

How to keep a git branch in sync with master

Asked 6 years, 5 months ago Active 1 year, 6 months ago Viewed 250k times

- ▲ 243 ▼ ★ 115
- At the moment git is doing my head in, I cannot come up with the best solution for the following.
- There are two branches, one called **master** and one called **mobiledevicesupport**. I want to keep mobiledevicesupport as a continuous branch that will be merged/synced with the master branch whenever mobiledevicesupport is stable. This would merge changes from mobiledevicesupport into master but also bring all the changes from master into mobiledevicesupport so that branch can continue to be worked on and the features improved or amended. This needs to work with a central repository and multiple developers.
- Please an example of similar workflows other people use or just tell me if this idea is stupid and I should consider other options. At the moment the workflow seems sound, but I just don't know how I can make git work this way.

Thanks, all help much appreciated.

Update 1: If I was to merge master into mobiledevicesupport and mobiledevice support into master, do I get replicated commits across both branches. Or is git smart enough to work out that I have pulled the latest changes from branch A into branch B and add merge commit C to branch B. And I have pulled the latest changes from branch B into branch A and add merge commit D to branch A?

I was going to post an image but I don't have enough reputation for it, so I guess the following illustration will have to do. Two branches continuously running with merges going both directions often. The key thing I am not sure about is how git will play out the commits and will it fill either branch with the commits from the other branch on merges or will it stay clean. I have used rebase before but it seems to end the branch and put all the commits into the master, or I did it wrong. Thanks for the help so far.

```

master
A--B--C-----H--I--J--M--N
      \   /   \
mobile  \ /   \
D--E--F--G-----K--L
  
```



edited May 31 '16 at 13:55



Martin Thoma

52.3k 68 360 592

asked May 2 '13 at 3:15



Mr. EZEKIEL

1,359 3 9 9

-
- 1 If you were, like me, looking for how to do it with GitHub *client*: help.github.com/articles/merging-branches – [cregox](#) Jan 5 '15 at 15:40
-
- 1 This question saved my life for centuries; Thanks for the great effort in taking time to set this wonderful question @Mr. EZEKIEL – [DJphy](#) May 7 '16 at 17:00
-
- In case you are working on a fork, you have to follow help.github.com/articles/syncing-a-fork – [koppor](#) Sep 11 '16 at 9:03
-

6 Answers



yes just do

372



```
git checkout master
git pull
git checkout mobiledevicesupport
git merge master
```



to keep mobiledevicesupport in sync with master

then when you're ready to put mobiledevicesupport into master, first merge in master like above, then ...

```
git checkout master
git merge mobiledevicesupport
git push origin master
```

and thats it.

the assumption here is that mobilexxx is a topic branch with work that isn't ready to go into your main branch yet. So only merge into master when mobiledevicesupport is in a good place

answered May 2 '13 at 3:44





[concept47](#)

13.7k 11 39 68

This sounds reasonable to me, I guess I am not sure how dirty this would make the commit history, I will update my question with an example of what I think would happen. – [Mr. EZEKIEL](#) May 2 '13 at 4:55

- 1 You will have quite a few "merge commits", essentially git trying to resolve differences between your branches. if you're worried about that AND you're the only one using the branch then do a "git rebase master" instead of a "git merge master" AND DO NOT PUSH THE COMMITS TO THE REMOTE

BRANCH. If you do you're going to find yourself doing a lot of force pushes (git push --force) to origin/mobiledevicesupport because you're going to (probably) always sending be it a history of commits that don't match what the remote branch has. more detail here git-scm.com/book/en/Git-Branching-Rebasing – [concept47](#) May 2 '13 at 10:19 

I believe this is the correct answer, it sounds exactly like what I want. I added an illustration above to make it a little more clear but if what you are saying is true, then this should work out exactly how I want. Thank you. – [Mr. EZEKIEL](#) May 2 '13 at 22:29 

- 2 Read this to understand why this is not especially good advice: kentnguyen.com/development/visualized-git-practices-for-team/.... That was written by the Git maintainer, so it's probably fair to say he knows what he's talking about with regard to this particular topic. – [Dan Moulding](#) Mar 12 '14 at 23:44
- 2 This get the commit history messy, see my answer do it via rebase. – [Gob00st](#) Nov 29 '16 at 15:30

Whenever you want to get the changes from master into your work branch, do a `git rebase <remote>/master` . If there are any conflicts. resolve them.

