Week 11 Lecture 3

Team Tools

Team SCM

SCM is different in teams

- Need to protect the master branch from arbitrary changes
- Each person needs a place to work
- Need a way to coordinate changes to the shared code base

Branches in Git

- Branches represent multiple concurrent versions of the same program
- Each member has their own version of the program
- Member's versions are *merged* with the main version
- If the merged version works, it becomes the new master version

Automated Testing

The Master Branch always works

- To know that it works, you need to test it.
- The sooner the error is found, the easier it is to fix.
- Automated tests are fast

Automated Build

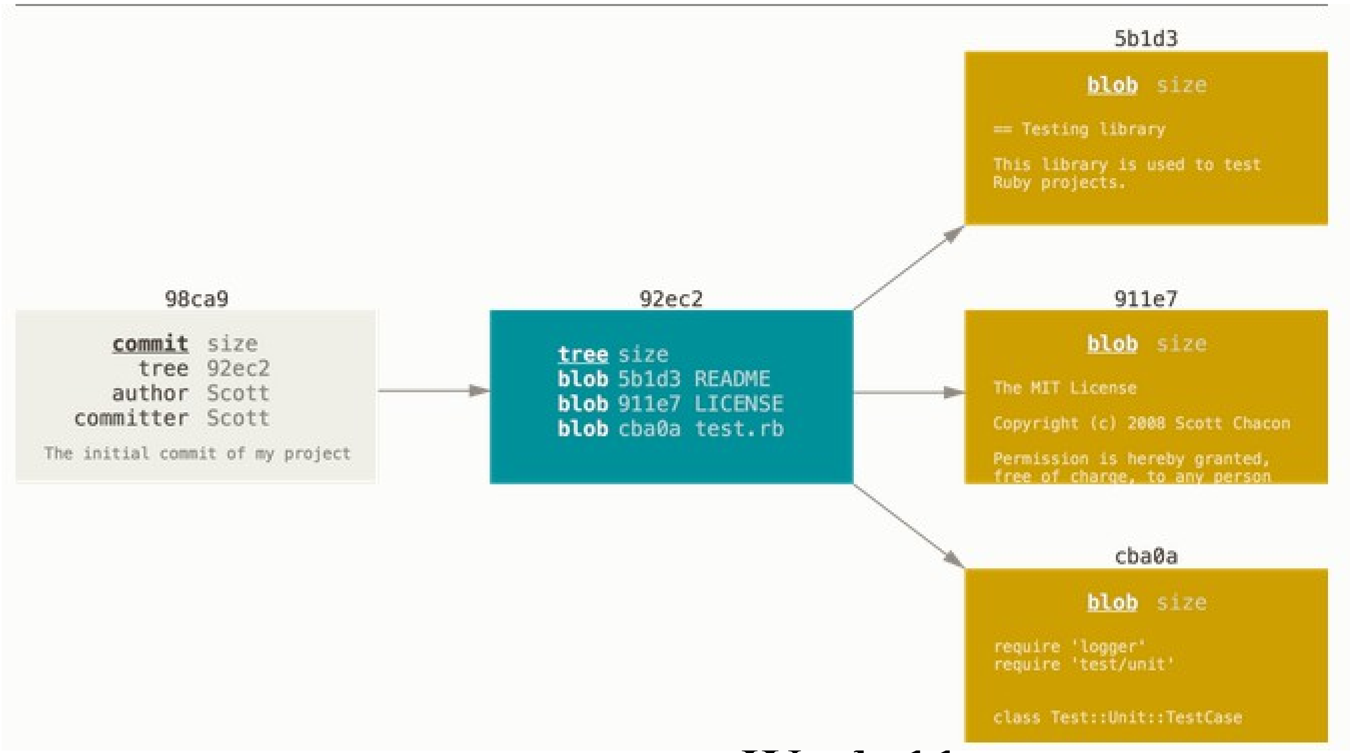
Applying Make to Team

- An automated build extends make to the team setting.
- To check in a file, the automated build first compiles the system and runs the tests against it.
  - Only if it compiles and passes the test can it be checked in to the team's server.
  - The team member may see it, but the rest of the team does not

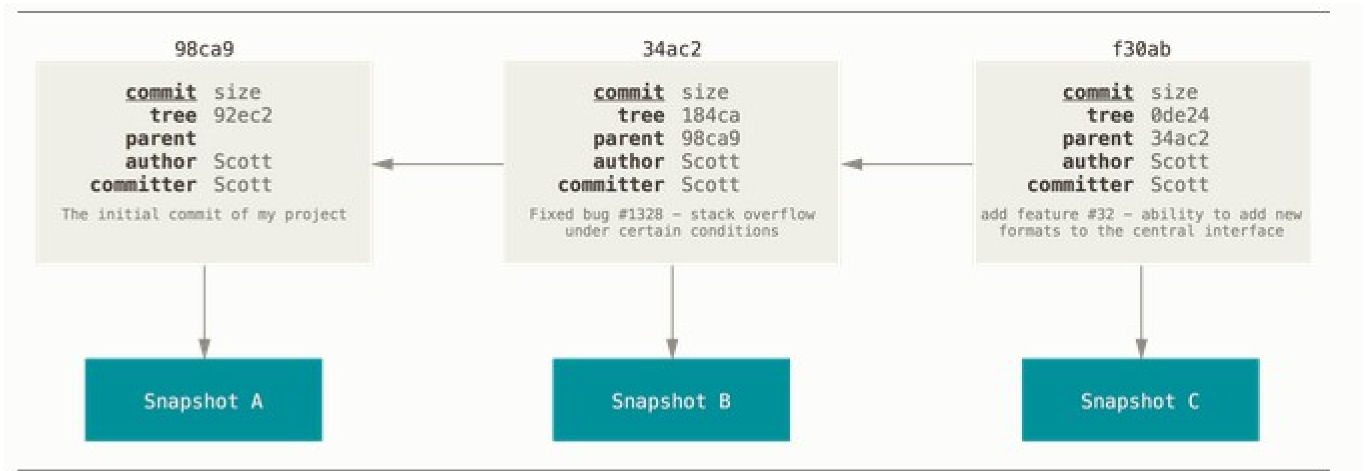
Extra: How to set up branches is Git

Checking in Files

```
$ git add README test.rb LICENSE
$ git commit -m 'initial commit of my project'
```



