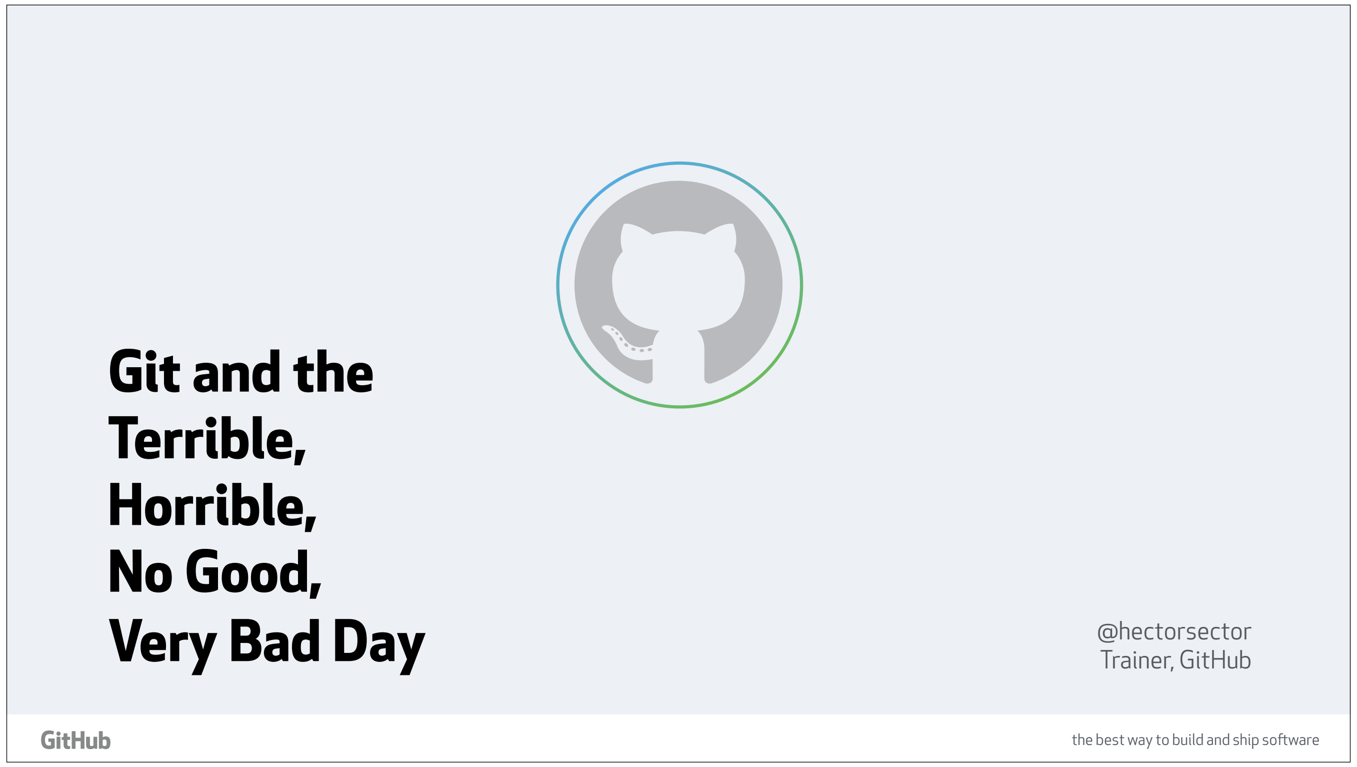


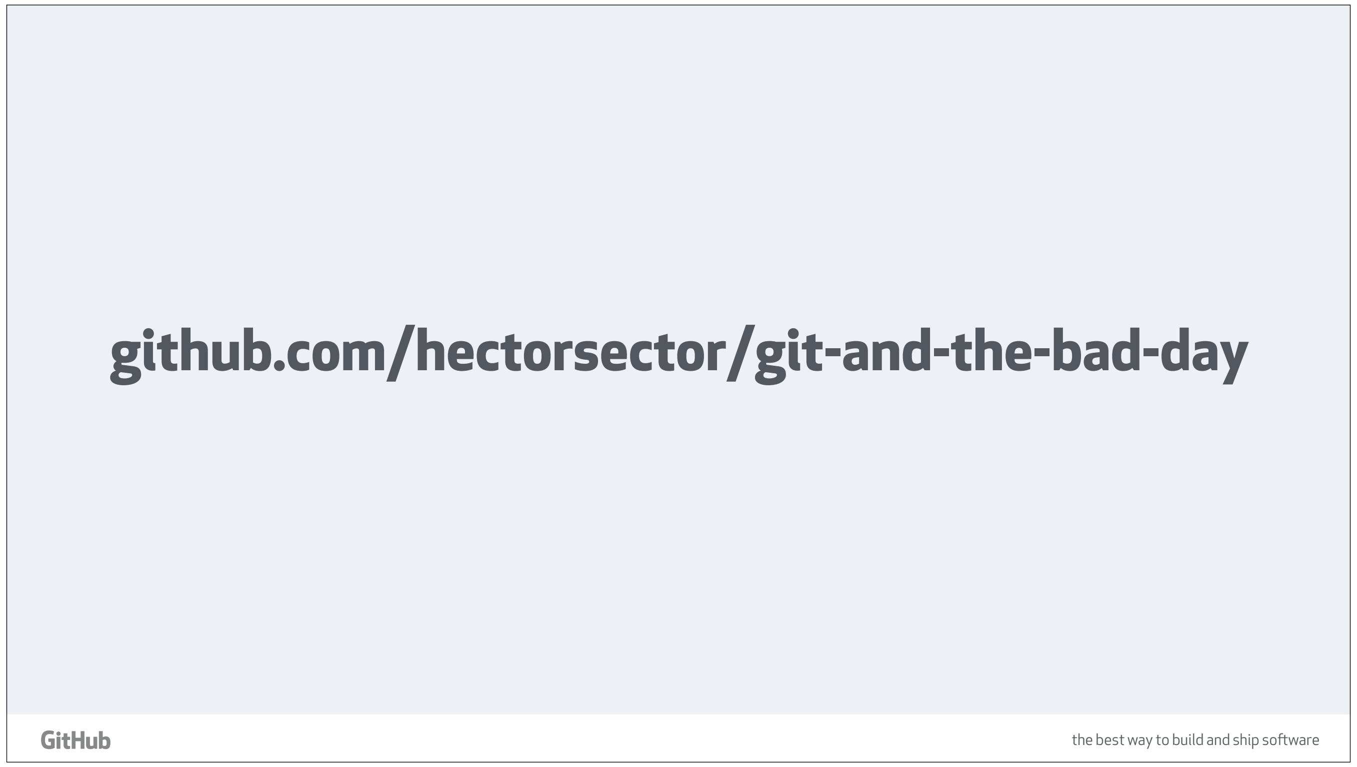
Git and the Terrible, Horrible, No Good, Very Bad Day

@hectorsector
Trainer, GitHub

GitHub

the best way to build and ship software

Hey there! I'm Hector and I'm with GitHub Training. You know, I love teaching anything, but Git is one of the best things to teach because there's always something new to learn, even for me. So much of the anxiety I encounter from folks learning Git for the first time, is the fear that they'll get themselves into a situation that they can't get themselves out of. So what I thought we'd do in session, is highlight some pretty gnarly but not too-far-off scenarios that you might find yourselves in, so that you can see that it's possible to get out of pretty much any sticky situation.

