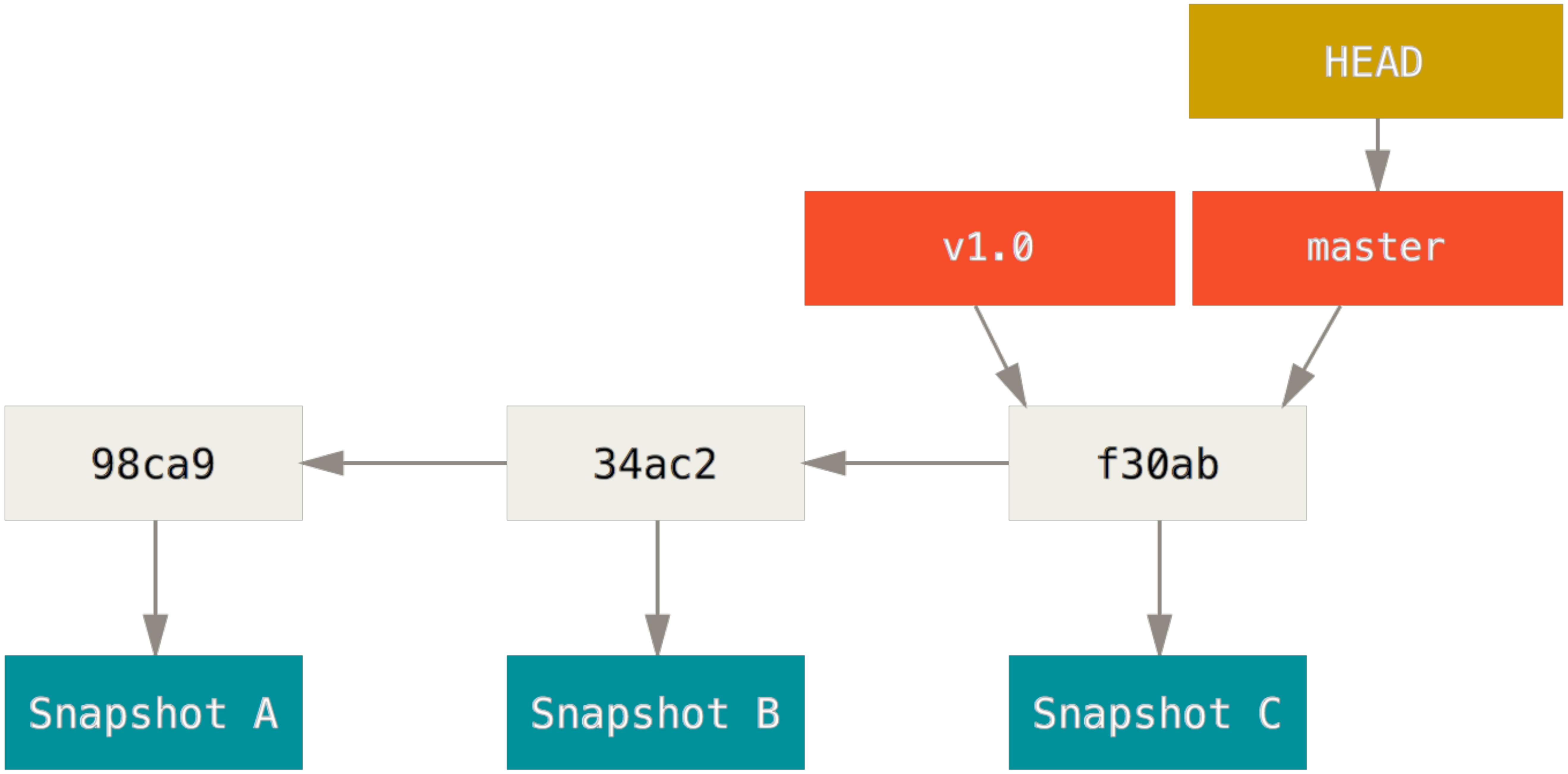


What is a branch?



The branches (in red) are pointers to commits

What is a branch?

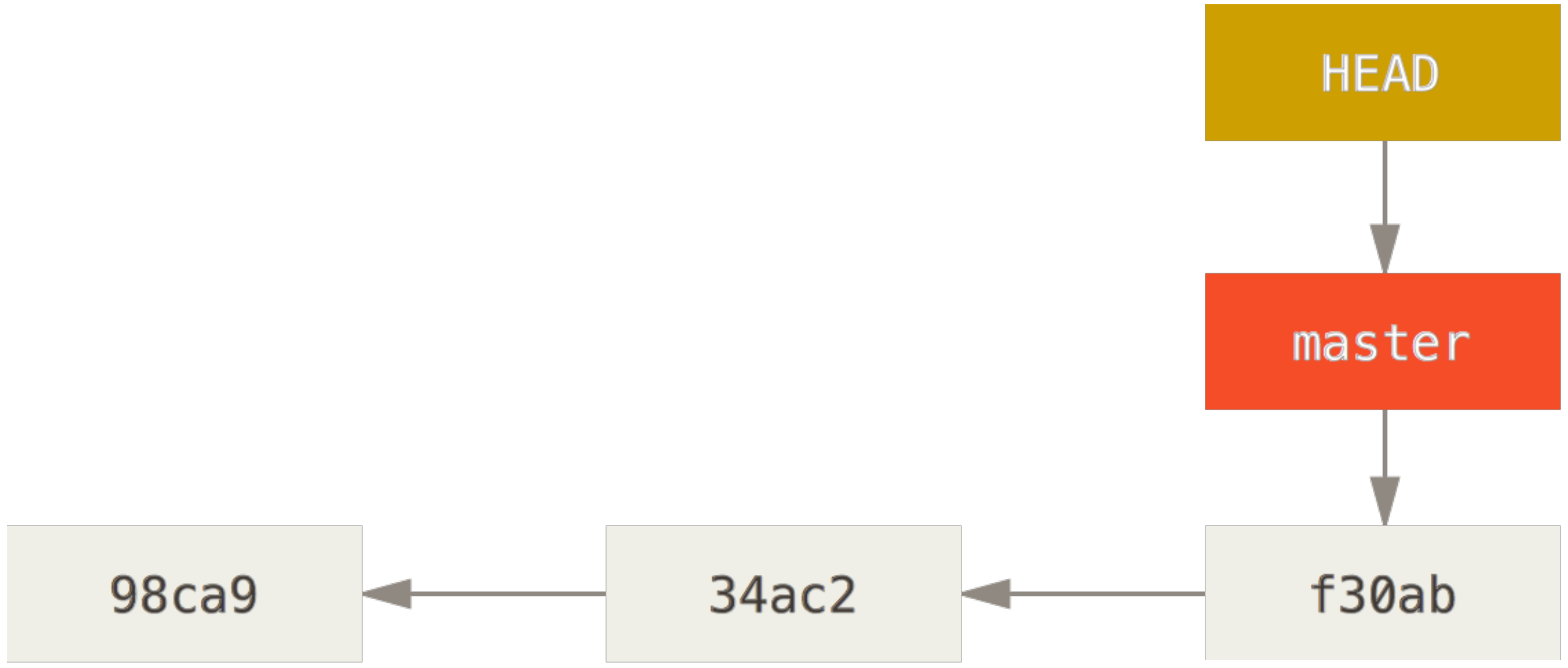


HEAD is a special pointer to your current branch stored by git

Basic git branch commands

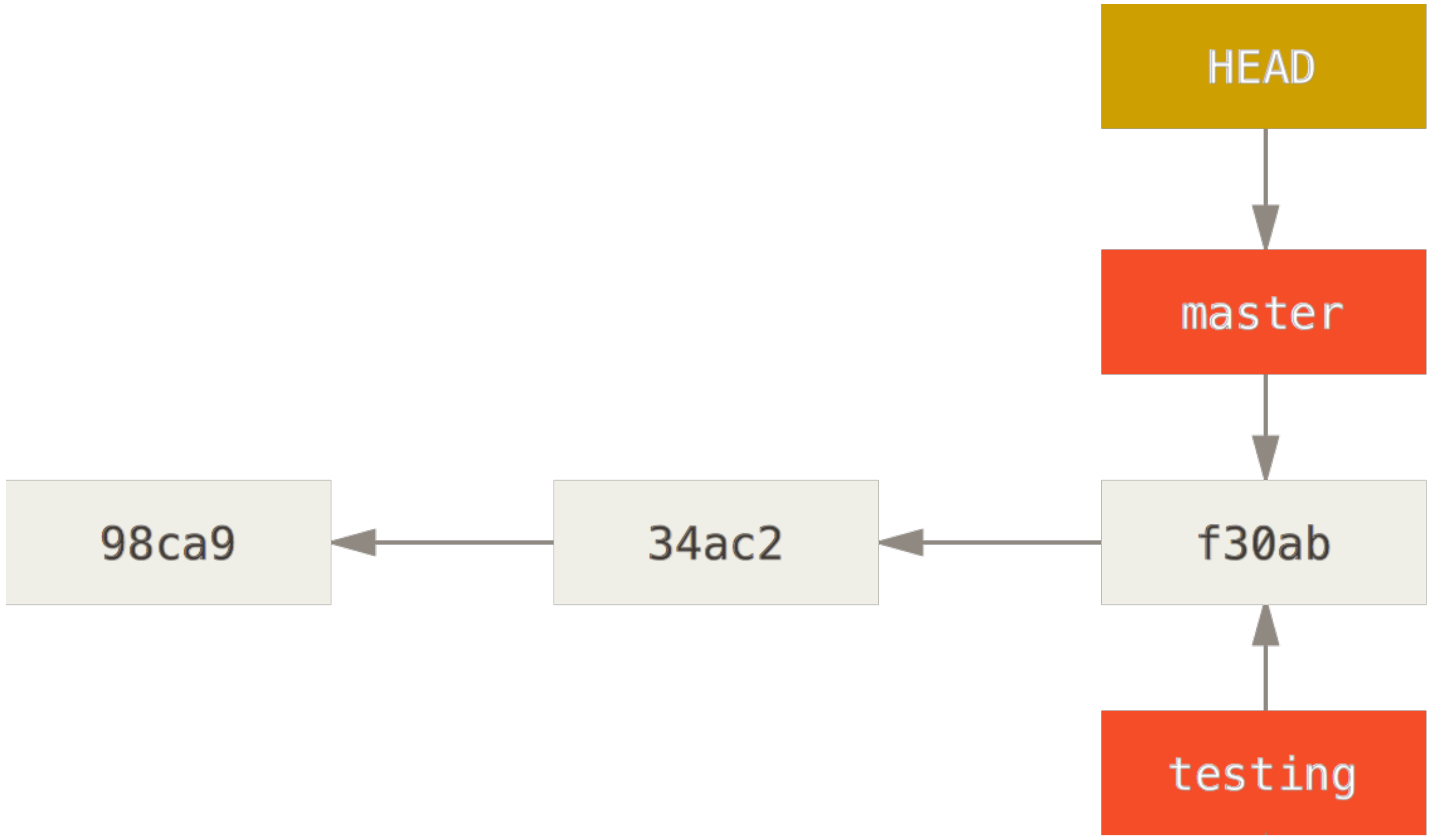
- git branch
  - lists branches
- git branch <name>
  - Makes a new branch (doesn't change HEAD)
  - Points to the commit you're on
- git checkout <branch name>
  - Change HEAD to <branch name>
- git checkout -b <branch name> [<branch/commit to base new branch on>]
  - Equivalent to git branch <branch name> then git checkout <branch name>
- git branch -d <branch name>
  - deletes <branch name>

