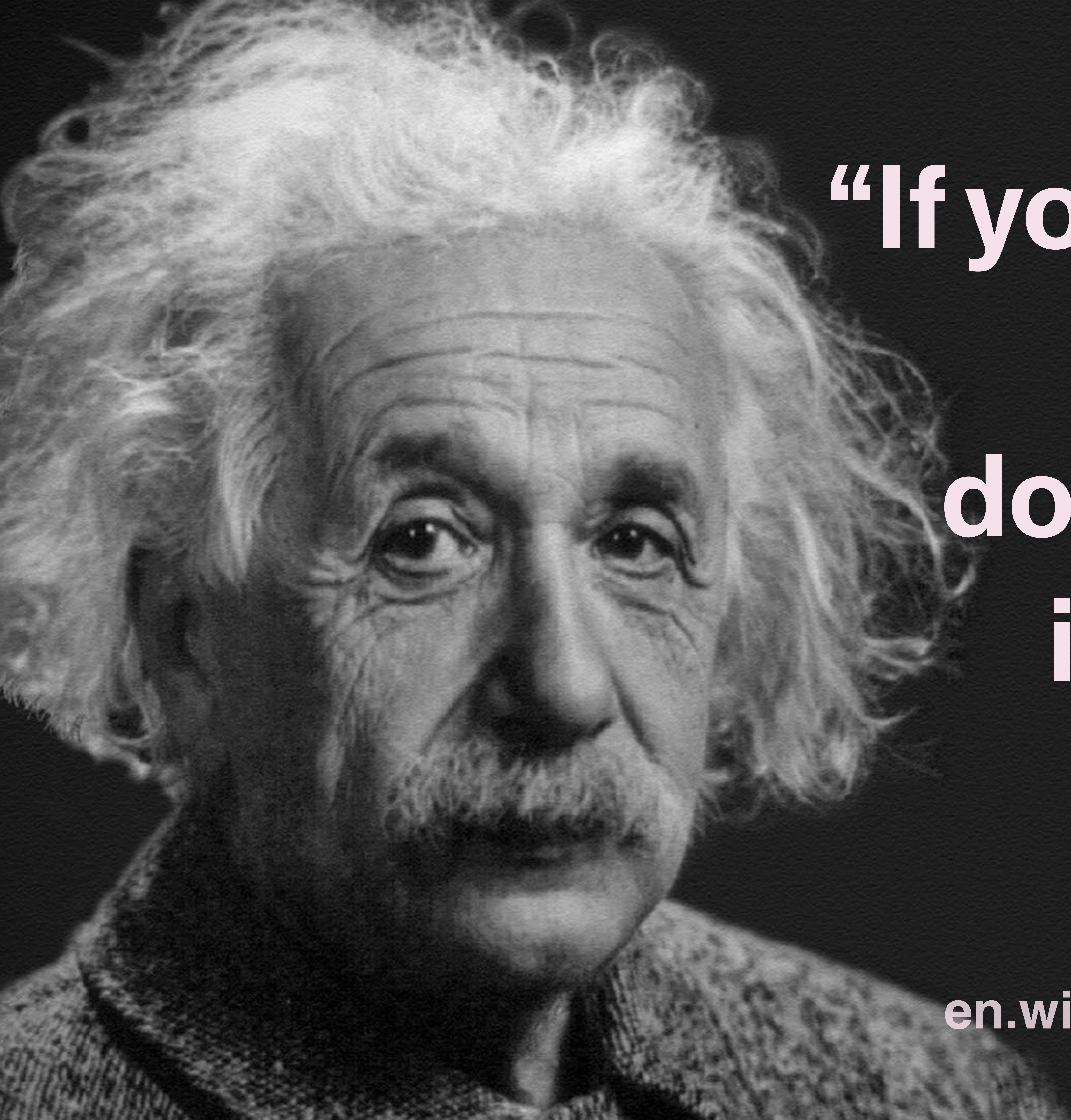
Simplify writing code with deliberate commits

LRUG - June 2019

Joel Chippindale - @joelchippindale

UNMADE

Joel Chippindale
CTO at
Unmade



“If you can’t explain it simply, you don’t understand it well enough.”

- Albert Einstein (probably)
en.wikiquote.org/wiki/Albert_Einstein

Our work requires us to make changes to complex systems

Image by Cory Doctorow (CC BY-SA)

We learn to break down complex problems into small, simple steps

Great developers are really good at breaking problems into simple steps

Image by WOCinTech Chat (CC BY)

This takes discipline and willpower…

...but we can learn to make it easier

When I started using git it changed the way that I developed software

5 Practices

1. Plan your commits
2. Use single purpose branches
3. Make atomic commits
4. Write good commit messages
5. Rewrite your history to tell a story (early and often)

Practice 1:

Plan your commits

Image by Alexander Baxevanis (CC BY)

Make a plan for the commits that you will make

What if you don't know enough yet to make a plan?

What if the plan changes?

