python-gitlab Documentation

Release 1.3.0

Gauvain Pocentek, Mika Mäenpää

Feb 18, 2018

Contents

Contents:

CHAPTER 1

Installation

`python-gitlab` is compatible with Python 2.7 and 3.4+.

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

```
$ sudo pip install --upgrade python-gitlab
```

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

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

CHAPTER 2

gitlab CLI usage

python-gitlab provides a **gitlab** command-line tool to interact with GitLab servers. It uses a configuration file to define how to connect to the servers.

2.1 Configuration

2.1.1 Files

gitlab looks up 2 configuration files by default:

/etc/python-gitlab.cfg System-wide configuration file

~/.python-gitlab.cfg User configuration file

You can use a different configuration file with the --config-file option.

2.1.2 Content

The configuration file uses the INI format. It contains at least a [global] section, and a specific section for each GitLab server. For example:

```
[global]
default = somewhere
ssl_verify = true
timeout = 5

[somewhere]
url = https://some.whe.re
private_token = vTbFeqJYCY3sibBP7BZM
api_version = 3

[elsewhere]
url = http://else.whe.re:8080
```

```
private_token = CkqsjqcQSFH5FQKDccu4
timeout = 1
```

The `default` option of the `[global]` section defines the GitLab server to use if no server is explicitly specified with the `--gitlab` CLI option.

The `[global]` section also defines the values for the default connection parameters. You can override the values in each GitLab server section.

Table 2.1: Global options

Option	Possible values	Description
`ssl_verify`	`True`, `False`, or a `str`	Verify the SSL certificate. Set to `False` to disable verification, though this will create warnings. Any other value is interpreted as path to a CA_BUNDLE file or directory with certificates of trusted CAs.
`timeout`	Integer	Number of seconds to wait for an answer before failing.
`api_version`	`3` or `4`	The API version to use to make queries. Requires python-gitlab >= 1.3.0.

You must define the `url` in each GitLab server section.

Only one of `private_token` or `oauth_token` should be defined. If neither are defined an anonymous request will be sent to the Gitlab server, with very limited permissions.

Table 2.2: GitLab server options

Option	Description
`url`	URL for the GitLab server
`private_token`	Your user token. Login/password is not supported. Refer to the official documentation to learn how to obtain a token.
`oauth_token`	An Oauth token for authentication. The Gitlab server must be configured to support this authentication method.
`api_version`	GitLab API version to use (`3` or `4`). Defaults to `4` since version 1.3.0.
`http_username`	Username for optional HTTP authentication
`http_password`	Password for optional HTTP authentication

2.2 CLI

2.2.1 Objects and actions

The `gitlab` command expects two mandatory arguments. The first one is the type of object that you want to manipulate. The second is the action that you want to perform. For example:

```
$ gitlab project list
```

Use the `--help` option to list the available object types and actions:

```
$ gitlab --help
$ gitlab project --help
```

Some actions require additional parameters. Use the `--help` option to list mandatory and optional arguments for an action:

```
$ gitlab project create --help
```

2.2.2 Optional arguments

Use the following optional arguments to change the behavior of `gitlab`. These options must be defined before the mandatory arguments.

`--verbose`, `-v` Outputs detail about retrieved objects. Available for legacy (default) output only.

`--config-file`, `-c` Path to a configuration file.

`--gitlab`, `-g` ID of a GitLab server defined in the configuration file.

`--output`, `-o` Output format. Defaults to a custom format. Can also be `yaml` or `json`.

`--fields`, `-f` Comma-separated list of fields to display (`yaml` and `json` output formats only). If not used, all the object fields are displayed.

Example:

```
$ gitlab -o yaml -f id,permissions -g elsewhere -c /tmp/gl.cfg project list
```

2.3 Examples

List the projects (paginated):

```
$ gitlab project list
```

List all the projects:

```
$ gitlab project list --all
```

Limit to 5 items per request, display the 1st page only

```
$ gitlab project list --page 1 --per-page 5
```

Get a specific project (id 2):

```
$ gitlab project get --id 2
```

Get a specific user by id:

```
$ gitlab user get --id 3
```

Get a list of snippets for this project:

```
$ gitlab project-issue list --project-id 2
```

Delete a snippet (id 3):

```
$ gitlab project-snippet delete --id 3 --project-id 2
```

Update a snippet:

```
$ gitlab project-snippet update --id 4 --project-id 2 \
    --code "My New Code"
```

Create a snippet:

```
$ gitlab project-snippet create --project-id 2
Impossible to create object (Missing attribute(s): title, file-name, code)
$ # oops, let's add the attributes:
$ gitlab project-snippet create --project-id 2 --title "the title" \
    --file-name "the name" --code "the code"
```

Define the status of a commit (as would be done from a CI tool for example):

```
$ gitlab project-commit-status create --project-id 2 \
    --commit-id a43290c --state success --name ci/jenkins \
    --target-url http://server/build/123 \
    --description "Jenkins build succeeded"
```

Use sudo to act as another user (admin only):

```
$ gitlab project create --name user_project1 --sudo username
```

CHAPTER 3

Getting started with the API

python-gitlab supports both GitLab v3 and v4 APIs.

v3 being deprecated by GitLab, its support in python-gitlab will be minimal. The development team will focus on v4.

v4 is the default API used by python-gitlab since version 1.3.0.

3.1 `gitlab.Gitlab` class

To connect to a GitLab server, create a `gitlab.Gitlab` object:

```
import gitlab

# private token or personal token authentication
gl = gitlab.Gitlab('http://10.0.0.1', private_token='JVNSESs8EwWRx5yDxM5q')

# oauth token authentication
gl = gitlab.Gitlab('http://10.0.0.1', oauth_token='my_long_token_here')

# username/password authentication (for GitLab << 10.2)
gl = gitlab.Gitlab('http://10.0.0.1', email='jdoe', password='s3cr3t')

# anonymous gitlab instance, read-only for public resources
gl = gitlab.Gitlab('http://10.0.0.1')

# make an API request to create the gl.user object. This is mandatory if you
# use the username/password authentication.
gl.auth()
```

You can also use configuration files to create `gitlab.Gitlab` objects:

```
gl = gitlab.Gitlab.from_config('somewhere', ['/tmp/gl.cfg'])
```

See the *Configuration* section for more information about configuration files.

3.1.1 Note on password authentication

The `/session` API endpoint used for username/password authentication has been removed from GitLab in version 10.2, and is not available on gitlab.com anymore. Personal token authentication is the prefered authentication method.

If you need username/password authentication, you can use cookie-based authentication. You can use the web UI form to authenticate, retrieve cookies, and then use a custom `requests.Session` object to connect to the GitLab API. The following code snippet demonstrates how to automate this: https://gist.github.com/gpocentek/bd4c3fbf8a6ce226ebddc4aad6b46c0a.

See issue 380 for a detailed discussion.

3.2 API version

`python-gitlab` uses the v4 GitLab API by default. Use the `api_version` parameter to switch to v3:

```
import gitlab

gl = gitlab.Gitlab('http://10.0.0.1', 'JVNSESs8EwWRx5yDxM5q', api_version=3)
```

Warning: The python-gitlab API is not the same for v3 and v4. Make sure to read *Switching to GitLab API v4* if you are upgrading from v3.

3.3 Managers

The `gitlab.Gitlab` class provides managers to access the GitLab resources. Each manager provides a set of methods to act on the resources. The available methods depend on the resource type.

Examples:

```
# list all the projects
projects = gl.projects.list()
for project in projects:
    print(project)

# get the group with id == 2
group = gl.groups.get(2)
for group in groups:
    print()

# create a new user
user_data = {'email': 'jen@foo.com', 'username': 'jen', 'name': 'Jen'}
user = gl.users.create(user_data)
print(user)
```

You can list the mandatory and optional attributes for object creation with the manager's `get_create_attrs()` method. It returns 2 tuples, the first one is the list of mandatory attributes, the second one the list of optional attribute:

```
# v4 only
print(gl.projects.get_create_attrs())
(('name',), ('path', 'namespace_id', ...))
```

The attributes of objects are defined upon object creation, and depend on the GitLab API itself. To list the available information associated with an object use the python introspection tools for v3, or the `attributes` attribute for v4:

```
project = gl.projects.get(1)

# v3
print(vars(project))
# or
print(project.__dict__)

# v4
print(project.attributes)
```

Some objects also provide managers to access related GitLab resources:

```
# list the issues for a project
project = gl.projects.get(1)
issues = project.issues.list()
```

3.4 Gitlab Objects

You can update or delete a remote object when it exists locally:

```
# update the attributes of a resource
project = gl.projects.get(1)
project.wall_enabled = False
# don't forget to apply your changes on the server:
project.save()

# delete the resource
project.delete()
```

Some classes provide additional methods, allowing more actions on the GitLab resources. For example:

```
# star a git repository
project = gl.projects.get(1)
project.star()
```

3.5 Base types

The `gitlab` package provides some base types.

- `gitlab.Gitlab` is the primary class, handling the HTTP requests. It holds the GitLab URL and authentication information.

For v4 the following types are defined:

- `gitlab.base.RESTObject` is the base class for all the GitLab v4 objects. These objects provide an abstraction for GitLab resources (projects, groups, and so on).
- `gitlab.base.RESTManager` is the base class for v4 objects managers, providing the API to manipulate the resources and their attributes.

For v3 the following types are defined:

- `gitlab.base.GitlabObject` is the base class for all the GitLab v3 objects. These objects provide an abstraction for GitLab resources (projects, groups, and so on).
- `gitlab.base.BaseManager` is the base class for v3 objects managers, providing the API to manipulate the resources and their attributes.

3.6 Lazy objects (v4 only)

To avoid useless calls to the server API, you can create lazy objects. These objects are created locally using a known ID, and give access to other managers and methods.

The following exemple will only make one API call to the GitLab server to star a project:

```
# star a git repository
project = gl.projects.get(1, lazy=True)  # no API call
project.star()  # API call
```

3.7 Pagination

You can use pagination to iterate over long lists. All the Gitlab objects listing methods support the `page` and `per_page` parameters:

```
ten_first_groups = gl.groups.list(page=1, per_page=10)
```

Note: The first page is page 1, not page 0, except for project commits in v3 API.

By default GitLab does not return the complete list of items. Use the `all` parameter to get all the items when using listing methods:

```
all_groups = gl.groups.list(all=True)
all_owned_projects = gl.projects.owned(all=True)
```

Warning: python-gitlab will iterate over the list by calling the corresponding API multiple times. This might take some time if you have a lot of items to retrieve. This might also consume a lot of memory as all the items will be stored in RAM. If you're encountering the python recursion limit exception, use `safe_all=True` instead to stop pagination automatically if the recursion limit is hit.

With v4, `list()` methods can also return a generator object which will handle the next calls to the API when required:

```
items = gl.groups.list(as_list=False)
for item in items:
    print(item.attributes)
```

The generator exposes extra listing information as received by the server:

- `current_page`: current page number (first page is 1)
- `prev_page`: if `None` the current page is the first one
- `next_page`: if `None` the current page is the last one

- `per_page`: number of items per page
- `total_pages`: total number of pages available
- `total`: total number of items in the list

3.8 Sudo

If you have the administrator status, you can use `sudo` to act as another user. For example:

```
p = gl.projects.create({'name': 'awesome_project'}, sudo='user1')
```

3.9 Advanced HTTP configuration

python-gitlab relies on `requests` `Session` objects to perform all the HTTP requests to the Gitlab servers.

You can provide your own `Session` object with custom configuration when you create a `Gitlab` object.

3.9.1 Context manager

You can use `Gitlab` objects as context managers. This makes sure that the `requests.Session` object associated with a `Gitlab` instance is always properly closed when you exit a `with` block:

```
with gitlab.Gitlab(host, token) as gl:
    gl.projects.list()
```

Warning: The context manager will also close the custom `Session` object you might have used to build a `Gitlab` instance.

3.9.2 Proxy configuration

The following sample illustrates how to define a proxy configuration when using python-gitlab:

```
import gitlab
import requests

session = requests.Session()
session.proxies = {
    'https': os.environ.get('https_proxy'),
    'http': os.environ.get('http_proxy'),
}
gl = gitlab.gitlab(url, token, api_version=4, session=session)
```

Reference: http://docs.python-requests.org/en/master/user/advanced/#proxies

3.9.3 Client side certificate

The following sample illustrates how to use a client-side certificate:

```
import gitlab
import requests

session = requests.Session()
s.cert = ('/path/to/client.cert', '/path/to/client.key')
gl = gitlab.gitlab(url, token, api_version=4, session=session)
```

Reference: http://docs.python-requests.org/en/master/user/advanced/#client-side-certificates

CHAPTER 4

Switching to GitLab API v4

GitLab provides a new API version (v4) since its 9.0 release. `python-gitlab` provides support for this new version, but the python API has been modified to solve some problems with the existing one.

GitLab will stop supporting the v3 API soon, and you should consider switching to v4 if you use a recent version of GitLab (>= 9.0), or if you use http://gitlab.com.

4.1 Using the v4 API

python-gitlab uses the v4 API by default since the 1.3.0 release. To use the old v3 API, explicitly define `api_version` in the `Gitlab` constructor:

```
gl = gitlab.Gitlab(..., api_version=3)
```

If you use the configuration file, also explicitly define the version:

```
[my_gitlab]
...
api_version = 3
```

4.2 Changes between v3 and v4 API

For a list of GitLab (upstream) API changes, see https://docs.gitlab.com/ce/api/v3_to_v4.html.

The `python-gitlab` API reflects these changes. But also consider the following important changes in the python API:

- managers and objects don't inherit from `GitlabObject` and `BaseManager` anymore. They inherit from *`RESTManager`* and *`RESTObject`*.
- You should only use the managers to perform CRUD operations.

  The following v3 code:

```
gl = gitlab.Gitlab(...)
p = Project(gl, project_id)
```

Should be replaced with:

```
gl = gitlab.Gitlab(...)
p = gl.projects.get(project_id)
```

- Listing methods (`manager.list()` for instance) can now return generators (*RESTObjectList*). They handle the calls to the API when needed to fetch new items.

  By default you will still get lists. To get generators use `as_list=False`:

```
all_projects_g = gl.projects.list(as_list=False)
```

- The "nested" managers (for instance `gl.project_issues` or `gl.group_members`) are not available anymore. Their goal was to provide a direct way to manage nested objects, and to limit the number of needed API calls.

  To limit the number of API calls, you can now use `get()` methods with the `lazy=True` parameter. This creates shallow objects that provide usual managers.

  The following v3 code:

```
issues = gl.project_issues.list(project_id=project_id)
```

  Should be replaced with:

```
issues = gl.projects.get(project_id, lazy=True).issues.list()
```

  This will make only one API call, instead of two if `lazy` is not used.