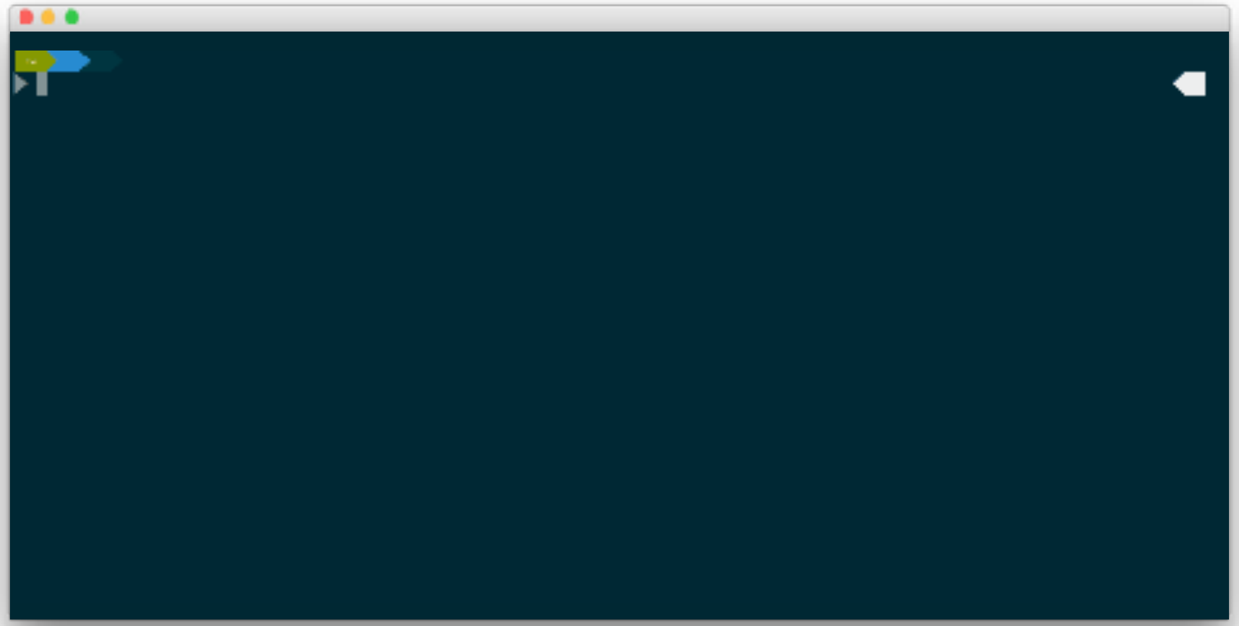


github.com/hectorsector/git-and-the-bad-day

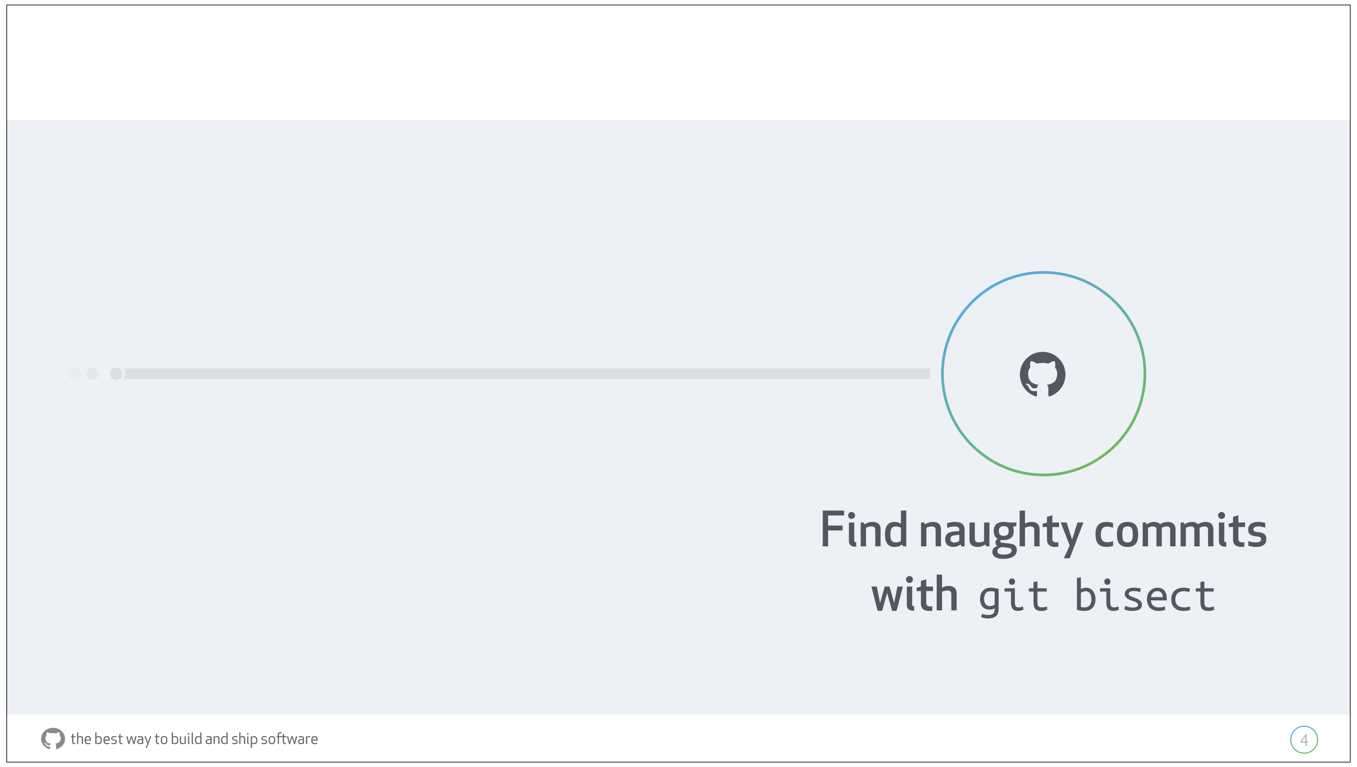
GitHub

the best way to build and ship software

If you'd like to follow along, just clone this repo and I'll list the commands as we go through them.



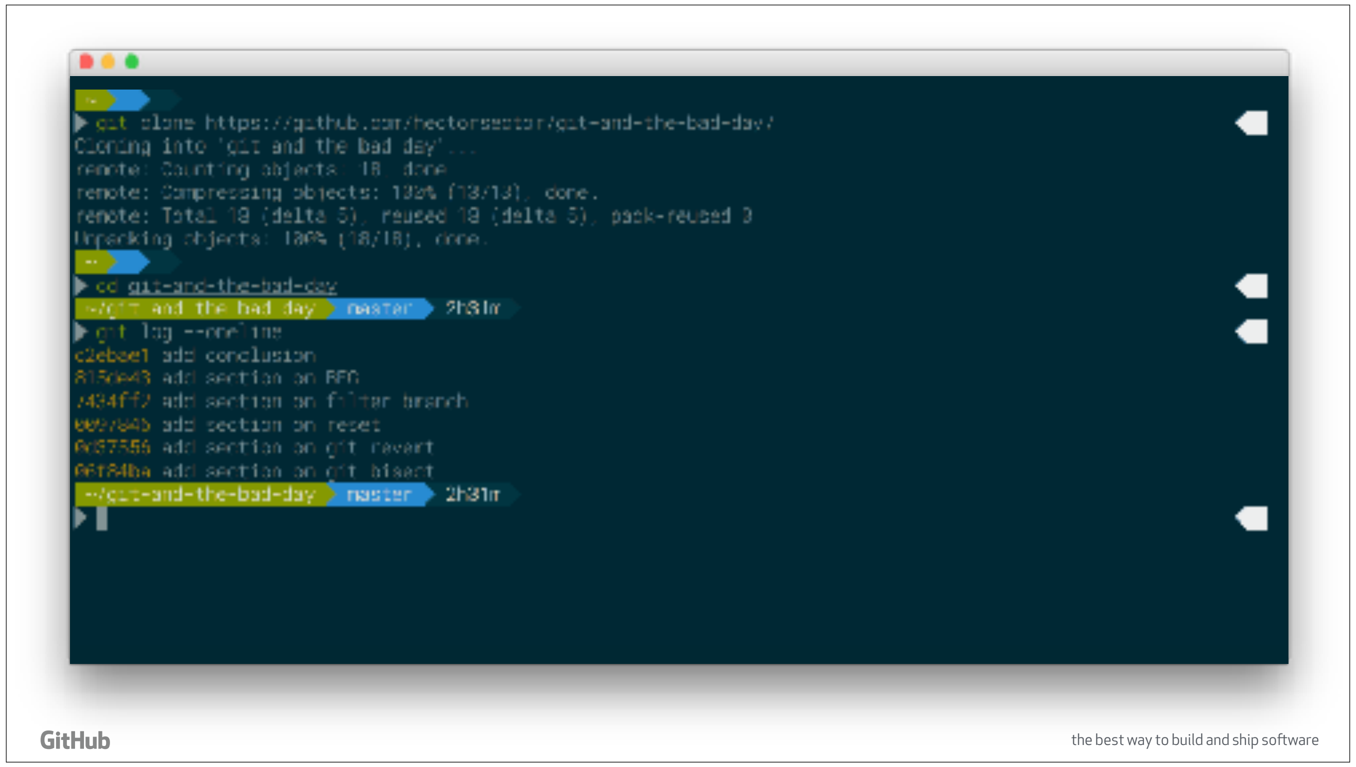
We'll cover porcelain commands, but won't really get into plumbing.
Once you've closed the repo, let's take a look at our log to figure out what our history currently looks like.
As you can see if you've got the repo, this is a story I've written modeled after the children's book *Alexander and the Terrible, Horrible, No Good, Very Bad Day*. It will double as the repo that we will work on together.
So let's dive right in.



Find naughty commits with `git bisect`

Sometimes, we know there's a commit that's broken our build, but we can't place our finger on which commit is naughty, or perhaps our history is too massive to manually sort through hundreds of commits.