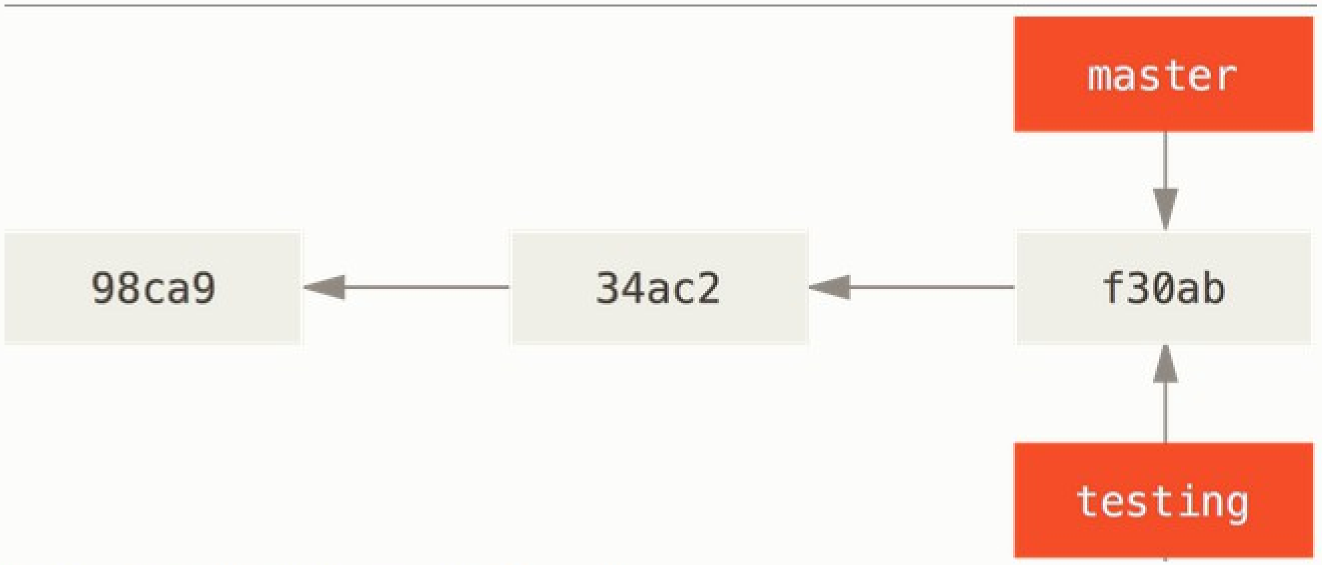
After three commits



Commits and their parents

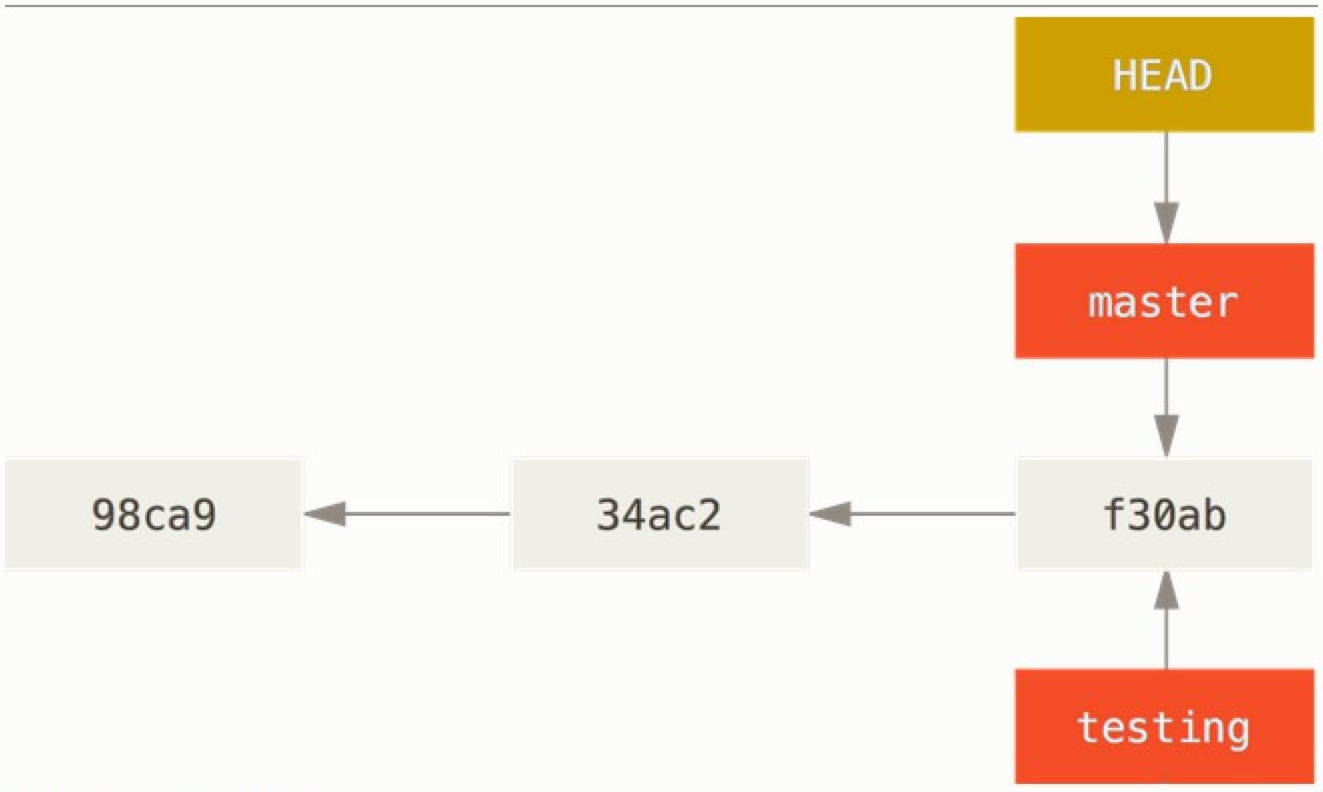
Create a new branch

```
$ git branch testing
```