Plan your commits ahead of time and re-plan when you need to

Practice 2:

Use single purpose branches

Image by Jon Bennet

Name your branch to reflect it's (single) purpose

Notice when you are starting to work on something else

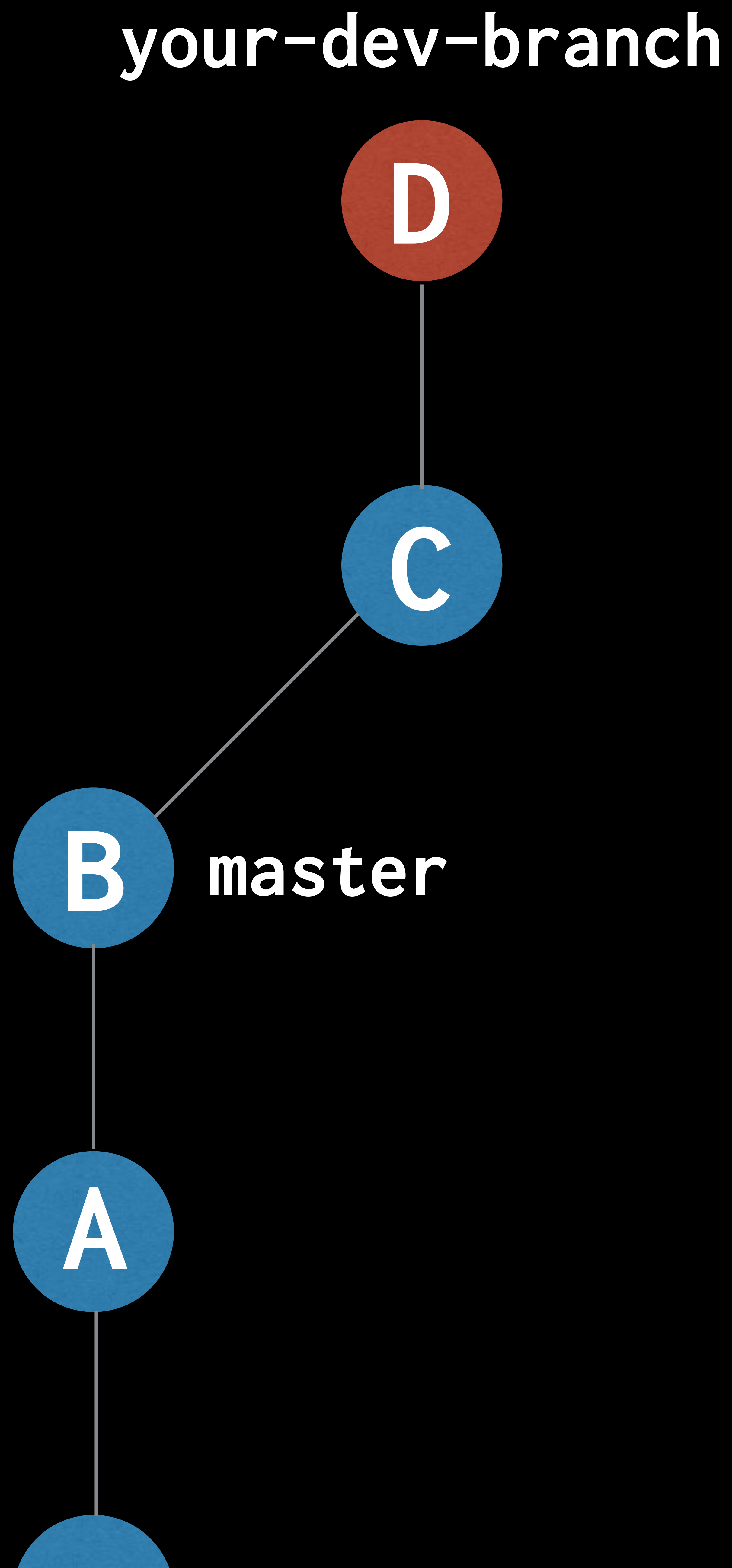
Notice if a commit has value independent of the branch

...and if it does, then 'cherry pick' it onto master

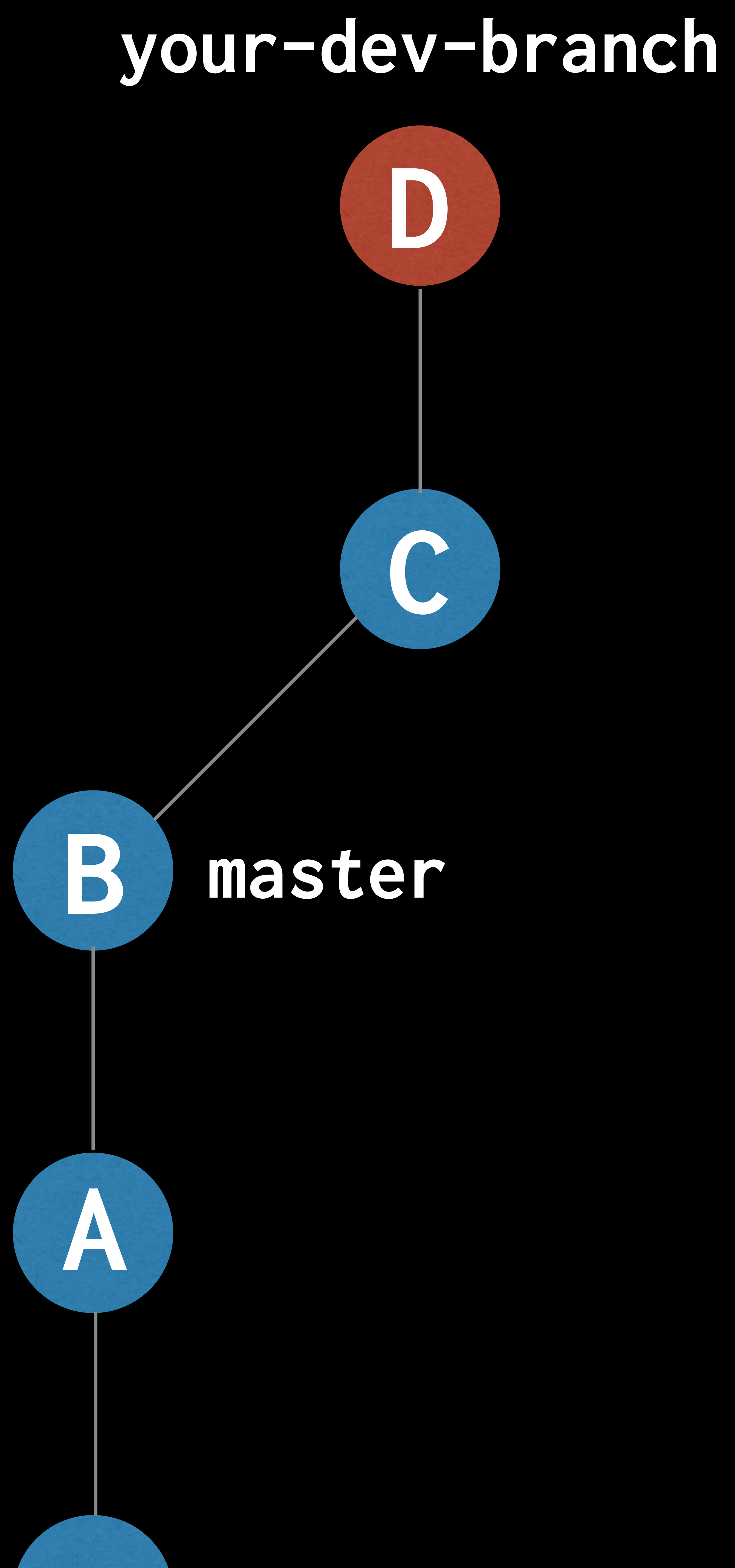
```
git cherry-pick

# Enables you to apply a single
# commit from another branch
# onto master
```

