

Rebasing





Rebasing

When I first learned Git, I was told to avoid rebasing at all costs.

"It can really #%@* things up"

"It's not for beginners!"





Rebasing

So I avoided the **git rebase** command for YEARS!





Rebasing

It's actually very useful, as long as you know when NOT to use it!





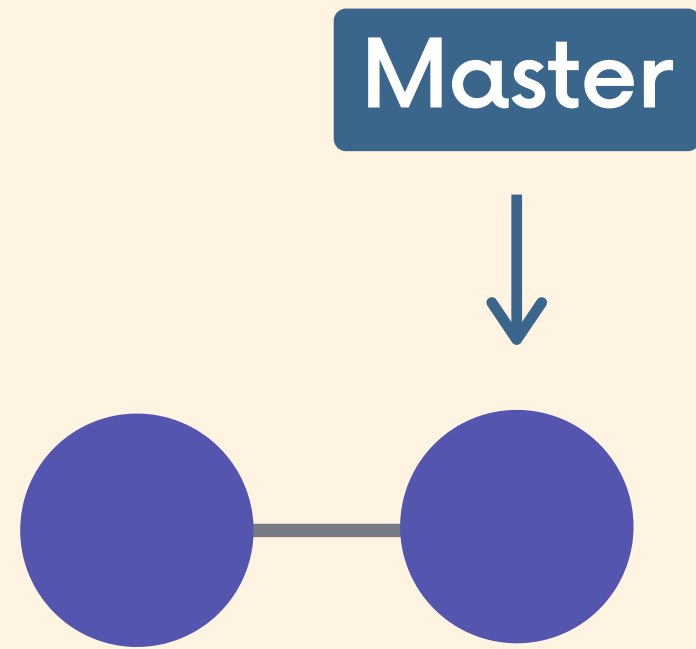
Rebasing

There are two main ways to use the git rebase command:

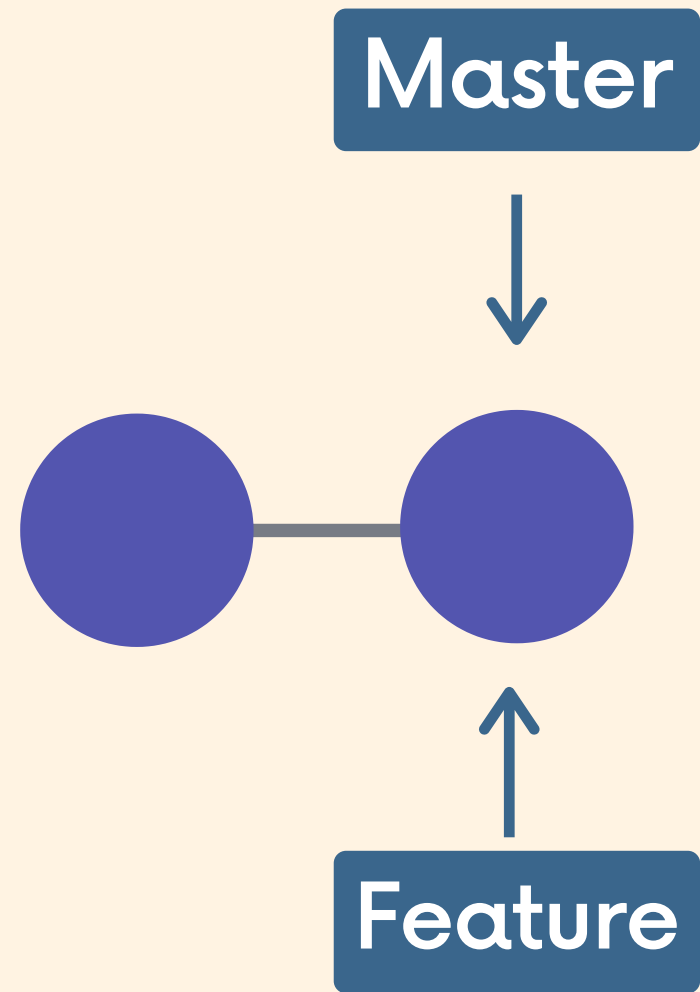
- as an alternative to merging
- as a cleanup tool



