Christoph Rüegg

Math.NET, distributed computing and how an electrical engineer sees the world of complex software

Git Howto: Revert a Commit Already Pushed to a Remote Repository

Wednesday, May 05, 2010

So you've just pushed your local branch to a remote branch, but then realized that one of the commits should not be there, or that there was some unacceptable typo in it. No problem, you can fix it. But you should do it rather fast *before anyone fetches the bad commits*, or you won't be very popular with them for a while ;)

First two alternatives that will keep the history intact:

Alternative: Correct the mistake in a new commit

Simply remove or fix the bad file in a new commit and push it to the remote repository. This is the most natural way to fix an error, always safe and totally non-destructive, and how you should do it 99% of the time. The bad commit remains there and accessible, but this is usually not a big deal, unless the file contains sensitive information.

Alternative: Revert the full commit

Sometimes you may want to undo a whole commit with all changes. Instead of going through all the changes manually, you can simply tell git to revert a commit, which does not even have to be the last one. Reverting a commit means to create a new commit that undoes all changes that were made in the bad commit. Just like above, the bad commit remains there, but it no longer affects the the current master and any future commits on top of it.

```
1 $ git revert dd61ab32
```

About History Rewriting

People generally avoid history rewiriting, for a good reason: it will fundamentally diverge your repository from anyone who cloned or forked it. People cannot just *pull* your rewritten history as usual. If they have local changes, they have to do some work to get in sync again; work which requires a bit more knowledge on how Git works to do it properly.

However, sometimes you *do* want to rewrite the history. Be it because of leaked sensitive information, to get rid of some very large files that should not have been there in the first place, or just because you want a clean history (I certainly do).

I usually also do a lot of very heavy history rewriting when converting some repository from Subversion or Mercurial over to Git, be it to enforce internal LF line endings, fixing committer names and email addresses or to completely delete some large folders from all revisions. I recently also had to rewrite a large git repository to get rid of some corruption in an early commit that started causing more and more problems.

Yes, you should avoid rewriting history which already passed into other forks if possible, but the world does not end if you do nevertheless. For example you can still cherry-pick commits between the histories, e.g. to fetch some pull requests on top of the old history.

In opensource projects, always contact the repository maintainer first before doing any history rewriting. There are maintainers that do not allow any rewriting in general and block any non-fastforward pushes. Others prefer doing such rewritings themselves.

Case 1: Delete the last commit

Deleting the last commit is the easiest case. Let's say we have a remote *mathnet* with branch *master* that currently points to commit *dd61ab32*. We want to remove the top commit. Translated to git terminology, we want to force the *master* branch of the *mathnet* remote repository to the parent of *dd61ab32*:

```
1 $ git push mathnet +dd61ab32^:master
```

Where git interprets `x^` as the parent of x and `+` as a forced non-fastforward push. If you have the master branch checked out locally, you can also do it in two simpler steps: First reset the branch to the parent of the current commit, then force-push it to the remote.

```
1 $ git reset HEAD^ --hard
2 $ git push mathnet -f
```

Case 2: Delete the second last commit

Let's say the bad commit *dd61ab32* is not the top commit, but a slightly older one, e.g. the second last one. We want to remove it, but keep all commits that followed it. In other words, we want to rewrite the history and force the result back to *mathnet/master*. The easiest way to rewrite history is to do an interactive rebase down to the parent of the offending commit:

```
1 $ git rebase -i dd61ab32^
```

This will open an editor and show a list of all commits since the commit we want to get rid of:

```
pick dd61ab32
pick dsadhj278
...
```

Simply remove the line with the offending commit, likely that will be the first line (vi: delete current line = `dd`). Save and close the editor (vi: press `:wq` and return). Resolve any conflicts if there are any, and your local branch should be fixed. Force it to the remote and you're done:

```
1 $ git push mathnet -f
```

Case 3: Fix a typo in one of the commits

This works almost exactly the same way as case 2, but instead of removing the line with the bad commit, simply replace its `pick` with `edit` and save/exit. Rebase will then stop at that commit, put the changes into the index and then let you change it as you like. Commit the change and continue the rebase (git will tell you how to keep the commit message and author if you want). Then push the changes as described above. The same way you can even split commits into smaller ones, or merge commits together.

Posted by Christoph Rüegg Wednesday, May 05, 2010 Git

« Lost in Math.NET Codenames? How to create 2048bit Certificate CSRs for Dell's iDRAC6 »

Comments

mltsy · 4 years ago

I would recommend looking into `git revert` before using any of these strategies on a remote repo. Rebasing and force pushing on a shared repo can cause issues for people downwind of your changes. Revert was designed specifically for this purpose! :) (I'm just researching it myself, not having used it before)

26 · Reply · Share ›

Christoph Rüegg Mod · 3 years ago

Thanks for your feedback. I've extended the post to explain the issue a bit, and now fist mention the non-destructive approach with `git revert`. I hope that prevents some pain caused by unnecessary history rewrites.

