How do I force “git pull” to overwrite local files?

Asked 10 years, 11 months ago Active 19 days ago Viewed 4.9m times

7262

2698

How do I force an overwrite of local files on a `git pull` ?

The scenario is the following:

- A team member is modifying the templates for a website we are working on
- They are adding some images to the images directory (but forgets to add them under source control)
- They are sending the images by mail, later, to me
- I'm adding the images under the source control and pushing them to GitHub together with other changes
- They cannot pull updates from GitHub because Git doesn't want to overwrite their files.

This is the error I'm getting:

> error: Untracked working tree file 'public/images/icon.gif' would be overwritten by merge

How do I force Git to overwrite them? The person is a designer - usually, I resolve all the conflicts by hand, so the server has the most recent version that they just need to update on their computer.

git version-control overwrite git-pull git-fetch

edited Dec 2 '19 at 8:05
Mohammad Masoumi
2,399 3 14 27

asked Jul 14 '09 at 14:58
Jakub Troszok
77.1k 9 35 47

17 anyone reading this who thinks they might lose files, I've been in this position and found Sublime Text's buffer has saved me - if I'm working on something, then accidentally delete everything by trying to solve a similar problem to this or by using an answer on this question and have had the files open in Sublime (which there's a good chance of) then the files will still be there is Sublime, either just there, or in the undo history – Toni Leigh Jan 20 '16 at 8:51

70 `git reset --hard origin/branch_to_overwrite` – Andrew Atkinson Mar 22 '16 at 8:37

1 basically, only do a pull from develop after the initial checkout -b. do your work, then push back in. – ldgorman Aug 22 '18 at 9:09

1 Short answer: delete and re-create branch. 1. Delete branch: `git branch <branch> -D` 2. Reset to a commit before the conflict: `git reset <commit> --hard` 3. Re-create the branch: `git branch <branch>` 4. Set tracking to the server: `git --set-upstream-to=origin/<branch> <branch> 5. Pull:` git pull` – Nino Filiu Sep 24 '18 at 8:54

1 To change all CRLF to LF endings, (start clean) `git config core.autocrlf false; git ls-files -z | xargs -0 rm; git checkout .` – Chloe Jan 17 '19 at 2:46

1 2 Next

10197

21

⚠ Important: If you have any local changes, they will be lost. With or without `--hard` option, any local commits that haven't been pushed will be lost.[*]

If you have any files that are *not* tracked by Git (e.g. uploaded user content), these files will not be affected.

I think this is the right way:

```
git fetch --all
```

Then, you have two options:

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

OR If you are on some other branch:

```
git reset --hard origin/<branch_name>
```

Explanation:

`git fetch` downloads the latest from remote without trying to merge or rebase anything.

Then the `git reset` resets the master branch to what you just fetched. The `--hard` option changes all the files in your working tree to match the files in `origin/master`

Maintain current local commits

[*]: It's worth noting that it is possible to maintain current local commits by creating a branch from `master` before resetting:

```
git checkout master
git branch new-branch-to-save-current-commits
git fetch --all
git reset --hard origin/master
```

After this, all of the old commits will be kept in `new-branch-to-save-current-commits .`

Uncommitted changes

Uncommitted changes, however (even staged), will be lost. Make sure to stash and commit anything you need. For that you can run the following:

```
git stash pop
```

edited Jun 17 at 9:06

mikemaccana
70.6k 59 282 358

answered Jan 17 '12 at 0:02

RNA
114k 11 42 58

14 Watch out! If you have local unpushed commits this will remove them from your branch! This solution keeps untracked files not in the repository intact, but overwrites everything else. – Matthijs P May 17 '12 at 8:18

480 It's a popular question, so I'd like to clarify on the top comment here. I just executed commands as described in this answer and it hasn't removed ALL the local files. Only the remotely tracked files were overwritten, and every local file that has been here was left untouched. – Red Nov 22 '12 at 10:38

15 in case you're pulling from a repo that has its remote branch name different from "master", use `git reset --hard origin/branch-name` – Nerrve Dec 17 '13 at 11:17

104 Given the amount of upvotes to this question and answer, I think that git should incorporate a command like `git pull -f` – Sophivorus Aug 26 '14 at 1:33