Basic example of branches



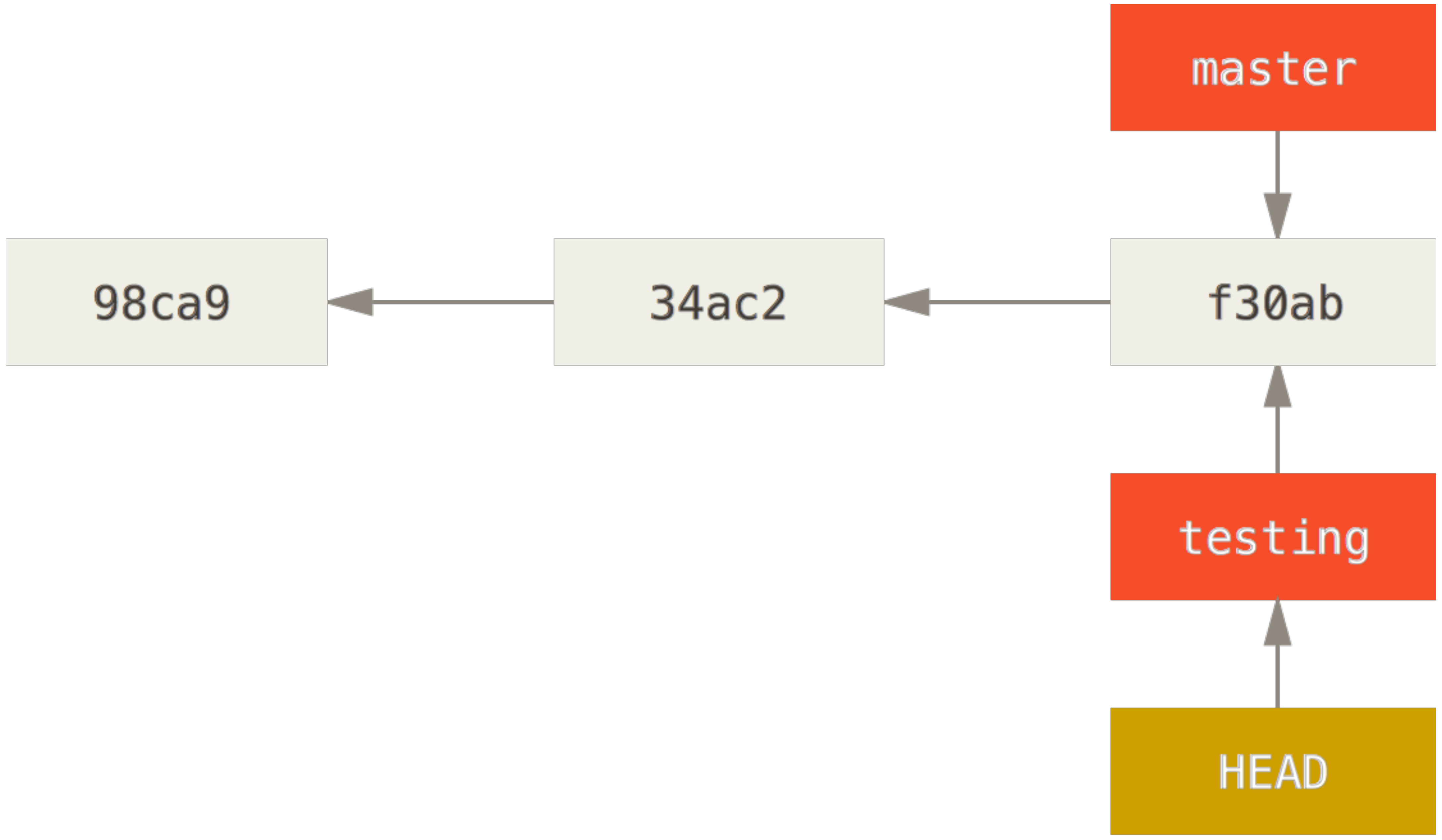
Start with HEAD at master (note - this branch is not special just default name)

Basic example of branches



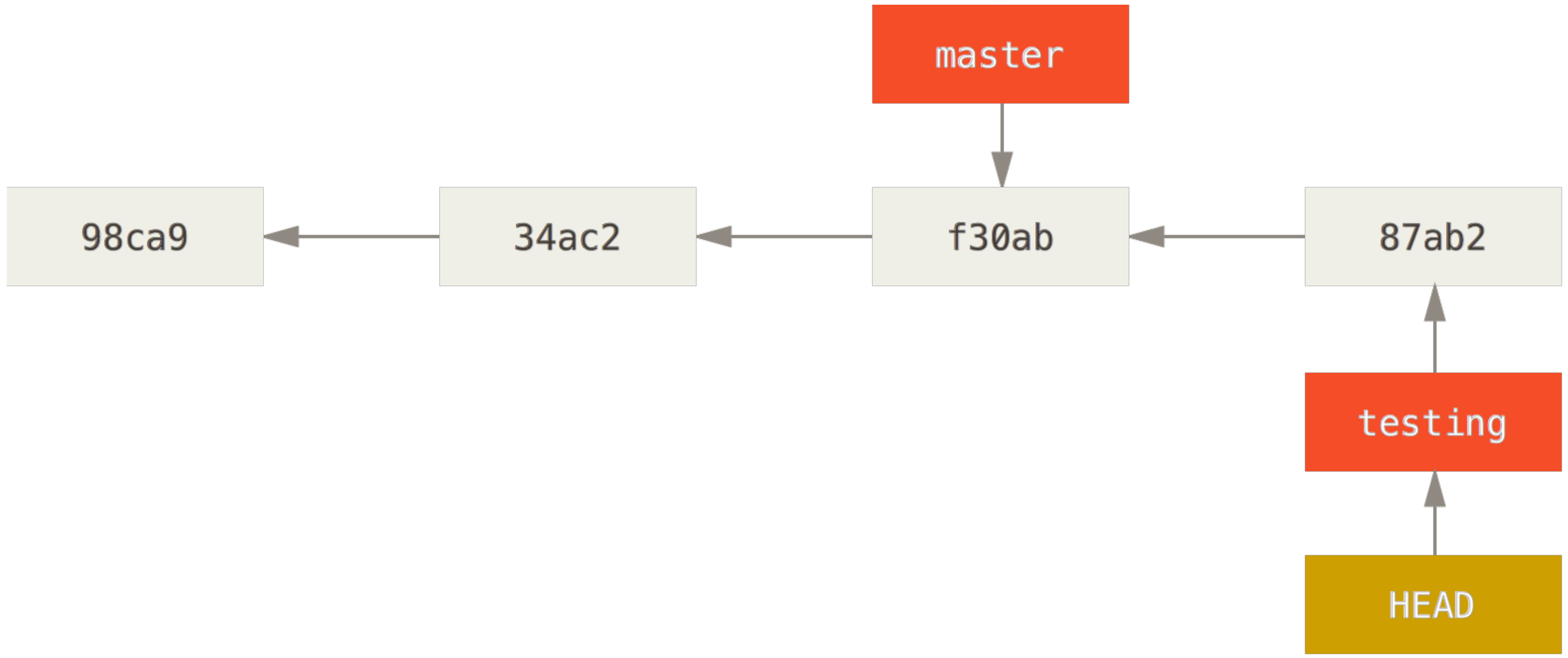
git branch testing - make testing branch