Two branches pointing into the same series of commits

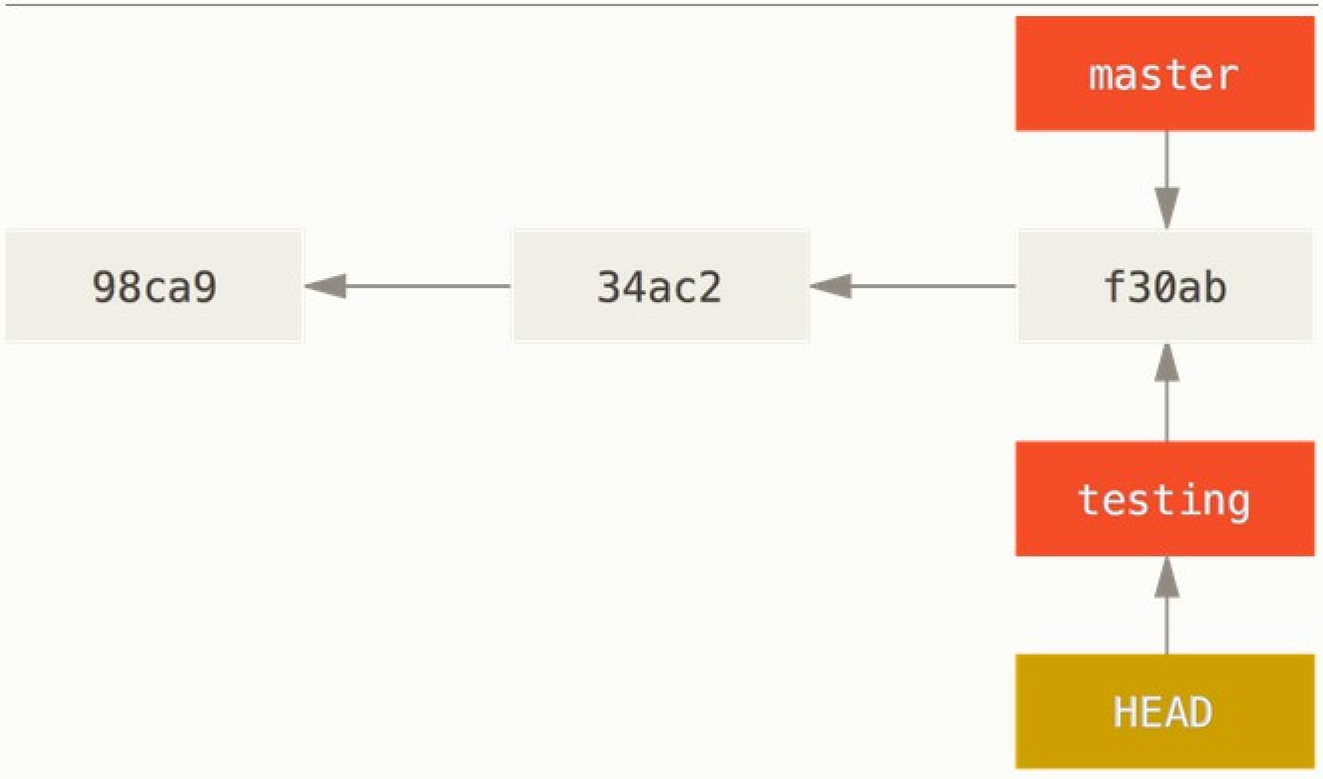
HEAD



HEAD pointing to a branch

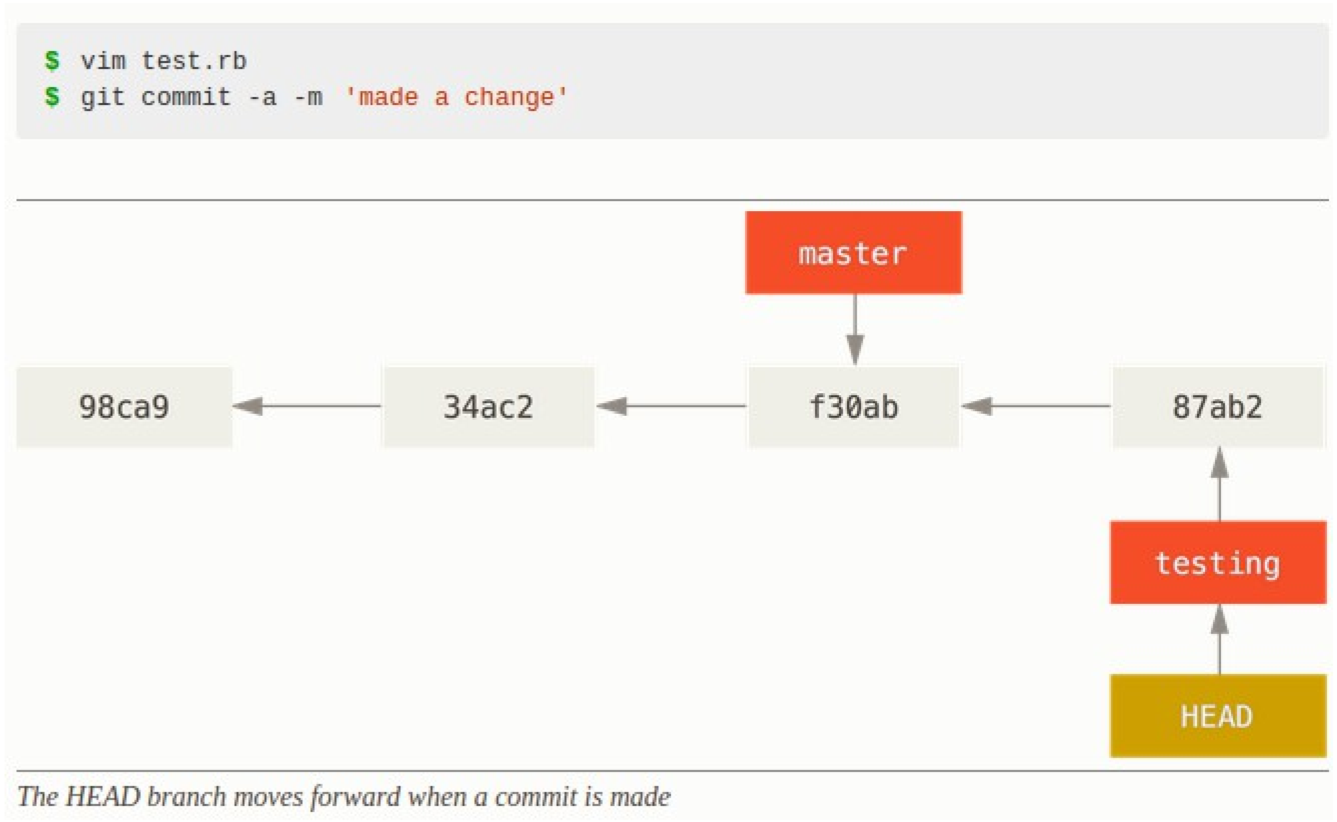
Checking out a branch

```
$ git checkout testing
```



HEAD points to the current branch

Adding to branch

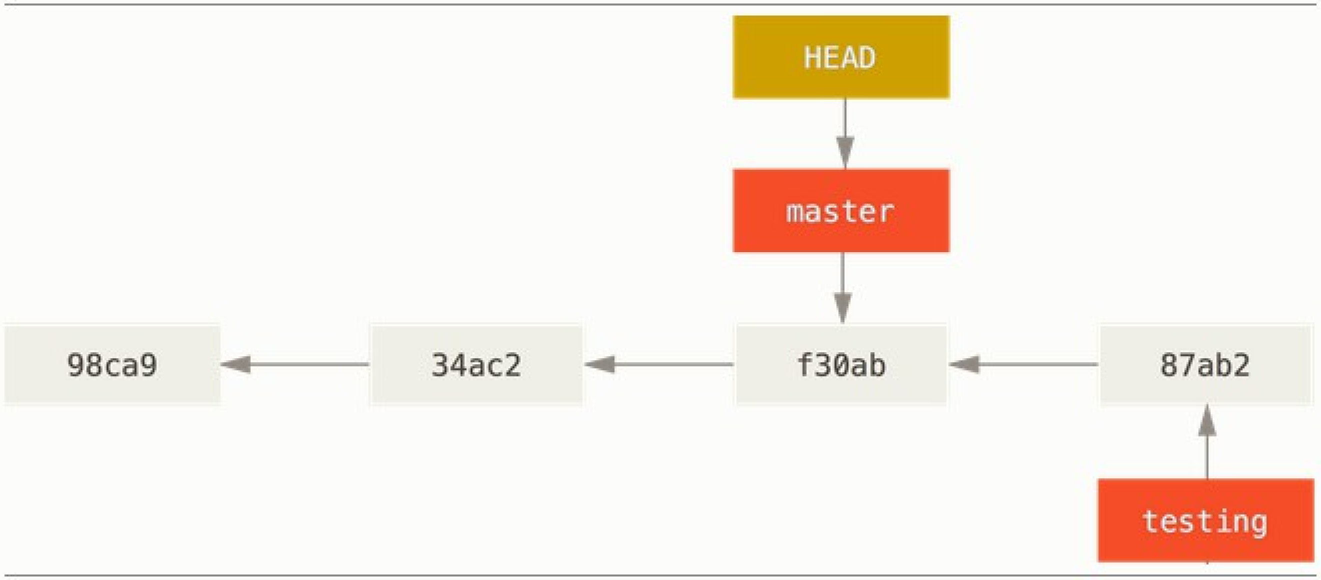


```
$ vim test.rb
$ git commit -a -m 'made a change'
```

