Undoing a git rebase

Asked 14 years, 2 months ago Modified 3 months ago Viewed 1.7m times

3998

How do I easily undo a git rebase? A lengthy manual method is:

1. checkout the commit parent to both of the branches
2. create and checkout a temporary branch
3. cherry-pick all commits by hand
4. reset the faulty rebased branch to point to the temporary branch

In my current situation, this works because I can easily spot commits from both branches (one was my stuff, the other was my colleague's stuff). However, my approach strikes me as suboptimal and error-prone (let's say I had just rebased with two of my own branches).

Clarification: I am talking about a rebase during which *multiple* commits were replayed, not only one.

git rebase git-rebase undo

Share Edit Follow

edited Aug 7 at 20:15
Mateen Ulhaq
22.7k 16 88 129

asked Sep 25, 2008 at 17:59
webmat
56.2k 12 54 59

19 Answers

Sorted by: Highest score (default)

5514

The easiest way would be to find the head commit of the branch as it was immediately before the rebase started in the reflog...

```
git reflog
```

and to reset the current branch to it (with the usual caveats about being absolutely sure before reseting with the `--hard` option).

Suppose the old commit was `HEAD@{2}` in the ref log:

```
git reset --hard HEAD@{2}
```

In Windows, you may need to quote the reference:

```
git reset --hard "HEAD@{2}"
```

You can check the history of the candidate old head by just doing a `git log HEAD@{2}` (*Windows:* `git log "HEAD@{2}"`).

If you've not disabled per branch reflogs you should be able to simply do `git reflog branchname@{1}` as a rebase detaches the branch head before reattaching to the final head. I would double check this, though as I haven't verified this recently.

Per default, all reflogs are activated for non-bare repositories:

```
[core]
    logAllRefUpdates = true
```

Share Edit Follow

edited Sep 22, 2021 at 7:07
Jonathan
8,265 8 52 71

answered Sep 25, 2008 at 19:56
CB Bailey
724k 101 623 650

183 Git reflog is awesome, just remember you can get better formatted output with `git log -g` (tip from Scott Chacon's progit.org/book). – karmi Jul 23, 2010 at 10:14

2 Per Allan's answer below a `git rebase -i --abort` is needed as well. The above alone is not enough. – Hazok Jun 14, 2011 at 22:36

81 @Zach: `git rebase --abort` (`-i` makes no sense with `--abort`) is for abandoning a rebase that hasn't been completed - either because there were conflicts or because it was interactive or both; it's not about undoing a successful rebase which is what the question is about. You would either use `rebase --abort` or `reset --hard` depending on which situation you were in. You shouldn't need to do both. – CB Bailey Jun 15, 2011 at 20:40

411 Just in case, make a backup first: `git tag BACKUP` . You can return to it if something goes wrong: `git reset --hard BACKUP` – kolypto Nov 5, 2012 at 14:00

28 If you've made a lot of commits the HEAD@{#} you're looking for will be prefaced with `commit:` as opposed to `rebase:` . Sounds obvious but it confused me for a bit. – Warpling Aug 26, 2013 at 20:43

1866

Actually, rebase saves your starting point to `ORIG_HEAD` so this is usually as simple as:

```
git reset --hard ORIG_HEAD
```

However, the `reset` , `rebase` and `merge` all save your original `HEAD` pointer into `ORIG_HEAD` so, if you've done any of those commands since the rebase you're trying to undo then you'll have to use the reflog.

Share Edit Follow

answered Mar 28, 2009 at 13:24
Pat Notz
203k 30 89 92

47 In case `ORIG_HEAD` is no longer useful, you can also use the `branchName@{n}` syntax, where `n` is the nth prior position of the branch pointer. So for example, if you rebase `featureA` branch onto your `master` branch, but you don't like the result of the rebase, then you can simply do `git reset --hard featureA@{1}` to reset the branch

back to exactly where it was before you did the rebase. You can read more about the branch@{n} syntax at the official Git docs for revisions. – user456814 May 24, 2013 at 5:17

19 This is the easiest. Follow it up with a `git rebase --abort` though. – Seph Sep 16, 2015 at 19:24

12 @Seph Can you explain why you suggest following up with `git rebase --abort` ? – UpTheCreek Jan 28, 2019 at 8:50

@Seph I agree with UpTheCreek as in my case it was not necessary. Maybe it is needed when things get odd during an interactive rebase? I would suggest anyone to try a `git status` to see if it mentions about the rebase or not first. Anyway doing a `git rebase --abort` should be harmless. If there is no ongoing rebase it will just fail complaining " `fatal: No rebase in progress?` ". – Jacopo Tedeschi Aug 20, 2021 at 14:44

2 Is it possible to check what the commit of ORIG_HEAD is first? – Simon May 9 at 12:13

490

Charles's answer works, but you may want to do this:

```
git rebase --abort
```

to clean up after the `reset` .