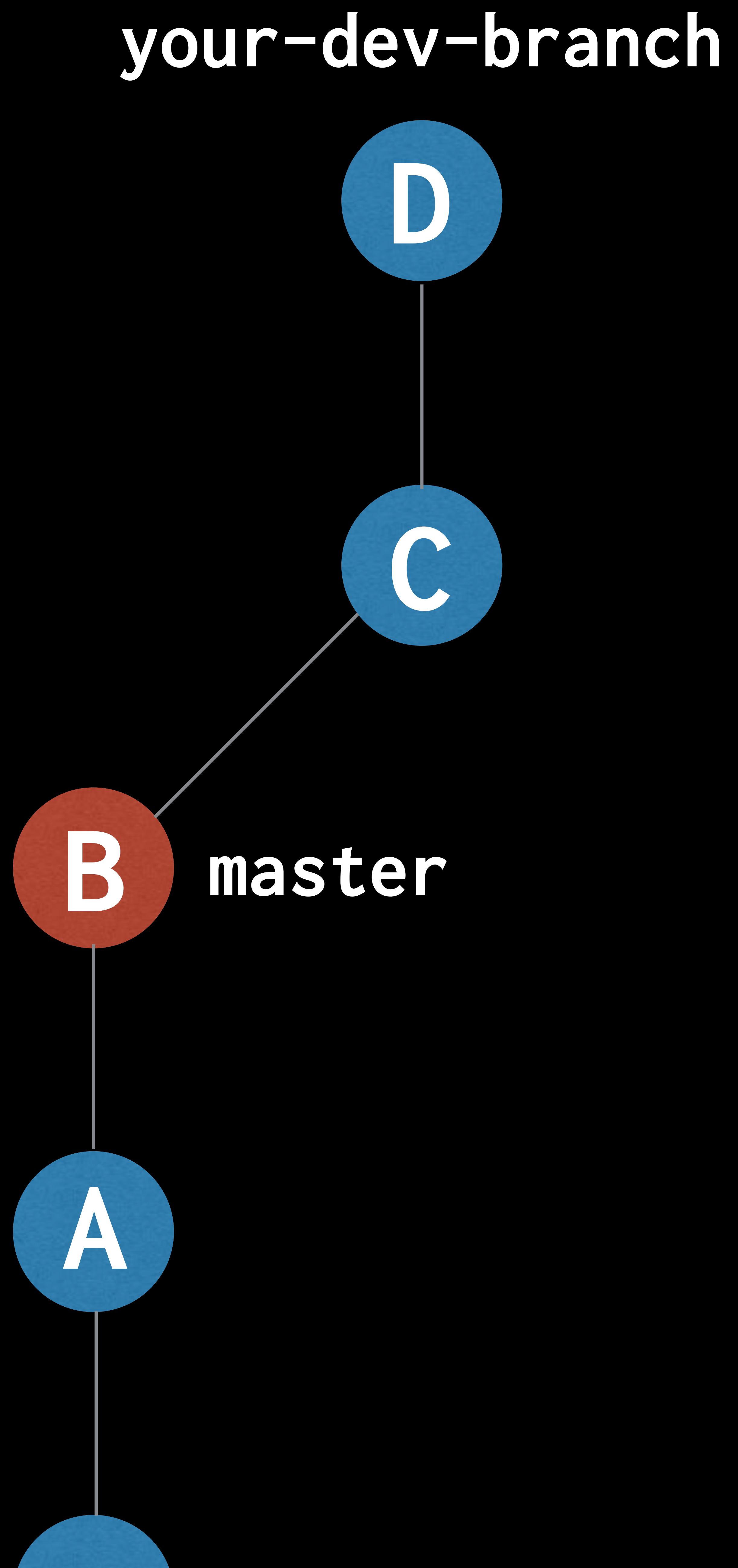
```
$
```


your-dev-branch
D
C
B
master
A

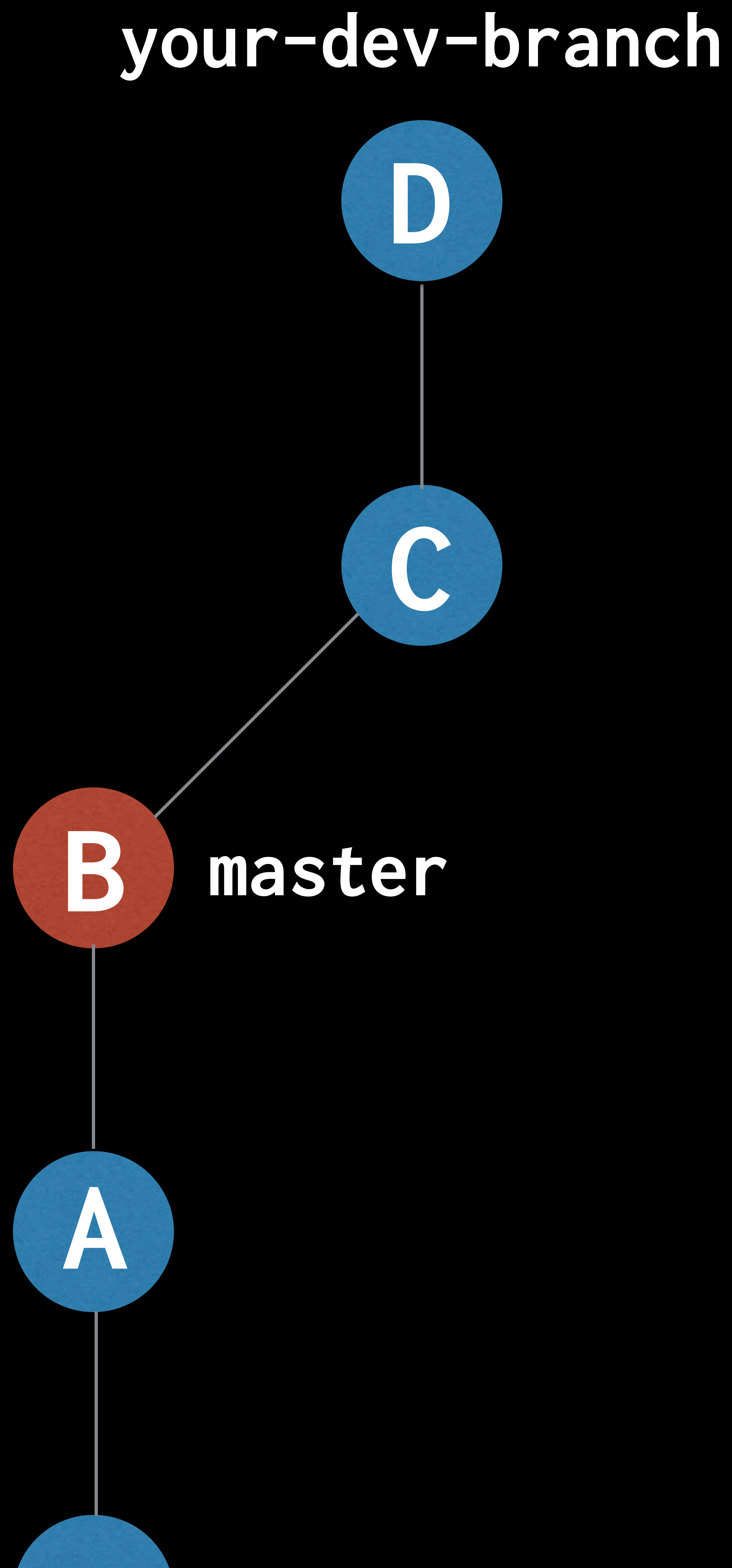