- The following *Gitlab* methods should not be used anymore for v4:
  - `list()`
  - `get()`
  - `create()`
  - `update()`
  - `delete()`
- If you need to perform HTTP requests to the GitLab server (which you shouldn't), you can use the following *Gitlab* methods:
  - *http_request*
  - *http_get*
  - *http_list*
  - *http_post*
  - *http_put*
  - *http_delete*

CHAPTER 5

API examples

5.1 Access requests

Users can request access to groups and projects.

When access is granted the user should be given a numerical access level. The following constants are provided to represent the access levels:

- `gitlab.GUEST_ACCESS`: `10`
- `gitlab.REPORTER_ACCESS`: `20`
- `gitlab.DEVELOPER_ACCESS`: `30`
- `gitlab.MASTER_ACCESS`: `40`
- `gitlab.OWNER_ACCESS`: `50`

5.1.1 References

- v4 API:
  - *gitlab.v4.objects.ProjectAccessRequest*
  - *gitlab.v4.objects.ProjectAccessRequestManager*
  - `gitlab.v4.objects.Project.accessrequests`
  - *gitlab.v4.objects.GroupAccessRequest*
  - *gitlab.v4.objects.GroupAccessRequestManager*
  - `gitlab.v4.objects.Group.accessrequests`
- v3 API:
  - *gitlab.v3.objects.ProjectAccessRequest*
  - *gitlab.v3.objects.ProjectAccessRequestManager*

  - *gitlab.v3.objects.Project.accessrequests*
  - gitlab.Gitlab.project_accessrequests
  - *gitlab.v3.objects.GroupAccessRequest*
  - *gitlab.v3.objects.GroupAccessRequestManager*
  - *gitlab.v3.objects.Group.accessrequests*
  - gitlab.Gitlab.group_accessrequests
- GitLab API: https://docs.gitlab.com/ce/api/access_requests.html

5.1.2 Examples

List access requests from projects and groups:

```
p_ars = project.accessrequests.list()
g_ars = group.accessrequests.list()
```

Get a single request:

```
p_ar = project.accessrequests.get(user_id)
g_ar = group.accessrequests.get(user_id)
```

Create an access request:

```
p_ar = project.accessrequests.create({})
g_ar = group.accessrequests.create({})
```

Approve an access request:

```
ar.approve()  # defaults to DEVELOPER level
ar.approve(access_level=gitlab.MASTER_ACCESS)  # explicitly set access level
```

Deny (delete) an access request:

```
project.accessrequests.delete(user_id)
group.accessrequests.delete(user_id)
# or
ar.delete()
```

5.2 Award Emojis

5.2.1 Reference

- v4 API:
  - *gitlab.v4.objects.ProjectIssueAwardEmoji*
  - *gitlab.v4.objects.ProjectIssueNoteAwardEmoji*
  - *gitlab.v4.objects.ProjectMergeRequestAwardEmoji*
  - *gitlab.v4.objects.ProjectMergeRequestNoteAwardEmoji*
  - *gitlab.v4.objects.ProjectSnippetAwardEmoji*

  - *gitlab.v4.objects.ProjectSnippetNoteAwardEmoji*
  - *gitlab.v4.objects.ProjectIssueAwardEmojiManager*
  - *gitlab.v4.objects.ProjectIssueNoteAwardEmojiManager*
  - *gitlab.v4.objects.ProjectMergeRequestAwardEmojiManager*
  - *gitlab.v4.objects.ProjectMergeRequestNoteAwardEmojiManager*
  - *gitlab.v4.objects.ProjectSnippetAwardEmojiManager*
  - *gitlab.v4.objects.ProjectSnippetNoteAwardEmojiManager*
- GitLab API: https://docs.gitlab.com/ce/api/award_emoji.html

5.2.2 Examples

List emojis for a resource:

```
emojis = obj.awardemojis.list()
```

Get a single emoji:

```
emoji = obj.awardemojis.get(emoji_id)
```

Add (create) an emoji:

```
emoji = obj.awardemojis.create({'name': 'tractor'})
```

Delete an emoji:

```
emoji.delete
# or
obj.awardemojis.delete(emoji_id)
```

5.3 Branches

5.3.1 References

- v4 API:
  - *gitlab.v4.objects.ProjectBranch*
  - *gitlab.v4.objects.ProjectBranchManager*
  - gitlab.v4.objects.Project.branches
- v3 API:
  - *gitlab.v3.objects.ProjectBranch*
  - *gitlab.v3.objects.ProjectBranchManager*
  - *gitlab.v3.objects.Project.branches*
- GitLab API: https://docs.gitlab.com/ce/api/branches.html

5.3.2 Examples

Get the list of branches for a repository:

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

Get a single repository branch:

```
branch = project.branches.get('master')
```

Create a repository branch:

```
# v4
branch = project.branches.create({'branch': 'feature1',
                                  'ref': 'master'})

#v3
branch = project.branches.create({'branch_name': 'feature1',
                                  'ref': 'master'})
```

Delete a repository branch:

```
project.branches.delete('feature1')
# or
branch.delete()
```

Protect/unprotect a repository branch:

```
branch.protect()
branch.unprotect()
```

Note: By default, developers are not authorized to push or merge into protected branches. This can be changed by passing `developers_can_push` or `developers_can_merge`:

```
branch.protect(developers_can_push=True, developers_can_merge=True)
```

5.4 Protected branches

You can define a list of protected branch names on a repository. Names can use wildcards (`*`).

5.4.1 References

- v4 API:
  - *gitlab.v4.objects.ProjectProtectedBranch*
  - *gitlab.v4.objects.ProjectProtectedBranchManager*
  - `gitlab.v4.objects.Project.protectedbranches`
- GitLab API: https://docs.gitlab.com/ce/api/protected_branches.html#protected-branches-api

5.4.2 Examples

Get the list of protected branches for a project:

```
p_branches = project.protectedbranches.list()
```

Get a single protected branch:

```
p_branch = project.protectedbranches.get('master')
```

Create a protected branch:

```
p_branch = project.protectedbranches.create({'name': '*-stable'})
```

Delete a protected branch:

```
project.protectedbranches.delete('*-stable')
# or
p_branch.delete()
```

5.5 Broadcast messages

You can use broadcast messages to display information on all pages of the gitlab web UI. You must have administration permissions to manipulate broadcast messages.

5.5.1 References

- v4 API:
  - *gitlab.v4.objects.BroadcastMessage*
  - *gitlab.v4.objects.BroadcastMessageManager*
  - gitlab.Gitlab.broadcastmessages
- v3 API:
  - *gitlab.v3.objects.BroadcastMessage*
  - *gitlab.v3.objects.BroadcastMessageManager*
  - gitlab.Gitlab.broadcastmessages
- GitLab API: https://docs.gitlab.com/ce/api/broadcast_messages.html

5.5.2 Examples

List the messages:

```
msgs = gl.broadcastmessages.list()
```

Get a single message:

```
msg = gl.broadcastmessages.get(msg_id)
```

Create a message:

```
msg = gl.broadcastmessages.create({'message': 'Important information'})
```

The date format for `starts_at` and `ends_at` parameters is `YYYY-MM-ddThh:mm:ssZ`.

Update a message:

```
msg.font = '#444444'
msg.color = '#999999'
msg.save()
```

Delete a message:

```
gl.broadcastmessages.delete(msg_id)
# or
msg.delete()
```

5.6 Pipelines, Builds and Jobs

Build and job are two classes representing the same object. Builds are used in v3 API, jobs in v4 API.

5.6.1 Project pipelines

A pipeline is a group of jobs executed by GitLab CI.

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectPipeline*
  - *gitlab.v4.objects.ProjectPipelineManager*
  - gitlab.v4.objects.Project.pipelines
- v3 API:
  - *gitlab.v3.objects.ProjectPipeline*
  - *gitlab.v3.objects.ProjectPipelineManager*
  - *gitlab.v3.objects.Project.pipelines*
  - gitlab.Gitlab.project_pipelines
- GitLab API: https://docs.gitlab.com/ce/api/pipelines.html

Examples

List pipelines for a project:

```
pipelines = project.pipelines.list()
```

Get a pipeline for a project:

```
pipeline = project.pipelines.get(pipeline_id)
```

Create a pipeline for a particular reference:

```
pipeline = project.pipelines.create({'ref': 'master'})
```

Retry the failed builds for a pipeline:

```
pipeline.retry()
```

Cancel builds in a pipeline:

```
pipeline.cancel()
```

5.6.2 Triggers

Triggers provide a way to interact with the GitLab CI. Using a trigger a user or an application can run a new build/job for a specific commit.

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectTrigger*
  - *gitlab.v4.objects.ProjectTriggerManager*
  - gitlab.v4.objects.Project.triggers
- v3 API:
  - *gitlab.v3.objects.ProjectTrigger*
  - *gitlab.v3.objects.ProjectTriggerManager*
  - *gitlab.v3.objects.Project.triggers*
  - gitlab.Gitlab.project_triggers
- GitLab API: https://docs.gitlab.com/ce/api/pipeline_triggers.html

Examples

List triggers:

```
triggers = project.triggers.list()
```

Get a trigger:

```
trigger = project.triggers.get(trigger_token)
```

Create a trigger:

```
trigger = project.triggers.create({}) # v3
trigger = project.triggers.create({'description': 'mytrigger'}) # v4
```

Remove a trigger:

```
project.triggers.delete(trigger_token)
# or
trigger.delete()
```

5.6.3 Pipeline schedule

You can schedule pipeline runs using a cron-like syntax. Variables can be associated with the scheduled pipelines.

Reference

- v4 API
  - *gitlab.v4.objects.ProjectPipelineSchedule*
  - *gitlab.v4.objects.ProjectPipelineScheduleManager*
  - gitlab.v4.objects.Project.pipelineschedules
  - *gitlab.v4.objects.ProjectPipelineScheduleVariable*
  - *gitlab.v4.objects.ProjectPipelineScheduleVariableManager*
  - gitlab.v4.objects.Project.pipelineschedules
- GitLab API: https://docs.gitlab.com/ce/api/pipeline_schedules.html

Examples

List pipeline schedules:

```
scheds = project.pipelineschedules.list()
```

Get a single schedule:

```
sched = projects.pipelineschedules.get(schedule_id)
```

Create a new schedule:

```
sched = project.pipelineschedules.create({
    'ref': 'master',
    'description': 'Daily test',
    'cron': '0 1 * * *'})
```

Update a schedule:

```
sched.cron = '1 2 * * *'
sched.save()
```

Delete a schedule:

```
sched.delete()
```

Create a schedule variable:

```
var = sched.variables.create({'key': 'foo', 'value': 'bar'})
```

Edit a schedule variable:

```
var.value = 'new_value'
var.save()
```

Delete a schedule variable:

```
var.delete()
```

5.6.4 Projects and groups variables

You can associate variables to projects and groups to modify the build/job scripts behavior.

Reference

- v4 API
  - *gitlab.v4.objects.ProjectVariable*
  - *gitlab.v4.objects.ProjectVariableManager*
  - gitlab.v4.objects.Project.variables
  - *gitlab.v4.objects.GroupVariable*
  - *gitlab.v4.objects.GroupVariableManager*
  - gitlab.v4.objects.Group.variables
- v3 API
  - *gitlab.v3.objects.ProjectVariable*
  - *gitlab.v3.objects.ProjectVariableManager*
  - *gitlab.v3.objects.Project.variables*
  - gitlab.Gitlab.project_variables
- GitLab API
  - https://docs.gitlab.com/ce/api/project_level_variables.html
  - https://docs.gitlab.com/ce/api/group_level_variables.html

Examples

List variables:

```
p_variables = project.variables.list()
g_variables = group.variables.list()
```

Get a variable:

```
p_var = project.variables.get('key_name')
g_var = group.variables.get('key_name')
```

Create a variable:

```
var = project.variables.create({'key': 'key1', 'value': 'value1'})
var = group.variables.create({'key': 'key1', 'value': 'value1'})
```

Update a variable value:

```
var.value = 'new_value'
var.save()
```

Remove a variable:

```
project.variables.delete('key_name')
group.variables.delete('key_name')
# or
var.delete()
```

5.6.5 Builds/Jobs

Builds/Jobs are associated to projects, pipelines and commits. They provide information on the builds/jobs that have been run, and methods to manipulate them.

Reference

- v4 API
  - *gitlab.v4.objects.ProjectJob*
  - *gitlab.v4.objects.ProjectJobManager*
  - gitlab.v4.objects.Project.jobs
- v3 API
  - gitlab.v3.objects.ProjectJob
  - gitlab.v3.objects.ProjectJobManager
  - gitlab.v3.objects.Project.jobs
  - gitlab.Gitlab.project_jobs
- GitLab API: https://docs.gitlab.com/ce/api/jobs.html

Examples

Jobs are usually automatically triggered, but you can explicitly trigger a new job:

Trigger a new job on a project:

```
project.trigger_build('master', trigger_token,
                      {'extra_var1': 'foo', 'extra_var2': 'bar'})
```

List jobs for the project:

```
builds = project.builds.list()  # v3
jobs = project.jobs.list()  # v4
```

To list builds for a specific commit, create a *ProjectCommit* object and use its *builds* method (v3 only):

```
# v3 only
commit = gl.project_commits.get(commit_sha, project_id=1)
builds = commit.builds()
```

To list builds for a specific pipeline or get a single job within a specific pipeline, create a *ProjectPipeline* object and use its `jobs` method (v4 only):

```
# v4 only
project = gl.projects.get(project_id)
pipeline = project.pipelines.get(pipeline_id)
jobs = pipeline.jobs.list()  # gets all jobs in pipeline
job = pipeline.jobs.get(job_id)  # gets one job from pipeline
```

Get a job:

```
project.builds.get(build_id)  # v3
project.jobs.get(job_id)  # v4
```

Get a job artifact:

```
build_or_job.artifacts()
```

Warning: Artifacts are entirely stored in memory in this example.

You can download artifacts as a stream. Provide a callable to handle the stream:

```
class Foo(object):
    def __init__(self):
        self._fd = open('artifacts.zip', 'wb')

    def __call__(self, chunk):
        self._fd.write(chunk)

target = Foo()
build_or_job.artifacts(streamed=True, action=target)
del(target)  # flushes data on disk
```

In this second example, you can directly stream the output into a file, and unzip it afterwards:

```
zipfn = "___artifacts.zip"
with open(zipfn, "wb") as f:
    build_or_job.artifacts(streamed=True, action=f.write)
subprocess.run(["unzip", "-bo", zipfn])
os.unlink(zipfn)
```

Mark a job artifact as kept when expiration is set:

```
build_or_job.keep_artifacts()
```

Get a job trace:

```
build_or_job.trace()
```

Warning: Traces are entirely stored in memory unless you use the streaming feature. See *the artifacts example*.