Basic example of branches



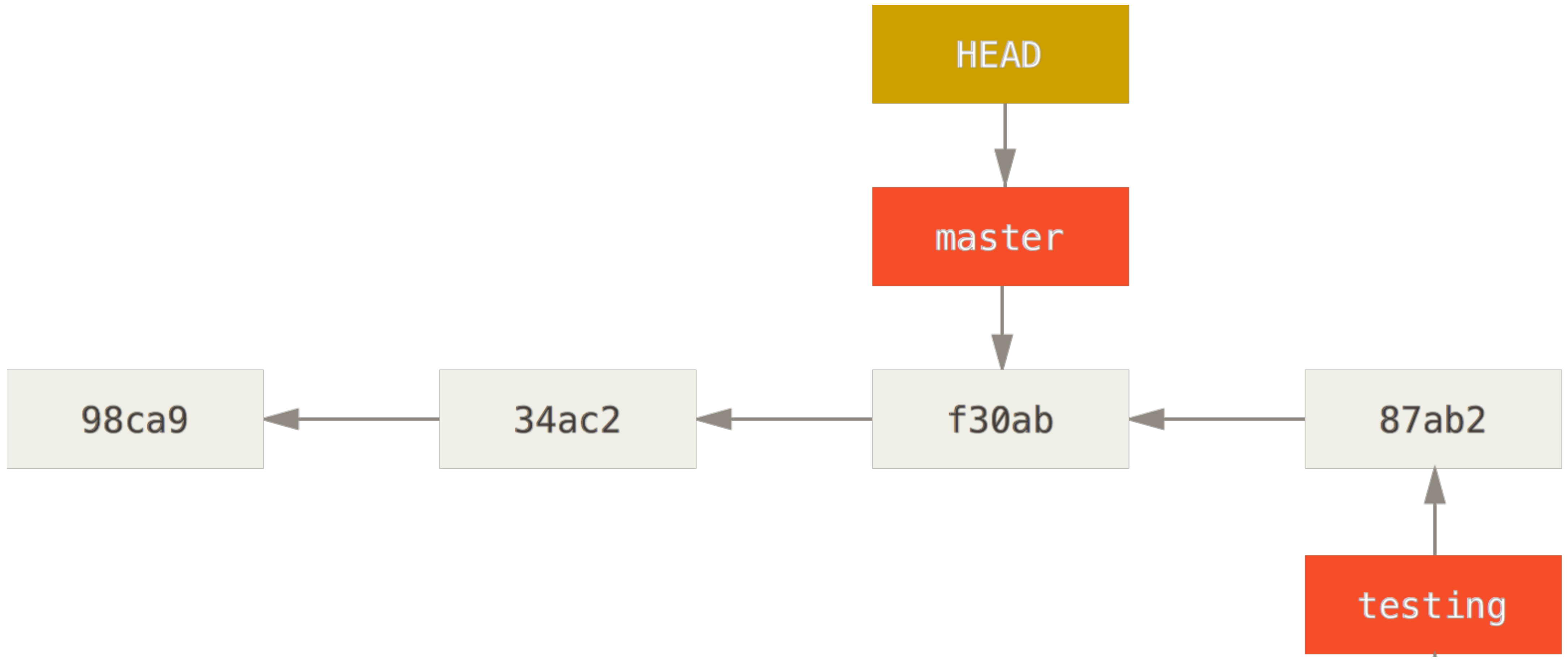
git checkout testing - moves HEAD to testing

Basic example of branches



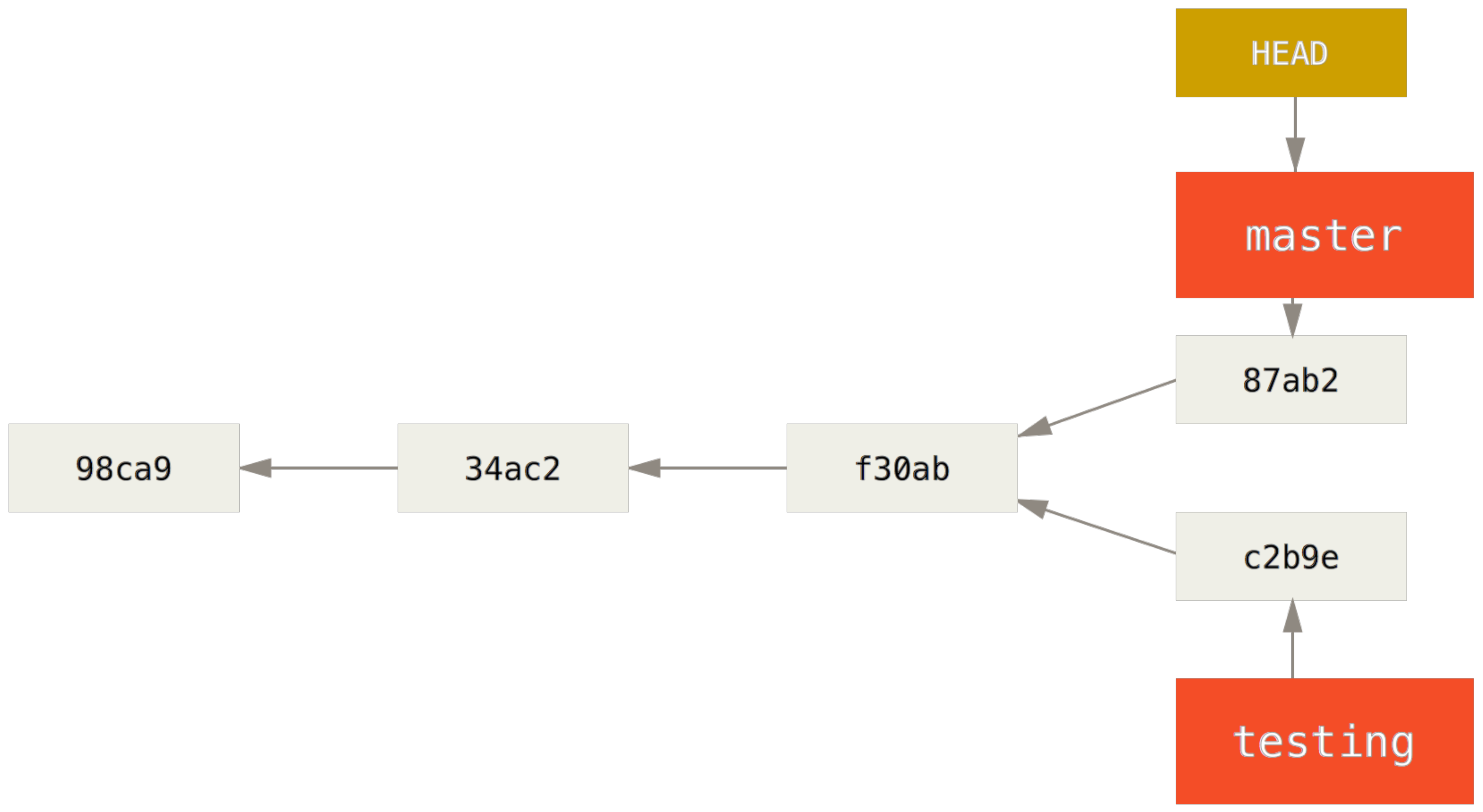
Make commit - automatically moves HEAD branch to this commit

Basic example of branches



Can swap back to master (git checkout master) - changes made in previous commit rewinded

Basic example of branches

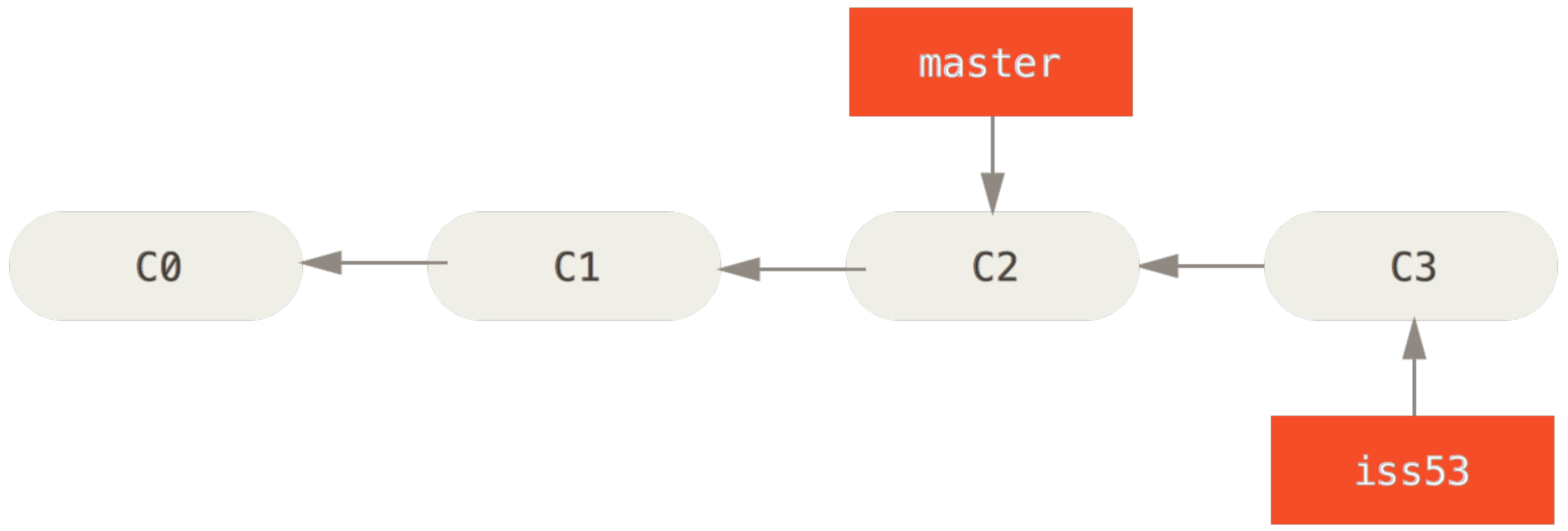


Commit again - branches have diverged (but it's possible to merge changes)

