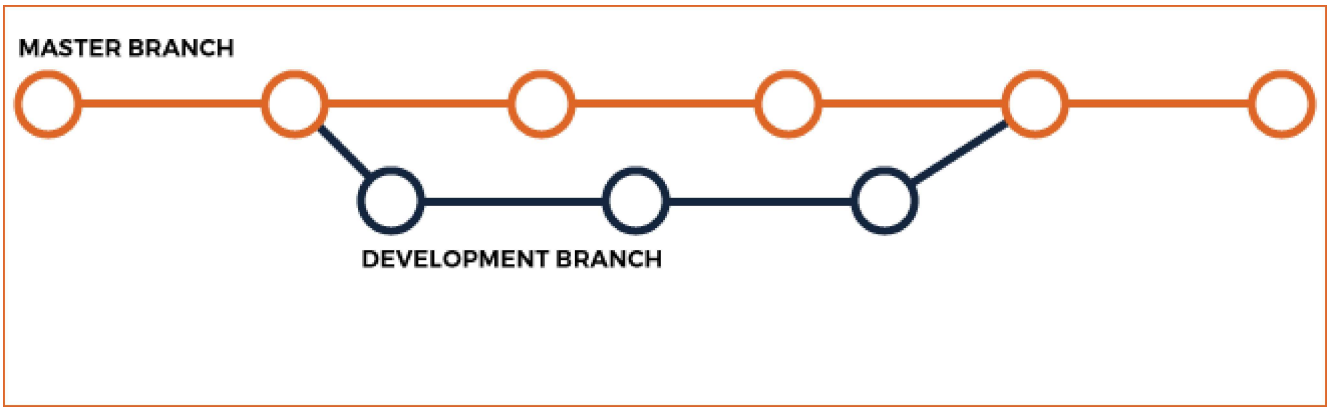
Git branches are just references to commits

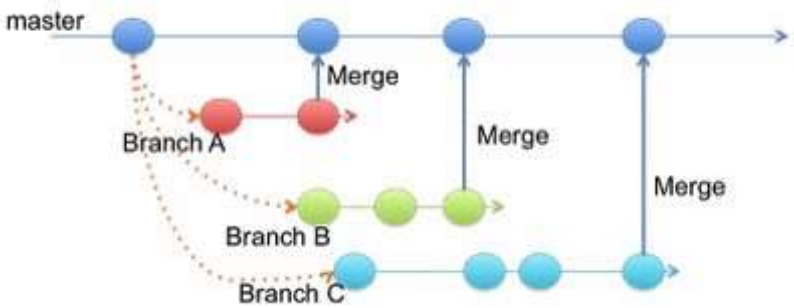
Sergeon
Dec 10, 2017 · 3 min read

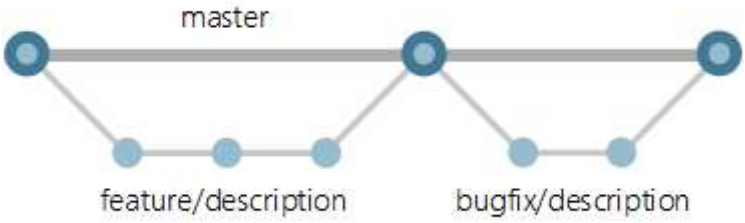
Git branches are just pointers to commits. They are not actually 'branches' at all, since don't contain nor group several commits: every branch just refers to a **single commit**. Since a Git repository is a 'tree' of nodes -the commits themselves- it sure has 'branches': but a 'git branch' has not a 1:1 correspondence to a 'tree branch' in that sense.

I actually needed a LOT OF TIME to realize this fact. I've met a lot of developers who also don't know what git branches are exactly, so I hope this story will help someone out there.

One of the biggest problems when dealing with branches as a git newbye is that most articles about git and git branches usually give a deceitful graphical representation of them, **pretending that branches are actually groups of commits** instead of what they are: mere names for commits. Googling for images about 'git branches' gives a lot of images like those ones:







The problem with such visualizations is that they seem to pretend that, at a git level, there are such things as 'branches' that somehow represent a group of commits. (of course, I'm not saying that the articles containing such representations are deceitful. They are fine if you know git well, but can be misleading if you don't know exactly what branches are at low level).

However, at git level you're dealing basically *only with commits*. Your project story is stored as several commits, which represent *versions*. Branches are just ways to name commits, so you can do:

```
git checkout develop
```

instead of:

```
git checkout 2d3dabff7754f56862aa7e791b0a48e9725c8af4
```

or:

```
git merge develop master
```

instead of:

```
git merge 2d3dabff7754f56862aa7e791b0a48e9725c8af4
39ba55984ece3cf50d644c78cd47e28c6830552b
```

Of course, **at project level**, when we create a new topic branch and start a new feature, we're actually creating a group of commits that relate to that new feature. But that's not *exactly* the way git branches work from git perspective: in git, a branch points only to a single commit.

Ok, so what's the deal?

Not much, actually. However, I've seen several problems related to this lack of branches understanding, mainly those:

- Developers get afraid to delete branches.

Once you've merged your 'change-header-brand' into master, it means that your master branch now points to a commit that contains -i.e., is descendant of- the last commit performing that feature. That commit -that version of your project, in which the new brand header is finished and ready for production- has now a reference other than the topic branch: so it can be deleted safely, without any risk of losing the work.

Developers who fail to realize this usually ends with a lot of local branches, which makes working with git really annoying.

- Lack of proper understanding of git merges

To understand merges correctly can be taunting, and without the proper guidance is pretty easy to mess up things a lot when resolving conflicts in git: either you push conflicting files, or don't add files to the merge commit that should be there, and you're in the way to break a remote build.

If you don't have a clear understanding of what commits and branches really are, chances that you will make such mistakes get bigger and bigger. To know what a git branch is and what is not is not the only thing you must to know in order to master git merges, but sure is one of them.

Git