Cancel/retry a job:

```
build_or_job.cancel()
build_or_job.retry()
```

Play (trigger) a job:

```
build_or_job.play()
```

Erase a job (artifacts and trace):

```
build_or_job.erase()
```

5.7 Commits

5.7.1 Commits

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectCommit*
  - *gitlab.v4.objects.ProjectCommitManager*
  - gitlab.v4.objects.Project.commits
- v3 API:
  - *gitlab.v3.objects.ProjectCommit*
  - *gitlab.v3.objects.ProjectCommitManager*
  - *gitlab.v3.objects.Project.commits*
  - gitlab.Gitlab.project_commits
- GitLab API: https://docs.gitlab.com/ce/api/commits.html

Warning: Pagination starts at page 0 in v3, but starts at page 1 in v4 (like all the v4 endpoints).

Examples

List the commits for a project:

```
commits = project.commits.list()
```

You can use the `ref_name`, `since` and `until` filters to limit the results:

```
commits = project.commits.list(ref_name='my_branch')
commits = project.commits.list(since='2016-01-01T00:00:00Z')
```

Create a commit:

```
# See https://docs.gitlab.com/ce/api/commits.html#create-a-commit-with-multiple-files-
↪and-actions
# for actions detail
data = {
    'branch_name': 'master',  # v3
    'branch': 'master',  # v4
    'commit_message': 'blah blah blah',
    'actions': [
        {
            'action': 'create',
            'file_path': 'blah',
            'content': 'blah'
        }
    ]
}

commit = project.commits.create(data)
```

Get a commit detail:

```
commit = project.commits.get('e3d5a71b')
```

Get the diff for a commit:

```
diff = commit.diff()
```

Cherry-pick a commit into another branch:

```
commit.cherry_pick(branch='target_branch')
```

5.7.2 Commit comments

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectCommitComment*
  - *gitlab.v4.objects.ProjectCommitCommentManager*
  - gitlab.v4.objects.Commit.comments
- v3 API:
  - *gitlab.v3.objects.ProjectCommit*
  - *gitlab.v3.objects.ProjectCommitManager*
  - gitlab.v3.objects.Commit.comments
  - gitlab.v3.objects.Project.commit_comments
  - gitlab.Gitlab.project_commit_comments
- GitLab API: https://docs.gitlab.com/ce/api/commits.html

Examples

Get the comments for a commit:

```
comments = commit.comments.list()
```

Add a comment on a commit:

```
# Global comment
commit = commit.comments.create({'note': 'This is a nice comment'})
# Comment on a line in a file (on the new version of the file)
commit = commit.comments.create({'note': 'This is another comment',
                                 'line': 12,
                                 'line_type': 'new',
                                 'path': 'README.rst'})
```

5.7.3 Commit status

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectCommitStatus*
  - *gitlab.v4.objects.ProjectCommitStatusManager*
  - gitlab.v4.objects.Commit.statuses
- v3 API:
  - *gitlab.v3.objects.ProjectCommit*
  - *gitlab.v3.objects.ProjectCommitManager*
  - gitlab.v3.objects.Commit.statuses
  - gitlab.v3.objects.Project.commit_statuses
  - gitlab.Gitlab.project_commit_statuses
- GitLab API: https://docs.gitlab.com/ce/api/commits.html

Examples

Get the statuses for a commit:

```
statuses = commit.statuses.list()
```

Change the status of a commit:

```
commit.statuses.create({'state': 'success'})
```

5.8 Deploy keys

5.8.1 Deploy keys

Reference

- v4 API:
  - *gitlab.v4.objects.DeployKey*
  - *gitlab.v4.objects.DeployKeyManager*
  - gitlab.Gitlab.deploykeys
- v3 API:
  - *gitlab.v3.objects.DeployKey*
  - *gitlab.v3.objects.DeployKeyManager*
  - gitlab.Gitlab.deploykeys
- GitLab API: https://docs.gitlab.com/ce/api/deploy_keys.html

Examples

List the deploy keys:

```
keys = gl.deploykeys.list()
```

Get a single deploy key:

```
key = gl.deploykeys.get(key_id)
```

5.8.2 Deploy keys for projects

Deploy keys can be managed on a per-project basis.

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectKey*
  - *gitlab.v4.objects.ProjectKeyManager*
  - gitlab.v4.objects.Project.keys
- v3 API:
  - *gitlab.v3.objects.ProjectKey*
  - *gitlab.v3.objects.ProjectKeyManager*
  - *gitlab.v3.objects.Project.keys*
  - gitlab.Gitlab.project_keys
- GitLab API: https://docs.gitlab.com/ce/api/deploy_keys.html

Examples

List keys for a project:

```
keys = project.keys.list()
```

Get a single deploy key:

```
key = project.keys.get(key_id)
```

Create a deploy key for a project:

```
key = project.keys.create({'title': 'jenkins key',
                           'key': open('/home/me/.ssh/id_rsa.pub').read()})
```

Delete a deploy key for a project:

```
key = project.keys.list(key_id)
# or
key.delete()
```

Enable a deploy key for a project:

```
project.keys.enable(key_id)
```

Disable a deploy key for a project:

```
project_key.delete()  # v4
project.keys.disable(key_id)  # v3
```

5.9 Deployments

5.9.1 Reference

- v4 API:
  - *gitlab.v4.objects.ProjectDeployment*
  - *gitlab.v4.objects.ProjectDeploymentManager*
  - gitlab.v4.objects.Project.deployments
- v3 API:
  - *gitlab.v3.objects.ProjectDeployment*
  - *gitlab.v3.objects.ProjectDeploymentManager*
  - *gitlab.v3.objects.Project.deployments*
  - gitlab.Gitlab.project_deployments
- GitLab API: https://docs.gitlab.com/ce/api/deployments.html

5.9.2 Examples

List deployments for a project:

```
deployments = project.deployments.list()
```

Get a single deployment:

```
deployment = project.deployments.get(deployment_id)
```

5.10 Environments

5.10.1 Reference

- v4 API:
  - *gitlab.v4.objects.ProjectEnvironment*
  - *gitlab.v4.objects.ProjectEnvironmentManager*
  - gitlab.v4.objects.Project.environments
- v3 API:
  - *gitlab.v3.objects.ProjectEnvironment*
  - *gitlab.v3.objects.ProjectEnvironmentManager*
  - *gitlab.v3.objects.Project.environments*
  - gitlab.Gitlab.project_environments
- GitLab API: https://docs.gitlab.com/ce/api/environments.html

5.10.2 Examples

List environments for a project:

```
environments = project.environments.list()
```

Get a single environment:

```
environment = project.environments.get(environment_id)
```

Create an environment for a project:

```
environment = project.environments.create({'name': 'production'})
```

Update an environment for a project:

```
environment.external_url = 'http://foo.bar.com'
environment.save()
```

Delete an environment for a project:

```
environment = project.environments.delete(environment_id)
# or
environment.delete()
```

5.11 Events

5.11.1 Reference

- v4 API:
  - *gitlab.v4.objects.Event*
  - *gitlab.v4.objects.EventManager*
  - gitlab.Gitlab.events
  - *gitlab.v4.objects.ProjectEvent*
  - *gitlab.v4.objects.ProjectEventManager*
  - gitlab.v4.objects.Project.events
  - *gitlab.v4.objects.UserEvent*
  - *gitlab.v4.objects.UserEventManager*
  - gitlab.v4.objects.User.events
- v3 API (projects events only):
  - *gitlab.v3.objects.ProjectEvent*
  - *gitlab.v3.objects.ProjectEventManager*
  - *gitlab.v3.objects.Project.events*
  - gitlab.Gitlab.project_events
- GitLab API: https://docs.gitlab.com/ce/api/events.html

5.11.2 Examples

You can list events for an entire Gitlab instance (admin), users and projects. You can filter you events you want to retrieve using the `action` and `target_type` attributes. The possibole values for these attributes are available on the gitlab documentation.

List all the events (paginated):

```
events = gl.events.list()
```

List the issue events on a project:

```
events = project.events.list(target_type='issue')
```

List the user events:

```
events = project.events.list()
```

5.12 Features flags

5.12.1 Reference

- v4 API:
  - *gitlab.v4.objects.Feature*
  - *gitlab.v4.objects.FeatureManager*
  - gitlab.Gitlab.features
- GitLab API: https://docs.gitlab.com/ce/api/features.html

5.12.2 Examples

List features:

```
features = gl.features.list()
```

Create or set a feature:

```
feature = gl.features.set(feature_name, True)
feature = gl.features.set(feature_name, 30)
```

5.13 Groups

5.13.1 Groups

Reference

- v4 API:
  - *gitlab.v4.objects.Group*
  - *gitlab.v4.objects.GroupManager*
  - gitlab.Gitlab.groups
- v3 API:
  - *gitlab.v3.objects.Group*
  - *gitlab.v3.objects.GroupManager*
  - gitlab.Gitlab.groups
- GitLab API: https://docs.gitlab.com/ce/api/groups.html

Examples

List the groups:

```
groups = gl.groups.list()
```

Get a group's detail:

```
group = gl.groups.get(group_id)
```

List a group's projects:

```
projects = group.projects.list()
```

You can filter and sort the result using the following parameters:

- `archived`: limit by archived status
- `visibility`: limit by visibility. Allowed values are `public`, `internal` and `private`
- `search`: limit to groups matching the given value
- `order_by`: sort by criteria. Allowed values are `id`, `name`, `path`, `created_at`, `updated_at` and `last_activity_at`
- `sort`: sort order: `asc` or `desc`
- `ci_enabled_first`: return CI enabled groups first

Create a group:

```
group = gl.groups.create({'name': 'group1', 'path': 'group1'})
```

Update a group:

```
group.description = 'My awesome group'
group.save()
```

Remove a group:

```
gl.group.delete(group_id)
# or
group.delete()
```

5.13.2 Subgroups

Reference

- v4 API:
  - *gitlab.v4.objects.GroupSubgroup*
  - *gitlab.v4.objects.GroupSubgroupManager*
  - gitlab.v4.objects.Group.subgroups

Examples

List the subgroups for a group:

```
subgroups = group.subgroups.list()

# The GroupSubgroup objects don't expose the same API as the Group
# objects. If you need to manipulate a subgroup as a group, create a new
# Group object:
```

```
real_group = gl.groups.get(subgroup_id, lazy=True)
real_group.issues.list()
```

5.13.3 Group custom attributes

Reference

- v4 API:
  - *gitlab.v4.objects.GroupCustomAttribute*
  - *gitlab.v4.objects.GroupCustomAttributeManager*
  - gitlab.v4.objects.Group.customattributes
- GitLab API: https://docs.gitlab.com/ce/api/custom_attributes.html

Examples

List custom attributes for a group:

```
attrs = group.customattributes.list()
```

Get a custom attribute for a group:

```
attr = group.customattributes.get(attr_key)
```

Set (create or update) a custom attribute for a group:

```
attr = group.customattributes.set(attr_key, attr_value)
```

Delete a custom attribute for a group:

```
attr.delete()
# or
group.customattributes.delete(attr_key)
```

Search groups by custom attribute:

```
group.customattributes.set('role': 'admin')
gl.groups.list(custom_attributes={'role': 'admin'})
```

5.13.4 Group members

The following constants define the supported access levels:

- gitlab.GUEST_ACCESS = 10
- gitlab.REPORTER_ACCESS = 20
- gitlab.DEVELOPER_ACCESS = 30
- gitlab.MASTER_ACCESS = 40
- gitlab.OWNER_ACCESS = 50

Reference

- v4 API:
  - *gitlab.v4.objects.GroupMember*
  - *gitlab.v4.objects.GroupMemberManager*
  - gitlab.v4.objects.Group.members
- v3 API:
  - *gitlab.v3.objects.GroupMember*
  - *gitlab.v3.objects.GroupMemberManager*
  - *gitlab.v3.objects.Group.members*
  - gitlab.Gitlab.group_members
- GitLab API: https://docs.gitlab.com/ce/api/groups.html

Examples

List group members:

```
members = group.members.list()
```

Get a group member:

```
members = group.members.get(member_id)
```

Add a member to the group:

```
member = group.members.create({'user_id': user_id,
                               'access_level': gitlab.GUEST_ACCESS})
```

Update a member (change the access level):

```
member.access_level = gitlab.DEVELOPER_ACCESS
member.save()
```

Remove a member from the group:

```
group.members.delete(member_id)
# or
member.delete()
```

5.14 Issues

5.14.1 Reported issues

Reference

- v4 API:
  - *gitlab.v4.objects.Issue*

  - *gitlab.v4.objects.IssueManager*
  - gitlab.Gitlab.issues
- v3 API:
  - *gitlab.v3.objects.Issue*
  - *gitlab.v3.objects.IssueManager*
  - gitlab.Gitlab.issues
- GitLab API: https://docs.gitlab.com/ce/api/issues.html

Examples

List the issues:

```
issues = gl.issues.list()
```

Use the `state` and `label` parameters to filter the results. Use the `order_by` and `sort` attributes to sort the results:

```
open_issues = gl.issues.list(state='opened')
closed_issues = gl.issues.list(state='closed')
tagged_issues = gl.issues.list(labels=['foo', 'bar'])
```

5.14.2 Group issues

Reference

- v4 API:
  - *gitlab.v4.objects.GroupIssue*
  - *gitlab.v4.objects.GroupIssueManager*
  - gitlab.v4.objects.Group.issues
- v3 API:
  - *gitlab.v3.objects.GroupIssue*
  - *gitlab.v3.objects.GroupIssueManager*
  - *gitlab.v3.objects.Group.issues*
  - gitlab.Gitlab.group_issues
- GitLab API: https://docs.gitlab.com/ce/api/issues.html

Examples

List the group issues:

```
issues = group.issues.list()
# Filter using the state, labels and milestone parameters
issues = group.issues.list(milestone='1.0', state='opened')
# Order using the order_by and sort parameters
issues = group.issues.list(order_by='created_at', sort='desc')
```

5.14.3 Project issues

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectIssue*
  - *gitlab.v4.objects.ProjectIssueManager*
  - gitlab.v4.objects.Project.issues
- v3 API:
  - *gitlab.v3.objects.ProjectIssue*
  - *gitlab.v3.objects.ProjectIssueManager*
  - *gitlab.v3.objects.Project.issues*
  - gitlab.Gitlab.project_issues
- GitLab API: https://docs.gitlab.com/ce/api/issues.html

