Diffs and Commenting on Diffs

Create Deep Dive
Oswaldo Ferreira - Backend Engineer
January 28, 2018
Today we’ll cover

- Overview of what a “git diff” is
- Where do we present diffs on GitLab? (Demo / Introduction)
- Overview of how diffs are stored, fetched and presented on GitLab and Gitaly
  - For standard comparison view
  - For merge requests
  - For comments on merge requests and commits
Table of Contents

- Demo
- Workflows
- Tables
- Caching layers
- Code dive
- Questions
**What are git diffs?**

*git-diff* is a function that takes two input data sets and outputs the changes between them. *git diff* is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be commits, branches, files and more.
What are git diffs?
Demo
Standard comparison view diffs (workflow)

● Fetching
  ○ Submits a diff request to Gitaly (through `diff#commit_diff` RPC) with limits and refs
  ○ **Diff limits** are applied (on Gitaly) to the diff file collection
    ■ `Gitlab::Diff::FileCollection::Compare`
    ■ `Gitlab::Git::DiffCollection`
  ○ Most of the process is triggered via `Gitlab::Diff::FileCollection::Compare`

● Presentation
  ○ Each diff file is parsed
    ■ `Gitlab::Diff::File`
    ■ `Gitlab::Diff::Line`
  ○ Load through `project/diffs/_diffs.html.haml` (not async)
1. Storage
2. Fetching
3. Presentation
Merge request diffs (Storage)

create_table "merge_request_diffs", force: :cascade do |t|-
  t.string "state"-
  t.integer "merge_request_id", null: false-
  t.datetime "created_at"-
  t.datetime "updated_at"-
  t.string "base_commit_sha"-
  t.string "real_size"-
  t.string "head_commit_sha"-
  t.string "start_commit_sha"-
  t.integer "commits_count"-
  t.index ["merge_request_id", "id"], name: "index_merge_request_diffs_on_merge_request_id_and_id", using: :btree-
end-

create_table "merge_request_diff_files", id: false, force: :cascade do |t|-
  t.integer "merge_request_diff_id", null: false-
  t.integer "relative_order", null: false-
  t.boolean "new_file", null: false-
  t.boolean "renamed_file", null: false-
  t.boolean "deleted_file", null: false-
  t.boolean "too_large", null: false-
  t.string "a_mode", null: false-
  t.string "b_mode", null: false-
  t.text "new_path", null: false-
  t.text "old_path", null: false-
  t.text "diff", null: false-
  t.boolean "binary"-
  t.index ["merge_request_diff_id", "relative_order"], name: "index_merge_request_diff_files_on_mr_diff_id_and_order", unique: true, using: :btree-
end-
When a push is received for a branch (source), a new MR version is stored (fetch is done via Gitaly though the same process of the comparison view):
  ○ `MergeRequests::ReloadDiffsService#execute`

Creates a new `merge_request_diffs` record and one or more `merge_request_diff_files` (one for each file)

Refreshes the diff highlighting cache (Highlighting is essentially a heavy process)

If a new push is received, a new `merge_request_diffs` is created, and `merge_request_diff_files` are re-created (no deletion or update happens here)

We’re looking forward to store these in Object Storage soon
Once we have the persisted MR version, we fetch it from DB

- Gitlab::Diff::FileCollection::MergeRequestDiff

- The diff highlighting cache is refreshed (if empty) and used (7 days cache)

- Diff stats (files additions and deletions) for each file are also fetched from Gitaly (diff_service#diff_stats RPC) on this process

- If there is any diff comments in positions outside the diff (the diff was expanded by the user and a comment was left), we unfold it (see: Gitlab::Diff::LinesUnfolder) on the fly

```plaintext
... @@ -90,9 +87,11 @@
  87 },
  88 |
  89 | "buildsystem": "meson",
```
● Unlike the standard comparison view, we do load diffs async for MR Diffs tab
● Diff files and lines serialization is mainly done by `DiffFileEntity`
● We can see a few performance issues with the actual size of the serialized JSON and we’re looking forward to improve that by:
  ○ Reducing the amount of data we return (being mindful if everything is really needed by FE)
  ○ In the future, loading diffs in sequential batches, which should lead to a much better UX
Merge request diffs (caching layers)

