Check out our blog: Bringing Change to the Car Repair Space through an Innovative Platform (https://www.netguru.co/blog/bringing-change-to-the-car-repair-space-through-an-innovative-platform)

(https://twitter.com/netguru) (https://www.facebook.com/netguru) (https://www.linkedin.com/company/netguru) (https://github.com/netguru) (https://dribbble.com/netguru) (https://www.behance.net/netguru)

All (/codestories) Ruby on Rails (/codestories/topic/ruby-on-rails) Node JS (/codestories/topic/nodejs) Android (/codestories/topic/android) iOS (/codestories/topic/ios) React Native (/codestories/topic/react-native) Frontend (/codestories/topic/frontend)

Sneaky Bugs and How to Find Them (with git bisect)

Wiktor Czajkowski
Jan 8, 2018 | 8 min read
App Development (/codestories/topic/app-development)
Git (/codestories/topic/git)
Version control system (/codestories/topic/version-control-system)

TL;DR

Git bisect is immensely useful tool that allows us to fully automate binary searching through the commit history.

Usage:

```
$ git bisect start head head~4
$ git bisect bad
$ git bisect good
<some-long-sha> is the first bad commit
$ git bisect reset
```

Fully automate the search using:

```
$ git bisect start head head~4
$ git bisect run test.sh
```

Use bad/good, new/old or custom terms with:

```
git bisect --term-new newer --term-old older
```

These pesky bugs

You happily code a shiny new feature, when suddenly a wild bug appears. After quick debugging you discover it has been there for quite a while. You stare blindly at the log, unable to figure out which commit is responsible. In the last jump of faith, you start to go through the log commit by commit with the fury written on your face.

What if I told you, that you might use an algorithm for that? Also, what if there was another, faster, yet still simple algorithm? What if you had the tool to do it semi-automatically? What if you could actually fully automate that semi-automation and just SEE YOUR BUG GETTING DISCOVERED?

It's all possible. Let's do some computer science first.

The search begins

The simplest approach to searching through stuff is just compare each element to the one we're looking for until we find it or run out of elements. This is the approach we might take from within the desperation well of being behind the already-postponed deadline, for example. Assuming we will find the bug the first time we check a particular commit (which might not be the case in such traumatic circumstances), we can see that the worst-case (https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/) time complexity (https://en.wikipedia.org/wiki/Time_complexity) of that approach is O(n), where n is the number of commits. That is because if the bug lied in the very last commit, we would need to look through all of them.

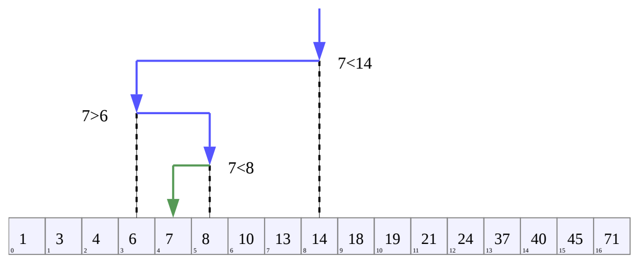
Binary search

O(n) isn't bad, but we can do better. Incidentally, there is an algorithm which is only slightly more complex, but gives us much better time complexity of O(log n). That's like, exponentially better!

There is one caveat though - the elements that we search through have to be sorted. *russian accent on* Do not worry though, my friend *russian accent off*, since our commits are sorted exactly how we like 'em - chronologically. (I was thinking about some pun really heavily here, I swear.)

The algorithm works by comparing the middle element of the array to the target value, then going recursively (https://en.wikipedia.org/wiki/Recursion_(computer_science))

into the part that the element must be in, until, again, the element is found or there is no elements to look through. It is nicely exemplified by the following picture from the Wikipedia article on binary search (https://en.wikipedia.org/wiki/Binary_search_algorithm):



Let's see how to apply that idea to our commits!

