Bisecting: The Session at a High Level

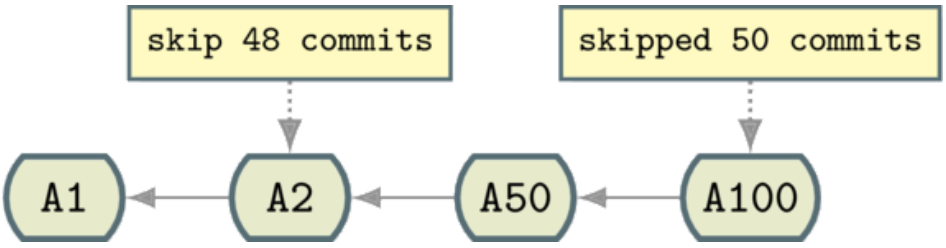
Learn about "bisection" concepts at a higher level.

We'll cover the following

First, I'll explain what a `git bisect` session might look like in the abstract before running through it on the command line in detail.

Scenario

Let's say you have a set of 100 commits on a `master` branch:



Git repository with 100 commits

Not All Commits Shown

To make the graphs legible, I have not shown all 100 commits in a line. Where items have been skipped, I've put a pointer indicating that they are being skipped or were skipped. For example, in the above diagram, 48 commits were skipped between A2 and A50, and 50 commits were skipped between A50 and A100.

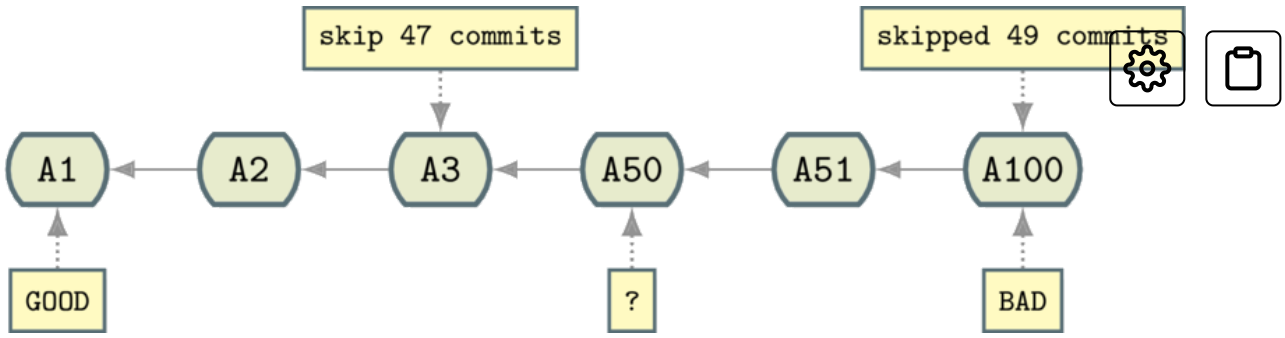
You discover a previously-unseen bug at point `A100` and want to debug it. One way to do this is to read over the code, add logging, etc. This can be time-consuming, and there is a simpler way to gather information about what change caused the bug.

The `git bisect` command

The `git bisect` command is a useful tool for finding out where a bug was introduced. If you know where a bug was introduced, you can look at the diff of the commit that caused it and work out what the source of the problem is.

How does it works?

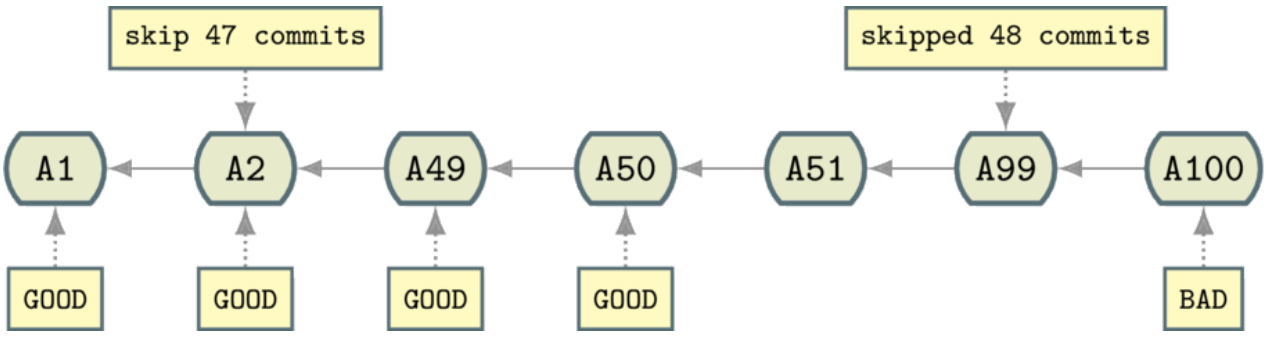
It works by picking a starting point where the bug definitely did not exist (the "good" point). In this case, you'll choose point `A1` . Then you pick a point where the bug definitely did exist (the "bad" point). In this case, that's `A100` .



Rebase - A1 good, A100 bad, A50...?