Git Bisect can help us find naughty commits using a simple binary search.



```
~
▶ git clone https://github.com/hectorsector/git-and-the-bad-day/
Cloning into 'git-and-the-bad-day'...
remote: Counting objects: 18, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 18 (delta 5), reused 18 (delta 5), pack-reused 0
Unpacking objects: 100% (18/18), done.
~
▶ cd git-and-the-bad-day
~/git-and-the-bad-day  master  2h31m
▶ git log --oneline
c2ebee1 add conclusion
815de43 add section on BFG
7434ff2 add section on filter branch
0027845 add section on reset
0c67556 add section on git revert
06f84ba add section on git bisect
~/git-and-the-bad-day  master  2h31m
▶ 
```

GitHub

the best way to build and ship software

And that's precisely what's happened in my repo. I wanted to cover two options for removing files from history: `git filter-branch` and BFG. But I didn't have enough time, so I want to undo the commit that adds BFG to my story, but first I have to find it.

The bisect starts with `git bisect start`. You then have to tell git where you first noticed the problem, and where you last had no problem. You do that with `git bisect bad` and `git bisect good` followed by the commit id. This divides your the problematic history in half and checks you out to one of the two halves in your history.

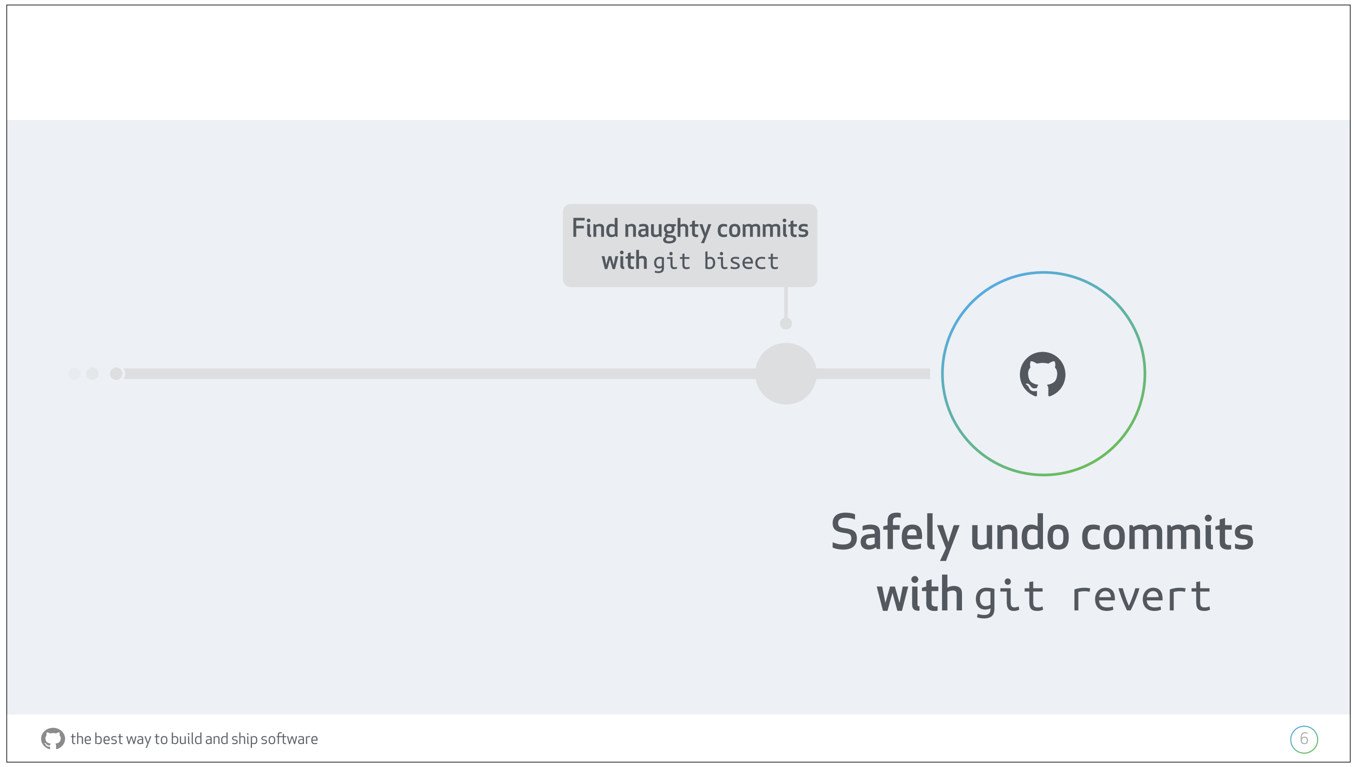
You can run some tests on that commit, if it looks good you run `git bisect good` and git won't check that half of the commit history anymore. If it looks bad, you can run `git bisect bad`.

In my case, there's no problem with this commit as there's no mention of BFG, so I'll run `git bisect good`.

The next commit shows me I mentioned BFG, so I'll run `git bisect bad`. Git will then cut that history in half, and show me the next commit. It looks like there's a problem there so I'll run `git bisect bad` again, and finally `git bisect good` since this last commit looks fine.

Git then shows me the offending commit, so I can revert it.

However, because I'm still checked out to the last commit in the bisect chain, I'll have to run `git bisect reset` to check me back out to master.

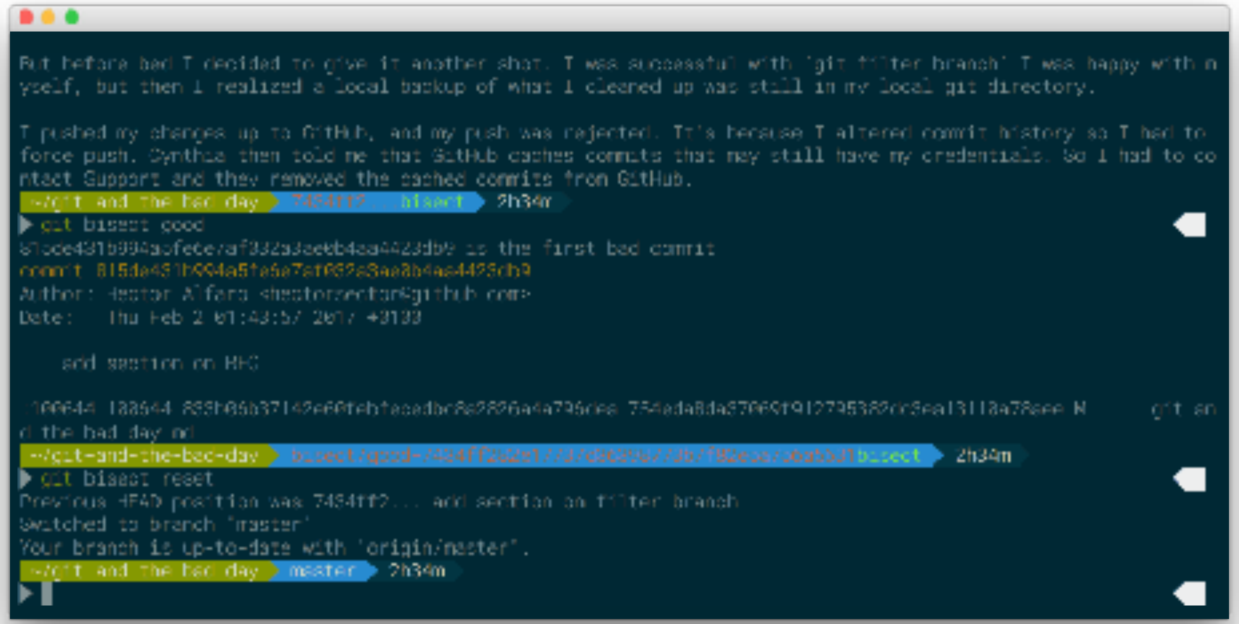


Find naughty commits with `git bisect`

Safely undo commits with `git revert`

Now that we’ve found an offending commit, I want to make sure it is indeed naughty and then revert it.

Revert is a command that is generally safe because it doesn’t alter history.



