master branch and 'origin/master' have diverged, how to 'undiverge' branches'?

up vote459down vote**229**
Somehow my master and my origin/master branch have diverged. I actually don't want them to be diverged. How can I view these differences and 'merge' them?

3,69771727
You can review the differences with a:

```
git log HEAD..origin/master
```

When you have a message like:

> "Your branch and 'origin/master' have diverged, # and have 1 and 1 different commit(s) each, respectively."

, check if you need to update `origin`. If `origin` is up-to-date, then some commits have been pushed to `origin` from another repo while you made your own commits locally.

```
... o ---- o ---- A ---- B  origin/master (upstream work)
                  \
                   C  master (your work)
```

You based commit C on commit A because that was the latest work you had fetched from upstream at the time.

However, before you tried to push back to origin someone else pushed commit B.
Development history has diverged into separate paths.

You can then merge or rebase. See Pro Git: Git Branching - Rebasing for details.

Merge
Use the git merge command:

```
$ git merge origin/master
```

This tells Git to integrate the changes from `origin/master` into your work and create a merge commit.
The graph of history now looks like this:

```
... o ---- o ---- A ---- B  origin/master (upstream work)
                  \      \
                   C ---- M  master (your work)
```

The new merge commit M has two parents, each representing one path of development that led to the content stored in the commit.

Note that the history behind M is now non-linear.

Rebase
Use the git rebase command:

```
$ git rebase origin/master
```

This tells Git to replay commit C (your work) as if you had based it on commit B instead of A.
CVS and Subversion users routinely rebase their local changes on top of upstream work when they update before commit.
Git just adds explicit separation between the commit and rebase steps.

The graph of history now looks like this:

```
... o ---- o ---- A ---- B  origin/master (upstream work)
                         \
                          C'  master (your work)
```

Commit C' is a new commit created by the git rebase command.
It is different from C in two ways:

1. It has a different history: B instead of A.
2. It's content accounts for changes in both B and C: it is the same as M from the merge example.

Note that the history behind C' is still linear.
We have chosen (for now) to allow only linear history in `cmake.org/cmake.git`.
This approach preserves the CVS-based workflow used previously and may ease the transition.
An attempt to push C' into our repository will work (assuming you have permissions and no one has pushed while you were rebasing).

The git pull command provides a shorthand way to fetch from origin and rebase local work on it:

```
$ git pull --rebase
```

This combines the above fetch and rebase steps into one command.

up vote12down vote
I found myself in this situation when I tried to *rebase* a branch that was tracking a remote branch, and I was trying to rebase it on master. In this scenario if you try to rebase, you'll most likely find your branch *diverged* and it can create a mess that isn't for git nubees!

Let's say you are on branch my_remote_tracking_branch, which was branched from master

> `$ git status`
> # On branch my_remote_tracking_branch
>
> nothing to commit (working directory clean)

And now you are trying to rebase from master as:

> git rebase master

STOP NOW and save yourself some trouble! Instead, use merge as:

> git merge master

Yes, you'll end up with extra commits on your branch. But unless you are up for "un-diverging" branches, this will be a much smoother workflow than rebasing. See this blog for a much more detailed explanation.

On the other hand, if your branch is only a *local* branch (i.e. not yet pushed to any remote) you should definitely do a rebase (and your branch will not *diverge* in this case).

Now if you are reading this because you already *are* in a "diverged" scenario due to such rebase, you can get back to the last commit from origin (i.e. in an un-diverged state) by using:

> git reset --hard origin/my_remote_tracking_branch

up vote0down vote
In my case it caused by this sequence: I first ran `git pull` command. changes in origin had conflicts with my local repo. I resolved conflicts and I forgot to commit. then I changed some files and when I ran git status command I saw my local modification as unstaged local modification and merged changes as staged local modification. so I should commit changes from merge by `git commit` at first and then commit unstaged changes by `git commit -a` or commit them altogether by `git commit -a` If I checkout my working copy instead of commit all changes of my colleagues was destroyed.