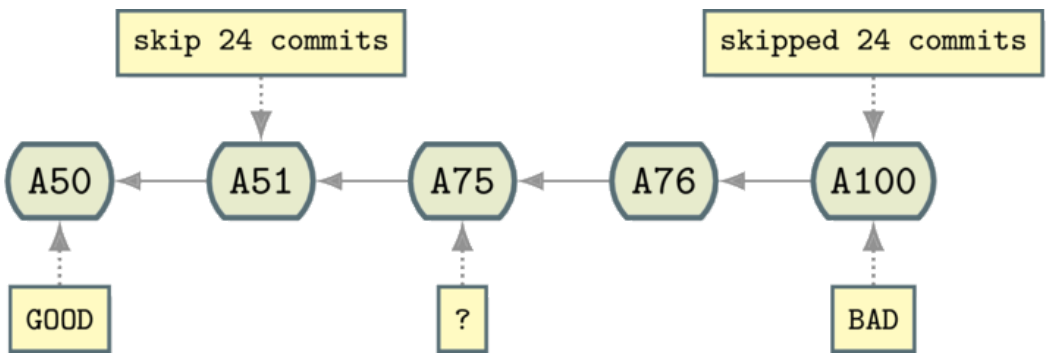
Once the Git bisect session has that information, it can hand you a version at the halfway point between the “good” and “bad” points and ask you to run whatever you need to run to determine whether it’s good or bad. At this point, the halfway point is `A50`.

If you tell it it’s good it will mark all versions at that point and before as good.



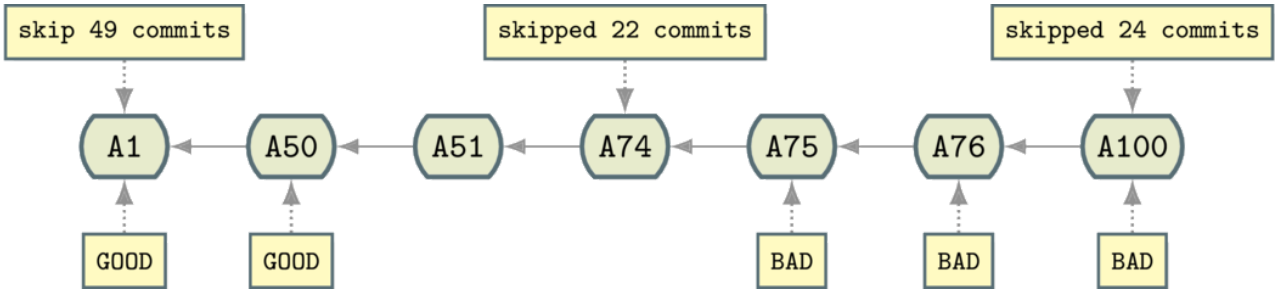
Rebase A1-A50 marked as good

It then repeats the process, giving you a version at the halfway point between good and bad, asking you for its status. In this sequence, you are given `A75`:



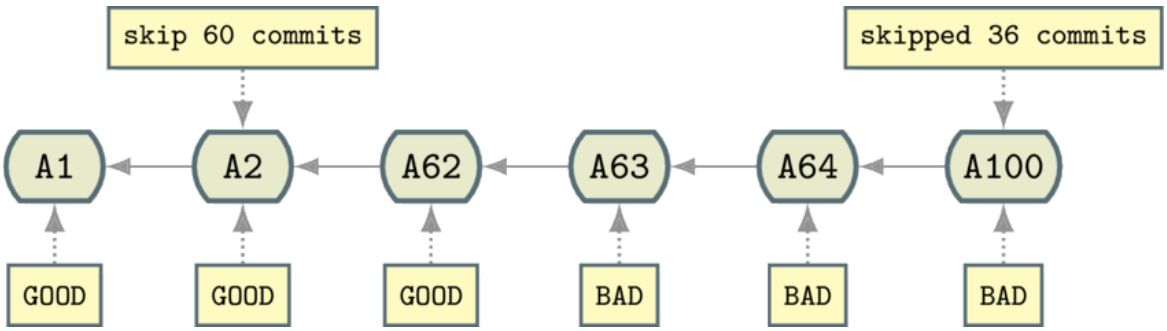
Rebase - A1-A50 marked as good, A75...?

If you determine that this version was ‘bad’, then all the versions after it are marked as bad:



Rebase - A75-A100 marked as bad

This binary search process repeats until you know which versions were good and bad. One outcome might be:



Rebase - A1-A62 good, A63-A100 bad

Binary Search?

Binary search is a way of searching an ordered list of items in an algorithmic way with the fewest number of ‘moves’. If you are looking for a book by an author (Ian Miell) in a library that’s on a single and very long shelf that’s ordered by the author name, you might start at the middle. If the middle book is by an author before Miell you would move halfway towards the remainder of the shelf and then likely back half as far again, and so on until you get to the book.

Once you know that the first bad commit was `A63`, you can examine the difference between `A62` and `A63`, and this gives you a clue as to what caused the bug.

Back

Introduction: Git Bisect

Next

A Real 'git bisect' Session

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/bisecting-the-session-at-a-high-level__git-bisect__learn-git-the-hard-way)