Examples

List the project issues:

```
issues = project.issues.list()
# Filter using the state, labels and milestone parameters
issues = project.issues.list(milestone='1.0', state='opened')
# Order using the order_by and sort parameters
issues = project.issues.list(order_by='created_at', sort='desc')
```

Get a project issue:

```
issue = project.issues.get(issue_id)
```

Get a project issue from its *iid* (v3 only. Issues are retrieved by iid in V4 by default):

```
issue = project.issues.list(iid=issue_iid)[0]
```

Create a new issue:

```
issue = project.issues.create({'title': 'I have a bug',
                               'description': 'Something useful here.'})
```

Update an issue:

```
issue.labels = ['foo', 'bar']
issue.save()
```

Close / reopen an issue:

```
# close an issue
issue.state_event = 'close'
issue.save()
# reopen it
issue.state_event = 'reopen'
issue.save()
```

Delete an issue:

```
project.issues.delete(issue_id)
# pr
issue.delete()
```

Subscribe / unsubscribe from an issue:

```
issue.subscribe()
issue.unsubscribe()
```

Move an issue to another project:

```
issue.move(new_project_id)
```

Make an issue as todo:

```
issue.todo()
```

Get time tracking stats:

```
issue.time_stats()
```

Set a time estimate for an issue:

```
issue.set_time_estimate({'duration': '3h30m'})
```

Reset a time estimate for an issue:

```
issue.reset_time_estimate()
```

Add spent time for an issue:

```
issue.add_time_spent({'duration': '3h30m'})
```

Reset spent time for an issue:

```
issue.reset_time_spent()
```

Get user agent detail for the issue (admin only):

```
detail = issue.user_agent_detail()
```

5.15 Labels

5.15.1 Reference

- v4 API:
  - *gitlab.v4.objects.ProjectLabel*
  - *gitlab.v4.objects.ProjectLabelManager*
  - gitlab.v4.objects.Project.labels
- v3 API:

  - *gitlab.v3.objects.ProjectLabel*
  - *gitlab.v3.objects.ProjectLabelManager*
  - *gitlab.v3.objects.Project.labels*
  - gitlab.Gitlab.project_labels
- GitLab API: https://docs.gitlab.com/ce/api/labels.html

5.15.2 Examples

List labels for a project:

```
labels = project.labels.list()
```

Get a single label:

```
label = project.labels.get(label_name)
```

Create a label for a project:

```
label = project.labels.create({'name': 'foo', 'color': '#8899aa'})
```

Update a label for a project:

```
# change the name of the label:
label.new_name = 'bar'
label.save()
# change its color:
label.color = '#112233'
label.save()
```

Delete a label for a project:

```
project.labels.delete(label_id)
# or
label.delete()
```

Managing labels in issues and merge requests:

```
# Labels are defined as lists in issues and merge requests. The labels must
# exist.
issue = p.issues.create({'title': 'issue title',
                         'description': 'issue description',
                         'labels': ['foo']})
issue.labels.append('bar')
issue.save()
```

5.16 Notification settings

You can define notification settings globally, for groups and for projects. Valid levels are defined as constants:

- gitlab.NOTIFICATION_LEVEL_DISABLED
- gitlab.NOTIFICATION_LEVEL_PARTICIPATING

- gitlab.NOTIFICATION_LEVEL_WATCH
- gitlab.NOTIFICATION_LEVEL_GLOBAL
- gitlab.NOTIFICATION_LEVEL_MENTION
- gitlab.NOTIFICATION_LEVEL_CUSTOM

You get access to fine-grained settings if you use the NOTIFICATION_LEVEL_CUSTOM level.

5.16.1 Reference

- v4 API:
  - *gitlab.v4.objects.NotificationSettings*
  - *gitlab.v4.objects.NotificationSettingsManager*
  - gitlab.Gitlab.notificationsettings
  - *gitlab.v4.objects.GroupNotificationSettings*
  - *gitlab.v4.objects.GroupNotificationSettingsManager*
  - gitlab.v4.objects.Group.notificationsettings
  - *gitlab.v4.objects.ProjectNotificationSettings*
  - *gitlab.v4.objects.ProjectNotificationSettingsManager*
  - gitlab.v4.objects.Project.notificationsettings
- v3 API:
  - *gitlab.v3.objects.NotificationSettings*
  - *gitlab.v3.objects.NotificationSettingsManager*
  - gitlab.Gitlab.notificationsettings
  - *gitlab.v3.objects.GroupNotificationSettings*
  - *gitlab.v3.objects.GroupNotificationSettingsManager*
  - *gitlab.v3.objects.Group.notificationsettings*
  - *gitlab.v3.objects.ProjectNotificationSettings*
  - *gitlab.v3.objects.ProjectNotificationSettingsManager*
  - *gitlab.v3.objects.Project.notificationsettings*
- GitLab API: https://docs.gitlab.com/ce/api/notification_settings.html

5.16.2 Examples

Get the settings:

```
# global settings
settings = gl.notificationsettings.get()
# for a group
settings = gl.groups.get(group_id).notificationsettings.get()
# for a project
settings = gl.projects.get(project_id).notificationsettings.get()
```

Update the settings:

```
# use a predefined level
settings.level = gitlab.NOTIFICATION_LEVEL_WATCH
# create a custom setup
settings.level = gitlab.NOTIFICATION_LEVEL_CUSTOM
settings.save()  # will create additional attributes, but not mandatory

settings.new_merge_request = True
settings.new_issue = True
settings.new_note = True
settings.save()
```

5.17 Merge requests

You can use merge requests to notify a project that a branch is ready for merging. The owner of the target projet can accept the merge request.

The v3 API uses the `id` attribute to identify a merge request, the v4 API uses the `iid` attribute.

5.17.1 Reference

- v4 API:
  - *gitlab.v4.objects.ProjectMergeRequest*
  - *gitlab.v4.objects.ProjectMergeRequestManager*
  - gitlab.v4.objects.Project.mergerequests
- v3 API:
  - *gitlab.v3.objects.ProjectMergeRequest*
  - *gitlab.v3.objects.ProjectMergeRequestManager*
  - *gitlab.v3.objects.Project.mergerequests*
  - gitlab.Gitlab.project_mergerequests
- GitLab API: https://docs.gitlab.com/ce/api/merge_requests.html

5.17.2 Examples

List MRs for a project:

```
mrs = project.mergerequests.list()
```

You can filter and sort the returned list with the following parameters:

- `iid`: iid (unique ID for the project) of the MR (v3 API)
- `state`: state of the MR. It can be one of `all`, `merged`, `opened` or `closed`
- `order_by`: sort by `created_at` or `updated_at`
- `sort`: sort order (`asc` or `desc`)

For example:

```
mrs = project.mergerequests.list()
```

Get a single MR:

```
mr = project.mergerequests.get(mr_id)
```

Create a MR:

```
mr = project.mergerequests.create({'source_branch': 'cool_feature',
                                   'target_branch': 'master',
                                   'title': 'merge cool feature',
                                   'labels': ['label1', 'label2']})
```

Update a MR:

```
mr.description = 'New description'
mr.labels = ['foo', 'bar']
mr.save()
```

Change the state of a MR (close or reopen):

```
mr.state_event = 'close'  # or 'reopen'
mr.save()
```

Delete a MR:

```
project.mergerequests.delete(mr_id)
# or
mr.delete()
```

Accept a MR:

```
mr.merge()
```

Cancel a MR when the build succeeds:

```
mr.cancel_merge_when_build_succeeds()  # v3
mr.cancel_merge_when_pipeline_succeeds()  # v4
```

List issues that will close on merge:

```
mr.closes_issues()
```

Subscribe/unsubscribe a MR:

```
mr.subscribe()
mr.unsubscribe()
```

Mark a MR as todo:

```
mr.todo()
```

List the diffs for a merge request:

```
diffs = mr.diffs.list()
```

Get a diff for a merge request:

```
diff = mr.diffs.get(diff_id)
```

5.18 Namespaces

5.18.1 Reference

- v4 API:
  - *gitlab.v4.objects.Namespace*
  - *gitlab.v4.objects.NamespaceManager*
  - gitlab.Gitlab.namespaces
- v3 API:
  - *gitlab.v3.objects.Namespace*
  - *gitlab.v3.objects.NamespaceManager*
  - gitlab.Gitlab.namespaces
- GitLab API: https://docs.gitlab.com/ce/api/namespaces.html

5.18.2 Examples

List namespaces:

```
namespaces = gl.namespaces.list()
```

Search namespaces:

```
namespaces = gl.namespaces.list(search='foo')
```

5.19 Milestones

5.19.1 Reference

- v4 API:
  - *gitlab.v4.objects.ProjectMilestone*
  - *gitlab.v4.objects.ProjectMilestoneManager*
  - gitlab.v4.objects.Project.milestones
  - *gitlab.v4.objects.GroupMilestone*
  - *gitlab.v4.objects.GroupMilestoneManager*
  - gitlab.v4.objects.Group.milestones
- v3 API:
  - *gitlab.v3.objects.ProjectMilestone*

  - *gitlab.v3.objects.ProjectMilestoneManager*
  - *gitlab.v3.objects.Project.milestones*
  - gitlab.Gitlab.project_milestones
- GitLab API:
  - https://docs.gitlab.com/ce/api/milestones.html
  - https://docs.gitlab.com/ce/api/group_milestones.html

5.19.2 Examples

List the milestones for a project or a group:

```
p_milestones = project.milestones.list()
g_milestones = group.milestones.list()
```

You can filter the list using the following parameters:

- `iid`: unique ID of the milestone for the project
- `state`: either `active` or `closed`
- `search`: to search using a string

```
p_milestones = project.milestones.list(state='closed')
g_milestones = group.milestones.list(state='active')
```

Get a single milestone:

```
p_milestone = project.milestones.get(milestone_id)
g_milestone = group.milestones.get(milestone_id)
```

Create a milestone:

```
milestone = project.milestones.create({'title': '1.0'})
```

Edit a milestone:

```
milestone.description = 'v 1.0 release'
milestone.save()
```

Change the state of a milestone (activate / close):

```
# close a milestone
milestone.state_event = 'close'
milestone.save()

# activate a milestone
milestone.state_event = 'activate'
milestone.save()
```

List the issues related to a milestone:

```
issues = milestone.issues()
```

List the merge requests related to a milestone:

```
merge_requests = milestone.merge_requests()
```

5.20 Pages domains

5.20.1 Admin

References

- v4 API:
  - *gitlab.v4.objects.PagesDomain*
  - *gitlab.v4.objects.PagesDomainManager*
  - gitlab.Gitlab.pagesdomains
- GitLab API: https://docs.gitlab.com/ce/api/pages_domains.html#list-all-pages-domains

Examples

List all the existing domains (admin only):

```
domains = gl.pagesdomains.list()
```

5.20.2 Project pages domain

References

- v4 API:
  - *gitlab.v4.objects.ProjectPagesDomain*
  - *gitlab.v4.objects.ProjectPagesDomainManager*
  - gitlab.v4.objects.Project.pagesdomains
- GitLab API: https://docs.gitlab.com/ce/api/pages_domains.html#list-pages-domains

Examples

List domains for a project:

```
domains = project.pagesdomains.list()
```

Get a single domain:

```
domain = project.pagesdomains.get('d1.example.com')
```

Create a new domain:

```
domain = project.pagesdomains.create({'domain': 'd2.example.com})
```

Update an existing domain:

```
domain.certificate = open('d2.crt').read()
domain.key = open('d2.key').read()
domain.save()
```

Delete an existing domain:

```
domain.delete
# or
project.pagesdomains.delete('d2.example.com')
```

5.21 Projects

5.21.1 Projects

Reference

- v4 API:
  - *gitlab.v4.objects.Project*
  - *gitlab.v4.objects.ProjectManager*
  - gitlab.Gitlab.projects
- v3 API:
  - *gitlab.v3.objects.Project*
  - *gitlab.v3.objects.ProjectManager*
  - gitlab.Gitlab.projects
- GitLab API: https://docs.gitlab.com/ce/api/projects.html

Examples

List projects:

The API provides several filtering parameters for the listing methods:

- `archived`: if `True` only archived projects will be returned
- `visibility`: returns only projects with the specified visibility (can be `public`, `internal` or `private`)
- `search`: returns project matching the given pattern

Results can also be sorted using the following parameters:

- `order_by`: sort using the given argument. Valid values are `id`, `name`, `path`, `created_at`, `updated_at` and `last_activity_at`. The default is to sort by `created_at`
- `sort`: sort order (`asc` or `desc`)

```
# Active projects
projects = gl.projects.list()
# Archived projects
projects = gl.projects.list(archived=1)
# Limit to projects with a defined visibility
projects = gl.projects.list(visibility='public')
```

```
# List owned projects
projects = gl.projects.owned()

# List starred projects
projects = gl.projects.starred()

# List all the projects
projects = gl.projects.all()

# Search projects
projects = gl.projects.list(search='keyword')
```

Get a single project:

```
# Get a project by ID
project = gl.projects.get(10)
# Get a project by userspace/name
project = gl.projects.get('myteam/myproject')
```

Create a project:

```
project = gl.projects.create({'name': 'project1'})
```

Create a project for a user (admin only):

```
alice = gl.users.list(username='alice')[0]
user_project = alice.projects.create({'name': 'project'})
user_projects = alice.projects.list()
```

Create a project in a group:

You need to get the id of the group, then use the namespace_id attribute to create the group:

```
group_id = gl.groups.search('my-group')[0].id
project = gl.projects.create({'name': 'myrepo', 'namespace_id': group_id})
```

Update a project:

```
project.snippets_enabled = 1
project.save()
```

Delete a project:

```
gl.projects.delete(1)
# or
project.delete()
```

Fork a project:

```
fork = project.forks.create({})

# fork to a specific namespace
fork = project.forks.create({'namespace': 'myteam'})
```

Create/delete a fork relation between projects (requires admin permissions):

```
project.create_fork_relation(source_project.id)
project.delete_fork_relation()
```

Star/unstar a project:

```
project.star()
project.unstar()
```

Archive/unarchive a project:

```
project.archive()
project.unarchive()
```

Note: Previous versions used `archive_` and `unarchive_` due to a naming issue, they have been deprecated but not yet removed.

Start the housekeeping job:

```
project.housekeeping()
```

List the repository tree:

```
# list the content of the root directory for the default branch
items = project.repository_tree()

# list the content of a subdirectory on a specific branch
items = project.repository_tree(path='docs', ref='branch1')
```

Get the content and metadata of a file for a commit, using a blob sha:

```
items = project.repository_tree(path='docs', ref='branch1')
file_info = p.repository_blob(items[0]['id'])
content = base64.b64decode(file_info['content'])
size = file_info['size']
```

Get the repository archive:

```
# get the archive for the default branch
tgz = project.repository_archive()

# get the archive for a branch/tag/commit
tgz = project.repository_archive(sha='4567abc')
```

Warning: Archives are entirely stored in memory unless you use the streaming feature. See *the artifacts example*.