Through the commits

Now, the process gets a bit more complicated. First step is to find a commit that we know works, i.e. the one somewhere before the bug was introduced. Depending on your situation you might want to look for the last commit on an upstream branch, the last commit deployed to production, etc. You don't need to care much about the number, because our search works in a logarithmic time, so it will work quickly even for big numbers. Let's say we arbitrarily picked the commit nine (for simplicity's sake) commits ago and it worked.

```
$ git checkout head~9
```

Now we look at the commit that's right in the middle:

```
# go back
$ git checkout -

# 10 / 2 = 5
$ git checkout head~5
```

Let's say it has the bug. Now we know that every commit after that will also have it, so we can safely ignore them and look before this one, again in the middle:

```
# 5 / 2 = 2
$ git show head~2
```

Let's say that now the bug is gone. Now we know it won't be present in any commit before the current one. That means we need to look a single commit ahead in the history:

```
# go back to `git checkout head~5`
$ git checkout -

$ git checkout head~1
```

Now we're 6 commits away from our initial `HEAD`. If this one is clean, we know it's `HEAD~5`, if not - this is the one.

Phew! That was something! And it took us just 3 steps, which incidentally is roughly what $\log_2(10)$ equals to!

Now guess what - Linus Torvalds thought you might find it useful (https://github.com/git/git/commit/8b3a1e056f2107deedfdada86046971c9ad7bb87) and made it a built-in git command!

git bisect

Now that we know what binary search is, `git bisect` (https://git-scm.com/docs/git-bisect) is really simple - it semi(for now)-automates binary searching through our commits. Let's see:

Let's start the process using:

```
$ git bisect start
```

Now we tell git which commit is the working one:

```
$ git bisect good head~9
```

We also tell it that the bug is present in the HEAD:

```
$ git bisect bad head
```

Since HEAD is the default value, this is an equivalent:

```
$ git bisect bad
```

We can also mark the bad and the good (but not the ugly 😕) commits at once (in this order) using:

```
$ git bisect head head~9
```

Now comes the sweet part - git will automatically checkout commits in a similar fashion to the one we did in previous section of the article. Our role is to mark them by either good or bad:

```
$ git bisect head head~9
Bisecting: 4 revisions left to test after this (roughly 2 steps)
[2d550db5bdebcccd03f02b15c41d1c0b3c4b31fa] Commit 5 from HEAD

$ git bisect bad
Bisecting: 2 revisions left to test after this (roughly 1 step)
[f239cb8b6717c7cb21db9823628488291a982dc6] Commit 7 from HEAD

$ git bisect good
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[30947a352127e4e4359fea5407227ffcd8a2c18c] Commit 6 from HEAD

$ git bisect bad
30947a352127e4e4359fea5407227ffcd8a2c18c is the first bad commit
commit 30947a352127e4e4359fea5407227ffcd8a2c18c
Author: Wiktor Czajkowski <wiktor.czajkowski@gmail.com>
Date:   Wed Jan 3 18:09:46 2018 +0100

    Commit 6 from HEAD

:000000 100644 0000000000000000000000000000000000000000 e69de29bb2d
```

Join community of tech minded people like you

Folks from our dev team work on best projects, tools and tech stacks. And they share their experience and knowledge here. No edits, no PR, no bullshit - just cool codestories🙌.

So why not leave an email and get frequent updates from them? 😎

You can subscribe here:

name@example.com

Netguru

Don't forget to go back to the place you started from, or weird things will happen:

```
git bisect reset
```

That's it! Wasn't it amazingly wonderfully miraculous? No? You're right. But it was still pretty good I'd say.

Automate all the things!

Now comes the sweet sweet automation part. If you can check for existence of a bug programatically, you can let git bisect do all the work while you sip your now-cold-from-being-immersed-in-this-article-and-forgetting-to-drink-it coffee. By programatically I mean by running a shell command, of course. It might be a test suite:

```
# start the bisect, then...

$ git bisect run yarn test
running yarn test
```

or a script:

```
# start the bisect, then...

$ git bisect run test-for-the-pesky-bug.sh
running test-for-the-pesky-bug.sh
```

Now this is miraculous! It is, isn't it? Still no? Well, ok, but still, I deeply regret not knowing it for so long. Don't make my mistake!