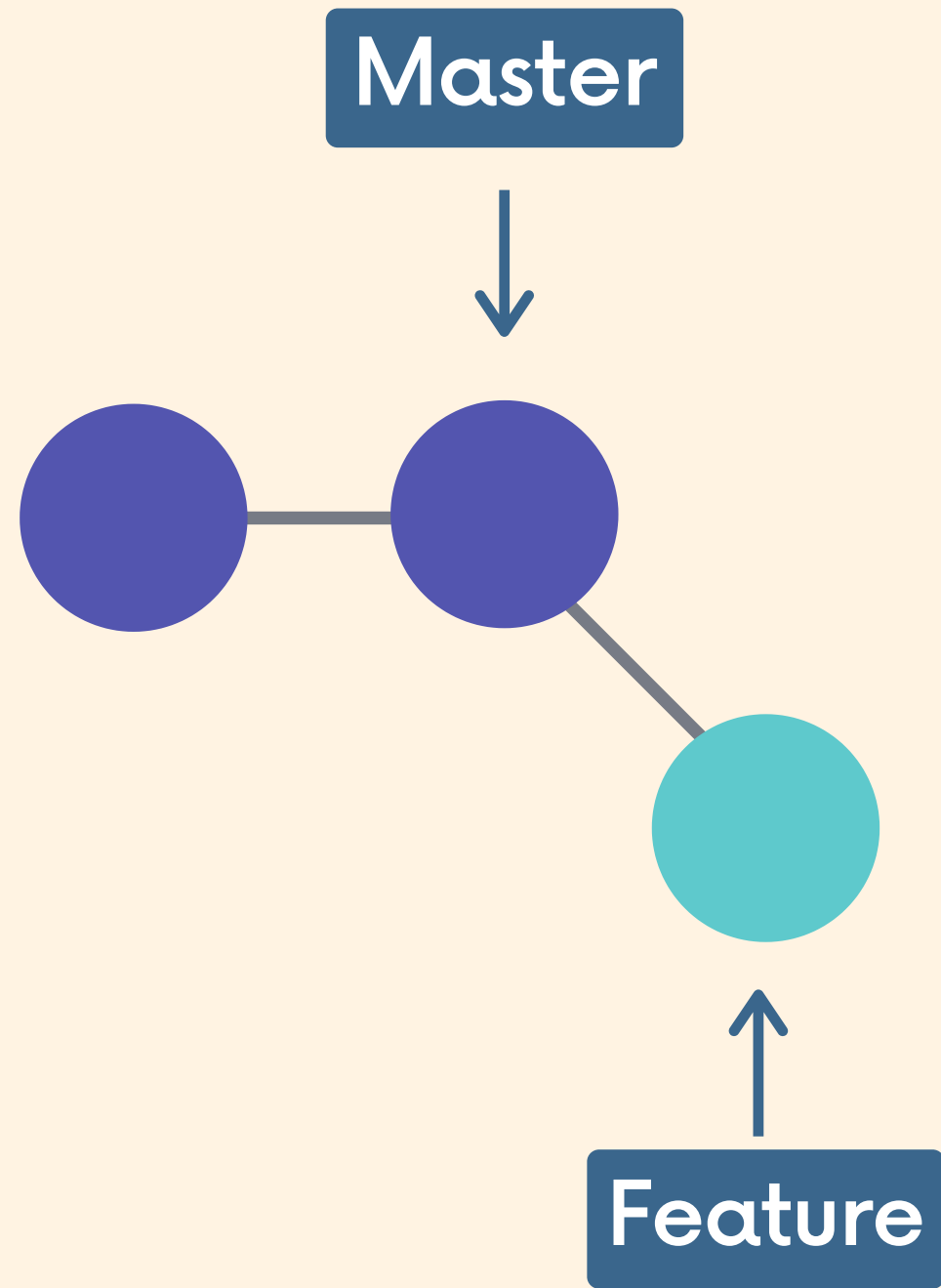
I'm working on a collaborative project



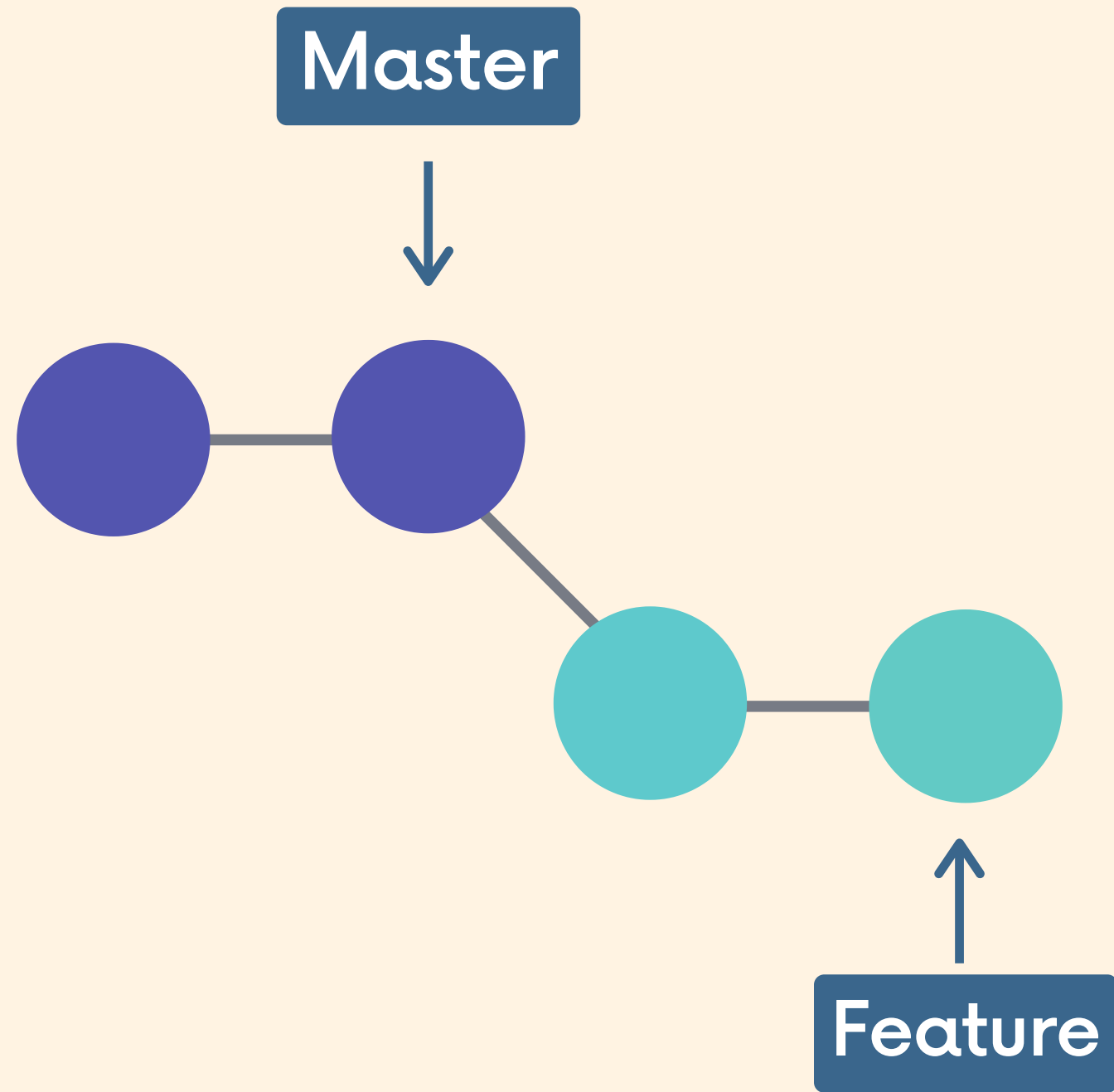
I make a feature branch!