The HEAD branch moves forward when a commit is made

Checking out Master

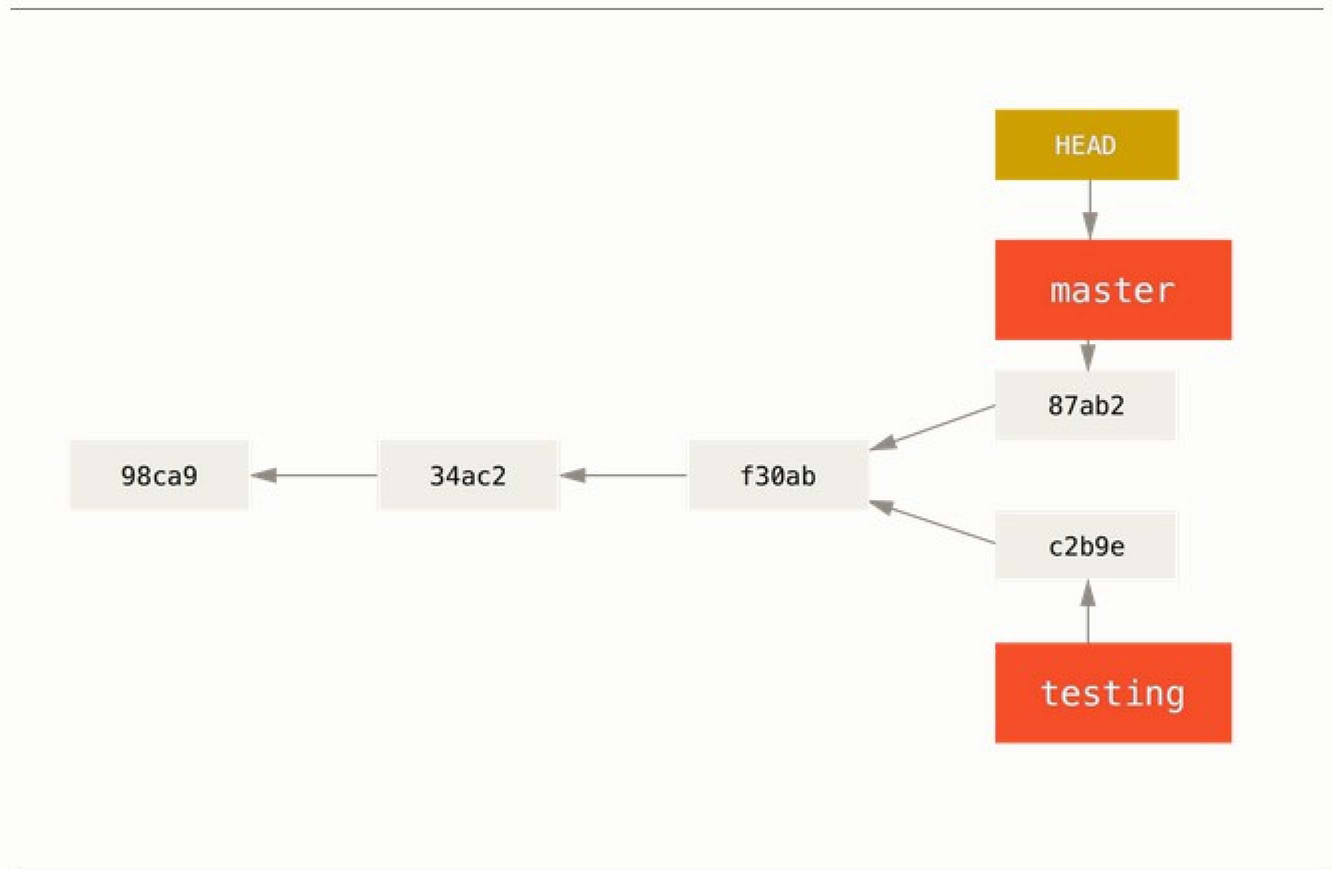
```
$ git checkout master
```



HEAD moves when you checkout

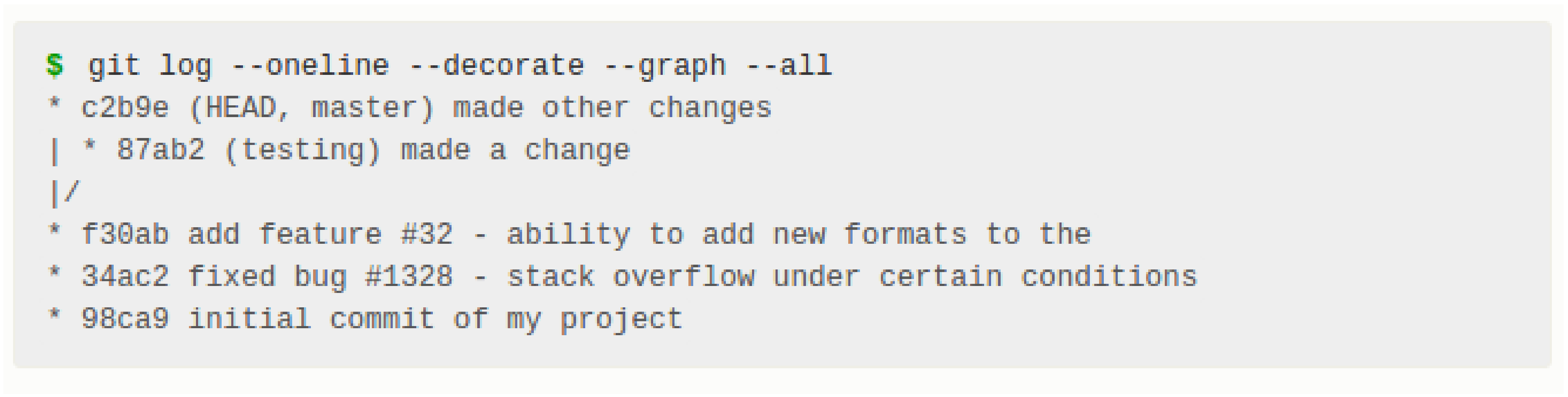
Branch splitting off

```
$ vim test.rb
$ git commit -a -m 'made other changes'
```



