# How can I reconcile detached HEAD with master/origin?

I'm new at the branching complexities of Git. I always work on a single branch and commit changes and then periodically push to my remote origin.

Somewhere recently, I did a reset of some files to get them out of commit staging, and later did a `rebase -i` to get rid of a couple recent local commits. Now I'm in a state I don't quite understand.

In my working area, `git log` shows exactly what I'd expect-- I'm on the right trail with the commits I didn't want gone, and new ones there, etc.

But I just pushed to the remote repository, and what's there is different-- a couple of the commits I'd killed in the rebase got pushed, and the new ones committed locally aren't there.

I think "master/origin" is detached from HEAD, but I'm not 100% clear on what that means, how to visualize it with the command line tools, and how to fix it.

git

edited Jun 19 at 17:49
John Saunders
**132k** 16 154 303

asked Apr 24 '11 at 17:51
Ben Zotto
**37.9k** 16 101 173

Have you pushed the commits before the rebase? – manojlds Apr 24 '11 at 18:02

@manojlds: Not sure what you mean. I pushed some time before the rebase, but not immediately before. – Ben Zotto Apr 24 '11 at 18:18

As in did you previously push the commits that you removed in the rebase -i.. From your answer I think not. – manojlds Apr 24 '11 at 18:22

@manojlds: Correct. I only killed commits that were more recent than the most recent push. (Although as I mentioned, I have since pushed, since I thought everything was OK) – Ben Zotto Apr 24 '11 at 18:23

Can you explain what you did in `I did a reset of some files to get them out of commit staging` part? sorry for the questions :) – manojlds Apr 24 '11 at 18:31

## 12 Answers

First, let's clarify what HEAD is and what it means when it is detached.

HEAD is the symbolic name for the currently checked out commit. When HEAD is not detached (the "normal"[1] situation: you have a branch checked out), HEAD actually points to a branch's "ref" and the branch points to the commit. HEAD is thus "attached" to a branch. When you make a new commit, the branch that HEAD points to is updated to point to the new commit. HEAD follows automatically since it just points to the branch.

- `git symbolic-ref HEAD` yields `refs/heads/master`
  The branch named "master" is checked out.
- `git rev-parse refs/heads/master` yield `17a02998078923f2d62811326d130de991d1a95a`
  That commit is the current tip or "head" of the master branch.
- `git rev-parse HEAD` also yields `17a02998078923f2d62811326d130de991d1a95a`
  This is what is means to be a "symbolic ref". It points to an object through some other reference.
  (Symbolic refs were originally implemented as symbolic links, but later changed to plain files with extra interpretation so that they could be used on platforms that do not have symlinks.)

We have `HEAD` → `refs/heads/master` → `17a02998078923f2d62811326d130de991d1a95a`

When HEAD is detached, it points directly to a commit—instead of indirectly pointing to one through a branch. You can think of a detached HEAD as being on an unnamed branch.

- `git symbolic-ref HEAD` fails with `fatal: ref HEAD is not a symbolic ref`
- `git rev-parse HEAD` yields `17a02998078923f2d62811326d130de991d1a95a`
  Since it is not a symbolic ref, it must point directly to the commit itself.

We have `HEAD` → `17a02998078923f2d62811326d130de991d1a95a`

The important thing to remember with a detached HEAD is that if the commit it points to is otherwise unreferenced (no other ref can reach it), then it will become “dangling” when you checkout some other commit. Eventually, such dangling commits will be pruned through the garbage collection process (by default, they are kept for at least 2 weeks and may be kept longer by being referenced by HEAD’s reflog).

[1] It is perfectly fine to do “normal” work with a detached HEAD, you just have to keep track of what you are doing to avoid having to fish dropped history out of the reflog.

The intermediate steps of an interactive rebase are done with a detached HEAD (partially to avoid polluting the active branch’s reflog). If you finish the full rebase operation, it will update your original branch with the cumulative result of the rebase operation and reattach HEAD to the original branch. My guess is that you never fully completed the rebase process; this will leave you with a detached HEAD pointing to the commit that was most recently processed by the rebase operation.

To recover from your situation, you should create a branch that points to the commit currently pointed to by your detached HEAD:

```
git branch temp
git checkout temp
```

(these two commands can be abbreviated as `git checkout -b temp` )

This will reattach your HEAD to the new `temp` branch.

Next, you should compare the current commit (and its history) with the normal branch on which you expected to be working:

```
git log --graph --decorate --pretty=oneline --abbrev-commit master origin/master temp
git diff master temp
git diff origin/master temp
```

(You will probably want to experiment with the log options: add `-p` , leave off `--pretty=…` to see the whole log message, etc.)