```
$ git checkout master
```



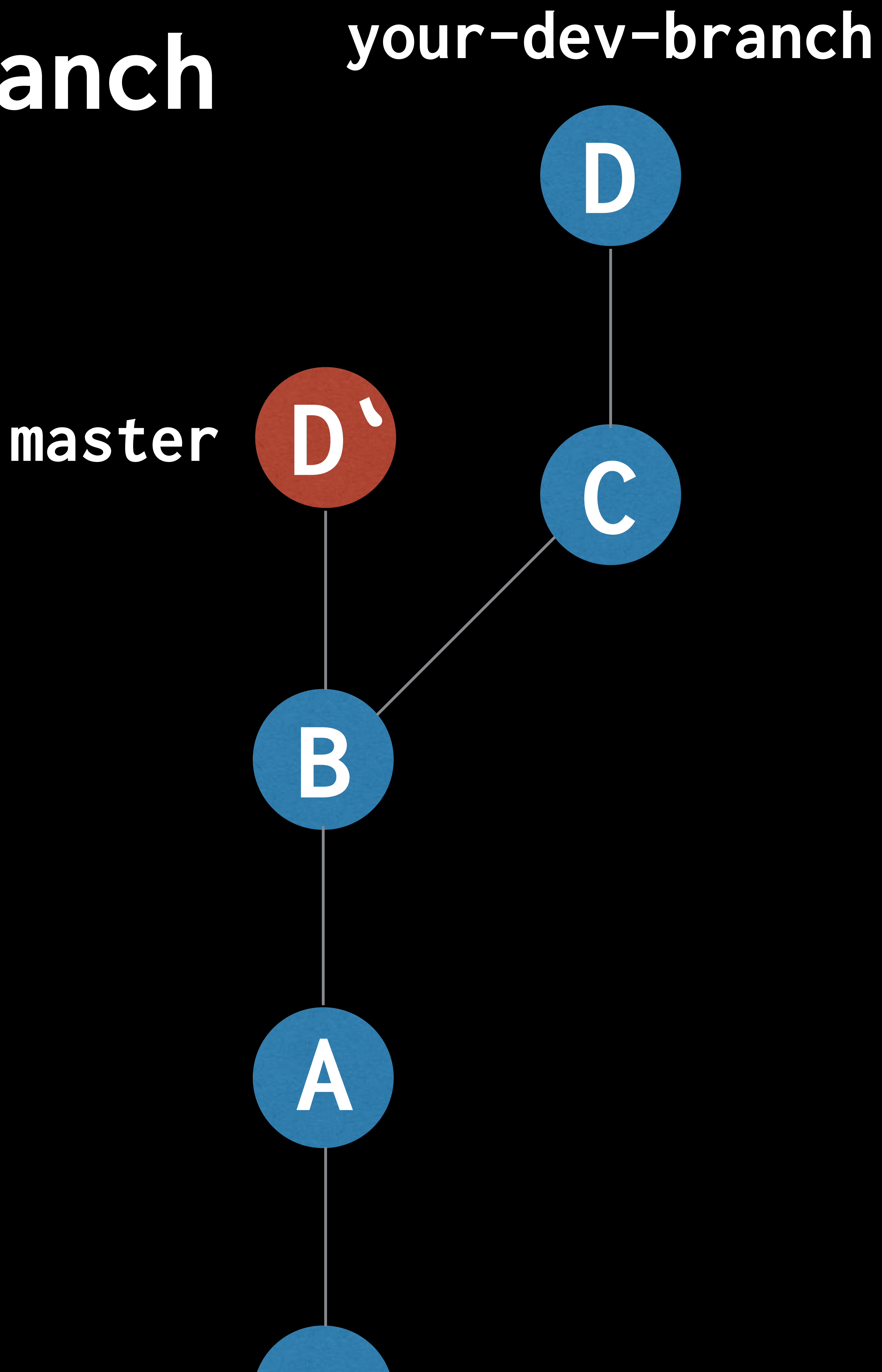
```