Divergent history

Seeing branch split in Git

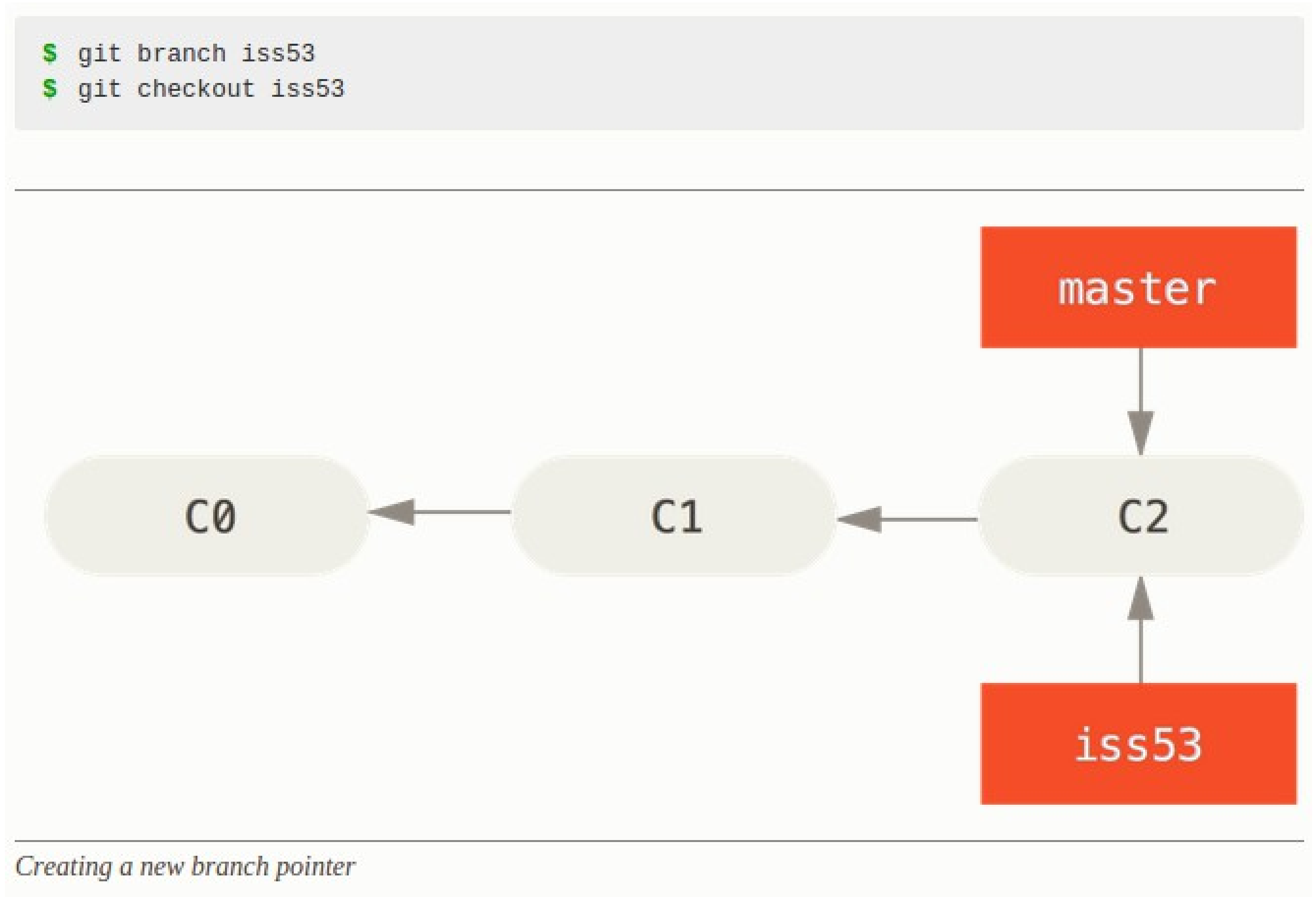


```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) made other changes
| * 87ab2 (testing) made a change
|/
* f30ab add feature #32 - ability to add new formats to the
* 34ac2 fixed bug #1328 - stack overflow under certain conditions
* 98ca9 initial commit of my project
```

Merging Branches in Git

- Branches need to be merged because different team members work on different things.
- Git can manage merges most of the time.
- Occasionally, you will need to tell which of the changes you want to save.

Create a new branch



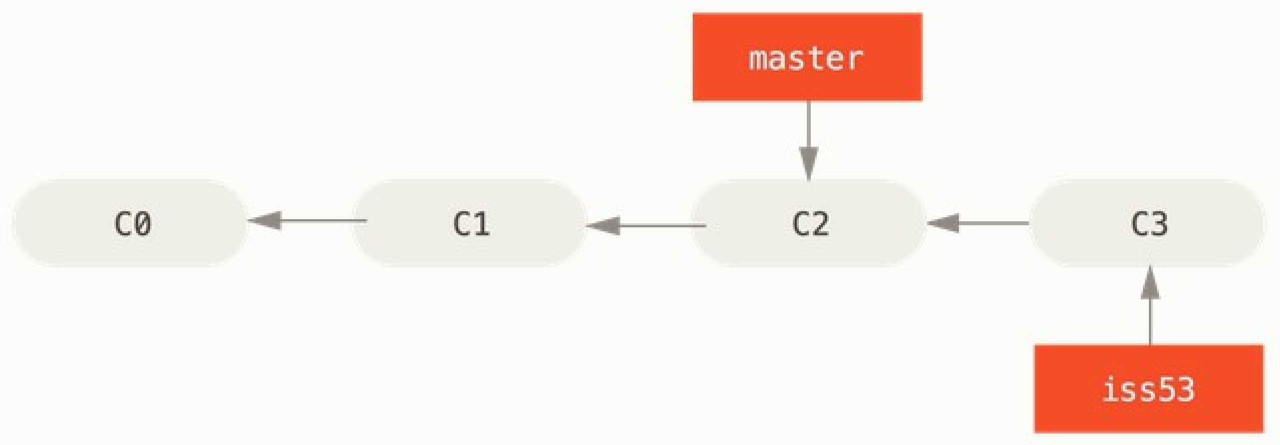
```
$ git branch iss53
$ git checkout iss53
```

$$\begin{tikzcd} & & \text{master} \arrow[d] \\ C0 & C1 \arrow[l] & C2 \arrow[l] \\ & & \text{iss53} \arrow[u] \end{tikzcd}$$

Creating a new branch pointer

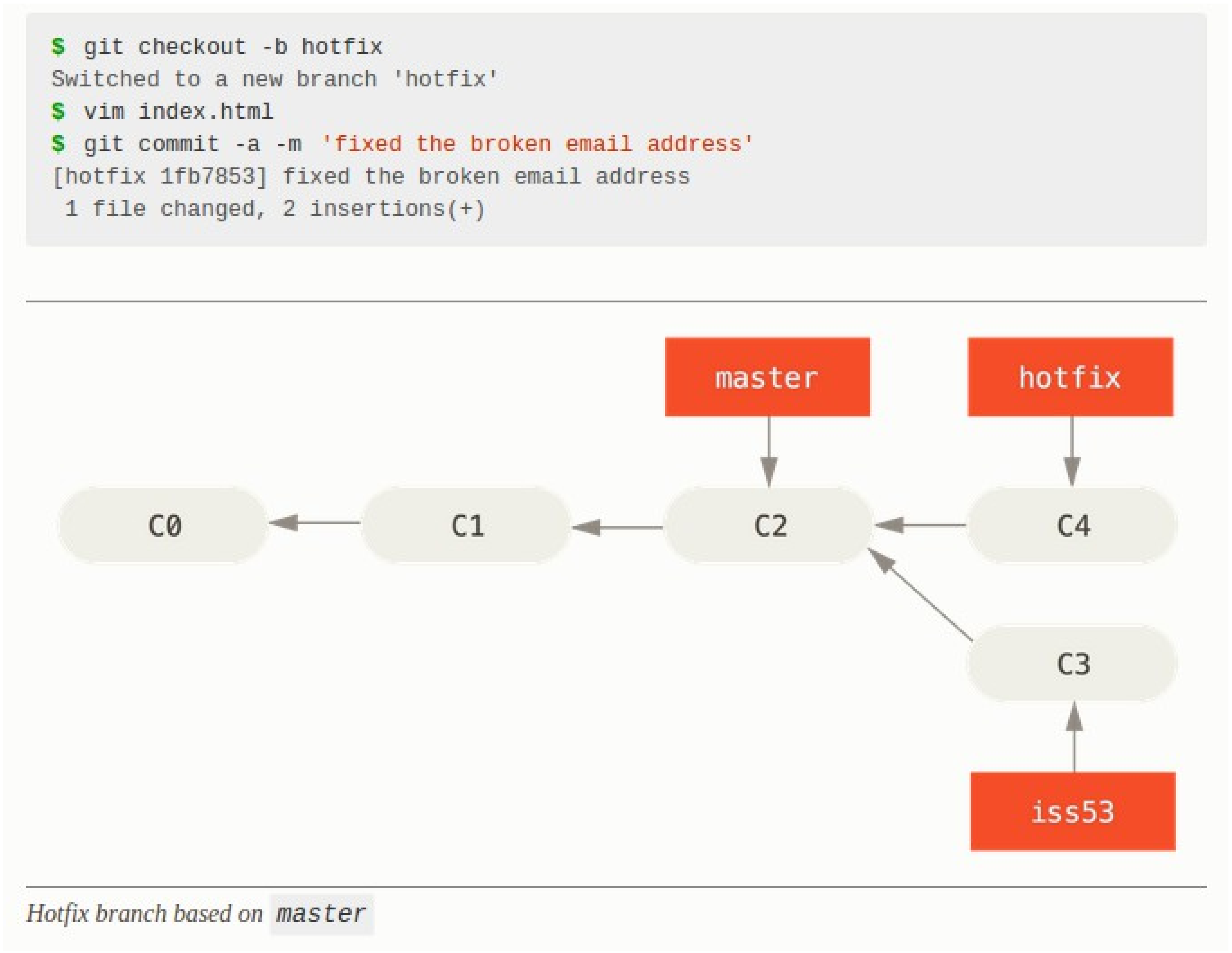
Work starts on ISS53

```
$ vim index.html
$ git commit -a -m 'added a new footer [issue 53]'
```