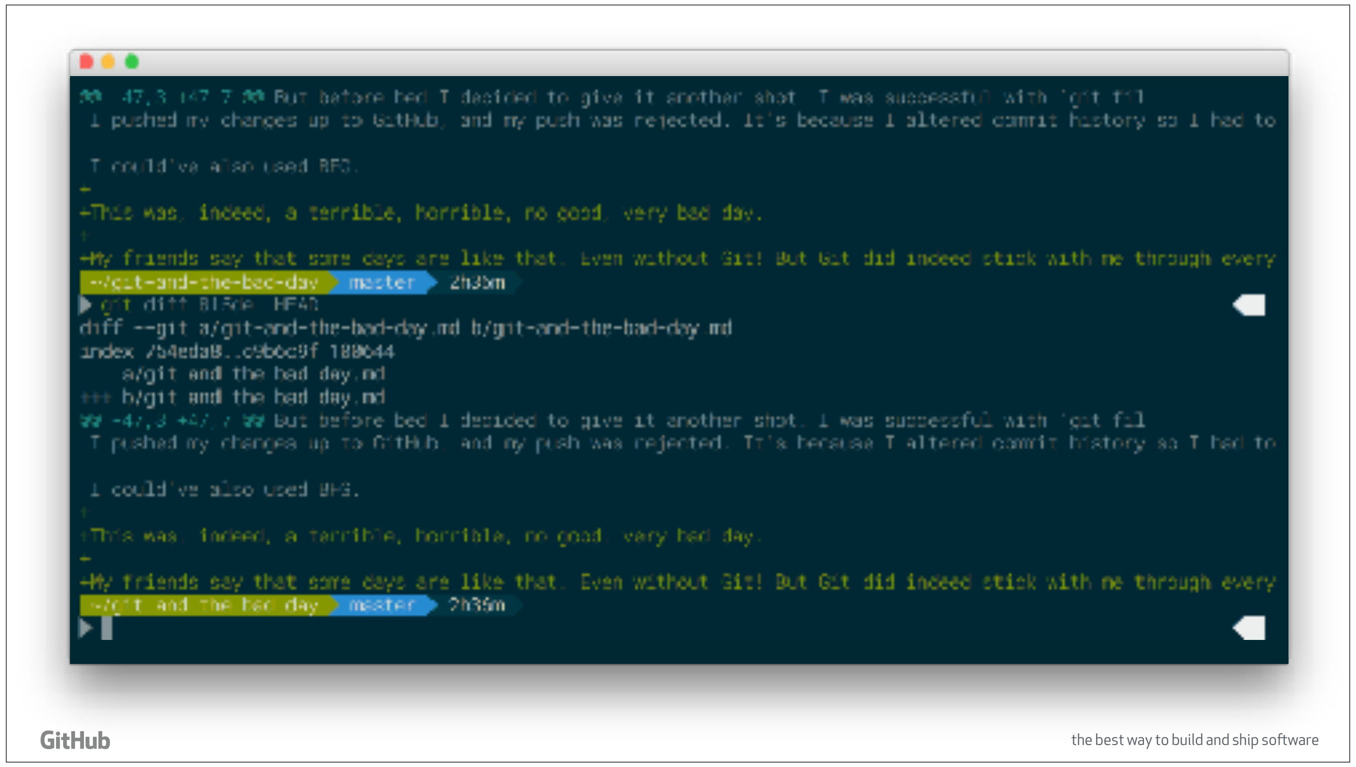
```
But before bed I decided to give it another shot. I was successful with 'git filter branch' I was happy with m
yself, but then I realized a local backup of what I cleaned up was still in my local git directory.

I pushed my changes up to GitHub, and my push was rejected. It's because I altered commit history so I had to
force push. Cynthia then told me that GitHub caches commits that may still have my credentials. So I had to co
ntact Support and they removed the cached commits from GitHub.
~/git and the bad day  7484ff2...bisect  2h34m
▶ git bisect good
815de431b994a5fe6e7af832a8ae8b4aa4423db9 is the first bad commit
commit 815de431b994a5fe6e7af832a8ae8b4aa4423db9
Author: Hector Alfaro <hectorsector@github.com>
Date:   Thu Feb 2 01:43:57 2017 +0100

    add section on RHC

:100644 100644 833b06b37142e60feb1acadbc8a2826a4a736dea 754ada8da57069f912795382cc9ea13118a78eee M      git an
d the bad day md
~/git-and-the-bad-day  bisect/good-7484ff2b62e1773/c8c3507736f82eca/a6ab531bisect  2h34m
▶ git bisect reset
Previous HEAD position was 7484ff2... add section on filter branch
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
~/git and the bad day  master  2h34m
▶ ▌
```

GitHub   the best way to build and ship software

First, I want to look at the offending commit to ensure that it does indeed contain what I want to undo.

And if you'd like, you can also diff it against our HEAD commit, which is the most recent commit on master.

You'll notice a git diff defaults to performing diff against HEAD as well.



```
@@ -47,3 +47,7 @@ But before bed I decided to give it another shot. I was successful with `git fil
 I pushed my changes up to GitHub, and my push was rejected. It's because I altered commit history so I had to

 I could've also used BFG.
+
+This was, indeed, a terrible, horrible, no good, very bad day.
+
+My friends say that some days are like that. Even without Git! But Git did indeed stick with me through every
 ~/git-and-the-bad-day  master  2h35m
▶ git diff 815ce HEAD
diff --git a/git-and-the-bad-day.md b/git-and-the-bad-day.md
index 7b4eda8..c9bcc9f 100644
--- a/git and the bad day.md
+++ b/git and the bad day.md
@@ -47,3 +47,7 @@ But before bed I decided to give it another shot. I was successful with `git fil
 I pushed my changes up to GitHub, and my push was rejected. It's because I altered commit history so I had to

 I could've also used BFG.
+
+This was, indeed, a terrible, horrible, no good, very bad day.
+
+My friends say that some days are like that. Even without Git! But Git did indeed stick with me through every
 ~/git and the bad day  master  2h36m
▶ ▌
```

GitHub   the best way to build and ship software

So to perform the revert, we'll do git revert and the commit id.

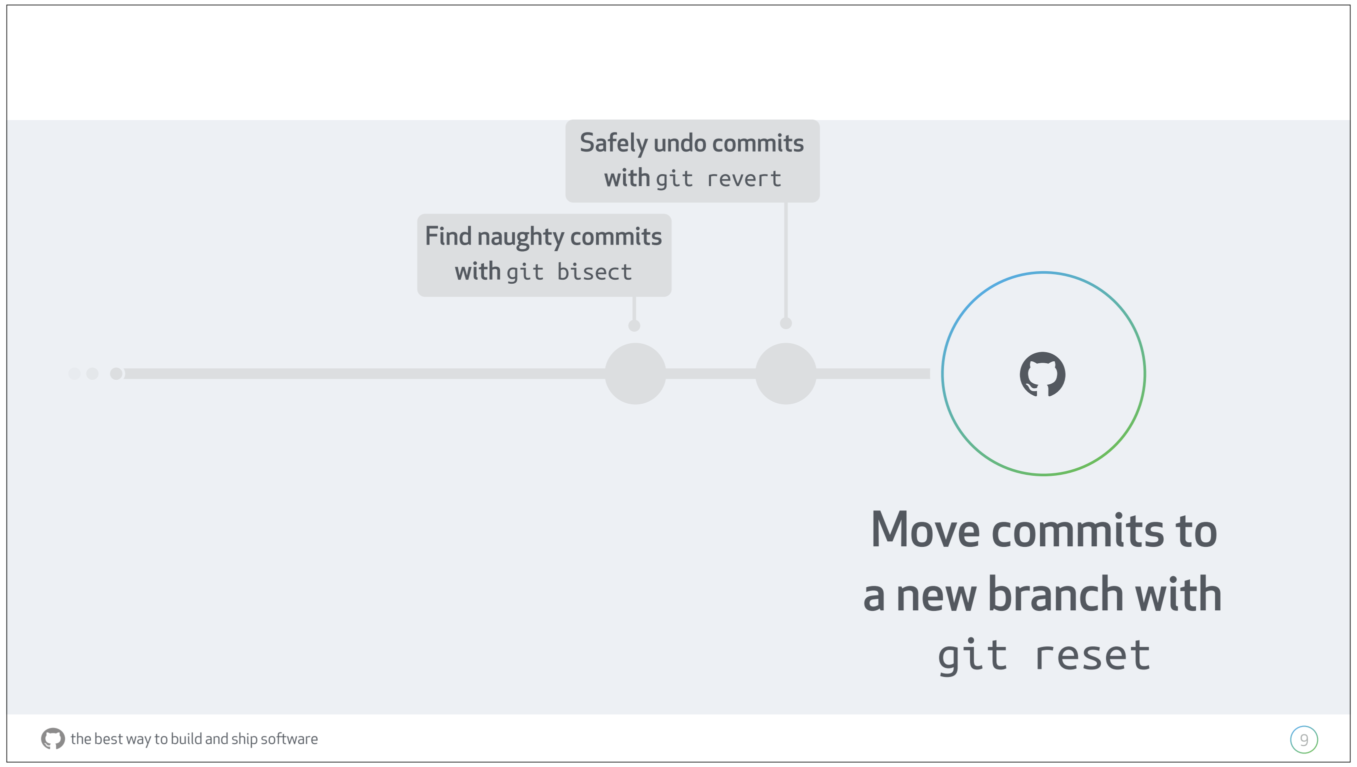
In this case, we've run into a merge conflict. So let's figure out why the conflict occurred.

