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

Asked 10 years, 4 months ago Active 3 months ago Viewed 4.3m times

6655

2544

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

The scenario is 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 needs to update on their computer.

git version-control overwrite git-pull git-fetch

edited Dec 23 '18 at 15:33
Mark Amery
82.8k 38 290 335

asked Jul 14 '09 at 14:58
Jakub Troszok
71k 9 33 46

12 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

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

42 Answers

1 2 next

9262

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 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
```

And then to reapply these uncommitted changes:

```
git stash pop
```

edited Jul 17 '18 at 21:53

Ahmad Awais
15.4k 3 51 44

answered Jan 17 '12 at 0:02

RNA
104k 11 41 53

10 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

458 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

14 This worked for me and my local files were NOT deleted. – Tastybrownies May 21 '13 at 18:16

13 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

71 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

I did this to get it to work:

1

```
git push --all
```

On computer where I wanted the new branch to show:

```
git fetch --all
```

answered Aug 30 at 8:53

dadde

566 1 8 23

You could try

1

```
git pull --force
```

if you want to overwrite all the local files

answered Aug 22 at 16:34

Ultan Kearns

51 4

```
git fetch --all
```

3

then if you are on the master branch

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

else

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

SUJITKUMAR SINGH
41 3

11

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

answered May 12 at 15:47

Suge
2,385 2 35 62

48

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:

```
cd your_git_repo  # where 'your_git_repo' is your git repository folder
rm -rfv *         # WARNING: only run inside your git repository!
git pull          # pull the sources again
```

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 at 15:48

answered Oct 26 '12 at 9:17

kenorb

87k 37 469 490

45

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 at 23:00

SherylHohman

7,643 9 48 60

answered Jan 12 '11 at 23:58

John John Pichler

3,446 4 33 66

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

1

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

2: Delete Untracked Files

```
git clean -f
```

3: Pull the commits

```
git pull
```

Sources:

- https://git-scm.com/docs/git-reset
- https://git-scm.com/docs/git-clean/

answered Nov 10 '18 at 2:27

abhijithvijayan
147 2 7

18

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

Blodwyn Pig
2,061 16 16

answered Sep 5 '15 at 18:23

ddmytrenko
728 7 15

441

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
```

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
5,758 2 16 22

31 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

6 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

4 @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

1

if you want to reset to the remote tracking branch in a generic way use:

```
git fetch
git reset --keep origin/$(git rev-parse --abbrev-ref HEAD)
```

if you want to reset your local changes too:

```
git fetch
git reset --hard origin/$(git rev-parse --abbrev-ref HEAD)
```

edited Jun 21 '18 at 7:27

answered May 25 '18 at 6:31

warch
1,132 11 29

1 If you find yourself using this frequently add a bash shortcut `alias gplf='git fetch && echo "HEAD was at $(git rev-parse --short HEAD)" && git reset --hard origin/$(git rev-parse --abbrev-ref HEAD)'` – Paul Odeon Nov 13 at 9:43

38

Bonus:

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

```
git pull --rebase
```

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 Apr 27 '18 at 13:02

Peter Mortensen
24.1k 19 89 118

answered Dec 23 '15 at 15:41

Sazzad Hissain Khan
21k 10 86 127

3

On Windows, do this single command:

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

edited Apr 27 '18 at 12:56

Peter Mortensen
24.1k 19 89 118

answered Apr 10 '18 at 7:22

Luca C.
4,970 50 51

87

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
2,720 25 27

109

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.

```
$ git stash list
stash@{0}: WIP on <branch>: ...
stash@{1}: 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

4,439 3 28 36

1 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

2 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

following script on every untracked file:

17

```
git rm [file]
```

Then I am able to pull just fine.

edited Dec 5 '17 at 4:35

Jacob Gunther
185 1 11

answered Sep 19 '11 at 14:18

Chen Zhang
187 1 3

Don't use `git reset --hard`. That will wipe their changes which may well be completely undesirable. Instead:

8