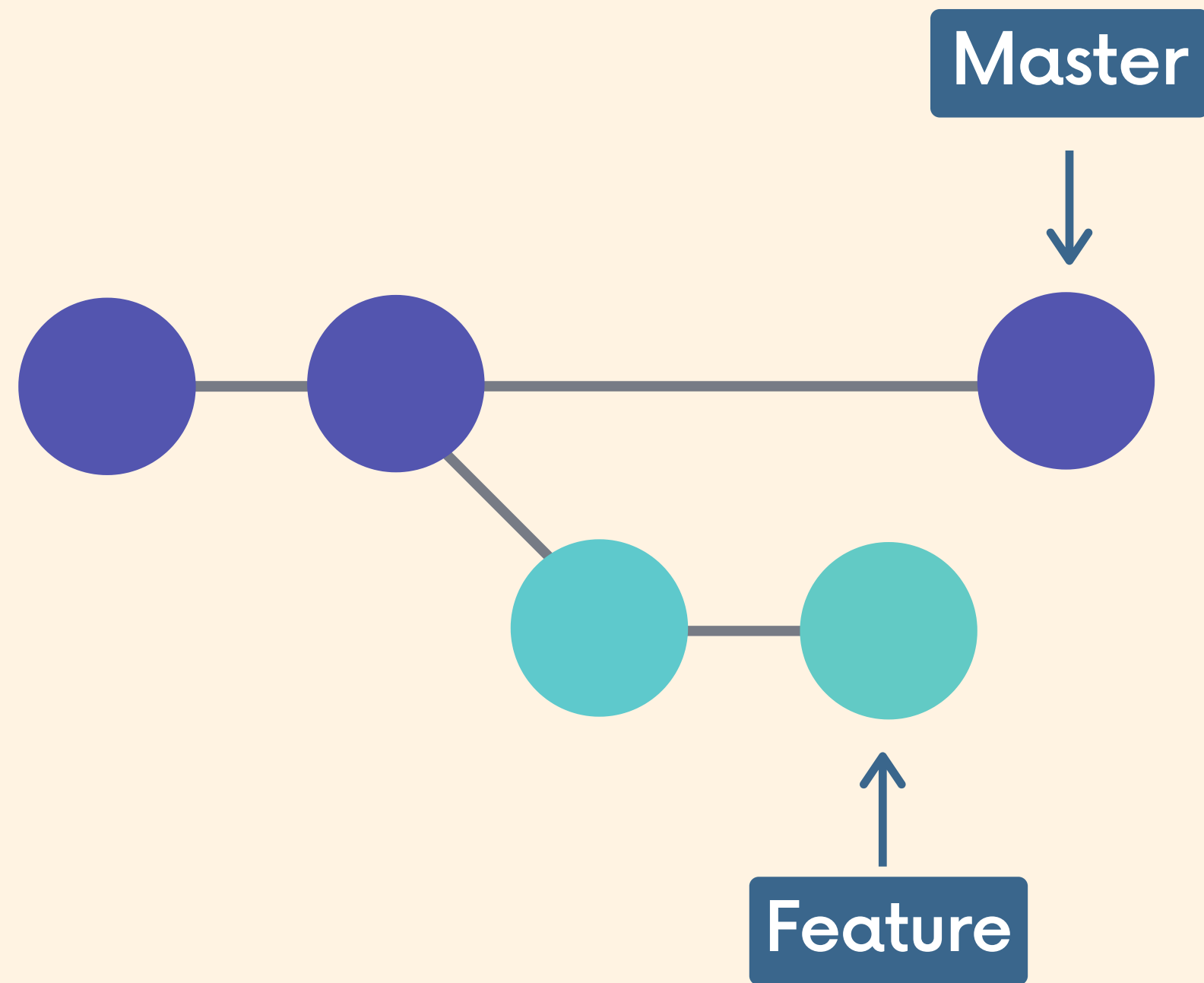
I do some work on the branch



I do some more work

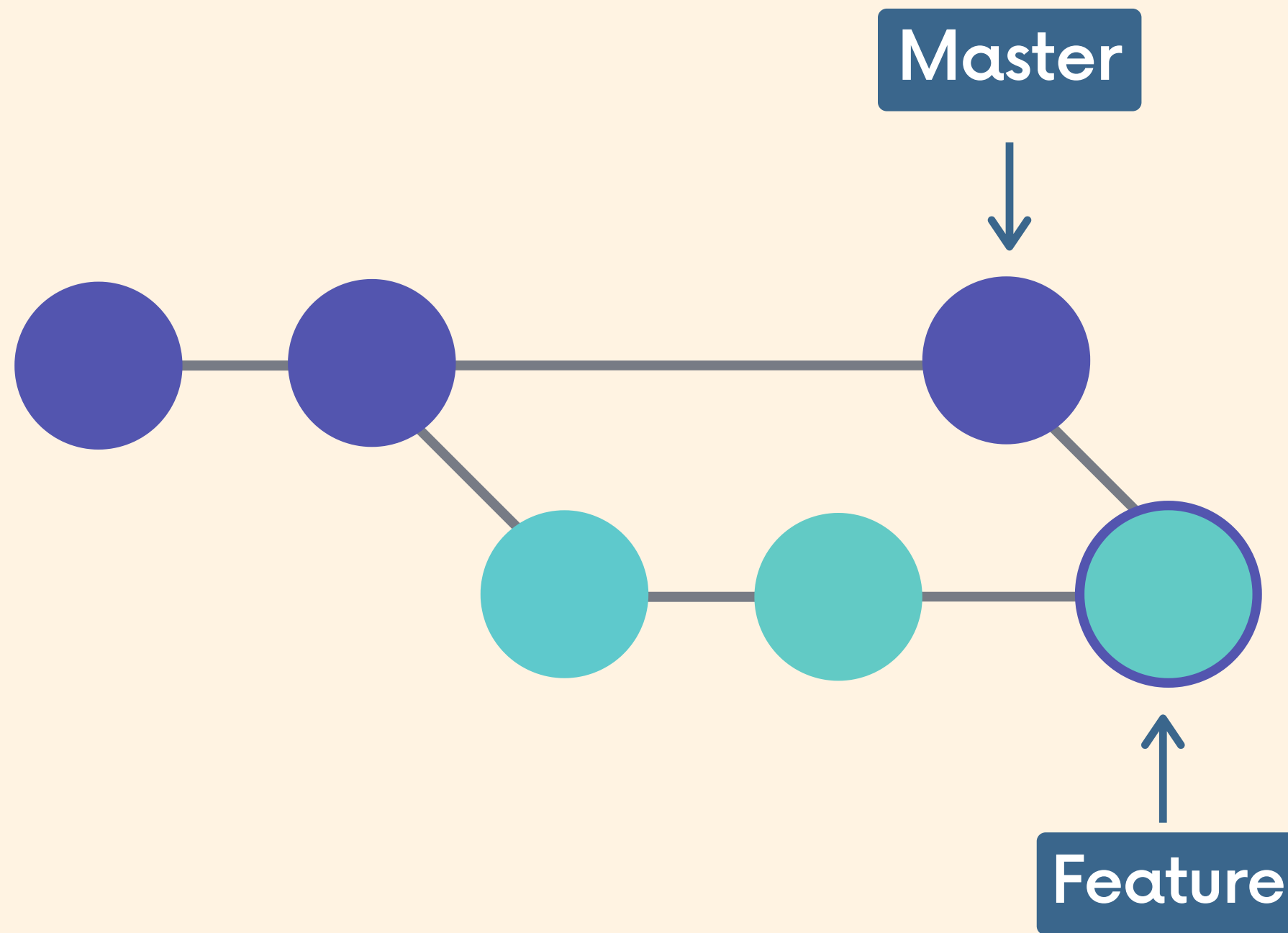


Master has new work on it!



