How do I properly force a Git push?

Asked 9 years, 3 months ago Active 1 year, 2 months ago Viewed 1.3m times

1287

328

I've set up a remote non-bare "main" repo and cloned it to my computer. I made some local changes, updated my local repository, and pushed the changes back to my remote repo. Things were fine up to that point.

Now, I had to change something in the remote repo. Then I changed something in my local repo. I realized that the change to the remote repo was not needed. So I tried to `git push` from my local repo to my remote repo, but I got an error like:

> To prevent you from losing history, non-fast-forward updates were rejected Merge the remote changes before pushing again. See the 'Note about fast-forwards' section of `git push --help` for details.

I thought that probably a

```
git push --force
```

would force my local copy to push changes to the remote one and make it the same. **It does force the update**, but when I go back to the remote repo and make a commit, I notice that the files contain outdated changes (ones that the main remote repo previously had).

As I mentioned in the comments to one of the answers:

> [I] tried forcing, but when going back to master server to save the changes, i get outdated staging. Thus, when i commit the repositories are not the same. And when i try to use git push again, i get the same error.

How can I fix this issue?

git push git-push git-non-bare-repository

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

asked Apr 1 '11 at 5:35
Spyros
38.2k 20 78 118

4 You will soon (git1.8.5, Q4 2013) be able to do a `git push -force` more carefully. – VonC Sep 10 '13 at 8:42

1 Related: Force git to overwrite remote files on push. – user456814 Jul 15 '14 at 21:42

6 As I detail in my own answer, `git push --force` is indeed another valid way to force push, and will push branches just as well as `git push origin master --force` with Git's default `push.default config settings`, though which branches specifically get pushed differs between Git versions prior to 2.0 versus after 2.0. – user456814 Aug 5 '14 at 17:51

2 `git push --force` works fine these days, FWIW... – rogerdpack Apr 10 '18 at 20:25

8 Answers

Active Oldest Votes

2335

+150

3

Just do:

```
git push origin <your_branch_name> --force
```

or if you have a specific repo:

```
git push https://git.... --force
```

This will delete your previous commit(s) and push your current one.

It may not be proper, but if anyone stumbles upon this page, thought they might want a simple solution...

Short flag

Also note that `-f` is short for `--force`, so

```
git push origin <your_branch_name> -f
```

will also work.

edited Jun 21 '17 at 13:19
Alex Zhukovskiy
7,755 2 42 112

answered Sep 26 '12 at 21:31
Katie
33.8k 16 72 101

59 You can use `git push origin +master` instead, which allow you push multiple refspecs without forcing them all. – nickgrim Nov 20 '13 at 10:36

5 Be aware that, if you accidentally do just `git push --force`, you might end up messing you master branch (depending on your push default behavior).. Which might suck.. a bit.. :D – Jeewes Aug 6 '14 at 7:18

9 @Jeewes starting with Git version 2.0, the *default* behavior of `git push --force` is basically to force push the currently checked-out branch to its remote-counter part, so if you have the master branch checked out, then it's identical to `git push origin master --force`. It'll be different if you're using the `matching` setting for `push.default`, which is the default for Git versions prior to 2.0. `matching` pushes *all* locals branches to remote ones that have the same name, so force pushing then could definitely be not what you want to do... – user456814 Aug 12 '14 at 14:22

@Jeewes But with Git 2.0, the default is safer, or at least it's no more dangerous than `git push origin master --force` is. – user456814 Aug 12 '14 at 14:24

2 push -f is good but not recoomended for master since most corporate repositories have -f disabled for master. the `merge -s ours` worked for me – mihai May 26 '16 at 21:27

And if `push --force` doesn't work you can do **`push --delete`**. Look at 2^{nd} line on this instance:

But beware...

Never ever go back on a public git history!

In other words:

- Don't ever `force` push on a public repository.
- Don't do this or anything that can break someone's `pull`.
- Don't ever `reset` or `rewrite` history in a *repo* someone might have already pulled.

Of course there are exceptionally rare exceptions even to this rule, but in most cases it's not needed to do it and it will generate problems to everyone else.

Do a revert instead.

And **always be careful with what you push to a public repo**. Reverting:

```
git revert -n HEAD~3..HEAD  # prepare a new commit reverting last 3 commits
git commit -m "sorry - revert last 3 commits because I was not careful"
git push origin master  # regular push
```

In effect, **both** origin HEADs (from the **revert** and from the **evil reset**) will contain the same files.

edit to add updated info and more arguments around `push --force`

Consider pushing force with lease instead of push, but still prefer revert

Another problem `push --force` may bring is when someone push anything before you do, but after you've already fetched. If you push force your *rebased* version now you will **replace work from others**.

`git push --force-with-lease` introduced in the git 1.8.5 (thanks to @VonC comment on the question) tries to address this specific issue. Basically, it will bring an error and not push if the remote was modified since your latest fetch.

This is good if you're really sure a `push --force` is needed, but still want to prevent more problems. I'd go as far to say it should be the default `push --force` behaviour. But it's still far from being an excuse to force a `push`. People who *fetched* before your *rebase* will still have lots of troubles, which could be easily avoided if you had *reverted* instead.