```
git pull
git reset origin/master
git checkout <file1> <file2> ...
```

You can of course use `git fetch` instead of `git pull` since it clearly isn't going to merge, but if you usually pull it makes sense to continue to pull here.

So what happens here is that `git pull` **updates your origin/master reference**; `git reset` **updates your local branch reference** on to be the same as origin/master without updating any files, so your checked-out state is unchanged; then `git checkout` **reverts files to your local branch index state** as needed. In cases where exactly the same file has been added on live and on upstream master, the index already matches the file following the reset, so in the common case you don't need to do `git checkout` at all.

If the upstream branch also contains commits which you want to apply automatically, you can follow a subtle variation on the process:

```
git pull
git merge <commit before problem commit>
git reset <problem commit>
git checkout <file1> <file2> ...
git pull
```

answered Nov 28 '17 at 11:19

Jim Driscoll
825 8 6

5

I was trying to use the Material2 branch on the Angular2-Webpack-Starter and had a heck of a time. This was the only way I could download and use that branch.

```
git clone --depth 1 https://github.com/angularclass/angular2-webpack-starter.git

cd angular2-webpack-starter/

git checkout -b material2
```

Open the project folder and delete all non-hidden files and folders. Leave all the hidden ones.

```
git add .

git commit -m "pokemon go"

git reset --hard

git pull origin material2
```

(When the editor pops up, hit ':wq', and then press Enter)

Now you are ready.

edited Apr 29 '17 at 0:46

MarkHu
1,299 13 23

answered Jul 27 '16 at 16:17

Helzgate
6,894 2 34 38

371

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

```
git add *
git commit -a -m "local file server commit message"
```

```
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
5,635 5 30 51

answered Apr 11 '12 at 20:13

Richard Kersey
4,022 1 9 13

56 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

4 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

1 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

1 Hang on...doesn't have `git add *` and `git commit -a <more-options-here>` have the same effect? Why would you need both? – Marcel Stör Apr 25 '17 at 20:41

13

Just do

```
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

answered Oct 19 '15 at 9:54

user2696128

9

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
24.1k 19 89 118

answered Sep 1 '15 at 23:00
vezenkov
2,682 22 26

28

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
24.1k 19 89 118

answered Aug 7 '15 at 3:03
Robert Moon
897 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

`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. Thank you for your comment. :) – Robert Moon Apr 20 '16 at 2:12

60

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 Bllante
633 5 3

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
24.1k 19 89 118

answered Dec 3 '10 at 15:00

Simon B.
1,481 11 23

92

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
24.1k 19 89 118

answered Aug 5 '10 at 18:06

Vishal
15.1k 15 64 85

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

884

Try this:

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

edited Jan 14 '17 at 15:10

Peter Mortensen
24.1k 19 89 118

answered May 9 '10 at 19:45

Travis Reeder
26.4k 10 69 72

15 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

25 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

245 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

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
24.1k 19 89 118

answered Jul 14 '09 at 15:16

Jakub Troszok
71k 9 33 46

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

42 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

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

270

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

3,681 1 14 31

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

5 Yeah, most of my rep is coming from here :) This will also remove all untracked files. Something I had forgotten and was painfully reminded of just 2 days ago... – Johanneke Jan 9 '14 at 12:01

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

7

This is the best practice for reverting changes:

- `git commit` Commit your staged changes so they will be saved in the **reflog** (see below)
- `git fetch` Fetch the latest upstream changes
- `git reset --hard origin/master` Hard reset to the origin master branch

The reflog *records branches and other references being updated in the local repository*. Or simply put - the **reflog** is the **history of your changes**.

answered Aug 12 '16 at 15:56

Jordan Georgiev
220 3 7

9

I read through all the answers but I was looking for a single command to do this. Here is what I did. Added a git alias to .gitconfig

```
[alias]
      fp = "!f(){ git fetch ${1} ${2} && git reset --hard ${1}/${2};};f"
```

Run your command as

```
git fp origin master
```

equivalent to

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

answered Jul 8 '16 at 13:11

Venkat Kotra
8,095 2 31 43

---

1 2 next