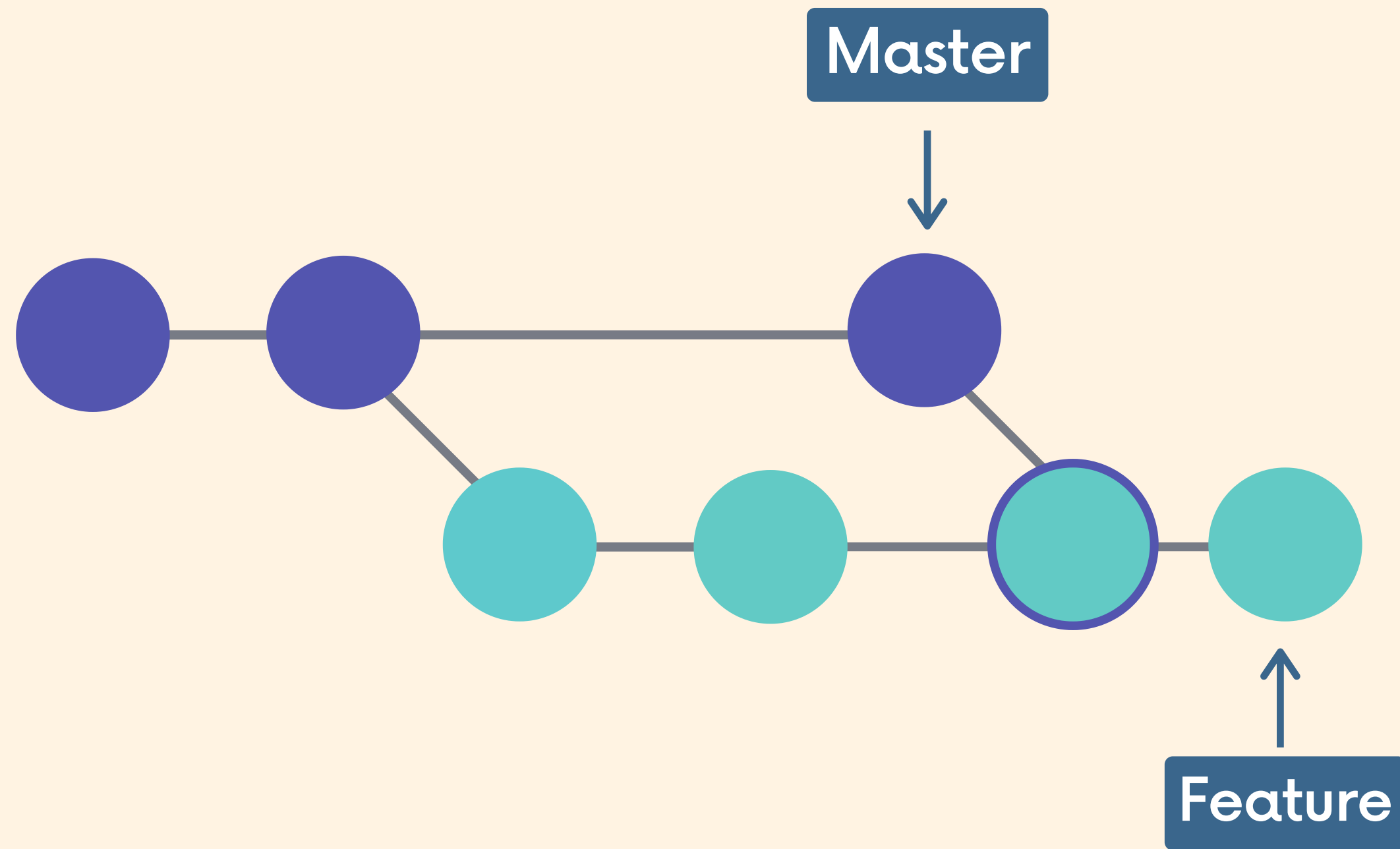
My feature branch doesn't have this work!

