Revert to a commit by a SHA hash in Git? [duplicate]

Asked 13 years, 2 months ago Modified 1 year, 2 months ago Viewed 672k times

725

This question already has answers here:
How do I revert a Git repository to a previous commit? (41 answers)
Closed 8 years ago.

I'm not clear on how `git revert` works. For example, I want to revert to a commit six commits behind the head, reverting all the changes in the intermediary commits in between.

Say its SHA hash is `56e05fced214c44a37759efa2dfc25a65d8ae98d`. Then why can't I just do something like:

```
git revert 56e05fced214c44a37759efa2dfc25a65d8ae98d
```

git

Share Edit Follow Flag

edited Jan 26, 2018 at 15:54

asked Dec 12, 2009 at 23:34
JP Silvashy
46.2k ● 48 ● 150 ● 222

1 Even though this question is actually older than the one it's now marked as a duplicate of, that one has a better answer. meta.stackexchange.com/questions/147643/... – GS - Apologise to Monica Jun 28, 2014 at 19:39

19 This question and the top answer here may confuse git users. Just to help understand the terminology, you don't *revert to* a commit. You can either *reset to* a commit (which is like going back in time using time machine) or *revert* a commit (which is like pulling out a commit as if it never existed - however it does preserve the revert info in history, allowing you to revert a revert if you wanted to) Note also that you shouldn't use the m flag and type a commit message if you get conflicts in the process. The auto message git provides is more informative when looking back in history. – alexrogers Mar 12, 2015 at 12:30

1 @alexrogins what does pulling out a commit as if it never existed mean? Not sure what 'revert a revert' refers to either - appreciate the comment though, good info, just looking for more detail on your perspective. – Joe Jan 23, 2018 at 17:15

2 @Joe as in if you add a line of code then commit that line, if you were to revert it you would be undoing that line of code (wherever it was first written in history, doesn't have to be the last commit). That then makes a revert commit. If you revert that revert commit then you're essentially undoing the undo (i.e. redoing the original line again) – alexrogers Jan 24, 2018 at 21:59

9 Answers

Sorted by: Highest score (default)

1320

If you want to commit on top of the current HEAD with the exact state at a different commit, undoing all the intermediate commits, then you can use `reset` to create the correct state of the index to make the commit.

```
# Reset the index and working tree to the desired tree
# Ensure you have no uncommitted changes that you want to keep
git reset --hard 56e05fced

# Move the branch pointer back to the previous HEAD
git reset --soft "HEAD@{1}"

git commit -m "Revert to 56e05fced"
```

Share Edit Follow Flag

edited Apr 21, 2021 at 16:06
metal
6,087 ● 1 ● 36 ● 48

answered Dec 12, 2009 at 23:51
CB Bailey
734k ● 101 ● 626 ● 651

90 Wouldn't it be equivalent (and one command shorter) to do: `git reset --hard 56e05fced` as the first command, and then skip the final `git reset --hard` ? – Mark Longair Mar 2, 2012 at 8:20

27 When I did this I ended up with a bunch of `Untracked Files` in the working tree. However looking at the history I could see that those files did have a corresponding delete commit in that "Revert to SHA" commit. So after `git reset --hard` at the end, you can do `git clean -f -d` to clean up any untracked files that lingered about. Also, thank you so much this helped me solve a crisis! – nzifnab Apr 27, 2012 at 19:33

5 do I have to do the `git reset --soft HEAD@{1}` unconditionally? I mean always with a value of 1? – deprecated Sep 17, 2013 at 9:23

7 @vemv Yes, unless you want to throw away commits on the tip of the branch. `git reset 56e05fced` adds another entry to the reflog (run `git reflog`), so `git reset --soft HEAD@{1}` simply moves the pointer back to the `HEAD` prior to calling `git reset 56e05fced` . Using a higher number (e.g. `git reset --soft HEAD@{2}`) would append the new commit on a *previous* commit. That is, increasing the number would essentially throw away `N-1` commits where `N` is the number you replace `1` with. – 0b10011 Sep 18, 2013 at 18:37

5 @Tom `HEAD@{1}` should be quoted as `'HEAD{@1}'` otherwise, it won't work for me (possibly every zsh users) – jilen Apr 1, 2015 at 1:56

178

What git-revert does is create a commit which undoes changes made in a given commit, creating a commit which is reverse (well, reciprocal) of a given commit. Therefore

```
git revert <SHA-1>
```

should and does work.

