CSE Git Tutorial

Adam Kosiorek, 2014
reference: http://git-scm.com/book
ppt based on: http://tinyurl.com/UW-git

Agenda

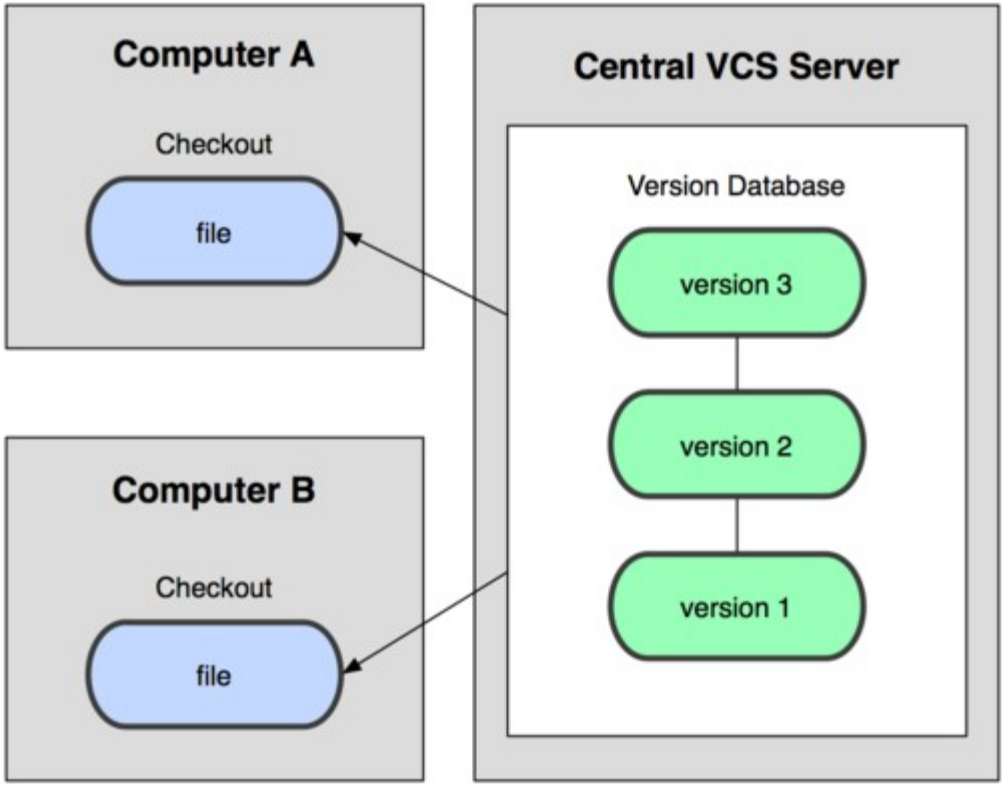
- We will:
  - Motivate version control
  - Discuss the basic Git model
  - Use command line a lot
  - Work with local repo
  - Branch, merge and resolve conflicts
  - Work with remote repo
  - Discuss Git Workflow
- We won't:
  - Discuss Git GUIs

Git History

- Came out of Linux development community
- Linus Torvalds, 2005
- Initial goals:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like Linux efficiently

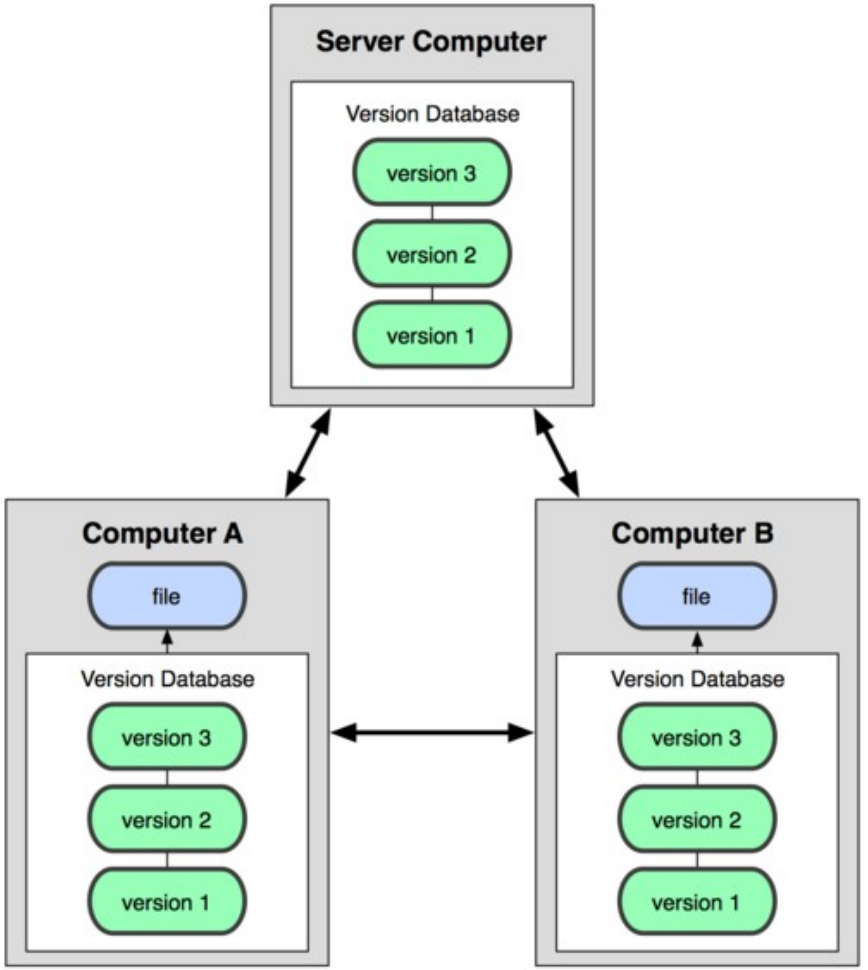
Git uses a distributed model

Centralized Model



(CVS, Subversion, Perforce)

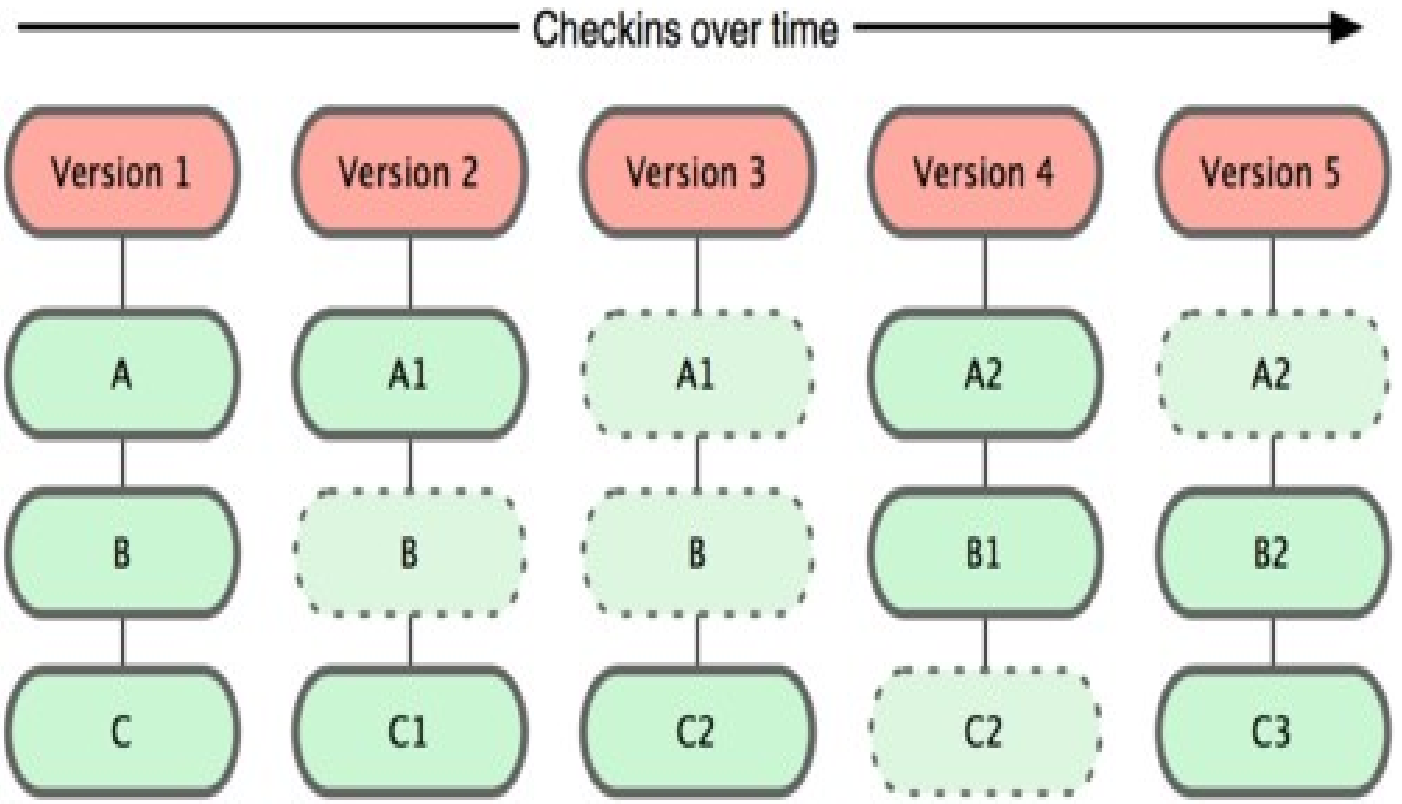
Distributed Model



(Git, Mercurial)

Result: Many operations are local

Git takes snapshots



Git uses checksums

- Commits: Unique SHA-1 hash – 40 character string
- we only see the first 7 characters:

1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit

Git Resources

- At the command line: (where verb = config, add, commit, etc.)

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

- Free on-line book: http://git-scm.com/book
- Git tutorial: http://schacon.github.com/git/gittutorial.html
- Reference page for Git: http://gitref.org/index.html
- Git website: http://git-scm.com/

- Git for Computer Scientists http://eagain.net/articles/git-for-computer-scientists/

Get ready to use Git!

- Installing Git: http://tinyurl.com/installing-git
- Set the name and email globally:

```
$ git config --global user.name “Bugs Bunny”
$ git config --global user.email bugs@gmail.com
```

- Call **`git config --list`** to verify
- Set variables on a project-only without the **`--global`** flag.
- Commit message editor:

```
$ git config --global core.editor vim
```

Create a local copy of a repo

- Two common scenarios:
  - To clone an already existing repo:

    $ git clone <url> [dir name]

  - To create a repo in your current directory:

    $ git init

    $ git add README.md

    $ git commit –m “initial commit”

    $ git remote add origin https://github.com/<user>/<repo>

    $ git push origin master

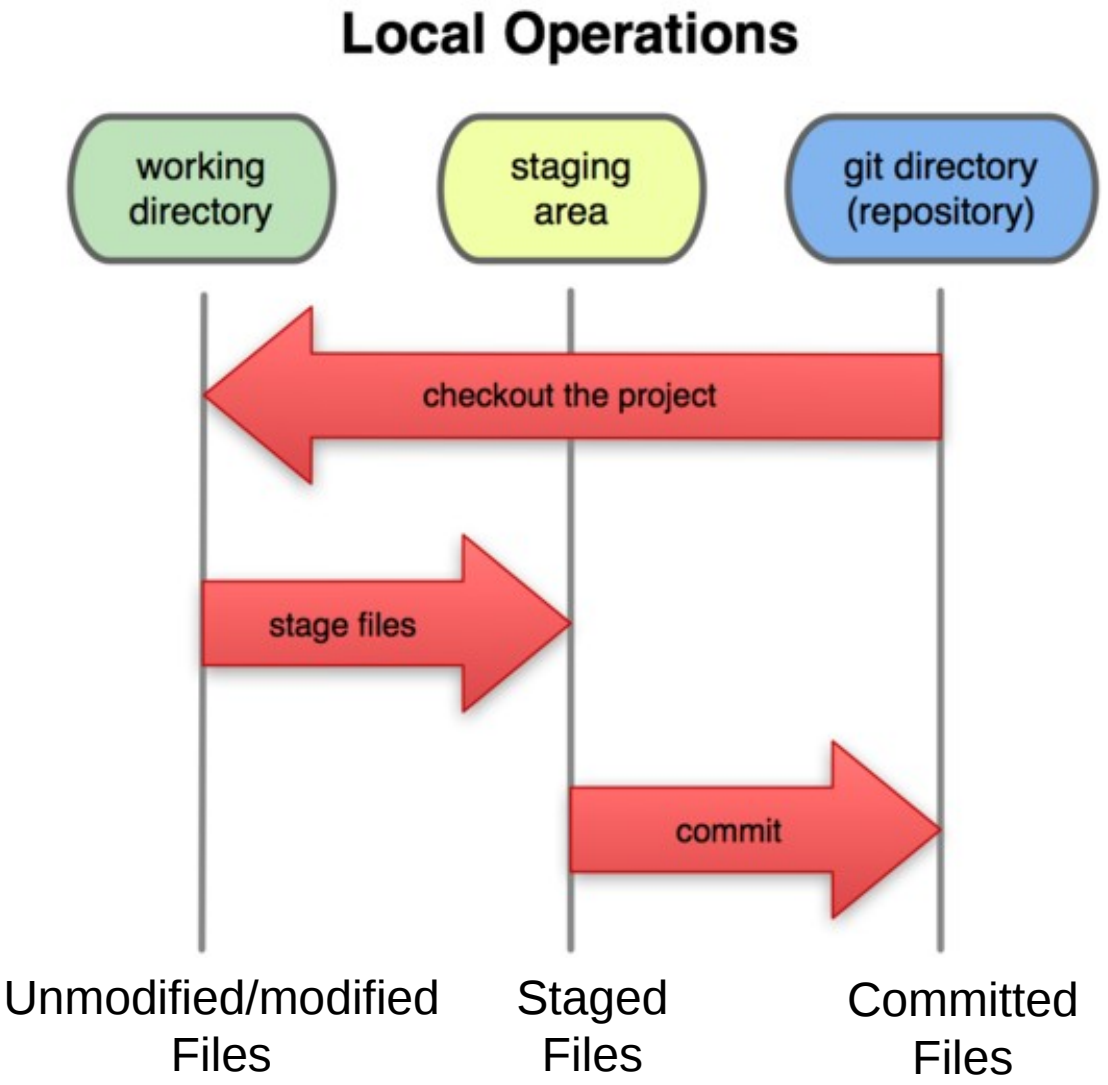
So what is github?

