



# Branching and merging with Git

# Branches in Git

We've said before that Git enables collaboration but we haven't really got into how.

**Branches** are one of the features of Git that makes collaboration easier.

# Branches in Git

You can think of a **branch** as a trajectory of work in the project.

When you `git status`, you may have noticed the mention of a branch called **master**:

```
$ git status  
On branch master
```

# Branches in Git

The `master` branch is created by default and, by convention, represents the main line for work in the project.

As you may have guessed, that means that they can be more than one simultaneous line of work in the project. All you have to do is create a new branch.

# Creating a branch

You can create a new branch with the `git branch` command:

```
$ git branch newstuff
```

# Creating a branch

You can then run `git branch` without the branch's name to list all the branches:

```
$ git branch newstuff
```

```
$ git branch
```

```
*master
```

```
newstuff
```

# Creating a branch

```
$ git branch newstuff
```

```
$ git branch  
*master  
newstuff
```

Notice that `*` before the `master` branch. Even though we created the new branch, our repo is still on the `master` branch.

# Switching to a branch

To switch to the `newstuff` branch we use the `git checkout` command:

```
$ git checkout newstuff  
Switched to branch newstuff
```

```
$ git branch  
master  
*newstuff
```



# Switching to a branch

You can switch back and forth between branches at will.

```
$ git checkout master  
Switched to branch 'master'
```

```
$ git checkout newstuff  
Switched to branch 'newstuff'
```

# The nature of a branch

But what is a branch really? Well, a branch is a **reference** to a specific commit.

Using the `--verbose` option, we can see this more clearly when we list the branches:

```
$ git branch --verbose
master      90180bf Add important README
* newstuff  90180bf Add important README
```

# The nature of a branch

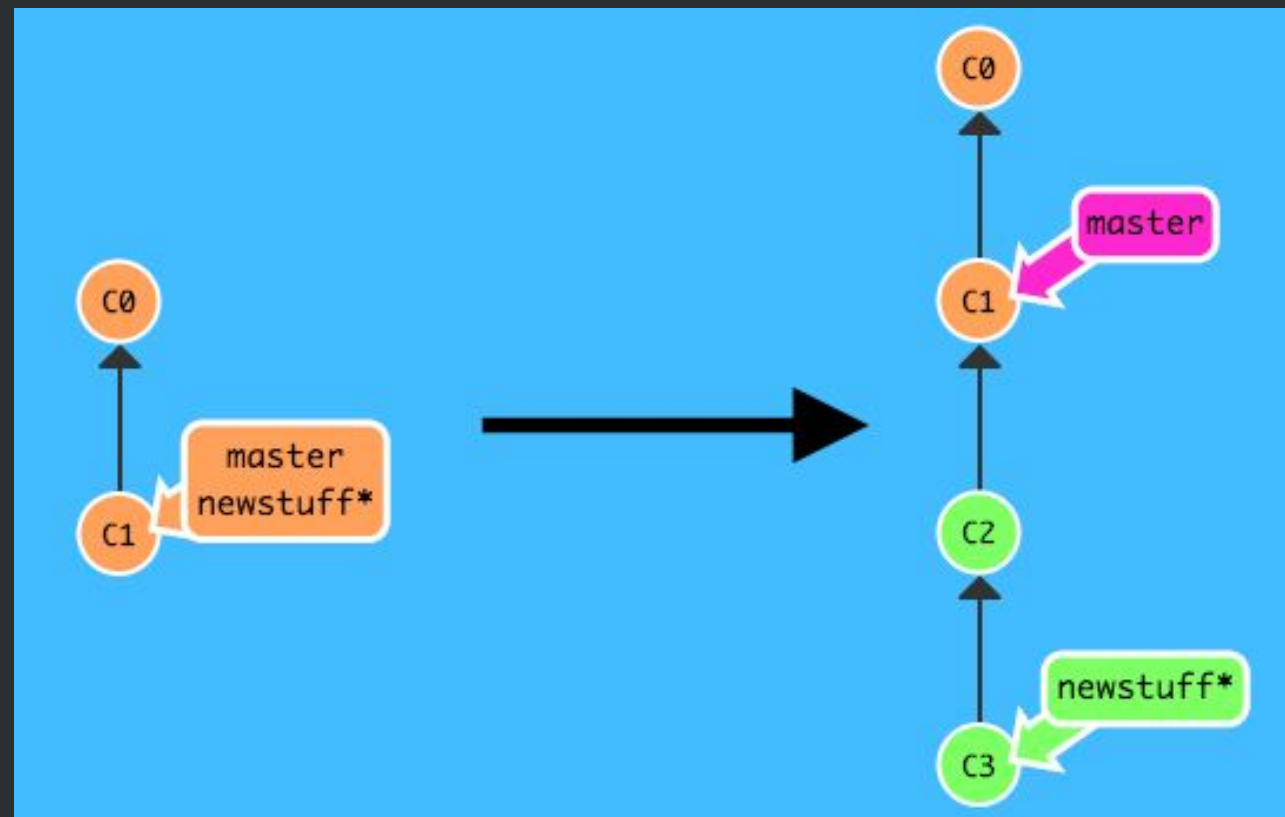
```
$ git branch --verbose  
  master      90180bf Add important README  
* newstuff 90180bf Add important README
```