Alternative terms

Now that you use git bisect all the time (or not, because your software is bug-free, idk), you might find yourself in a situation where you're not looking for a bug, but for some other change, like a performance improvement. In such case the good and bad terms might be confusing. Luckily, git provides alternative terms - `old` instead of `good` and `new` instead of `bad`:

```
$ git bisect start
$ git bisect new
$ git bisect old head~99999
```

And that's not all! You can also play on your own terms (kek), using:

```
$ git bisect start --term-new funny --term-bad not-funny
$ git bisect funny
$ git bisect not-funny head~42
```

Summary

Git bisect is immensely useful tool that allows to fully automate binary searching through the commit history.

Git logo by Jason Long. Hat icon by Alexey Voropaev from the Noun Project.

(https://www.netguru.co/about-us/team/wiktor-4400739680)

Wiktor Czajkowski (https://www.netguru.co/about-us/team/wiktor-4400739680)
Frontend Developer

READ ALSO FROM APP DEVELOPMENT

Read also

Few Tips That Will Make Your PWA on iOS Feel Like Native (https://www.netguru.co/codestories/few-tips-that-will-make-your-pwa-on-ios-feel-like-native)

How to Start with Kotlin/Native? (https://www.netguru.co/codestories/how-to-start-with-kotlin/native)

SOLID Principles #3: Liskov Substitution Principle (https://www.netguru.co/codestories/solid-principles-3-lsp)

Need a successful project?

Estimate project (/estimate-project?ref=successful-project-footer-cta)

or contact us (/contact)

(https://twitter.com/netguru)

f (https://facebook.com/netguru)

in (https://www.linkedin.com/company/netguru)

(https://www.github.com/netguru)

(https://www.dribbble.com/netguru)

Bē (https://www.behance.net/netguru)

netguru

hello@netguru.co (mailto:hello@netguru.co)
+44 20 3871 3389 (tel:+442038713389)

VAT-ID: PL7781454968
REGON: 300826280
KRS: 0000306593

MENU

About us (https://www.netguru.co/about-us)

SERVICES

Ruby on Rails (/services/rubyonrails)
Node.js (/services/nodejs)

AWARDS

Deloitte. Technology Fast50 (https://www.netguru.co/blog/netguru-deloitte-fast-50)

Join community of tech minded people like you

Folks from our dev team work on best projects, tools and tech stacks. And they share their experience and knowledge here. No edits, no PR, no bullshit - just cool codestories🐰.

So why not leave an email and get frequent updates from them? 😎

You can subscribe here:

name@example.com

Netguru

+48 534 205 209
(tel:+48534205209)
ul. Wojskowa 6
60-792 Poznań,
Poland

Team
(https://www.netguru.co/about-us/team)
How to join
(https://www.netguru.co/blog/how-to-join-netguru)
Clients
(https://www.netguru.co/clients)
Blog
(https://www.netguru.co/blog)
Contact
(https://www.netguru.co/contact)
Terms of use
(https://www.netguru.co/tos)

React.js
(/services/react-js)
iOS (/services/ios-mobile-development)
Android
(/services/android-mobile-development)
React Native
(/services/react-native)
Ember.js
(/services/ember-js)
Progressive Web Apps
(/services/progressive-web-apps)
Vue.js
(/services/vue-js)

Forbes
Inc. 5000

(https://www.netguru.co/location/helsinki)
(https://www.netguru.co/location/dublin)
(https://www.netguru.co/location/london)
(https://www.netguru.co/location/berlin)
(https://www.netguru.co/location/usa)
(https://www.netguru.co/location/caribbean)

Join community of tech minded people like you

Folks from our dev team work on best projects, tools and tech stacks. And they share their experience and knowledge here. No edits, no PR, no bullshit - just cool codestories👌.

So why not leave an email and get frequent updates from them? 😎

You can subscribe here:

name@example.com

Netguru