Get the content of a file using the blob id:

```
# find the id for the blob (simple search)
id = [d['id'] for d in p.repository_tree() if d['name'] == 'README.rst'][0]

# get the content
file_content = p.repository_raw_blob(id)
```

Warning: Blobs are entirely stored in memory unless you use the streaming feature. See *the artifacts example*.

Compare two branches, tags or commits:

```
result = project.repository_compare('master', 'branch1')

# get the commits
for commit in result['commits']:
    print(commit)

# get the diffs
for file_diff in result['diffs']:
    print(file_diff)
```

Get a list of contributors for the repository:

```
contributors = project.repository_contributors()
```

Get a list of users for the repository:

```
users = p.users.list()

# search for users
users = p.users.list(search='pattern')
```

5.21.2 Project custom attributes

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectCustomAttribute*
  - *gitlab.v4.objects.ProjectCustomAttributeManager*
  - gitlab.v4.objects.Project.customattributes
- GitLab API: https://docs.gitlab.com/ce/api/custom_attributes.html

Examples

List custom attributes for a project:

```
attrs = project.customattributes.list()
```

Get a custom attribute for a project:

```
attr = project.customattributes.get(attr_key)
```

Set (create or update) a custom attribute for a project:

```
attr = project.customattributes.set(attr_key, attr_value)
```

Delete a custom attribute for a project:

```
attr.delete()
# or
project.customattributes.delete(attr_key)
```

Search projects by custom attribute:

```
project.customattributes.set('type': 'internal')
gl.projects.list(custom_attributes={'type': 'internal'})
```

5.21.3 Project files

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectFile*
  - *gitlab.v4.objects.ProjectFileManager*
  - gitlab.v4.objects.Project.files
- v3 API:
  - *gitlab.v3.objects.ProjectFile*
  - *gitlab.v3.objects.ProjectFileManager*
  - *gitlab.v3.objects.Project.files*
  - gitlab.Gitlab.project_files
- GitLab API: https://docs.gitlab.com/ce/api/repository_files.html

Examples

Get a file:

```
f = project.files.get(file_path='README.rst', ref='master')

# get the base64 encoded content
print(f.content)

# get the decoded content
print(f.decode())
```

Create a new file:

```
# v4
f = project.files.create({'file_path': 'testfile',
                          'branch': 'master',
                          'content': file_content,
                          'commit_message': 'Create testfile'})

# v3
f = project.files.create({'file_path': 'testfile',
                          'branch_name': 'master',
                          'content': file_content,
                          'commit_message': 'Create testfile'})
```

Update a file. The entire content must be uploaded, as plain text or as base64 encoded text:

```
f.content = 'new content'
f.save(branch='master', commit_message='Update testfile')  # v4
f.save(branch_name='master', commit_message='Update testfile')  # v3

# or for binary data
# Note: decode() is required with python 3 for data serialization. You can omit
# it with python 2
f.content = base64.b64encode(open('image.png').read()).decode()
f.save(branch='master', commit_message='Update testfile', encoding='base64')
```

Delete a file:

```
f.delete(commit_message='Delete testfile')
```

5.21.4 Project tags

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectTag*
  - *gitlab.v4.objects.ProjectTagManager*
  - gitlab.v4.objects.Project.tags
- v3 API:
  - *gitlab.v3.objects.ProjectTag*
  - *gitlab.v3.objects.ProjectTagManager*
  - *gitlab.v3.objects.Project.tags*
  - gitlab.Gitlab.project_tags
- GitLab API: https://docs.gitlab.com/ce/api/tags.html

Examples

List the project tags:

```
tags = project.tags.list()
```

Get a tag:

```
tag = project.tags.get('1.0')
```

Create a tag:

```
tag = project.tags.create({'tag_name': '1.0', 'ref': 'master'})
```

Set or update the release note for a tag:

```
tag.set_release_description('awesome v1.0 release')
```

Delete a tag:

```
project.tags.delete('1.0')
# or
tag.delete()
```

5.21.5 Project snippets

The snippet visibility can be definied using the following constants:

- `gitlab.VISIBILITY_PRIVATE`
- `gitlab.VISIBILITY_INTERNAL`
- `gitlab.VISIBILITY_PUBLIC`

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectSnippet*
  - *gitlab.v4.objects.ProjectSnippetManager*
  - `gitlab.v4.objects.Project.files`
- v3 API:
  - *gitlab.v3.objects.ProjectSnippet*
  - *gitlab.v3.objects.ProjectSnippetManager*
  - *gitlab.v3.objects.Project.files*
  - `gitlab.Gitlab.project_files`
- GitLab API: https://docs.gitlab.com/ce/api/project_snippets.html

Examples

List the project snippets:

```
snippets = project.snippets.list()
```

Get a snippet:

```
snippets = project.snippets.list(snippet_id)
```

Get the content of a snippet:

```
print(snippet.content())
```

Warning: The snippet content is entirely stored in memory unless you use the streaming feature. See *the artifacts example*.

Create a snippet:

```
snippet = project.snippets.create({'title': 'sample 1',
                                   'file_name': 'foo.py',
                                   'code': 'import gitlab',
                                   'visibility_level':
                                   gitlab.VISIBILITY_PRIVATE})
```

Update a snippet:

```
snippet.code = 'import gitlab\nimport whatever'
snippet.save
```

Delete a snippet:

```
project.snippets.delete(snippet_id)
# or
snippet.delete()
```

5.21.6 Notes

You can manipulate notes (comments) on the issues, merge requests and snippets.

- `ProjectIssue` with `ProjectIssueNote`
- `ProjectMergeRequest` with `ProjectMergeRequestNote`
- `ProjectSnippet` with `ProjectSnippetNote`

Reference

- v4 API:

  Issues:

  - *gitlab.v4.objects.ProjectIssueNote*
  - *gitlab.v4.objects.ProjectIssueNoteManager*
  - `gitlab.v4.objects.ProjectIssue.notes`

  MergeRequests:

  - *gitlab.v4.objects.ProjectMergeRequestNote*
  - *gitlab.v4.objects.ProjectMergeRequestNoteManager*
  - `gitlab.v4.objects.ProjectMergeRequest.notes`

  Snippets:

  - *gitlab.v4.objects.ProjectSnippetNote*
  - *gitlab.v4.objects.ProjectSnippetNoteManager*
  - `gitlab.v4.objects.ProjectSnippet.notes`

- v3 API:

  Issues:

  - *gitlab.v3.objects.ProjectIssueNote*
  - *gitlab.v3.objects.ProjectIssueNoteManager*

- *gitlab.v3.objects.ProjectIssue.notes*
- gitlab.v3.objects.Project.issue_notes
- gitlab.Gitlab.project_issue_notes

MergeRequests:

- *gitlab.v3.objects.ProjectMergeRequestNote*
- *gitlab.v3.objects.ProjectMergeRequestNoteManager*
- *gitlab.v3.objects.ProjectMergeRequest.notes*
- gitlab.v3.objects.Project.mergerequest_notes
- gitlab.Gitlab.project_mergerequest_notes

Snippets:

- *gitlab.v3.objects.ProjectSnippetNote*
- *gitlab.v3.objects.ProjectSnippetNoteManager*
- *gitlab.v3.objects.ProjectSnippet.notes*
- gitlab.v3.objects.Project.snippet_notes
- gitlab.Gitlab.project_snippet_notes

- GitLab API: https://docs.gitlab.com/ce/api/repository_files.html

Examples

List the notes for a resource:

```
i_notes = issue.notes.list()
mr_notes = mr.notes.list()
s_notes = snippet.notes.list()
```

Get a note for a resource:

```
i_note = issue.notes.get(note_id)
mr_note = mr.notes.get(note_id)
s_note = snippet.notes.get(note_id)
```

Create a note for a resource:

```
i_note = issue.notes.create({'body': 'note content'})
mr_note = mr.notes.create({'body': 'note content'})
s_note = snippet.notes.create({'body': 'note content'})
```

Update a note for a resource:

```
note.body = 'updated note content'
note.save()
```

Delete a note for a resource:

```
note.delete()
```

5.21.7 Project members

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectMember*
  - *gitlab.v4.objects.ProjectMemberManager*
  - gitlab.v4.objects.Project.members
- v3 API:
  - *gitlab.v3.objects.ProjectMember*
  - *gitlab.v3.objects.ProjectMemberManager*
  - *gitlab.v3.objects.Project.members*
  - gitlab.Gitlab.project_members
- GitLab API: https://docs.gitlab.com/ce/api/members.html

Examples

List the project members:

```
members = project.members.list()
```

Search project members matching a query string:

```
members = project.members.list(query='bar')
```

Get a single project member:

```
member = project.members.get(1)
```

Add a project member:

```
member = project.members.create({'user_id': user.id, 'access_level':
                                 gitlab.DEVELOPER_ACCESS})
```

Modify a project member (change the access level):

```
member.access_level = gitlab.MASTER_ACCESS
member.save()
```

Remove a member from the project team:

```
project.members.delete(user.id)
# or
member.delete()
```

Share the project with a group:

```
project.share(group.id, gitlab.DEVELOPER_ACCESS)
```

5.21.8 Project hooks

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectHook*
  - *gitlab.v4.objects.ProjectHookManager*
  - gitlab.v4.objects.Project.hooks
- v3 API:
  - *gitlab.v3.objects.ProjectHook*
  - *gitlab.v3.objects.ProjectHookManager*
  - *gitlab.v3.objects.Project.hooks*
  - gitlab.Gitlab.project_hooks
- GitLab API: https://docs.gitlab.com/ce/api/projects.html#hooks

Examples

List the project hooks:

```
hooks = project.hooks.list()
```

Get a project hook:

```
hook = project.hooks.get(1)
```

Create a project hook:

```
hook = gl.project_hooks.create({'url': 'http://my/action/url',
                                'push_events': 1},
                               project_id=1)
# or
hook = project.hooks.create({'url': 'http://my/action/url', 'push_events': 1})
```

Update a project hook:

```
hook.push_events = 0
hook.save()
```

Delete a project hook:

```
project.hooks.delete(1)
# or
hook.delete()
```

5.21.9 Project Services

Reference

- v4 API:

  - *gitlab.v4.objects.ProjectService*
  - *gitlab.v4.objects.ProjectServiceManager*
  - gitlab.v4.objects.Project.services
- v3 API:
  - *gitlab.v3.objects.ProjectService*
  - *gitlab.v3.objects.ProjectServiceManager*
  - *gitlab.v3.objects.Project.services*
  - gitlab.Gitlab.project_services
- GitLab API: https://docs.gitlab.com/ce/api/services.html

Exammples

Get a service:

```
# For v3
service = project.services.get(service_name='asana', project_id=1)
# For v4
service = project.services.get('asana')
# display its status (enabled/disabled)
print(service.active)
```

List the code names of available services (doesn't return objects):

```
services = gl.project_services.available()
```

Configure and enable a service:

```
service.api_key = 'randomkey'
service.save()
```

Disable a service:

```
service.delete()
```

5.21.10 Issue boards

Boards are a visual representation of existing issues for a project. Issues can be moved from one list to the other to track progress and help with priorities.

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectBoard*
  - *gitlab.v4.objects.ProjectBoardManager*
  - gitlab.v4.objects.Project.boards
- v3 API:

  - *gitlab.v3.objects.ProjectBoard*
  - *gitlab.v3.objects.ProjectBoardManager*
  - *gitlab.v3.objects.Project.boards*
  - gitlab.Gitlab.project_boards
- GitLab API: https://docs.gitlab.com/ce/api/boards.html

Examples

Get the list of existing boards for a project:

```
boards = project.boards.list()
```

Get a single board for a project:

```
board = project.boards.get(board_id)
```

5.21.11 Board lists

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectBoardList*
  - *gitlab.v4.objects.ProjectBoardListManager*
  - gitlab.v4.objects.Project.board_lists
- v3 API:
  - *gitlab.v3.objects.ProjectBoardList*
  - *gitlab.v3.objects.ProjectBoardListManager*
  - gitlab.v3.objects.ProjectBoard.lists
  - *gitlab.v3.objects.Project.board_lists*
  - gitlab.Gitlab.project_board_lists
- GitLab API: https://docs.gitlab.com/ce/api/boards.html

Examples

List the issue lists for a board:

```
b_lists = board.lists.list()
```

Get a single list:

```
b_list = board.lists.get(list_id)
```

Create a new list:

```
# First get a ProjectLabel
label = get_or_create_label()
# Then use its ID to create the new board list
b_list = board.lists.create({'label_id': label.id})
```

Change a list position. The first list is at position 0. Moving a list will set it at the given position and move the following lists up a position:

```
b_list.position = 2
b_list.save()
```

Delete a list:

```
b_list.delete()
```

5.21.12 File uploads

Reference

- v4 API:
  - *gitlab.v4.objects.Project.upload*
- v3 API:
  - *gitlab.v3.objects.Project.upload*
- Gitlab API: https://docs.gitlab.com/ce/api/projects.html#upload-a-file

Examples

Upload a file into a project using a filesystem path:

```
# Or provide a full path to the uploaded file
project.upload("filename.txt", filepath="/some/path/filename.txt")
```

Upload a file into a project without a filesystem path:

```
# Upload a file using its filename and filedata
project.upload("filename.txt", filedata="Raw data")
```

Upload a file and comment on an issue using the uploaded file's markdown:

```
uploaded_file = project.upload("filename.txt", filedata="data")
issue = project.issues.get(issue_id)
issue.notes.create({
    "body": "See the attached file: {}".format(uploaded_file["markdown"])
})
```

Upload a file and comment on an issue while using custom markdown to reference the uploaded file:

```
uploaded_file = project.upload("filename.txt", filedata="data")
issue = project.issues.get(issue_id)
issue.notes.create({
    "body": "See the [attached file]({})".format(uploaded_file["url"])
})
```

5.22 Runners

Runners are external processes used to run CI jobs. They are deployed by the administrator and registered to the GitLab instance.

Shared runners are available for all projects. Specific runners are enabled for a list of projects.

5.22.1 Global runners (admin)

Reference

- v4 API:
  - *gitlab.v4.objects.Runner*
  - *gitlab.v4.objects.RunnerManager*
  - gitlab.Gitlab.runners
- v3 API:
  - *gitlab.v3.objects.Runner*
  - *gitlab.v3.objects.RunnerManager*
  - gitlab.Gitlab.runners
- GitLab API: https://docs.gitlab.com/ce/api/runners.html

Examples

Use the `list()` and `all()` methods to list runners.