Otherwise, you may get the message “ `Interactive rebase already started` ”.

Share Edit Follow

edited Jan 20, 2021 at 12:58
Borislav Ivanov
4,306 2 28 51

answered Jul 27, 2009 at 18:21
Allan
5,278 1 15 3

155 That was not the question. The question asks how to undo a finished rebase. – Arunav Sanyal Apr 4, 2017 at 18:29

1 While it may not be the answer, this info is essential and the answer doesn't have it. After using `reflog` and `reset --hard` , you still may need to abort the rebase to get back to good. – mattmc3 Aug 15 at 16:41

109

Resetting the branch to the dangling commit object of its old tip is of course the best solution, because it restores the previous state without expending any effort. But if you happen to have lost those commits (f.ex. because you garbage-collected your repository in the meantime, or this is a fresh clone), you can always rebase the branch again. The key to this is the `--onto` switch.

Let’s say you had a topic branch imaginatively called `topic` , that you branched off `master` when the tip of `master` was the `0deadbeef` commit. At some point while on the `topic` branch, you did `git rebase master` . Now you want to undo this. Here’s how:

```
git rebase --onto 0deadbeef master topic
```

This will take all commits on `topic` that aren’t on `master` and replay them on top of `0deadbeef` .

With `--onto` , you can rearrange your history into pretty much *any shape whatsoever*.

Have fun. :-)

Share Edit Follow

edited Sep 26, 2008 at 2:47

answered Sep 26, 2008 at 2:08

Aristotle Pagaltzis
110k 22 97 96

3 I think this is the best option because of its flexibility. I branched b1 off master, then rebased b1 into a new branch b2, then wanted to revert b1 to be based on master again. I just love git - thanks! – ripper234 Aug 17, 2011 at 12:59

2 This is the best option here! It kept all changes I have on my current branch, and removed all the unwanted ones! – Alicia Tang Mar 15, 2016 at 20:43

for some reason your answer made me realize I can do a git rebase -i commitish and then EDIT the commit I wasn't satisfied with :) – Devin Rhode Oct 18, 2020 at 4:10

99

In case **you had pushed your branch to remote repository** (usually it's origin) and then you've done a succesfull rebase (without merge) (`git rebase --abort` gives "No rebase in progress") you can easily **reset branch** using command:

> git reset --hard origin/{branchName}

Example:

```
$ ~/work/projects/{ProjectName} $ git status
On branch {branchName}
Your branch is ahead of 'origin/{branchName}' by 135 commits.
  (use "git push" to publish your local commits)

nothing to commit, working directory clean

$ ~/work/projects/{ProjectName} $ git reset --hard origin/{branchName}
HEAD is now at 6df5719 "Commit message".

$ ~/work/projects/{ProjectName} $ git status
On branch {branchName}
Your branch is up-to-date with 'origin/{branchName}.

nothing to commit, working directory clean
```

Share Edit Follow

edited Feb 5, 2016 at 10:12

Nick Roz
3,649 2 35 54

answered Sep 28, 2015 at 12:43

Maksym
4,226 3 25 45

77

```
git reset --hard origin/{branchName}
```

is the correct solution to reset all your local changes done by rebase.

Share Edit Follow

edited Dec 7, 2020 at 10:23

Tiago Martins Peres
13.3k 17 83 124

answered May 6, 2019 at 8:04

D

Damodar P
1,025 6 3

4 This is a very clean solution. Thank you for posting. I always push to remote before doing a rebase and this post saved my bacon, as I was ~3 days out of date w/ master on a big project. Thank you :) – Adrian Aug 3, 2021 at 2:23

6 This can cause problems if there are local commits that weren't pushed. Be warned – Onewildgamer Feb 14 at 4:41

75

I actually put a backup tag on the branch before I do any nontrivial operation (most rebases are trivial, but I'd do that if it looks anywhere complex).

Then, restoring is as easy as `git reset --hard BACKUP`.