The iss53 branch has moved forward with your work

From Master add Hotfix

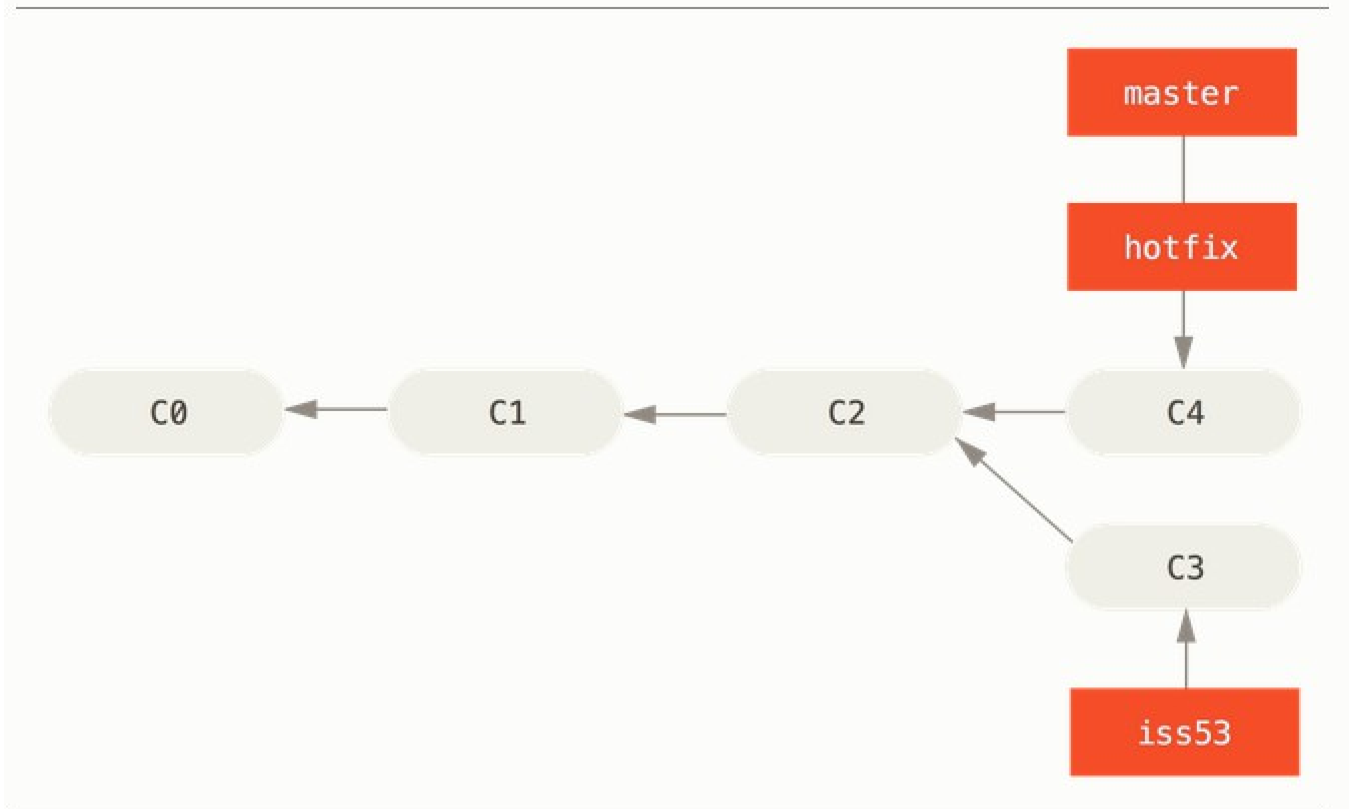


```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'fixed the broken email address'
[hotfix 1fb7853] fixed the broken email address
 1 file changed, 2 insertions(+)
```