I merge master into feature

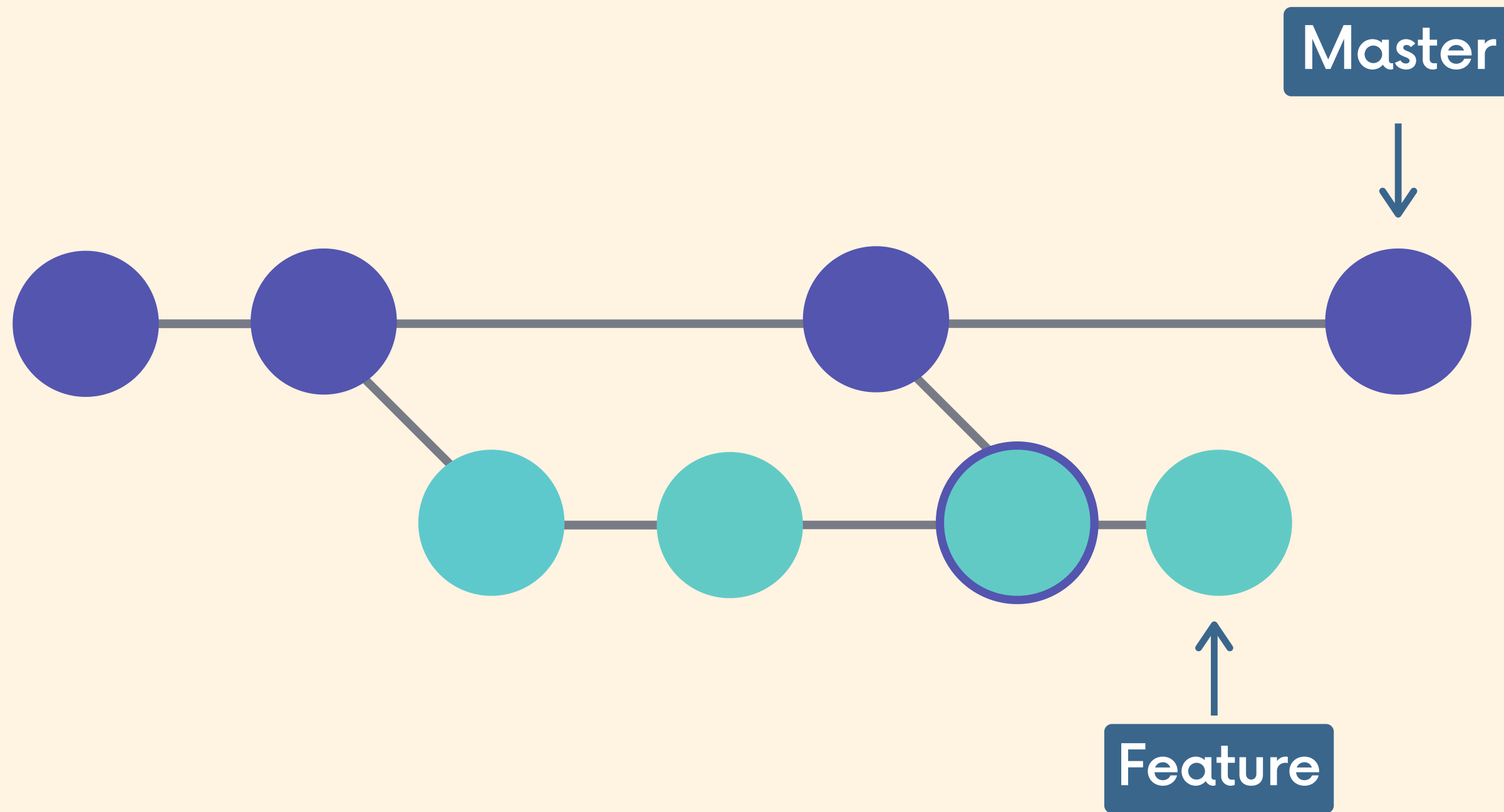


This results in a new merge commit

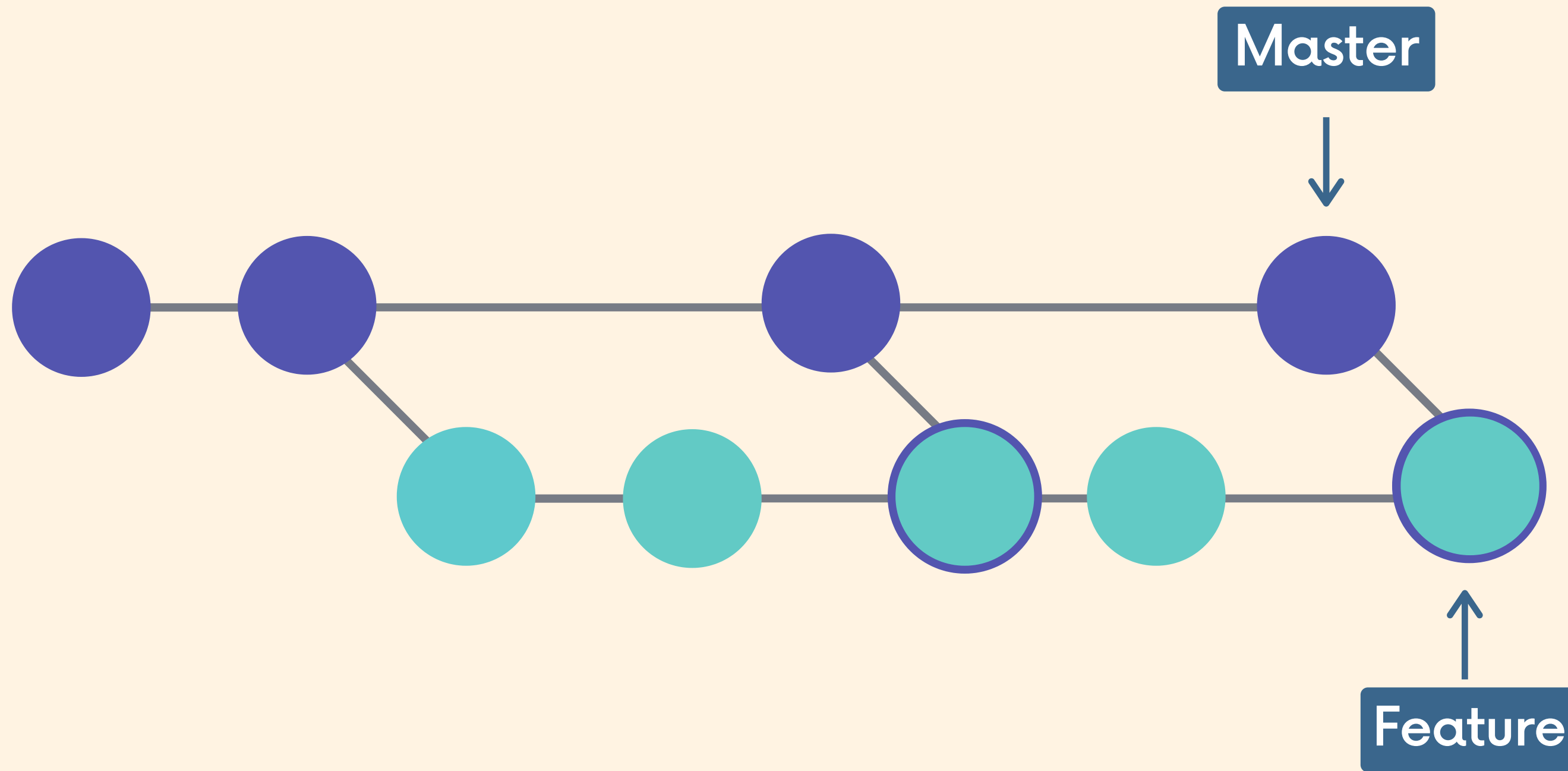
I keep working on my feature branch



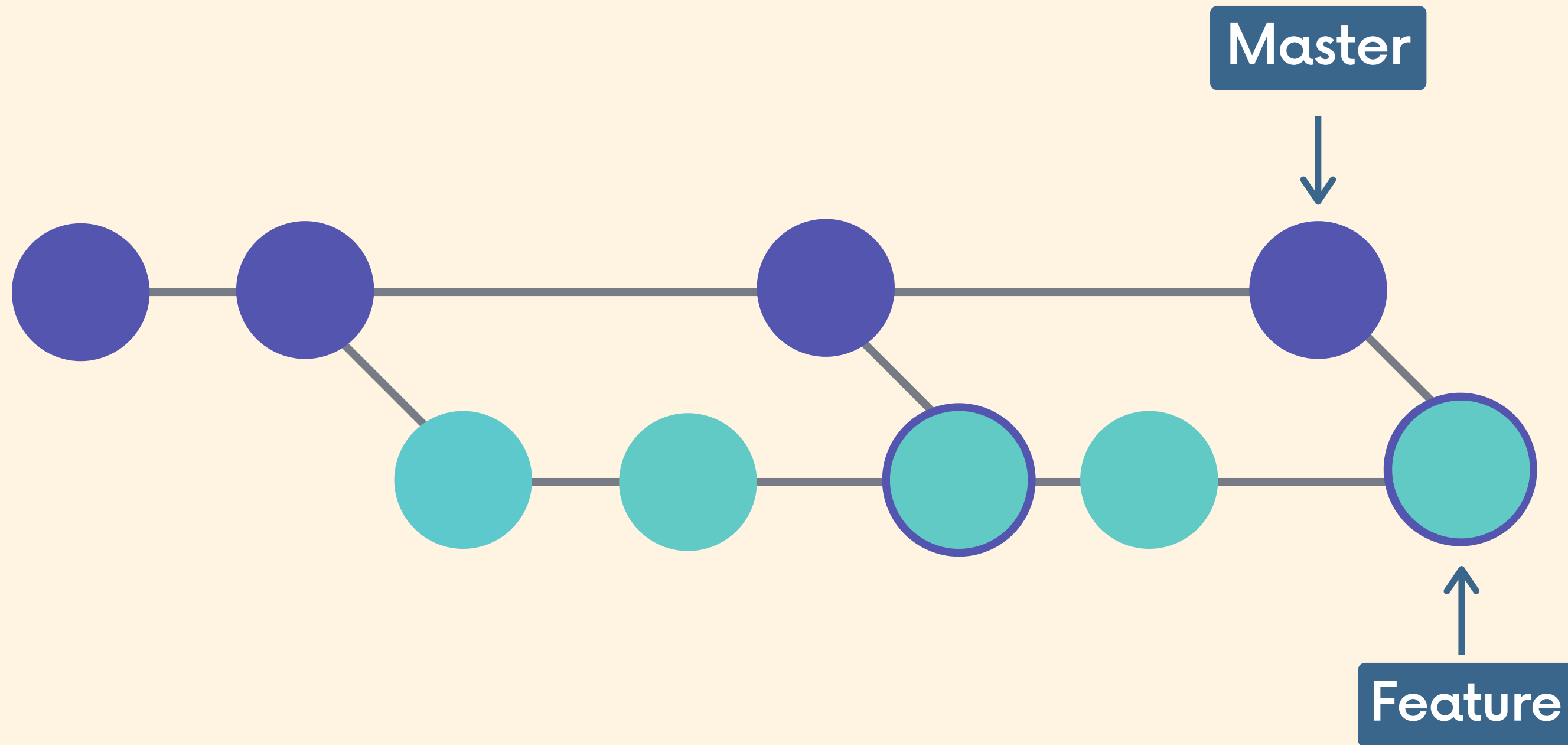
A coworker adds new work to master



