Stop cherry-picking, start merging, Part 7: Preventing a change from leaving a branch

devblogs.microsoft.com/oldnewthing/20180320-00

March 20, 2018

Raymond Chen

Still continuing our exploration of using merges as a replacement for cherry-picking, here's another scenario you can now solve:

How do I make a change in a branch that will not propagate when it merges?

Suppose you have a develop branch where active development occurs, and you also have a release branch that represents the current release candidate. Bug fixes in the release branch are regularly merged back into the develop branch. But what if there's a fix in the release branch that you don't want to merge back into the develop branch?

For example, maybe the component in question has already fixed the problem in the develop branch, but the fix came as part of a larger refactoring of the component. You want to apply a targeted fix to the release branch, but that targeted fix should not merge back to the develop branch.

It turns out that you already know how to do this.

This is basically the same thing as Replacing the temporary fix with the permanent fix, just with different names for the branches.

Here's the original diagram from last time:

	apple		berry	
	M1		M2	master
apple		berry		
A		P		patch
	F1		F2	feature

	apple		apple

In the diagram from last time, we have a common commit A onto which which a master branch and feature branch have been committing. We create a patch branch from the common commit A and apply a fix to it (changing `apple` to `berry`), producing commit P. We merge that branch into the master branch (producing commit M2 with the fix), and we also merge it into the feature branch with `-s ours` to indicate that we don't want any code changes from this merge; we are creating it only for bookkeeping purposes. The line remains unchanged in the feature branch.

From what we saw last time, this means that when the master branch merges into the feature branch, the line will remain unchanged.

Okay, now rename the branches. Rename the master branch to the release branch, and rename the feature branch to the develop branch.

	apple		berry	
	M1		M2	release
apple		berry		
A		P		patch

	F1		F2	develop
	apple		apple	

Again, we start with a common commit A onto which a release branch and a develop branch have been committing. We create a patch branch that contains the version of the fix that goes into the release branch and merge that into the release branch. We then take that same patch branch and merge it with `-s ours` into the develop branch, resulting in no code change.

When the release branch merges into the develop branch, the changes will be discarded because the patch commit P will be a merge base, and relative to that merge base, the release branch didn't do anything, and the develop branch reverted the change. Therefore, the result of the merge is a revert of P.

I've heard people say that "In git, there is no way to prevent a change from propagating during a merge." Well, maybe not, but I just did it.

What happens, is that the commit propagates during the merge, but we have prearranged the merge base so that the result of the propagation is “no change”. Technically the commit propagated: After the merge of the release branch into the develop branch, you can ask for branches that contain commit M2, and it will say that the commit is present in the develop branch.

The git-theorists are correct in that the commit did propagate during the merge. The git-pragmatists are correct in that the effect of the propagation is no change to the develop branch. Both sides are correct. Everybody wins.

Raymond Chen

Follow