$ git checkout master
```



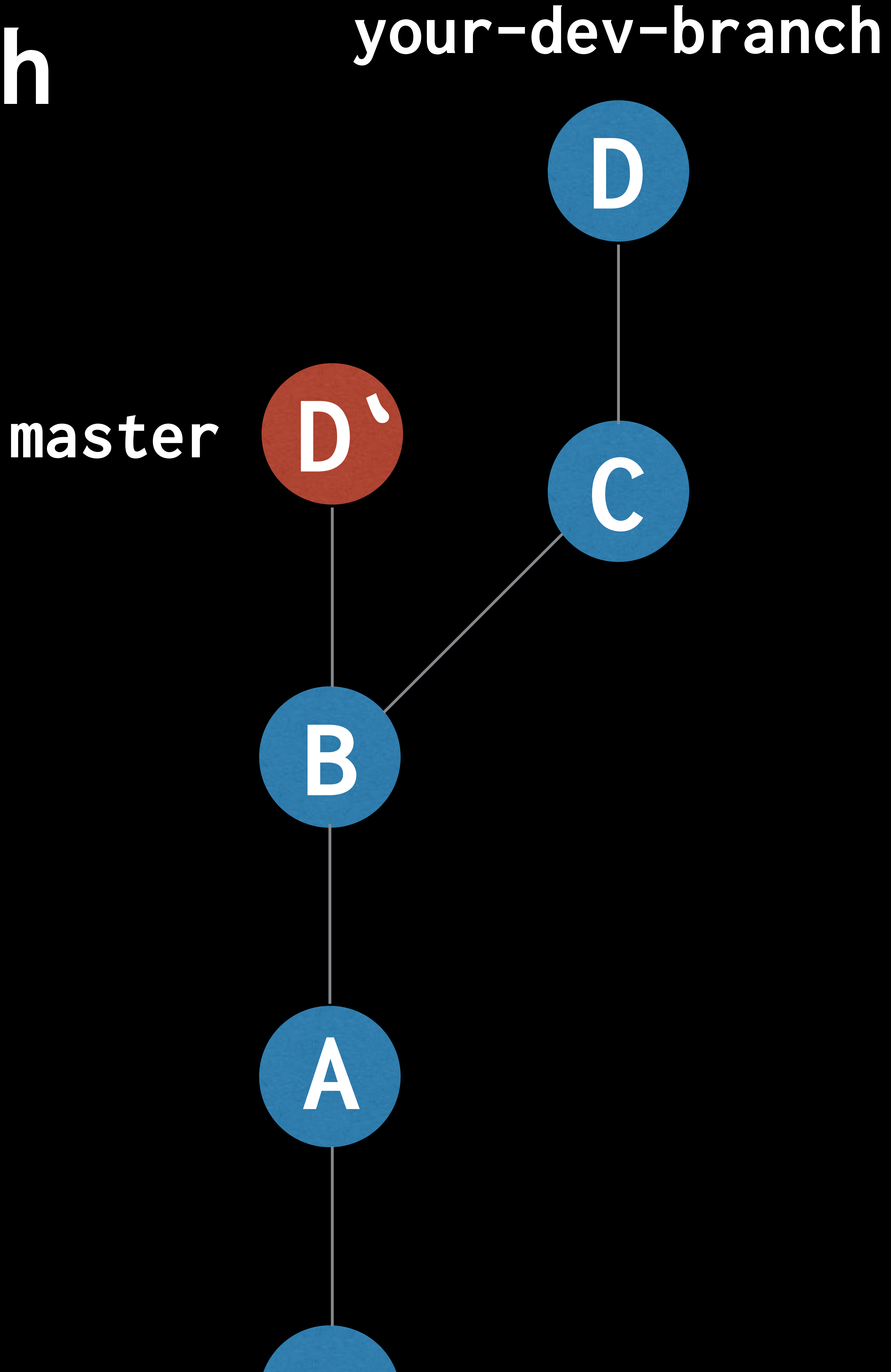
```
$ git cherry-pick your-dev-branch
```



