

# **Fix git “tip of your current branch is behind its remote counterpart” - 4 real-world solutions**



When working with `git` a selection of GitLab, GitHub, BitBucket and rebase-trigger-happy colleagues/collaborators, it's a rite of passage to see a message like the following:

```
Pushing to git@github.com:some-project/some-repo.git
To git@github.com:some-project/some-repo.git
    ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'git@github.com:some-project/some-repo.git'
hint: Updates were rejected because the tip of your current branch is b
hint: its remote counterpart. Merge the remote changes (e.g. 'git pull')
hint: before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

# What causes “tip of your current branch is behind”?

Git works with the concept of local and remote branches. A local branch is a branch that exists in your local version of the git repository. A remote branch is one that exists on the remote location (most repositories usually have a remote called `origin`). A remote equates roughly to a place where your git repository is hosted (eg. a GitHub/GitLab/BitBucket/self-hosted Git server repository instance).

Remotes are useful to share your work or collaborate on a branch.

“the tip of your current branch is behind its remote counterpart” means that there have been changes on the remote branch that you don’t have locally.

There tend to be 2 types of changes to the remote branch: someone added commits or someone modified the history of the branch (usually some sort of rebase).

These 2 cases should be dealt with differently.

# How can you get your local branch back to a state that's pushable?

We're now going to explore how to achieve a state in the local branch where the remote won't reject the push.

## 1. No rebase(s): merge the remote branch into local

In the message we can see:

*Updates were rejected because the tip of your current branch is behind its remote counterpart. Merge the remote changes (e.g. 'git pull') before pushing again.*

So is it as simple as doing:

```
git pull
```

And solving any conflicts that arise.

We shouldn't do this if someone has rebased on the remote. The history is different and a merge could have a nasty effect on the history. There will be a weird history with equivalent commits in 2 places plus a merge commit.

Read on for solutions to the “remote has been rebased” case.

## **2. Remote rebase + no local commits: force git to overwrite files on pull**

If you don't have any changes that aren't on the remote you can just do:

**Warning:** *this is a destructive action, it overwrites all the changes in your local branch with the changes from the remote*

```
git reset --hard origin/branch-name
```

This is of course very seldom the case but offers a path to the two following solutions.

Solutions 3. and 4. save the local changes somewhere else (the git stash or another branch).

They reset the local branch from the origin using the above command. Finally they re-apply any local changes and send them up.

### **3. Remote rebase + local commits: soft git reset, stash, "hard pull", pop stash, commit**

Say you've got local changes (maybe just a few commits).

A simple way to use the knowledge from 2. is to do a “soft reset”.

## Options to “soft reset”

Option 1, say the first commit you've added has sha `<first-commit-sha>` use:

*Note the `^` which means the commit preceding `<first-commit-sha>`*

```
git reset <first-commit-sha>^ .
```

Option 2, if you know the number of commits you've added, you can also use the following,

replace `3` with the number of commits you want to “undo”:

```
git reset HEAD~3 .
```

You should now be able to run `git status` and see un-staged (ie. “modified”) file changes from the local commits we’ve just “undone”.

## Save your changes to the stash

Run `git stash` to save them to the stash (for more information [see git docs for stash](#)).

If you run `git status` you’ll see the un-staged (“modified”) files aren’t there any more.

## Run the hard pull as seen in the previous section

Run `git reset --hard origin/branch-name` as seen in 2.

## Un-stash and re-commit your changes



To restore the stashed changes:

```
git stash pop
```

You can now use `git add` (hopefully with the `-p` option, eg. `git add -p .`) followed by `git commit` to add your local changes to a branch that the remote won't reject on push.

Once you've added your changes, `git push` shouldn't get rejected.

## **4. Remote rebase + local commits 2: checkout to a new temp branch, “hard pull” the original branch, cherry-pick from temp onto branch**

That alternative to using stash is to branch off of the local branch, and re-apply the commits of

a “hard pull”-ed version of the branch.

## Create a new temp branch

To start with we'll create a new temporary local branch. Assuming we started on

branch `branch-name` branch (if not, run `git checkout branch-name`) we can do:

```
git checkout -b temp-branch-name
```

This will create a new branch `temp-branch-name` which is a copy of our changes but in a new

branch

## Go back to the branch and “hard pull”

We'll now go back to branch `branch-name` and overwrite our local version with the remote one:

```
git checkout branch-name
```

Followed by `git reset --hard origin/branch-name` as seen in 2.

## Cherry-pick the commits from temp branch onto the local branch

We'll now want to switch back to `temp-branch-name` and get the SHAs of the commits we want to apply:

```
git checkout temp-branch-name
```

Followed by

```
git log
```

To see which commits we want to apply (to exit `git log` you can use `q`).

## Cherry-pick each commit individually

Say we want to apply commits `<commit-sha-1>` and `<commit-sha-2>`.

We'll switch to the branch that has been reset to the remote version using:

```
git checkout branch-name
```

We'll then use cherry-pick ([see cherry-pick git docs](#)) to apply those commits:

```
git cherry-pick <commit-sha1> && git cherry-pick <commit-sha2>
```

# Cherry-pick a range of commits

If you've got a bunch of commits and they're sequential, you can use the following (for git 1.7.2+)

We'll make sure to be on the branch that has been reset to the remote version using:

```
git checkout branch-name
```

*For git version 1.7.2+, credit to [François Marier](#) in “[Cherry-picking a range of git commits](#)” - [Feeding the Cloud](#)*

```
git cherry-pick <first-commit-sha>^..<last-commit-sha>
```

You should now be able to `git push` the local branch to the remote without getting rejected.