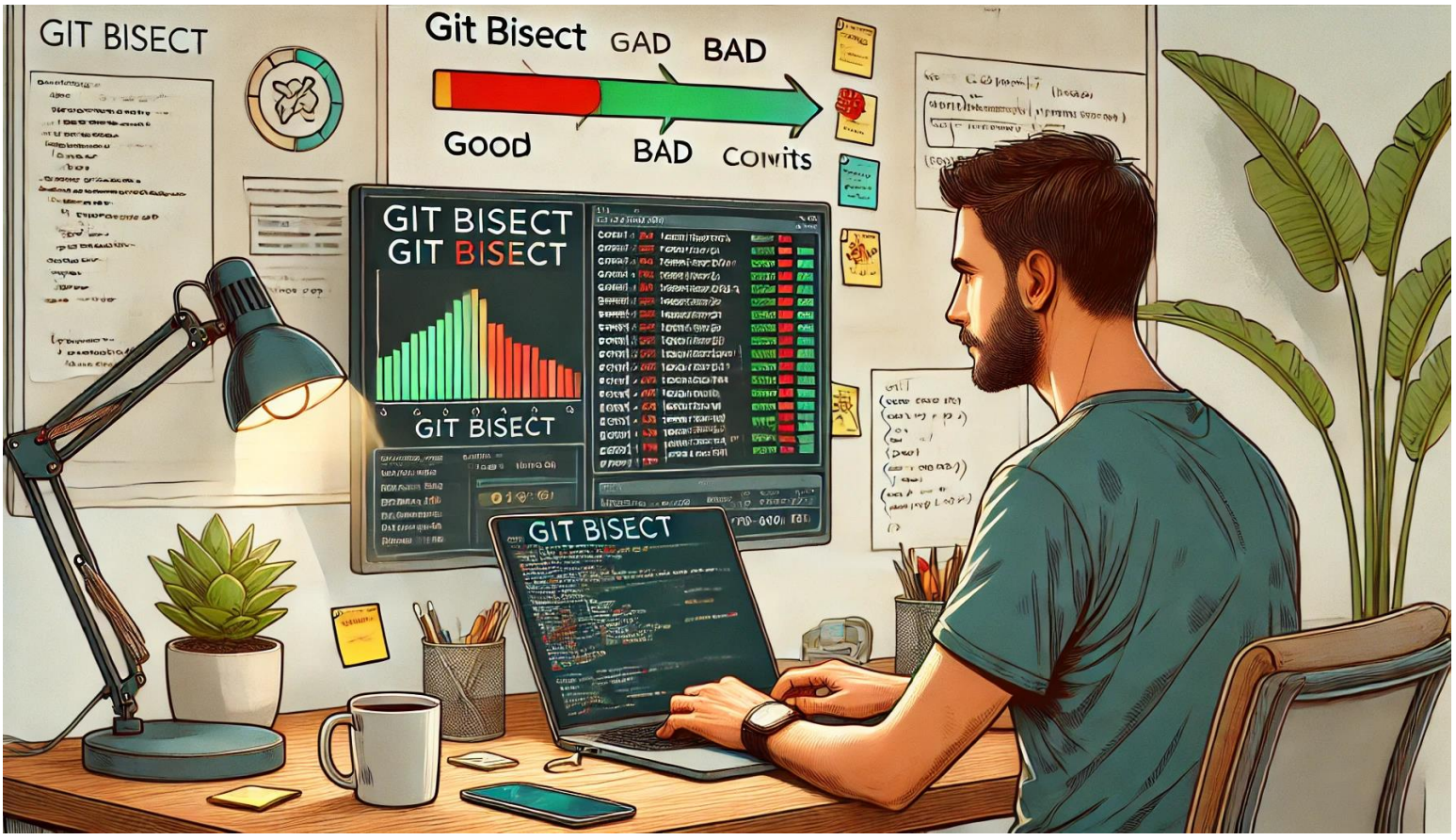
DevOps Shack

GIT BISECT : Finding Commit That Introduced Bug

[Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps]



Purpose

`git bisect` is used to find the commit that introduced a bug in your codebase. It uses a binary search algorithm to quickly narrow down the problematic commit by testing a series of commits between a known good state and a known bad state.

Example Scenario

Suppose you have a repository, and you noticed that a bug was introduced at some point in the commit history. You don't know which commit caused the bug, but you know it wasn't there in an earlier commit.

Steps to Use `git bisect`

1. **Start the Bisect Process**:

   ```
   git bisect start
   ```

2. **Mark the Current Commit as Bad**:

   ```
   git bisect bad
   ```

   This tells Git that the current commit contains the bug.

3. **Mark an Earlier Commit as Good**:

   ```
   git bisect good <commit>
   ```
   Replace `<commit>` with the hash of a commit you know doesn't have the bug.

Detailed Example

1. **Initialize the Repository**: Let's say your repository has the following commits:

2. `commit 6: "Fix typo in README"`
3. `commit 5: "Add new feature"`
4. `commit 4: "Refactor codebase"`
5. `commit 3: "Fix critical bug"`
6. `commit 2: "Initial setup"`
7. `commit 1: "Initial commit"`
8. **Identify the Bug**: You are currently on `commit 6` and notice the bug. You know the bug wasn't present in `commit 3`.

9. **Start Bisecting**:

   ```
   git bisect start
   ```

10. **Mark the Current Commit as Bad**:

    ```
    git bisect bad
    ```

11. **Mark the Known Good Commit**:

    ```
    git bisect good <commit-hash-of-commit-3>
    ```

12. **Git will Now Checkout a Commit in the Middle**: Git will check out a commit in the middle of the range. Suppose it checks out `commit 4`.
13. **Test for the Bug**: You test your code at `commit 4`:
    - If the bug is present, mark it as bad:

      ```
      git bisect bad
      ```

    - If the bug is not present, mark it as good:

      ```
      git bisect good
      ```

14. **Continue the Process**: Git will continue to check out commits, narrowing down the range until it identifies the exact commit that introduced the bug. Each time, you'll test for the bug and mark the commit as good or bad.

Final Step

Once Git has narrowed it down, it will tell you which commit introduced the bug. You can then review that commit to understand what changes caused the issue.

Example Output

Suppose the bug was introduced in `commit 5`. After marking `commit 4` as good and `commit 5` as bad, Git might output something like:

```
8f7f3c2 is the first bad commit
commit 8f7f3c2
Author: Example Author <author@example.com>
Date:   Sat Jun 23 12:00:00 2023 +0000

    Add new feature

:100644 100644 bcd1234... 56789ef... M  file.txt
```

This tells you that `commit 5` (with hash `8f7f3c2`) is the commit where the bug was introduced.

Conclusion

`git bisect` is a powerful tool that can save a lot of time when debugging issues in large projects with many commits. By systematically narrowing down the commits, you can quickly identify the problematic change.