8 Commits that weren't pushes before the hard reset can be recovered using `git reflog`, which list all commits, also those without a base. Until you cleanup your local copy using `git gc`, then all is lost – Koen. Feb 10 '15 at 22:24

941

1

Try this:

```
git reset --hard HEAD
git pull
```

It should do what you want.

edited Jan 14 '17 at 15:10

Peter Mortensen
26.2k 21 90 120

answered May 9 '10 at 19:45

Travis Reeder
28.2k 10 71 77

16 I've done this and some local files that were no longer in repo were left on the disk. – Piotr Owsiak Apr 8 '11 at 16:00

27 I do not think that this is correct. the above will perform a merge, not overwrite which was requested in the question: "How to force git to overwrite them?" I do not have the answer, I am currently looking for it.. at the moment I switch to the branch with with the code that I want to keep "git checkout BranchWithCodeToKeep", then do "git branch -D BranchToOverwrite" and then finally "git checkout -b BranchToOverwrite". you will now have the exact code from BranchWithCodeToKeep on the branch BranchToOverwrite without having to perform a merge. – felbus Jul 13 '11 at 10:11

253 instead of merging using 'git pull', try git fetch --all followed by 'git reset --hard origin/master' – Lloyd Moore Feb 21 '12 at 14:56

5 yep, the @lloydmoore solution worked for me. Could do with being an answer rather than just a comment. – Max Williams Nov 19 '12 at 9:54

2 This will reset the current changes back to the last branch commit pulled. Then git pull merges the changes from the latest branch. This did exactly what I wanted it to do.. Thanks! – Codeversed Dec 5 '14 at 17:42

464

WARNING: `git clean` deletes all your untracked files/directories and can't be undone.

Sometimes just `clean -f` does not help. In case you have untracked DIRECTORIES, -d option also needed:

```
# WARNING: this can't be undone!

git reset --hard HEAD
git clean -f -d
git pull
```

1

WARNING: `git clean` deletes all your untracked files/directories and can't be undone.

Consider using `-n` (`--dry-run`) flag first. This will show you what will be deleted without actually deleting anything:

```
git clean -n -f -d
```

Example output:

```
Would remove untracked-file-1.txt
Would remove untracked-file-2.txt
Would remove untracked/folder
...
```

edited Aug 17 '18 at 19:32

answered Mar 19 '11 at 9:10

David Avsajanishvili
6,063 2 18 22

33 Awesome... Ran this against my dotfiles repo... In my home directory. Good that I didn't really have anything important there... – Lauri Dec 11 '11 at 10:35

7 I think the scenario description makes it clear that he doesn't really want to throw away the content. Rather what he wants is to stop git baulking at overwriting the files. @Lauri, this should not have happened to you. Unfortunately people seem to have misread the essence of scenario description - see my suggestion. – Hedgehog Feb 11 '12 at 23:05

19 **FINALLY**. git clean -f -d is handy when make clean fails to clean everything. – earthmeLon Jun 23 '12 at 4:32

7 @crizCraig unless they are added in `.gitignore` – Bleeding Fingers Jun 13 '13 at 6:58

6 @earthmeLon, for that you might want `git clean -dfx` . The `-x` ignores .gitignore. Typically your build products will be in .gitignore. – Paul Draper Aug 12 '15 at 18:28

390

Like Hedgehog I think the answers are terrible. But though Hedgehog's answer might be better, I don't think it is as elegant as it could be. The way I found to do this is by using "fetch" and "merge" with a defined strategy. Which should make it so that your local changes are preserved as long as they are not one of the files that you are trying to force an overwrite with.