If your new `temp` branch looks good, you may want to update (e.g.) `master` to point to it:

```
git branch -f master temp
git checkout master
```

(these two commands can be abbreviated as `git checkout -B master temp` )

You can then delete the temporary branch:

```
git branch -d temp
```

Finally, you will probably want to push the reestablished history:

```
git push origin master
```

You many need to use `--force` to push if the remote branch can not be “fast-forwarded” to the new commit (i.e. you dropped, or rewrote some existing commit, or otherwise rewrote some bit of history).

If you were in the middle of a rebase operation you should probably clean it up. You can check whether a rebase was in process by looking for the directory `.git/rebase-merge/` . You can manually clean up the in-progress rebase by just deleting that directory (e.g. if you no longer remember the purpose and context of the active rebase operation). Usually you would use `git rebase --abort` , but that does some extra resetting that you probably want to avoid (it moves HEAD back to the original branch and resets it back to the original commit, which will undo some of the work we did above).

edited Dec 23 '14 at 19:55 answered Apr 24 '11 at 19:56

Steve **14.3k** 18 83 135 Chris Johnsen **73.7k** 16 125 141

This may also be of help to some: sitaramc.github.com/concepts/detached-head.html – Menefee Jan 16 '13 at 1:44

4 Interesting from `man git-symbolic-ref` : "In the past, `.git/HEAD` was a symbolic link pointing at `refs/heads/master` . When we wanted to switch to another branch, we did `ln -sf refs/heads/newbranch .git/HEAD` , and when we wanted to find out which branch we are on, we did `readlink .git/HEAD` . But symbolic links are not entirely portable, so they are now deprecated and symbolic refs (as described above) are used by default." – dimadima Aug 2 '13 at 2:43

2 This answer was the final step that helped me get my origin/master back in sync with my local after accidentally doing a git reset --hard <sha> in Eclipse. The first step was to do a git reflog and recover the local commits (see stackoverflow.com/questions/5473/undoing-a-git-reset-hard-head1). Thank you. – jlpp Nov 27 '13 at 15:02

6 Awesome. Stuff like this shows that there is room for improvement in git, though. Sometimes while in the middle of project work there is simply no room to waste time with s**t like this and it's simply annoying. But other than that, git is really awesome. – badcat Jun 13 '14 at 6:36

5 It is unbelievable how complex versioning systems became – Antonio Sesto Mar 29 at 18:20

Just do this:

```
git checkout master
```

If that hasn't fixed it, this will:

```
git checkout -b temp
git branch -f master temp
git checkout master
```

edited Oct 8 at 1:21 answered Sep 18 '13 at 7:23
Daniel Alexiuc **4,019** 7 33 54

4 What did `git branch -f master temp` do? – Kirk Strobeck Jul 19 '14 at 17:22

3 @KirkStrobeck `git help branch` you will learn `git branch --force <old> <new>` so it is a copy. the `-f` is silly and will only serve to make you lose work if you already had a `temp` branch. So much for short instructions. – gcb Dec 13 '14 at 2:56

4 Be careful. If you are on a detached head then git checkout master won't automatically merge commits on that detached head. You have to find the detached commit with reflog and merge it into master. – nathan Feb 15 at 6:47

6 down vote, for losing all my changes on the detached branch :( – csomakk Apr 7 at 21:10

"git checkout master" will cause all changes to be lost if the detached head is not part of master – PiersyP Sep 4 at 9:32

Look here for basic explanation of detached head:

http://git-scm.com/docs/git-checkout

Command line to visualize it:

```
git branch
```

or

```
git branch -a
```

you will get output like below:

```
* (no branch)
master
branch1
```

The `* (no branch)` shows you are in detached head.

You could have come to this state by doing a `git checkout somecommit` etc. and it would have warned you with the following:

> You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.
>
> If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:
>
> git checkout -b new_branch_name

**Now, to get them onto master:**

Do a `git reflog` or even just `git log` and note your commits. Now `git checkout master` and `git merge` the commits.

```
git merge HEAD@{1}
```

Edit:

To add, use `git rebase -i` not only for deleting / killing commits that you don't need, but also for editing them. Just mention "edit" in the commit list and you will be able to amend your commit and then issue a `git rebase --continue` to go ahead. This would have ensured that you never came in to a detached HEAD.

edited Apr 24 '11 at 18:52 answered Apr 24 '11 at 18:41

manojlds
**127k** 16 227 260

Thanks for the detail and great information pointers here. Seems like an explicit merge wasn't necessary, but this visualized some concepts I'll go back to. Thanks. – Ben Zotto Apr 24 '11 at 21:42

What does "@{1}" do? – ebi May 1 at 16:11

I just ran into this issue and as soon as I read

> HEAD is the symbolic name for the currently checked out commit.

in the top-voted answer, I thought: "Ah-ha! I'll just `rebase`!" I wanted to bring the commits behind and including the detached `HEAD` that weren't yet in `master` into `master`, and `git rebase` lets you do just that:

```
git rebase HEAD master
```

To me this says:

> Take the parent commits of HEAD back to the point at which the histories of HEAD and master diverged, and play those commits on top of master. Then check out master.

