How to keep a branch synchronized/updated with master?

Asked 10 years, 5 months ago Modified 1 year, 5 months ago Viewed 543k times

392

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
```

git git-merge git-branch

Share Improve this question Follow

edited Apr 9, 2021 at 0:28
ivanleoncz
9,200 ● 7 ● 58 ● 49

asked May 2, 2013 at 3:15
Mr. EZEKIEL
4,084 ● 3 ● 14 ● 9

2 If you were, like me, looking for how to do it with GitHub *client*: help.github.com/articles/merging-branches – cregox Jan 5, 2015 at 15:40

3 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, 2016 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, 2016 at 9:03

Googlers: the answers below, naturally, end up with `rebase` vs `merge` dilemma; see here for comparison: perforce.com/blog/vcs/git-rebase-vs-merge-which-better – MEHOV Aug 13, 2021 at 10:11

8 Answers

Sorted by: Highest score (default)

639

Yes, just do:

```
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 that's 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.

Share Improve this answer Follow

edited Apr 24, 2022 at 2:43
Oleg Valter is with Ukraine
9,649 ●8 ●37 ●59

answered May 2, 2013 at 3:44
concept47
30.4k ●12 ●52 ●75

3 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, 2013 at 4:55

3 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, 2013 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, 2013 at 22:29

3 This get the commit history messy, see my answer do it via rebase. – RoundPi Nov 29, 2016 at 15:30

3 @oneworld to make sure you have the latest version of master. You can accomplish the same thing by just doing a `git pull origin master` from the mobiledevicesupport branch – concept47 Aug 1, 2017 at 0:13

51

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.

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).

Share Improve this answer Follow

answered May 2, 2013 at 5:16
euphoria83
14.8k ● 17 ● 63 ● 73

4 What is the pros/cons of doing it this way instead of as in the accepted answer? – Hampus Ahlgren Jan 14, 2015 at 16:46

49 Sane until you spend 5 hours in rebase hell – IcedDante Jan 21, 2015 at 16:49

26 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, 2015 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, 2017 at 11:26

27

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.

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

Share Improve this answer Follow

answered Jun 1, 2015 at 8:23
a1pancakes
503 ● 4 ● 7

3 More info and graphics on `--no-ff`: stackoverflow.com/questions/9069061/… – MEHOV Jun 2, 2021 at 12:05

17

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 should arise, resolve them one by one

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

5 `git push --force`

Share Improve this answer Follow

edited Mar 23, 2022 at 21:03
Ruby Racer
5,690 ●1 ●26 ●43

answered Nov 29, 2016 at 15:36
RoundPi
5,849 ●7 ●49 ●76

2 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, 2018 at 8:53

1 Recommend to use `git push --force-with-lease` instead of `git push --force` – se0kjun Aug 3, 2020 at 2:42

9

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

```
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.

Share Improve this answer Follow edited May 2, 2013 at 4:32 answered May 2, 2013 at 3:32 Sachin Jain 21.4k ● 34 ● 104 ● 168

1 So cross merging like this isn't an issue? – Mr. EZEKIEL May 2, 2013 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. – Sachin Jain May 2, 2013 at 4:33

8

Run the following commands:

```
$ git checkout mobiledevice
$ git pull origin master
```

This would merge all the latest commits to your branch. If the merge results in some conflicts, you'll need to fix them.

I don't know if this is the best practice but works for me.

Share Improve this answer Follow edited Oct 30, 2020 at 16:23 aalbagarcia 1,019 ● 7 ● 20 answered Oct 30, 2020 at 5:52 DEVSE 81 ● 1 ● 1

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.

Share Improve this answer Follow answered May 2, 2013 at 3:21 faisal

1,327 ● 10 ● 19

0

```
using yourBranch = mobiledevicesupport;
```

Keeping *yourBranch* in sync with `master` consists of two important details:

1. update the *yourBranch* version in the cloud according to `master`
2. update the *yourBranch* version on your PC according to `master`

How To:

Assuming *yourBranch* is behind `master`

1. create a merge request from `master` to *yourBranch* (e.g. via GitLab web UI)
2. accept/solve the merge request from `master` to *yourBranch* (e.g. via GitLab web UI)

- now you have changes from `master` in *yourBranch*
- this has loaded `master` changes to *yourBranch* on remote (in the cloud) but not locally yet

3. update the old version of *yourBranch* which you have locally (on your PC)

- make sure you are now locally interacting with your branch
- `git fetch` will check for changes
- `git status` will tell you the status of your local repository (of your local *yourBranch*) - e.g. 5 commits behind master
- `git pull` will download the new updated version from the cloud (from "remote") where changes from `master` are added - now you will be in sync with master also in the local repository (on your machine)

Share Improve this answer Follow

edited Mar 21, 2022 at 17:39

answered Mar 21, 2022 at 17:34
user3625699
147 ● 1 ● 9