First do a commit of your changes

Then fetch the changes and overwrite if there is a conflict

```
git fetch origin master
git merge -s recursive -X theirs origin/master
```

"-X" is an option name, and "theirs" is the value for that option. You're choosing to use "their" changes, instead of "your" changes if there is a conflict.

edited Feb 23 '17 at 13:45

Veve
5,856 5 32 51

answered Apr 11 '12 at 20:13

Richard Kersey
4,253 1 10 14

58 This is the best answer I've seen so far. I haven't tried it, but unlike other answers, this doesn't attempt to nuke all your untracked files, which is very dangerous for obvious reasons. – huyz May 7 '12 at 9:36

5 Ditto - this worked for me when doing a very large merge (GitHub pull request) where I just wanted to accept it all on top of what I had. Good answer! In my case the last two commands were: 1) `get fetch other-repo` ; 2) `git merge -s recursive -X theirs other-repo/master` – quux00 Jul 27 '12 at 1:44

2 This will overwrite any conflicts with the repositories files and not your local ones, correct? – Nathan F. Dec 5 '14 at 11:40

2 Best answer. The highest accepted answer left me in my case on detached head. I switched back to local master branch and ran `git merge -X theirs origin/master` – petergus Mar 11 '16 at 12:46

2 The problem with this (excellent) answer, is it adds the all the local files, which sometimes may not be what you want. You may just want to add the specific files that were omitted. But the best thing about it is, it gets him to do what he should have done -- add them locally. You probably won't need the -X theirs strategy, since they're the same image. In fact, I'd suggest leaving it off at first, just to find out if there are any anomalies, and add it in if there are, after reviewing that 'theirs' is always the correct choice. But then, I'm paranoid. – Bob Kerns Jun 28 '18 at 20:27

282

2

Instead of doing:

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

I'd advise doing the following:

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

No need to fetch all remotes and branches if you're going to reset to the origin/master branch right?

edited Sep 14 '16 at 9:46

answered Apr 26 '13 at 13:48

Johanneke
3,861 2 15 32

3 Your answer is just what you needed for your rep. I must ask, does this also remove all untracked files? – Nicolas De Jay Jan 7 '14 at 6:38

1 See the comments on this other answer: stackoverflow.com/a/8888015/2151700 – Johanneke Jan 9 '14 at 12:02

This did not remove my untracked files; which is actually what I'd expect. Is there a reason it might for some people and not for others? – arichards Apr 19 '16 at 15:27

Untracked files are not affect4ed by git reset. If you want them to be removed as well, do `git add .` first, before `git reset --hard` – Johanneke Aug 15 '17 at 9:12

131

It looks like the best way is to first do:

```
git clean
```

To delete all untracked files and then continue with the usual `git pull` ...

edited Jan 14 '17 at 15:10

Peter Mortensen
26.2k 21 90 120

answered Jul 14 '09 at 15:16

Jakub Troszok
77.1k 9 35 47

4 I tried using "git clean" to solve the same issue, but it did not resolve it. git status says "Your branch and 'origin/master' have diverged, # and have 2 and 9 different commit(s) each, respectively." and git pull says something similar to what you have above. – slacy Sep 24 '09 at 4:25

43 git clean is a rather blunt instrument, and could throw away a lot of things that you may want to keep. Better to remove or rename the files that git is complaining about until the pull succeeds. – Neil Mayhew Jul 2 '10 at 13:21

2 I do not think this works in general. Isn't there a way to do basically a git clone remote via a forced git pull? – mathtick Nov 29 '10 at 18:30

10 @mathick: `git fetch origin && git reset --hard origin/master` – Arrowmaster Feb 23 '11 at 4:24

3 Is `git clean` the best answer here? Seems like removing files isn't necessarily what the OP wants. They asked for 'an overwrite of local files' not deletion. – JohnAllen Mar 4 '14 at 8:28

112

Warning, doing this will permanently delete your files if you have any directory/* entries in your gitignore file.

Some answers seem to be terrible. Terrible in the sense of what happened to @Lauri by following David Avsajanishvili suggestion.

Rather (git > v1.7.6):

```
git stash --include-untracked
git pull
```

Later you can clean the stash history.

