Git: (some of) the nifty parts

Matthias Nilsson

May 31, 2016

Introduction

About me:

- Git user since 2010
- Worked with 100+ developers on the same code base for multiple years
- My favorite methodology is Cookie Driven Development

Setting the stage

git bisect

Relies on good commit history

Commit history

What is a commit?

What is a commit?

- A way of saving your work

What is a commit?

- A way of saving your work
- A snapshot of a previous state

What is a commit?

- A way of saving your work
- A snapshot of a previous state
- A way to document development over time

Why is history important?

Because it tells us what has changed

```
d690bbc add content to foo
85c6bfe fix typo
b4a6944 WIP: add content to foo
```

vs.

```
0a5e89d add content to foo
```

Not all history is worth keeping

Not every commit should be saved

Cleaning up history

```
git commit --amend
```

Lets you edit the last commit

But what if that's not enough?

```
git rebase --interactive
```

From the manual for `git-rebase(1)`:

> The interactive mode is meant for this type of workflow:
>
> 1. have a wonderful idea
> 2. hack on the code
> 3. prepare a series for submission
> 4. submit

Our original history:

```
885bb69 add content to bar
d690bbc add content to foo
85c6bfe fix typo
b4a6944 WIP: add content to foo
bb3be90 initial commit
```

Specify starting point:

```
$ git rebase -i bb3be90 # initial commit
```

Specify starting point:

```
$ git rebase -i bb3be90 # initial commit
```

This lets you edit the list of commits in your editor.

Specify starting point:

```
$ git rebase -i bb3be90 # initial commit
```

This lets you edit the list of commits in your editor.

(Pro tip: always have an initial commit you can use as a starting point.)

We decide what should be done with the commits:

```
reword b4a6944 WIP: add content to foo
fixup 85c6bfe fix typo
fixup d690bbc add content to foo
pick 885bb69 add content to bar
```

Our new history looks like this:

```
6a38155 add content to bar
1436cf1 add content to foo
bb3be90 initial commit
```

(Note that the hashes have changed.)

You will have to tell Git to overwrite the remote:

```
$ git push --force
```

(Provided that the commits we rebased had been pushed earlier.)

Use it to clean up branches before merging

NOTE

Make sure you coordinate with your collaborators

Rewriting history can lead to trouble for others

Rewriting history can lead to trouble for others

A branch containing commits that no longer exist on the remote will result in nasty conflicts.

Coordination is the key

Don’t use `rebase` for ”public” commits

Don’t use `rebase` for ”public” commits

(E.g. `master` in a public repo or a branch that many are based on)

Recovering from a history rewrite

Our collaborator has rebased the branch we are working on

Our collaborator has rebased the branch we are working on

We have unpushed commits in that branch

Our branch:

```
819defb add README
885bb69 add content to bar
d690bbc add content to foo
85c6bfe fix typo
b4a6944 WIP: add content to foo
bb3be90 initial commit
```

The remote branch:

```
6a38155 add content to bar
1436cf1 add content to foo
bb3be90 initial commit
```

Create a temp branch based on your local branch:

```
$ git checkout -b add-foo-and-bar-temp
```

Get your local branch up to date with the remote:

```
$ git branch -D add-foo-and-bar
$ git checkout -t origin/add-foo-and-bar
```

Get your local branch up to date with the remote:

```
$ git branch -D add-foo-and-bar
$ git checkout -t origin/add-foo-and-bar
```

Or

```
$ git checkout add-foo-and-bar
# find out how many commits differ
$ git fetch --all && git status
$ git reset --hard HEAD~n
$ git pull
```

Get your local branch up to date with the remote:

```
$ git branch -D add-foo-and-bar
$ git checkout -t origin/add-foo-and-bar
```

Or

```
$ git checkout add-foo-and-bar
# find out how many commits differ
$ git fetch --all && git status
$ git reset --hard HEAD~n
$ git pull
```

Or

```
$ git checkout add-foo-and-bar
$ git reset --hard origin/add-foo-and-bar
```

Pick your unpushed commits:

```
# on branch add-foo-and-bar
# get commit hashes
$ git log add-foo-and-bar-temp
$ git cherry-pick 819defb
```

(I tend to only cherry-pick individual commits.)

Other flavors of rebase

Rant: `git pull` **should never result in a merge**

Tell Git to only apply remote changes if they can be fast-forwarded:

```
git config --global merge.ff only
```

Pull in remote changes and apply your local commits after:

```
git pull --rebase
```

Home assignment:

How can we use `git pull --rebase` to recover from a history rewrite?

Home assignment:

How can we use `git pull --rebase` to recover from a history rewrite?

(Tip: `git rebase --interactive` **is a good start.)**

Home assignment:

How can we use `git pull --rebase` to recover from a history rewrite?

(Tip: `git rebase --interactive` is a good start.)

Bonus question: When will this not work?

Use `git rebase` to keep up to date:

```
$ git checkout master && git pull
$ git checkout -b my-feature-branch
# do stuff: add/commit/push
# prepare for pull request
$ git checkout master && git pull
$ git checkout my-feature-branch
# add the new changes to your branch
$ git rebase master
```

When things go horribly wrong

git reflog

`git reflog` tracks changes to `HEAD` locally:

```
3783c49 HEAD@{0}: cherry-pick: completely rewrite bar
d5b962e HEAD@{1}: reset: moving to HEAD~1
ac48e40 HEAD@{2}: rebase -i (finish): returning to refs/heads/master
ac48e40 HEAD@{3}: rebase -i (pick): completely rewrite bar
d5b962e HEAD@{4}: rebase -i (pick): add content to bar
97e39a3 HEAD@{5}: rebase -i (fixup): add content to foo
28a4547 HEAD@{6}: rebase -i (fixup): # This is a combination of 2 commits.
d47e71c HEAD@{7}: rebase -i (reword): add content to foo
b4a6944 HEAD@{8}: cherry-pick: fast-forward
bb3be90 HEAD@{9}: rebase -i (start): checkout bb3be9022765add167e4d014479a8104d5c7db79
819defb HEAD@{10}: checkout: moving from foo to master
```

Using `git reflog` we can find a point to step back to:

```
$ git rebase -i <commit>
# here things go horribly wrong
$ git reflog # to find a point before our mistake
$ git reset --hard <another commit>
```

NOTE

Careful when deleting branches, `git reflog` **will not help you there.**

Recap

Git commands to make your life easier:

- `git rebase` **to ensure you have a clean history**
- `git cherry-pick` **to pick (individual) commits**
- `git reflog` **to recover from mishaps**

Communication is essential

Questions?