

# Advanced git topics

Philip Lijnzaad

Princess Maxima Center for Pediatric Oncology  
May 2017

# General

- **git** helps you ~~develop~~ grow software collaboratively
  - talking face to face is also very effective!
- When contributing to existing projects: stick to their coding style!
  - Reformatting just clutters the history
- Use a graphical user interface
  - git is complex ...
- Commit messages: first line < 60 characters, then empty line, then details (line breaks at ~ 72 characters)
- Don't commit broken code: commit-hooks

# Pre-commit hooks

- Fork (or clone) <https://github.com/plijnzaad/advanced-git>
- Inside the working copy, copy (or symlink) file `git-pre-commit-hook.sh` to [`./.git/hooks/pre-commit`](#)
  - make sure it is executable: `chmod a+x thatfile`
- Add a script with an error, or introduce a deliberate error in the R script, and try to commit it

# Pull Requests

- Very useful for collaborations with remote projects
  - Within one room: commit to the same repo
- A Pull Request ('PR') is essentially asking for peer review
- Your code, and your commit history should be **clean**
  - if not, it will reduce the chances of your improvements being accepted
- How to achieve that clean history?
  - by modifying it before pushing
- Cheating? No: let's you get rid of all useless commits
  - 'typo', 'oops', 'off-by-one', 'missing semi'