Manually, one-by-one:

```
$ git stash list
stash@{0}: WIP on <branch>: ...
```

```
$ git stash drop stash@{0}
$ git stash drop stash@{1}
```

Brutally, all-at-once:

```
$ git stash clear
```

Of course if you want to go back to what you stashed:

```
$ git stash list
...
$ git stash apply stash@{5}
```

edited Mar 7 '18 at 7:00

answered Feb 11 '12 at 23:00

Hedgehog
4,639 3 29 38

2 No I don't think so. Stashing just moves uncommitted files out of the way. The above also moves (stashes) files that git does not track. This prevents files that have been added to the remote, which have not yet pulled down to your machine - but which you have created (!) - to be pulled down. All without destroying the uncommitted work. Hope that makes sense? – Hedgehog Mar 20 '12 at 23:54

3 If you don't have 1.7.6, you can mimic `--include-untracked` simply by temporarily `git add` -ing your entire repo, then immediately stashing it. – nategood May 1 '12 at 22:48

3 I agree with Hedgehog. If you do the popular answers here, you are more than likely going to find you've inadvertently killed a lot of stuff that you didn't really want to lose. – Guardius Jan 31 '13 at 21:28

1 I had other untracked files--besides the one the merge/pull wanted to overwrite, so this solution worked best. `git stash apply` brought back all my untracked files with the exception (rightly) of the ones that the merge had already created: "already exists, no checkout." Worked perfectly. – BigBlueHat Apr 25 '13 at 4:55

2 This is the cleanest answer, and should be the accepted one. To save some typing you can use the short form: `git stash -u .` – ccpizza Mar 23 '17 at 8:30

93

You might find this command helpful to throw away local changes:

```
git checkout <your-branch> -f
```

And then do a cleanup (removes untracked files from the working tree):

```
git clean -f
```

If you want to remove untracked directories in addition to untracked files:

```
git clean -fd
```

edited Jan 14 '17 at 15:12

Peter Mortensen
26.2k 21 90 120

answered Aug 5 '10 at 18:06

Vishal
16.5k 16 68 90

Rather what he wants is to stop git baulking at overwriting the files. See my suggestion. – Hedgehog Feb 11 '12 at 23:03

3 Though that answer might not fit exactly the description, it still saved me from the frustration of git twiddling with the carriage returns (event with autocrlf false). When git reset --hard HEAD does not leave you with "no" modified files, these "-f" flags are quite helpful. Thanks a bunch. – Kellindil Jan 16 '13 at 10:28

88

Instead of merging with `git pull`, try this:

```
git fetch --all
```

followed by:

```
git reset --hard origin/master .
```

edited Mar 21 '18 at 7:21 — sbarb — 55 2 8

answered Nov 22 '12 at 10:56 — Lloyd Moore — 2,789 26 28

61

The only thing that worked for me was:

```
git reset --hard HEAD~5
```

This will take you back five commits and then with

```
git pull
```

I found that by looking up how to undo a Git merge.

edited May 23 '17 at 10:31 — Community ♦ — 1 1

answered May 5 '11 at 21:53 — Chris BIllante — 643 5 3

This was what ultimately worked for me as I had force pushed my branch to the origin repo and kept getting merge conflicts when trying to pull it to my remote repo.. – jwfrench May 7 '14 at 5:16

Hi, actually this is a trick for a `work around` but really effective. Because some conflicts may happen just in few commits then reverting 5 commits will make sure no conflicts with remote code. – Hoang Le Nov 21 '14 at 10:03

54

The problem with all these solutions is that they are all either too complex, or, an even bigger problem, is that they remove all untracked files from the web server, which we don't want since there are always needed configuration files which are on the server and not in the Git repository.

Here is the cleanest solution which we are using:

```
# Fetch the newest code
```

```
# are no conflicts with untracked files
for file in `git diff HEAD..origin/master --name-status | awk '/^A/ {print $2}'`
do
    rm -f -- "$file"
done

# Checkout all files which were locally modified
for file in `git diff --name-status | awk '/^[CDMRTUX]/ {print $2}'`
do
    git checkout -- "$file"
done

# Finally pull all the changes
# (you could merge as well e.g. 'merge origin/master')
git pull
```

- The first command fetches newest data.
- The second command checks if there are any files which are being added to the repository and deletes those untracked files from the local repository which would cause conflicts.
- The third command checks-out all the files which were locally modified.
- Finally we do a pull to update to the newest version, but this time without any conflicts, since untracked files which are in the repo don't exist anymore and all the locally modified files are already the same as in the repository.

