Git reflog (and Git bisect)

Use case 1

- Have you ever committed something, tried to `git rebase`, and everything went horribly horribly wrong?
- Or accidentally `git reset --hard HEAD^` when you meant to `git reset HEAD^`?

Have no fear, **git reflog** is here!

✓ When you use git, commits are never lost

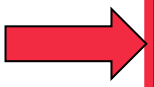
✓ You can always **get back to any repository state** as long as you committed your changes

– The only time commits are actually deleted is if you `git gc --prune` (so be careful with that one!)

Git reflog

- You can reset commits and alot of actions (change branches, merges, pulls, rebases)
- Use `git reflog` to get an overview of your last actions:

SHA1 commit pointer action message



```
$ git reflog
0db8285 HEAD@{0}: HEAD~5: updating HEAD
177762a HEAD@{1}: commit: change affiliate field names
bb5d920 HEAD@{2}: pull origin develop: Merge made by recursive.
f6fade5 HEAD@{3}: commit: sort the category brands by name on ...
9309873 HEAD@{4}: checkout: moving from feature/affiliate to
develop
92a80c2 HEAD@{5}: checkout: moving from develop to
feature/affiliate
```

```
$ git reset 177762a
```

or `git cherry-pick, merge ...`

Git reflog<subcommand> <options>

- `git reflog show` (default)
  shows the log of the reference provided in the command-line (or HEAD, by default)

- `git reflog expire`
  `[--expire=<time>]`
  `[--expire-unreachable=<time>]`
  prune older reflog entries older than `expire time` or `expire-unreachable time` && not reachable from the current tip

- `git reflog delete ref@{specifier}`
  deleting single entries from the reflog

git bisect

Use case 2

- Something broke and you found out it worked six commits ago (in the release tagged „v0.1.24“) by doing a quick `git checkout v0.1.24` and running your tests

- You don‘t know which commit introduced the bug, but you want to find out
  - what changed,
  - who did it,
  - if you can revert it quickly

Git bisect

- After `git bisect start`, you have to specify a „good“ commit and a „bad“ one

```
$ git bisect start
$ git bisect good v0.1.24
$ git bisect bad develop
Bisecting: 2 revisions left to test after this (roughly 1 step)
[5dec197fedabd9db02cc1621f5bbdb2e8defeb48] Merge branch hotfix/...
```

- You swich off your development branch and `git bisect` took you back 3 commits → you are in the middle between the „good“ and the „bad“ one

- Next, run your tests
- See, if the bug was in this commit already
- Let‘s say it was → mark this commit „bad“ as well

Git bisect

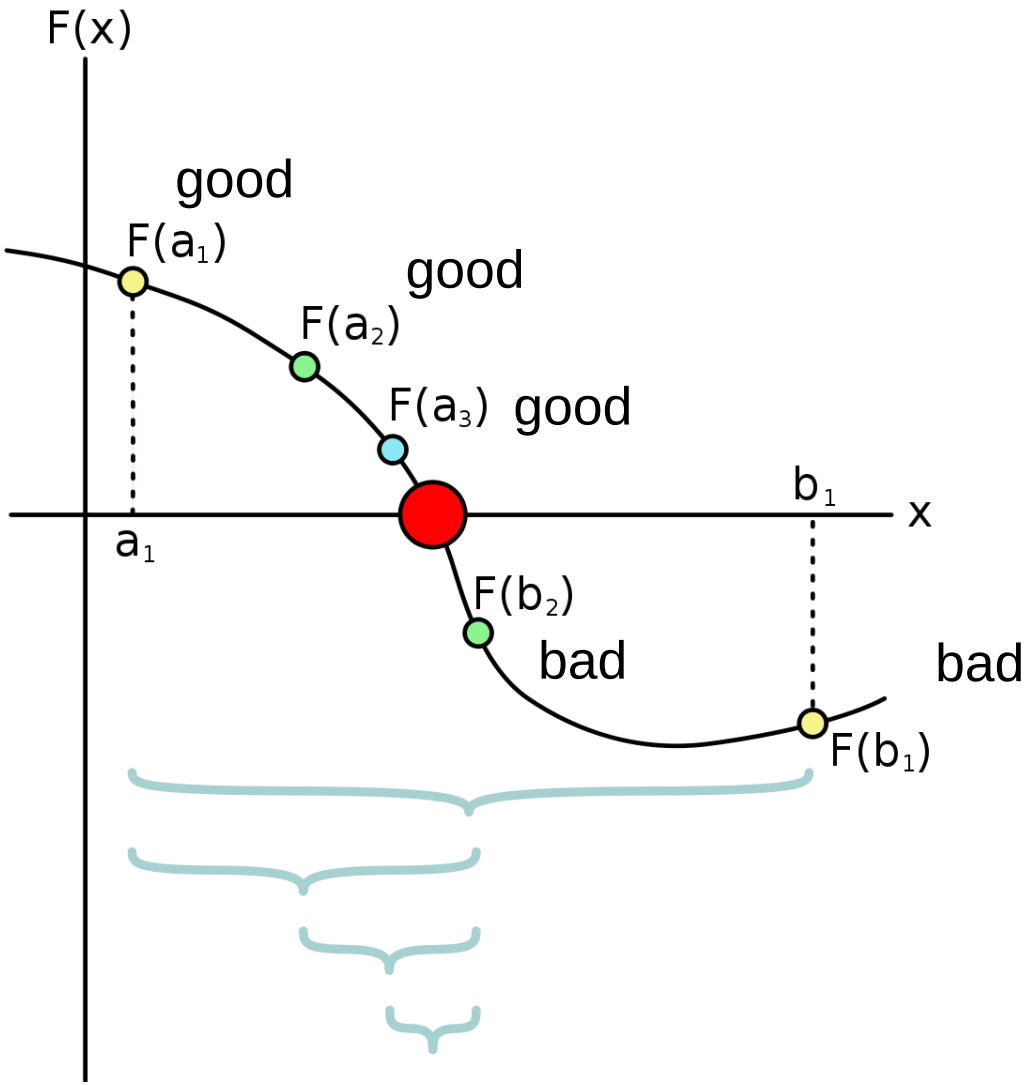
```
$ git bisect bad
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[9dea486b10f14475beb56e9d67c6dd45c8fab088] sort the category ...
```

- You're taken to the next commit to test → you find out it worked in this one, so you mark it "good" and git will tell you the commit that broke stuff:

```
$ git bisect good
154f34cd41619eaace63122480c8aa7180f7dbe6 is the first bad commit
commit 154f34cd41619eaace63122480c8aa7180f7dbe6
Author: Thijs Cadier
Date:   Sat Aug 28 19:20:40 2010 +0200
```

- Now we know which commit introduced the bug
  e.g. use `git show` to see what changed, do a `git revert` to fix it

Git bisect == binary search method



Thank you!

Bibliography

- http://www.kernel.org/pub/software/scm/git/docs/git-reflog.html
- http://gitfu.wordpress.com/2008/04/06/git-reflog-no-commits-left-behind/
- http://jeffkreeftmeijer.com/2010/the-mighty-reflog-and-the-amazing-bisect/
- http://de.gitready.com/intermediate/2009/02/09/reflog-your-safety-net.html
- http://en.wikipedia.org/wiki/Bisection_method