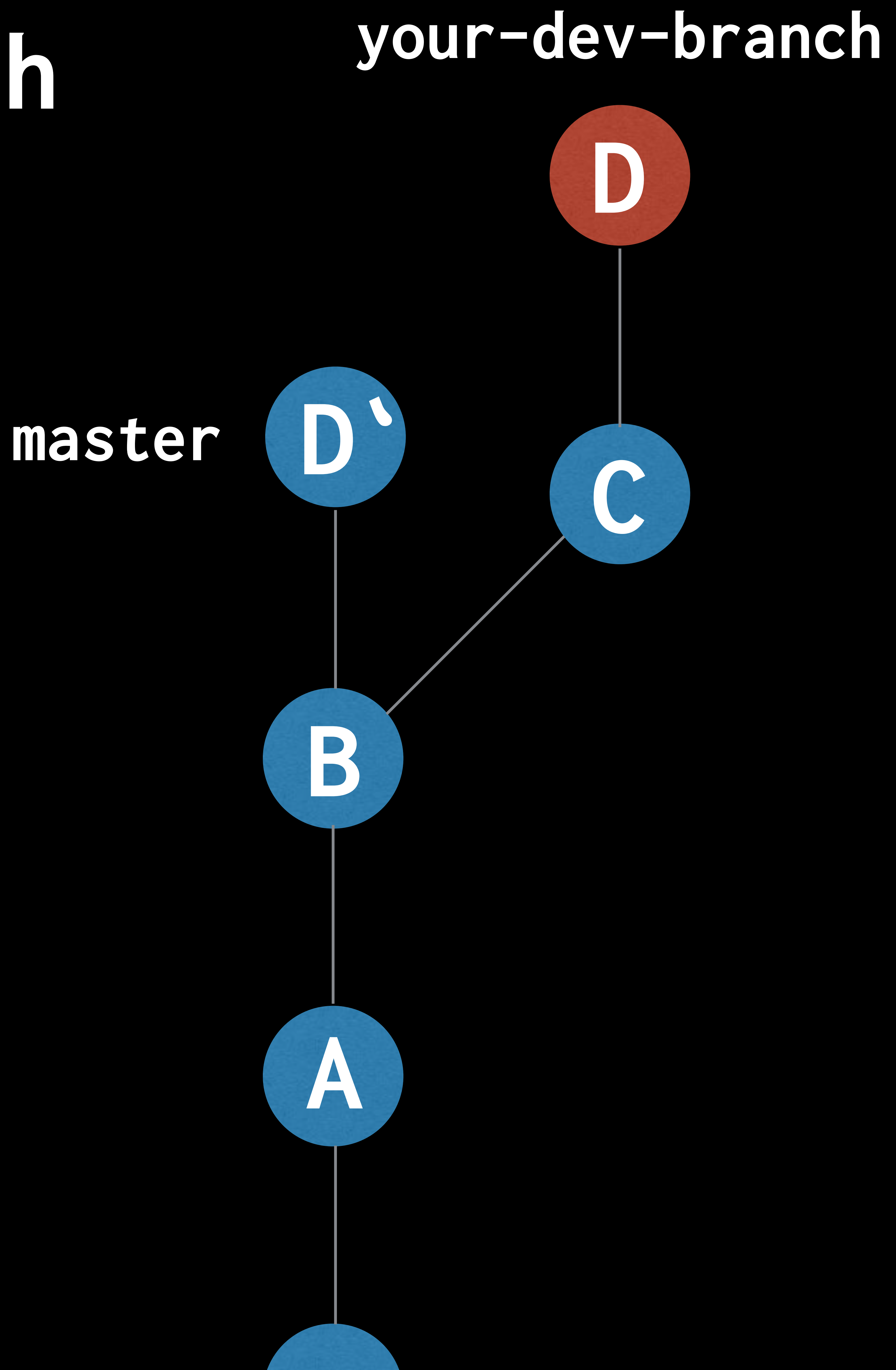
```
$ git cherry-pick your-dev-branch
```