edited May 23 '14 at 21:14
user456814

answered Nov 5 '12 at 23:32

Strahinja Kustudic
2,957 20 17

Using "git merge origin/master" as the last line (like you say in your note) instead of "git pull" will be faster as you've already pulled down any changes from the git repo. – Josh May 6 '13 at 6:21

1 Yeah of course, `git merge origin/master` will be faster and probably even safer. Since if someone pushed new changes during the removal of of files of this script (which is not likely to happen, but possible), the whole pull could fail. The only reason I put `pull` in there is because someone might not be working on the master branch, but some other branch and I wanted the script to be universal. – Strahinja Kustudic Sep 1 '13 at 22:25

If you have locally created files like option files, put them in `.gitignore` . – Sebi Nov 21 '17 at 11:41

53

First of all, try the standard way:

```
git reset HEAD --hard # To remove all not committed changes!
git clean -fd         # To remove all untracked (non-git) files and folders!
```

Warning: Above commands can results in data/files loss only if you don't have them committed! If you're not sure, make the backup first of your whole repository folder.

Then pull it again.

If above won't help and you don't care about your untracked files/directories (make the backup first just in case), try the following simple steps:

This will REMOVE all git files (excempt `.git/` dir, where you have all commits) and pull it again.

Why `git reset HEAD --hard` could fail in some cases?

1. Custom rules in `.gitattributes file`

   Having `eol=lf` rule in .gitattributes could cause git to modify some file changes by converting CRLF line-endings into LF in some text files.

   If that's the case, you've to commit these CRLF/LF changes (by reviewing them in `git status`), or try: `git config core.autcrlf false` to temporary ignore them.

2. File system incompability

   When you're using file-system which doesn't support permission attributes. In example you have two repositories, one on Linux/Mac (`ext3` / `hfs+`) and another one on FAT32/NTFS based file-system.

   As you notice, there are two different kind of file systems, so the one which doesn't support Unix permissions basically can't reset file permissions on system which doesn't support that kind of permissions, so no matter how `--hard` you try, git always detect some "changes".

edited Jan 31 '19 at 15:48

answered Oct 26 '12 at 9:17

kenorb
101k 47 527 557

47

I had the same problem. No one gave me this solution, but it worked for me.

I solved it by:

1. Delete all the files. Leave just the `.git` directory.
2. `git reset --hard HEAD`
3. `git pull`
4. `git push`

Now it works.

edited Jan 18 '19 at 23:00

SherylHohman
9,925 14 55 69

answered Jan 12 '11 at 23:58

John John Pichler
3,656 4 34 67

1 Same here. Sometimes only the very hard solution works, it happens often that only reset and clean are not enough somehow... – jdehaan Dec 15 '11 at 11:28

42

Bonus:

In speaking of pull/fetch/merge in the previous answers, I would like to share an interesting

This above command is the most useful command in my Git life which saved a lot of time.

Before pushing your newly commit to server, try this command and it will automatically synchronise the latest server changes (with a fetch + merge) and will place your commit at the top in the Git log. There isn't any need to worry about manual pull/merge.

Find details in *What does "git pull --rebase" do?*.

edited Jun 20 at 9:12
Community ♦
1 1

answered Dec 23 '15 at 15:41
Sazzad Hissain Khan
24.6k 15 105 147

3 In short: `git pull -r .` – kenorb Oct 15 '19 at 9:29

29

I had a similar problem. I had to do this:

```
git reset --hard HEAD
git clean -f
git pull
```

edited Nov 6 '11 at 16:35
Alexsander Akers
15.5k 11 53 79

answered Jan 14 '11 at 15:18
Ryan
331 3 2

6 use `git clean` with caution – nategood Mar 30 '12 at 16:39

29

I summarized other answers. You can execute `git pull` without errors:

```
git fetch --all
git reset --hard origin/master
git reset --hard HEAD
git clean -f -d
git pull
```

Warning: This script is very powerful, so you could lose your changes.

edited Jan 14 '17 at 15:42
Peter Mortensen
26.2k 21 90 120

answered Aug 7 '15 at 3:03
Robert Moon
928 9 17

2 This will overwrite modified files (files that were previously checked in) and it will remove untracked files (files that have never been checked in). Exactly what I was looking for, thanks! – styfle Mar 3 '16 at 16:01

3 I suspect the third line `git reset --hard HEAD` may be redundant; my local man page (2.6.3) say that `reset` in the second line `git reset --hard origin/master` *"defaults to HEAD in all forms."* – arichards Apr 19 '16 at 15:40

2 @arichards I think your suspect is right but if second line will not work(by any reason) third line work well to reset. This solution doesn't need to be optimized. I just summarized other answers. That's all.

