How To Find Bugs Using Git Bisect With This Easy Guide

A beginners guide to the world of git bisect

A source control system is an essential tool if you're writing software, even for your own side projects. Although I've used many version control software over the past 20+ years of development, I've only been using `git` for the past 3 or 4 years. However it's quickly become my version control of choice due to its ability to quickly create branches, and its distributed nature. It's easy to master the basics of `git` but, as with most pieces of software, there's a lot of power available if you choose to go beyond them.

One `git` command that you might not have seen or used before is `git bisect`, which is something I've only recently started using too. It's a great tool for quickly tracking down bugs in your project so let's take a look at how we can use it.

A wild bug appears

I'm currently working on some high-traffic Node.js projects for work and, as I normally do at the start of a new task, I ran `git pull` to ensure I had the latest code from our master branch, and then ran `npm install` to update any dependencies. I ran the tests to check everything was ok but an odd thing happened: our test suite hung when it finished although all of the tests still passed. **WHAT?** Time to investigate.

I had no idea why this bug had crept into the project, nor when it first appeared in the master branch. My initial thought was that it was the last pull request that had been merged, so I checked out the previous merge instead but again, the tests were still hanging. This felt like a perfect opportunity to try out `git bisect` for the first time in order to quickly work out which commit had caused the error.

What does the "git bisect" command do?

Bisect performs a binary search (https://en.wikipedia.org/wiki/Binary_search_algorithm) to quickly find out the revision that caused a bug in your project's history. You do this by telling it which was the "bad" commit, the one which contains the bug, and which was a "good" commit, one in which you think the bug wasn't there. You can then use `git bisect` to pick a commit which is halfway between the two endpoints and you determine whether that one is bad or good. This can be repeated by choosing the revision between these two points until you determine the specific commit which caused the bug. It's then up to you to look at the code to see what has changed and is probably the cause.

The "git bisect" process

The first thing we need to do is tell `git` that we're about to start using bisect by using the `git bisect start` command. Before you do this, you should make sure you've committed or stashed any local changes to avoid losing any work as it will checkout each revision as we attempt to find the version containing the bug.

```
git bisect start
```

The next step is to determine when the bug first occurred and when we think that the repository was ok, and tell `git bisect` which commit versions these were. This allows it to determine the range of revisions to search through.

```
# Use this if the bug occurs in the current HEAD revision
git bisect bad HEAD

# But if the bug occurs in a specific revision then use its SHA hash
git bisect bad 62c5fa0
```

For the good revision, you can choose an arbitrary place in the past but the more commits that have happened between the two endpoints, the more potential revisions `git bisect` will have to check. This will depend on the speed of development in your project but it's probably wise to not choose a revision in time that was not too far away.

```
# Choose your good revision
git bisect good da5e24d
```

Now that you've chosen your two endpoints, `git bisect` will choose a revision that's in the middle of these two and tell you how many more revisions it will need to check from this point.

```
> git bisect good da5e24d
Bisecting: 54 revisions left to test after this (roughly 6 steps)
[dac781864e62c49b279c122c42d447ed26ad16e2] Merge pull request #100 from features/my-feature
```

It's now a process of elimination to determine which specific commit caused the bug. You do this by running your test suite or using your project to see if the bug is still present in the current revision. At this point `git bisect` wants to know if this commit is good or bad so you answer `git bisect good` or `git bisect bad.`

```
> git bisect bad
Bisecting: 27 revisions left to test after this (roughly 5 steps)
[4f455b6e42487657e0492305f025dd82021735d5] some commit message from this revision
```

In my case, the `gulp` task was still hanging so I answered that the commit was bad. The revisions are then split into two again and you need to continue to answer whether the revisions are good or bad until you reach the last revision and find the guilty commit. At this point `git bisect` will show you which commit caused the problem and you can check the code in this revision to determine what has changed.

```
02ce44a7491e9b0151169647115f9a073513e0ce is the first bad commit
commit 02ce44a7491e9b0151169647115f9a073513e0ce
Author: some-author <some-author@gmail.com>
Date:   Mon Sep 12 13:59:20 2016 +0100

    Added some code that might break! :D
```

Once you've determined the cause, you'll want to finish your current `git bisect` session. For this you just use the `reset` command and it will reset to your **HEAD** to where you were before you started.

```
git bisect reset
```

Make sure you reset otherwise you will find your local working directory in a strange state, having checked out out a previous revision of the project.

Pro-tip: Bisect automation

Manually finding the bugs in your code is helped by using `git bisect` but you can make things even faster by automating the process. You simply have to pass a script to be executed by `git bisect run`. Doing this will run the script against each bisected commit in order to determine which one causes the script to fail.

```
# Set up our start and endpoints to check against
git bisect start
git bisect bad 62c5fa0
git bisect good da5e24d

# Run our gulp task which executes our
# test suite for each bisected commit
git bisect run gulp test
```

You'll have to ensure the script that you're using fails with a non-zero return value for this to work. For the code that fails in my above discussion, our test suite hung so it wasn't as easy to use automation here.

Conclusion: Use "git bisect"

As you can see, using `git bisect` is an easy way to quickly determine where defects have crept into a `git` project. You should start using it as its an effective tool to understand. For more information have a read of this great git SCM article on debugging with git (https://git-scm.com/book/en/v2/Git-Tools-Debugging-with-Git). Also, take some time to look at the git bisect documentation (https://git-scm.com/docs/git-bisect) too.

5 Comments **Marc Littlemore** 1 **Login**

Recommend Share Sort by Best

Join the discussion…

Syed Jafri • 6 months ago
You could use timeout https://www.gnu.org/softwar...
• Reply • Share ›

Marc Littlemore Mod → Syed Jafri • 6 months ago
Use mean use timeout with "git bisect run <script>" Syed?
Interesting. From the docs, it looks like you could maybe use "timeout --preserve-status git bisect run test_script"?

Thanks, I didn't know about that command.
1 • Reply • Share ›

Syed Jafri → Marc Littlemore • 6 months ago
I was actually thinking of git bisect run timeout DURATION --preserve-status COMMAND.

If you do the command you mentioned then it would wait until git bisect times out so git bisect would still lock up on whatever you reach that commit that makes gulp test hang but then you would kill git bisect after the timeout is reached.
• Reply • Share ›

Marc Littlemore Mod → Syed Jafri • 6 months ago
Ah yes! You're right. That's a great idea. Thanks for the tip.
That would have been perfect in my case.

Looks like I might need gtimeout on a Mac but same idea applies. Cheers!
• Reply • Share ›

Syed Jafri → Marc Littlemore • 6 months ago
No problem!
• Reply • Share ›

Subscribe Add Disqus to your site Add Disqus Add Privacy

NEXT POST → (/SPOOKY-HALLOWEEN-SOUND-EFFECTS-WITH-RASPBERRY-PI-AND-SPOTIFY/)

● (/feed.xml) ● (http://twitter.com/marclittlemore) ● (https://github.com/MarcL)
● (https://www.instagram.com/marclittlemore) ● (https://www.reddit.com/user/marclittlemore)
● (http://soundcloud.com/djcruze) ● (https://pinterest.com/marclittlemore/)
● (https://plus.google.com/+MarcLittlemore/)