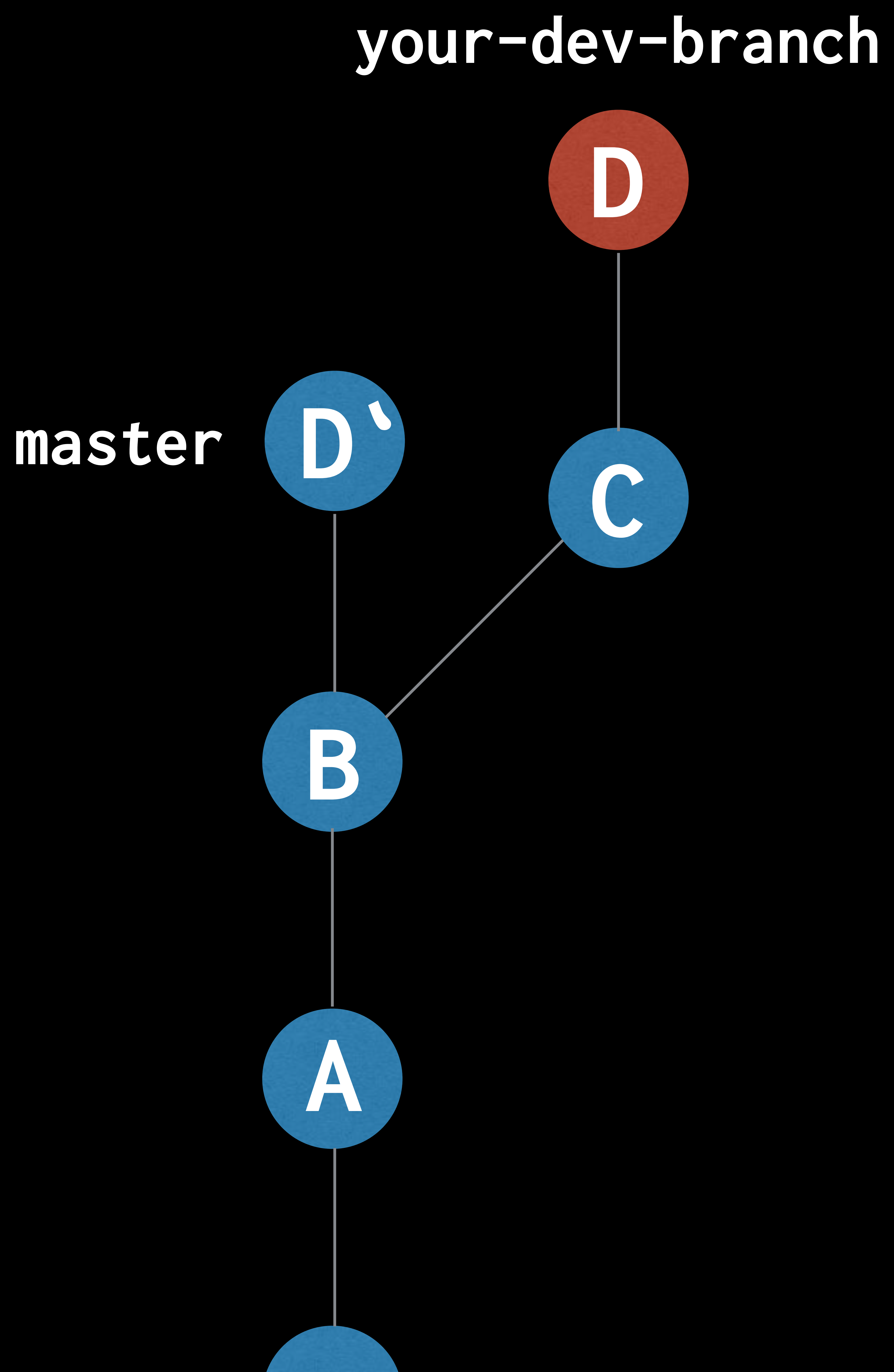
```
$ git checkout your-dev-branch
```



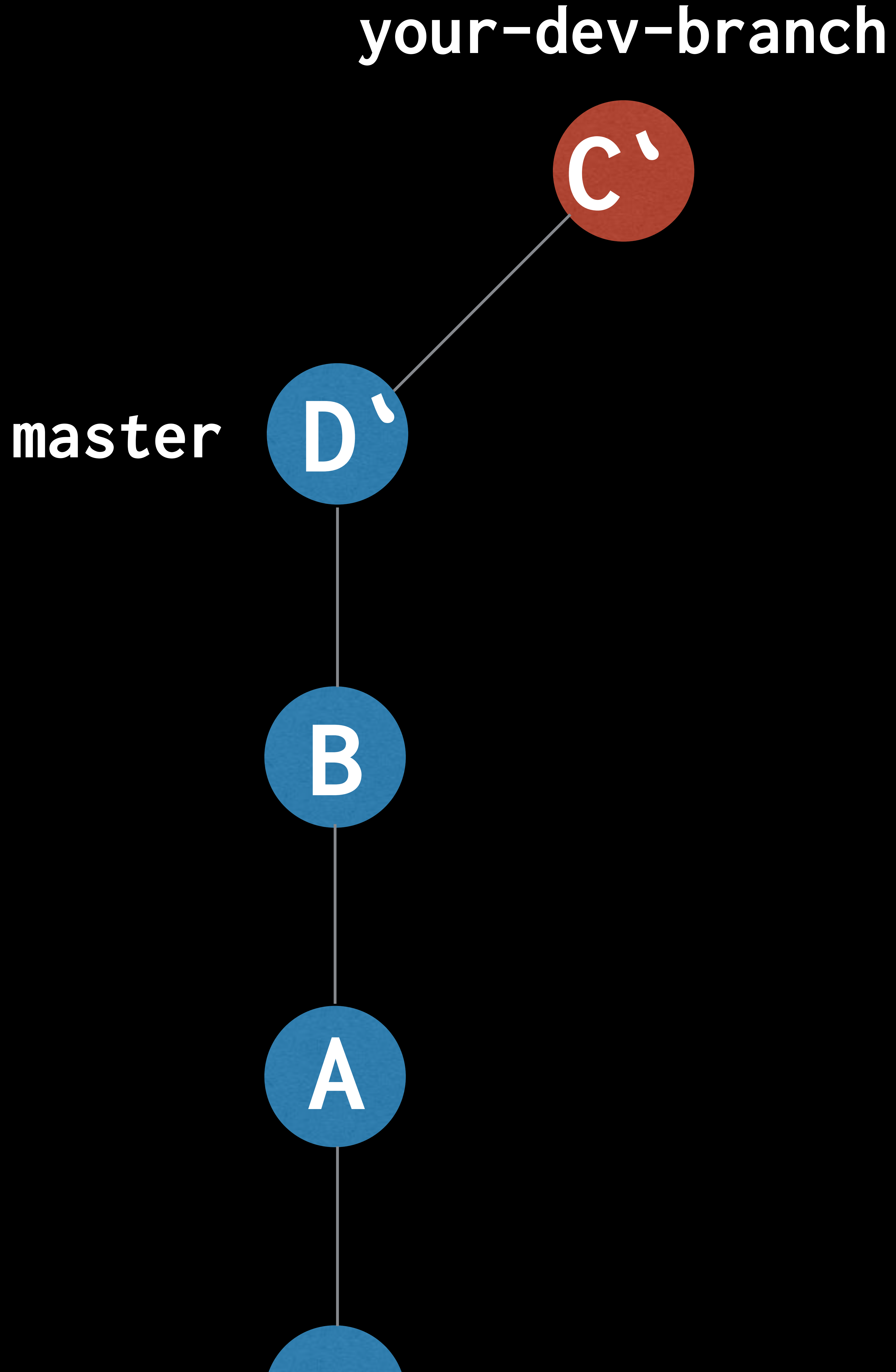
```
$ git checkout your-dev-branch
```



```
$ git rebase master
```



```
$ git rebase master
```