28

Based on my own similar experiences, the solution offered by Strahinja Kustudic above is by far the best. As others have pointed out, simply doing hard reset will remove **all** the untracked files which could include lots of things that you don't want removed, such as config files. What is safer, is to remove only the files that are about to be added, and for that matter, you'd likely also want to checkout any locally-modified files that are about to be updated.

That in mind, I updated Kustudic's script to do just that. I also fixed a typo (a missing ' in the original).

```
#/bin/sh

# Fetch the newest code
git fetch

# Delete all files which are being added,
# so there are no conflicts with untracked files
for file in `git diff HEAD..origin/master --name-status | awk '/^A/ {print $2}'`
do
    echo "Deleting untracked file $file..."
    rm -vf "$file"
done

# Checkout all files which have been locally modified
for file in `git diff HEAD..origin/master --name-status | awk '/^M/ {print $2}'`
do
    echo "Checking out modified file $file..."
    git checkout $file
done

# Finally merge all the changes (you could use merge here as well)
git pull
```

edited Aug 13 '15 at 23:12

Nathaniel Ford
15.8k 17 66 81

answered Feb 27 '13 at 14:43

Rolf Kaiser
499 6 8

Using "git merge origin/master" as the last line (like you say in your note) instead of "git pull" will be faster as you've already pulled down any changes from the git repo. – Josh May 6 '13 at 6:20

The checkout of modified files is needed, so this works 100% of times. I updated my script with that a long time ago, but forgot to update here as well. I also use it a little differently than you. I checkout files which have any type of modification, not just M, so it works all the time. – Strahinja Kustudic Sep 1 '13 at 22:48

24

I believe there are two possible causes of conflict, which must be solved separately, and as far as I can tell none of the above answers deals with both:

- Local files that are untracked need to be deleted, either manually (safer) or as suggested in other answers, by `git clean -f -d`
- Local commits that are not on the remote branch need to be deleted as well. IMO the easiest way to achieve this is with: `git reset --hard origin/master` (replace 'master' by whatever branch you are working on, and run a `git fetch origin` first)

edited Dec 12 '11 at 20:05

answered Dec 12 '11 at 19:54

22

An easier way would be to:

```
git checkout --theirs /path/to/file.extension
git pull origin master
```

This will override your local file with the file on git

answered May 5 '15 at 8:03
maximus 69
1,206 2 19 32

21

It seems like most answers here are focused on the `master` branch; however, there are times when I'm working on the same feature branch in two different places and I want a rebase in one to be reflected in the other without a lot of jumping through hoops.

Based on a combination of RNA's answer and torek's answer to a similar question, I've come up with this which works splendidly:

```
git fetch
git reset --hard @{u}
```

Run this from a branch and it'll only reset your local branch to the upstream version.

This can be nicely put into a git alias (`git forcepull`) as well:

```
git config alias.forcepull "!git fetch ; git reset --hard @{u}"
```

Or, in your `.gitconfig` file:

```
[alias]
  forcepull = "!git fetch ; git reset --hard @{u}"
```

Enjoy!

edited May 23 '17 at 12:18
Community ♦
1 1

answered Feb 25 '14 at 17:19
JacobEvelyn
3,275 1 33 43

This answer is also nice because it works regardless of which branch you are on! – leafmeal Sep 11 '18 at 20:14

19

I had the same problem and for some reason, even a `git clean -f -d` would not do it. Here is why: For some reason, if your file is ignored by Git (via a .gitignore entry, I assume), it still bothers about overwriting this with a later *pull*, but a *clean* will not remove it, unless you add `-x` .

edited Mar 18 '15 at 19:46
Peter Mortensen

answered Aug 3 '11 at 9:23
Tierlieb

19

I know of a much easier and less painful method:

```
$ git branch -m [branch_to_force_pull] tmp
$ git fetch
$ git checkout [branch_to_force_pull]
$ git branch -D tmp
```