Both methods accept a `scope` parameter to filter the list. Allowed values for this parameter are:

- `active`
- `paused`
- `online`
- `specific` (`all()` only)
- `shared` (`all()` only)

Note: The returned objects hold minimal information about the runners. Use the `get()` method to retrieve detail about a runner.

```
# List owned runners
runners = gl.runners.list()
# With a filter
runners = gl.runners.list(scope='active')
# List all runners, using a filter
runners = gl.runners.all(scope='paused')
```

Get a runner's detail:

```
runner = gl.runners.get(runner_id)
```

Update a runner:

```
runner = gl.runners.get(runner_id)
runner.tag_list.append('new_tag')
runner.save()
```

Remove a runner:

```
gl.runners.delete(runner_id)
# or
runner.delete()
```

5.22.2 Project runners

Reference

- v4 API:
  - *gitlab.v4.objects.ProjectRunner*
  - *gitlab.v4.objects.ProjectRunnerManager*
  - gitlab.v4.objects.Project.runners
- v3 API:
  - *gitlab.v3.objects.ProjectRunner*
  - *gitlab.v3.objects.ProjectRunnerManager*
  - *gitlab.v3.objects.Project.runners*
  - gitlab.Gitlab.project_runners
- GitLab API: https://docs.gitlab.com/ce/api/runners.html

Examples

List the runners for a project:

```
runners = project.runners.list()
```

Enable a specific runner for a project:

```
p_runner = project.runners.create({'runner_id': runner.id})
```

Disable a specific runner for a project:

```
project.runners.delete(runner.id)
```

5.23 Settings

5.23.1 Reference

- v4 API:
  - *gitlab.v4.objects.ApplicationSettings*
  - *gitlab.v4.objects.ApplicationSettingsManager*
  - gitlab.Gitlab.settings
- v3 API:
  - *gitlab.v3.objects.ApplicationSettings*
  - *gitlab.v3.objects.ApplicationSettingsManager*
  - gitlab.Gitlab.settings
- GitLab API: https://docs.gitlab.com/ce/api/settings.html

5.23.2 Examples

Get the settings:

```
settings = gl.settings.get()
```

Update the settings:

```
s.signin_enabled = False
s.save()
```

5.24 Snippets

You can store code snippets in Gitlab. Snippets can be attached to projects (see *Project snippets*), but can also be detached.

- Object class: gitlab.objects.Namespace
- Manager object: gitlab.Gitlab.snippets

5.24.1 Examples

List snippets woned by the current user:

```
snippets = gl.snippets.list()
```

List the public snippets:

```
public_snippets = gl.snippets.public()
```

Get a snippet:

```
snippet = gl.snippets.get(snippet_id)
# get the content
content = snippet.raw()
```

Warning: Blobs are entirely stored in memory unless you use the streaming feature. See *the artifacts example*.

Create a snippet:

```
snippet = gl.snippets.create({'title': 'snippet1',
                              'file_name': 'snippet1.py',
                              'content': open('snippet1.py').read()})
```

Update a snippet:

```
snippet.visibility_level = gitlab.Project.VISIBILITY_PUBLIC
snippet.save()
```

Delete a snippet:

```
gl.snippets.delete(snippet_id)
# or
snippet.delete()
```

5.25 System hooks

5.25.1 Reference

- v4 API:
  - *gitlab.v4.objects.Hook*
  - *gitlab.v4.objects.HookManager*
  - gitlab.Gitlab.hooks
- v3 API:
  - *gitlab.v3.objects.Hook*
  - *gitlab.v3.objects.HookManager*
  - gitlab.Gitlab.hooks
- GitLab API: https://docs.gitlab.com/ce/api/system_hooks.html

5.25.2 Examples

List the system hooks:

```
hooks = gl.hooks.list()
```

Create a system hook:

```
hook = gl.hooks.create({'url': 'http://your.target.url'})
```

Test a system hook. The returned object is not usable (it misses the hook ID):

```
gl.hooks.get(hook_id)
```

Delete a system hook:

```
gl.hooks.delete(hook_id)
# or
hook.delete()
```

5.26 Templates

You can request templates for different type of files:

- License files
- .gitignore files
- GitLab CI configuration files
- Dockerfiles

5.26.1 License templates

Reference

- v4 API:
  - *gitlab.v4.objects.License*
  - *gitlab.v4.objects.LicenseManager*
  - gitlab.Gitlab.licenses
- v3 API:
  - *gitlab.v3.objects.License*
  - *gitlab.v3.objects.LicenseManager*
  - gitlab.Gitlab.licenses
- GitLab API: https://docs.gitlab.com/ce/api/templates/licenses.html

Examples

List known license templates:

```
licenses = gl.licenses.list()
```

Generate a license content for a project:

```
license = gl.licenses.get('apache-2.0', project='foobar', fullname='John Doe')
print(license.content)
```

5.26.2 .gitignore templates

Reference

- v4 API:
  - *gitlab.v4.objects.Gitignore*
  - *gitlab.v4.objects.GitignoreManager*
  - gitlab.Gitlab.gitignores
- v3 API:
  - *gitlab.v3.objects.Gitignore*
  - *gitlab.v3.objects.GitignoreManager*
  - gitlab.Gitlab.gitignores
- GitLab API: https://docs.gitlab.com/ce/api/templates/gitignores.html

Examples

List known gitignore templates:

```
gitignores = gl.gitignores.list()
```

Get a gitignore template:

```
gitignore = gl.gitignores.get('Python')
print(gitignore.content)
```

5.26.3 GitLab CI templates

Reference

- v4 API:
  - *gitlab.v4.objects.Gitlabciyml*
  - *gitlab.v4.objects.GitlabciymlManager*
  - gitlab.Gitlab.gitlabciymls
- v3 API:
  - *gitlab.v3.objects.Gitlabciyml*
  - *gitlab.v3.objects.GitlabciymlManager*
  - gitlab.Gitlab.gitlabciymls
- GitLab API: https://docs.gitlab.com/ce/api/templates/gitlab_ci_ymls.html

Examples

List known GitLab CI templates:

```
gitlabciymls = gl.gitlabciymls.list()
```

Get a GitLab CI template:

```
gitlabciyml = gl.gitlabciymls.get('Pelican')
print(gitlabciyml.content)
```

5.26.4 Dockerfile templates

Reference

- v4 API:
  - *gitlab.v4.objects.Dockerfile*
  - *gitlab.v4.objects.DockerfileManager*
  - gitlab.Gitlab.gitlabciymls
- GitLab API: Not documented.

Examples

List known Dockerfile templates:

```
dockerfiles = gl.dockerfiles.list()
```

Get a Dockerfile template:

```
dockerfile = gl.dockerfiles.get('Python')
print(dockerfile.content)
```

5.27 Todos

Use `Todo` objects to manipulate todos. The `gitlab.Gitlab.todos` manager object provides helper functions.

5.27.1 Examples

List active todos:

```
todos = gl.todos.list()
```

You can filter the list using the following parameters:

- `action`: can be `assigned`, `mentioned`, `build_failed`, `marked`, or `approval_required`
- `author_id`
- `project_id`

- `state`: can be `pending` or `done`
- `type`: can be `Issue` or `MergeRequest`

For example:

```
todos = gl.todos.list(project_id=1)
todos = gl.todos.list(state='done', type='Issue')
```

Get a single todo:

```
todo = gl.todos.get(todo_id)
```

Mark a todo as done:

```
gl.todos.delete(todo_id)
# or
todo.delete()
```

Mark all the todos as done:

```
nb_of_closed_todos = gl.todos.delete_all()
```

5.28 Users and current user

The Gitlab API exposes user-related method that can be manipulated by admins only.

The currently logged-in user is also exposed.

5.28.1 Users

References

- v4 API:
  - *gitlab.v4.objects.User*
  - *gitlab.v4.objects.UserManager*
  - `gitlab.Gitlab.users`
- v3 API:
  - *gitlab.v3.objects.User*
  - *gitlab.v3.objects.UserManager*
  - `gitlab.Gitlab.users`
- GitLab API: https://docs.gitlab.com/ce/api/users.html

Examples

Get the list of users:

```
users = gl.users.list()
```

Search users whose username match the given string:

```
users = gl.users.list(search='oo')
```

Get a single user:

```
# by ID
user = gl.users.get(2)
# by username
user = gl.users.list(username='root')[0]
```

Create a user:

```
user = gl.users.create({'email': 'john@doe.com',
                        'password': 's3cur3s3cr3T',
                        'username': 'jdoe',
                        'name': 'John Doe'})
```

Update a user:

```
user.name = 'Real Name'
user.save()
```

Delete a user:

```
gl.users.delete(2)
user.delete()
```

Block/Unblock a user:

```
user.block()
user.unblock()
```

5.28.2 User custom attributes

References

- v4 API:
  - *gitlab.v4.objects.UserCustomAttribute*
  - *gitlab.v4.objects.UserCustomAttributeManager*
  - gitlab.v4.objects.User.customattributes
- GitLab API: https://docs.gitlab.com/ce/api/custom_attributes.html

Examples

List custom attributes for a user:

```
attrs = user.customattributes.list()
```

Get a custom attribute for a user:

```
attr = user.customattributes.get(attr_key)
```

Set (create or update) a custom attribute for a user:

```
attr = user.customattributes.set(attr_key, attr_value)
```

Delete a custom attribute for a user:

```
attr.delete()
# or
user.customattributes.delete(attr_key)
```

Search users by custom attribute:

```
user.customattributes.set('role': 'QA')
gl.users.list(custom_attributes={'role': 'QA'})
```

5.28.3 User impersonation tokens

References

- v4 API:
  - *gitlab.v4.objects.UserImpersonationToken*
  - *gitlab.v4.objects.UserImpersonationTokenManager*
  - gitlab.v4.objects.User.impersonationtokens
- GitLab API: https://docs.gitlab.com/ce/api/users.html#get-all-impersonation-tokens-of-a-user

List impersonation tokens for a user:

```
i_t = user.impersonationtokens.list(state='active')
i_t = user.impersonationtokens.list(state='inactive')
```

Get an impersonation token for a user:

```
i_t = user.impersonationtokens.get(i_t_id)
```

Create and use an impersonation token for a user:

```
i_t = user.impersonationtokens.create({'name': 'token1', 'scopes': ['api']})
# use the token to create a new gitlab connection
user_gl = gitlab.Gitlab(gitlab_url, private_token=i_t.token)
```

Revoke (delete) an impersonation token for a user:

```
i_t = user.impersonationtokens.list(state='active')
i_t = user.impersonationtokens.list(state='inactive')
```

5.28.4 Current User

References

- v4 API:

  - *gitlab.v4.objects.CurrentUser*
  - *gitlab.v4.objects.CurrentUserManager*
  - gitlab.Gitlab.user
- v3 API:
  - *gitlab.v3.objects.CurrentUser*
  - gitlab.v3.objects.CurrentUserManager
  - gitlab.Gitlab.user
- GitLab API: https://docs.gitlab.com/ce/api/users.html

Examples

Get the current user:

```
gl.auth()
current_user = gl.user
```

5.28.5 GPG keys

References

You can manipulate GPG keys for the current user and for the other users if you are admin.

- v4 API:
  - *gitlab.v4.objects.CurrentUserGPGKey*
  - *gitlab.v4.objects.CurrentUserGPGKeyManager*
  - gitlab.v4.objects.CurrentUser.gpgkeys
  - *gitlab.v4.objects.UserGPGKey*
  - *gitlab.v4.objects.UserGPGKeyManager*
  - gitlab.v4.objects.User.gpgkeys
- GitLab API: https://docs.gitlab.com/ce/api/users.html#list-all-gpg-keys

Exemples

List GPG keys for a user:

```
gpgkeys = user.gpgkeys.list()
```

Get an GPG gpgkey for a user:

```
gpgkey = user.gpgkeys.get(1)
```

Create an GPG gpgkey for a user:

```
# get the key with `gpg --export -a GPG_KEY_ID`
k = user.gpgkeys.create({'key': public_key_content})
```

Delete an GPG gpgkey for a user:

```
user.gpgkeys.delete(1)
# or
gpgkey.delete()
```

5.28.6 SSH keys

References

You can manipulate SSH keys for the current user and for the other users if you are admin.

- v4 API:
  - *gitlab.v4.objects.CurrentUserKey*
  - *gitlab.v4.objects.CurrentUserKeyManager*
  - gitlab.v4.objects.CurrentUser.keys
  - *gitlab.v4.objects.UserKey*
  - *gitlab.v4.objects.UserKeyManager*
  - gitlab.v4.objects.User.keys
- v3 API:
  - *gitlab.v3.objects.CurrentUserKey*
  - *gitlab.v3.objects.CurrentUserKeyManager*
  - gitlab.v3.objects.CurrentUser.keys
  - gitlab.Gitlab.user.keys
  - *gitlab.v3.objects.UserKey*
  - *gitlab.v3.objects.UserKeyManager*
  - *gitlab.v3.objects.User.keys*
  - gitlab.Gitlab.user_keys
- GitLab API: https://docs.gitlab.com/ce/api/users.html#list-ssh-keys

Exemples

List SSH keys for a user:

```
keys = user.keys.list()
```

Get an SSH key for a user:

```
key = user.keys.get(1)
```

Create an SSH key for a user:

```
k = user.keys.create({'title': 'my_key',
                      'key': open('/home/me/.ssh/id_rsa.pub').read()})
```

Delete an SSH key for a user:

```
user.keys.delete(1)
# or
key.delete()
```

5.28.7 Emails

References

You can manipulate emails for the current user and for the other users if you are admin.