Merging

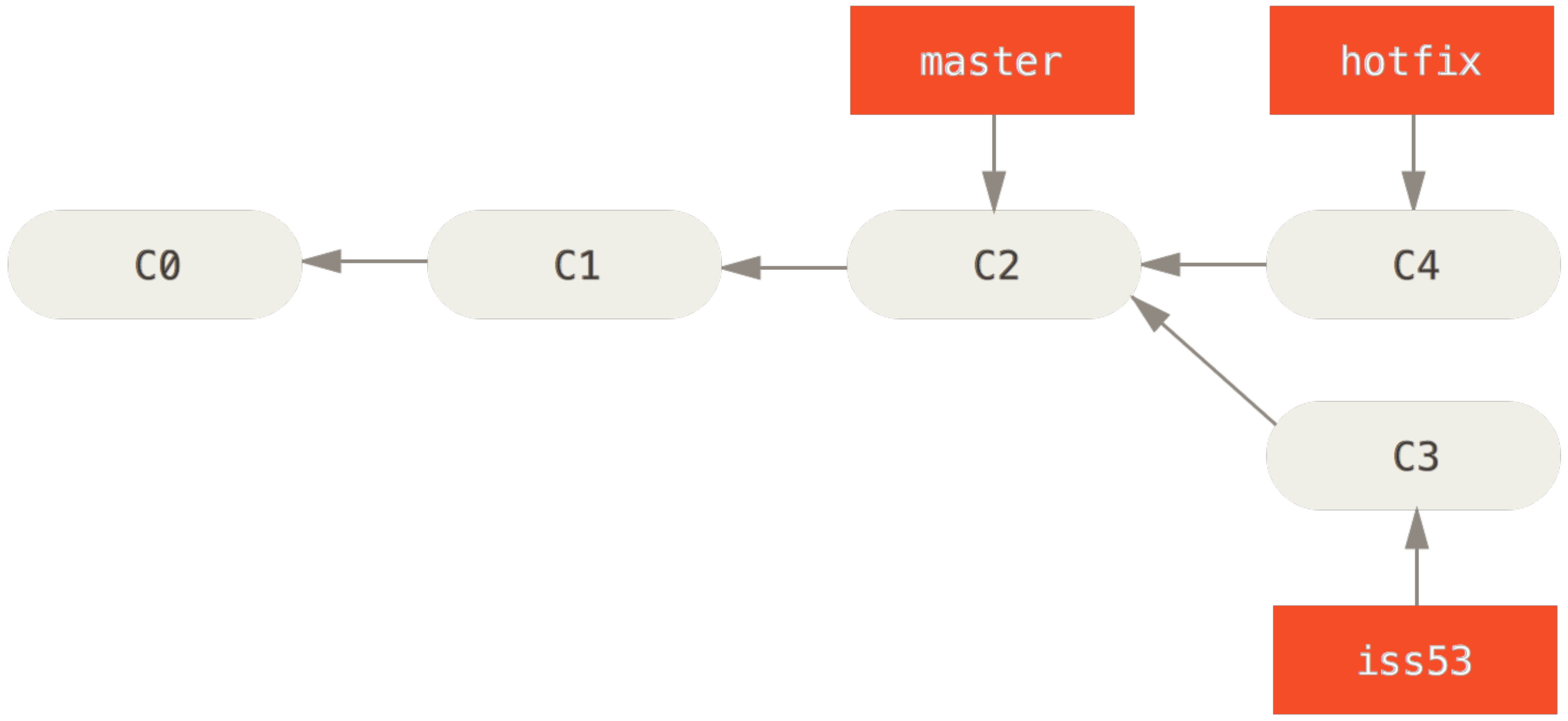
- The advantages of using branches clearest here
- Can use different branches to make experimental changes and merge when desired
- Necessary for collaborative projects (see later)
- git merge <branch name>
  - Merges <branch name> into HEAD

Merge example



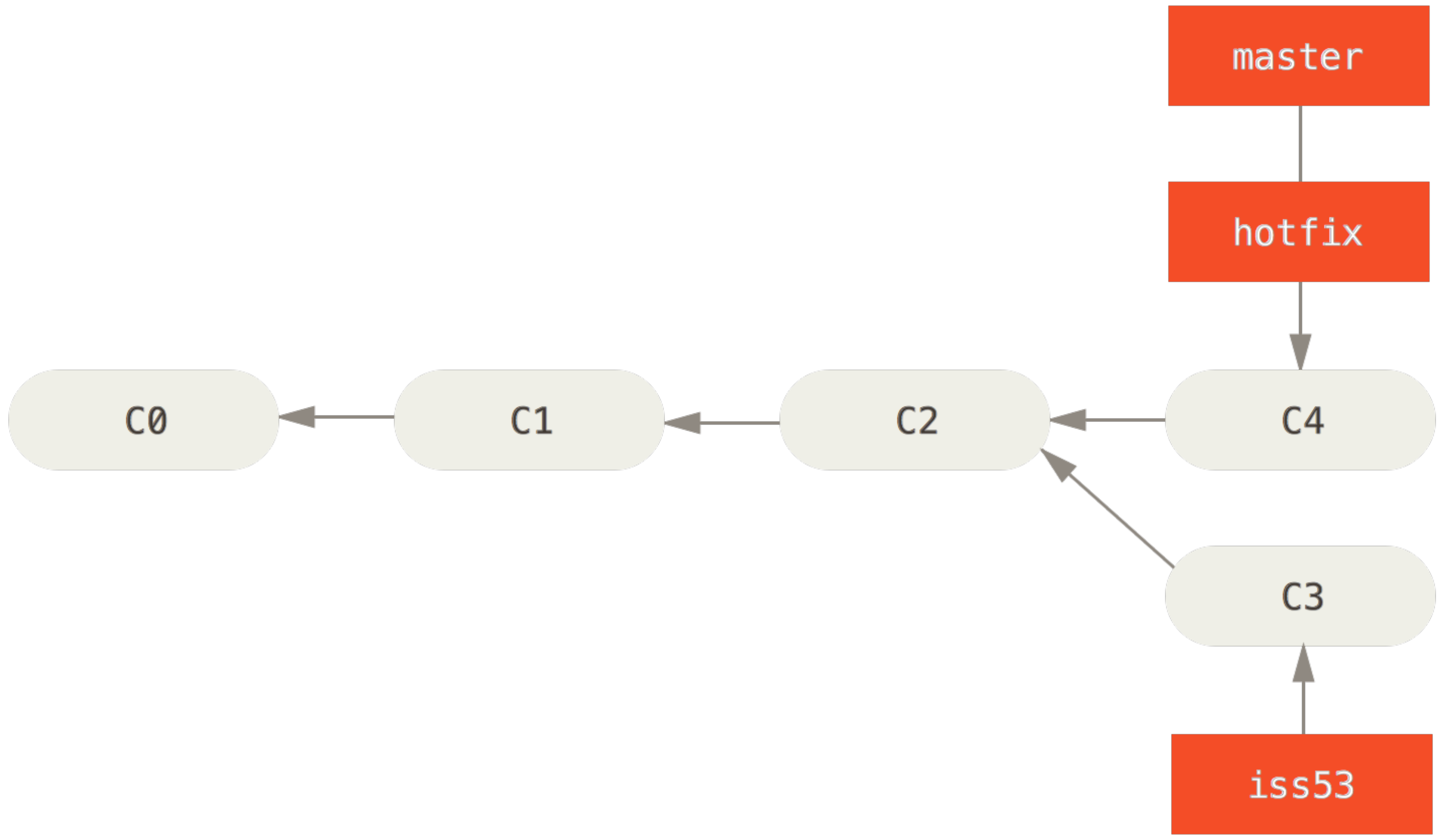
Consider situation where you're working on a development (iss53)

Merge example



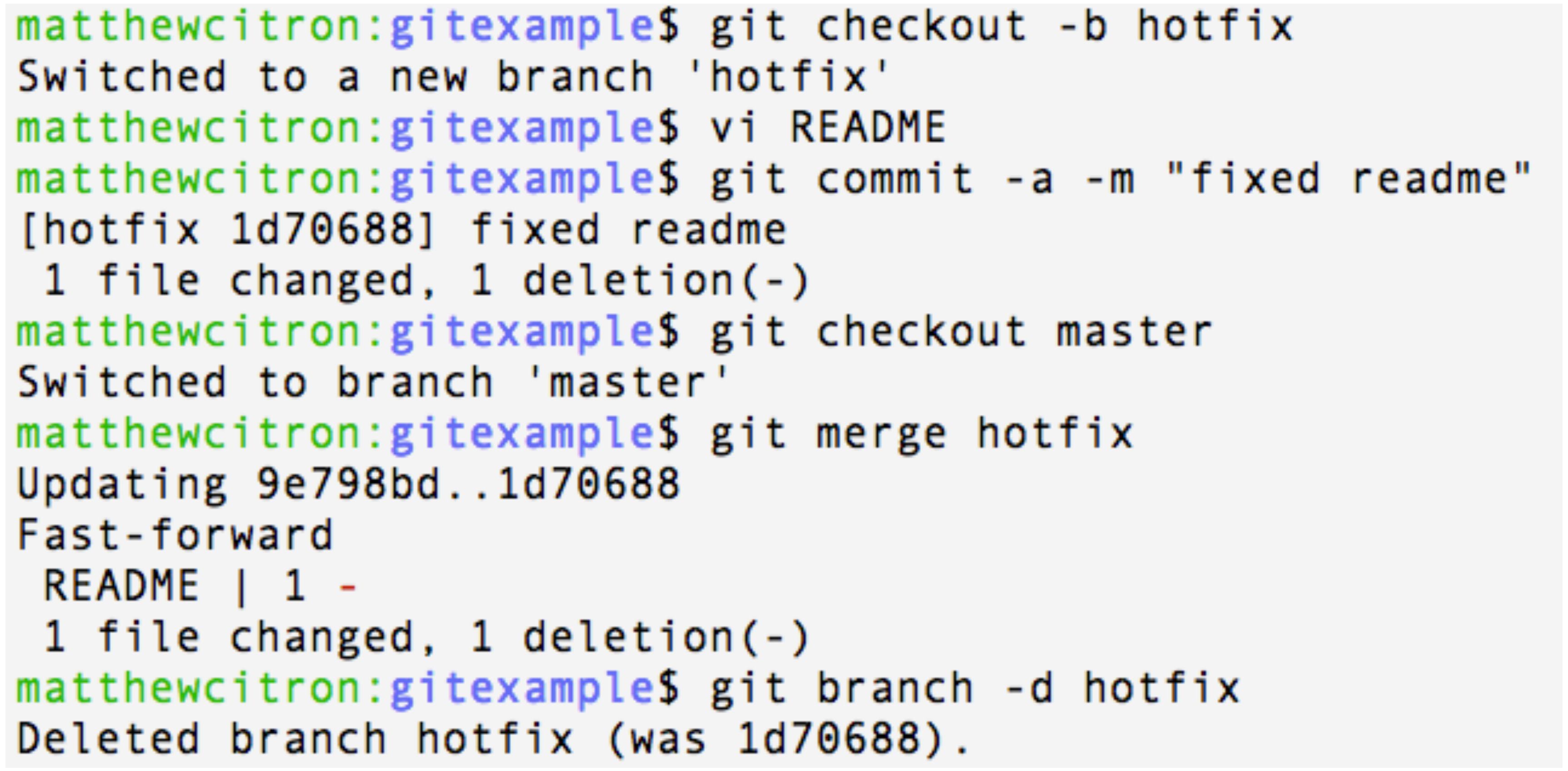
Find bug affecting master and decide to make hotfix branch to fix (from master position)