- On-line storage of Git repositories.
  - Continuous Integration powered by Travis-CI
- Open source projects → free
- Private projects → pay

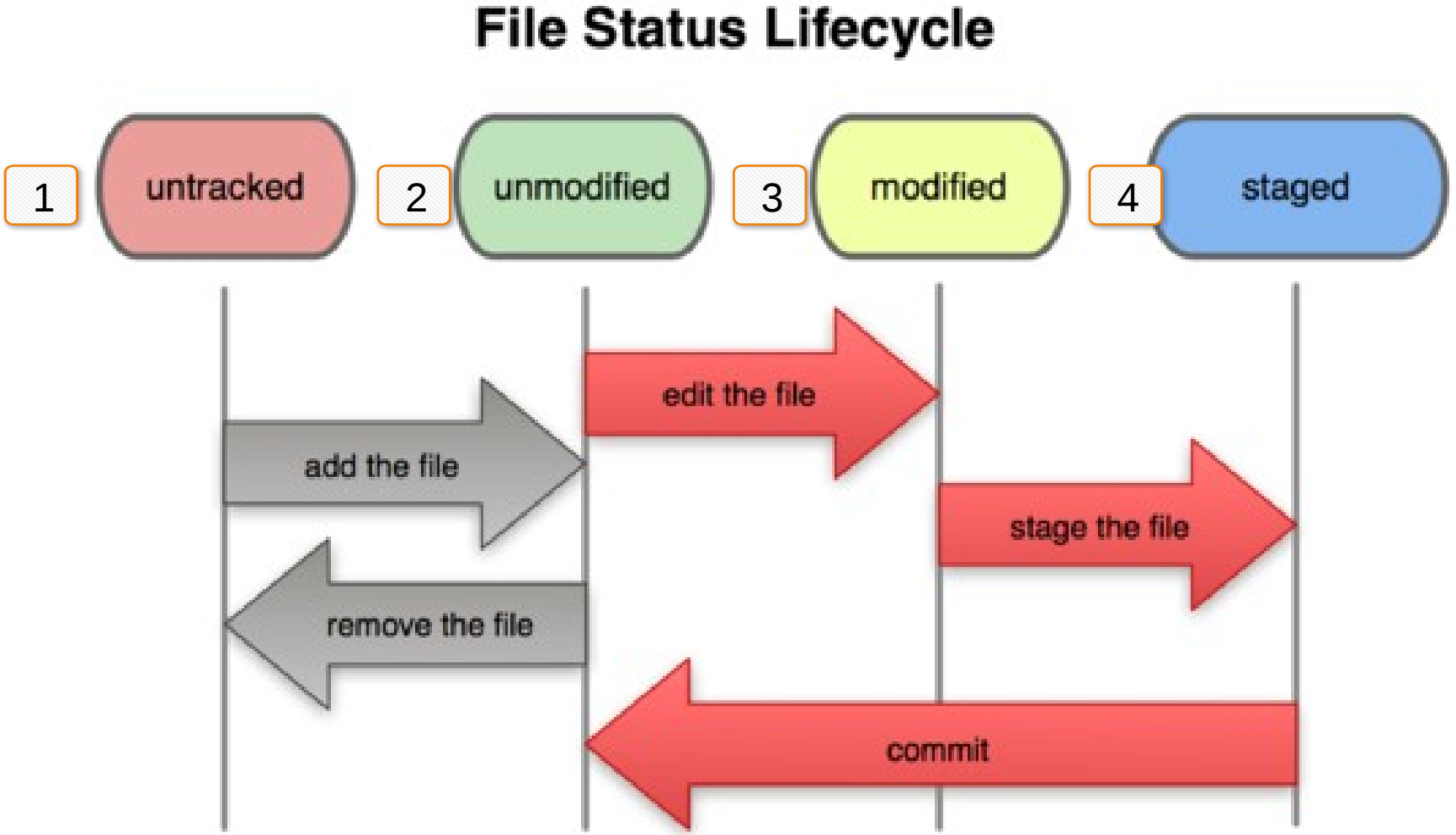
Other options:

- Local
- Different hosting

A Local Git project has three areas



Note: working directory sometimes called the “working tree”, staging area sometimes called the “index”.

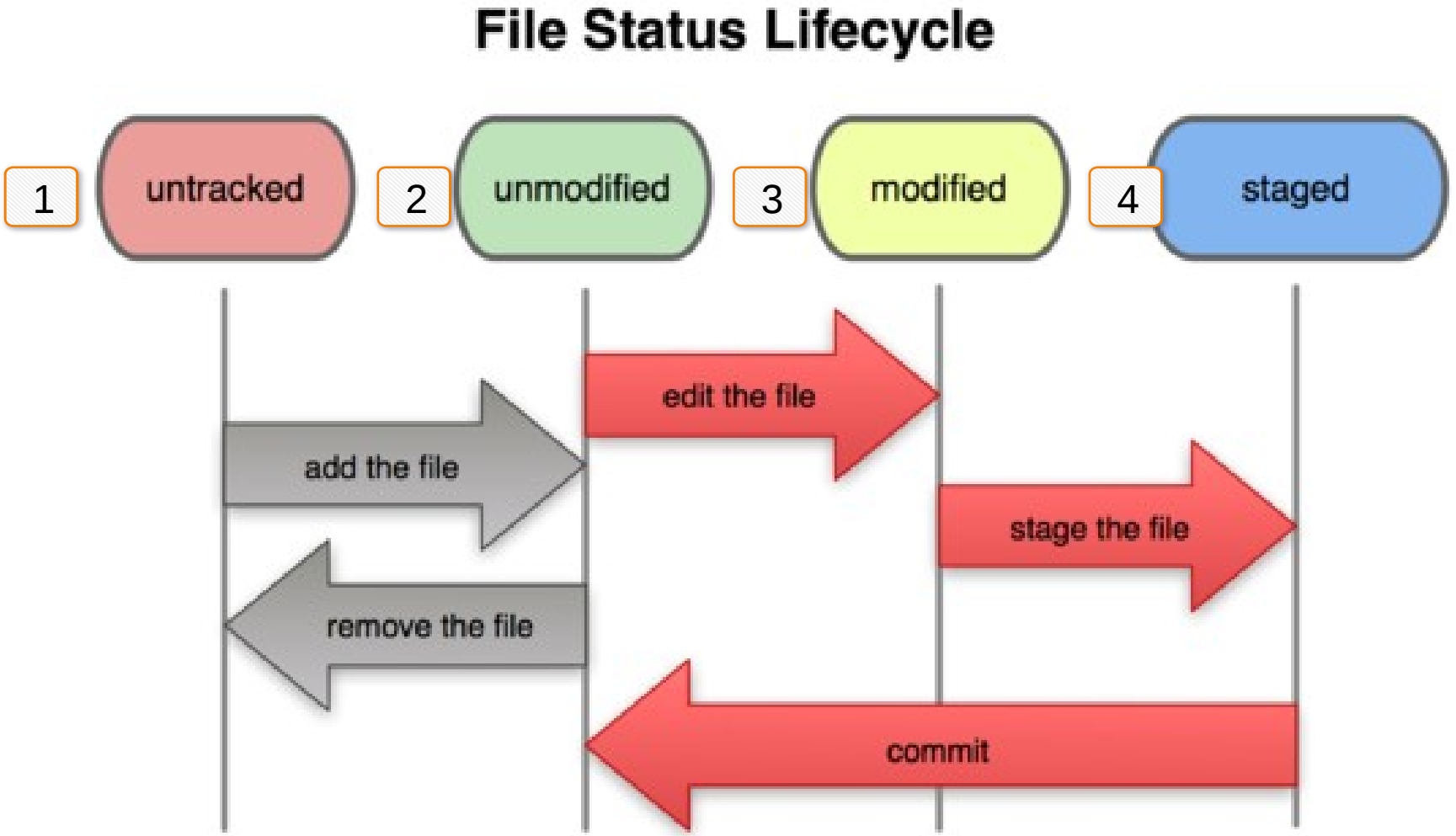


1 → 4: git add <file> (file wasn't tracked before)

4 → 3 modify a file that was already staged

3 → 4: git add <file> (file was tracked before)

4 → 2: git commit <file> (puts <file> to the repo)



2 → 3: modify a file that was already tracked (commited)

3 → 2: git checkout <file>

4 → 1: git reset HEAD <file> (<file wasn't tracked previously)

4 → 3: git reset HEAD <file> (<file> was tracked previously)

(2, 3, 4) → 1: git rm --cached <file>

Staging files

- Add to the staging area:

```
$ git add README.txt hello.java
```

- Add all untracked and modified in the current dir:

```
$ git add .
```

- Add all modified and deleted (but not new!)

```
$ git add -u (--update)
```

Commiting files

- Move into the repo:

```
$ git commit
```

- Bypass staging area:

```
$ git commit -a
```

- Short message:

```
$ git commit -m “Fixing bug #22”
```

Note: To unstage a change on a file before you have committed it:

```
$ git reset HEAD filename
```

Note: To unmodify a modified file:

```
$ git checkout HEAD filename
```

Note: These commands are just acting on **your local version of repo**.