And since we're talking about `git --push` instances...

Why would anyone want to force push?

@linquize brought a good push force example on the comments: **sensitive data**. You've wrongly leaked data that shouldn't be pushed. If you're fast enough, you can *"fix"* `*` it by forcing a push on top.

already, but you get the idea.

edited Jun 20 at 9:12
Community ♦
1 1

answered May 22 '13 at 22:03
cregox
14.4k 13 74 108

1 The problem, @rogerdpack, isn't if it's doable. It is. But it can sum up to a big disaster. The more someone do it (force push) and the less often you update (pull) from the public repo, the bigger the disaster. It can dismantle the world as you know it!!!111 At least the world comprising of that particular repository. – cregox Sep 24 '13 at 19:14

3 If you have sensitive data, force push it – linquize Nov 26 '13 at 4:58

3 @Cawas: I think he means that if you are trying to remove sensitive data from the repository, then you *want* to rewrite history. If you revert, the sensitive data is still there in the earlier commit. That said, if someone else has already pulled from the repository, then rewriting history won't help you prevent them from accessing the sensitive data - it's already too late at that point. – Stuart Golodetz Dec 12 '13 at 9:38

3 `git push origin master --delete # do a very very bad bad thing git push origin master # regular push` this actually solved my problem perfectly (on a repo with only me and my friend). maybe it's wrong for public repos but for a private one this is a life saver. – Can Poyrazoğlu Jan 26 '14 at 23:44

1 this happens automatically with some repo managers, a.k.a. auto-squash etc. force pushing after finishing a feature branch to reduce commits is common and expected. – FlavorScape Jan 11 '19 at 20:37

18

First of all, I would not make any changes directly in the "main" repo. If you really want to have a "main" repo, then you should only push to it, never change it directly.

Regarding the error you are getting, have you tried `git pull` from your local repo, and then `git push` to the main repo? What you are currently doing (if I understood it well) is forcing the push and then losing your changes in the "main" repo. You should merge the changes locally first.

answered Apr 1 '11 at 5:42
ubik
4,014 2 19 28

yes i tried a pull but i'm losing losing data because of that pull. I want to make my main repos as my local is, without first updating from the main. – Spyros Apr 1 '11 at 6:28

1 In that case use `git push -f` , but then if you change your main repo again, you have to go back to your local repo and `git pull` , so that it gets in sync with the latest changes. Then you can do your work, and push again. If you follow this "push-pull" workflow, you won't get the kind of error you were complaining about. – ubik Apr 1 '11 at 17:22

yeah, i understand that this was my fault :/ I will try that and get back in a little while thanx – Spyros Apr 1 '11 at 18:56

1 tried forcing, but when going back to master server to save the changes, i get outdated staging. Thus, when i commit the repositories are not the same. And when i try to use git push again, i get the same error. – Spyros Apr 2 '11 at 18:11

17

```
git push --force origin B:C
```

answered May 28 '15 at 18:25
IcedDante
4,426 8 43 76

2 I found out that even I'm on my local branch B, I still need to do `git push --force origin B:C` . In my case, it seems that `git push --force origin C` will only push from local master to remote C branch, regardless of which branch I'm currently on. `git version 2.3.8 (Apple Git-58)` – Weishi Zeng Oct 10 '15 at 5:28

12

use this following command:

```
git push -f origin master
```

answered Apr 10 '17 at 14:00
mustafa Elsayed
131 1 2

2 Maybe give some more explanation about why this answer is preferable to the other ones, and what makes it different. – Adam Apr 10 '17 at 15:41

oh ,sorry for inconvenience , I was having the same problem and this command solve it , i thought i should share it . – mustafa Elsayed Apr 11 '17 at 14:30

12 It's just the same as the others, you just changed the position of the `-f` flag... – svelandiag May 31 '17 at 15:38

11

I would really recommend to:

- push only to the main repo
- make sure that main repo is a bare repo, in order to never have any problem with the main repo working tree being not in sync with its `.git` base. See "How to push a local git repository to another computer?"
- If you do have to make modification in the main (bare) repo, clone it (on the main server), do your modification and push back to it

In other words, keep a bare repo accessible both from the main server and the local computer, in order to have a single upstream repo from/to which to pull/pull.

edited May 23 '17 at 12:02
Community ♦
1 1

answered Apr 1 '11 at 5:53
VonC
967k 392 3314 3883

This was our solution for replacing master on a corporate gitHub repository while maintaining `push -f` to master on corporate repositories is often disabled to maintain branch history. This solution worked for us.

```
git fetch desiredOrigin
git checkout -b master desiredOrigin/master // get origin master



git checkout currentBranch  // move to target branch
git merge -s ours master  // merge using ours over master
// vim will open for the commit message
git checkout master  // move to master
git merge currentBranch  // merge resolved changes into master
```

push your branch to `desiredOrigin` and create a PR

answered May 26 '16 at 21:31
mihai
3,780 3 21 26

3

I had the same question but figured it out finally. What you most likely need to do is run the following two git commands (replacing hash with the git commit revision number):

```
git checkout <hash>
git push -f HEAD:master
```

answered Apr 23 '19 at 18:02
Brian M.
59 2