Merge example



Need to update master - checkout master and run git merge hotfix

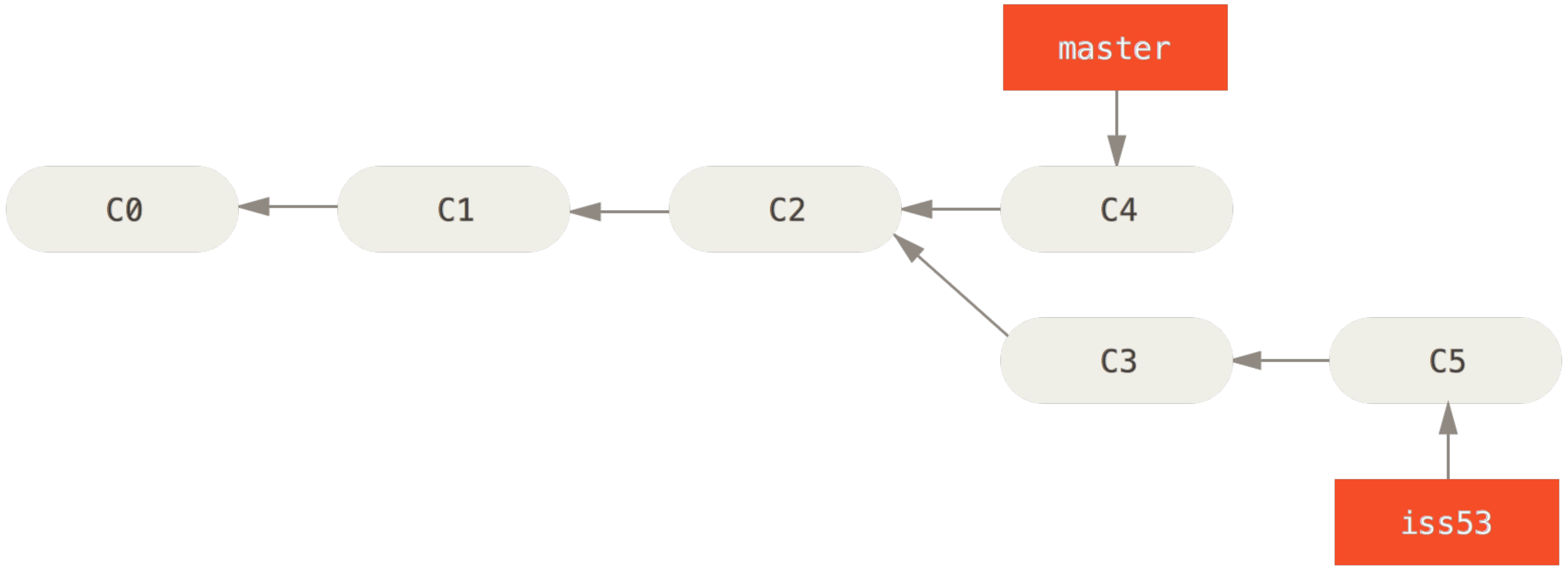
Merge example - command line



```
matthewcitron:gitexample$ git checkout -b hotfix
Switched to a new branch 'hotfix'
matthewcitron:gitexample$ vi README
matthewcitron:gitexample$ git commit -a -m "fixed readme"
[hotfix 1d70688] fixed readme
 1 file changed, 1 deletion(-)
matthewcitron:gitexample$ git checkout master
Switched to branch 'master'
matthewcitron:gitexample$ git merge hotfix
Updating 9e798bd..1d70688
Fast-forward
 README | 1 -
 1 file changed, 1 deletion(-)
matthewcitron:gitexample$ git branch -d hotfix
Deleted branch hotfix (was 1d70688).
```

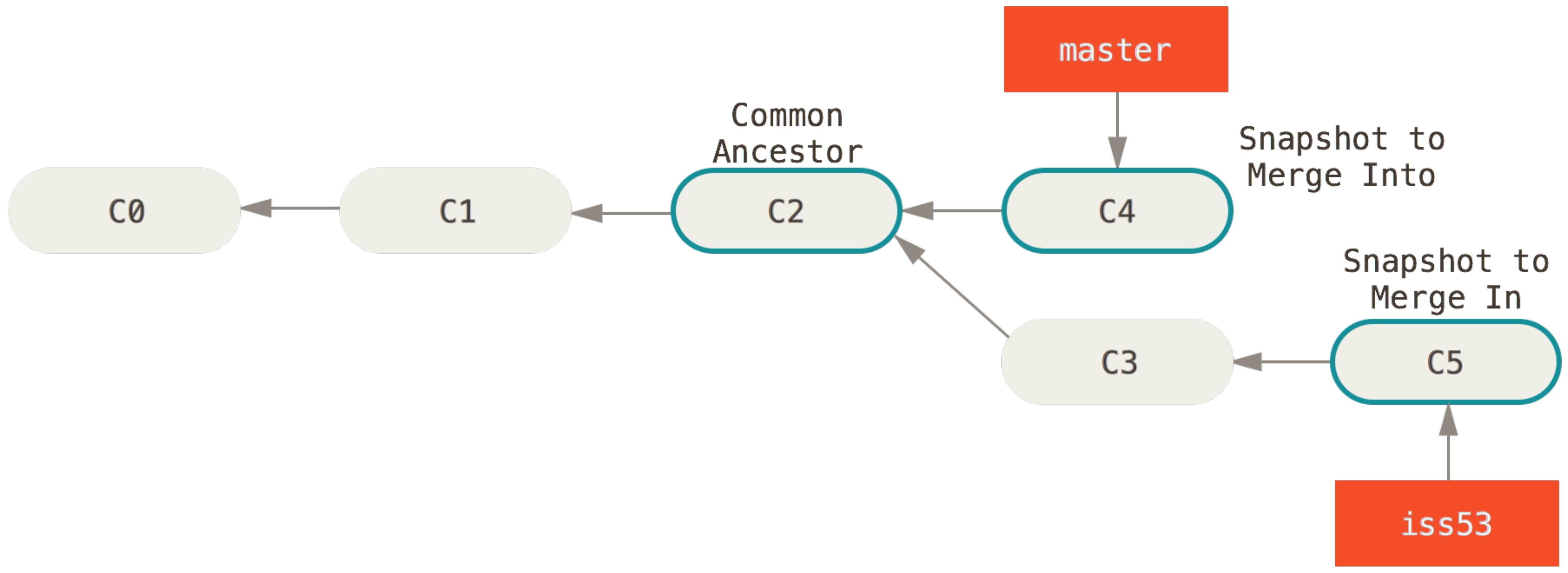
Note: Fast-forward - history is not divergent just need to add changes

Merge example



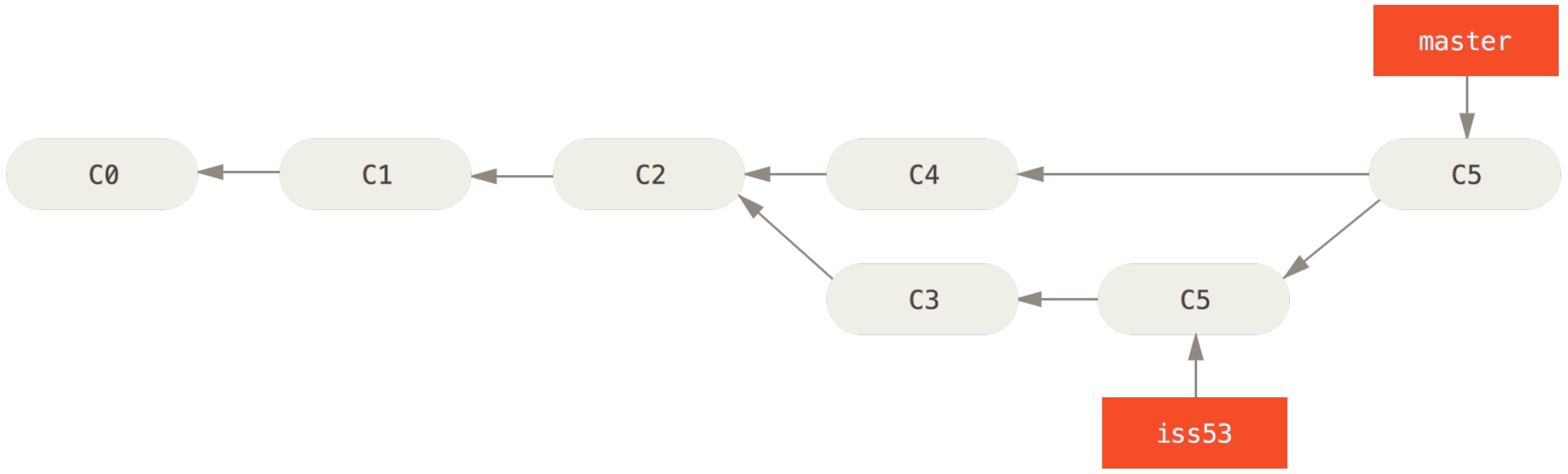
Then go back to iss53 and complete development (new commit). Want to add these changes to master but history has diverged.

Merge example



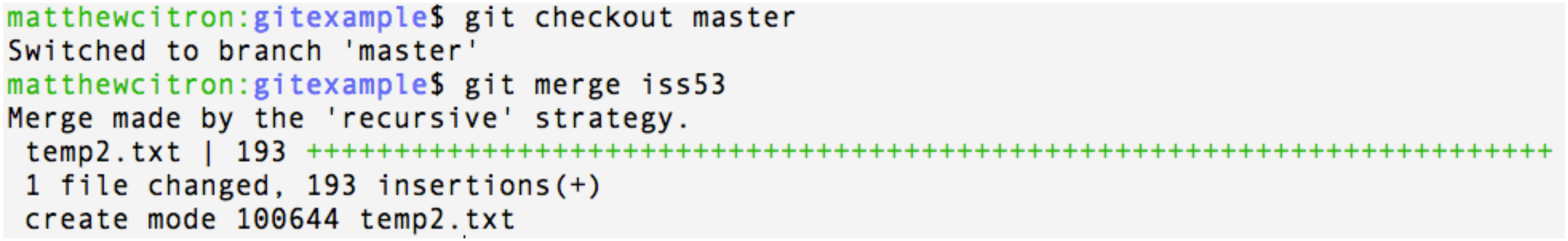
Need to checkout master and run git merge iss53. This uses common ancestor as well as current snapshots to merge branches.