43


When your work branch is ready, rebase again and then do `git push <remote> HEAD:master` . This will update the master branch on remote (central repo).

answered May 2 '13 at 5:16



[euphoria83](#)

7,471 16 53 67

- 3 What is the pros/cons of doing it this way instead of as in the accepted answer? – [Hampus Ahlgren](#) Jan 14 '15 at 16:46 
- 26 Sane until you spend 5 hours in rebase hell – [IcedDante](#) Jan 21 '15 at 16:49
- 17 This is true only if your branch is on a private repository. Never rebase something that has been pushed upstream. This is why: git-scm.com/book/en/v2/... – [Kleag](#) Aug 28 '15 at 15:57
- 3 The problem with rebasing being a rewrite of the history is that the rebased commit's SHAs are changed and thus you cannot rely on the output of (e.g.) `git branch --contains <commit>` . – [jnns](#) Sep 1 '17 at 11:26

concept47's approach is the right way to do it, but I'd advise to merge with the `--no-ff` option in order to keep your commit history clear.

11

```
git checkout develop
git pull --rebase
git checkout NewFeatureBranch
git merge --no-ff master
```

answered Jun 1 '15 at 8:23



lwishlcanFLighT

198 1 7

Yeah I agree with your approach. To merge mobiledevicesupport into master you can use

8

```
git checkout master
git pull origin master //Get all latest commits of master branch
git merge mobiledevicesupport
```

Similarly you can also merge master in mobiledevicesupport.

Q. If cross merging is an issue or not.

A. Well it depends upon the commits made in mobile* branch and master branch from the last time they were synced. Take this example: After last sync, following commits happen to these branches

```
Master branch: A -> B -> C [where A,B,C are commits]
Mobile branch: D -> E
```

Now, suppose commit B made some changes to file a.txt and commit D also made some changes to a.txt. Let us have a look at the impact of each operation of merging now,

```
git checkout master //Switches to master branch
git pull // Get the commits you don't have. May be your fellow workers have made them.
git merge mobiledevicesupport // It will try to add D and E in master branch.
```

Now, there are two types of merging possible

1. Fast forward merge
2. True merge (Requires manual effort)

Git will first try to make FF merge and if it finds any conflicts are not resolvable by git. It fails the merge and asks you to merge. In this case, a new commit will occur which is responsible for resolving conflicts in a.txt.

So Bottom line is Cross merging is not an issue and ultimately you have to do it and that is what syncing means. Make sure you dirty your hands in merging branches before doing anything in production.

edited May 2 '13 at 4:32

answered May 2 '13 at 3:32



sachinjain024

15.3k 25 79 137

1 So cross merging like this isn't an issue? – [Mr. EZEKIEL](#) May 2 '13 at 3:38

Cross merging is what we say syncing and is not an issue unless the commits in both branches does not cause any conflicts. Please see my updated answer. – [sachinjain024](#) May 2 '13 at 4:33



2



You are thinking in the right direction. Merge master with mobiledevicesupport continuously and merge mobiledevicesupport with master when mobiledevicesupport is stable. Each developer will have his own branch and can merge to and from either on master or mobiledevicesupport depending on their role.

answered May 2 '13 at 3:21



faisal

1,059 9 17



0



The accepted answer via git merge will get the job done but leaves a messy commit history, correct way should be 'rebase' via the following steps(assuming you want to keep your feature branch in sync with develop before you do the final push before PR).

1 `git fetch` from your feature branch (make sure the feature branch you are working on is update to date)

2 `git rebase origin/develop`

3 if any conflict shall arise, resolve them one by one

4 use `git rebase --continue` once all conflicts are dealt with

5 `git push --force`

edited Apr 20 '18 at 10:49



George

498 10 21

answered Nov 29 '16 at 15:36



Gob00st

3,579 6 36 67

This is both hard to read and hard to understand. Please update your answer and use proper code markdown, to separate out your comments from the

commands. – [not2qubit](#) Mar 29 '18 at 8:53