Share Edit Follow

edited May 24, 2013 at 5:08
user456814

answered May 12, 2009 at 20:57
Alex Gontmakher
1,337 11 10

14 You actually don't even need to make a backup branch, you can simply use the `branchName@{n}` syntax, here `n` is the nth prior position of the branch pointer. So for example, if you rebase `featureA` branch onto your `master` branch, but you don't like the result of the rebase, then you can simply do `git reset --hard featureA@{1}` to reset the branch back to exactly where it was before you did the rebase. You can read more about the `branch@{n}` syntax at the official Git docs for revisions. – user456814 May 24, 2013 at 5:12

32

Using `reflog` didn't work for me.

What worked for me was similar to as described here. Open the file in .git/logs/refs named after the branch that was rebased and find the line that contains "rebase finsihed", something like:

```
5fce6b51 88552c8f Kris Leech <me@example.com> 1329744625 +0000  rebase finished:
refs/heads/integrate onto 9e460878
```

Checkout the second commit listed on the line.

```
git checkout 88552c8f
```

Once confirmed this contained my lost changes I branched and let out a sigh of relief.

```
git log
git checkout -b lost_changes
```

Share Edit Follow

answered Feb 20, 2012 at 13:59
Kris
18.7k 8 89 108

2 Thanks man, you saved my life. For me I had `rebase (finish):` followed by 3 commit IDs. The second ID was the correct one. – PatPatchPatrick Jan 4 at 22:50

Mine listed 2 commit id's, and said `rebase (finish)`. Since I wanted the commit BEFORE I rebased, I used the *2nd* commit id on the line PRIOR to the rebase (=== the **1st** commit id on the line WITH the rebase command). Then I used the command `git reset --hard <commit id>` . Viola! Restored as if I never performed the rebase. The commit IDs seem to be in the format of *idBeforeCommand idAfterCommand Who Date/time Command* – SherylHohman Apr 18 at 17:33

16

For multiple commits, remember that any commit references all the history leading up to that commit. So in Charles' answer, read "the old commit" as "the newest of the old commits". If you reset to that commit, then all the history leading up to that commit will reappear. This should do what you want.

Share Edit Follow

answered Sep 25, 2008 at 21:36
Greg Hewgill
919k 178 1135 1270

14

If you successfully rebased against a remote branch and can not `git rebase --abort` you still can do some tricks to save your work and don't have forced pushes. Suppose your current branch that was rebased by mistake is called `your-branch` and is tracking `origin/your-branch`

- `git branch -m your-branch-rebased` # rename current branch
- `git checkout origin/your-branch` # checkout to latest state that is known to the origin
- `git checkout -b your-branch`
- check `git log your-branch-rebased` , compare to `git log your-branch` , and define commits that are missing from `your-branch`
- `git cherry-pick COMMIT_HASH` for every commit in `your-branch-rebased`
- push your changes. Please be aware that two local branches are associated with `remote/your-branch` and you should push only `your-branch`

Share Edit Follow

edited Aug 22, 2021 at 20:45
Vlad L.
154 1 9

answered Jun 23, 2016 at 9:23
Sergey P. aka azure
3,628 1 28 23

13

If you don't want to do a hard reset...

You can checkout the commit from the reflog, and then save it as a new branch:

```
git reflog
```

Find the commit just before you started rebasing. You may need to scroll further down to find it (press Enter or PageDown). Take note of the HEAD number and replace 57:

```
git checkout HEAD@{57}
```

Review the branch/commits, and if it's correct then create a new branch using this HEAD:

```
git checkout -b new_branch_name
```

Share Edit Follow

edited Sep 17, 2020 at 13:46

answered May 29, 2019 at 17:56

Andrew
17.4k 11 98 111

11

Following the solution of @Allan and @Zearin, I wish I could simply do a comment though but I don't enough reputation, so I have used the following command:

Instead of doing `git rebase -i --abort` (note the **-i**) I had to simply do `git rebase --abort` (**without** the **-i**).

Using both `-i` and `--abort` at the same time causes Git to show me a list of usage/options.

So my previous and current branch status with this solution is:

```
matbhz@myPc /my/project/environment (branch-123|REBASE-i)
$ git rebase --abort

matbhz@myPc /my/project/environment (branch-123)
$
```

Share Edit Follow

answered Mar 11, 2015 at 21:26

Matheus Felipe
2,352 23 24

7

Another way **that doesn't require doing a hard reset** is to create a new branch with your desired starting point.

As with the other solutions, you use the reflog to find the correct starting point.

```
git reflog
```

(you can also use `git log -g` here for more detail)

Then you note the reference to the commit SHA (ex: `e86a52b851e`).