Merge example



Run git merge iss53 from master.
Special merge commit (which master now points to) has two parents
See http://bit.ly/1tX5Grs for more complex example

Merge example - command line

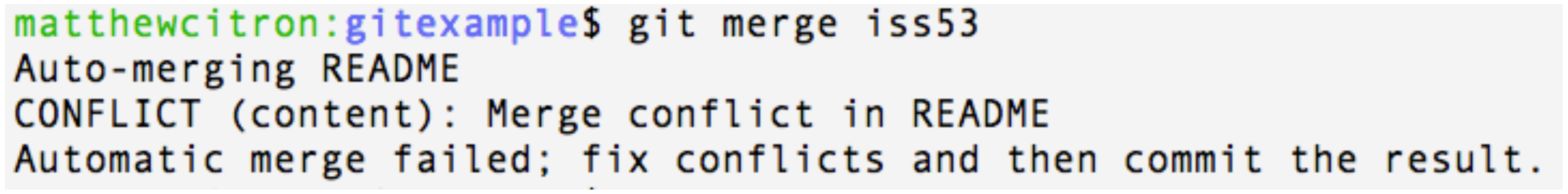


```
matthewcitron:gitexample$ git checkout master
Switched to branch 'master'
matthewcitron:gitexample$ git merge iss53
Merge made by the 'recursive' strategy.
 temp2.txt | 193 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 193 insertions(+)
 create mode 100644 temp2.txt
```

- Merge changes from iss53 to master (can choose merge strategy)
- Merge based on branch commits as well as common ancestor
- Git works out best common ancestor to use when merging
- May get merge conflict

Merge conflict

If git cannot automatically merge branches there is a merge conflict

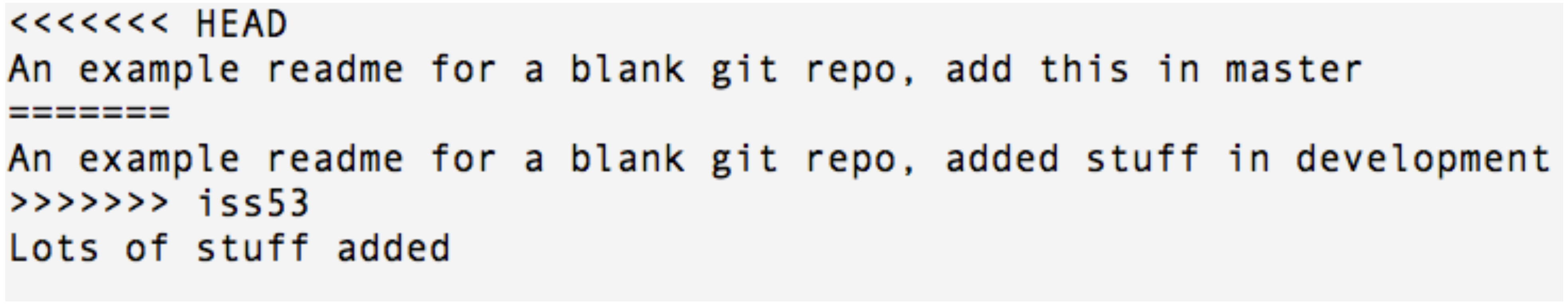


```
matthewcitron:gitexample$ git merge iss53
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then commit the result.
```

Git puts both versions in file and must fix manually

Current branch
(the branch merging into)

Merge branch
(the branch merging from)



```
<<<<<<< HEAD
An example readme for a blank git repo, add this in master
=======
An example readme for a blank git repo, added stuff in development
>>>>>>> iss53
Lots of stuff added
```

Once merge conflict fixed must commit again

Summary

- Branches are very cheap and useful tools in git
  - Not unusual to make and delete several branches per day
- Can have long running branches (like master) which will be used throughout project
- Also make topic branches to test ideas/developments before merging into long running branches
- Can make separate developments off common branch and merge
- Still only considered local git repo!

Remote Repositories

Remotes

- Remote repositories allow collaboration on projects
- Can merge local changes with those made on the remote (pushing and pulling)
- git clone adds remote as ‘origin’, however, this remote is not special.

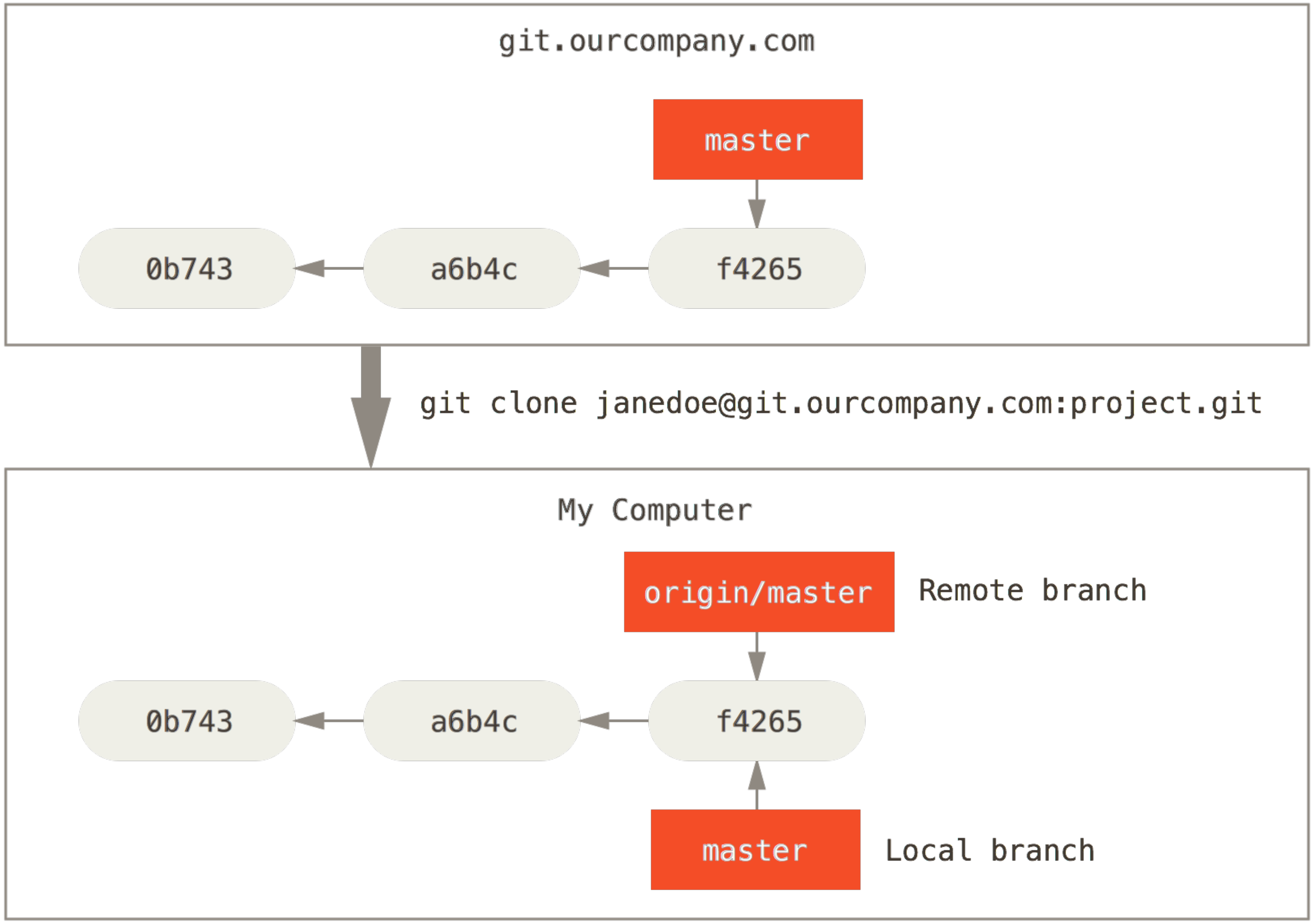
Basic commands for managing remotes

- git remote
    - Lists remotes (-v verbose)
- git remote add <(any) name> <url>
    - Adds new remote as <short name> (Make name useful!!!)
- git remote show <name>
    - Inspect remote
- git remote rm <name>
    - Remove remote
- git fetch <name>
    - fetch remote branches

Remote branches

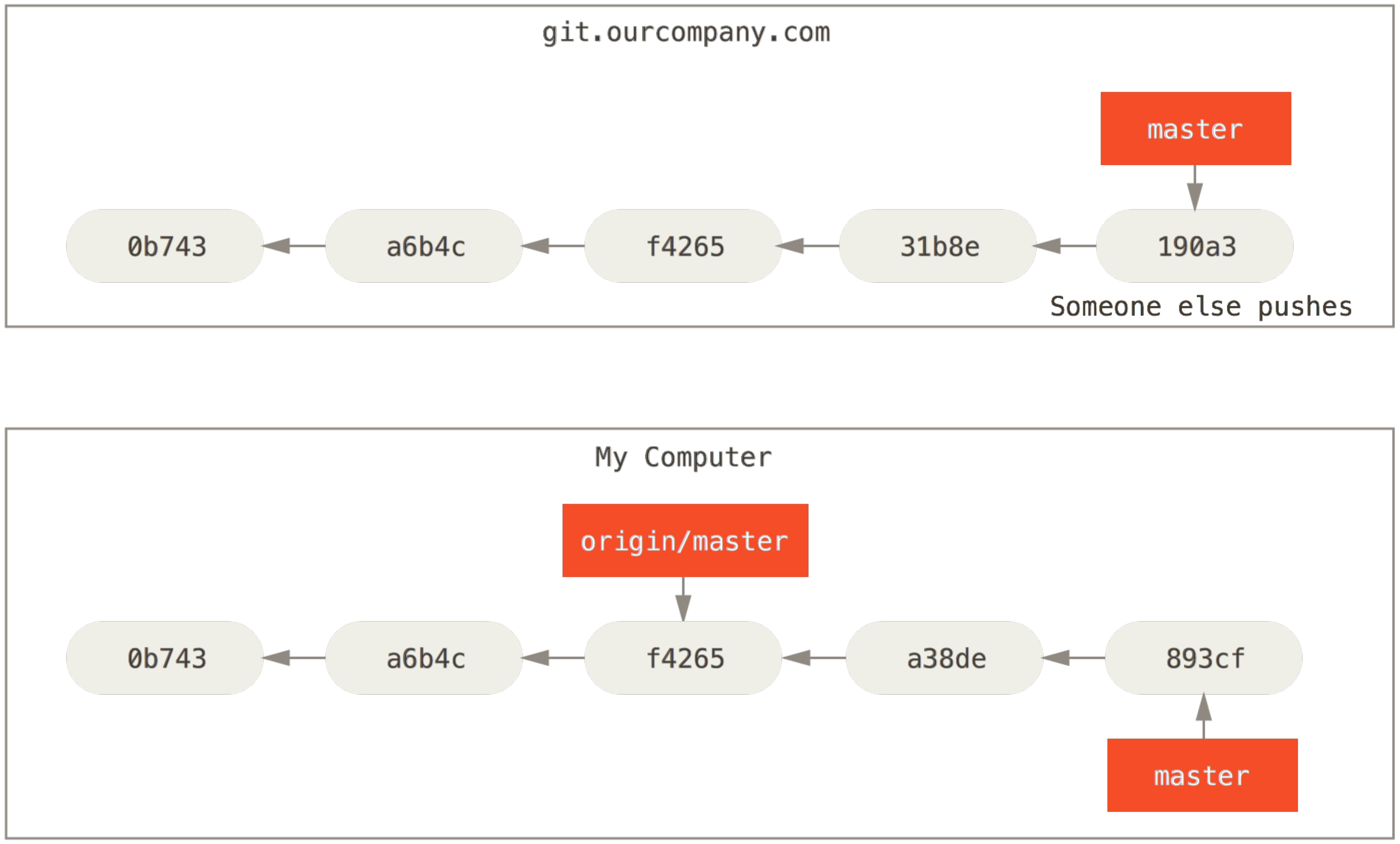
- Git fetch gets remote branches in the form (remote)/(branch)
- These are then local branches that cannot move (i.e. constant pointer) unless updated by later git fetch
- In other ways can be treated as normal branch - i.e. can make normal branch based on remote as well as merge changes from remote branch

