Collabora

Git crash course:

The collective noun for a group of programmers is a merge-conflict.

Miklós Vajna

2013-09-27

Agenda

- Motivation
  - Why version control?
  - Why distributed version control?
  - Why git?
- Git crash source
  - A bottom-up introduction
  - Contributing using git
- The LibreOffice perspective

Motivation

Why version control?

- Everyone uses version control:
  - Think of 'Save As'
  - Tarball + patches
- You can hardly avoid it if you collaborate
- Helps debugging
- Documentation tool

Why distributed version control?

- The full repo is available locally
  - Fast diff, blame, log, merge
- You don't have to be always online
- No SPoF
- Concept of committer may go away
- Backups are less important
- Easier branch / merge

Why git?

- #1 reason is of course: it's distributed
- But still, a few other unique features
  - git merge-recursive (e.g. cherry-pick handles renames)
  - git rerere
  - git blame – can detect the move of a code chunk
  - git grep
  - combined diff

Git crash course

A bottom-up introduction

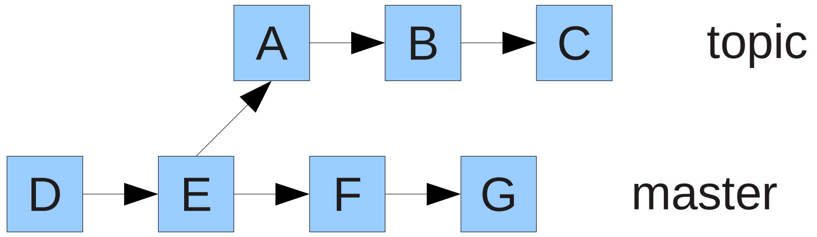
- Low-level: content-addressable filesystem
- 4 object type: blob, tree, commit, tag
- Blob: one version of a file
- Tree: contains at least one tree(s) or blob(s)
- Commit: contains 0..many parents and 1 tree
  - Also: message, date, name
- Tag: only used by annotated tags; can point to anything (usually points to a tag)

What is not an object

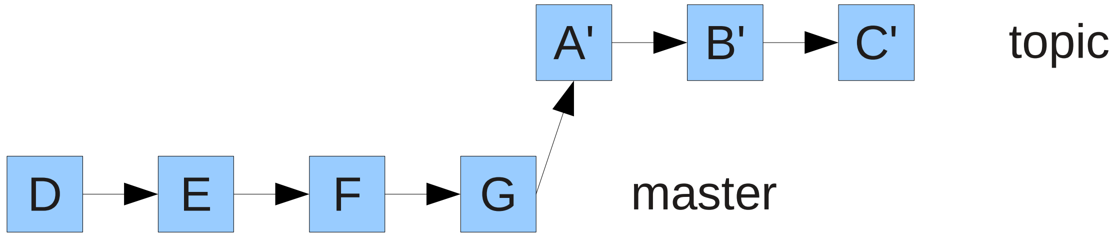
- Ref, symref
  - Non-annotated tags are refs, not tag objects
- Hook
- Reflog
- Config
- Index

Merge vs. rebase

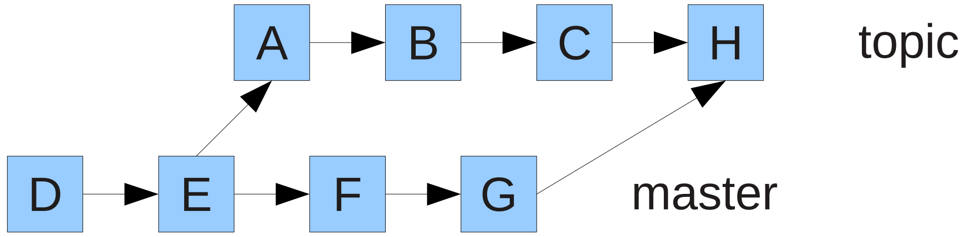
- At the beginning we have:



- Rebase:



- Merge:



Contributing using git

- By default, no push rights, sends patches
- Still works in git, locally
- Uses rebase, not merge
- Interactive rebase
  - Squash, split, reorder patches
- git format-patch, git am, git review
- Bundles: offline transfer or merges

Commands: too many

- Which ones do I need?
- Current git (1.8.1.4) has 161 commands
- Categories:
  - Main porcelain commands
  - Ancillary porcelain commands
  - Plumbing commands

Commands you *will* use

- init, clone, add, rm, mv
- status, branch, diff, log
- commit, reset (undo of commit and add)
- fetch, pull, push
- checkout, rebase, merge
- show, grep, bisect

Main porcelain commands

- archive, bundle, format-patch / am
- cherry-pick and revert
- describe, shortlog
- gc, clean, stash, submodule

Ancillary porcelain commands

- Manipulators: config, filter-branch
- Query: blame, fsck, verify-tag
- Talking to weird guys:
  - fast-import / fast-export
  - archimport, cvsimport/export
  - quiltimport, svn

Porcelain commands

- For scripts, this is the stable API of git
- Example: log vs. rev-list

```
$ git log --pretty=oneline HEAD~2..
3b3e7061d610fa83d15b0ba66aba08fd7e39e611 fdo#66743 fix
5abc99f2fc9db8aa4dbce293898e26561f947ece Show errors
$ git rev-list HEAD~2..
3b3e7061d610fa83d15b0ba66aba08fd7e39e611
5abc99f2fc9db8aa4dbce293898e26561f947ece
```