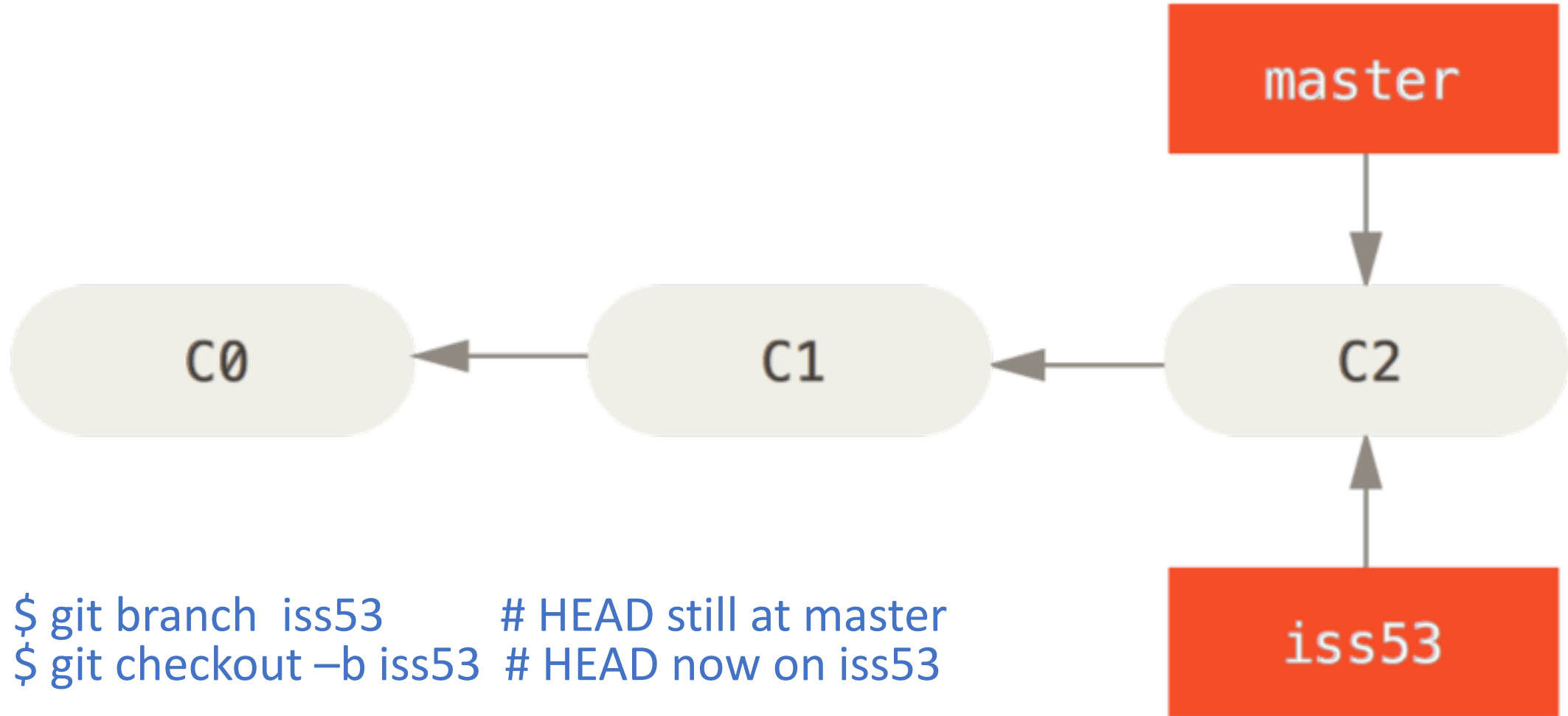
# Getting clean commits to start with

- Do your edits until you're happy
- Next: don't commit *everything*, but pick out the changes that 'belong together', by **interactively staging** them:
  - `git add -p`
- Commit only staged changes
  - `git commit` # **without any filename!**
- After this, continue with `git add -p`
  - or the equivalent in your graphical client
- Don't yet push anything!

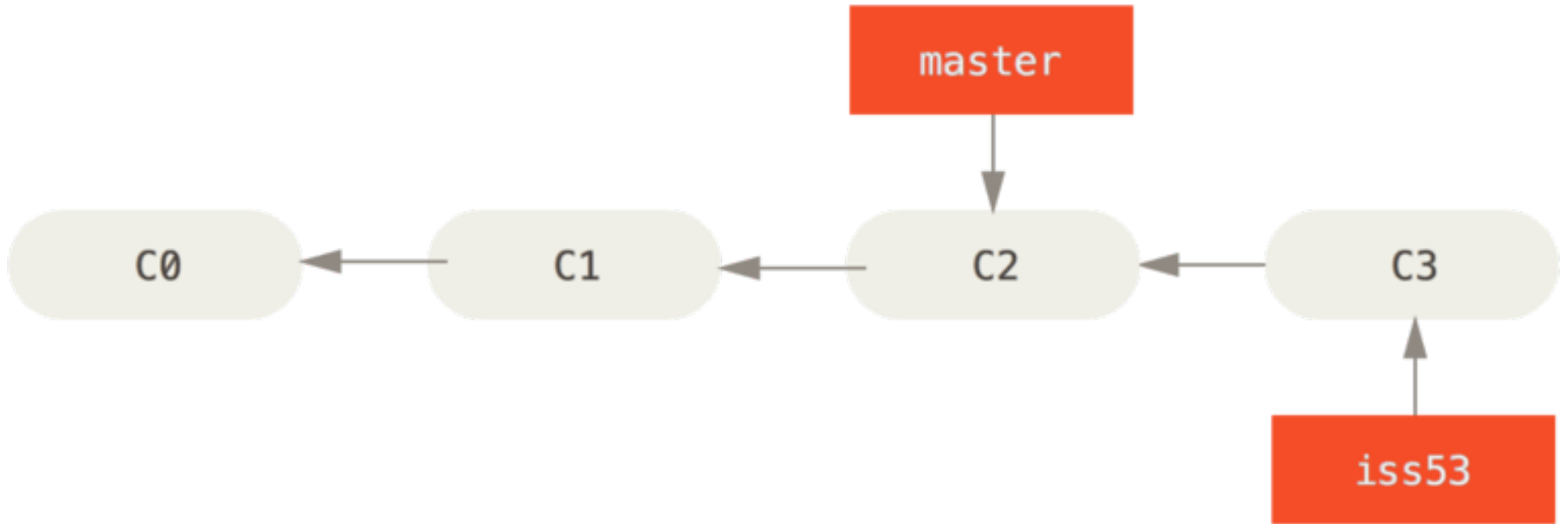
# Branching

- A branch is just a pointer to a commit
  - it moves automatically to the new commit
- Branches are cheap
- They help organize the flow of development