Remote branch example



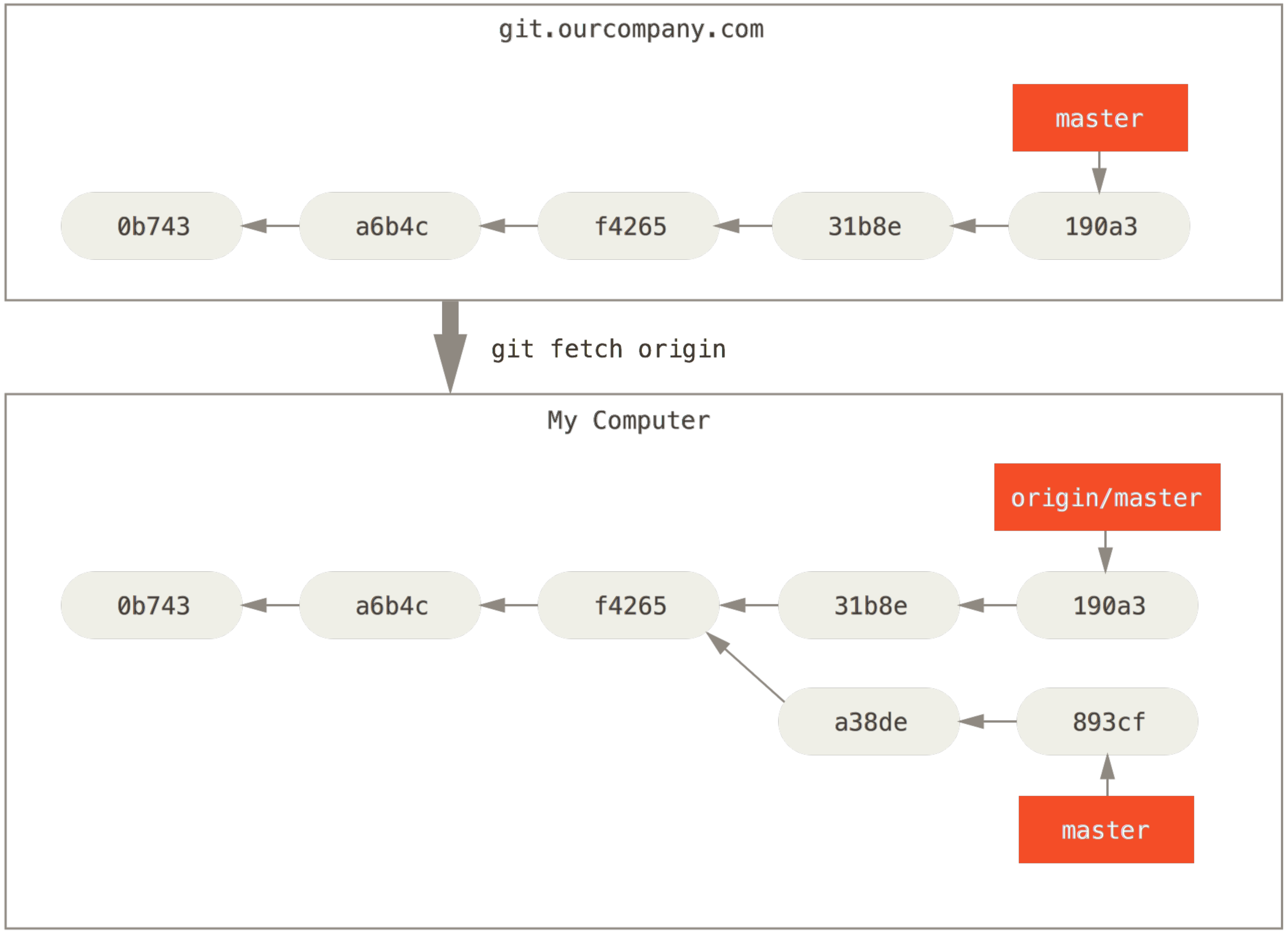
On running git clone both origin/master and master point to same commit

Remote branch example



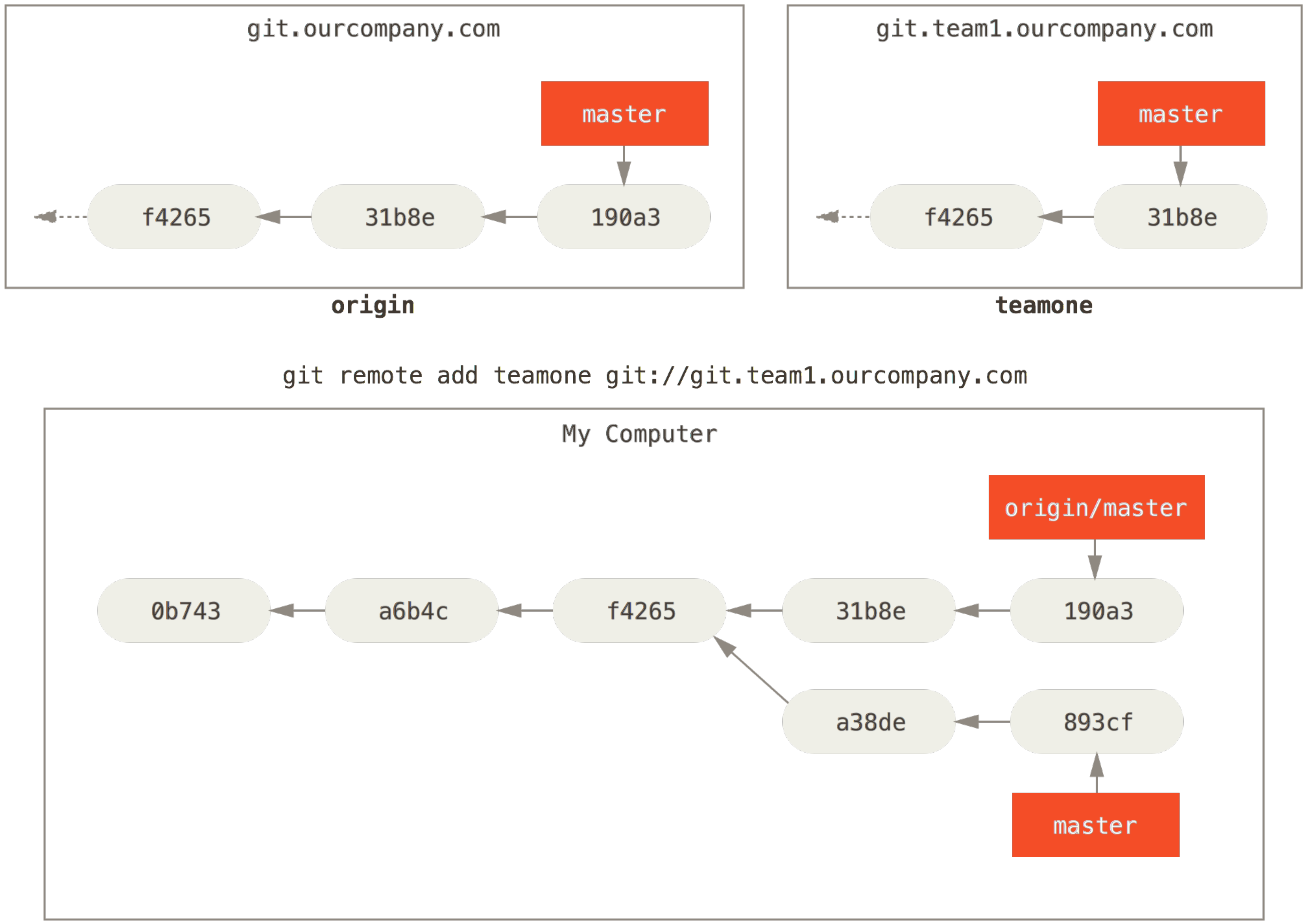
Make changes locally and someone updates the remote - origin/master stays at old position until call git fetch

