



GitLab

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

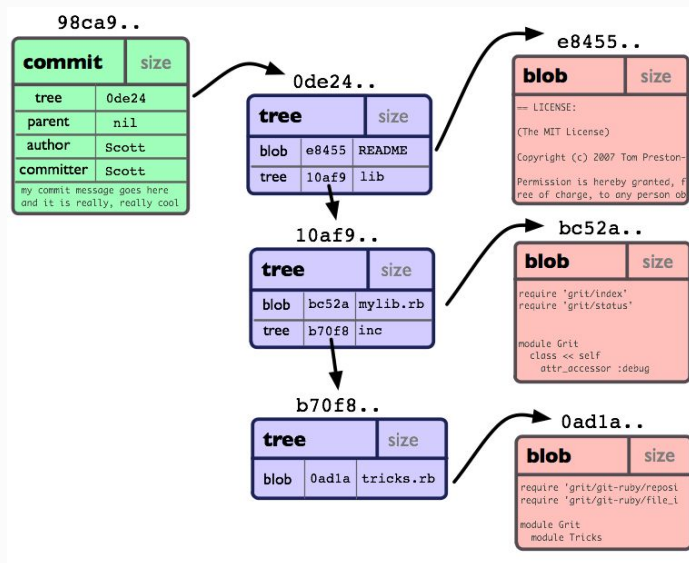


- Demo
- Workflows
- Tables
- Caching layers
- Code dive
- Questions

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?



```
diff --git a/lib/gitlab/gon_helper.rb b/lib/gitlab/gon_helper.rb
index 1513714639..9b1794eec9 100644
--- a/lib/gitlab/gon_helper.rb
+++ b/lib/gitlab/gon_helper.rb
@@ -8,10 +8,7 @@ module Gitlab
```

```
  def add_gon_variables
    gon.api_version = 'v4'
    gon.default_avatar_url = 'v4'
    Gitlab::Utils.append_path(
      Gitlab.config.gitlab.url,
      ActionController::Base.helpers.image_path('no_avatar.png'))
    gon.default_avatar_url = default_avatar_url
    gon.max_file_size = Gitlab::CurrentSettings.max_attachment_size
    gon.asset_host = ActionController::Base.asset_host
    gon.webpack_public_path = webpack_public_path
  end
  # use this method to push multiple feature flags.
  gon.push({ features: { var_name => enabled } }, true)
end
```

```
  def default_avatar_url
    # We can't use ActionController::Base.helpers.image_url because it
    # doesn't return an actual URL because request is nil for some reason.
    #
    # We also can't use Gitlab::Utils.append_path because the image path
    # may be an absolute URL.
    URI.join(Gitlab.config.gitlab.url,
      ActionController::Base.helpers.image_path('no_avatar.png')).to_s
  end
end
```

```
diff --git a/spec/lib/gitlab/gon_helper_spec.rb b/spec/lib/gitlab/gon_helper_spec.rb
index c6f09ca2112..1ff2334bacf 100644
--- a/spec/lib/gitlab/gon_helper_spec.rb
+++ b/spec/lib/gitlab/gon_helper_spec.rb
@@ -29,4 +29,13 @@ describe Gitlab::GonHelper do
  helper.push_frontend_feature_flag(:my_feature_flag, 10)
end

+ describe '#default_avatar_url' do
+   it 'returns an absolute URL' do
+     url = helper.default_avatar_url
+
+     expect(url).to match(/^http/)
+     expect(url).to match(/no_avatar.*png$/)
+   end
+ end
```



```
lib/gitlab/gon_helper.rb
... @@ -8,10 +8,7 @@ module Gitlab
  8
  9 def add_gon_variables
  10 gon.api_version = 'v4'
  11 gon.default_avatar_url = default_avatar_url
  12 Gitlab::Utils.append_path(
  13 Gitlab.config.gitlab.url,
  14 ActionController::Base.helpers.image_path('no_avatar.png'))
  15 gon.max_file_size = Gitlab::CurrentSettings.max_attachment_size
  16 gon.asset_host = ActionController::Base.asset_host
  17 gon.webpack_public_path = webpack_public_path
  ... @@ -50,5 +47,15 @@ module Gitlab
  50 # use this method to push multiple feature flags.
  51 gon.push({ features: { var_name => enabled } }, true)
  52 end
  53
  54 end
  55
  56 def default_avatar_url
  57 # We can't use ActionController::Base.helpers.image_url because
  58 it
  59 # doesn't return an actual URL because request is nil for some reason.
  60 #
  61 # We also can't use Gitlab::Utils.append_path because the image
  62 path
  63 # may be an absolute URL.
  64 URI.join(Gitlab.config.gitlab.url,
  65 ActionController::Base.helpers.image_path('no_avatar.png')).to_s
  66 end
  67 end
```

```
spec/lib/gitlab/gon_helper_spec.rb
... @@ -29,4 +29,13 @@ describe Gitlab::GonHelper do
  29 helper.push_frontend_feature_flag(:my_feature_flag, 10)
  30 end
  31 end
  32
  33 + describe '#default_avatar_url' do
  34 +   it 'returns an absolute URL' do
  35 +     url = helper.default_avatar_url
  36 +
  37 +     expect(url).to match(/^http/)
  38 +     expect(url).to match(/no_avatar.*png$/)
  39 +   end
  40 + end
  41 end
```



Demo



- 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~
```

Merge request diffs (Storage workflow)



- 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

Merge request diffs (Fetching workflow)



- 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

```
... @@ -90,9 +87,11 @@  
87         },  
88         {  
89             "buildsystem": "meson",
```

Merge request diffs (Presentation workflow)



- 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*





Comments on diffs (Discussion tab)




 Administrator @root started a discussion on an old version of the diff 1 week ago Toggle discussion

 **LICENSE** 

9	9	
10	10	The GNU General Public License is a free, copyleft license for
11	11	software and other kinds of works.
12		+ Adding x

 Administrator @root commented 1 week ago Maintainer    



Suggested change 

Applied

12		- Adding x
	12	+ Adding

Reply...

Resolve discussion



1. Storage
2. Fetching
3. Presentation

Comments on diffs (Storage)



```
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
```


Comments on merge request diffs (Storage)



```
create_table "note_diff_files", force: :cascade do |t|  
  t.integer "diff_note_id", null: false  
  t.text "diff", null: false  
  t.boolean "new_file", null: false  
  t.boolean "renamed_file", null: false  
  t.boolean "deleted_file", 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.index ["diff_note_id"], name: "index_note_diff_files_on_diff_note_id", unique: true, using: :btree  
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)

```
=> {:base_sha=>"2fa8f20abf2bce34419c967406dd71148af2d552",  
:start_sha=>"2fa8f20abf2bce34419c967406dd71148af2d552",  
:head_sha=>"db4ae70d7fd10d44e96ab1788c2cf0db8fd84a5b",  
: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



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 a 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?



Code Dive



Questions?



Thank you

Oswaldo Ferreira - Backend Engineer
oswaldo@gitlab.com