We can see that both branches are currently pointing to the same commit:

```
$ git status
```

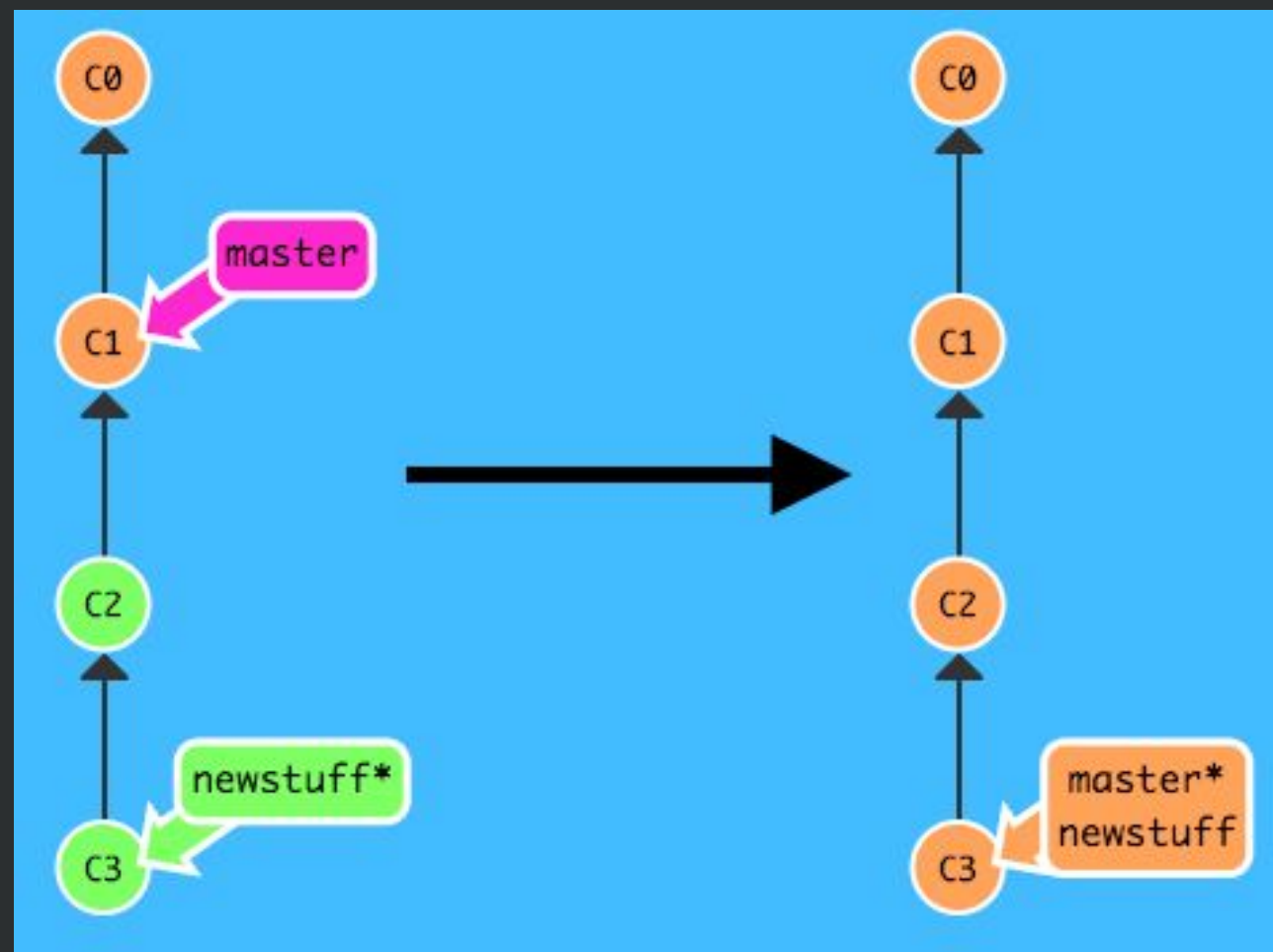
# The nature of a branch

Now that we have two different branches, we can make changes to the `newstuff` branch independently of the `master` branch.