Hotfix branch based on `master`

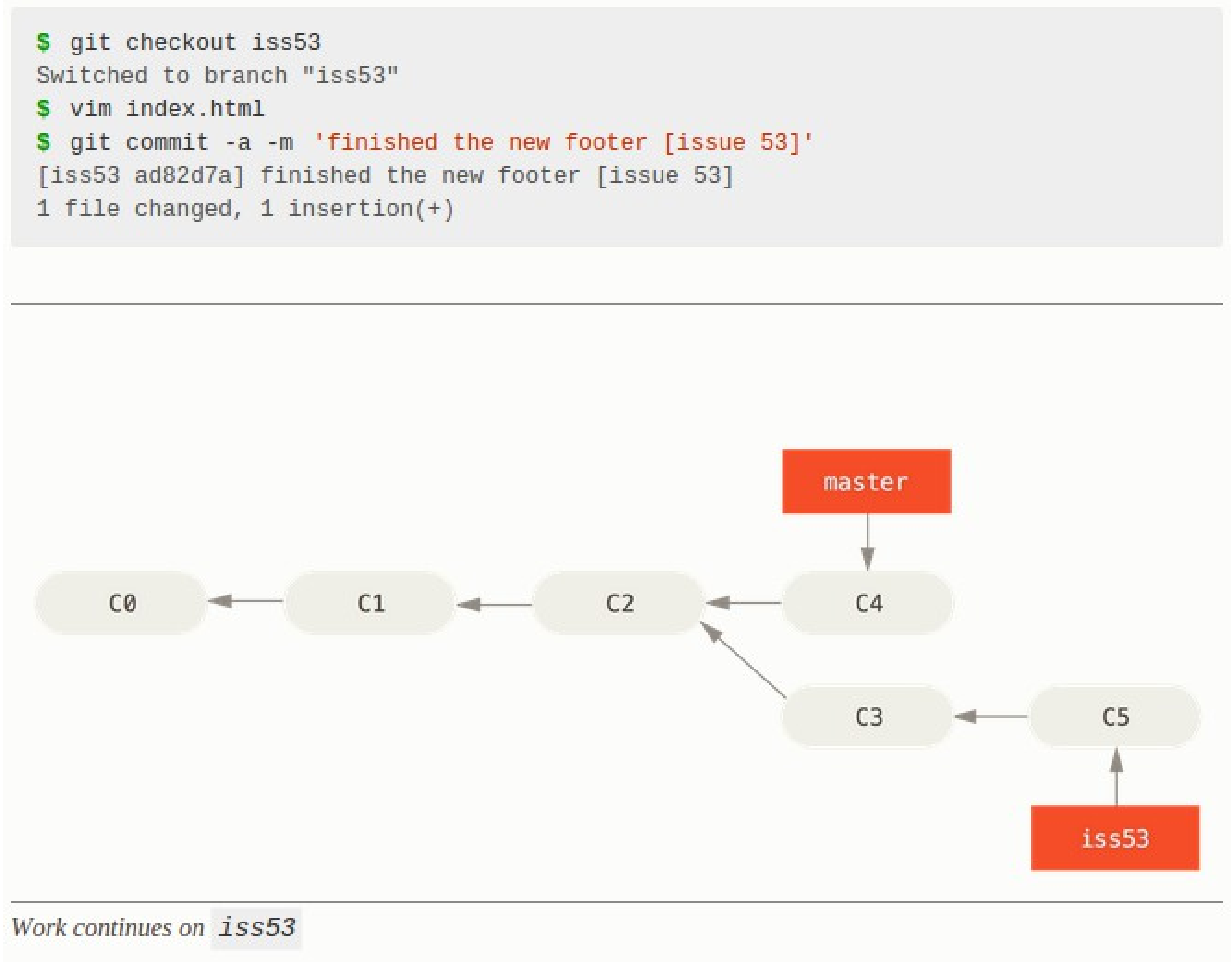
Merge Master and Hotfix

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```



`master` *is fast-forwarded to* `hotfix`

Delete Hotfix; Go back to ISS53

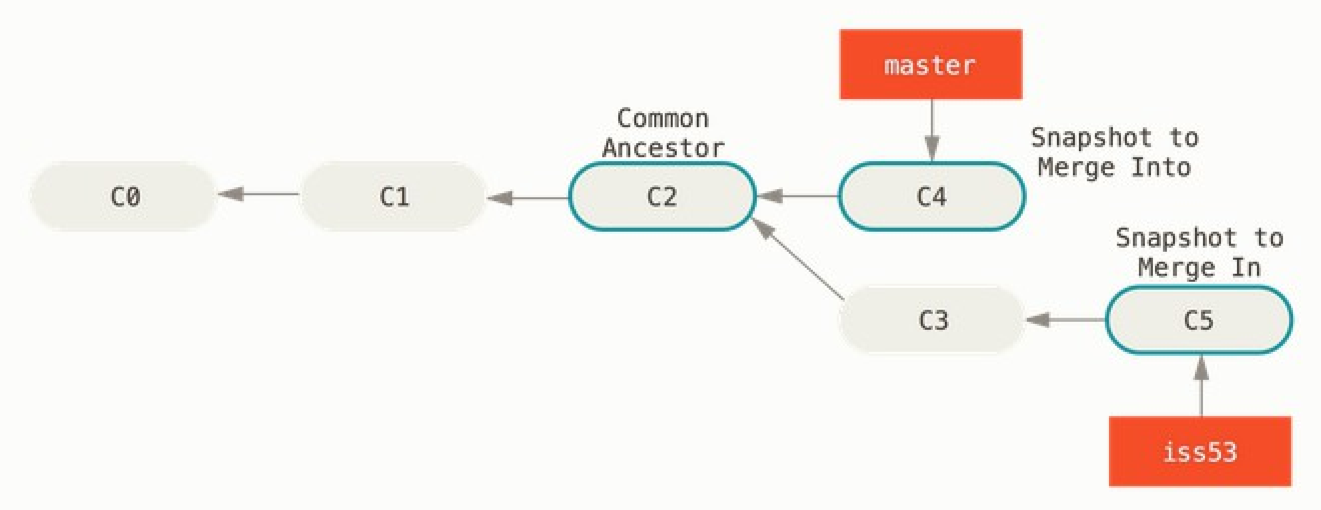


```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'finished the new footer [issue 53]'
[iss53 ad82d7a] finished the new footer [issue 53]
1 file changed, 1 insertion(+)
```

