Gerrit Client with Python

Release 1.0.1

Jialiang Shi

Dec 04, 2021

CONTENTS:

CHAPTER

ONE

INSTALLATION

`python-gerrit-api` is compatible with Python 2.7 3.5+.

Use **`pip`** to install the latest stable version of `python-gerrit-api`:

```
$ sudo pip install python-gerrit-api
```

The current development version is available on github. Use **`git`** and **`python setup.py`** to install it:

```
$ git clone https://github.com/shijl0925/python-gerrit-api.git
$ cd python-gerrit-api
$ sudo python setup.py install
```

CHAPTER

TWO

API EXAMPLES

2.1 api/projects

2.1.1 Examples

setup gerrit client:

```
from gerrit import GerritClient
client = GerritClient(base_url="https://yourgerrit", username='******', password='xxxxx')
```

Lists the projects:

```
projects = client.projects.list()
```

Retrieves a project.:

```
project = client.projects.get(project_name="MyProject")
```

Creates a new project.:

```
input_ = {
    "description": "This is a demo project.",
    "submit_type": "INHERIT",
    "owners": [
      "MyProject-Owners"
    ]
}
project = client.projects.create('MyProject', input_)
```

Delete the project, requires delete-project plugin:

```
client.projects.delete("MyProject")
# or
project = client.projects.get(project_name="MyProject")
project.delete()
```

2.2 api/project

2.2.1 Examples

setup gerrit client and retrieve one project instance:

```
from gerrit import GerritClient
client = GerritClient(base_url="https://yourgerrit", username='******', password='xxxxx')

project = client.projects.get('python-sonarqube-api')
```

Retrieves the description of a project.:

```
description = project.get_description()
```

Sets the description of a project.:

```
input_ = {
    "description": "Plugin for Gerrit that handles the replication.",
    "commit_message": "Update the project description"
}
result = project.set_description(input_)
```

Deletes the description of a project.:

```
project.delete_description()
```

Delete the project, requires delete-project plugin:

```
project.delete()
```

Sets the configuration of a project.:

```
input_ = {
    "description": "demo project",
    "use_contributor_agreements": "FALSE",
    "use_content_merge": "INHERIT",
    "use_signed_off_by": "INHERIT",
    "create_new_change_for_all_not_in_target": "INHERIT",
    "enable_signed_push": "INHERIT",
    "require_signed_push": "INHERIT",
    "reject_implicit_merges": "INHERIT",
    "require_change_id": "TRUE",
    "max_object_size_limit": "10m",
    "submit_type": "REBASE_IF_NECESSARY",
    "state": "ACTIVE"
}
result = project.set_config(input_)
```

Lists the access rights for a single project.:

```
access_rights = project.get_access_rights()
```

Create Change for review. support this method since v3.3.0:

```
input_ = {
    "subject": "Let's support 100% Gerrit workflow direct in browser",
    "branch": "stable",
    "topic": "create-change-in-browser",
    "status": "NEW"
}

result = project.create_change(input_)
```

List the branches of a project.:

```
branches = project.branches.list()
```

get a branch by ref:

```
branch = project.branches.get('refs/heads/stable')
```

Creates a new branch.:

```
input_ = {
    'revision': '76016386a0d8ecc7b6be212424978bb45959d668'
}
new_branch = project.branches.create('stable', input_)
```

Delete a branch.:

```
branch.delete()
```

Retrieves a commit of a project.:

```
commit = project.get_commit('c641ab4dd180b4184f2663bd28277aa796b36417')
```

Retrieves the branches and tags in which a change is included.:

```
result = commit.get_include_in()
```

Gets the content of a file from a certain commit.:

```
content = commit.get_file_content('sonarqube/community/components.py')
```

Cherry-picks a commit of a project to a destination branch.:

```
input_ = {
    "message": "Test Cherry Pick",
    "destination": "stable"
}

commit.cherry_pick(input_)
```

Lists the files that were modified, added or deleted in a commit.:

```
result = commit.list_change_files()
```

2.3 api/changes

2.3.1 Examples

setup gerrit client:

```
from gerrit import GerritClient
client = GerritClient(base_url="https://yourgerrit", username='******', password='xxxxx')
```

Queries changes.:

```
result = client.changes.search(query=['status:open'])
# or
query = ["is:open+owner:self", "is:open+reviewer:self+-owner:self",
↪"is:closed+owner:self+limit:5"]
result = client.changes.search(query=query, options=["LABELS"])
```

Retrieves a change.:

```
change = client.changes.get("MyProject~master~I39b027b763fb0b0dc7ed6c9e6bb5128d882dbe7c")
```

create a change.:

```
input_ = {
    "project": "myProject",
    "subject": "Let's support 100% Gerrit workflow direct in browser",
    "branch": "stable",
    "topic": "create-change-in-browser",
    "status": "NEW"
}
result = client.changes.create(input_)
```

Deletes a change.:

```
client.changes.delete("MyProject~master~I39b027b763fb0b0dc7ed6c9e6bb5128d882dbe7c")
# or
change = client.changes.get("MyProject~master~I39b027b763fb0b0dc7ed6c9e6bb5128d882dbe7c")
change.delete()
```

2.4 api/change

2.4.1 Examples

setup gerrit client and retrieve one change instance:

```
from gerrit import GerritClient
client = GerritClient(base_url="https://yourgerrit", username='******', password='xxxxx')

change = client.changes.get(
    "MyProject~master~I39b027b763fb0b0dc7ed6c9e6bb5128d882dbe7c"
)
```

Update an existing change by using a MergePatchSetInput entity.:

```
input_ = {
    "subject": "Merge master into stable",
    "merge": {
      "source": "refs/heads/master"
    }
}

result = change.update(input_)
```

Creates a new patch set with a new commit message.:

```
input_ = {
    "message": "New Commit message \\n\\nChange-Id:␣
↪I10394472cbd17dd12454f229e4f6de00b143a444\\n"
}

result = change.set_commit_message(input_)
```

Retrieves the topic of a change.:

```
topic = change.get_topic()
```

Sets the topic of a change.:

```
topic = change.set_topic("test topic")
```

Deletes the topic of a change.:

```
change.delete_topic()
```

Retrieves the account of the user assigned to a change.:

```
assignee = change.get_assignee()
```

Sets the assignee of a change.:

```
input_ = {
    "assignee": "jhon.doe"
}
result = change.set_assignee(input_)
```

Returns a list of every user ever assigned to a change, in the order in which they were first assigned.:

```
result = change.get_past_assignees()
```

Deletes the assignee of a change.:

```
result = change.delete_assignee()
```

Abandons a change.:

```
result = change.abandon()
```

Restores a change.:

```
result = change.restore()
```

Deletes a change.:

```
change.delete()
```

Marks the change to be private. Only open changes can be marked private.:

```
input_ = {
    "message": "After this security fix has been released we can make it public now."
}
change.mark_private(input_)
```

Marks the change to be non-private. Note users can only unmark own private changes.:

```
input_ = {
    "message": "This is a security fix that must not be public."
}
change.unmark_private(input_)
# or
change.unmark_private()
```

get one revision by revision id.:

```
revision = change.get_revision("534b3ce21655a092eccf72680f2ad16b8fecf119")
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search