That's it!

edited Sep 5 '18 at 16:52

Ricky McMaster
2,467 17 18

answered Sep 5 '15 at 18:23

ddmytrenko
758 7 15

18

I just solved this myself by:

```
git checkout -b tmp # "tmp" or pick a better name for your local changes branch
git add -A
git commit -m 'tmp'
git pull
git checkout master # Or whatever branch you were on originally
git pull
git diff tmp
```

where the last command gives a list of what your local changes were. Keep modifying the "tmp" branch until it is acceptable and then merge back onto master with:

```
git checkout master && git merge tmp
```

For next time, you can probably handle this in a cleaner way by looking up "git stash branch" though stash is likely to cause you trouble on the first few tries, so do first experiment on a non-critical project...

edited Jan 14 '17 at 15:13

Peter Mortensen
26.2k 21 90 120

answered Dec 3 '10 at 15:00

Simon B.
1,778 15 27

18

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

answered May 12 '19 at 15:47

Suge
2,439 2 37 63

I have a strange situation that neither `git clean` or `git reset` works. I have to remove the conflicting file from `git index` by using the following script on every untracked file:

Then I am able to pull just fine.

edited Dec 5 '17 at 4:35
Jacob Gunther
193 1 13

answered Sep 19 '11 at 14:18
Chen Zhang
187 1 3

14

Despite the original question, the top answers can cause problems for people who have a similar problem, but don't want to lose their local files. For example, see Al-Punk and crizCraig's comments.

The following version commits your local changes to a temporary branch (`tmp`), checks out the original branch (which I'm assuming is `master`) and merges the updates. You could do this with `stash` , but I've found it's usually easier to simply use the branch / merge approach.

```
git checkout -b tmp
git add *; git commit -am "my temporary files"
git checkout master

git fetch origin master
git merge -s recursive -X theirs origin master
```

where we assume the **other repository** is `origin master` .

edited Mar 18 '15 at 19:43
Peter Mortensen
26.2k 21 90 120

answered Oct 22 '14 at 17:31
Snowcrash
60k 51 176 286

13

These four commands work for me.

```
git reset --hard HEAD
git checkout origin/master
git branch -D master
git checkout -b master
```

To check/pull after executing these commands

```
git pull origin master
```

I tried a lot but finally got success with these commands.

answered Mar 20 '14 at 4:24
vishesh chandra
6,473 6 31 36

2 "git branch -D master" delete the branch. so be careful with it. I prefer to use "git checkout origin/master -b <new branch name>" which create a new branch with a new name and you done need 3,4 lines. Also recommended to use "git clean -f" as well. – Chand Priyankara Apr 5 '14 at 11:49

13

```
git fetch origin branchname
git checkout -f origin/branchname // This will overwrite ONLY new included files
git checkout branchname
git merge origin/branchname
```

So you avoid all unwanted side effects, like deleting files or directories you wanted to keep, etc.

edited Jan 14 '17 at 15:48
Peter Mortensen
26.2k 21 90 120

answered Oct 19 '15 at 9:54
user2696128
151 1 2

12

Reset the index and the head to `origin/master`, but do not reset the working tree:

```
git reset origin/master
```

answered Feb 15 '13 at 13:41
user811773

> I personally found this to be most useful. It then keeps your working tree so you can check it in again. For my issue, I had the same files deleted as being added so it was stuck. Weird, I know. – Jason Sebring Jan 4 '14 at 21:03

12

Requirements:

1. Track local changes so no-one here ever loses them.
2. Make the local repository match the remote origin repository.

Solution:

1. **Stash** the local changes.
2. **Fetch** with a **clean** of **files** and **directories** ignoring **.gitignore** and **hard reset** to **origin**.

```
git stash --include-untracked
git fetch --all
git clean -fdx
git reset --hard origin/master
```

edited Jan 14 '17 at 15:44
Peter Mortensen
26.2k 21 90 120

answered Sep 1 '15 at 23:00
vezenkov
2,960 22 26

---

1 2 Next

Highly active question. Earn 10 reputation in order to answer this question. The reputation requirement