Finally, you use the git branch command.

```
git branch recover-branch e86a52b851e
```

Reference: https://git-scm.com/book/en/v2/Git-Internals-Maintenance-and-Data-Recovery#_data_recovery

Share Edit Follow

answered Jul 1, 2021 at 22:47
GuyStalks
131 1 4

4

If you are on a branch you can use:

```
git reset --hard @{1}
```

There is not only a reference log for HEAD (obtained by `git reflog`), there are also reflogs for each branch (obtained by `git reflog <branch>`). So, if you are on `master` then `git reflog master` will list all changes to that branch. You can refer to that changes by `master@{1}`, `master@{2}`, etc.

`git rebase` will usually change HEAD multiple times but the current branch will be updated only once.

`@{1}` is simply a shortcut for the current branch, so it's equal to `master@{1}` if you are on `master`.

`git reset --hard ORIG_HEAD` will not work if you used `git reset` during an interactive `rebase`.

Share Edit Follow

answered Jan 10, 2019 at 12:21
devconsole
7,746 1 32 42

4

Let's say I rebase master to my feature branch and I get 30 new commits which break something. I've found that often it's easiest to just remove the bad commits.

```
git rebase -i HEAD~31
```

Interactive rebase for the last 31 commits (it doesn't hurt if you pick way too many).

Simply take the commits that you want to get rid of and mark them with "d" instead of "pick". Now the commits are deleted effectively undoing the rebase (if you remove only the commits you just got when rebasing).

Share Edit Follow

answered Feb 9, 2017 at 12:25
Hardev
10.1k 2 15 17

3

It annoys me to no end that none of these answers is fully automatic, despite the fact that it should be automatable (at least mostly). I created a set of aliases to try to remedy this:

```
# Useful commands
#################

# Undo the last rebase
undo-rebase = "! f() { : git reset ; PREV_COMMIT=`git x-rev-before-rebase` && git reset --merge
\"$PREV_COMMIT\" \"$@\";}; f"

# See what changed since the last rebase
rdiff = "!f() { : git diff ; git diff `git x-rev-before-rebase` "$@";}; f"

# Helpers
########
# Get the revision before the last rebase started
x-rev-before-rebase = !git reflog --skip=1 -1 \"`git x-start-of-rebase`\" --format=\"%gD\"

# Get the revision that started the rebase
x-start-of-rebase = reflog --grep-reflog '^rebase (start)' -1 --format="%gD"
```

You should be able to tweak this to allow going back an arbitrary number of rebases pretty easily (juggling the args is the trickiest part), which can be useful if you do a number of rebases in quick succession and mess something up along the way.

Caveats

It will get confused if any commit messages begin with "rebase (start)" (please don't do this). You could make the regex more resilient to improve the situation by matching something like this for your regex:

```
--grep-reflog "^rebase (start): checkout "
```

WARNING: not tested (regex may need adjustments)

The reason I haven't done this is because I'm not 100% that a rebase *always* begins with a checkout. Can anyone confirm this?

[If you're curious about the null (`:`) commands at the beginning of the function, that's a way of setting up bash completions for the aliases]

Share Edit Follow

edited Dec 7, 2020 at 10:23
Tiago Martins Peres
13.3k 17 83 124

answered Jul 19, 2020 at 22:47
DylanYoung
2,203 24 28

3

What I usually do is `git reset #commit_hash`

to the last commit where I think rebase had no effect.

then `git pull`

Now your branch should match exactly like master and rebased commits should not be in it.

Now one can just cherry-pick the commits on this branch.

Share Edit Follow

answered Jun 19, 2019 at 14:39
mrigendra
1,370 3 16 33

1

I tried all suggestions with reset and reflog without any success. Restoring local history of IntelliJ resolved the problem of lost files

Share Edit Follow

answered Feb 16, 2020 at 19:06
user3638751
73 3

-5

If you mess something up within a git rebase, e.g. `git rebase --abort`, while you have uncommitted files, they will be lost and `git reflog` will not help. This happened to me and you will need to think outside the box here. If you are lucky like me and use IntelliJ Webstorm then you can `right-click->local history` and can revert to a previous state of your file/folders no matter what mistakes you have done with versioning software. It is always good to have another failsafe running.

Share Edit Follow

edited Jul 18, 2018 at 12:25
Ed Avis
1,310 16 36

answered Mar 9, 2016 at 9:19
stevek-pro
14.4k 14 84 140

6 `git rebase --abort` aborts an active rebase, it doesn't *undo* a rebase. Also, using two VCS's at the same time is a bad idea. Its a nice feature in Jetbrains software but you shouldn't use both. It's better to just learn Git, particularly when answering questions on Stack Overflow that are about Git. – dudewad May 15, 2017 at 17:56