Status and Diff

- To view the **status** of your files in the working directory and staging area:

```
$ git status
$ git status -s
```
(one line version)

- To see what is modified but unstaged:

```
$ git diff
```

- Staged changes:

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

- Changes between HEAD and a commit:

```
$ git diff <hash>
```

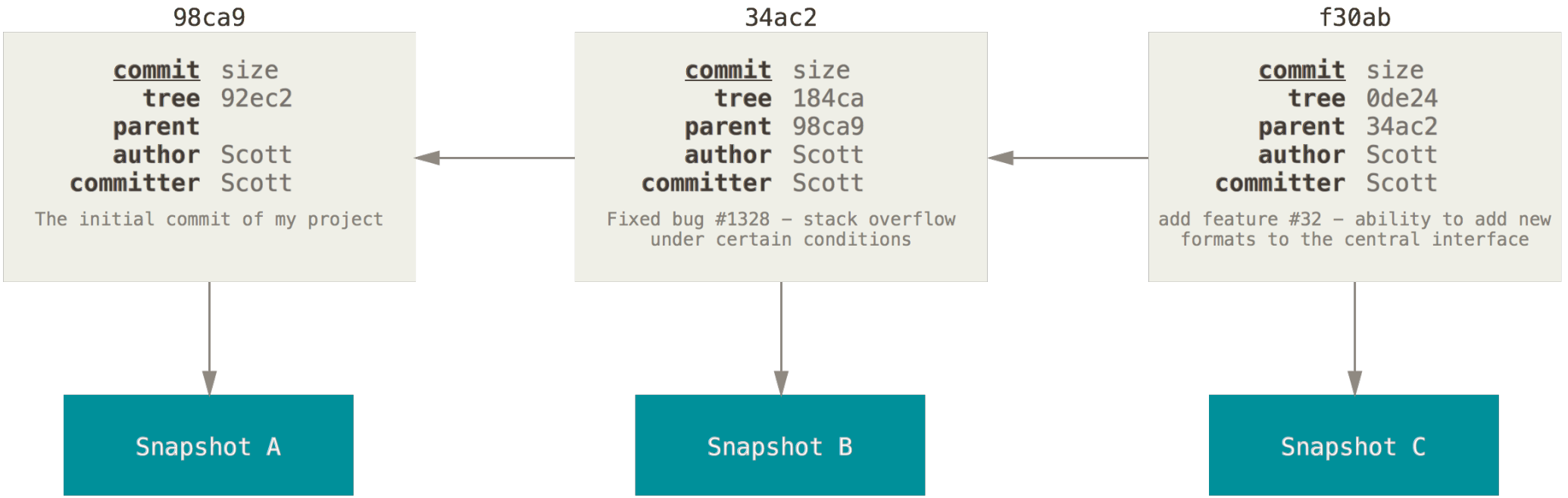
Viewing logs

To see a log of all changes in your local repo:

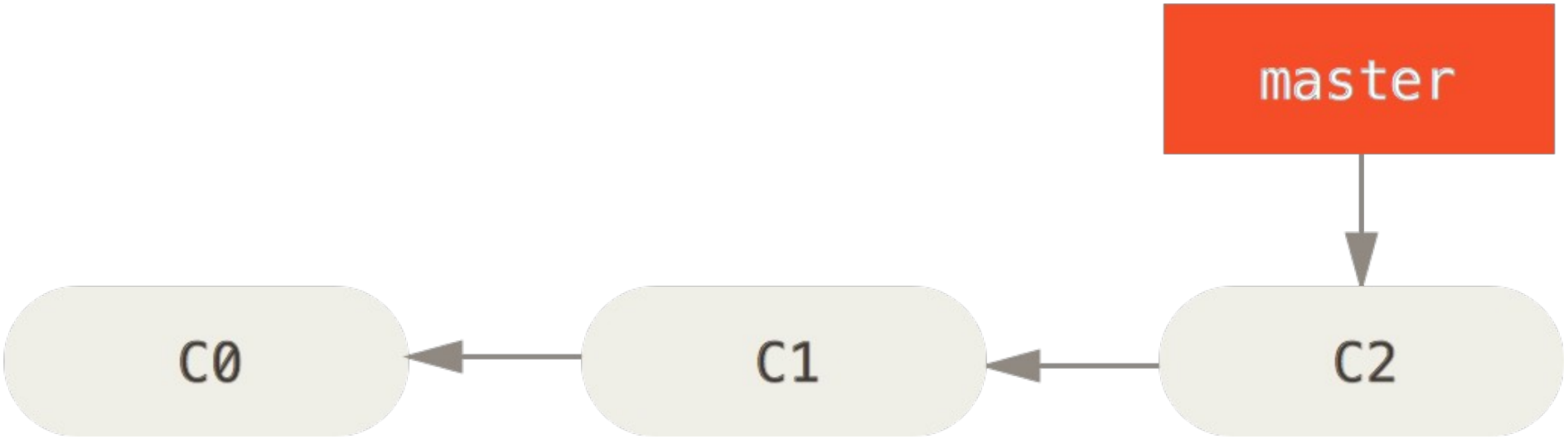
```
$ git log
$ git log --oneline   (to show a shorter version)
$ git log -5 (n last commits)
$ git log -p (commit content)
$ git log --author=”Adam Kosiorek”
$ git log --since=”1 week”
$ git log --graph (commit tree)
```

What is a commit?