If you want to rewind back to a specified commit, and you can do this because this part of history was not yet published, you need to use git-reset, not git-revert:

```
git reset --hard <SHA-1>
```

(Note that `--hard` would make you lose any non-committed changes in the working directory).

Additional Notes

By the way, perhaps it is not obvious, but everywhere where documentation says `<commit>` or `<commit-ish>` (or `<object>`), you can put an SHA-1 identifier (full or shortened) of commit.

Share Edit Follow Flag

edited Jul 20, 2014 at 9:31
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Dec 13, 2009 at 9:54
Jakub Narębski
302k ● 65 ● 219 ● 230

9 **In the case that you're history has already been pushed to a remote before you did the hard reset**, you would need to force push the newly reset branch with `git push -f`, but **Be Warned** that this could possibly unintentionally delete other users' commits, and if not delete new commits, then it will force other users to resynchronize their work with the reset branch, **so make sure this is OK with your collaborators first.** – user456814 Jun 28, 2014 at 17:25

4 This seems to be the best answer. It also tells clearly the difference between git revert and git reset. – kta May 14, 2018 at 3:14

91

It reverts the said commit, that is, adds the commit opposite to it. If you want to checkout an earlier revision, you do:

```
git checkout 56e05fced214c44a37759efa2dfc25a65d8ae98d
```

Share Edit Follow Flag

edited Jul 20, 2014 at 9:25
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Dec 12, 2009 at 23:40
Michael Krelin - hacker
136k ● 24 ● 192 ● 173

1 then I can just merge this with the head? What if I anticipate having TONS of conflicts, can I just force this commit to be the head "as-is" and just overwrite any conflicts? – JP Silvashy Dec 12, 2009 at 23:43

1 I'm not sure what head you're talking about. You can just *move* your head back to this commit. (for instance by deleting and creating branch). If you want to do a "merge" commit into the head, which is effectively the reversal of the intermediate commits, you can use merge with "ours" strategy. Pick your option and read manpages. The power is waiting for you to use it ;-) – Michael Krelin - hacker Dec 12, 2009 at 23:48

That makes sense, the reason I ask is that git now tells me that I'm not on any branch. – JP Silvashy Dec 12, 2009 at 23:51

9 because you aren't. if you type `git branch` you will clearly see it. You can do for instance `git checkout -b mybranch 56e05` to get it with branch. – Michael Krelin - hacker Dec 13, 2009 at 0:06

86

The best way to rollback to a specific commit is:

```
git reset --hard <commit-id>
```

Then:

```
git push <reponame> -f
```

Share Edit Follow Flag

edited Jul 20, 2014 at 9:26
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Nov 19, 2012 at 13:50
darshit khatri
901 ● 6 ● 2

43 Novices should be aware that `push -f` can destroy history. However, sometimes this is what you want :) – Jared Beck Feb 26, 2013 at 0:30

1 Sometimes you are really glad that the history is deleted...was looking for this -f option, thks ! – Antoine Mar 26, 2013 at 4:54

Thanks, to be literal, I had to type in -> git push origin master -f where <reponame> can't just be origin at least for me – SWoo Aug 15, 2013 at 3:10

As people have mentioned above, If we want our repo head pointing to a specific commit without maintaining history then use above steps other wise we can use git revert. – minhas23 May 26, 2014 at 7:48

Is there anyway that we know who had reset (i.e rollback commit) and forced push on a specific branch? – datnt Mar 27, 2015 at 2:21

75

If your changes have already been pushed to a **public, shared** remote, and you want to revert all commits between `HEAD` and `<sha-id>`, then you can pass a commit range to `git revert`,

```
git revert 56e05f..HEAD
```

and it will revert all commits between `56e05f` and `HEAD` (excluding the start point of the range, `56e05f`).

Share Edit Follow Flag

edited Jun 28, 2014 at 19:44
user456814

answered Dec 19, 2011 at 13:15
Flueras Bogdan
9,047 ●8 ●31 ●30

2 Note that if you're reverting a few hundred commits, this could take a while because you have to commit each revert individually. – splicer Jun 27, 2013 at 6:14

9 @splicer you don't have to revert each commit individually, you can either pass the `--no-edit` option to avoid having to make individual commit messages, or you can use `--no-commit` to commit the reversions all at once. – user456814 Jun 28, 2014 at 17:40

@Cupcake you are right **HEAD..56e05f** doesn't work for me but **56e05f..HEAD** did the trick – Inder Kumar Rathore Aug 19, 2014 at 19:13