1. Postgres: The actual raw diff files content
   a. Why: At some point in time, we didn’t have keep-around refs, therefore, after a MR was merged it was impossible to present the diffs
   b. “Side-effect”: Performance improvement 🚀
   c. How: `merge_request_diff_files` table

2. Redis: Latest highlighted diff content
   a. Why: Generating diff highlight HTML (today we use Rouge) is a relatively slow task to make under a request
   b. How: `Gitlab::Diff::HighlightCache`
The GNU General Public License is a free, copyleft license for software and other kinds of works.

+ Adding x

Suggested change

- Adding x

+ Adding
Comments on diffs

1. Storage
2. Fetching
3. Presentation
create_table "notes", force: :cascade do |t|

  t.string "line_code" File path SHA, e.g. 02d635fb83402a9a1a0c113772f1e6d365723b95_93_90

  t.text "position"
  t.text "original_position"

  t.text "change_position"
end
Obs: We delete `merge_request_diff_files` after a MR gets merged, therefore reusing all diffs from MRs is not possible.
Comments on diffs (Storage)

- **DiffNote positions**
  - **original_position**
    - Mainly used to present the comment in the **Discussion tab** of Merge Requests
    - Isn’t updated as new MR versions are added
  - **position**
    - Mainly used to know where exactly we should present the comment in the **Diffs tab**
    - Is updated to the latest MR version if the line wasn’t changed (outdated)
  - **change_position**
    - Mainly used to know in which **context the commented line was changed**
    - Is updated when the line the note was left was changed (**position** stops being updated)

```json
{:base_sha="2fa8f26abf2bce34419c967466dd71148af2d552",
:start_sha="2fa8f26abf2bce34419c967466dd71148af2d552",
:head_sha="db4ae7d7fd18d44e96ab1788c2cf0db8fd84a5b",
:old_path="client/dial_test.go",
:new_path="client/dial_test.go",
:position_type="text",
:old_line=nil,
:new_line=>47}
```

- **base_sha**: Point in time where it was branched off of the target branch
- **start_sha**: Latest HEAD of target branch
- **head_sha**: Latest HEAD of source branch
1. When someone comments in a diff (the whole diff is not persisted):
   ○ It fetches the raw diff for the `original_position` ref (which won’t change after updating the MR), `commit_service#find_commit` RPC is used
     ■ `original_position` is a `Gitlab::Diff::Position` containing the line positions and ref in the diff when it was originally received
   ○ It chunks the diff (because we don’t need all of it), then persist on separate `note_diff_files`
   ○ The same process happens when leaving a comment in a commit

2. When the diff is updated (push for instance):
   ○ We use the `Gitlab::Diff::PositionTracer` to update the `position` of every diff note (if needed)
     ■ That’s exactly what makes a diff note outdated or not, or move it around if needed. If the `position` reference stays in a revision behind the MR HEAD, we got an outdated note
Comments on diffs (caching layers)

1. Postgres: The commented diff file hunks
   a. Why: As you might expect, in the past we were fetching diffs in a N+1 manner for different revisions from Gitaly. In a MR with more than 100 comments, things started getting bit out of control.
      i. Additionally, we started deleting diff files from DB after the MR got merged (table getting too big), therefore, no way to reuse all existing persisted diffs.
   b. How: Chunking the diff file (from top to commented line) and persisting the end result to `note_diff_files`

2. Redis: The highlighted diff file hunks
   a. Why: Same as the standard MR diff file problem. It was spending way too much time highlighting every diff hunk
   b. How: `Gitlab::DiscussionsDiff::HighlightCache`
Questions?
Questions?
Thank you

Oswaldo Ferreira - Backend Engineer
oswaldo@gitlab.com