I merge master in to my feature branch



This results in yet another merge commit!



The feature branch has a bunch of merge commits. If the master branch is very active, my feature branch's history is muddled

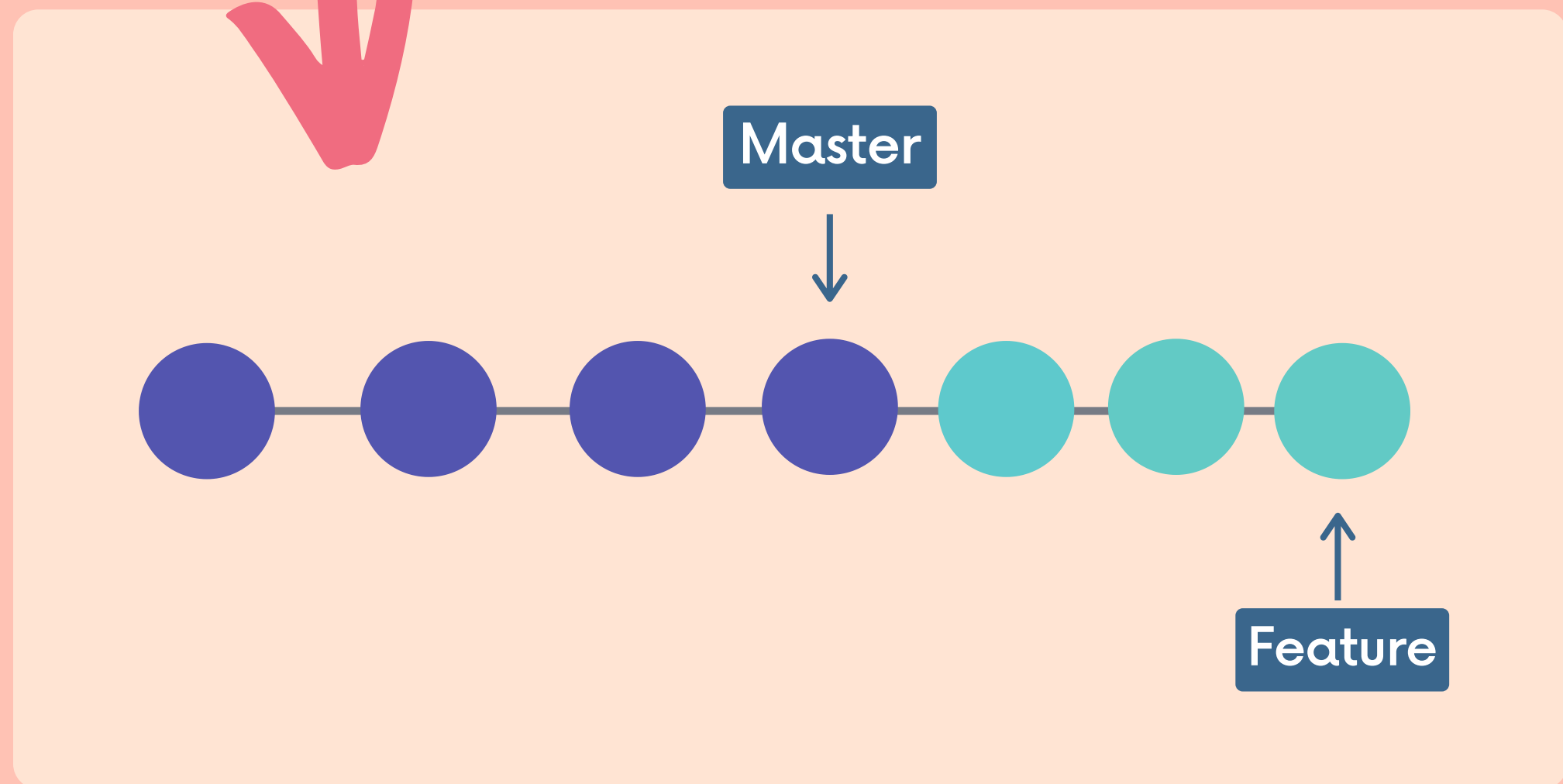
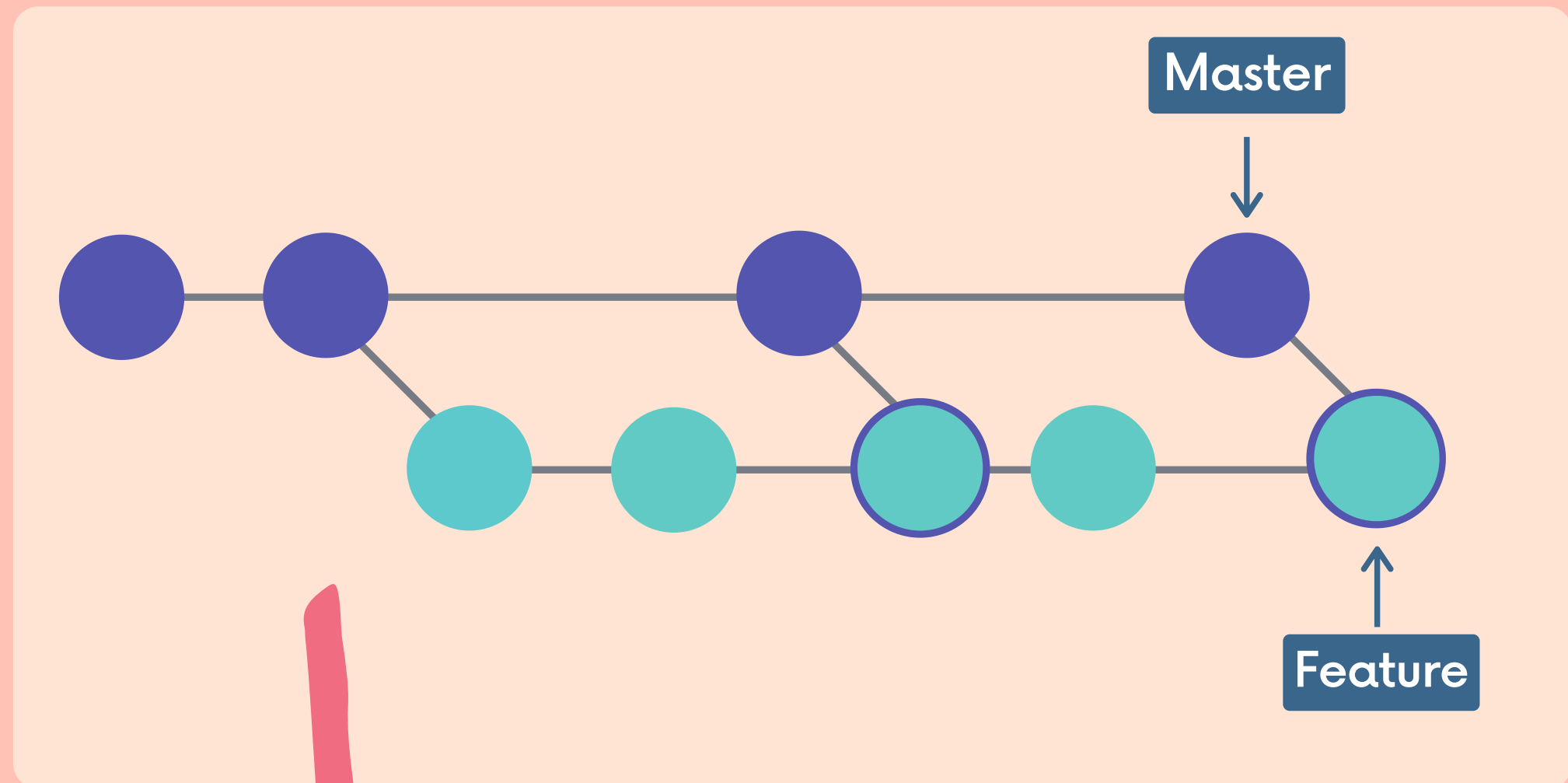
Rebasing!

