Ankur Biswas

Follow   527 Followers   About

Most common Git mistakes and how to fix them

Ankur Biswas Sep 7, 2018 · 4 min read

Photo by Ilya Pavlov on Unsplash

If you ever worked on a big project with many collaborators, then obviously you were using **Git** as your control system. And when you are working with something as complex as **Git**, we all make mistakes. In this article, I am going to discuss some very common mistakes programmers made while using **Git** and how to solve them. Now, without any further ado, let's get started 🏃,

Spelled last commit message wrong

After a good few hours of coding, it's easy for a spelling error to sneak into your commit messages. Luckily, there's a simple fix.

```
git commit --amend
```

This will open up your editor and allow you to make a change to that last commit message. No one needs to know you spelled, "Initial commment" with three "m"s.

Spelling mistake on branch name

Let's suppose, it is almost 15:00 and you have not had your lunch yet, so in your hunger, you have named your new branch `feature-brunch`. Delicious.

There's a solution for this too. We rename this branch in a similar way to how we rename a file with the `mv` command: by moving it to a new location with the correct name.

```
git branch -m feature-brunch feature-branch
```

If you have already pushed this branch, there are a couple of extra steps required. We need to delete the old branch from the remote and push up the new one:

```
git push origin --delete feature-brunch
git push origin feature-branch
```

Accidentally committed all changes to the master branch

So you are working on a new feature and in your haste, you forgot to open a new branch for it. You have already committed a load of files and now those commits are all sitting on the master branch.

So we can roll back all those changes to a new branch with the following three commands:

Note: Make sure you commit or stash your changes first, or all will be lost!

```
git branch feature-branch
git reset HEAD~ --hard
git checkout feature-branch
```

This creates a new branch, then rolls back the master branch to where it was before you made changes, before finally checking out your new branch with all your previous changes intact.

Forgot to add a file to that last commit

Another common Git pitfall is committing too early. You missed a file, forgot to save it, or need to make a minor change for the last commit to make sense. `--amend` is your friend once again.

Add that missed file then run that trusty command.

```
git add missed-file.txt
git commit --amend
```

At this point, you can either amend the commit message or just save it to keep it the same.

Added a wrong file in the repo

But what if you do the exact opposite? What if you added a file that you didn't want to commit? A rogue ENV file, a build directory, a picture of your dog that you accidentally saved to the wrong folder? It's all fixable.

If all you did was stage the file and you haven't committed it yet, it's as simple as resetting that staged file:

```
git reset /assets/img/misty-and-pepper.jpg
```

If you have gone as far as committing that change, no need to worry. You just need to run an extra step before:

```
git reset --soft HEAD~1
git reset /assets/img/misty-and-pepper.jpg
rm /assets/img/misty-and-pepper.jpg
git commit
```

This will undo the commit, remove the image, then add a new commit in its place.

Oops... I did it again

This command is for when everything has gone wrong. When you have copy-pasted one too many solutions from Stack Overflow and your repo is in a worse state than it was when you started. We have all been there.

`git reflog` shows you a list of all the things you've done. It then allows you to use Git's magical time-traveling skills to go back to any point in the past. I should note, this is a last resort thing and should not be used lightly. To get this list, type:

```
git reflog
```

Every step we took, every move we made, Git was watching us. Running that on our project gives us this:

```
3ff8691 (HEAD -> feature-branch) HEAD@{0}: Branch: renamed
refs/heads/feature-brunch to refs/heads/feature-branch
3ff8691 (HEAD -> feature-branch) HEAD@{2}: checkout: moving
from master to feature-brunch
2b7e508 (master) HEAD@{3}: reset: moving to HEAD~
3ff8691 (HEAD -> feature-branch) HEAD@{4}: commit: Adds the
client logo
2b7e508 (master) HEAD@{5}: reset: moving to HEAD~1
37a632d HEAD@{6}: commit: Adds the client logo to the project
2b7e508 (master) HEAD@{7}: reset: moving to HEAD
2b7e508 (master) HEAD@{8}: commit (amend): Added contributing
info to the site
dfa27a2 HEAD@{9}: reset: moving to HEAD
dfa27a2 HEAD@{10}: commit (amend): Added contributing info to
the site
700d0b5 HEAD@{11}: commit: Addded contributing info to the site
efba795 HEAD@{12}: commit (initial): Initial commit
```

Take note of the left-most column, as this is the index. If you want to go back to any point in the history, run the below command, replacing `{index}` with that reference, e.g. `dfa27a2`.

```
git reset HEAD@{index}
```

Have some Git tips of your own? Let us know in the comments below, I'd love to hear them.

Thanks for reading! If you found this helpful, don't forget to leave some claps 👏🏼. Share this with your friends and followers! Tweet to me at iAnkurBiswas and let me know how you liked this article!

Git Programming Development Tech Productivity