Keep focussed by making each development branch single purpose

Practice 3:

Make atomic commits

Image by lupusphotos (CC BY)

Decide the one change you are going to make and commit it

How big a change should I make?

Minimum valuable commit

Be suspicious of 'and' in your commit message

Notice when you start doing something else and stop

```
git add --patch

# Enables you to review all your
# changes and decide which to
# add to your commit
```

```
$ git add --patch
diff --git a/generate_load.rb b/generate_load.rb
index 581b4a6..d59e157 100755
--- a/generate_load.rb
+++ b/generate_load.rb
@@ -6,7 +6,7 @@ options = {}
 optparse = OptionParser.new do|opts|
   opts.banner = "Usage: script.rb [options] url"
   options[:number_of_requests] = 10
-  opts.on( '-n', '--number_of_requests REQUESTS', 'Number of requests
default to true' ) do |requests|
+  opts.on( '-n', '--number_of_requests REQUESTS', "Number of requests
default to #{options[:number_of_requests]}" ) do |requests|
     options[:number_of_requests] = requests.to_i
   end
   opts.on( '-h', '--help', 'Display this screen' ) do
Stage this hunk [y,n,q,a,d,e,?]?
```

Make each step simple by making atomic commits

Practice 4:

Write good commit messages

```
dc8f609 It worked for me
a813998 Final commit, ready for tagging
834a288 Don't push this commit
7cd9b24 WTF
901b51c done. going to bed now.
57d298f WIP
704de26 This will definitely work
b4512c6 This might work
d90b710 Trying to fix it again
c57b012 Debug stuff
```

Note: These messages are mostly taken from http://whatthecommit.com

What does good look like?

Short one line title

Longer description of what the change achieves (if the title isn’t enough).

An explanation of why the change is being made.

Perhaps a discussion of context and/or alternatives that were considered.

Short one line title

Longer description of what the change achieves (if the title isn’t enough).

An explanation of why the change is being made.

Perhaps a discussion of context and/or alternatives that were considered.

Short one line title

Longer description of what the change achieves (if the title isn’t enough).

An explanation of why the change is being made.

Perhaps a discussion of context and/or alternatives that were considered.

Short one line title

Longer description of what the change achieves (if the title isn’t enough).

An explanation of why the change is being made.

Perhaps a discussion of context and/or alternatives that were considered.

```
Short one line title

Longer description of what the change
achieves (if the title isn’t enough).

An explanation of why the change is being
made.

Perhaps a discussion of context and/or
alternatives that were considered.
```

Correct the colour of FAQ link in course notice footer

PT: https://www.pivotaltracker.com/story/show/84753832

In some email clients the colour of the FAQ link in the course notice footer was being displayed as blue instead of white. The examples given in PT are all different versions of Outlook. Outlook won't implement CSS changes that include `!important` inline[1]. Therefore, since we were using it to define the colour of that link, Outlook wasn't applying that style and thus simply set its default style (blue, like in most browsers). Removing that `!important` should fix the problem.

[1] https://www.campaignmonitor.com/blog/post/3143/outlook-2007-and-the-inline-important-declaration/

Clear space in your head by writing good commit messages

Practice 5:

Rewrite your history to tell a simple story (early and often)

Image by Nic McPhee (CC BY-SA)

You make mistakes and change your mind

```
git rebase --interactive

# Enables you to move, reorder,
# edit, merge and split your
# commits
```

```
343eed2 Fix typo in foo
ba66794 Add bar
90328f9 Add foo
```

```
git rebase --interactive master
```

```
pick 90328f9 Add foo
pick ba66794 Add bar
pick 343eed2 Fix typo in foo

# Rebase c405e59..343eed2 onto c405e59 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
```

```
pick 90328f9 Add foo
pick 343eed2 Fix typo in foo
pick ba66794 Add bar

# Rebase c405e59..343eed2 onto c405e59 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
```

```
pick 90328f9 Add foo
fixup 343eed2 Fix typo in foo
pick ba66794 Add bar

# Rebase c405e59..343eed2 onto c405e59 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
```

```
4a14d7b Add bar
c296093 Add foo
```

Make your progress clear by rewriting your history to tell a simple story

To recap the 5 practices

1. Plan your commits
2. Use single purpose branches
3. Make atomic commits
4. Write good commit messages
5. Rewrite your history to tell a story (early and often)

```
git cherry-pick
git add --patch
git rebase --interactive
```

Following these 5 practices will free up your brain and help you break work into small steps

As an added bonus...

...it can also provide valuable documentation for your code

“Every line of code is always documented”

- Mislav Marohnić
mislav.net/2014/02/hidden-documentation/

Thank you

Joel Chippindale - CTO at Unmade - @joelchippindale

Thanks to my teams at Unmade, FutureLearn and Econsultancy who have all helped develop and refine these habits.

Come and work with us at Unmade
www.unmade.com/careers/

Links for more info

- A Branch in Time (a story about revision histories)
- Every line of code is always documented
- Telling stories with your Git history