- v4 API:
  - *gitlab.v4.objects.CurrentUserEmail*
  - *gitlab.v4.objects.CurrentUserEmailManager*
  - gitlab.v4.objects.CurrentUser.emails
  - *gitlab.v4.objects.UserEmail*
  - *gitlab.v4.objects.UserEmailManager*
  - gitlab.v4.objects.User.emails
- v3 API:
  - *gitlab.v3.objects.CurrentUserEmail*
  - *gitlab.v3.objects.CurrentUserEmailManager*
  - gitlab.v3.objects.CurrentUser.emails
  - gitlab.Gitlab.user.emails
  - *gitlab.v3.objects.UserEmail*
  - *gitlab.v3.objects.UserEmailManager*
  - *gitlab.v3.objects.User.emails*
  - gitlab.Gitlab.user_emails
- GitLab API: https://docs.gitlab.com/ce/api/users.html#list-emails

Exemples

List emails for a user:

```
emails = user.emails.list()
```

Get an email for a user:

```
email = gl.user_emails.list(1, user_id=1)
# or
email = user.emails.get(1)
```

Create an email for a user:

```
k = user.emails.create({'email': 'foo@bar.com'})
```

Delete an email for a user:

```
user.emails.delete(1)
# or
email.delete()
```

5.28.8 Users activities

References

- v4 only
- admin only
- v4 API:
  - *gitlab.v4.objects.UserActivities*
  - *gitlab.v4.objects.UserActivitiesManager*
  - gitlab.Gitlab.user_activities
- GitLab API: https://docs.gitlab.com/ce/api/users.html#get-user-activities-admin-only

Examples

Get the users activities:

```
activities = gl.user_activities.list(all=True, as_list=False)
```

5.29 Sidekiq metrics

5.29.1 Reference

- v4 API:
  - *gitlab.v4.objects.SidekiqManager*
  - gitlab.Gitlab.sidekiq
- v3 API:
  - *gitlab.v3.objects.SidekiqManager*
  - gitlab.Gitlab.sidekiq
- GitLab API: https://docs.gitlab.com/ce/api/sidekiq_metrics.html

5.29.2 Examples

```
gl.sidekiq.queue_metrics()
gl.sidekiq.process_metrics()
gl.sidekiq.job_stats()
gl.sidekiq.compound_metrics()
```

5.30 Wiki pages

5.30.1 References

- v4 API:
  - *gitlab.v4.objects.ProjectWiki*
  - *gitlab.v4.objects.ProjectWikiManager*
  - gitlab.v4.objects.Project.wikis

Examples

Get the list of wiki pages for a project:

```
pages = project.wikis.list()
```

Get a single wiki page:

```
page = project.wikis.get(page_slug)
```

Create a wiki page:

```
page = project.wikis.create({'title': 'Wiki Page 1',
                             'content': open(a_file).read()})
```

Update a wiki page:

```
page.content = 'My new content'
page.save()
```

Delete a wiki page:

```
page.delete()
```

CHAPTER 6

gitlab package

6.1 Subpackages

6.1.1 gitlab.v3 package

Submodules

gitlab.v3.objects module

class gitlab.v3.objects.**ApplicationSettings**(*gl*, *data=None*, ***kwargs*)

Bases: *gitlab.base.GitlabObject*

save(***kwargs*)

Send the modified object to the GitLab server. The following attributes are sent:

- `after_sign_out_path` (optional)
- `container_registry_token_expire_delay` (optional)
- `default_branch_protection` (optional)
- `default_project_visibility` (optional)
- `default_projects_limit` (optional)
- `default_snippet_visibility` (optional)
- `domain_blacklist` (optional)
- `domain_blacklist_enabled` (optional)
- `domain_whitelist` (optional)
- `enabled_git_access_protocol` (optional)
- `gravatar_enabled` (optional)
- `home_page_url` (optional)

- max_attachment_size (optional)
- repository_storage (optional)
- restricted_signup_domains (optional)
- restricted_visibility_levels (optional)
- session_expire_delay (optional)
- sign_in_text (optional)
- signin_enabled (optional)
- signup_enabled (optional)
- twitter_sharing_enabled (optional)
- user_oauth_applications (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

canCreate = False

canDelete = False

canList = False

getRequiresId = False

optionalUpdateAttrs = ['after_sign_out_path', 'container_registry_token_expire_delay',

class gitlab.v3.objects.**ApplicationSettingsManager**(*gl*, *parent=None*, *args=[]*)

Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ApplicationSettings'> objects.

<class 'gitlab.v3.objects.ApplicationSettings'> objects can be updated.

get(***kwargs*)

Get a single object of type <class 'gitlab.v3.objects.ApplicationSettings'>.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls

alias of *ApplicationSettings*

class gitlab.v3.objects.**BroadcastMessage**(*gl*, *data=None*, ***kwargs*)

Bases: *gitlab.base.GitlabObject*

save(***kwargs*)

Send the modified object to the GitLab server. The following attributes are sent:

- message (optional)
- starts_at (optional)
- ends_at (optional)
- color (optional)
- font (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

optionalCreateAttrs = ['starts_at', 'ends_at', 'color', 'font']

optionalUpdateAttrs = ['message', 'starts_at', 'ends_at', 'color', 'font']

requiredCreateAttrs = ['message']

requiredUpdateAttrs = []

class gitlab.v3.objects.**BroadcastMessageManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.BroadcastMessage'> objects.

<class 'gitlab.v3.objects.BroadcastMessage'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.BroadcastMessage'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.BroadcastMessage'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.BroadcastMessage'>.

`data` is a dict defining the object attributes. Available attributes are:

- `message` (required)
- `starts_at` (optional)
- `ends_at` (optional)
- `color` (optional)
- `font` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *BroadcastMessage*

class gitlab.v3.objects.**CurrentUser**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canList = False

canUpdate = False

managers = (('emails', 'CurrentUserEmailManager', [('user_id', 'id')]), ('keys', 'Curr

shortPrintAttr = 'username'

class gitlab.v3.objects.**CurrentUserEmail**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

requiredCreateAttrs = ['email']

shortPrintAttr = 'email'

class gitlab.v3.objects.**CurrentUserEmailManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.CurrentUserEmail'> objects.

<class 'gitlab.v3.objects.CurrentUserEmail'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.CurrentUserEmail'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.CurrentUserEmail'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.CurrentUserEmail'>.

`data` is a dict defining the object attributes. Available attributes are:

- `email` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *CurrentUserEmail*

class gitlab.v3.objects.**CurrentUserKey**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

requiredCreateAttrs = ['title', 'key']

shortPrintAttr = 'title'

class gitlab.v3.objects.**CurrentUserKeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.CurrentUserKey'> objects.

<class 'gitlab.v3.objects.CurrentUserKey'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.CurrentUserKey'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.CurrentUserKey'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.CurrentUserKey'>.

`data` is a dict defining the object attributes. Available attributes are:

- `title` (required)
- `key` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *CurrentUserKey*

class gitlab.v3.objects.**DeployKey**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canGet = 'from_list'

canUpdate = False

class gitlab.v3.objects.**DeployKeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.DeployKey'> objects.

<class 'gitlab.v3.objects.DeployKey'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.DeployKey'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.DeployKey'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *DeployKey*

class gitlab.v3.objects.**Gitignore**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canUpdate = False

idAttr = 'name'

class gitlab.v3.objects.**GitignoreManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Gitignore'> objects.

<class 'gitlab.v3.objects.Gitignore'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Gitignore'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Gitignore'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *Gitignore*

class gitlab.v3.objects.**Gitlabciyml**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canUpdate = False

idAttr = 'name'

class gitlab.v3.objects.**GitlabciymlManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Gitlabciyml'> objects.

<class 'gitlab.v3.objects.Gitlabciyml'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Gitlabciyml'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Gitlabciyml'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *Gitlabciyml*

class gitlab.v3.objects.**Group**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

accessrequests
GroupAccessRequestManager - Manager for objects.

members
GroupMemberManager - Manager for objects.

notificationsettings
GroupNotificationSettingsManager - Manager for objects.

projects
GroupProjectManager - Manager for objects.

issues
GroupIssueManager - Manager for objects.

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `name` (optional)

- path (optional)
- description (optional)
- visibility_level (optional)
- lfs_enabled (optional)
- request_access_enabled (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

DEVELOPER_ACCESS = 30

GUEST_ACCESS = 10

MASTER_ACCESS = 40

OWNER_ACCESS = 50

REPORTER_ACCESS = 20

VISIBILITY_INTERNAL = 10

VISIBILITY_PRIVATE = 0

VISIBILITY_PUBLIC = 20

managers = (('accessrequests', 'GroupAccessRequestManager', [('group_id', 'id')]), ('m

optionalCreateAttrs = ['description', 'visibility_level', 'parent_id', 'lfs_enabled',

optionalUpdateAttrs = ['name', 'path', 'description', 'visibility_level', 'lfs_enabled

requiredCreateAttrs = ['name', 'path']

shortPrintAttr = 'name'

transfer_project(*id*, ***kwargs*)

Transfers a project to this new groups.

Parameters **id** (*int*) – ID of the project to transfer.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabTransferProjectError – If the server fails to perform the request.

class gitlab.v3.objects.**GroupAccessRequest**(*gl*, *data=None*, ***kwargs*)

Bases: *gitlab.base.GitlabObject*

approve(*access_level=30*, ***kwargs*)

Approve an access request.

Parameters **access_level** (*int*) – The access level for the user.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabUpdateError – If the server fails to perform the request.

canGet = 'from_list'

canUpdate = False

class gitlab.v3.objects.**GroupAccessRequestManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.GroupAccessRequest'> objects.

<class 'gitlab.v3.objects.GroupAccessRequest'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.GroupAccessRequest'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.GroupAccessRequest'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.GroupAccessRequest'>.

`data` is a dict defining the object attributes. Available attributes are:

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *GroupAccessRequest*

class gitlab.v3.objects.**GroupIssue**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canGet = 'from_list'

canUpdate = False

optionalListAttrs = ['state', 'labels', 'milestone', 'order_by', 'sort']

requiredUrlAttrs = ['group_id']

class gitlab.v3.objects.**GroupIssueManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.GroupIssue'> objects.

<class 'gitlab.v3.objects.GroupIssue'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.GroupIssue'>.

Available keys for `kwargs` are:

- `state` (optional)
- `labels` (optional)
- `milestone` (optional)
- `order_by` (optional)
- `sort` (optional)
- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.GroupIssue'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *GroupIssue*

class gitlab.v3.objects.**GroupManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Group'> objects.

<class 'gitlab.v3.objects.Group'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Group'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Group'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.Group'>.

`data` is a dict defining the object attributes. Available attributes are:

- `name` (required)
- `path` (required)

- `description` (optional)
- `visibility_level` (optional)
- `parent_id` (optional)
- `lfs_enabled` (optional)
- `request_access_enabled` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *Group*

search(*query*, ***kwargs*)
Searches groups by name.

Parameters

- **query** (*str*) – The search string
- **all** (*bool*) – If True, return all the items, without pagination

Returns a list of matching groups.

Return type *list*(*Group*)

class gitlab.v3.objects.**GroupMember**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `access_level` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

canGet = 'from_list'

optionalCreateAttrs = ['expires_at']

requiredCreateAttrs = ['access_level', 'user_id']

requiredUpdateAttrs = ['access_level']

requiredUrlAttrs = ['group_id']

shortPrintAttr = 'username'

class gitlab.v3.objects.**GroupMemberManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.GroupMember'> objects.

<class 'gitlab.v3.objects.GroupMember'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.GroupMember'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.GroupMember'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.GroupMember'>.

`data` is a dict defining the object attributes. Available attributes are:

- `group_id` (required if not discovered on the parent objects)
- `access_level` (required)
- `user_id` (required)
- `expires_at` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *GroupMember*

class gitlab.v3.objects.**GroupNotificationSettings**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.v3.objects.NotificationSettings*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `level` (optional)
- `notification_email` (optional)
- `new_note` (optional)
- `new_issue` (optional)
- `reopen_issue` (optional)
- `close_issue` (optional)
- `reassign_issue` (optional)
- `new_merge_request` (optional)

- reopen_merge_request (optional)
- close_merge_request (optional)
- reassign_merge_request (optional)
- merge_merge_request (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

requiredUrlAttrs = ['group_id']

class gitlab.v3.objects.**GroupNotificationSettingsManager**(*gl*, *parent=None*, *args=[]*)

Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.GroupNotificationSettings'> objects.

<class 'gitlab.v3.objects.GroupNotificationSettings'> objects can be updated.

get(***kwargs*)

Get a single object of type <class 'gitlab.v3.objects.GroupNotificationSettings'>.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls

alias of *GroupNotificationSettings*

class gitlab.v3.objects.**GroupProject**(**args*, ***kwargs*)

Bases: *gitlab.v3.objects.Project*

canCreate = False

canDelete = False

canGet = 'from_list'

canUpdate = False

optionalListAttrs = ['archived', 'visibility', 'order_by', 'sort', 'search', 'ci_enabl

class gitlab.v3.objects.**GroupProjectManager**(*gl*, *parent=None*, *args=[]*)

Bases: *gitlab.v3.objects.ProjectManager*

Manager for <class 'gitlab.v3.objects.GroupProject'> objects.

<class 'gitlab.v3.objects.GroupProject'> objects **cannot** be updated.

list(***kwargs*)

Returns a list of objects of type <class 'gitlab.v3.objects.GroupProject'>.

Available keys for kwargs are:

- archived (optional)
- visibility (optional)
- order_by (optional)
- sort (optional)
- search (optional)
- ci_enabled_first (optional)
- per_page (int): number of item per page. May be limited by the server.

- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.GroupProject'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *GroupProject*

class gitlab.v3.objects.**Hook**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

requiredCreateAttrs = ['url']

shortPrintAttr = 'url'

class gitlab.v3.objects.**HookManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Hook'> objects.

<class 'gitlab.v3.objects.Hook'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Hook'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Hook'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.Hook'>.

data is a dict defining the object attributes. Available attributes are:

- url (required)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *Hook*

class gitlab.v3.objects.**Issue**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canGet = 'from_list'

canUpdate = False

optionalListAttrs = ['state', 'labels', 'order_by', 'sort']

shortPrintAttr = 'title'

class gitlab.v3.objects.**IssueManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Issue'> objects.

<class 'gitlab.v3.objects.Issue'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Issue'>.

Available keys for `kwargs` are:

- `state` (optional)
- `labels` (optional)
- `order_by` (optional)
- `sort` (optional)
- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Issue'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *Issue*

class gitlab.v3.objects.**License**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canUpdate = False

idAttr = 'key'

optionalGetAttrs = ['project', 'fullname']

optionalListAttrs = ['popular']

class gitlab.v3.objects.**LicenseManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.License'> objects.

<class 'gitlab.v3.objects.License'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.License'>.

Available keys for `kwargs` are:

- `popular` (optional)
- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.License'> using its `id`.

Available keys for `kwargs` are:

- `project` (optional)
- `fullname` (optional)
- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *License*

class gitlab.v3.objects.**Namespace**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canGet = 'from_list'

canUpdate = False

optionalListAttrs = ['search']

class gitlab.v3.objects.**NamespaceManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Namespace'> objects.

<class 'gitlab.v3.objects.Namespace'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Namespace'>.

Available keys for `kwargs` are:

- `search` (optional)
- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve

- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Namespace'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *Namespace*

class gitlab.v3.objects.**NotificationSettings**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `level` (optional)
- `notification_email` (optional)
- `new_note` (optional)
- `new_issue` (optional)
- `reopen_issue` (optional)
- `close_issue` (optional)
- `reassign_issue` (optional)
- `new_merge_request` (optional)
- `reopen_merge_request` (optional)
- `close_merge_request` (optional)
- `reassign_merge_request` (optional)
- `merge_merge_request` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

canCreate = False

canDelete = False

canList = False

getRequiresId = False

optionalUpdateAttrs = ['level', 'notification_email', 'new_note', 'new_issue', 'reopen

class gitlab.v3.objects.**NotificationSettingsManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.NotificationSettings'> objects.