# The nature of a branch

The idea is for that branch to eventually be reconciled back with **master** once the work on it is completed.



# The nature of a branch

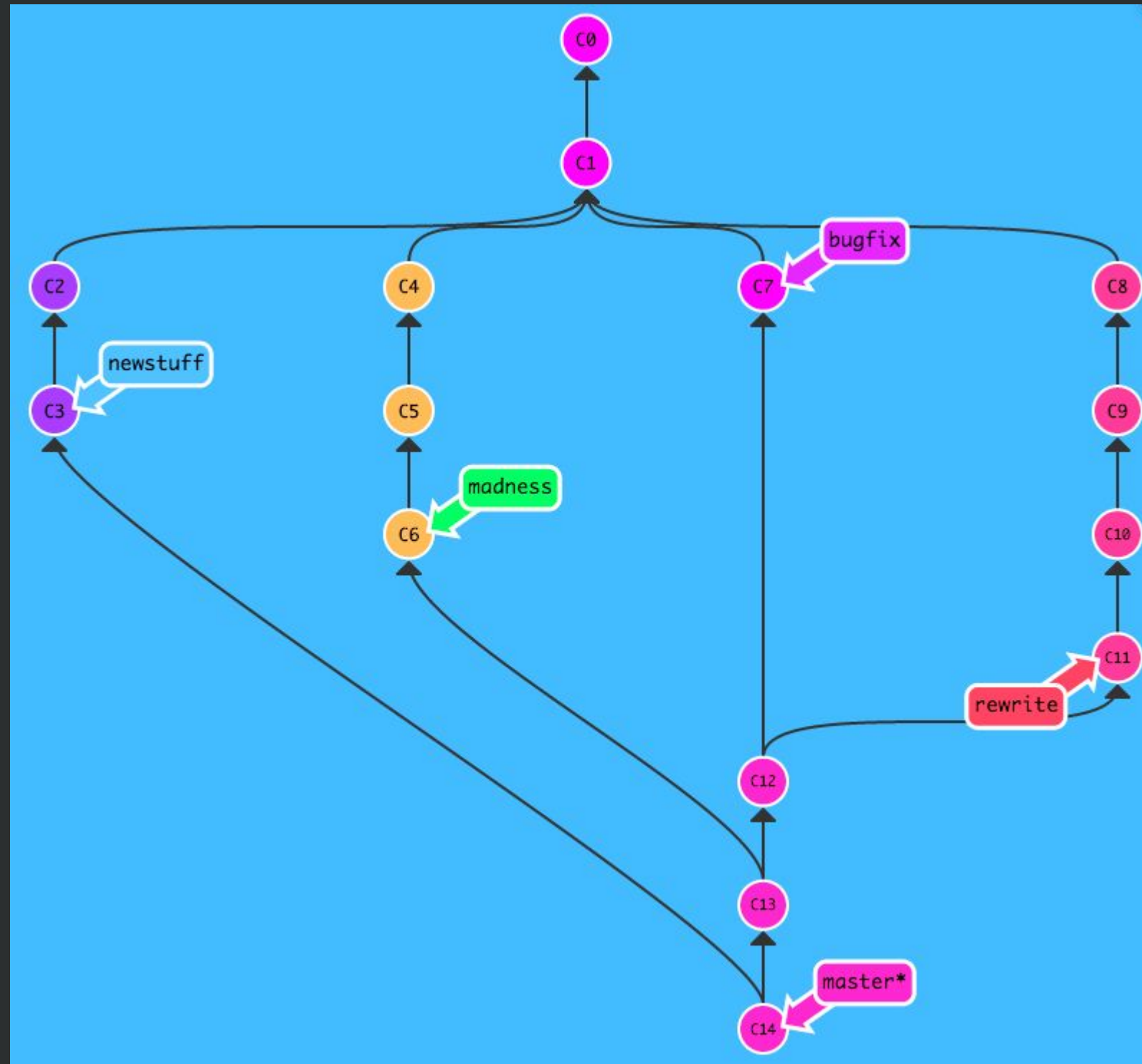
And there can be *a bunch* of branches going at a time. It's common for teams to do this to keep themselves organized.