Reconciling further with `origin/master` is then a matter of reconciling the newly-rebased `master` with `origin/master`. For this you can `ls-remote` or `fetch origin/master` and compare logs.

The other answers on this thread are quite nice and definitely worth carefully reading, but my approach seems fine if you're in a simple situation.

edited Aug 20 '13 at 13:48 answered Aug 2 '13 at 3:10

dimadima
**5,378** 4 36 55

git: "First, rewinding head to replay your work on top of it... Fast-forwarded master to HEAD." me: "nice!" – Benjamin Sep 10 at 22:37

## Get your detached commit onto its own branch

Simply run `git checkout -b mynewbranch`.

Then run `git log`, and you'll see that commit is now `HEAD` on this new branch.

answered May 20 '13 at 2:44
Rose Perrone
**21.6k** 15 112 131

If I do this, does `mynewbranch` attach to anything? – Benjohn Oct 5 at 15:00

Yes, it attaches to where the detached head would have been attached, which is exactly what I wanted. Thanks! – Benjohn Oct 6 at 11:01

---

if you have just master branch and wanna back to "develop" or a feature juste do this :

```
git checkout origin/develop
```

Note: checking out 'origin/develop'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout...

then

```
git checkout -b develop
```

It works :)

answered Nov 8 '13 at 13:24
amdev
**325** 2 13

1 What has worked for me is not 'git checkout origin/develop' but 'git checkout develop'. Using 'origin/develop' has always resulted in no changes, thereby remaining in "HEAD detached at origin/develop". Skipping the 'origin' part fixed everything. – DrStrangepork Apr 23 '14 at 18:58

---

All you have to do is 'git checkout [branch-name]' where [branch-name] is the name of the original branch from which you got into a detached head state. The (detached from asdfasdf) will disappear.

So for example, in branch 'dev' you checkout the commit asdfasd14314 ->

```
'git checkout asdfasd14314'
```

you are now in a detached head state

'git branch' will list something like ->

```
* (detached from asdfasdf)
  dev
  prod
  stage
```

but to get out of the detached head state and back to dev ->

```
'git checkout dev'
```

and then 'git branch' will list ->

```
* dev
  prod
  stage
```

but that is of course if you do not intend on keeping any changes from the detached head state but I find myself doing this a lot not intending to make any changes but just to look at a previous commit

answered Oct 17 '14 at 19:49
Adam Freeman
**665** 5 13

I got into a really silly state, I doubt anyone else will find this useful.... but just in case

```
git ls-remote origin
0d2ab882d0dd5a6db93d7ed77a5a0d7b258a5e1b        HEAD
6f96ad0f97ee832ee16007d865aac9af847c1ef6        refs/heads/HEAD
0d2ab882d0dd5a6db93d7ed77a5a0d7b258a5e1b        refs/heads/master
```

which I eventually fixed with

```
git push origin :HEAD
```

answered Oct 18 '13 at 3:14
KCD
2,576 1 17 32

---

for me it was as easy as deleting the local branch again, since I didn't have any local commits that I wanted to push:

so I did: git branch -d branchname

and then checking the branch out again

git checkout branchname

edited Jul 11 at 4:57
Do Young Kim
3 1

answered Jun 16 at 10:25
Klaus
79 1 5

---

If you want to push your current detached HEAD (check `git log` before), try:

```
git push origin HEAD:master
```

to send your detached HEAD into master branch at origin. If your push gets rejected, try `git pull origin master` first to get the changes from origin. If you don't care about the changes from origin and it's rejected, because you did some intentional rebase and you want to replace origin/master with your currently detached branch - then you may force it ( `-f` ).

answered Sep 17 at 20:35
kenorb
7,883 6 61 59

---

In my case, I run `git status` and I saw that I had a few untracked files on my working directory.

To make the rebase work, I just had to clean them (since I didn't need them).

answered Jun 19 at 17:47
falsarella
4,911 3 21 59

---

I just ran into this issue today. Pretty sure I solved it by doing:

```
git branch temp
git checkout master
git merge temp
```

I was on my work computer when I figured out how to do this, and now I'm running into the same problem on my personal comp. So will have to wait till monday when I'm back at work comp to see exactly how I did it.

edited Sep 17 at 20:39
kenorb
7,883 6 61 59

answered Nov 8 '14 at 4:46
adamwineguy
1

1 the second git command seems wrong.. – StarShine Mar 6 at 13:20