I can see that the offending lines are down at the bottom, so I'll remove the conflict markers and the line I don't want, but I'll keep the stuff that comes after it.

Because I've now staged my conflicting file, I can run `git diff —staged` to ensure I'm seeing what I expect.

Because the revert stopped halfway, we can finish it up with `git revert —continue`

Git then lets us input the final commit message, and there we go, we've reverted the commit.

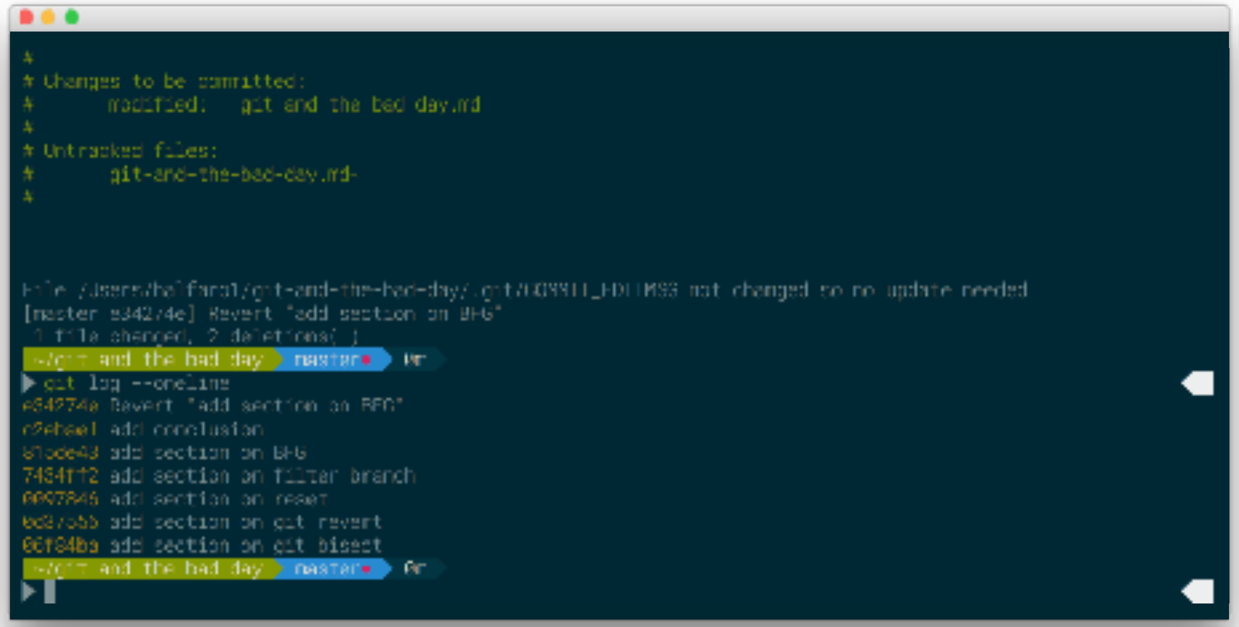


Safely undo commits with `git revert`
Find naughty commits with `git bisect`

Move commits to a new branch with `git reset`

A common scenario is working on a local copy of the repo, and committing directly to master as opposed to a new branch. That's what's just happened, too. I reverted my commit, but I reverted it directly on master. If this was production code, I wouldn't have an opportunity to test it or open PRs to allow others to collaborate with me. I could end up with a broken build.

So let's take a look at how to move this commit to its own branch.



```
#
# Changes to be committed:
#       modified:   git and the bad day.md
#
# Untracked files:
#       git-and-the-bad-day.md-
#



File /Users/halfarol/git-and-the-bad-day/.git/COMMIT_EDITMSG not changed so no update needed
[master e3427 4e] Revert "add section on BFG"
 1 file changed, 2 deletions(-)
~/git and the bad day  master●  ✔
▶ git log --oneline
e34274e Revert "add section on BFG"
c2ebae1 add conclusion
87ode43 add section on BFG
7434ff2 add section on filter branch
0097846 add section on reset
bd37c55 add section on git revert
06f84ba add section on git bisect
~/git and the bad day  master●  ✔
▶
```

GitHub

the best way to build and ship software

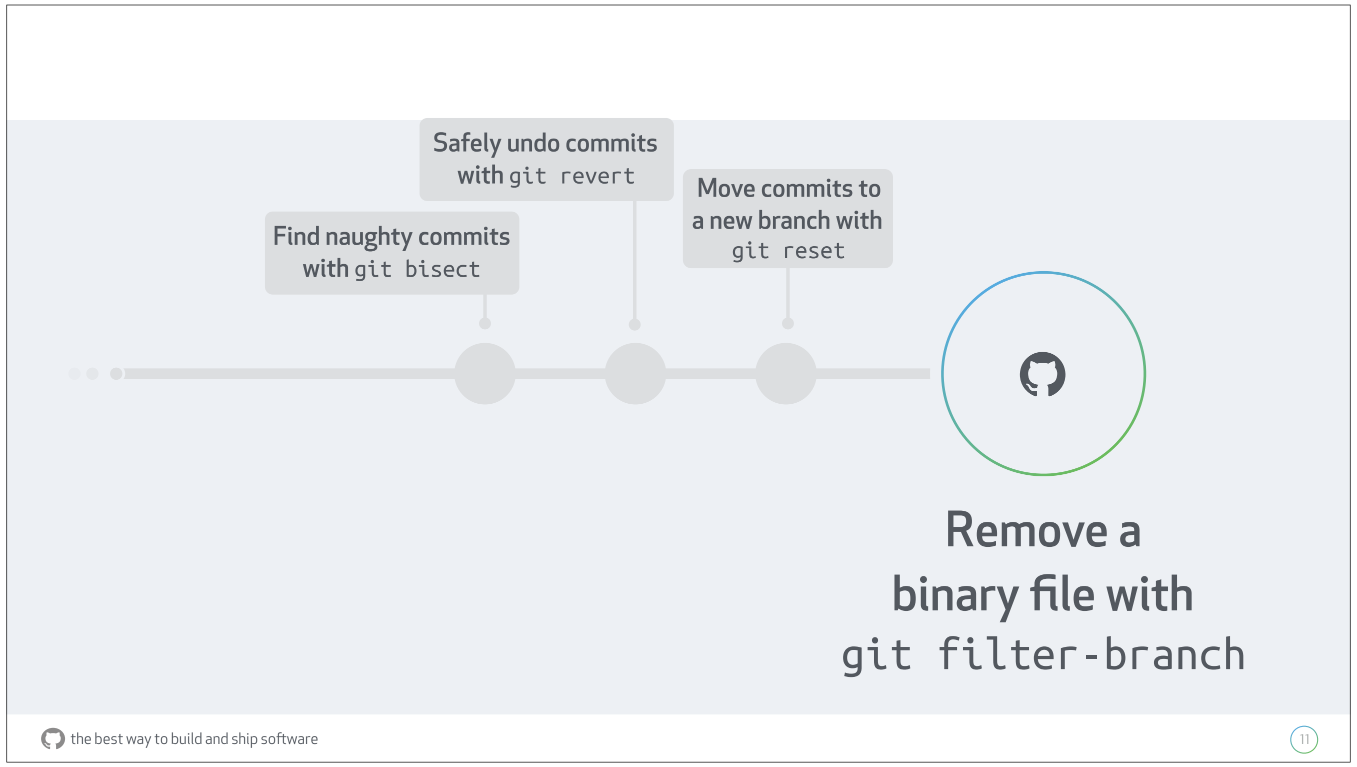
We'll first create the branch, I'll call it `remove-bfg`.

Next, we'll use a reset command. Reset is a command which points HEAD to a commit that you specify. In this case, we'll point head back one commit, but only on the master branch.

When we check out to the `remove-bfg` branch, you can see that the revert commit is still present.

Now reset does require a force push, as it is a command that can alter the git history that the rest of your collaborators is counting on. It's safe to do on local commits, but not commits that have been pushed to a remote.

There's another way I could have moved my commit to a branch, and that's with a reset, followed by a cherry pick on a new branch.