# The nature of a branch

At one point or another, all of them will be folded back into `master`.

# The nature of a branch





# Merging a branch

The way you reconcile is a branch with `master` is by merging the two branches.

Let's make some commits on the `newstuff` branch to then *merge* them into the `master` branch.

# Developing on a branch

Let's add some more code to our `important_program.rb`:

```
class ImportantThing
  def initialize(thing)
    @thing = thing
  end

  def thing
    return "You can't have '#{@thing}'. It's
too important. I swear it really is"
  end
end

pizza = ImportantThing.new("pizza")
puts pizza.thing
```

# Developing on a branch

Now add and commit those changes:

```
$ git add important_program.rb  
  
$ git commit -m "Add ImportantThing class"  
[newstuff 8149249] Add ImportantThing  
class  
1 file changed, 13 insertions(+)
```

# Developing on a branch

```
$ git add important_program.rb  
  
$ git commit -m "Small change to  
ImportantThing class"  
[newstuff 8149249] Small change to  
ImportantThing class  
1 file changed, 13 insertions(+)
```

Notice how it mentions the `newstuff` branch in the commit result.



# Developing on a branch

```
$ git add important_program.rb  
  
$ git commit -m "Small change to  
ImportantThing class"
```

Now if we use the `--verbose` option (just `-v` for the lazy) to take a look at the branches again we see that they are now on different commits.

```
$ git branch -v  
  master    90180bf Add important README  
* newstuff  8149249 Small change to  
ImportantThing class
```

# Merging a branch

Now we are ready to merge our changes. First thing we need to do is switch to the branch that will receive the merge. In this case it's `master`.

```
$ git checkout master
```

# Merging a branch

Now we can merge with `newstuff`:

```
$ git checkout master

$ git merge newstuff
Updating 90180bf..8149249
Fast-forward
 important_program.rb | 13 ++++++++
1 file changed, 13 insertions(+)
```

Notice that it says *Fast-forward*. We will talk about that in a bit.

# Merging a branch

Let's take another look at the branches. They should both be pointing to the same commit again.

```
$ git branch -v
* master      8149249 Add ImportantThing
              class
  newstuff    8149249 Add ImportantThing
              class
```



# Deleting a branch

Now that the branch is merged, it can safely be deleted. It's important to take this step so that old branches don't accumulate over time.

```
$ git branch --delete newstuff  
Deleted branch newstuff (was 8149249).
```

# Visualizing merges

Since this is hard to picture yourself, let's present branches and merges in a more interactive way.

Let's visit the [Learn Git Branching Sandbox](#)

# More practice with branches

For more practice do some of the levels of [Learn Git Branching](http://pcottle.github.io/learnGitBranching)

<http://pcottle.github.io/learnGitBranching>

Remember kids: ABC

**ALWAYS BE  
COMMITTING**