# Create a branch:

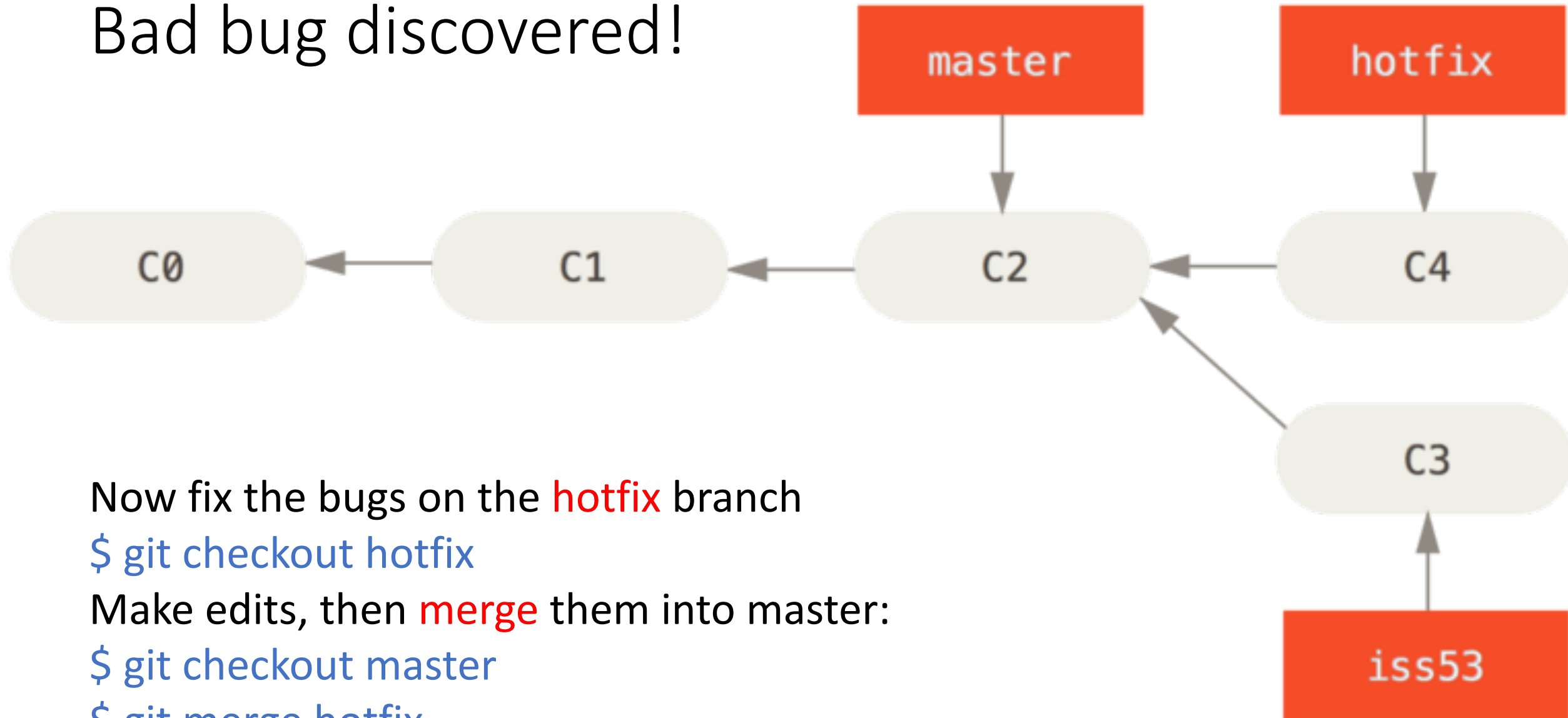


After commit:





# Bad bug discovered!



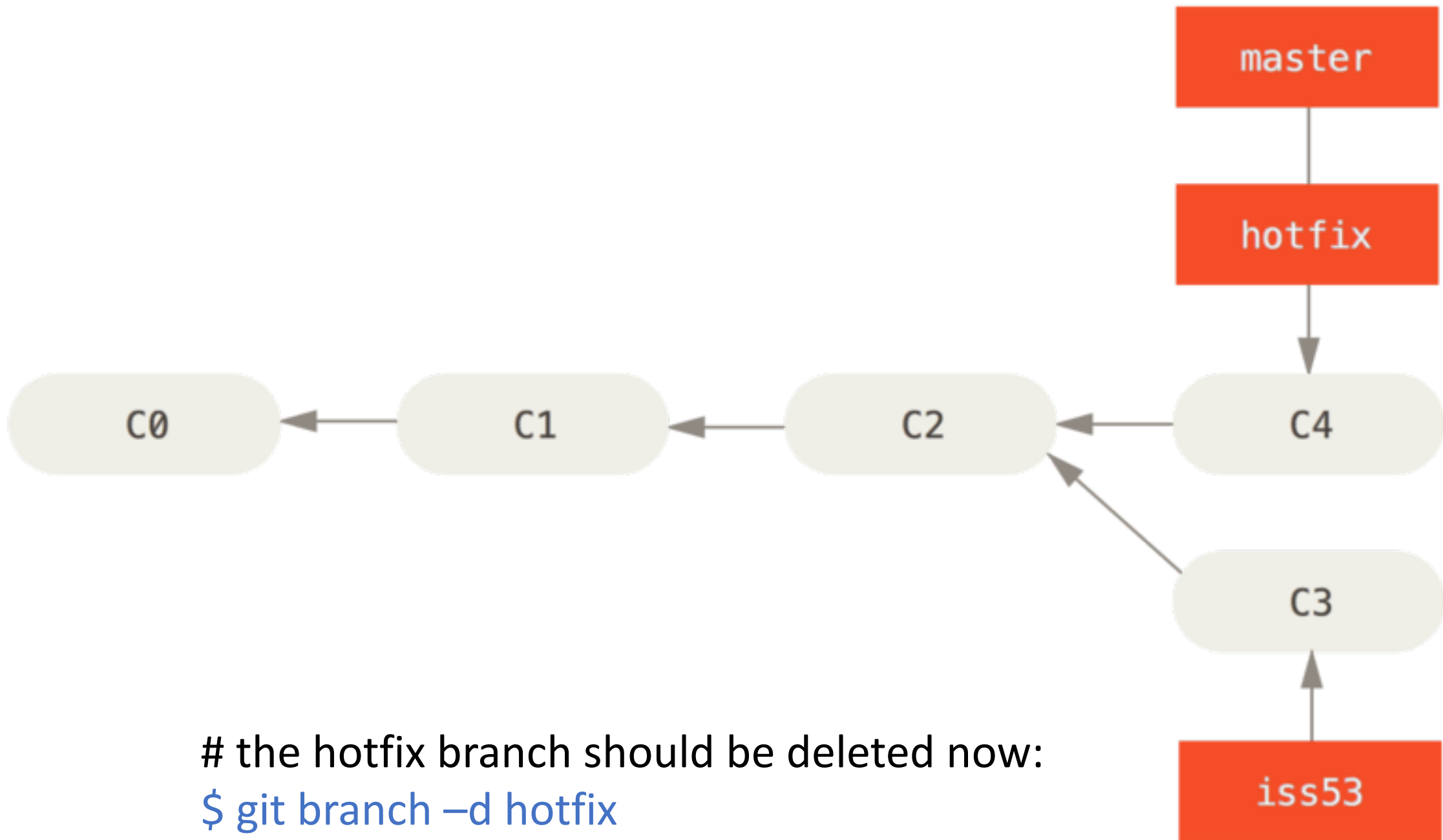
Now fix the bugs on the **hotfix** branch

```
$ git checkout hotfix
