Stop cherry-picking, start merging, Part 9: Chasing the commit

devblogs.microsoft.com/oldnewthing/20180322-00

March 22, 2018

Raymond Chen

Consider the following situation:

	apple					
	M1	master				
apple						
A						
	apple		apple	berry		
	V1		V2	V3	victim	
		F1	F2		F3	feature
		apple	berry		cherry	

From a starting common commit A (where the line is “apple”), the master branch makes an unrelated commit M1. Meanwhile we branch off from commit A with a new branch called “victim”, on which unrelated commits V1 and V2 are made. From commit V1, another branch called “feature” is created, where an unrelated commit F1 is made. After commit F1, there is another commit F2 which changes the line from “apple” to “berry”. At this point, the feature branch merges back to the victim branch, resulting in a merge commit V3, where the line is now “berry”. After the merge, another commit F3 is made to the feature branch, which changes the line from “berry” to “cherry”.

At this point, you decide that you want commit F2 (the one that changed “apple” to “berry”) to go to master. Maybe there was some problem that F2 fixes which you thought was local to your feature branch, but it turns out that it affected the master branch too, and now the people who run the master branch want your temporary fix.

So we follow our cookbook. The patch branch uses commit A as its starting point. It cherry-picks a copy of F2 and merges it into the master branch.

	apple			berry	
	M1			M2	master
apple			berry		
A			P		patch
	apple	apple	berry		
	V1	V2	V3	?	victim

	F1	F2	F3	feature
	apple	berry	cherry	

But what about the other half of the merge pair? Does the patch branch merge into the feature branch?

No, merging into the feature branch won't help. Commit F2 has already been merged into the victim branch, and is on its way to merging into the master branch. Any changes to the feature branch at this point will have no effect on the payload that is already on the train.

You have to merge the patch branch into the branches that have carried the original change closest to its destination, and the cherry-pick closest to the source. In our example, the source is the feature branch and the destination is the master branch. The commit has merged as far as the victim branch, so that's where the patch needs to go. Because the point of the patch branch is to make sure the right thing happens when the original commit (F2) and its cherry-picked doppelgänger (M2) meet and need to merge together.

In other words, you need to *catch the train*.

	apple			berry		
	M1			M2	M3	master
apple			berry		↙	
A			P		↙	patch
	apple	apple	berry	↙		

V1		V2	V3	V4	victim

	F1	F2	F3	feature
	apple	berry	cherry	

The correct merge destination for the patch branch is the victim branch, resulting in commit V4. That way, when the victim branch merges with the master branch as commit M3, commit P becomes an eligible merge base.

Raymond Chen

Follow