Working with comments

In this article

Using the REST API, you can access and manage comments in your pull requests, issues, or commits.

For any Pull Request, GitHub Enterprise Cloud provides three kinds of comment views: comments on the Pull Request as a whole, comments on a specific line within the Pull Request, and comments on a specific commit within the Pull Request.

Each of these types of comments goes through a different portion of the GitHub API. In this guide, we'll explore how you can access and manipulate each one. For every example, we'll be using this sample Pull Request made on the "octocat" repository. As always, samples can be found in our platform-samples repository.

Pull Request Comments

To access comments on a Pull Request, you'll use the endpoints to manage issues. This may seem counterintuitive at first. But once you understand that a Pull Request is just an Issue with code, it makes sense to use these endpoints to create comments on a Pull Request.

We'll demonstrate fetching Pull Request comments by creating a Ruby script using Octokit.rb. You'll also want to create a personal access token.

The following code should help you get started accessing comments from a Pull Request using Octokit.rb:

```
require 'octokit'

# !!! DO NOT EVER USE HARD-CODED VALUES IN A REAL APP !!!
# Instead, set and test environment variables, like below
client = Octokit::Client.new :access_token => ENV['MY_PERSONAL_TOKEN']

client.issue_comments("octocat/Spoon-Knife", 1176).each do |comment|
  username = comment[:user][:login]
  post_date = comment[:created_at]
  content = comment[:body]

  puts "#{username} made a comment on #{post_date}. It says:\n'#{content}'\n"
end
```

Here, we're specifically calling out to the API to get the comments (`issue_comments`), providing both the repository's name (`octocat/Spoon-Knife`), and the Pull Request ID we're interested in (`1176`). After that, it's simply a matter of iterating through the comments to fetch information about each one.

Pull Request Comments on a Line

Within the diff view, you can start a discussion on a particular aspect of a singular change made within the Pull Request. These comments occur on the individual lines within a changed file. The endpoint URL for this discussion comes from the endpoint to manage pull request reviews.

The following code fetches all the Pull Request comments made on files, given a single Pull Request number:

```
require 'octokit'

# !!! DO NOT EVER USE HARD-CODED VALUES IN A REAL APP !!!
# Instead, set and test environment variables, like below
client = Octokit::Client.new :access_token => ENV['MY_PERSONAL_TOKEN']

client.pull_request_comments("octocat/Spoon-Knife", 1176).each do |comment|
  username = comment[:user][:login]
  post_date = comment[:created_at]
  content = comment[:body]
  path = comment[:path]
  position = comment[:position]

  puts "#{username} made a comment on #{post_date} for the file called #{path},
on line #{position}. It says:\n'#{content}'\n"
end
```

You'll notice that it's incredibly similar to the example above. The difference between this view and the Pull Request comment is the focus of the conversation. A comment made on a Pull Request should be reserved for discussion or ideas on the overall direction of the code. A comment made as part of a Pull Request review should deal specifically with the way a particular change was implemented within a file.

Commit Comments

The last type of comments occur specifically on individual commits. For this reason, they make use of the endpoint to manage commit comments.

To retrieve the comments on a commit, you'll want to use the SHA1 of the commit. In other words, you won't use any identifier related to the Pull Request. Here's an example:

```
require 'octokit'

# !!! DO NOT EVER USE HARD-CODED VALUES IN A REAL APP !!!
# Instead, set and test environment variables, like below
client = Octokit::Client.new :access_token => ENV['MY_PERSONAL_TOKEN']

client.commit_comments("octocat/Spoon-Knife",
"cbc28e7c8caee26febc8c013b0adfb97a4edd96e").each do |comment|
  username = comment[:user][:login]
  post_date = comment[:created_at]
  content = comment[:body]

  puts "#{username} made a comment on #{post_date}. It says:\n'#{content}'\n"
end
```

Note that this API call will retrieve single line comments, as well as comments made on the entire commit.