git pull from master into the development branch

Ask Question

375

167

I have a branch called dmgr2 (development) and I want to pull from the master branch (live site) and incorporate all the changes into my development branch. is there a better way to do this? here is what I had planned on doing, after committing changes:

```
git checkout dmgr2
git pull origin master
```

this should pull the live changes into my development branch, or do I have this wrong?

git branch pull

edited Aug 22 '18 at 8:29
Mosh Feu
16.2k 11 54 86

asked Nov 20 '13 at 16:53
Matthew Colley
2,765 9 32 57

1 first commit all your changes in dmgr2 branch. and then point to master 1.git checkout master and then get the latest change 2.git pull 3.git merge dmgr2 4.git push -u origin master And then go back to your dmgr2 5.git checkout dmgr2 – mat_vee Nov 20 '13 at 16:57

i have already committed all my changes to the dmgr2 branch, sorry forgot to add that – Matthew Colley Nov 20 '13 at 16:58

if i perform step 4, wont that push my development changes into master? i dont want to do that – Matthew Colley Nov 20 '13 at 17:17

So what you're saying is you want to bring the changes from your master branch, into your dev branch? – JcKelley Nov 20 '13 at 17:49

5 Switch to `dev` branch with a `git checkout dev`. Then `git pull --rebase origin master`. If you are lucky, there will be no conflicts and dev will have the latest changes from

master. – jww May 2 '16 at 12:06

3 Answers

545

✓

The steps you listed will work, but there's a longer way that gives you more options:

```
git checkout dmgr2      # gets you "c
git fetch origin        # gets you up
git merge origin/master
```

The `fetch` command can be done at any point before the `merge` , i.e., you can swap the order of the fetch and the checkout, because `fetch` just goes over to the named remote (`origin`) and says to it: "gimme everything you have that I don't", i.e., all commits on all branches. They get copied to your repository, but named `origin/branch` for any branch named `branch` on the remote.

At this point you can use any viewer (`git log` , `gitk` , etc) to see "what they have" that you don't, and vice versa. Sometimes this is only useful for Warm Fuzzy Feelings ("ah, yes, that is in fact what I want") and sometimes it is useful for changing strategies entirely ("whoa, I don't want THAT stuff yet").

Finally, the `merge` command takes the given commit, which you can name as `origin/master` , and does whatever it takes to bring in that commit and its ancestors, to whatever branch you are on when you run the `merge` . You can insert `--no-ff` or `--ff-only` to prevent a fast-forward, or merge only if the result is a fast-forward, if you like.

When you use the sequence:

```
git checkout dmgr2
git pull origin master
```

the `pull` command instructs git to run `git fetch` , and then the moral equivalent of `git merge origin/master` . So this is *almost* the same as doing the two steps by hand, but there are some subtle differences that probably are not too concerning to you. (In particular the `fetch` step run by `pull` brings over *only* `origin/master`, and it does not update the ref in your repo:[1] any new commits winds up referred-to only by the special `FETCH_HEAD` reference.)

If you use the more-explicit `git fetch origin` (then optionally look around) and then `git merge origin/master` sequence, you can also bring your own local `master` up to date with the remote, with only one `fetch` run across the network:

```
git fetch origin
git checkout master
git merge --ff-only origin/master
git checkout dmgr2
git merge --no-ff origin/master
```

for instance.

[1]This second part has been changed—I say "fixed"—in git 1.8.4, which now updates "remote branch" references opportunistically. (It was, as the release notes say, a deliberate design decision to skip the update, but it turns out that more people prefer that git update it. If you want the old remote-branch SHA-1, it defaults to being saved in, and thus recoverable from, the reflog. This also enables a new git 1.9/2.0 feature for finding upstream rebases.)

edited Mar 11 '14 at 6:25

answered Nov 20 '13 at 17:58
torek
203k 20 254 340

22 I'm just asking for a, uh, friend - how would you go about undoing the first code block you have here (checkout/fetch/merge)? – Rich Bradshaw Sep 17 '14 at 9:13

5 @RichBradshaw: `git checkout` is normally non-destructive and there's normally no reason to undo a `git fetch`, so it sounds like you're asking how to back out a merge commit. The answer is the same as

for other commits: either `git reset` or `git revert`. For *unpublished* changes `git reset` is usually the best method; for changes others already have, `git revert` may be better, but see Linus Torvald's advice on reverting a merge: kernel.org/pub/software/scm/git/docs/howto/… – torek Sep 17 '14 at 16:15

How do you set which merge viewer program to use? – PositiveGuy Aug 10 '15 at 19:22

2 @WeDoTDD: I don't understand the question. There are a number of commands for viewing the commit graph (`gitk` , `git log --graph` with or without `--oneline` , and so on) and you can `git show` or `git show -m` a merge commit, or use `git diff` . In all of these cases, you're specifying the program as you enter the command on the command-line. – torek Aug 10 '15 at 20:02

Thanks for explanation, these subtleties with FETCH_HEAD refs (and possibly other issues which were unknown to me as a Git newbie) were the reasons why I hesitated to pull from a different branch. I always went the long way - `checkout, pull, checkout, merge` . But now I might give a try for straight `pull` . – JustAMartin Oct 1 '15 at 7:54

8

Situation: Working in my local branch, but I love to keep-up updates in the development branch named `dev` .

Solution: Usually, I prefer to do :

```
git fetch
git rebase origin/dev
```

edited Aug 4 '18 at 21:08
David C.
1,443 1 13 24

answered Apr 23 '18 at 15:19
greenridinghood
91 1 3

8 With the usual disclaimer that rebase should only be done if the local branch is local only, that is, have not been pushed anywhere as it rewrites history. – Locus Apr 24 '18 at 8:51

history. – Locus Apr 24 '18 at 8:51

0

Scenario:

I have master updating and my branch updating, I want my branch to keep track of master with rebasing, to keep all history tracked properly, let's call my branch Mybranch

Solution:

```
git checkout master
git pull --rebase
git checkout Mybranch
git rebase master
git push -f origin Mybranch
```

- need to resolve all conflicts with git mergetool &, git rebase --continue, git rebase --skip, git add -u, according to situation and git hints, till all is solved

(correction to last stage, in courtesy of Tzachi Cohen, using "-f" forces git to "update history" at server)

now branch should be aligned with master and rebased, also with remote updated, so at git log there are no "behind" or "ahead", just need to remove all local conflict *.orig files to keep folder "clean"

edited Feb 14 at 11:02

answered Feb 13 at 10:48

user10556443

78 8