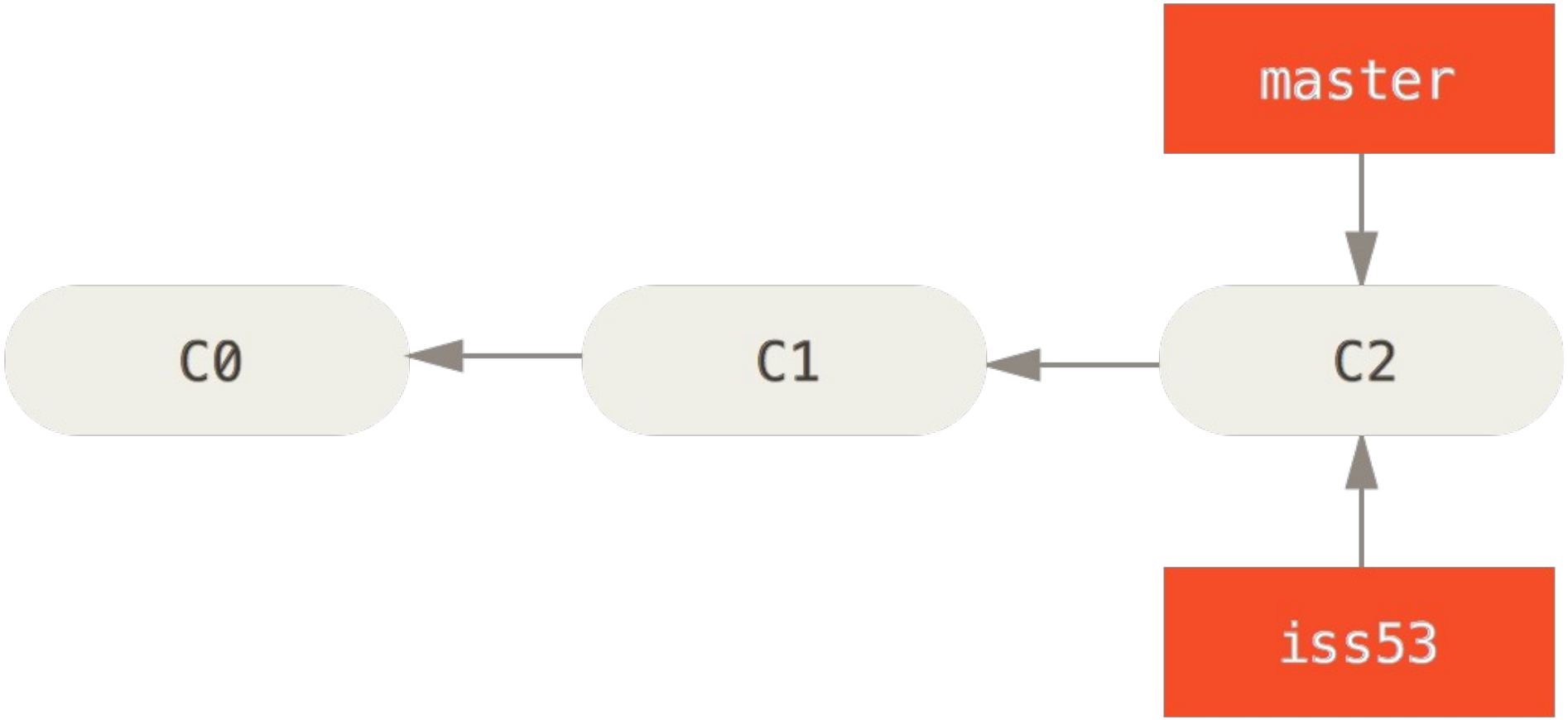
- Snapshot of the repo



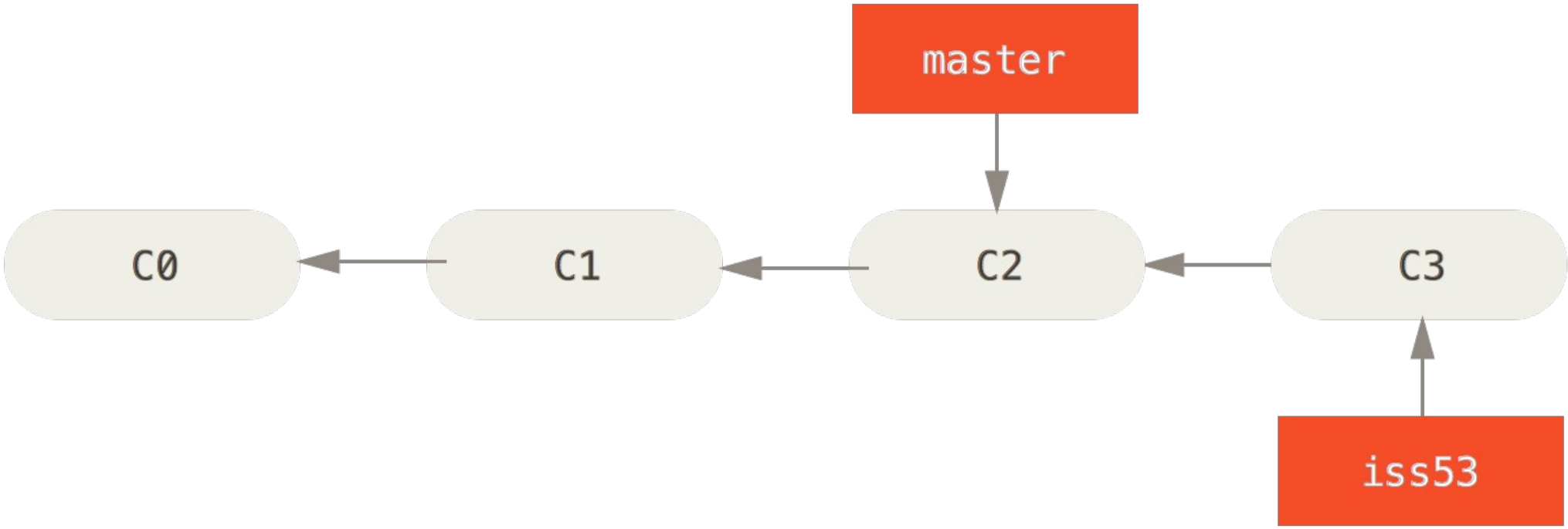
Branching – Master Branch



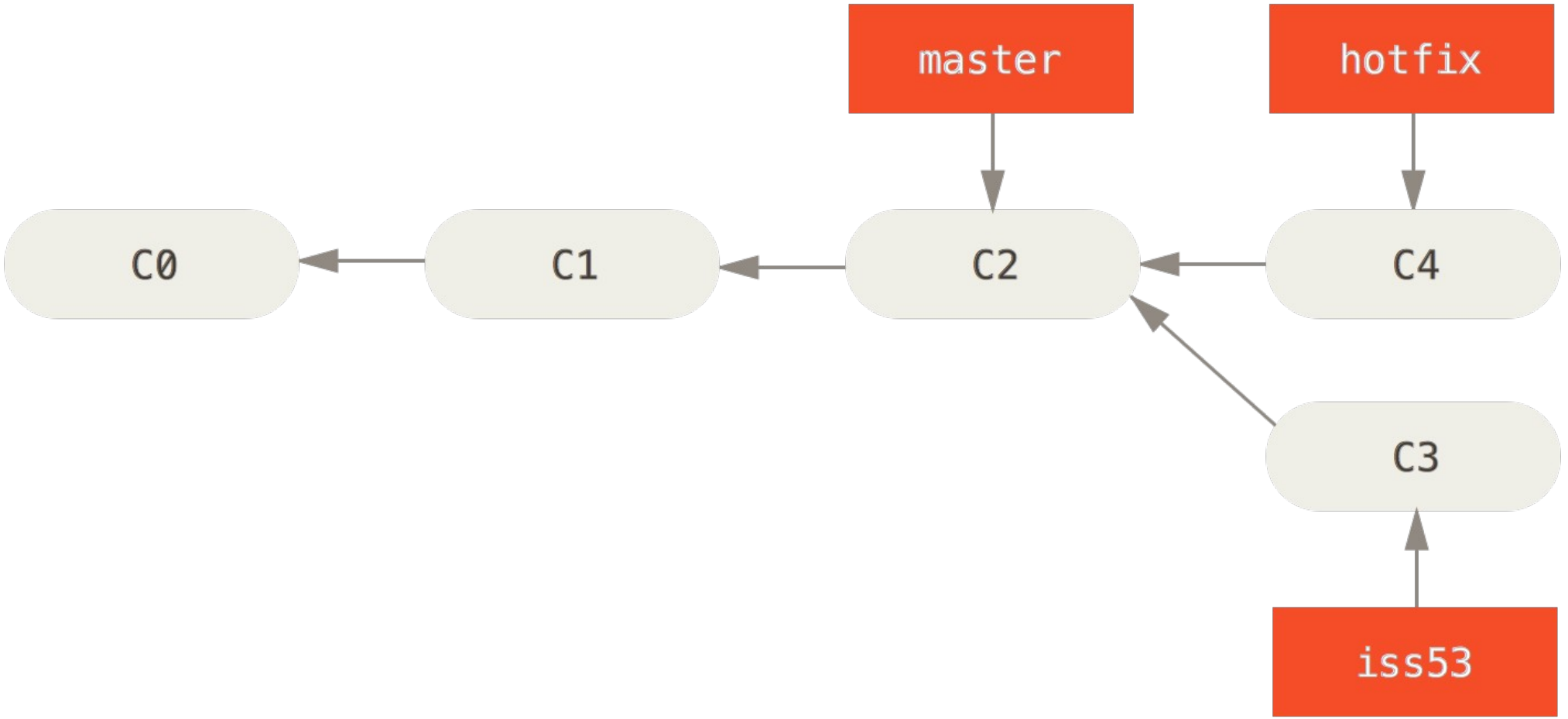
Branching – Topic branches



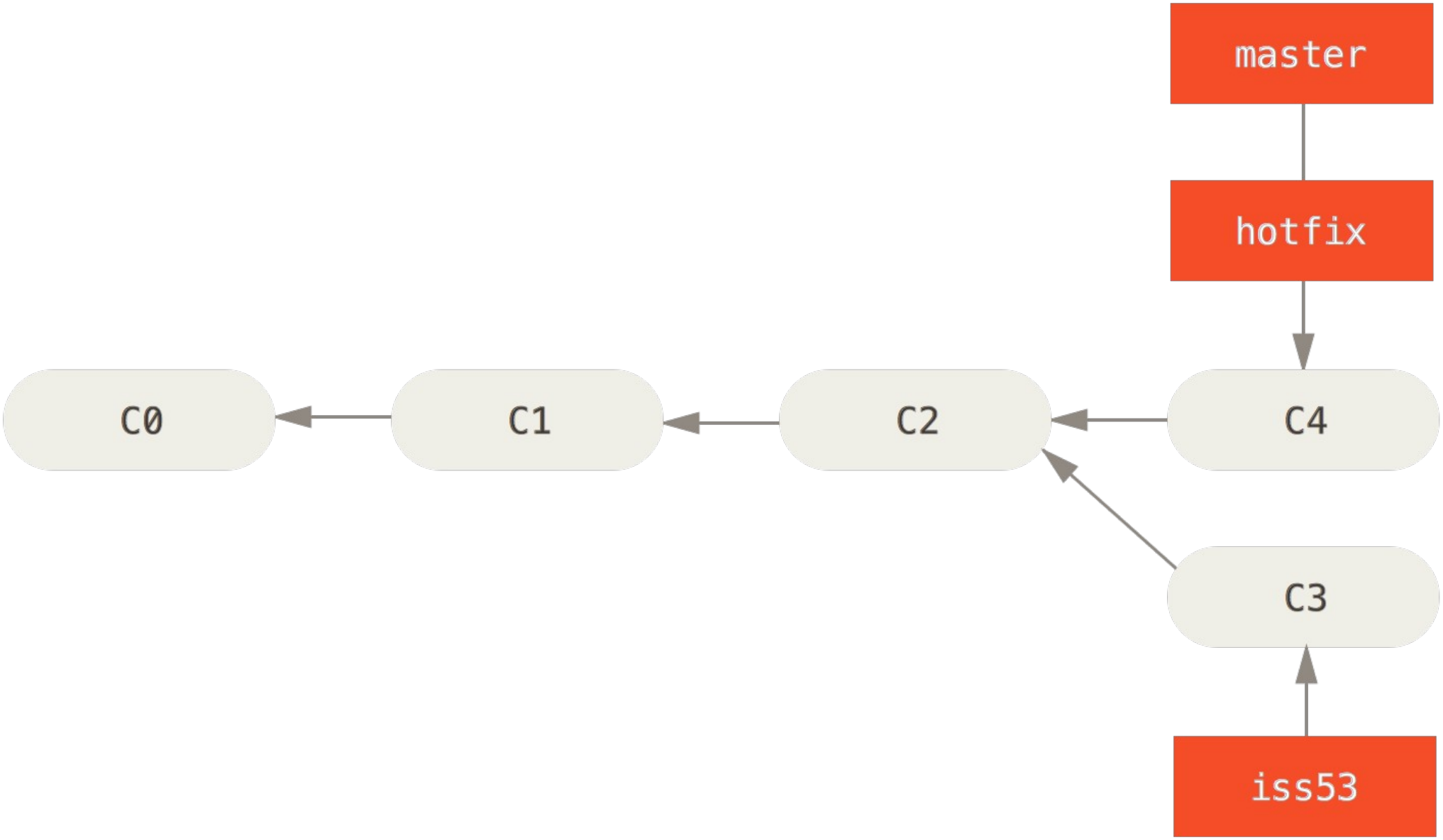
Branching – Topic branches



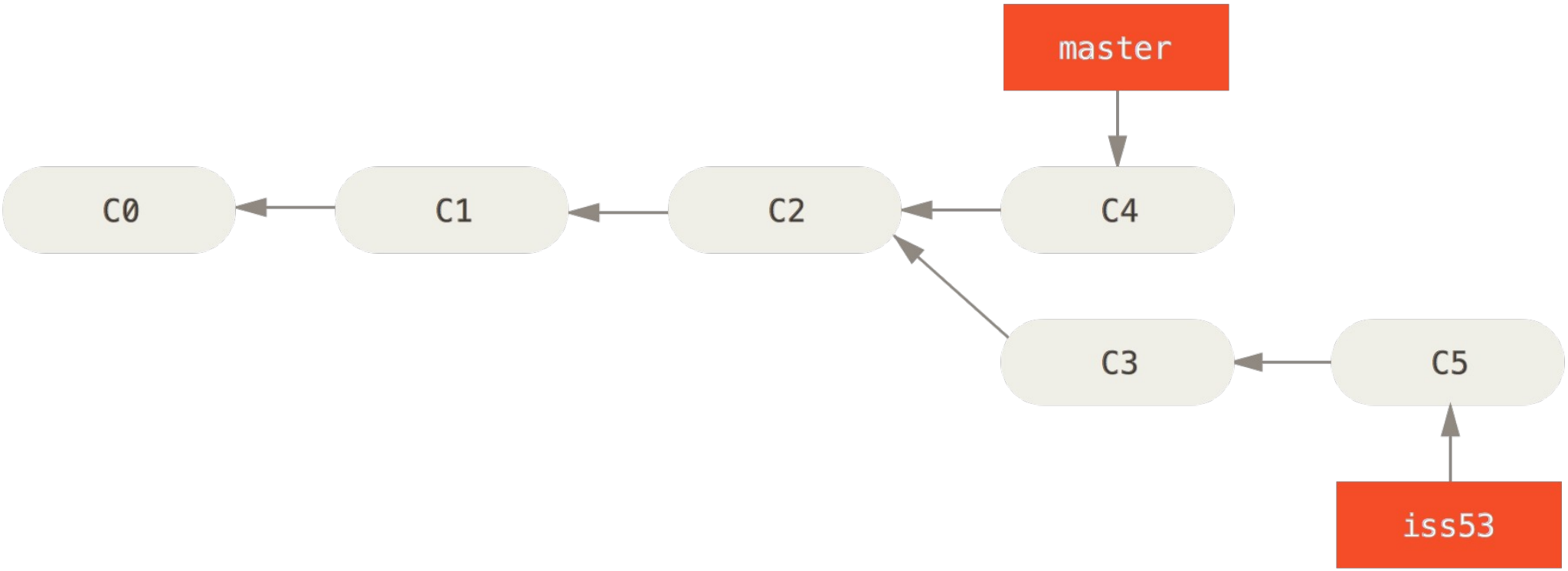
Branching – Topic branches



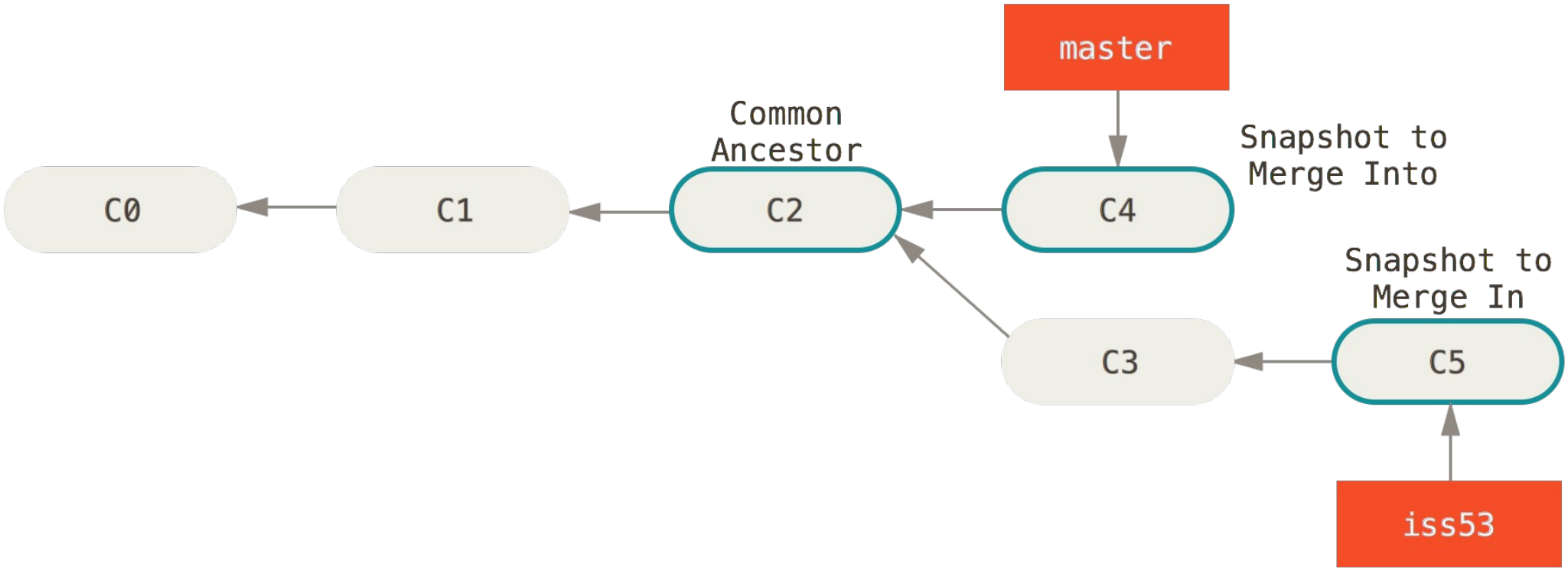
Merging – Fast-Forward Merge



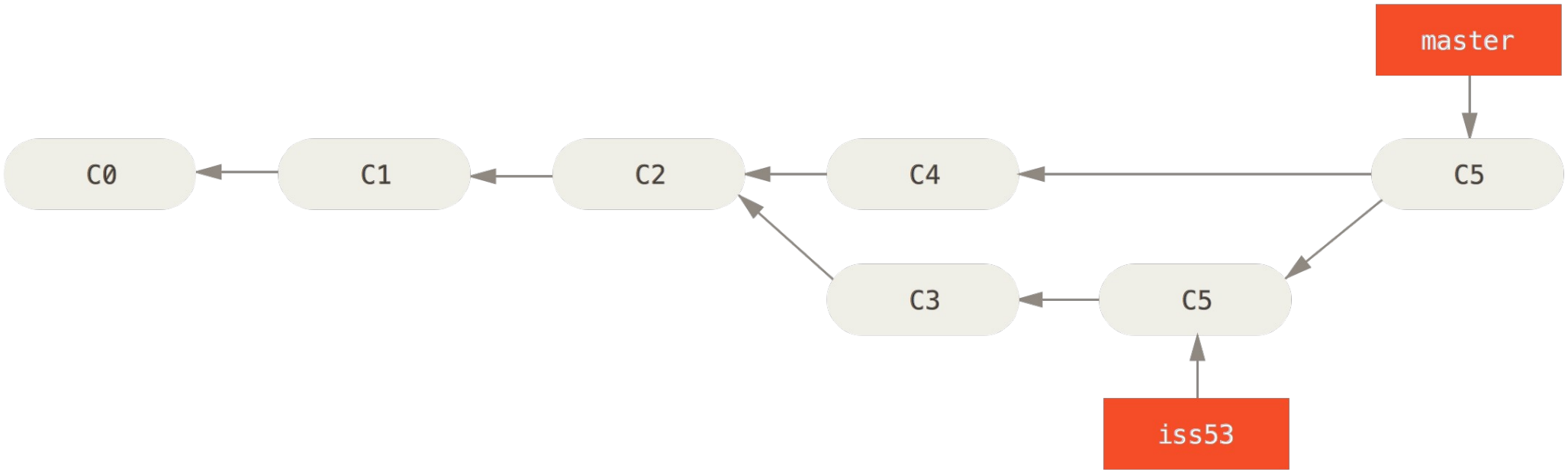
Merging – 3-way Merge



Merging – 3-way Merge



Merging – 3-way Merge



Branching

To create a branch:

- `$ git branch <name>`

To list all branches:

- `$ git branch`

To switch to a branch:

- `$ git checkout <name>`

To merge a branch:

- `$ git checkout master`
- `$ git merge <name>`

Note: `git log --graph` can be useful for showing branches.