Remote branch example



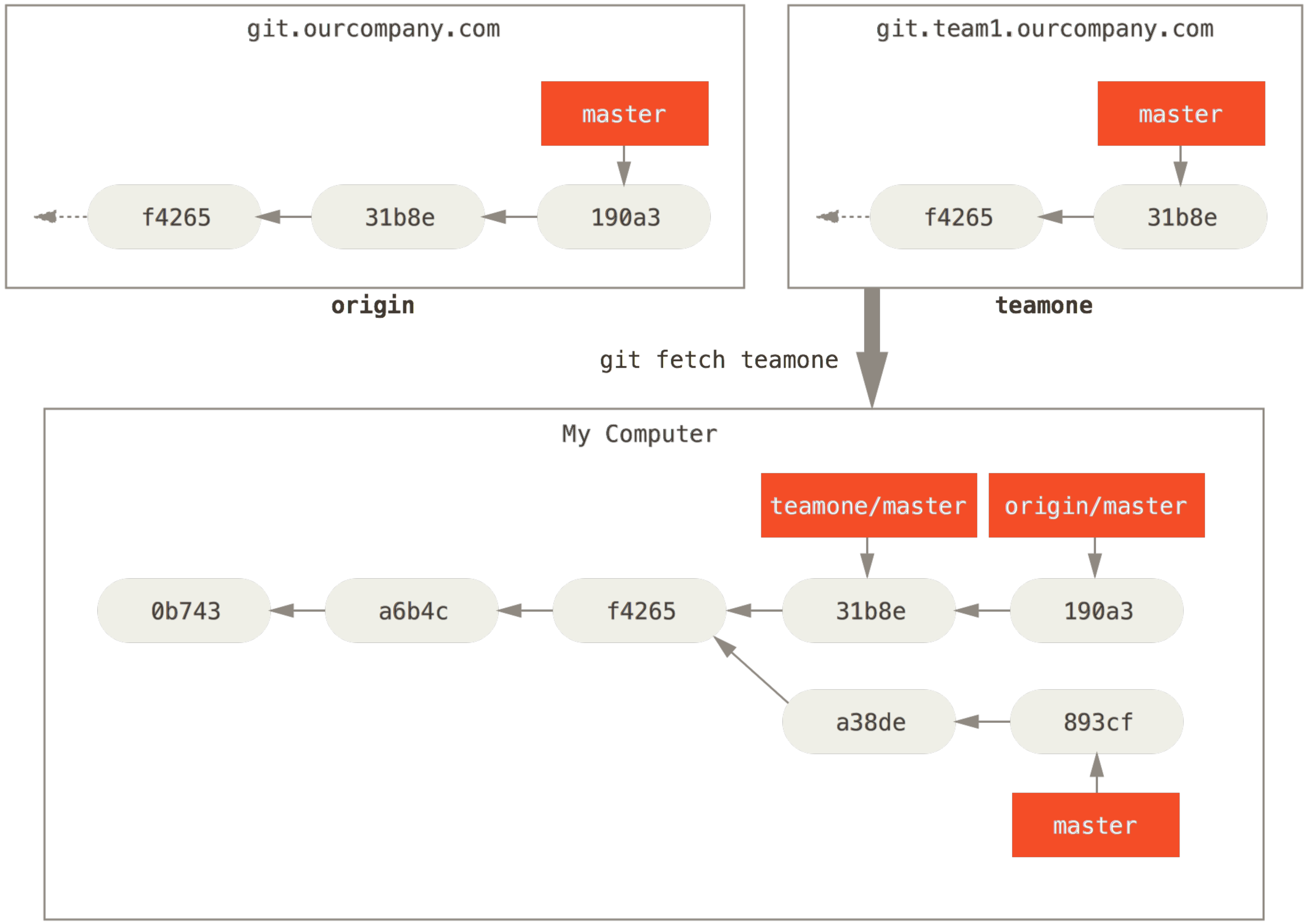
git fetch updates origin/master - can now merge changes if desired

Remote branch example



Can add multiple remotes as desired

Remote branch example



git fetch teamone will then add their remote branch

Basic commands for remote branches

- git pull <remote repo> <remote branch>
  - Merges changes from remote repo into current branch
  - Equivalent to git fetch <remote name> then git merge <branch name>
  - This can be confusing so may be better to avoid initially
- git push <remote repo> <local branch>:<remote branch>
  - Merges local changes with server branch
  - If branches have same name can use: git push <remote name> <local branch>
  - If someone has already updated remote branch must merge their changes into local first
- git push <remote repo> --delete <remote branch>
  - Delete remote branch

Summary and notes

- Interacting with remotes simple extension of local working.
- Very common to have central project remote (e.g. CMSSW) as well as personal fork of project remote.
- Give remote repo useful name!
- Never change the history of something that is public
- Many different workflows for collaborating with remotes
- Github is biggest host of remote repos

Rebasing

Rebasing

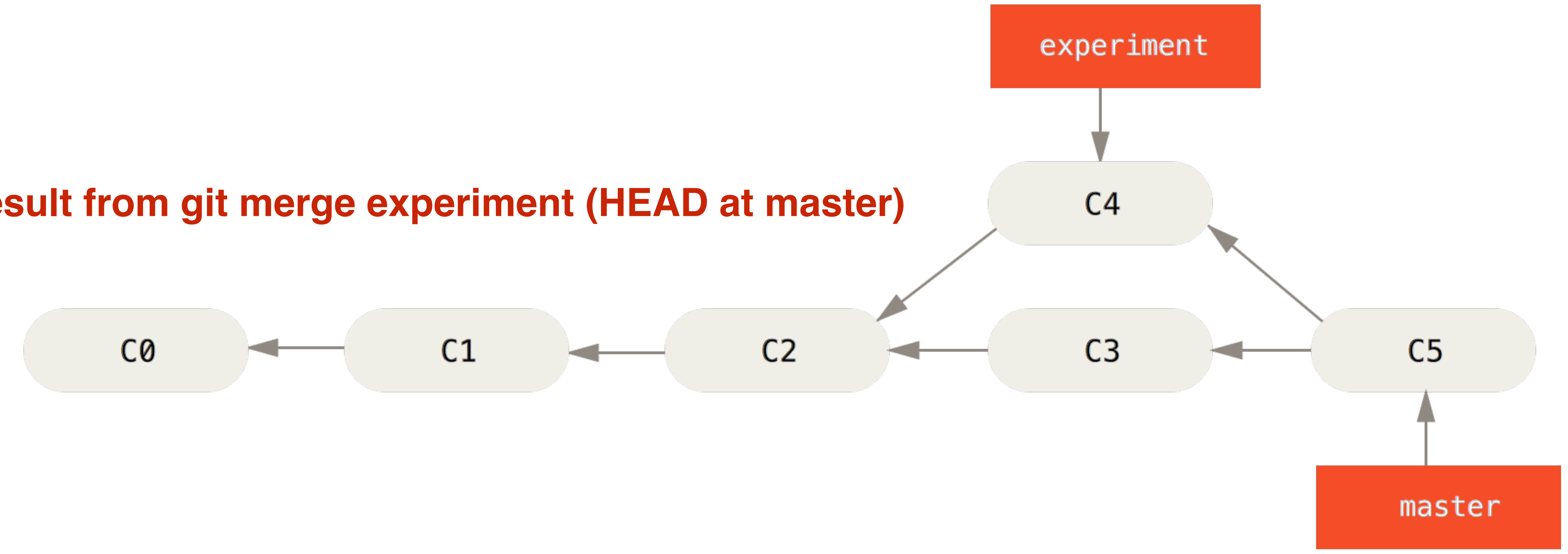
- The rebase is an alternative to the merge which provides a cleaner, more linear history
- The commit that results has exactly the same content as it would from merge
- Controversial as resultant history is technically lies
- NEVER rebase commits which have already been pushed to the remote

Basic command for rebasing

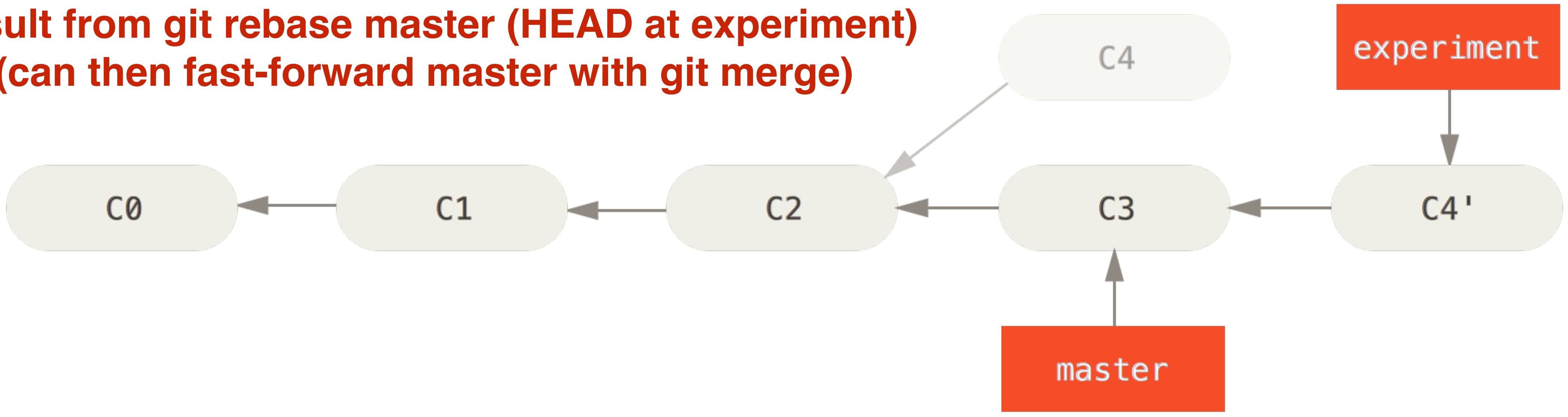
- git rebase <branch name>
    - First changes to the common ancestors of the two branches
    - Finds the diffs made by the branch you're on
    - Resets the branch to the branch you're rebasing onto
    - Applies the diffs to that branch
- See http://bit.ly/1smHkM7 and http://githowto.com/rebasing for more info

Rebase example

Result from git merge experiment (HEAD at master)



Result from git rebase master (HEAD at experiment)
(can then fast-forward master with git merge)



Summary

- Git is a very useful tool for working in isolation or within a collaboration.
- Can be unintuitive but most ‘errors’ come from
  - Not resolving merge conflicts
  - Trying to push without pulling (-f flag will get you killed)
  - Forgetting to git add a file (especially when pushing) to remote
  - Uncommitted changes before merging, changing branch etc…
- Anything that is committed can be changed without fear of loss.
- More info look here http://git-scm.com/book/en/v2
- If collaborating on a large project often a common git workflow is used (see http://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows)
- For those who use vim - fugitive is an amazing git plugin.