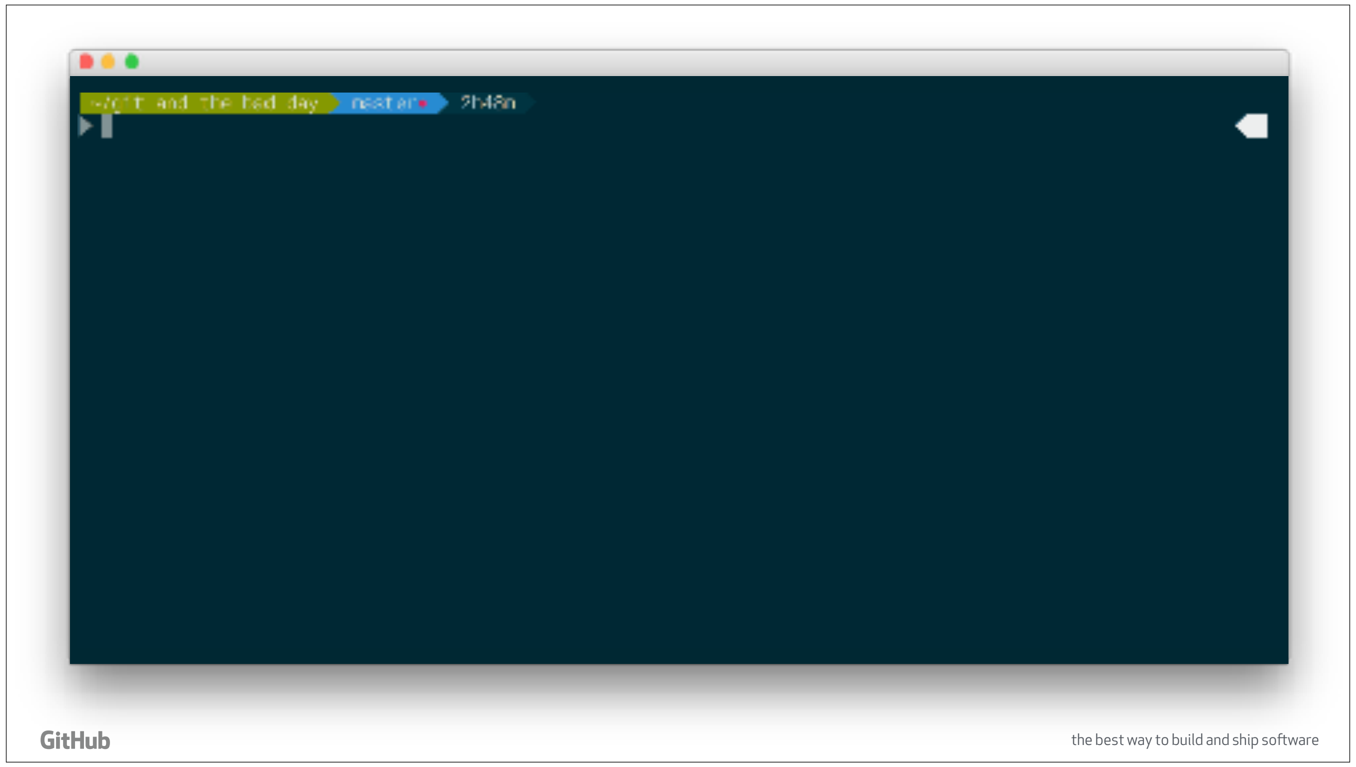
Find naughty commits with git bisect
Safely undo commits with git revert
Move commits to a new branch with git reset

Remove a binary file with `git filter-branch`

One of the defining characteristics of Git is that its history has integrity, and that all copies of the codebase are safely kept — that's why we want version control, right?

But this can be problematic when someone adds a large binary file to the repository, or accidentally commits their credentials. If they've not pushed it up the remote, it's fairly straight forward to blast off that last commit with a `reset`, but if they've crafted commits atop the large binary or credentials, or worse, pushed those artifacts to a remote, you're in trouble. Because simply removing said artifacts in the next commit from remove them from the project's history.

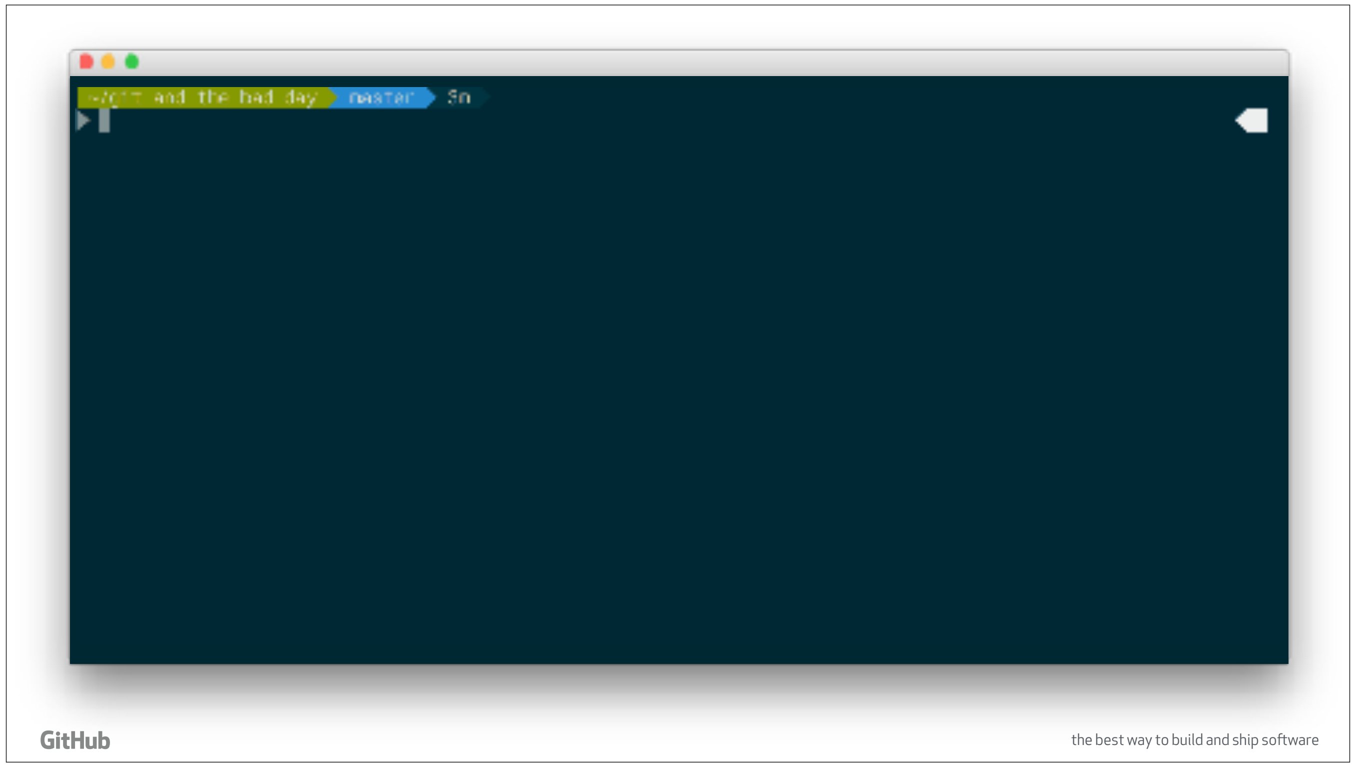
So that's what's happened to me here. I'll add a "large" binary to the repo, and then we'll use `git filter-branch` to help us remove it from history.



```
~/git and the bad day  master•  2h48m
▶
```

First, I'll create a little history here to make it interesting. I'll commit the Keynote file to `master`. And so you can see that we have two branches here. So let's say I've tested my revert and I'm satisfied, and I'm merging that in too.

`git lol` is just an alias I've set up for `git log` with some extra options.



