How can I reconcile detached HEAD with master/origin?

Asked 8 years, 11 months ago Active 1 year, 3 months ago Viewed 954k times

1546

734

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 train 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 May 29 '17 at 18:21

asked Apr 24 '11 at 17:51

Ben Zotto
64.5k 22 132 198

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

2507

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 it 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
```

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 may need to add `--force` to the end of this command 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 Jan 15 '19 at 18:16
hellter
687 9 26

answered Apr 24 '11 at 19:56
Chris Johnsen
171k 24 188 181

6 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." – Dmitry Minkovsky Aug 2 '13 at 2:43

10 I agree with @AntonioSesto: for most projects (even fairly large ones) you don't need the mind-boggling complexity that is Git. My brain rebels at grappling with something that is so clearly over-engineered. I don't need it, and I don't want it. – Jasper Sprengers Sep 17 '15 at 11:24

keep your current head but to designate it as `master`. Other goals also make sense, and call for other recipes. – Adrian Ratnapala Nov 6 '15 at 10:28

37 Lol at the comment gurning about length. Whereas the rest of us simply scan through till we reach the line that says "To recover from your situation [...]", and go from there - while making a mental note that there's a useful well-explained backstory that we can read on a rainy day. The *option* to read more does not hurt you, but it *does* stand to benefit others. – underscore_d Mar 10 '16 at 22:53

5 THIS is why I hate git. – Monica Heddneck Jan 5 '19 at 2:25

624

Just do this:

```
git checkout master
```

Or, if you have changes that you want to keep, do this:

```
git checkout -b temp
git checkout -B master temp
```

edited Jan 25 '17 at 0:12

answered Sep 18 '13 at 7:23

Daniel Alexiuc
12k 9 51 71

57 This is a dangerous response. People arriving at this answer have different states and "just do this to fix it" responses don't answer questions. This one can easily destroy work. – Archonic Mar 10 '16 at 18:51

15 !"git checkout master" will cause all changes to be lost if the detached head is not part of master !! – Tony Mar 28 '16 at 20:53

3 @Blauhirn You probably had the commit checked out, not the branch. The branch still points to the same commit, but you're in a different 'mode'. – Daniel Alexiuc Jun 13 '16 at 1:12

1 `git reset` should come with a warning "If you have no idea what you're doing, stop it". Just recovered from an hour of terror thinking I'd lost the last week of work. Thanks! – Opus1217 Jun 13 '16 at 1:26

1 Agree with @Archonic It is important to understand how git works before you blindly run any commands. You can save time by not reading the big answer, but can lose more time if your work is lost. – Yusufali2205 Mar 11 '19 at 16:15

127

I ran into this issue and when I read in the top voted answer:

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

I thought: Ah-ha! If `HEAD` is the symbolic name for the currenlty checkout commit, I can reconcile it against `master` by rebasing it against `master`:

```
git rebase HEAD master
```

This command:

3. plays those commits on top of `master`

The end result is that all commits that were in `HEAD` but not `master` are then also in `master`. `master` remains checked out.

Regarding the remote:

> a couple of the commits I'd killed in the rebase got pushed, and the new ones committed locally aren't there.

The remote history can no longer be fast-forwarded using your local history. You'll need to force-push (`git push -f`) to overwrite the remote history. If you have any collaborators, it usually makes sense to coordinate this with them so everyone is on the same page.

After you push `master` to remote `origin`, your remote tracking branch `origin/master` will be updated to point to the same commit as `master`.

edited Sep 27 '17 at 21:38

answered Aug 2 '13 at 3:10

Dmitry Minkovsky
25.1k 19 83 122

3 git: "First, rewinding head to replay your work on top of it... Fast-forwarded master to HEAD." me: "nice!" – Benjamin Sep 10 '15 at 22:37

81

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

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
239k 53 412 386

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

6 What does "@{1}" do? – ebi May 1 '15 at 16:11

Get your detached commit onto its own branch

35

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
51.5k 41 188 216

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

1 Yes, it attaches to where the detached head would have been attached, which is exactly what I wanted. Thanks! – Benjohn Oct 6 '15 at 11:01

22

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

```
git checkout origin/develop
```

Note: checking out **origin/develop**.

branches by performing another checkout...

then

```
git checkout -b develop
```

It works :)

edited May 12 '16 at 8:19
Abhishek Joshi
3 3

answered Nov 8 '13 at 13:24
amdev
1,954 21 38

7 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

18

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`). In case you lost some access to previous commits, you can always run `git reflog` to see the history from all branches.

To get back on a master branch, while keeping the changes, try the following commands:

```
git rebase HEAD master
git checkout master
```

See: Git: "Not currently on any branch." Is there an easy way to get back on a branch, while keeping the changes?

edited May 23 '17 at 10:31
Community ♦
1 1

answered Sep 17 '15 at 20:35
kenorb
95.4k 42 503 544

2 This indeed sends the detached commits to origin/master. To attach the head to the local branch do this: stackoverflow.com/a/17667057/776345 – Paschalis Jul 13 '16 at 7:21

When I do this I get This repository is configured for Git LFS but 'git-lfs' was not found on your path. If you no longer wish to use Git LFS, remove this hook by deleting .git/hooks/post-checkout. – user2568374 Mar 12 '18 at 18:07

I found this question when searching for `You are in 'detached HEAD' state.`

My normal flow is:

```
git checkout master
git fetch
git checkout my-cool-branch
git pull
```

This time I did:

```
git checkout master
git fetch
git checkout origin/my-cool-branch
# You are in 'detached HEAD' state.
```

The problem is that I accidentally did:

```
git checkout origin/my-cool-branch
```

Rather than:

```
git checkout my-cool-branch
```

The fix (in my situation) was simply to run the above command and then continue the flow:

```
git checkout my-cool-branch
git pull
```

answered Nov 30 '18 at 19:52

user664833

14.4k 18 75 117

11

The following worked for me (using only branch master):

```
git push origin HEAD:master
git checkout master
git pull
```

The first one pushes the detached HEAD to remote origin.

The second one moves to branch master.

The third one recovers the HEAD that becomes attached to branch master.

Problems might arise at the first command if the push gets rejected. But this would no longer be a problem of detached head, but is about the fact that the detached HEAD is not aware of some remote changes.

edited Jan 7 '18 at 12:31

answered Jan 7 '18 at 12:13

Daniel Porumbel

155 1 5

currently on a branch. Please specify which branch you want to merge with. – user2568374 Mar 12 '18 at 16:59

11

I just ran into this issue today and am 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 computer. So will have to wait till Monday when I'm back at the work computer to see exactly how I did it.

edited Aug 20 '18 at 15:24
Peter Mortensen
25.4k 21 90 118

answered Nov 8 '14 at 4:46
adamwineguy
131 1 2

@StarShine Kenorb fixed it. Now it saves your detached commits to a new branch, temp, switches to master, and merges temp into master. – Cees Timmerman Jan 15 '16 at 10:42

I don't know why ppl are downvoting this, it fixed my problem stat but you may want to include the delete temp branch command. – GlassGhost Mar 23 '16 at 4:58

8

If you are completely sure HEAD is the good state:

```
git branch -f master HEAD
git checkout master
```

You probably can't push to origin, since your master has diverged from origin. If you are sure no one else is using the repo, you can force-push:

```
git push -f
```

Most useful if you are on a feature branch no one else is using.

edited Mar 15 '16 at 11:01

answered Mar 1 '16 at 9:31
geon
6,190 2 28 34

6

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'
```

```
* (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
1,113 8 17

6

As pointed by Chris, I had following situation

`git symbolic-ref HEAD` fails with `fatal: ref HEAD is not a symbolic ref`

However `git rev-parse refs/heads/master` was pointing to a good commit from where I could recover (In my case last commit and you can see that commit by using `git show [SHA]`

I did a lot messy things after that, but what seems to have fixed is just,

```
git symbolic-ref HEAD refs/heads/master
```

And head is re attached!

edited Oct 11 '17 at 10:16

answered Jul 15 '17 at 17:04

Varun Garg
1,582 15 31

1 Thanks! My head had gotten detached. I could catch it up to master but they just happened to be pointing at the same commit rather than head pointing to master which pointed to the commit. Good tip =D – RagingRoosevelt Mar 22 '18 at 19:51

4

Instead of doing `git checkout origin/master`

just do `git checkout master`

then `git branch` will confirm your branch.


181 5

4

I had this problem today, where I had updated a submodule, but wasn't on any branch. I had already committed, so stashing, checkout, unstashing wouldn't work. I ended up cherry-picking the detached head's commit. So immediately after I committed (when the push failed), I did:

```
git checkout master
git cherry-pick 99fe23ab
```

My thinking went: I'm on a detached head, but I want to be on master. Assuming my detached state is not very different from master, if I could apply my commit to master, I'd be set. This is exactly what cherry-pick does.

answered Apr 11 '18 at 5:58


prewett
1,330 11 17

3

If you did some commits *on top of master* and just want to "backwards merge" `master` there (i.e. you want `master` to point to `HEAD`), the one-liner would be:

```
git checkout -B master HEAD
```

1. That creates a new branch named `master`, even if it exists already (which is like moving `master` and that's what we want).
2. The newly created branch is set to point to `HEAD`, which is where you are.
3. The new branch is checked out, so you are on `master` afterwards.

I found this especially useful in the case of sub-repositories, which also happen to be in a detached state rather often.

answered Apr 7 '17 at 14:14


quazgar
2,920 24 36

3

I had the same problem and I have resolved it by going through the following steps.

If you need to keep your changes

1. First you need to run `git checkout master` command to put you back to the master branch.
2. If you need to keep your changes just run `git checkout -b changes` and `git checkout -B master changes`

If you don't need your changes

1. To removes all untracked files from your branch run `git clean -df`.
2. Then you need to clear all unstaged changes within your repository. In order to do that

3. Finally you have to put your branch back to the master branch by using `git checkout master` command.

edited Dec 27 '17 at 11:53

answered Dec 27 '17 at 11:46

Aravinda Meewalaarachchi

926 1 7 15

3

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 Aug 20 '18 at 15:25

Peter Mortensen

25.4k 21 90 118

answered Jun 16 '15 at 10:25

Klaus

107 1 5

1

When I personally find myself in a situation when it turns out that I made some changes while I am not in `master` (i.e. `HEAD` is detached right above the `master` and there are no commits in between) stashing might help:

```
git stash # HEAD has same content as master, but we are still not in master
git checkout master  # switch to master, okay because no changes and master
git stash apply  # apply changes we had between HEAD and master in the first place
```

answered Oct 11 '17 at 21:03

Ben Usman

4,523 5 36 53

1

In simple words, the detached HEAD state means **you are not checked out to HEAD (or tip) of any branch**.

Understand With an Example

A branch in most of the cases is sequence of multiple commits like:

Commit 1: master-->branch_HEAD(123be6a76168aca712aea16076e971c23835f8ca)

Commit 2: master-->123be6a76168aca712aea16076e971c23835f8ca-->branch_HEAD(100644a76168aca712aea16076e971c23835f8ca)

As you can see above in case of sequence of commits, your branch points to your latest

since HEAD of your branch points to *100644a76168aca712aea16076e971c23835f8ca* and technically you are checked out at HEAD of no branch. Hence, you are in the detached HEAD state.

Theoretical Explanation

In this Blog it's clearly stating a Git repository is a tree-of-commits, with each commit pointing to its ancestor with each commit pointer is updated and these pointers to each branch are stored in the .git/refs sub-directories. Tags are stored in .git/refs/tags and branches are stored in .git/refs/heads. If you look at any of the files, you'll find each tag corresponds to a single file, with a 40-character commit hash and as explained above by @Chris Johnsen and @Yaroslav Nikitenko, you can check out these references.

edited Aug 20 '18 at 15:28

Peter Mortensen
25.4k 21 90 118

answered Jul 23 '16 at 5:39

Keshav
982 9 16

0

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
7,696 3 52 60

0

This worked for me perfectly:

1. `git stash` to save your local modifications

If you want to discard changes

```
git clean -df
git checkout -- .
```

git clean removes all untracked files (warning: while it won't delete ignored files mentioned directly in .gitignore, it may delete ignored files residing in folders) and git checkout clears all unstaged changes.

2. `git checkout master` to switch to the main branch (Assuming you want to use master)
3. `git pull` to pull last commit from master branch
4. `git status` in order to check everything looks great

```
On branch master
Your branch is up-to-date with 'origin/master'.
```


jmojico
957 10 17

0

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

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

edited Aug 20 '18 at 15:25

Peter Mortensen
25.4k 21 90 118

answered Jun 19 '15 at 17:47

falsarella
10.9k 7 58 98

0

If you are using EGit in Eclipse: assume your master is your main development branch

- commit you changes to a branch, normally a new one
- then pull from the remote
- then right click the project node, choose team then choose show history
- then right click the master, choose check out
- if Eclipse tells you, there are two masters one local one remote, choose the remote

After this you should be able to reattach to the origin-master.

edited Aug 20 '18 at 18:57

Peter Mortensen
25.4k 21 90 118

answered Jul 28 '17 at 23:21

Junchen Liu
4,297 8 44 56

-1

I had the same problem. I stash my changes with `git stash` and hard reset the branch in local to a previous commit(I thought it caused that) then did a `git pull` and I am not getting that head detached now. Dont forget `git stash apply` to have your changes again.

answered Nov 5 '18 at 14:42

SamL
23 1 7

-2

```
git checkout checksum  # You could use this to peek previous checkpoints
git status # You will see HEAD detached at checksum
git checkout master # This moves HEAD to master branch
```

edited Aug 20 '18 at 15:26

Peter Mortensen
25.4k 21 90 118

answered Jun 3 '16 at 18:50

David Yang
1 1

Highly active question. Earn 10 reputation in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.