We can instead rebase the feature branch onto the master branch. This moves the entire feature branch so that it **BEGINS** at the tip of the master branch. All of the work is still there, but **we have re-written history**.

Instead of using a merge commit, rebasing rewrites history by **creating new commits** for each of the original feature branch commits.

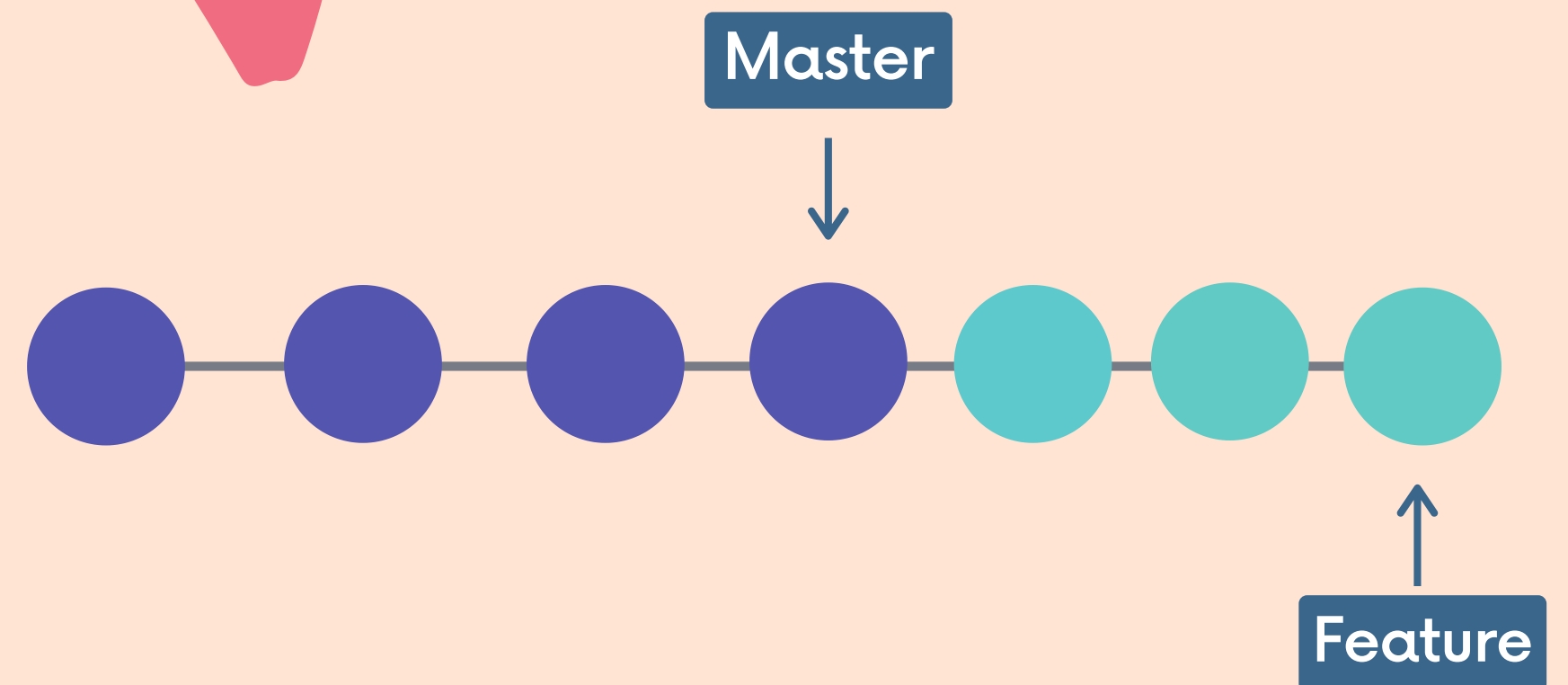
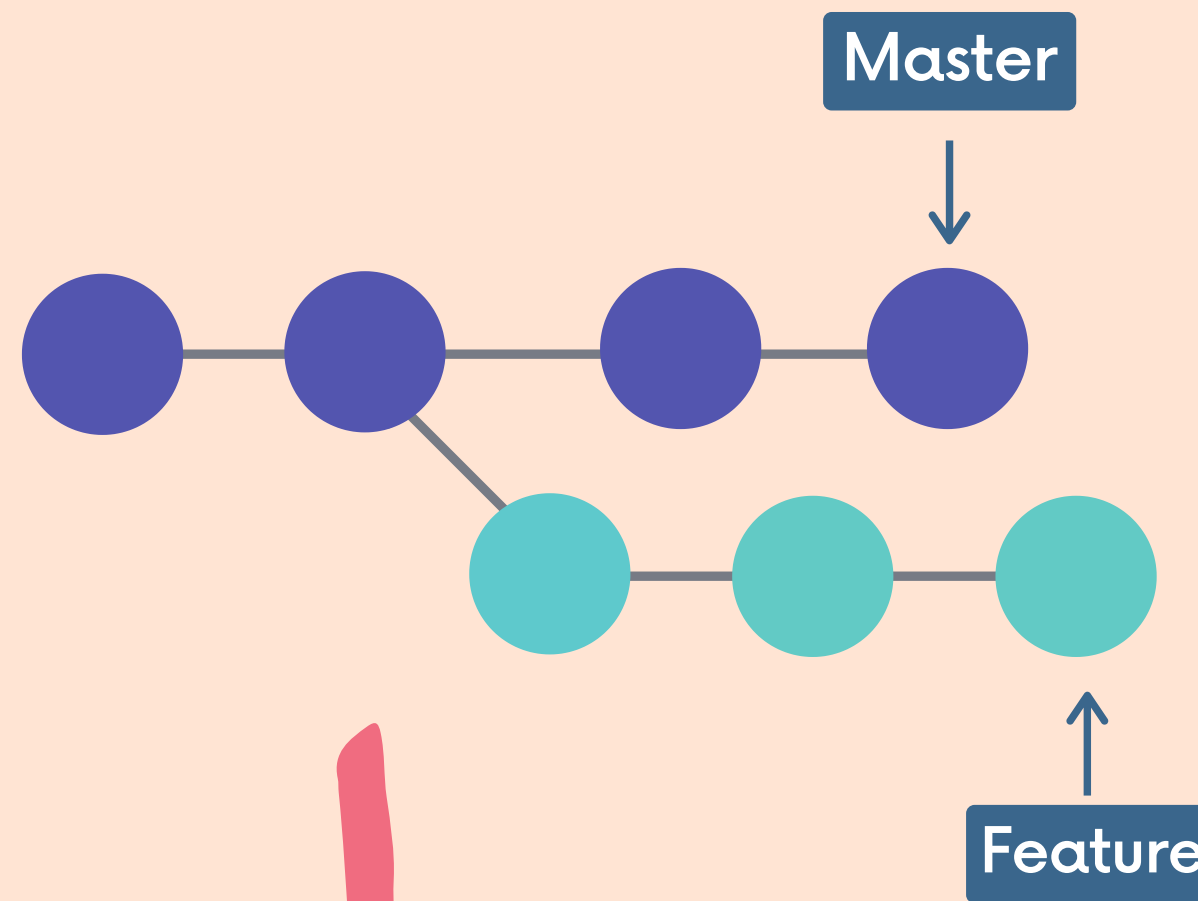


```
> git switch feature  
> git rebase master
```