```

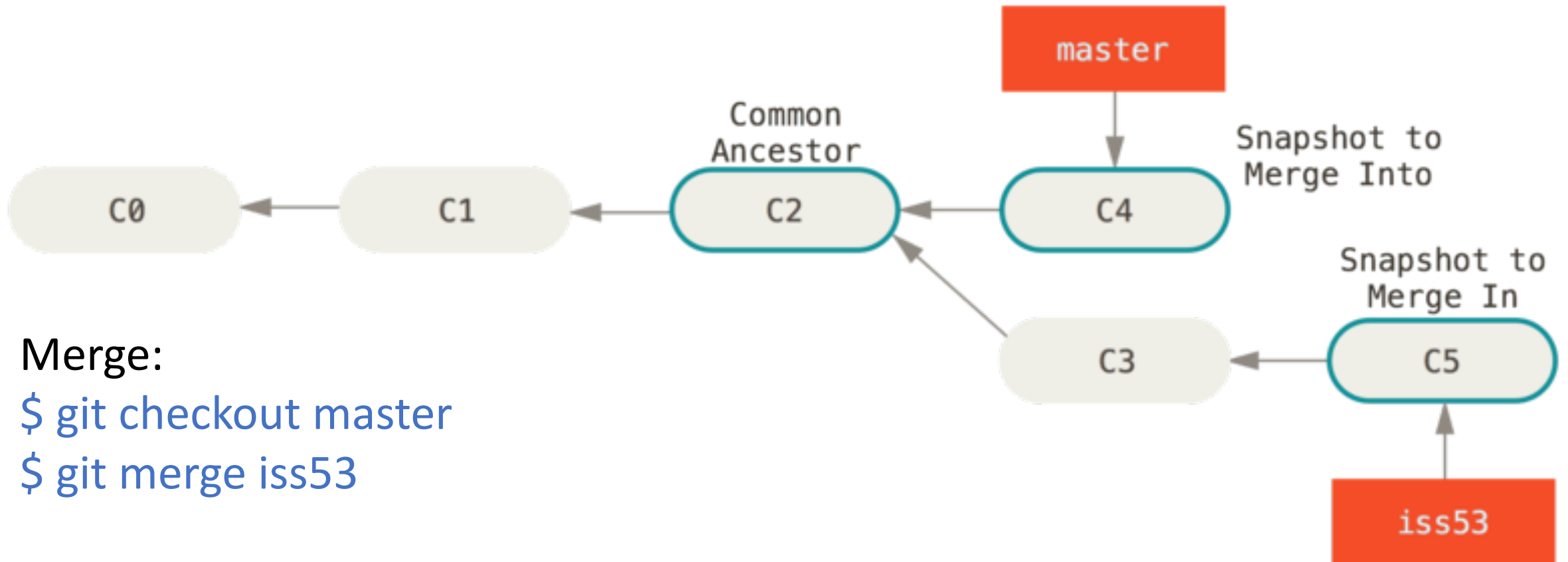
Make edits, then **merge** them into master:

```
$ git checkout master
```