3 This is by far my preferred way of rolling back, no matter if you pushed it or not. I added this to my global `~/.gitconfig` under the aliases section: `rollback = "!git revert --no-commit $1..HEAD #"` - so now I can just intuitively do `$ git rollback a1s2d3` – DannyB Feb 8, 2017 at 20:22

2 This seems super close to what I want, but I had about 30 commits to revert, but about half way through it fails on a merge commit with `error: Commit 6b3d9b3e05a9cd9fc1dbbebdd170bf083de02519 is a merge but no -m option was given. fatal: revert failed` - any suggestions? I tried adding -m but wasn't quite sure how that would work with this – Brad Parks Jan 28, 2020 at 13:03

62

Updated:

If there were no merge commits in between, this answer provides a is simpler method: https://stackoverflow.com/a/21718540/541862

But if there was one or more merge commits, that answer won't work, so stick to this one (that works in all cases).

Original answer:

```
# Create a backup of master branch
git branch backup_master

# Point master to '56e05fce' and
# make working directory the same with '56e05fce'
git reset --hard 56e05fce

# Point master back to 'backup_master' and
# leave working directory the same with '56e05fce'.
git reset --soft backup_master

# Now working directory is the same '56e05fce' and
# master points to the original revision. Then we create a commit.
git commit -a -m "Revert to 56e05fce"

# Delete unused branch
git branch -d backup_master
```

The two commands `git reset --hard` and `git reset --soft` are magic here. The first one changes the working directory, but it also changes head (the current branch) too. We fix the head by the second one.

Share Edit Follow Flag

edited Dec 2, 2021 at 3:34
Pedro A
3,695 ●3 ●33 ●55

answered Mar 22, 2013 at 5:09
Jacob Dam
2,088 ●17 ●18

2 The -a in your commit isn't necessary. – splicer Jun 27, 2013 at 6:21

4 Perfect. This should become one single command in the git cli, IMO. – Gabriel Queiroz Silva Dec 18, 2014 at 12:24

3 very nice! this is much better than `git revert 56e05fce..HEAD` because it's just one commit – knocte Jun 30, 2015 at 14:03

2 mmm, I take that back, this is simpler: stackoverflow.com/questions/4114095/... – knocte Jun 30, 2015 at 14:09

4 @knocte actually no, the link you gave is not simpler, even though it has thousands of upvotes. The reason is that it simply doesn't work if there is one or more merge commits in the range, which often happens. This one should be indeed the top answer. – Pedro A Dec 2, 2021 at 3:32

5

This is more understandable:

```
git checkout 56e05fced -- .
git add .
git commit -m 'Revert to 56e05fced'
```

And to prove that it worked:

```
git diff 56e05fced
```

Share Edit Follow Flag

answered Jun 23, 2011 at 16:29
Tuyen Tran
107 ● 1 ● 1

6 This isn't correct in general, I'm afraid. The checkout will only (I think) update paths that exist, so if a file has been deleted since `56e05fced`, it won't be staged by doing `git checkout 56e05fced -- .` – Mark Longair Mar 2, 2012 at 8:11

1 **This solution won't delete new files that have been added since** `56e05fced`, like a `git reset --hard` or a `git revert` would. **You really want to use those commands if you actually want to restore the state of** `56e05fced`, not `git checkout`. – user456814 Jun 28, 2014 at 17:22

Note: This will put you in a detached head state. Not advisable! – alexrogers Mar 12, 2015 at 12:41

1 This worked for me, but what does "--" mean in this context? – user755921 Apr 16, 2015 at 16:13

In Bash like commands, double-dashes means the end of command options, so any value added after double-dashes wont be interpreted as an option. In this context the dot (.) wont be interpreted as an option, instead it will act as the <pathspec> of checkout command. – snaphuman Jun 6, 2022 at 14:11

1

Should be as simple as:

```
git reset --hard 56e05f
```

That'll get you back to that specific point in time.

Share Edit Follow Flag

answered Jan 4, 2013 at 0:15
longda
10.1k ● 7 ● 45 ● 66

4 ...and is also very dangerous as will wipe all the history since including other peoples work. **Beware of this one!** – alexrogers Mar 12, 2015 at 12:42

-2

This might work:

```
git checkout 56e05f
echo ref: refs/heads/master > .git/HEAD
git commit
```

Share Edit Follow Flag

answered Jun 25, 2012 at 22:36
Jake
2,036 ● 1 ● 23 ● 22

3 This basically does the same thing as `git reset --hard 56e05f`, except this is less safe and more hacky. You might as well use Charle's solution or Jakub's solution. – user456814 Jun 28, 2014 at 17:14