12 · Reply · Share ›

Citizen Forker · 4 years ago

@mltsy is 100% right, I've just experienced the horror of the leaked malicious commit being pushed again into remote repo.

5 · Reply · Share ›

lorcha · 2 years ago

So in the case of pushing rewritten history to a project repository, and it breaks everybody's clones and working copies and local changes and whatnot, how do those users with the broken local repositories fix the damage?

2 · Reply · Share ›

Christoph Rüegg Mod ↗ lorcha · 2 years ago

Once you fetch the new tree you'll end up with two trees in parallel: the old one and the new rewritten one. You need to get rid of the old one entirely in your local repository. So you'll want to migrate all your local work over to sit on top of the new tree. The easiest ways to do this is cherry-picking or rebasing (rebasing might require some conflict resolution though). Just be sure *not* to merge between the two trees, ever! Once everything is over there, drop all local remaining refs (e.g. branches) to the old tree and gc/prune your repository it to get rid of it entirely.

1 · Reply · Share ›

kilianc ↗ lorcha · a year ago

Just pull-rebase instead of pull-merge. In general is a good practice, even if your parent commit changes, your unpushed commits will be applied on top of the new history.

· Reply · Share ›

snissen · 2 years ago

Thank you -- alternatives were especially helpful.

1 · Reply · Share ›

Tim Heider · 2 years ago

Thank you!!!

1 · Reply · Share ›

mig · 2 years ago

thank you :)

1 · Reply · Share ›

ilyavf · 3 years ago

Thank you for a great article! Can't wait to try all the options!

1 · Reply · Share ›

Thomas · 3 years ago

I could really kiss you right now. It worked like a charm; of course, I didn't doubt it. Thanks a lot. @mltsy definitely looking into that one as well :>

1 · Reply · Share ›

Nate · 3 years ago

I deleted files in "feature" and merged that into "master", then restored them in "master" via a remote hotfix branch because others were pulling. Meanwhile, 'feature' kept going with development. I want to know if merging 'feature' into master (which has the newly restored deleted files from a previous version of 'feature') will have the files deleted once more? Is there a way to make sure those files get deleted again? Or restore the same deleted files in 'feature', but not overwrite and still mark for deletion?

1 · Reply · Share ›

Christoph Rüegg Mod → Nate · 3 years ago

Since the first merge, master and feature have a common ancestor that is newer than the commit where the files had been deleted in feature, so they will not be deleted again in master when merging feature into master again. What could work is to `git cherry-pick` the hotfix (where the files were restored in master) into feature (so they are restored there again as well), and then revert that just picked commit again with `git revert` and hence removing them again in a second commit. Best to test it in a small repo first to be sure though.

1 · Reply · Share ›

Parcival Willems · 8 days ago

Be aware that ```git push mathnet -f``` does execute a forced push of all your local branches

· Reply · Share ›

Christoph Rüegg Mod → Parcival Willems · 8 days ago

That depends on your `push.default` config. By default this is set to simple (since Git 2.0) which only pushes the current/HEAD branch (and only if it is named the same on the remote). But yes, it would be safer to specify the branch explicitly.

· Reply · Share ›

Eagle · 18 days ago

lel, thank you, this article just saved my ass

· Reply · Share ›

Antony · a year ago

sweet post. thanks a bunch on the "git revert"

· Reply · Share ›

Allie · a year ago

thank you for this, I finally know how to do an interactive rebase! very clear :)

· Reply · Share ›

Eugeny Kozhanov · 2 years ago

Thank you very much for push -f !

 · Reply · Share ›

hv · 2 years ago

Life saver!!

 · Reply · Share ›

Eugene Fidelin · 2 years ago

Very useful advices - they saved me a lot of time :)

 · Reply · Share ›

Recent Posts

- Loading native DLLs in F# Interactive
- Towards Math.NET Numerics Version 3
- What's New in Math.NET Numerics 2.6
- Test your C# or F# Library on Mono with Vagrant
- What's New in Math.NET Numerics 2.5

Open Source



Math.NET

- Math.NET Numerics
  Linear Algebra, FFT, Distributions, ...
- Math.NET Iridium
  Predecessor of Math.NET Numerics
- Math.NET LinqAlgebra
  Symbolics with Linq Expressions

Math.NET Project Website
@mathnet on Github

Lokad.Cloud

- Lokad.Cloud Storage
  Robust Azure Storage Toolkit
- Lokad.Cloud Provisioning
  Auto-scale Azure Workers
- Lokad.Cloud AppHost
  Git-versioned Dynamic Deployments
- Lokad.Cloud AppHost Extensions
- Lokad.Cloud Service Framework
  Numbercrunching in Azure

Lokad.Cloud Project Website
@Lokad on Github