Rebasing!

We can also wait until we are done with a feature and then rebase the feature branch onto the master branch.



```
> git switch feature  
> git rebase master
```



Why Rebase?

We get a much cleaner project history. No unnecessary merge commits! We end up with a linear project history.





WARNING!

Never rebase commits that have been shared with others. If you have already pushed commits up to Github...DO NOT rebase them unless you are positive no one on the team is using those commits.





SERIOUSLY!

You do not want to rewrite any git history that other people already have. It's a pain to reconcile the alternate histories!





Rebasing

There are two main ways to use the **git rebase** command:

- as an alternative to merging
- as a cleanup tool





Rewriting History

Sometimes we want to rewrite, delete, rename, or even reorder commits (before sharing them)

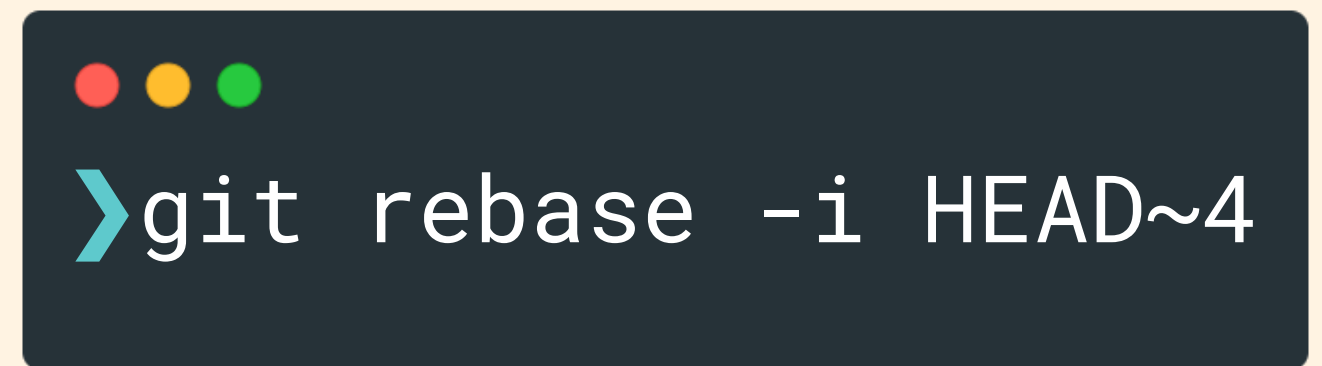
We can do this using **git rebase!**



Interactive Rebase

Running `git rebase` with the `-i` option will enter the interactive mode, which allows us to edit commits, add files, drop commits, etc. Note that we need to specify how far back we want to rewrite commits.

Also, notice that we are not rebasing onto another branch. Instead, we are rebasing a series of commits onto the HEAD they currently are based on.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. The command `git rebase -i HEAD~4` is entered in white text, with a cyan prompt character `>` at the beginning.

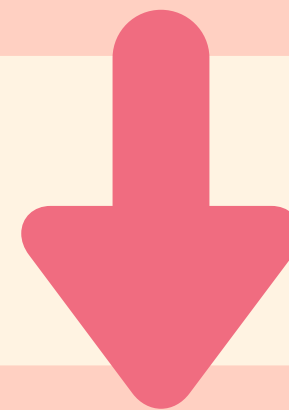
```
>git rebase -i HEAD~4
```

What Now?

In our text editor, we'll see a list of commits alongside a list of commands that we can choose from. Here are a couple of the more commonly used commands:

- `pick` - use the commit
- `reword` - use the commit, but edit the commit message
- `edit` - use commit, but stop for amending
- `fixup` - use commit contents but meld it into previous commit and discard the commit message
- `drop` - remove commit

```
pick f7f3f6d Change my name a bit  
pick 310154e Update README  
pick a5f4a0d Add cat-file
```



```
drop f7f3f6d Change my name a bit  
pick 310154e Update README  
reword a5f4a0d Add cat-file
```