Bisecting

coursera.org/learn/git-distributed-development/supplement/F6hzS/bisecting

Suppose you had a version of your code that worked, and now you are many versions (and commits) later and you have found out that it is no longer working.

git has the ability to bisect in order to rapidly find the change set that screwed things up. The number of steps is no more than the logarithm to the base 2 of the number of commits, which is much faster than a brute force approach. In other words, if a bad change has been done somewhere in the last 1024 commits, you can find it in no more than 10 bisection steps.

You do this by first typing:

```
$ git bisect start

$ git bisect bad

$ git bisect good V_10
```

where it is assumed that the current commit is bad and version V_10 is known to be good. git will then leave you at a commit halfway in between. You then test the code to see if the bug is still there. If it is, you type:

```
$ git bisect bad
```

If the code does not have the bug yet, you type:

```
$ git bisect good
```

You continue this iteratively until you find the bug. Then you type:

```
$ git bisect reset
```

to get back to your current working state.

For a working example, lets try the Linux kernel repository:

```
$ git bisect start
```

$ git bisect bad

$ git bisect good v2.6.30

Bisecting: 16539 revisions left to test after this

[b4f3fda5d475931d596d5cf599a193f42b857594] Staging: hv: coding style cleanup of include/HvVpApi.h

$ git bisect bad

Bisecting: 8270 revisions left to test after this

[0de4adfb8c9674fa1572b0ff1371acc94b0be901] Blackfin: fix accidental reset in some boot modes

$ git bisect good

Bisecting: 4136 revisions left to test after this

[b9caaabb995c6ff103e2457b9a36930b9699de7c] Merge branch ’master’ of git://git.kernel.org/pub/scm/linux/kernel/git/holtmann/bluetooth-next-2.6

.....

$ git bisect good

Bisecting: 60 revisions left to test after this

[5d48a1c20268871395299672dce5c1989c9c94e4] Staging: hv: check return value of device_register()

.....

/usr/src/GIT/work>git bisect bad

Bisecting: 3 revisions left to test after this

[b57a68dcd9030515763133f79c5a4a7c572e45d6] Staging: hv: blkvsc: fix up driver_data usage

$ git bisect good

```
Bisecting: 1 revisions left to test after this

[511bda8fe1607ab2e4b2f3b008b7cfbffc2720b1] Staging: hv: add the Hyper-V virtual

                                    network driver

$ git bisect good

Bisecting: 0 revisions left to test after this

[621d7fb7597e8cc2e24e6b0ca67118b452675d90] Staging: hv: netvsc: fix up

  driver_data usage

$ git bisect reset
```

If it is possible to construct a script that can test the current version to see if the bug is present, the process becomes even easier.

Suppose you have written a script, **my_script.sh**, that returns 0 if the current version is good, and any value between 1 and 127 if it is bad. Then, after initializing the bisection with a good and bad version, you can simply do:

```
$ git bisect run ./myscript.sh
```

and the process will terminate when it locates the bug.

You can replay the bisection history with **git bisect log** or **git bisect visualize**.

If you do small incremental changesets, bugs can be found very quickly with bisection. Commits which have many changes will mean quite a few places may have to be examined even after you identify the last working version and the first faulty one.