Note: These branches are in *your local repo*!

Branching Exercise 1

1. print `2 + 2 = 5` → commit (C1)
2. print `Hello World!` → commit (C2)
3. `2 + 2 != 5` → we have a bug!
   - Create branch `hotfix`
     - Fix bug → commit
     - Checkout branch *master*
     - Add a file and commit
   - Merge with `hotfix`
   - Delete branch `hotfix`

Branching Exercise 2

- Create branch ***mulitply***
  - implement ***float mulitply(float, float)*** function
  - print ***multiplication*** instead of ***sum***
- Create branch ***divide***
  - implement ***float divide(float, float)*** function
  - print ***division*** instead of ***sum***
- Checkout ***master***
  - Merge with ***mulitply***
  - Merge with ***divide***
- What happens?

Merge Conflicts

```
<<<<<<< HEAD
            std::cout << "2 / 2 = " << divide(2f, 2f)
            std::cout << std::endl;
=======
            std::cout << "2 * 2 = " << multiply(2, 2);
            std::cout << std::endl;
>>>>>>> multiply
```

- ***resolve***
- ***add***
- ***commit***

Remote Repository

- Immutable
- Used for:
  - Sharing your work
  - Backups

- Add new remote:

```
$ git remote add <remote_name> <url>
```

Cloning → default `origin` remote

Pulling and Pushing

- **Add** and **Commit** often

```
$ git add (…)
$ git commit (…)
```

  Pull from remote repo

```
$ git pull (<repo> <branch>)
```

- **Push** your changes to the remote repo

```
$ git push (<repo> <branch>)
```

Pushing Branches

- When you want to collaborate:

```
$ git push <remote> <branch>
$ git push <remote> <local_branch>:<remote_branch>
```

- Deleting branch

```
$ git push <remote> :<remote_branch>
```

- Nobody sees that → why?

Fetching & Pulling

- Fetch data from remote:

```
$ git fetch <remote> <branch>
```

- Pull from remote:

```
$ git pull <remote> <branch>
```

- To pull branches:

```
$ git checkout -b <local_branch> <remote>/<remote_branch>
$ git checkout --track <remote>/<branch>
```

Tracking Branches

- Local branch tracks remote branch

```
$ git branch --set-upstream-to <remote>/<branch>
$ git push <remote> <local_branch>:<remote_branch>
```

- Show tracking branches:

```
$ git branch -vv
```

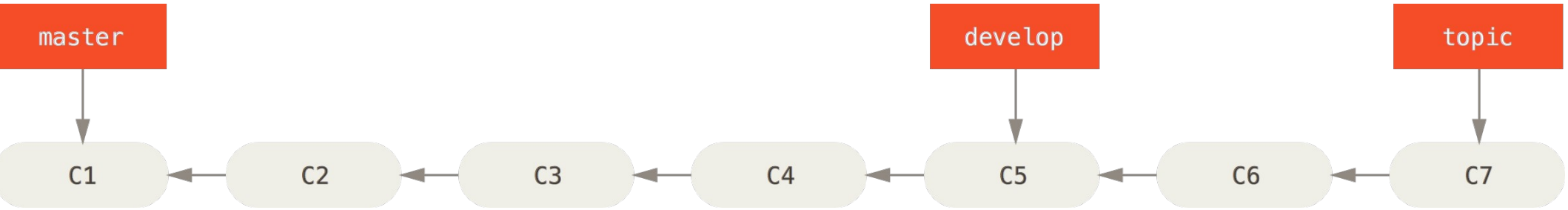
- Pull/push know what to do

```
$ git push
$ git pull
```

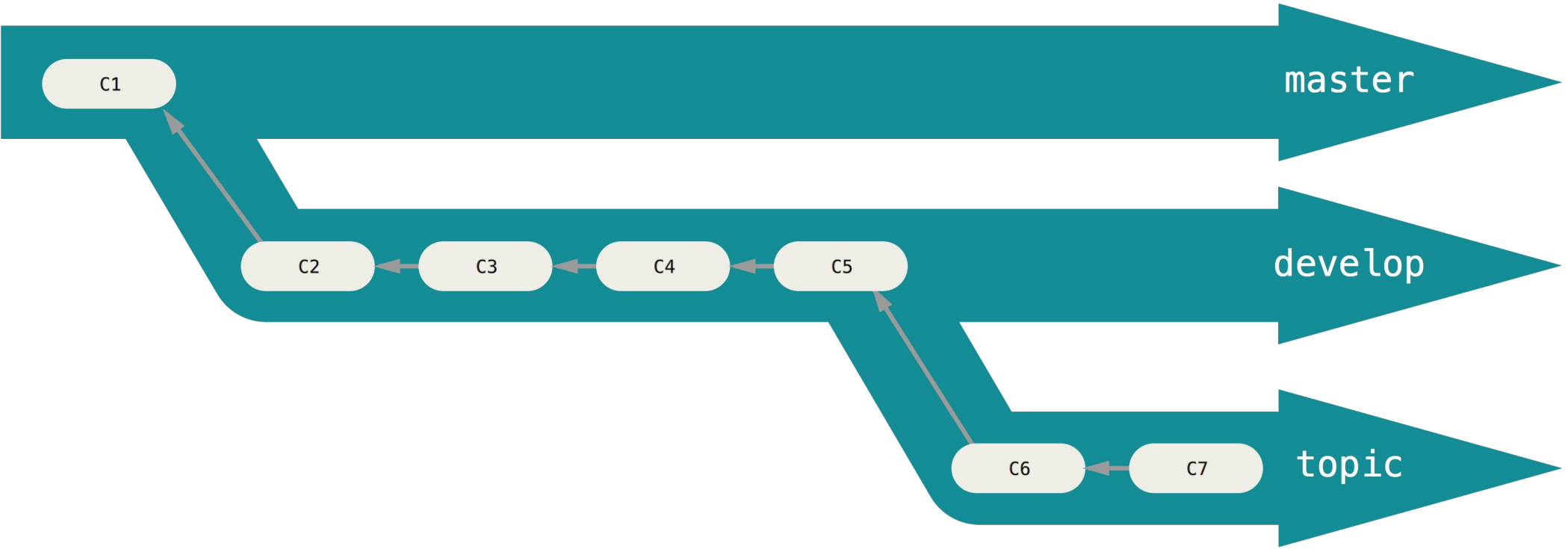
Remote Repo Exercise

1. Add a Remote Repository
2. Push to your Repo
3. Clone Repo to another dir
4. Add some changes, commit and push
5. Go back to the previous dir
6. Pull

