Resolve Git merge conflicts in favor of their changes during a pull

Asked 7 years, 6 months ago Active 1 month ago Viewed 706k times

1102

316

How do I resolve a git merge conflict in favor of pulled changes?

Basically I need to remove all conflicting changes from a working tree without having to go through all of the conflicts with a `git mergetool` while keeping all conflict-free changes. Preferably doing this while pulling, not afterwards.

git git-merge git-merge-conflict

edited May 6 '15 at 3:36

asked May 22 '12 at 7:18
sanmai
18.7k 9 44 69

2 possible duplicate of [git merge -s ours, what about "theirs"]() – user456814 Apr 12 '14 at 2:41

1 Duplicate of [git pull from remote.. can I force it to overwrite rather than report conflicts?]() You can see the same solution there. – Dan Dascalescu Jul 15 '14 at 0:39

3 @DanDascalescu Accepted answer there doesn't answer this questions, so clearly it isn't a duplicate. Plus, that other question is quite ambiguous: it is very hard to tell what is asked. All in all I can't agree with you. What is you point in this? – sanmai Jul 15 '14 at 0:44

1 @sanmai You have two answers - and you accepted one of them. Can you better explain what you are expecting in an answer and how much more detail do you want here? – Edward Thomson Jul 25 '14 at 2:31

2 @EdwardThomson well, actually I was thinking to give this reputation for the first answer, but if you ask, I might wait and see if a better answer comes up – sanmai Jul 25 '14 at 2:42

10 Answers

1006

+100

Or, simply, for the default repository:

```
git pull -X theirs
```

[If you're already in conflicted state...](#)

```
git checkout --theirs path/to/file
```

edited Jun 6 '17 at 0:21
sanmai
18.7k 9 44 69

answered Feb 14 '14 at 11:06
Pascal Fares
10.4k 1 9 11

37 Note that `-s recursive` here is redundant, since that's the default merge strategy. So you could simplify it to `git pull -X theirs`, which is basically equivalent to `git pull --strategy-option theirs`. – user456814 Jul 28 '14 at 3:26

3 If I do this, I end up back in the `MERGING` state. I can then `git merge --abort` and try again, but each time I end up with a merge occurring. … I know that a rebase was pushed to my upstream though, so perhaps that's causing this? – Benjohn Jul 14 '16 at 9:03

19 Be careful with `git checkout --theirs path/to/file`. Used it during rebase and got unexpected results. Found explanation in doc: *Note that during git rebase and git pull --rebase, ours and theirs may appear swapped; --ours gives the version from the branch the changes are rebased onto, while --theirs gives the version from the branch that holds your work that is being rebased.* – Vuk Djapic Jul 6 '17 at 15:13

9 Note that `git checkout --theirs/--ours path` man page states that it works for **unmerged paths**. So if there were no conflict in path, it is already merged this command will do nothing. This might case issues when you want for example 'theirs' version of a whole sub-folder. So in such case it would be safer to do `git checkout MERGE_HEAD path` or use commit hash. – fsw Aug 23 '17 at 9:17

3 `git pull -X theirs` creates a merge commit if there are conflicts (e.g. if another committer ran `git push -f` to the remote). If you don't want merge commits, run instead `git fetch && git reset --hard origin/master`. – Dan Dascalescu Jan 23 '18 at 10:02

937

You can use the recursive "theirs" strategy *option*:

```
git merge --strategy-option theirs
```

From the [man](#):

```
ours
```

```
        This should not be confused with the ours merge strategy, which does
        not even look at what the other tree contains at all. It discards
        everything the other tree did, declaring our history contains all that
        happened in it.

    theirs
        This is opposite of ours.
```

Note: as the man page says, the "ours" merge *strategy option* is very different from the "theirs" merge *strategy*.

edited Oct 18 at 13:02
Vaibhav Shukla
185 1 2 13

answered May 22 '12 at 7:24
Ikke
82.3k 23 85 112

Here is more detailed explanation: lostechies.com/joshuaflanagan/2010/01/29/… – mPrinC Sep 17 '13 at 0:34

