Git FAQ

Frequently asked questions around Git and Version Control.

Using `git bisect` to Quickly Find Bugs

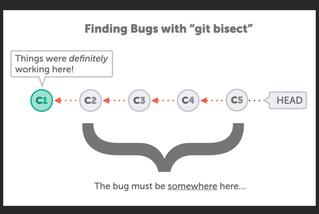
We've all been there: code that used to work like a charm suddenly doesn't anymore! **A bug has been introduced - but where and when exactly?** Especially in larger teams or after a larger series of new commits has been integrated, finding that nasty can be quite challenging.

Git offers a tool that can help make this "bug hunt" quicker and easier: "git bisect".

How `git bisect` Works

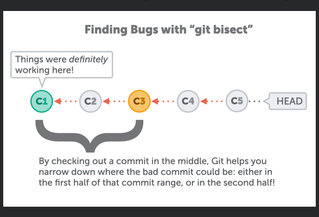
Git Bisect requires just two pieces information before it can start the bug hunt with you:

- A revision where things were *definitely* good.
- A revision where the bug is present.



Once you provide these two commit hashes, Git understands the bug must be somewhere between the "bad" and "good" commits. The way `git bisect` operates is fairly simple, but also quite effective: it splits the range of commits between "good" and "bad" in half - and checks out a commit in the middle.

The ball is now in our court: we have to run the application and test if it's working or if the bug is still present in this version. After we tell Git the result of our test, it automatically knows if the bug is contained in the first or in the second half of that commit range. Git will simply repeat this process until we've singled out the commit that contains the bug.



Git Bisect in Practice

Let's look at how `git bisect` works in practice. First of all, we have to explicitly start the process:

```
$ git bisect start
```

Now, Git is waiting for us to provide both a "bad" and a "good" commit. The bad commit is often easy because, in most cases, the *current* state is buggy. This means we can simply provide "HEAD" as the bad commit:

```
$ git bisect bad HEAD
```

Providing the "good" commit might be a tiny bit more work. But it's probably most effective to start by checking out an older revision where you are *quite confident* that everything was still fine. As soon as you've tested and verified this, we can go on:

```
$ git bisect good fc061994
```

With those two pieces of information provided, the actual "bisecting" process can start. Git will now check out a revision in the middle of this commit range between "good" and "bad":

```
Bisecting: 3 revisions left to test after this (roughly 1 step)
[3923c6ddf42d91fbd7e2d6a279c1f3bfcfba651b] Changing page structure
```

It's now on us to run / build our application and verify if the bug is still present or not. As soon as we've tested this, we need to tell Git the result, either with `git bisect bad` or `git bisect good`.

Git will use this information and repeat the process: it now splits the original commit range again - but depending on our answer, it will take either the first or the second half. And again, it will check out a revision in the middle and invite us to do some testing.

This process is repeated until we've successfully singled out the bad commit!

Once we've found the culprit, we can end the bisect process and wrap up:

```
$ git bisect reset
```

Git will then finish bisecting and take us back to our previous HEAD revision.

Learn More

- More frequently asked questions about Git & version control

About Us

As the makers of Tower, the best Git client for Mac and Windows, we help over 100,000 users in companies like Apple, Google, Amazon, Twitter, and Ebay get the most out of Git.

Just like with Tower, our mission with this platform is to help people become better professionals.

That's why we provide our guides, videos, and cheat sheets (about version control with Git and lots of other topics) for free.