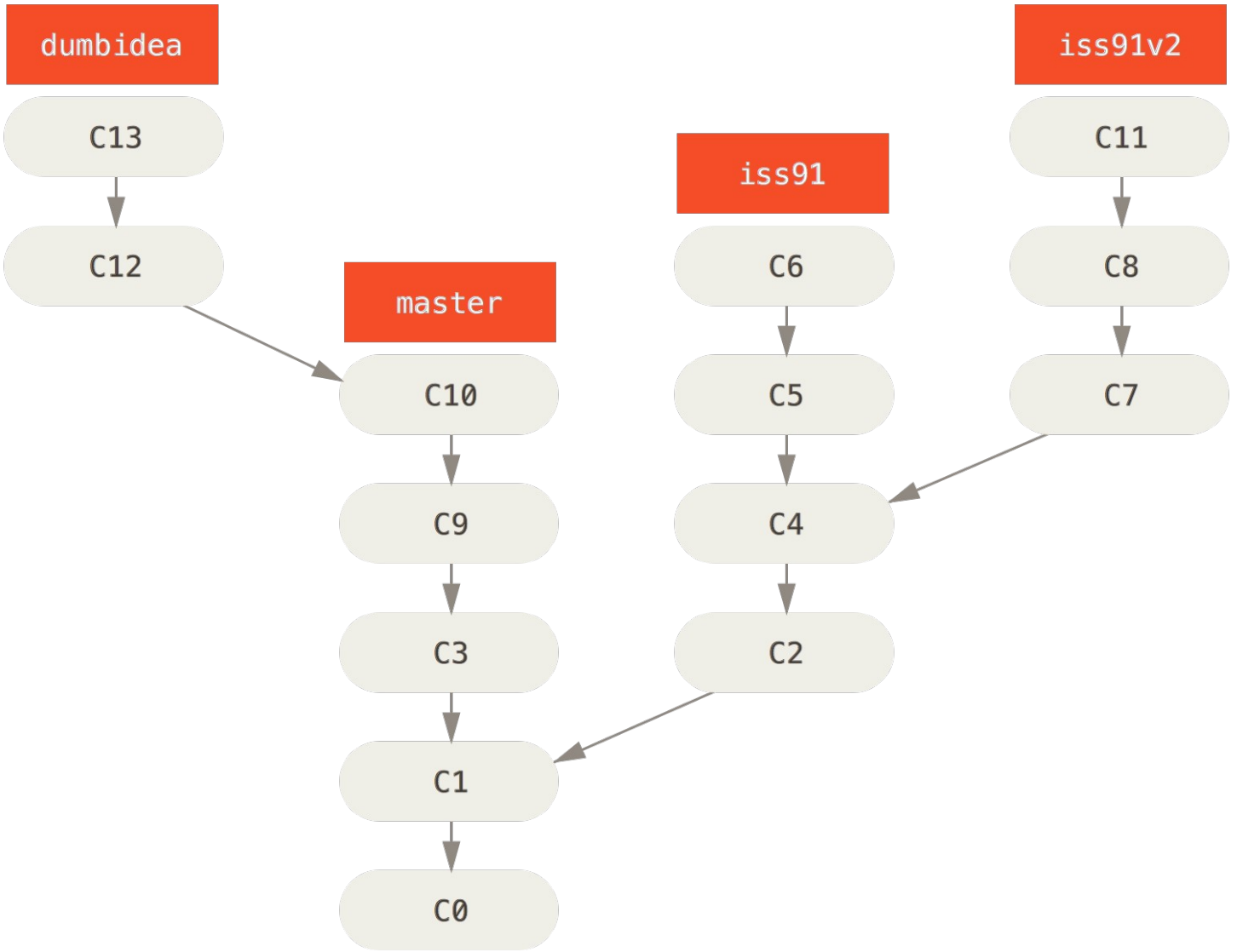
Branching Workflows – Pointers



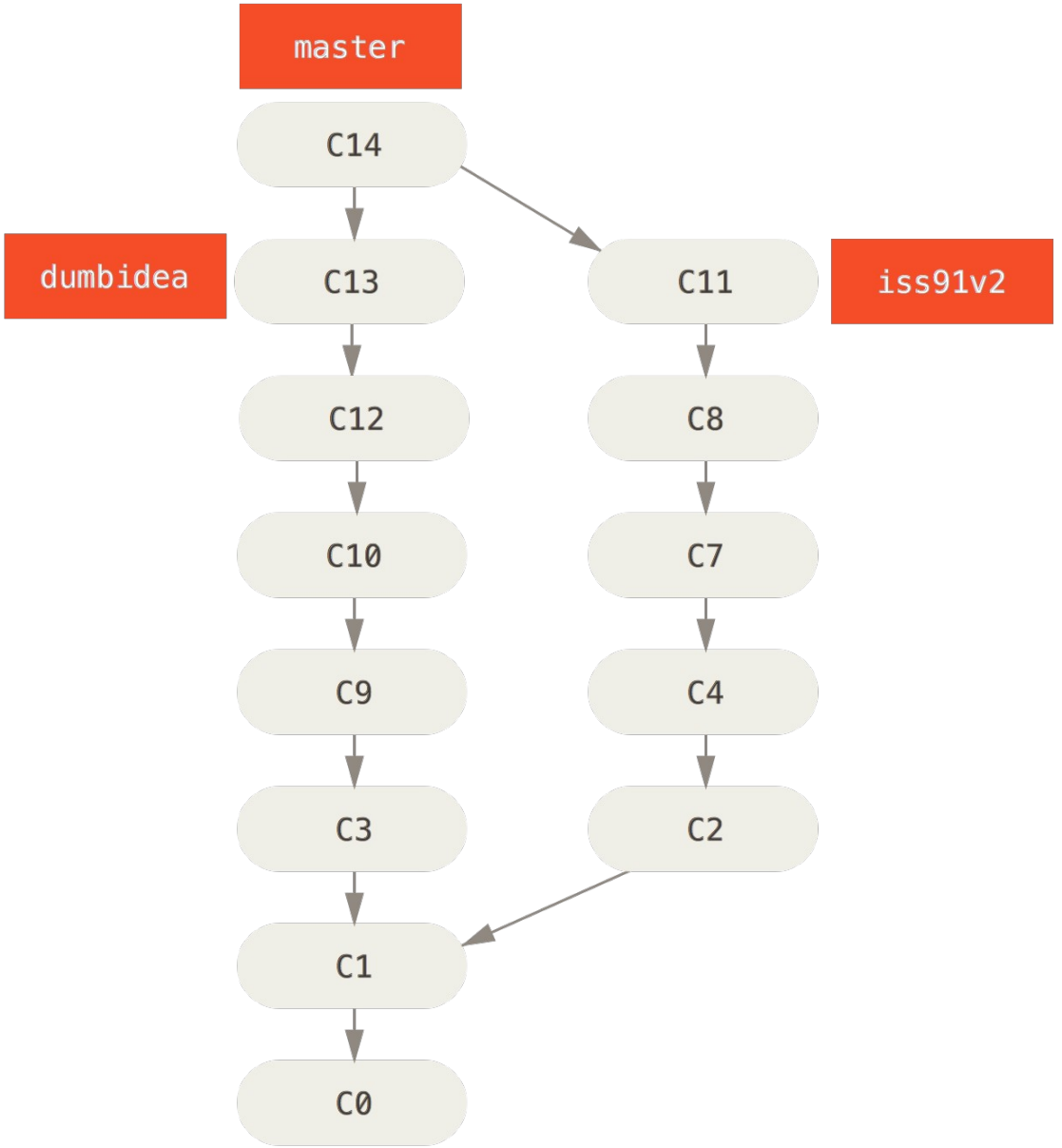
Branching Workflows – Work Silos



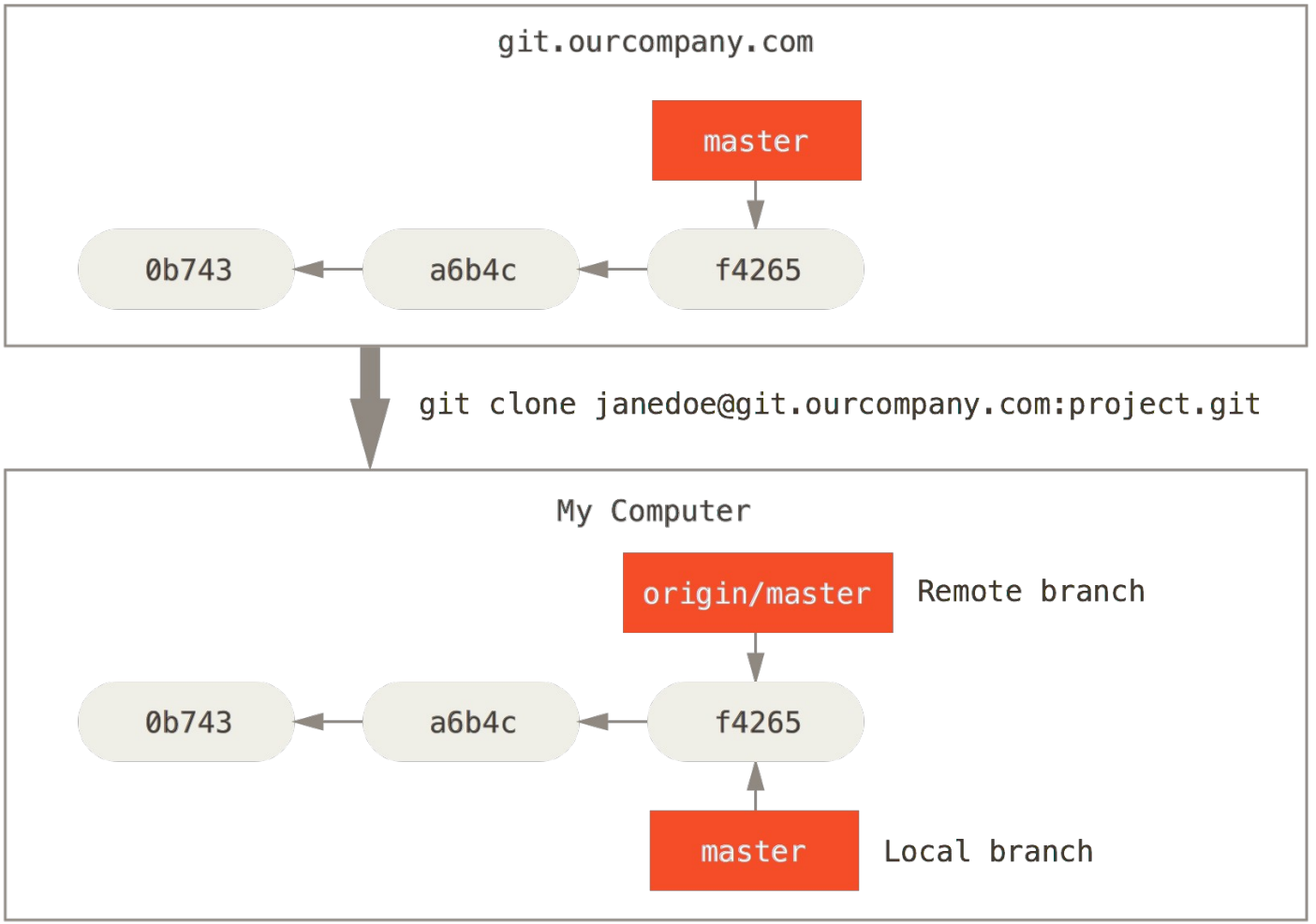
Branching Workflows – Topic Branches



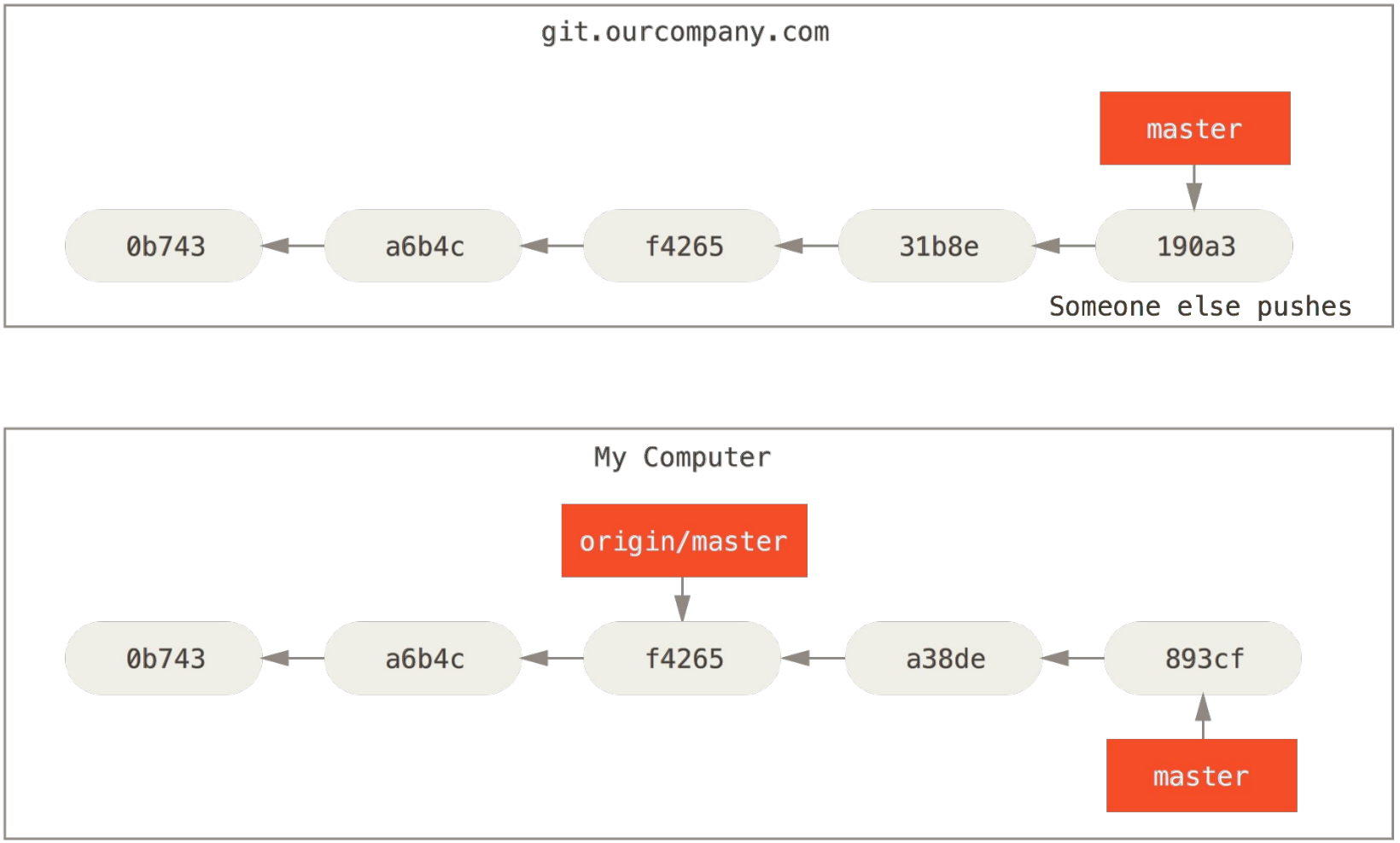
Branching Workflows – Topic Branches



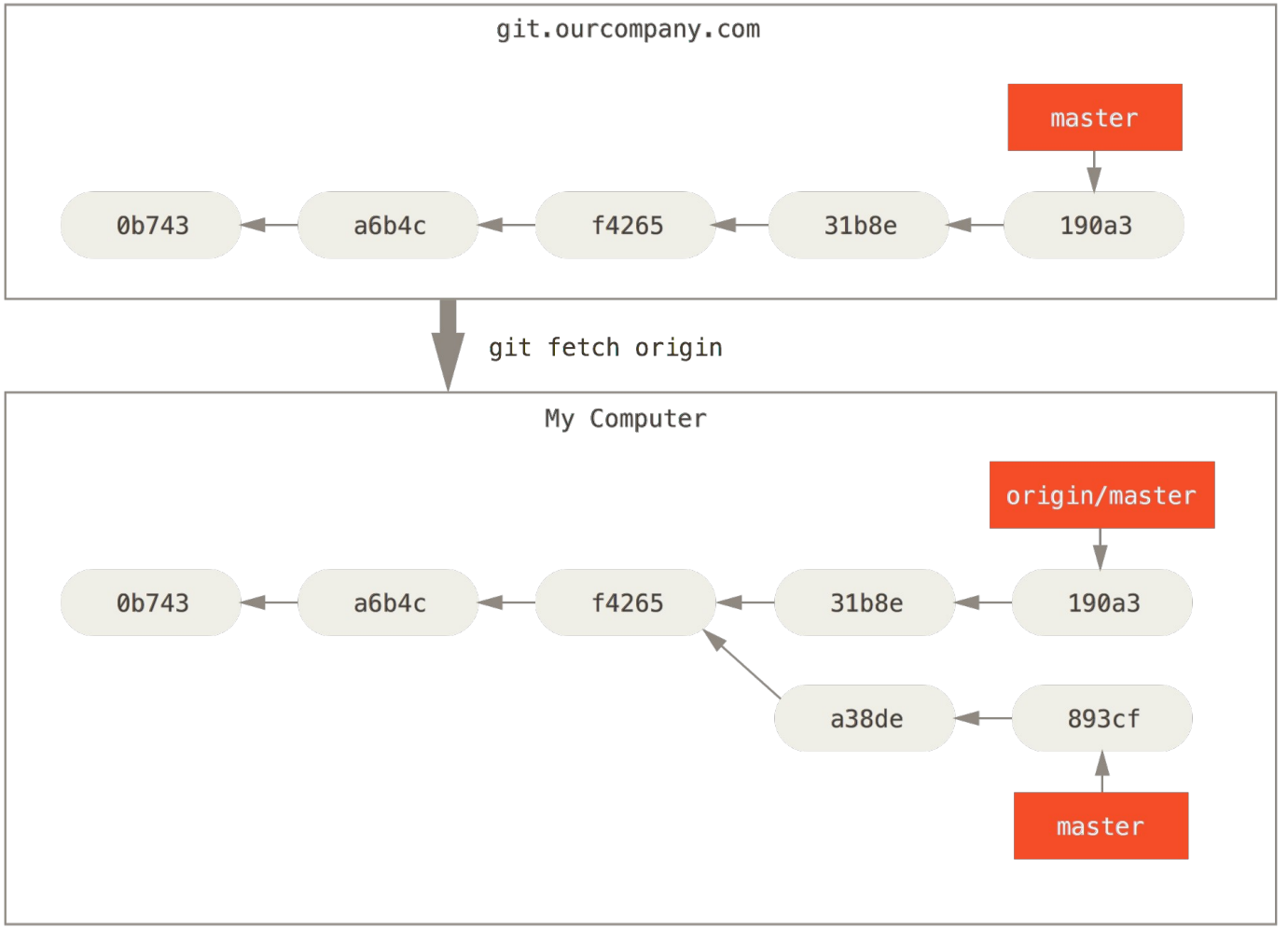