Symbolic names of commits

- Scary example:

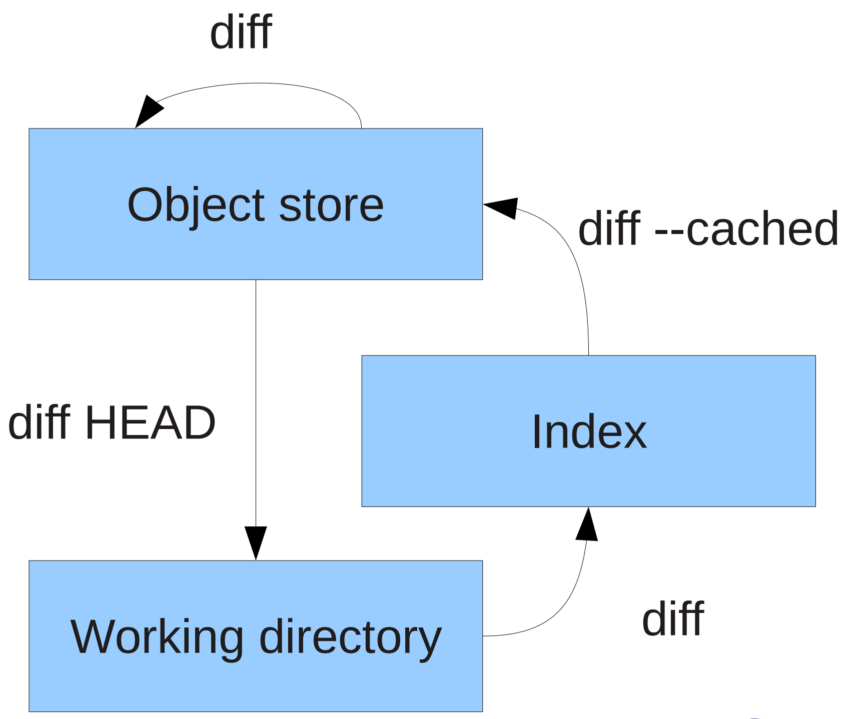
```
G   H   I   J  A =      = A^0
 \ /     \ /   B = A^   = A^1     = A~1
  D   E   F    C = A^2  = A^2
   \  |  / \   D = A^^  = A^1^1   = A~2
    \ | /   |  E = B^2  = A^^2
     \|/    |  F = B^3  = A^^3
      B     C  G = A^^^ = A^1^1^1 = A~3
       \   /
        \ /
         A

H = D^2  = B^^2    = A^^^2  = A~2^2
I = F^   = B^3^    = A^^3^
J = F^2  = B^3^2   = A^^3^2
```

- What to remember: ^ and ~N.

The Index

- Problem: two changes to the same file, we want to commit only one of them
- Or during conflict resolution: resolve conflicts one by one



The LibreOffice perspective

Submodules

- Submodule: a tree references a commit
  - When branches are matching, gerrit auto-commits in core
- In LibreOffice, disabled by default
- Needed by: dictionaries, help, translations
- Pain: have to commit them separately
- Gain: no need to download them by default

Gerrit: the manual way

- Gerrit gives virtual push rights to everyone:
  - git push origin HEAD:refs/for/master
  - Change-Id footer makes it explicit what is the same change
- Cherry-picking form gerrit:
  - git fetch origin refs/changes/12/6012/1
  - git cherry-pick FETCH_HEAD
- No hard dependency on external tools

Gerrit: helper tools

- git-review from OpenStack:
  - Auto-setup based on .gitreview file
  - git review: submits for review, prevents from accidental push of multiple commits
  - git review -x <int>: cherry-pick from gerrit
  - Packaged in most distributions
- logerrit: in-tree tool:
  - ./logerrit submit
  - ./logerrit cherry-pick <int>

Referenced clone

- Only interesting if you build release branches as well
  - git clone --reference /path/to/master <url>
- If you use submodules as well:
  - ./autogen.sh … --with-referenced-git=/path/to/master
- Saving is significant, .git of master / release branch is like: 1.2GB / 13MB

Interactive rebase

- In LibreOffice's case, this is especially useful when doing unit testing:
  1. Commit the fix
  2. Commit the testcase once it passes
  3. Revert the fix, make sure the testcase fails
  4. Interactive rebase:
     - Drop the revert
     - Squash the commit and the testcase into one commit

Bisect: binary search

- Bisect in general is extremely useful for our large code-base, but there is more
- Bibisect: to avoid bisecting for a full day
- Reverse bisect: when you are checking what commit to backport to a release branch
  - Swap bad and good in the git bisect start commandline
  - Also swap git bisect bad and git bisect good

Push tree

- Create a referenced clone, called master-push
- Instead of push, cherry-pick to master-push, and push from there
- Avoids expensive rebuilds in the middle of your productive hours
- The less frequiently you pull in your master tree, the less useful it is (more conflicts)
  - Still pull daily, weekly, etc. (depending on how fast your machine is)

Questions?

- Anyone?

Slides: http://vmiklos.hu/odp