Work continues on `iss53`

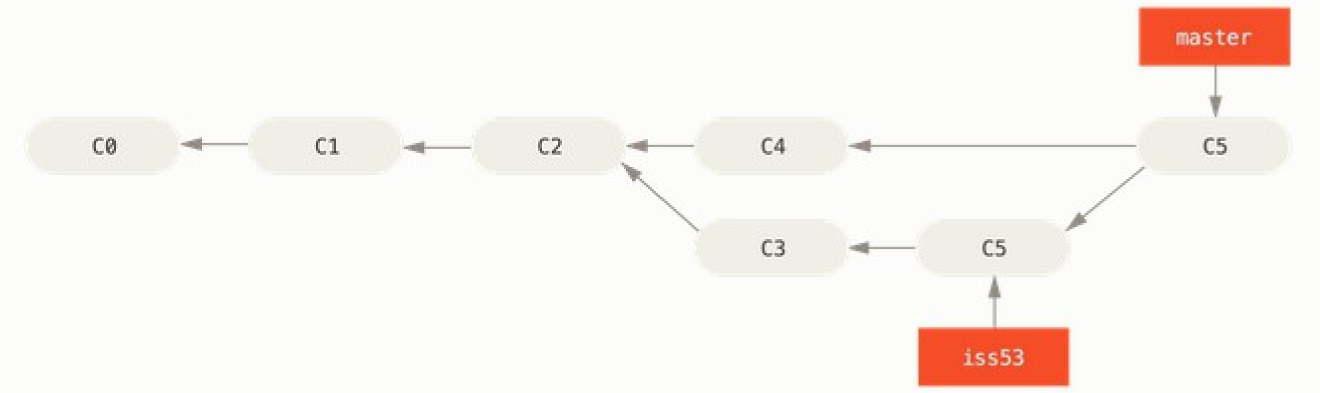
Merge Master and ISS53

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```



Three snapshots used in a typical merge

After the Merge

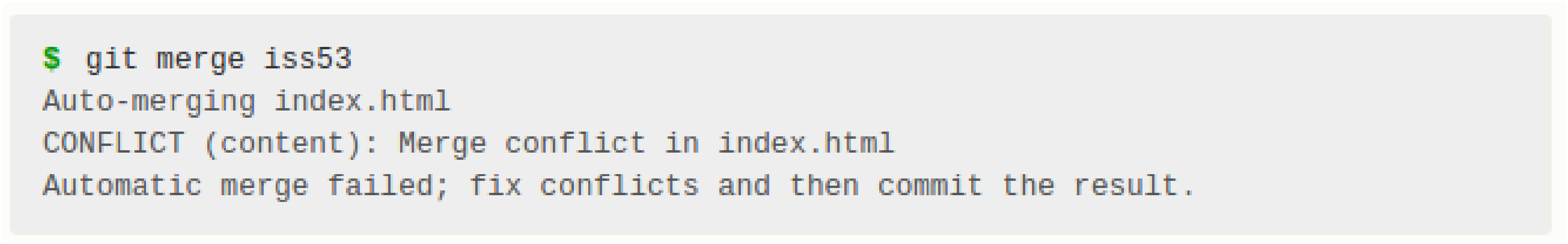


A merge commit

Conflicts

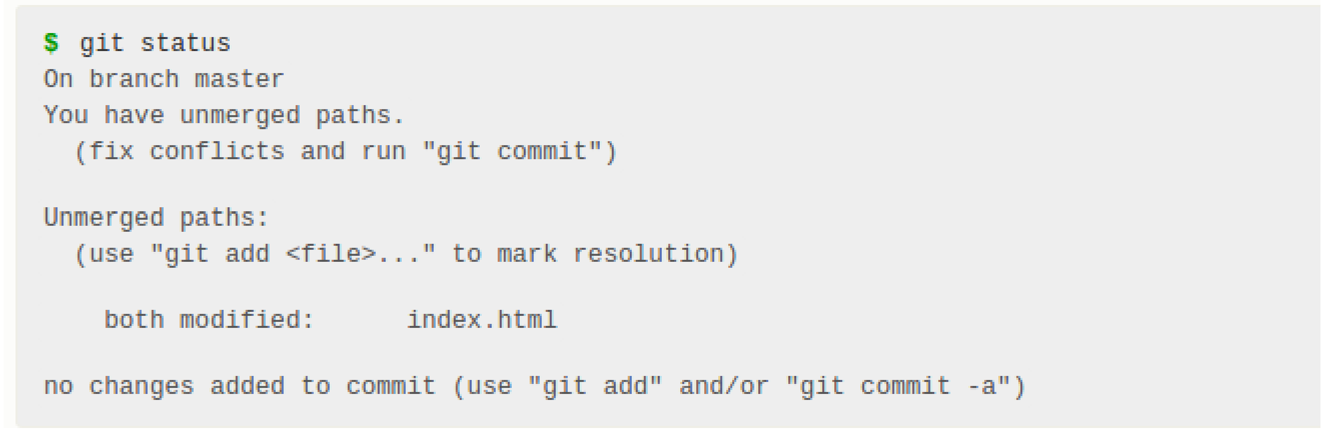
- If you changed the same place in two different ways, Git needs you to tell which change you want to keep.
- This is called *resolving* a *conflict*.

Message indicating conflict



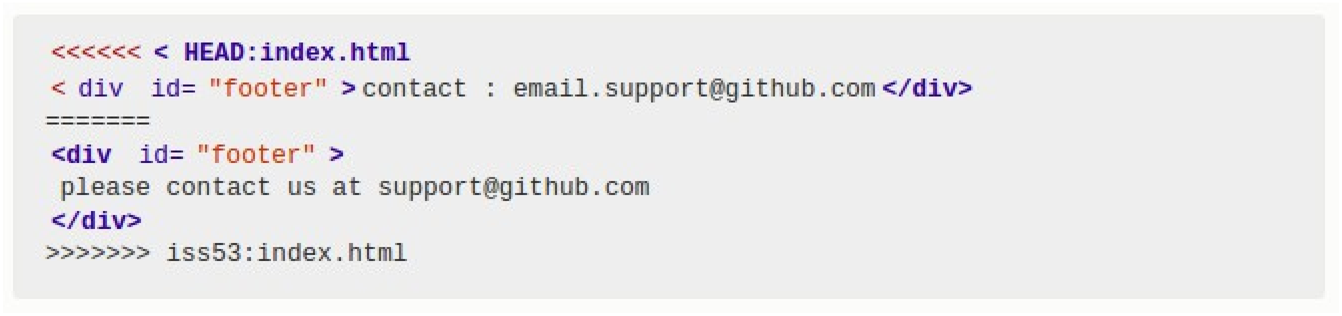
```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

See Conflict with Status



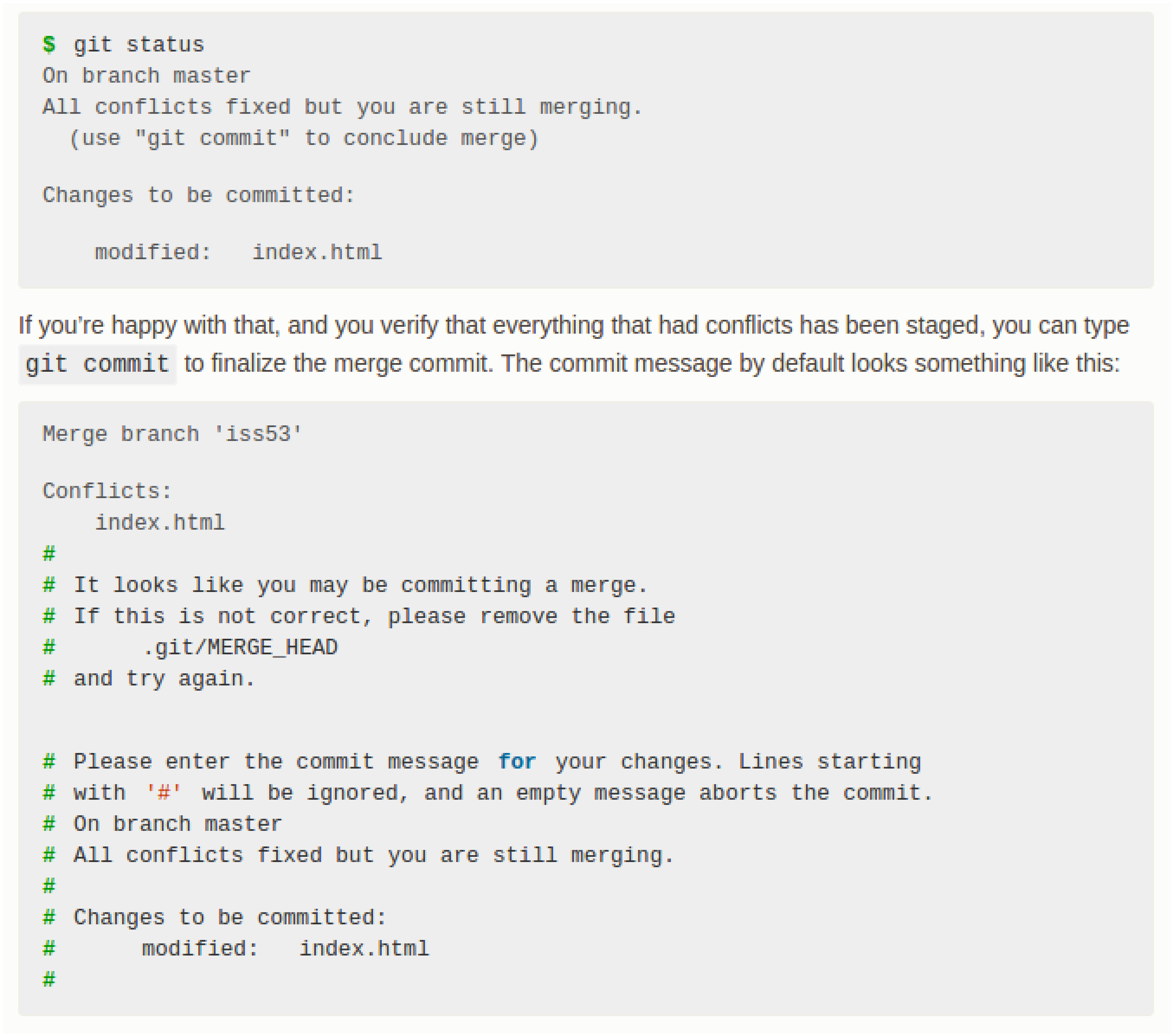
```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

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

Files point to conflict



```
<<<<<< < HEAD:index.html
< div  id= "footer" >contact : email.support@github.com</div>
=======
<div  id= "footer" >
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

Fix conflict and commit



```
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

    modified:   index.html
```

If you're happy with that, and you verify that everything that had conflicts has been staged, you can type `git commit` to finalize the merge commit. The commit message by default looks something like this:

```
Merge branch 'iss53'

Conflicts:
    index.html
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   index.html
#
```