Remote Branches



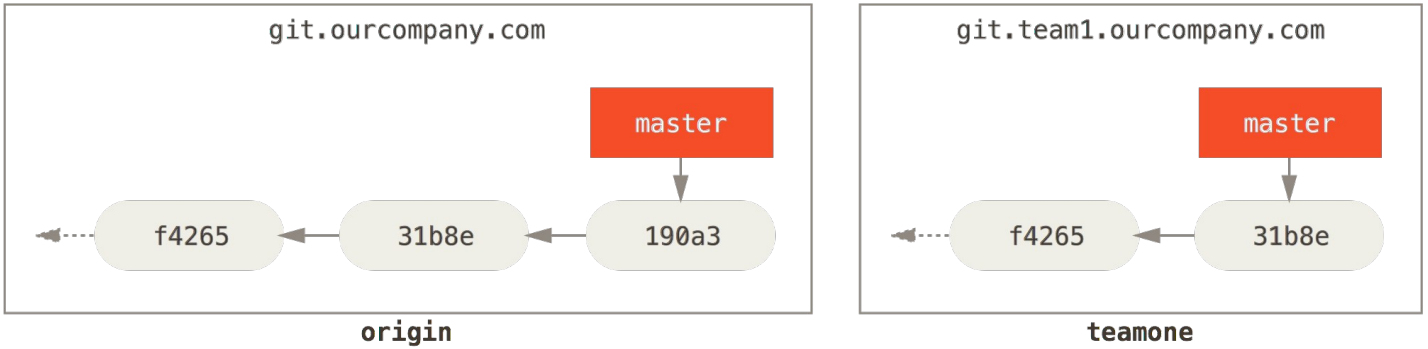
Remote Branches



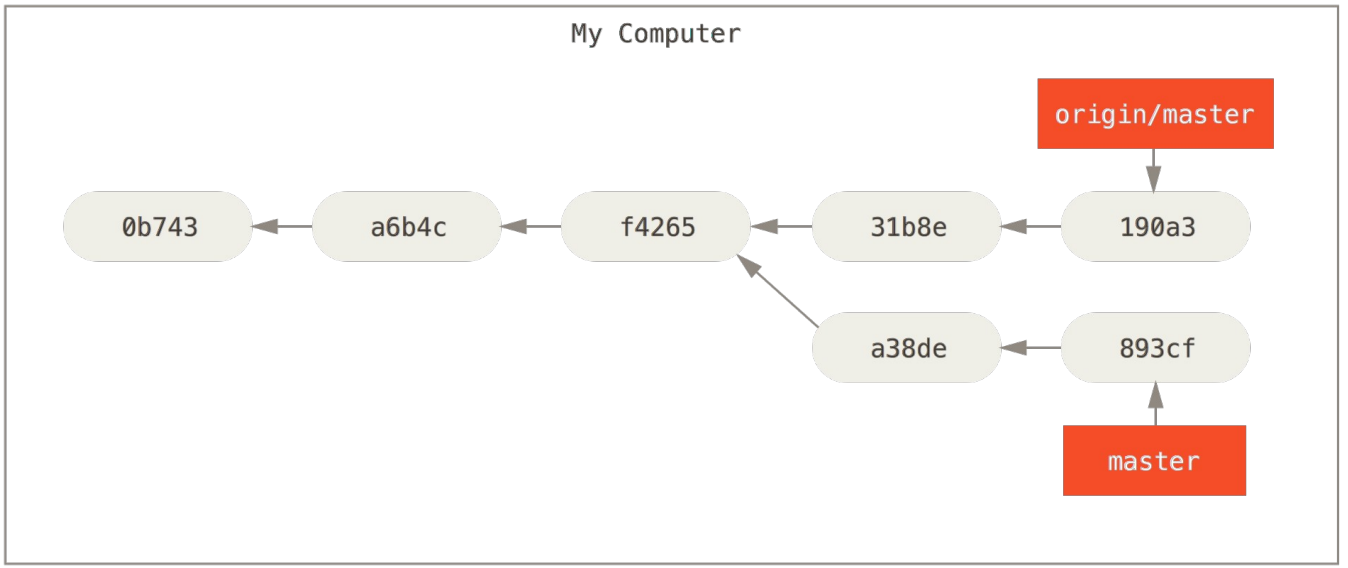
Remote Branches



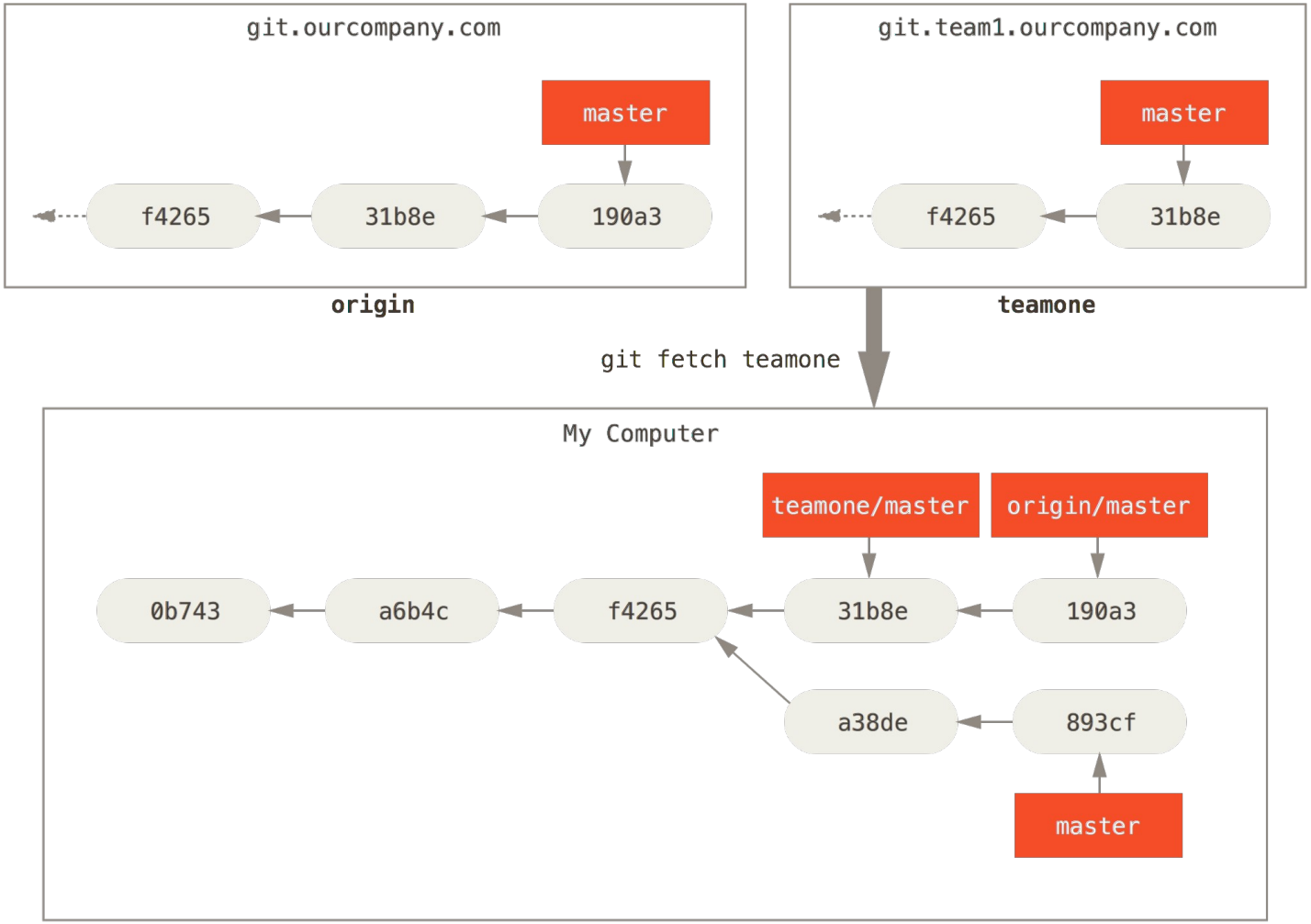
Multiple Remote Branches



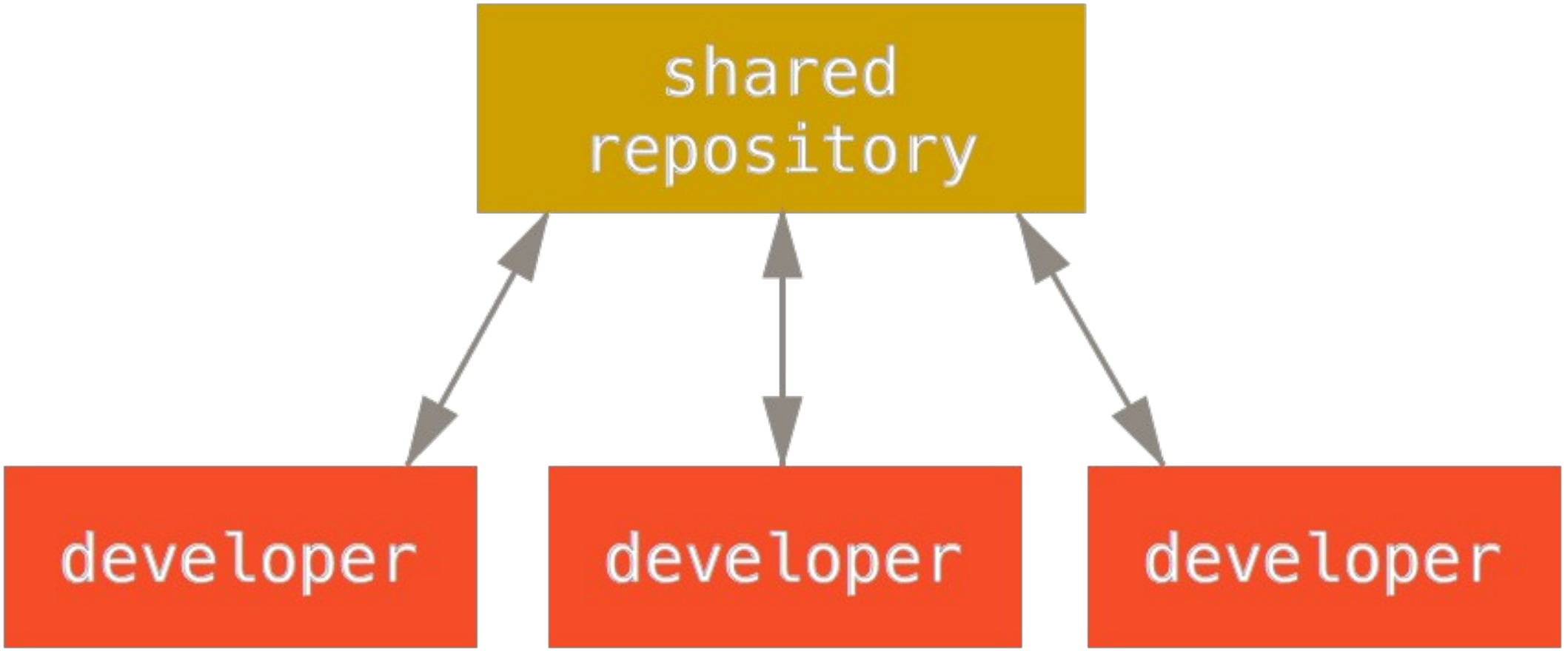

```
git remote add teamone git://git.team1.ourcompany.com
```