224 Also `git checkout --theirs` to operate on a single conflicting file – dvd Jul 31 '14 at 12:24

46 This doesn't work if you are already in the conflict resolution state. In that case I believe the best way to resolve is to `git checkout <ref to theirs> -- the/conflicted.file` ; and then `git add` their changes. – ThorSummoner Sep 12 '14 at 17:58

54 @ThorSummoner In that case, there is git checkout --theirs path/of/file. That way, you don't have to manually look up the correct hash. – Ikke Sep 13 '14 at 19:26

8 @Ikke That should basically be its own answer (and the accepted answer at that) if you ask me. – ThorSummoner Sep 14 '14 at 2:16

7

After git merge, if you get conflicts and you want either your or their

```
git checkout --theirs .
git checkout --ours.
```

answered Sep 13 at 15:36
Mohit Kanojia
71 1 3

11

If you want to accept ***all the incoming changes in the conflict file*** then do the following steps.

```
1. Go to command palette - Ctrl + Shift + P
2. Select the option - Merge Conflict: Accept All Incoming
```

Similarly you can do for other options like **Accept All Both, Accept All Current etc.,**

edited Nov 2 '18 at 10:02

answered Nov 1 '18 at 16:17

SridharKritha
3,213 1 26 29

2 That seems to only work for a single file, not all files with conflicts though. – Yoryo Jul 18 at 16:19

33

The `git pull -X theirs` answers may create an ugly merge commit, or issue an

> error: Your local changes to the following files would be overwritten by merge:

If you want to simply ignore any local modifications to files from the repo, for example on a client that should always be a mirror of an origin, run this (replace `master` with the branch you want):

```
git fetch && git reset --hard origin/master
```

How does it work? `git fetch` does `git pull` but without merge. Then `git reset --hard` makes your working tree match the last commit. All of your local changes to files in the repo will be discarded, but new local files will be left alone.

answered Jan 23 '18 at 11:00

Dan Dascalescu
83.5k 23 224 303

1 +1 - `git fetch && git reset --hard {remote}/{branch}` was what solved my problem. I needed to completely ditch my own changes in favour of "theirs" state of a branch, but the `git pull -X theirs` choked on some moved/renamed files. Thanks! – Ivaylo Slavov Feb 5 '18 at 18:28

-1

> from https://git-scm.com/book/en/v2/Git-Tools-Advanced-Merging
>
> This will basically do a fake merge. It will record a new merge commit with both branches as parents, but it will not even look at the branch you're merging in. It will simply record as the result of the merge the exact code in your current branch.
>
> ```
> $ git merge -s ours mundo
> ```
>
> Merge made by the 'ours' strategy.
>
> ```
> $ git diff HEAD HEAD~
> ```
>
> You can see that there is no difference between the branch we were on and the result of the merge.
>
> This can often be useful to basically trick Git into thinking that a branch is already merged when doing a merge later on. For example, say you branched off a release branch and have done some work on it that you will want to merge back into your master branch at some point. In the meantime some bugfix on master needs to be backported into your release branch. You can merge the bugfix branch into the release branch and also merge -s ours the same branch into your master branch (even though the fix is already there) so when you later merge the release branch again, there are no conflicts from the bugfix.

A situation I've found to be useful if I want master to reflect the changes of a new topic branch. I've noticed that -Xtheirs doesn't merge without conflicts in some circumstances... e.g.

```
$ git merge -Xtheirs topicFoo

CONFLICT (modify/delete): js/search.js deleted in HEAD and modified in topicFoo. Version
topicFoo of js/search.js left in tree.
```

In this case the solution I found was

```
$ git checkout topicFoo
```

from topicFoo, first merge in master using the -s ours strategy, this will create the fake commit that is just the state of topicFoo. $ git merge -s ours master

check the created merge commit

```
$ git log
```

```
$ git checkout master
```

merge the topic branch back but this time use the -Xtheirs recursive strategy, this will now present you with a master branch with the state of topicFoo.

```
$ git merge -X theirs topicFoo
```

answered Oct 5 '17 at 17:29

Amos Folarin
1,651 16 17

17

Please not that sometimes this will **not work**:

> git checkout --ours path/to/file

or

> git checkout --theirs path/to/file

I did this instead, assuming **HEAD is ours** and **MERGE_HEAD is theirs**

```
git checkout HEAD -- path/to/file
```

or:

```
git checkout MERGE_HEAD -- path/to/file
```

After we do this and we are good:

```
git add .
```

edited May 23 '17 at 12:10
Community ♦
1 1

answered Apr 27 '17 at 8:47
Nicolas D
1,002 11 33

19

To resolve all conflicts with the version in a particular branch:

```
git diff --name-only --diff-filter=U | xargs git checkout ${branchName}
```

So, if you are already in the merging state, and you want to keep the master version of the conflicting files:

```
git diff --name-only --diff-filter=U | xargs git checkout master
```

edited Dec 6 '15 at 8:54
CodeWizard
66.3k 12 88 110

answered Jul 14 '15 at 17:43
Ariel Alvarez
401 5 11

And for people on windows without access to pipe xargs, how does this translate? – tsemer Dec 9 '16 at 16:18

Wouldn't it skip your changes completely? Even those which are not in conflict state? – Valentin Heinitz Jan 15 '18 at 12:24

This is what I ended up doing. Unfortunately it subsequently requires either a `git cherry-pick --continue` *or* a `git commit --allow-empty` command to commit these changes, and there seems to be no system behind which command is required, which makes automating this a pain. I'm currently solving this by testing for the existence of a `.git/COMMIT_EDITMSG` file but that seems hacky and brittle, and I'm not yet convinced that it always works. – Konrad Rudolph May 7 at 16:41

417

If you're already in conflicted state, and you want to just accept *all* of theirs:

```
git checkout --theirs .
git add .
```

If you want to do the opposite:

This is pretty drastic, so make sure you really want to wipe everything out like this before doing it.

answered Nov 6 '15 at 15:18
lots-to-learn
4,371 1 6 8

44 or, don't use the `.` and specify the file(s) in place of the dot that you want to checkout. less "drastic" & exactly what you want to do, presumably. – manroe Dec 16 '15 at 0:56

9 this doesn't work if the file has been removed from the other branch: '<file>' does not have their version – Japster24 Mar 2 '16 at 16:52

1 Would upvote more if I could, keep coming back to this answer every now and then. – tarikki Sep 14 '16 at 4:04

6 Use `git add -u` instead to skip files that are not under version control. – Sudipta Basak Oct 27 '16 at 20:56

1 Hypothetical case, three changes in a file, one conflicting, two without conflict, would this solution apply non conflicting changes and resolve the conflicting change to ours? Or would it take only ours version ignoring non conflicting changes from theirs? – pedromarce Feb 28 '17 at 14:50

219

OK so, picture the scenario I was just in:

You attempt a `merge`, or maybe a `cherry-pick`, and you're stopped with

```
$ git cherry-pick 1023e24
error: could not apply 1023e24... [Commit Message]
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
```

Now, you view the conflicted file and you really don't want to keep your changes. In my case above, the file was conflicted on just a newline my IDE had auto-added. To undo your changes and accept their's, the easiest way is:

```
git checkout --theirs path/to/the/conflicted_file.php
git add path/to/the/conflicted_file.php
```

The converse of this (to overwrite the incoming version with your version) is

Surprisingly, I couldn't find this answer very easily on the Net.

answered Aug 1 '14 at 12:11
Theodore R. Smith
16.2k 9 41 71

14 Note that, if one performs a `git status` between `checkout` and `add` , the file still shows as "both modified". – bishop Mar 25 '16 at 12:56

Do you know if there is a way to do this for all files that are in a conflicted state? If so it'd be a nice extension to your answer. – Drew Noakes Mar 6 '18 at 8:42

2 I do this: `git reset --hard` and then `git pull [remote server name] [branch name] -Xtheirs` (undoes the merge then pulls the new stuff on top of my stuff) - not sure if this what you want. – ssaltman May 22 '18 at 19:17