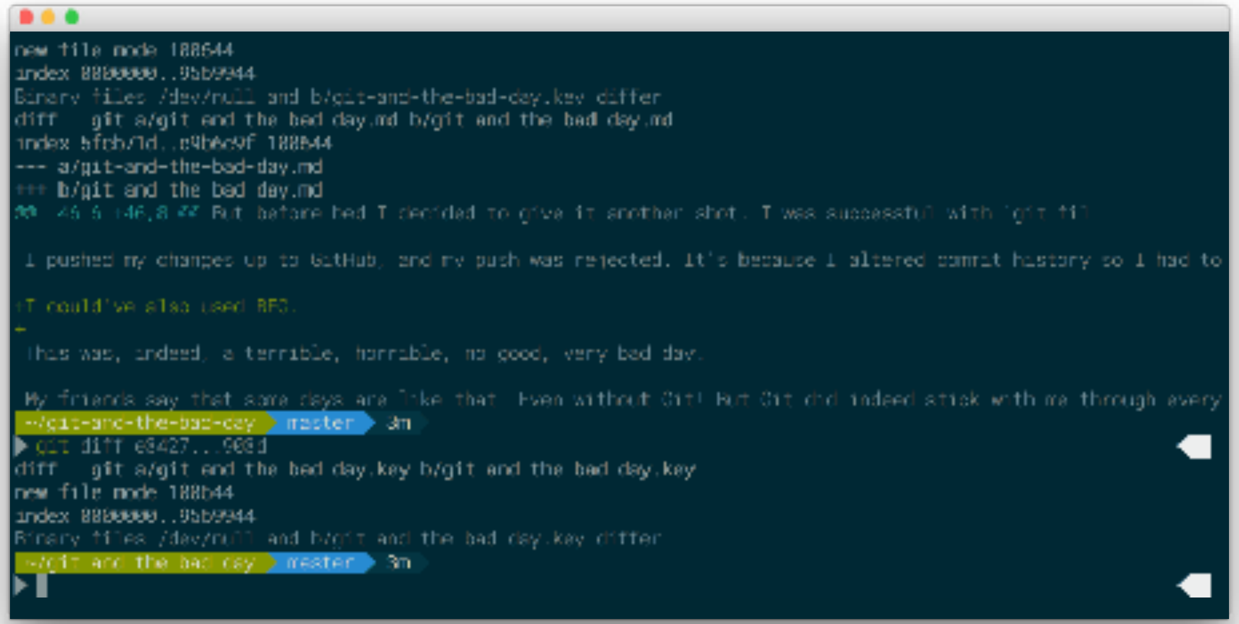
```
~/git and the bad day  master  $n
▶ 
```

GitHub

the best way to build and ship software

So the first thing we can do, is toy with the idea of a reset. In my case, I’ve only added my commit recently, so it may be relatively simple to reset that. So I’ll commit these two commits that I just merged here, to see what exactly was different between them two: and I can see two changes, I’ve added the keynote, and I’ve added BFG (from the viewpoint of commit 903d).

But that’s different from using 3-dot syntax, so I put this in here to give us a chance to talk about it, too. When you diff with 3-dot syntax, you’re looking at a diff between a commit, and its merge base with the second commit. So in this case, from the view point of 903d against the merge base, the only thing I’ve added is the keynote file.

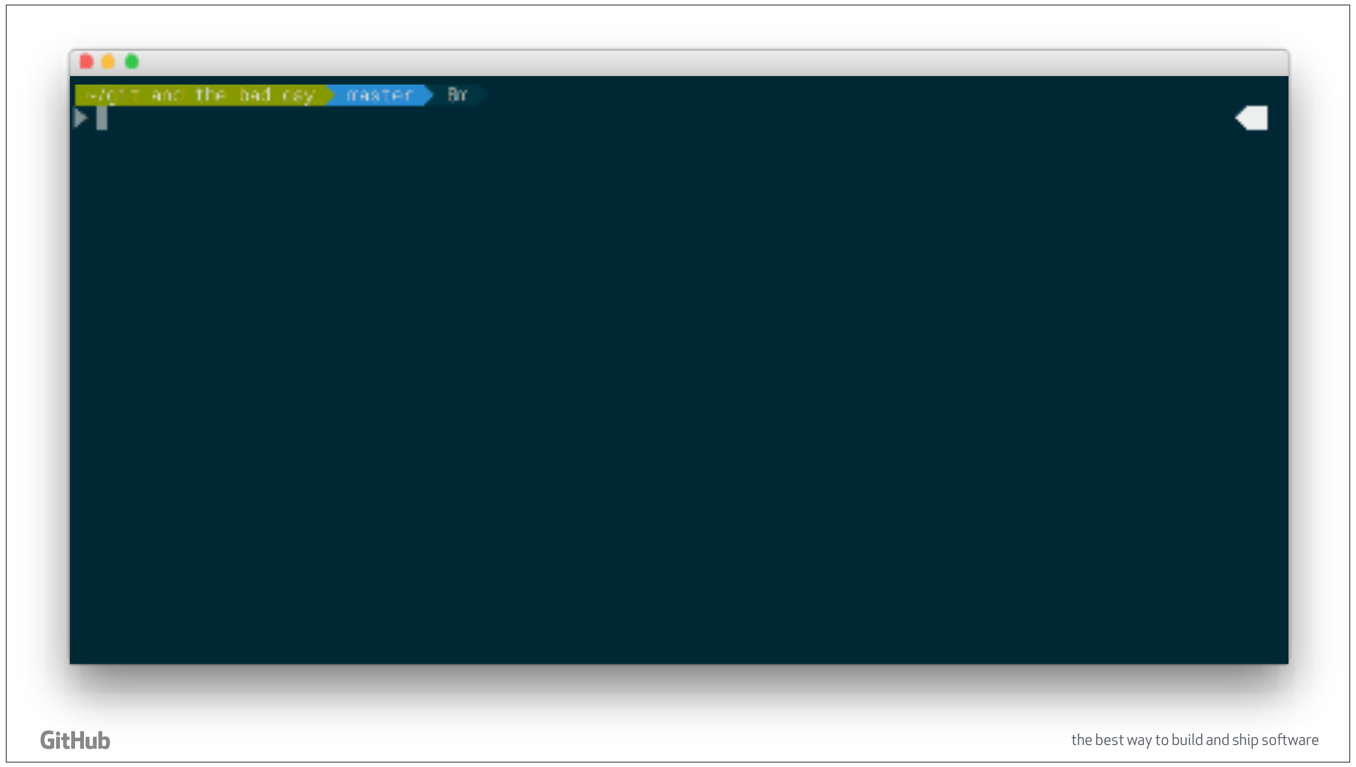


```
new file mode 100644
index 0000000..95b9944
Binary files /dev/null and b/git-and-the-bad-day.key differ
diff --git a/git-and-the-bad-day.md b/git-and-the-bad-day.md
index 5fcb71d..e9b6c9f 100644
--- a/git-and-the-bad-day.md
+++ b/git-and-the-bad-day.md
@@ -45,5 +46,8 @@ But before bed I decided to give it another shot. I was successful with `git fil

 I pushed my changes up to GitHub, and my push was rejected. It's because I altered commit history so I had to

+I could've also used BFG.
+
 This was, indeed, a terrible, horrible, no good, very bad day.

 My friends say that some days are like that. Even without Git! But Git did indeed stick with me through every
~/git-and-the-bad-day  master  3m
▶ git diff e3427...9631
diff --git a/git-and-the-bad-day.key b/git-and-the-bad-day.key
new file mode 100644
index 0000000..95b9944
Binary files /dev/null and b/git-and-the-bad-day.key differ
~/git-and-the-bad-day  master  3m
▶
```

GitHub

the best way to build and ship software

So armed with that information, it's safe to reset back, but when I reset past this merge commit here, I'll have to specify which parent to point HEAD to, so with the carat-2 syntax I'm specifying that I'd like to reset HEAD back to the second parent, in this case e3427, which is after we reverted BFG, but before we added the keynote.



```
~/git and the bad day  master  8m
▶
```

Let me reset back to our previous state, after the merge, to demonstrate something else.

Suppose that the keynote exists in previous commit history as well, or that instead of the keynote I've committed credentials that have to be thoroughly scrubbed. In that case, a simple reset might not be sufficient, so we can try `git filter-branch`.