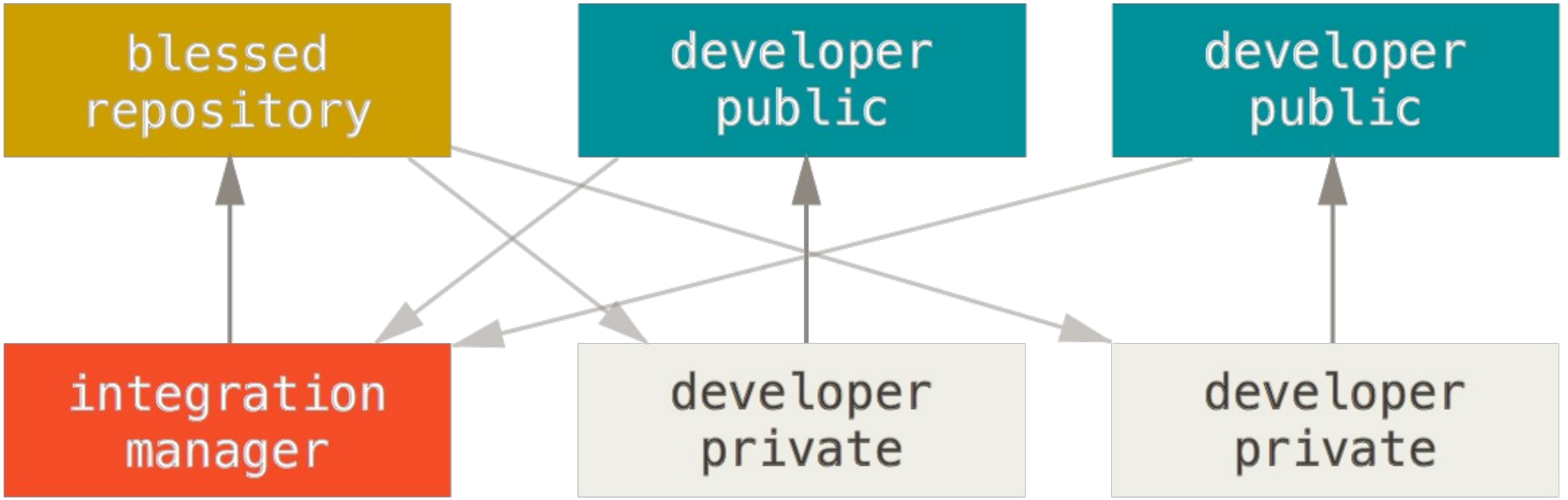
Multiple Remote Branches



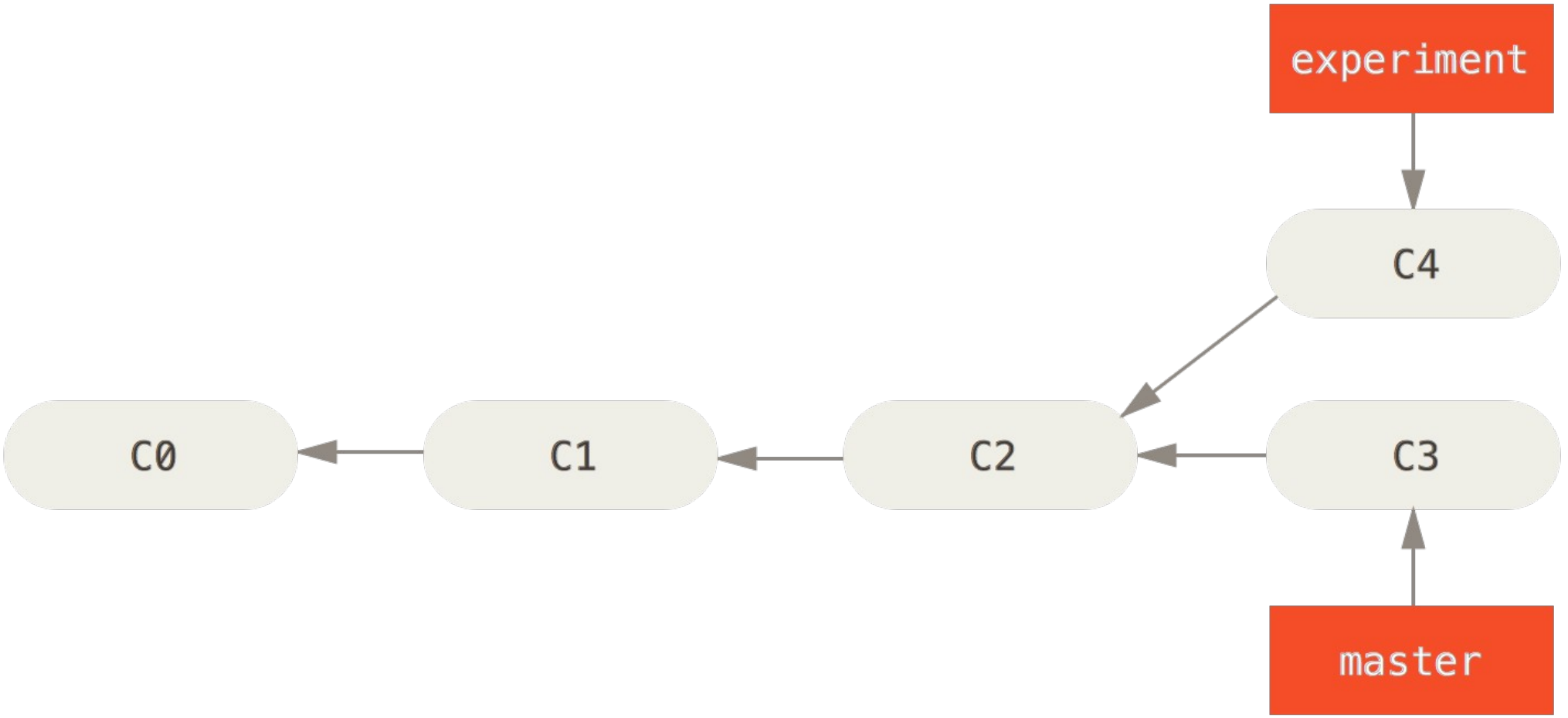
Branching Workflows - Centralized



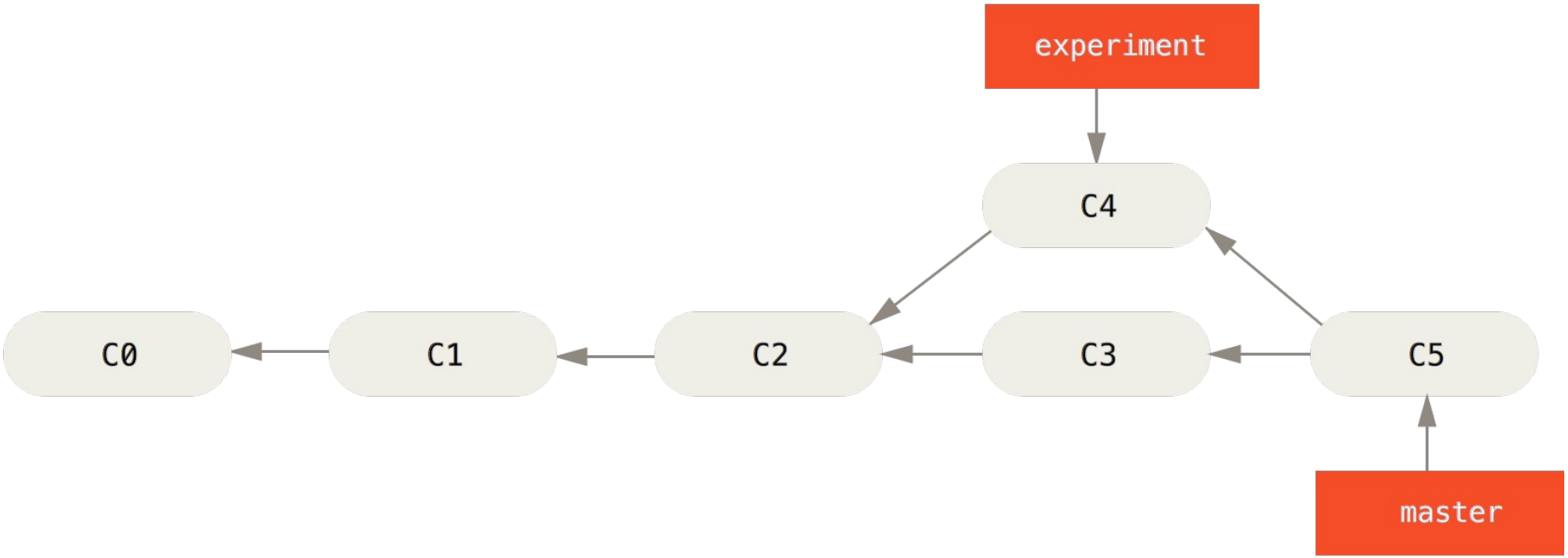
Branching Workflows – Pull Request



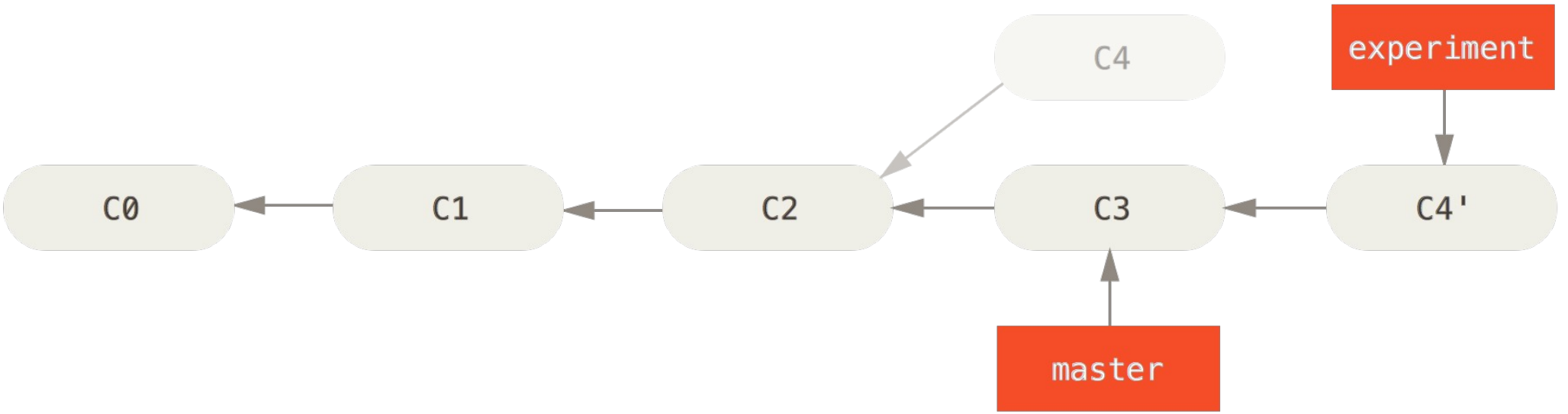
Rebasing



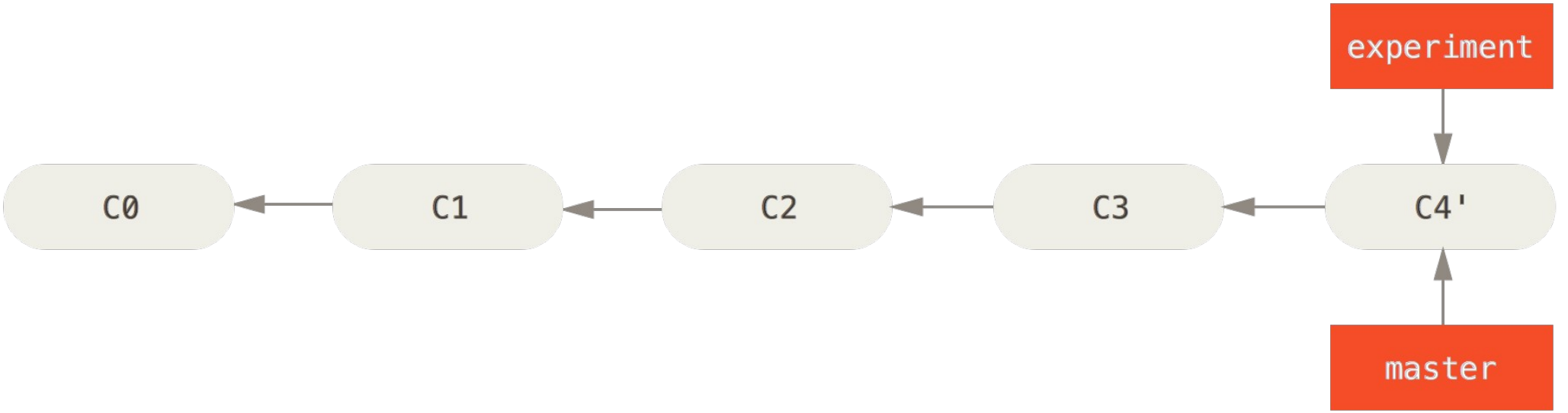
Rebasing



Rebasing



Rebasing



Rebasing

- Rebase master on origin/master

```
$ git checkout master
$ git fetch
$ git rebase origin/master
```

Further Topics

- bisect
- stash
- cherry-pick
- flow

Git commands

command	description
git clone *url [dir]*	copy a git repository
git add files	adds file contents to the staging area
git commit	records a snapshot of the staging area
git status	view the status of your files in the working directory and staging area
git diff	shows diff of what is staged and what is modified but unstaged
git help [command]	get help info about a particular command
git pull	fetch from a remote repo and try to merge into the current branch
git push	push your new branches and data to a remote repository
others: init, reset, branch, checkout, merge, log, tag	