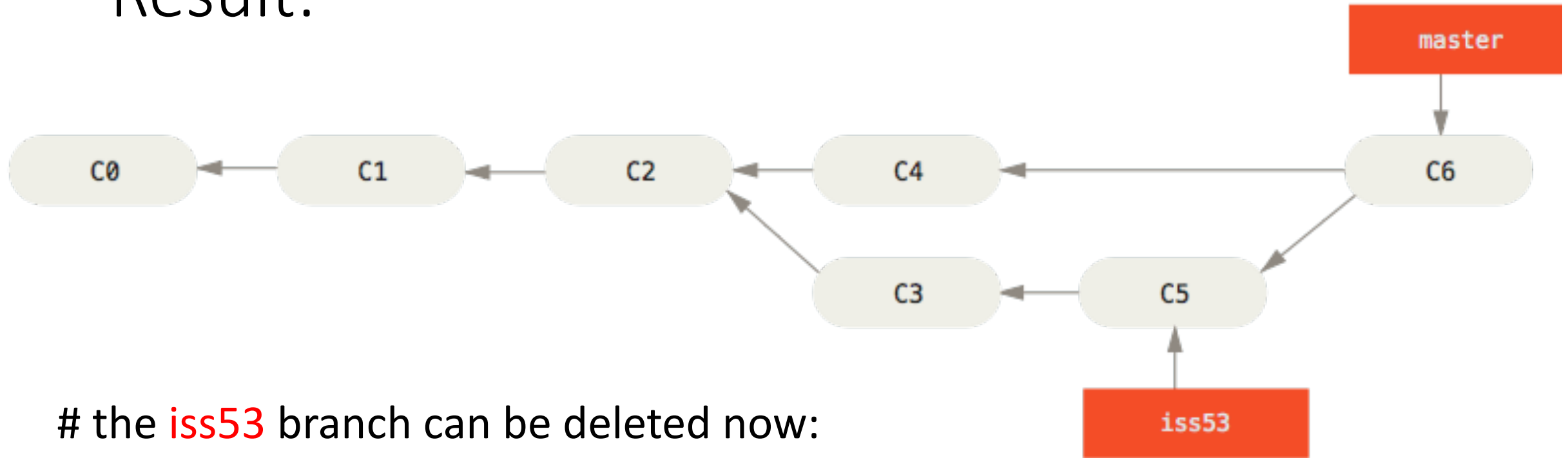
```
$ git merge hotfix
```



Meanwhile, work on **iss53** has progressed



# Result:



# the **iss53** branch can be deleted now:

```
$ git branch -d iss53
```

# seeing all branches:

```
$ git branch -a -vv
```

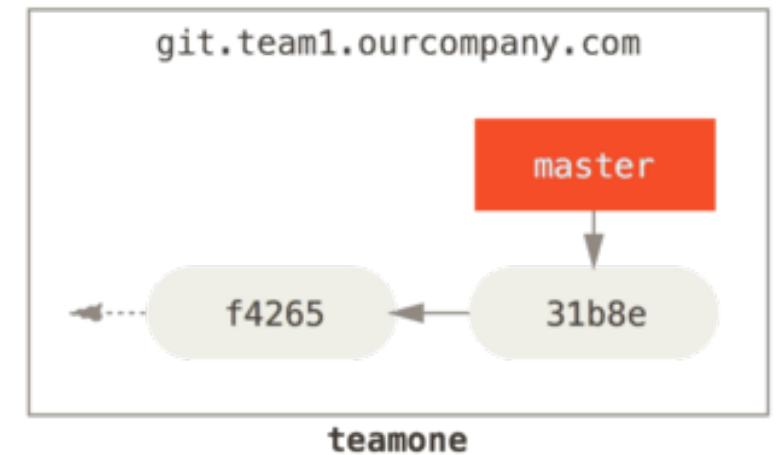
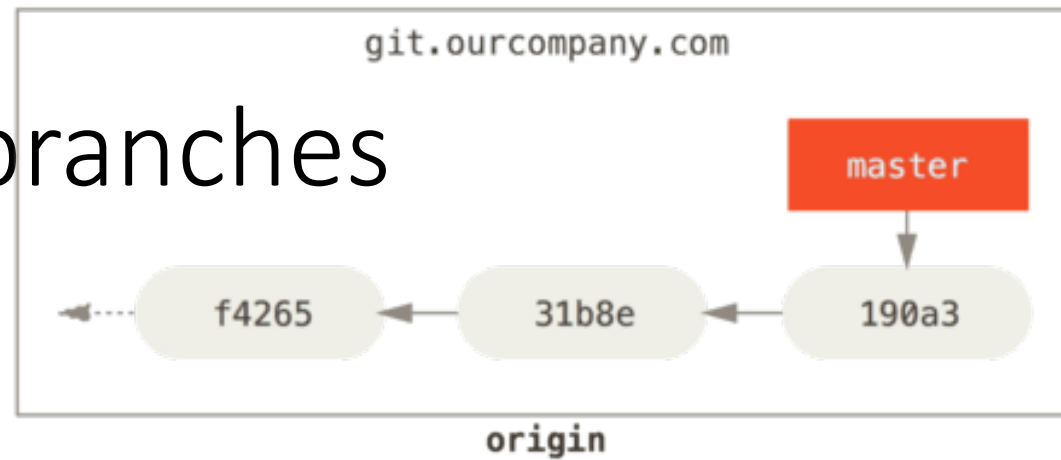
# Conflicts

- Can happen occasionally
- `git` will warn you
- Search for markers  
`<<<<<<<`, `=====  
and/or `>>>>>>>`
- Select/edit the text you want, and get rid of the markers
- Then git add the changes and commit:

```
$ git add thefile-with-conflicts  
$ git commit
```

```
file.txt  
Hunk 1 : Lines 1-11  
1 1 Git is really good at merging content,  
2 <<<<<<< HEAD  
2 3 Except when it doesn't do it  
3 4 Then it expects you to handle it yourself  
4 - and it sucks big time!  
  \ No newline at end of file  
5 <<<<<<< and it sucks big time!  
6 <<<<<<<  
7 <<<<<<< Except when its not good at it  
8 <<<<<<< And a conflict occurs  
9 <<<<<<< Then it expects you to handle it  
10 <<<<<<< and it sucks that you have to do it  
11 <<<<<<< >>>>>>> change
```

# Remote branches



Collect remote changes:

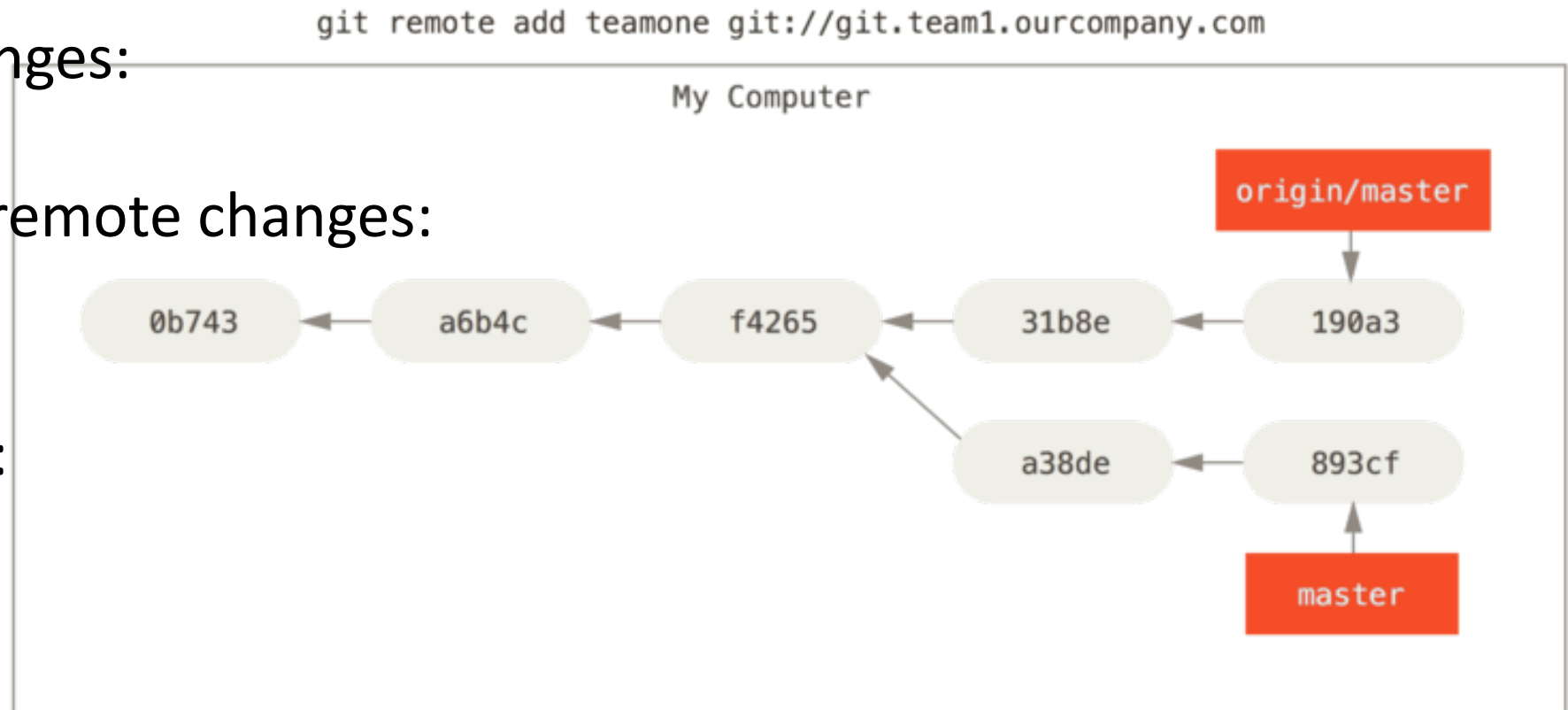
`$ git fetch`

Collect and merge remote changes:

`$ git pull`

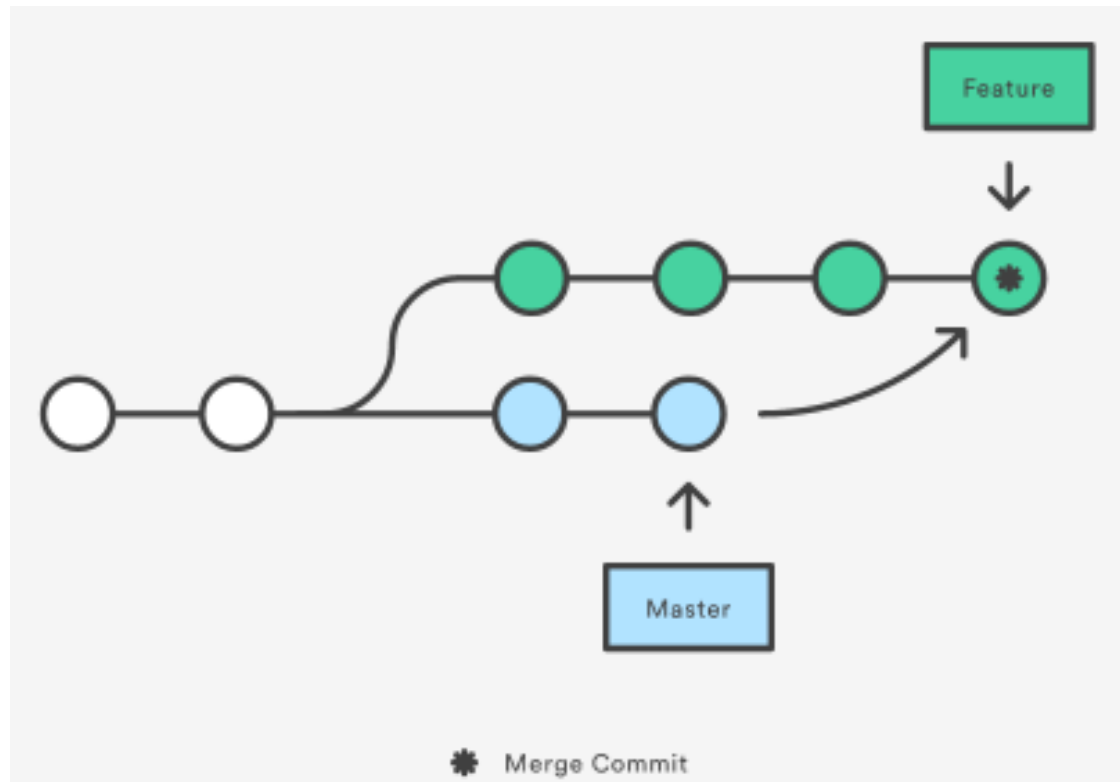
Send local changes:

`$ git push`

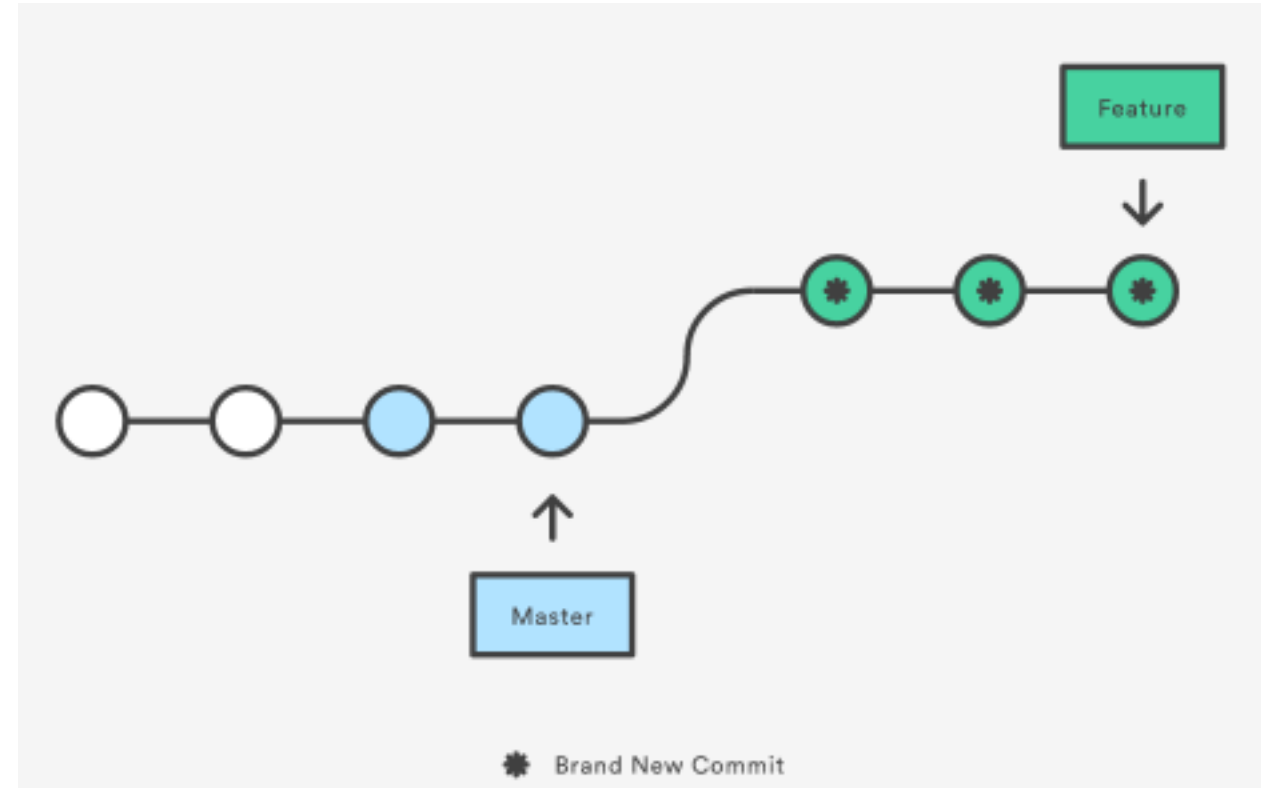


# Rebasing: 'transplant' changes onto the other branch

## Merge



## Rebase



# Why rebase?

- Simplifies history by making it linear
- Rebase rewrites the commits
- NEVER REBASE anything with a public history!
  - i.e.: only rebase local changes
- typical example: getting in the latest upstream changes
  - configure git to use rebase when pulling:  
`git config --global pull.rebase true`



# Advanced cleaning: interactive rebase

- `$ git rebase -i`
- Reorder, join or split commits
- **Only local commits!**
- Determine from where you want to start changing history, e.g.  
`$ git rebase -i HEAD~4`
- Next, use your editor to tell git what to do: `pick; reword; edit, drop, squash, fixup`.
- You can also reorder the lines to change the order (top to bottom)

# git rebase -i: splitting commits

- To split a commit that is too big, use `edit`
  - allows you to change history in more sophisticated manner
- Lands you in the command line, where you first must do  
`$ git reset HEAD^1`
- After this, reshape existing changes into separate commits, e.g. using a series of  
`$ git add -p ; $ git commit # commit without filename`
- Once ready, issue  
`$ git rebase -continue`