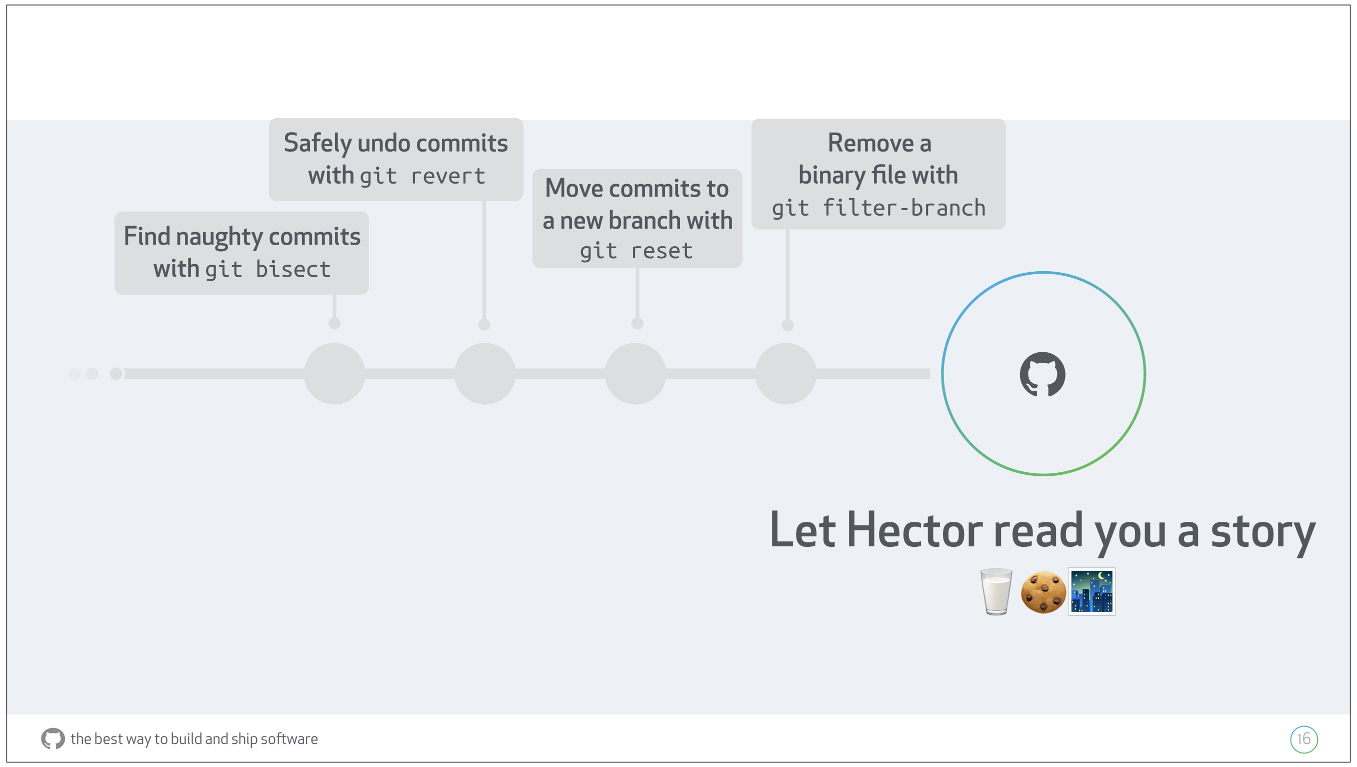
You command can run using a number of filters, we'll use the tree-filter which will rewrite commit trees and their contents to completely scrub that file.

We'll also indicate the command we'd like to run to modify every existing commit, which is to remove the keynote file in this case. And finally, we'd like this rewrite history saved on HEAD.

We can peek inside the git directory here, and also notice that the original history is saved within refs/original. We'll remove that under the assumption that it contains sensitive or unwanted artifacts.

And we'll also immediately prune our reflog and do a little garbage collection.

Now a couple of things to keep in mind are:
- if you are using a remote like github.com, it's possible that remote is caching some of your objects. For GitHub, in particular, you'll have to contact support to ensure that cache is cleared.
- even GitHub Support can't control what other folks have cloned, so like many other things on the internet, if it was once public it can very possibly still be retrieved
- A lot of folks have trouble with git filter-branch, so you can also check out Roberto Tyley's BFG repo cleaner, the one we reverted earlier, which can be easier and faster



Safely undo commits with `git revert`

Find naughty commits with `git bisect`

Move commits to a new branch with `git reset`

Remove a binary file with `git filter-branch`

Let Hector read you a story

🥛🍪🌃

Now that you've seen us get deep into some Git porcelain commands, I thought it'd be fun to take a tip from *Alexander and the Terrible, Horrible, No Good, Very Bad Day*, and put our little journey together into a story.

So grab your favorite blankie, pillow, milk, and cookies, and follow along with

Git and the Terrible, Horrible, No Good, Very Bad Day

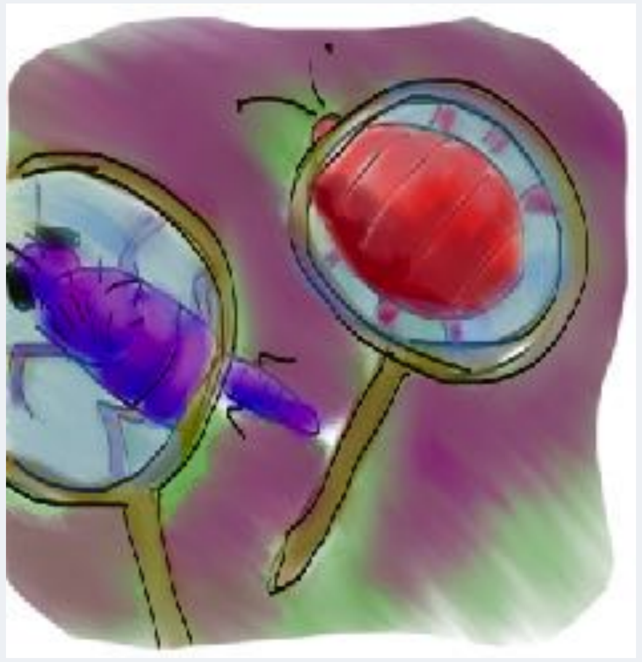
Illustrations by Nelson Torres

I did some late-night committing.

And I thought everything was fine.

And when I got out of bed, I spilled my morning coffee and by mistake I started a software update on my machine which bricked it for most of the morning.

When I finally logged on I realized I committed part of this story directly to master, which is horrible practice. So I could tell it was going to be a terrible, horrible, no good, very bad day.



So I decided to do some `git bisect` because on any other day it would probably help me find which commit broke my story. Besides, my friends Matt and Eric always found their bugs with `git bisect`.

My most recent commit was obviously bad.

And I don't even remember the last time I looked at my story, so I figured only the first commit was safe.

It took a couple of tries, but soon enough I found the commit that contained the offending line. But when I reverted the commit, I got a merge conflict!

When Matt used `bisect` he reverted
a missing semicolon,
a misspelled variable name,
and a pretty alarming lack of comments.

When Eric used `bisect` he reverted
some missing environment variables,
some badly indented blocks,
and part of his grocery list that got accidentally pasted into his code.

When I used `bisect`, I just got a gnarly merge conflict.

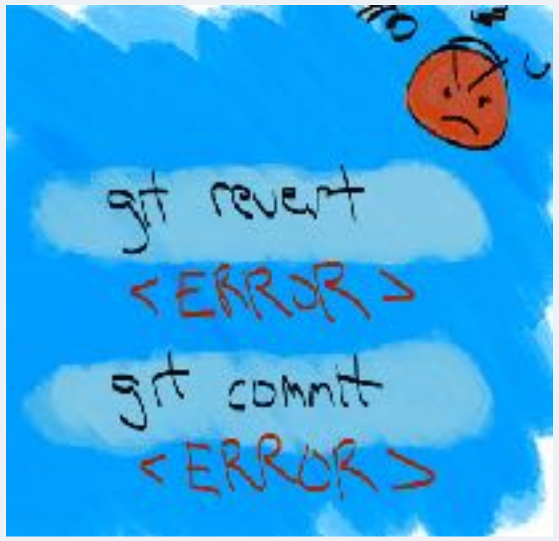
After breakfast, Briana got to teach her favorite group of students.

Cynthia spent some time watching trains with her son.

I was stuck trying to figure out how to resolve my merge conflict.

I could tell it was going to be a terrible, horrible, no good, very bad day.

Before lunch, I treated the conflict as any other conflict. I opened the file, and removed the conflict markers. Then I wondered: how do I finish this revert?



`git revert` didn't work

`git commit` didn't work

I wish I could revert this day.

Once the revert was finished, I realized I should've done it in its own branch. I wanted to `git reset` this terrible, horrible, no good, very bad day!

That's what it was, because that afternoon I actually tried `git reset`. My commit was made to the wrong branch, so I figured I'd create a new branch, then reset to the the previous commit, and when I switched to the new branch my revert was there!

I then realized a `git cherry-pick` would've done the same thing for me.

I wished I could cherry-pick myself out of this day.

Later that night, I realized I uploaded my Keynote presentation binary to the repo. What a waste of space!

I figured I could just revert the commit, but that just means that any previous commits would still have the file. This was a terrible, horrible, no good, very bad day.

Because each commit is a complete snapshot of the codebase, and I had no idea when I committed the binary, I knew this would be a messy clean-up. But `git filter-branch` should help me get out of this mess.

But before bed I was successful with `git filter-branch` I was happy with myself, but then I realized a local backup of what I cleaned up was still in my local git directory. So I had to clean that up, too.

I pushed my changes up to GitHub, and my push was rejected. It's because I altered commit history so I had to force push. Cynthia then told me that GitHub caches commits that may still have my credentials. So I had to contact Support and they removed the cached commits from GitHub.

This was, indeed, a terrible, horrible, no good, very bad day.

My friends say that some days are like that. Even without Git! But Git did indeed stick with me through every sticky situation, and its integrity was never compromised so, hey, Git helped me navigate my terrible, horrible, no good, very bad day.

Shoutouts to:

- Katie Sylor-Miller, ohshitgit.com
- @lorenallensmith, GitHub

@hectorsector
Trainer, GitHub