<class 'gitlab.v3.objects.NotificationSettings'> objects can be updated.

get(***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.NotificationSettings'>.

Available keys for `kwargs` are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *NotificationSettings*

class gitlab.v3.objects.**Project**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

accessrequests
ProjectAccessRequestManager - Manager for objects.

boards
ProjectBoardManager - Manager for objects.

board_lists
ProjectBoardListManager - Manager for objects.

branches
ProjectBranchManager - Manager for objects.

builds
ProjectBuildManager - Manager for objects.

commits
ProjectCommitManager - Manager for objects.

deployments
ProjectDeploymentManager - Manager for objects.

environments
ProjectEnvironmentManager - Manager for objects.

events
ProjectEventManager - Manager for objects.

files
ProjectFileManager - Manager for objects.

forks
ProjectForkManager - Manager for objects.

hooks
ProjectHookManager - Manager for objects.

keys
ProjectKeyManager - Manager for objects.

issues
ProjectIssueManager - Manager for objects.

labels
ProjectLabelManager - Manager for objects.

members
ProjectMemberManager - Manager for objects.

mergerequests
ProjectMergeRequestManager - Manager for objects.

milestones
ProjectMilestoneManager - Manager for objects.

notes
ProjectNoteManager - Manager for objects.

notificationsettings
ProjectNotificationSettingsManager - Manager for objects.

pipelines
ProjectPipelineManager - Manager for objects.

runners
ProjectRunnerManager - Manager for objects.

services
ProjectServiceManager - Manager for objects.

snippets
ProjectSnippetManager - Manager for objects.

tags
ProjectTagManager - Manager for objects.

triggers
ProjectTriggerManager - Manager for objects.

variables
ProjectVariableManager - Manager for objects.

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `name` (optional)
- `path` (optional)
- `default_branch` (optional)
- `description` (optional)
- `issues_enabled` (optional)
- `merge_requests_enabled` (optional)
- `builds_enabled` (optional)
- `wiki_enabled` (optional)
- `snippets_enabled` (optional)
- `container_registry_enabled` (optional)
- `shared_runners_enabled` (optional)
- `public` (optional)
- `visibility_level` (optional)
- `import_url` (optional)
- `public_builds` (optional)
- `only_allow_merge_if_build_succeeds` (optional)
- `only_allow_merge_if_all_discussions_are_resolved` (optional)
- `lfs_enabled` (optional)
- `request_access_enabled` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

VISIBILITY_INTERNAL = 10

VISIBILITY_PRIVATE = 0

VISIBILITY_PUBLIC = 20

archive(***kwargs*)
Archive a project.

Returns the updated Project

Return type *Project*

Raises

- GitlabCreateError – If the action cannot be done
- GitlabConnectionError – If the server cannot be reached.

create_fork_relation(*forked_from_id*)
Create a forked from/to relation between existing projects.

Parameters **forked_from_id** (*int*) – The ID of the project that was forked from

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabCreateError – If the server fails to perform the request.

delete_fork_relation()
Delete a forked relation between existing projects.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabDeleteError – If the server fails to perform the request.

managers = (('accessrequests', 'ProjectAccessRequestManager', [('project_id', 'id')]),

optionalCreateAttrs = ['path', 'namespace_id', 'description', 'issues_enabled', 'merge

optionalListAttrs = ['search']

optionalUpdateAttrs = ['name', 'path', 'default_branch', 'description', 'issues_enable

repository_archive(*sha=None*, *streamed=False*, *action=None*, *chunk_size=1024*, ***kwargs*)
Return a tarball of the repository.

Parameters

- **sha** (*str*) – ID of the commit (default branch by default).
- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data.
- **chunk_size** (*int*) – Size of each chunk.

Returns The binary data of the archive.

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the server fails to perform the request.

repository_blob(*sha*, *filepath*, *streamed=False*, *action=None*, *chunk_size=1024*, ***kwargs*)
Return the content of a file for a commit.

Parameters

- **sha** (*str*) – ID of the commit
- **filepath** (*str*) – Path of the file to return
- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data.
- **chunk_size** (*int*) – Size of each chunk.

Returns The file content

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the server fails to perform the request.

repository_compare(*from_*, *to*, ***kwargs*)
Returns a diff between two branches/commits.

Parameters

- **from** (*str*) – orig branch/SHA
- **to** (*str*) – dest branch/SHA

Returns The diff

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the server fails to perform the request.

repository_contributors()
Returns a list of contributors for the project.

Returns The contibutors

Return type *list*

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the server fails to perform the request.

repository_raw_blob(*sha*, *streamed=False*, *action=None*, *chunk_size=1024*, ***kwargs*)
Returns the raw file contents for a blob by blob SHA.

Parameters

- **sha** (*str*) – ID of the blob
- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data.

- **chunk_size** (*int*) – Size of each chunk.

Returns The blob content

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the server fails to perform the request.

repository_tree(*path=''*, *ref_name=''*, ***kwargs*)
Return a list of files in the repository.

Parameters

- **path** (*str*) – Path of the top folder (/ by default)
- **ref_name** (*str*) – Reference to a commit or branch

Returns The json representation of the tree.

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the server fails to perform the request.

requiredCreateAttrs = ['name']

share(*group_id*, *group_access*, ***kwargs*)
Share the project with a group.

Parameters

- **group_id** (*int*) – ID of the group.
- **group_access** (*int*) – Access level for the group.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabCreateError – If the server fails to perform the request.

shortPrintAttr = 'path'

star(***kwargs*)
Star a project.

Returns the updated Project

Return type *Project*

Raises

- GitlabCreateError – If the action cannot be done
- GitlabConnectionError – If the server cannot be reached.

trigger_build(*ref*, *token*, *variables={}*, ***kwargs*)
Trigger a CI build.

See https://gitlab.com/help/ci/triggers/README.md#trigger-a-build

Parameters

- **ref** (*str*) – Commit to build; can be a commit SHA, a branch name, ...
- **token** (*str*) – The trigger token
- **variables** (*dict*) – Variables passed to the build script

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabCreateError – If the server fails to perform the request.

unarchive (***kwargs*)
Unarchive a project.

Returns the updated Project

Return type *Project*

Raises

- GitlabDeleteError – If the action cannot be done
- GitlabConnectionError – If the server cannot be reached.

unstar (***kwargs*)
Unstar a project.

Returns the updated Project

Return type *Project*

Raises

- GitlabDeleteError – If the action cannot be done
- GitlabConnectionError – If the server cannot be reached.

upload (*filename*, *filedata=None*, *filepath=None*, ***kwargs*)
Upload the specified file into the project.

Note: Either filedata or filepath *MUST* be specified.

Parameters

- **filename** (*str*) – The name of the file being uploaded
- **filedata** (*bytes*) – The raw data of the file being uploaded
- **filepath** (*str*) – The path to a local file to upload (optional)

Raises

- GitlabConnectionError – If the server cannot be reached
- GitlabUploadError – If the file upload fails
- GitlabUploadError – If filedata and filepath are not specified
- GitlabUploadError – If both filedata and filepath are specified

Returns

A dict with the keys:

- alt - The alternate text for the upload

- `url` - The direct url to the uploaded file
- `markdown` - Markdown for the uploaded file

Return type dict

class gitlab.v3.objects.**ProjectAccessRequest**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

approve(*access_level=30*, ***kwargs*)
Approve an access request.

Parameters **access_level** (*int*) – The access level for the user.

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabUpdateError` – If the server fails to perform the request.

canGet = 'from_list'

canUpdate = False

class gitlab.v3.objects.**ProjectAccessRequestManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectAccessRequest'> objects.

<class 'gitlab.v3.objects.ProjectAccessRequest'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectAccessRequest'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectAccessRequest'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectAccessRequest'>.

`data` is a dict defining the object attributes. Available attributes are:

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectAccessRequest*

class gitlab.v3.objects.**ProjectBoard**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canGet = 'from_list'

canUpdate = False

managers = (('lists', 'ProjectBoardListManager', [('project_id', 'project_id'), ('boar

requiredUrlAttrs = ['project_id']

class gitlab.v3.objects.**ProjectBoardList**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `position` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

requiredCreateAttrs = ['label_id']

requiredUpdateAttrs = ['position']

requiredUrlAttrs = ['project_id', 'board_id']

class gitlab.v3.objects.**ProjectBoardListManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectBoardList'> objects.

<class 'gitlab.v3.objects.ProjectBoardList'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectBoardList'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectBoardList'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectBoardList'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)

- board_id (required if not discovered on the parent objects)
- label_id (required)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectBoardList*

class gitlab.v3.objects.**ProjectBoardManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectBoard'> objects.

<class 'gitlab.v3.objects.ProjectBoard'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectBoard'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectBoard'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectBoard*

class gitlab.v3.objects.**ProjectBranch**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

idAttr = 'name'

protect(*protect=True*, ***kwargs*)
Protects the branch.

requiredCreateAttrs = ['branch_name', 'ref']

requiredUrlAttrs = ['project_id']

unprotect(***kwargs*)
Unprotects the branch.

class gitlab.v3.objects.**ProjectBranchManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectBranch'> objects.

<class 'gitlab.v3.objects.ProjectBranch'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectBranch'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectBranch'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectBranch'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `branch_name` (required)
- `ref` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectBranch*

class gitlab.v3.objects.**ProjectBuild**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

artifacts(*streamed=False*, *action=None*, *chunk_size=1024*, ***kwargs*)
Get the build artifacts.

Parameters

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data.
- **chunk_size** (*int*) – Size of each chunk.

Returns The artifacts if *streamed* is False, None otherwise.

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the artifacts are not available.

canCreate = False

canDelete = False

canUpdate = False

cancel(***kwargs*)
Cancel the build.

erase(***kwargs*)
Erase the build (remove build artifacts and trace).

keep_artifacts(***kwargs*)
Prevent artifacts from being delete when expiration is set.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabCreateError – If the request failed.

play(***kwargs*)
Trigger a build explicitly.

requiredUrlAttrs = ['project_id']

retry(***kwargs*)
Retry the build.

trace(*streamed=False*, *action=None*, *chunk_size=1024*, ***kwargs*)
Get the build trace.

Parameters

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data.
- **chunk_size** (*int*) – Size of each chunk.

Returns The trace.

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the trace is not available.

class gitlab.v3.objects.**ProjectBuildManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectBuild'> objects.

<class 'gitlab.v3.objects.ProjectBuild'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectBuild'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.

- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get (*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectBuild'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectBuild*

class gitlab.v3.objects.**ProjectCommit** (*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

blob (*filepath*, *streamed=False*, *action=None*, *chunk_size=1024*, ***kwargs*)
Generate the content of a file for this commit.

Parameters

- **filepath** (*str*) – Path of the file to request.
- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data.
- **chunk_size** (*int*) – Size of each chunk.

Returns The content of the file

Return type str

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabGetError` – If the server fails to perform the request.

builds (***kwargs*)
List the build for this commit.

Returns A list of builds.

Return type *list*(*ProjectBuild*)

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabListError` – If the server fails to perform the request.

canDelete = False

canUpdate = False

cherry_pick (*branch*, ***kwargs*)
Cherry-pick a commit into a branch.

Parameters **branch** (*str*) – Name of target branch.

Raises `GitlabCherryPickError` – If the cherry pick could not be applied.

diff (***kwargs*)
Generate the commit diff.

managers = (('comments', 'ProjectCommitCommentManager', [('project_id', 'project_id'),

optionalCreateAttrs = ['author_email', 'author_name']

requiredCreateAttrs = ['branch_name', 'commit_message', 'actions']

requiredUrlAttrs = ['project_id']

shortPrintAttr = 'title'

class gitlab.v3.objects.**ProjectCommitComment**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canDelete = False

canGet = False

canUpdate = False

optionalCreateAttrs = ['path', 'line', 'line_type']

requiredCreateAttrs = ['note']

requiredUrlAttrs = ['project_id', 'commit_id']

class gitlab.v3.objects.**ProjectCommitCommentManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectCommitComment'> objects.

<class 'gitlab.v3.objects.ProjectCommitComment'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectCommitComment'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectCommitComment'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `commit_id` (required if not discovered on the parent objects)
- `note` (required)
- `path` (optional)
- `line` (optional)
- `line_type` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectCommitComment*

class gitlab.v3.objects.**ProjectCommitManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectCommit'> objects.

<class 'gitlab.v3.objects.ProjectCommit'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectCommit'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectCommit'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectCommit'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `branch_name` (required)
- `commit_message` (required)
- `actions` (required)
- `author_email` (optional)
- `author_name` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectCommit*

class gitlab.v3.objects.**ProjectCommitStatus**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canDelete = False

canUpdate = False

optionalCreateAttrs = ['description', 'name', 'context', 'ref', 'target_url']

optionalGetAttrs = ['ref_name', 'stage', 'name', 'all']

requiredCreateAttrs = ['state']

requiredUrlAttrs = ['project_id', 'commit_id']

class gitlab.v3.objects.**ProjectCommitStatusManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectCommitStatus'> objects.

<class 'gitlab.v3.objects.ProjectCommitStatus'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectCommitStatus'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectCommitStatus'> using its `id`.

Available keys for `kwargs` are:

- `ref_name` (optional)
- `stage` (optional)
- `name` (optional)
- `all` (optional)
- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectCommitStatus'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `commit_id` (required if not discovered on the parent objects)
- `state` (required)
- `description` (optional)
- `name` (optional)
- `context` (optional)
- `ref` (optional)
- `target_url` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectCommitStatus*

class gitlab.v3.objects.**ProjectDeployment**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canUpdate = False

class gitlab.v3.objects.**ProjectDeploymentManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectDeployment'> objects.

<class 'gitlab.v3.objects.ProjectDeployment'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectDeployment'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectDeployment'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectDeployment*

class gitlab.v3.objects.**ProjectEnvironment**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `name` (optional)
- `external_url` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

canGet = 'from_list'

optionalCreateAttrs = ['external_url']

optionalUpdateAttrs = ['name', 'external_url']

requiredCreateAttrs = ['name']

requiredUrlAttrs = ['project_id']

class gitlab.v3.objects.**ProjectEnvironmentManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectEnvironment'> objects.

<class 'gitlab.v3.objects.ProjectEnvironment'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectEnvironment'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.

- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectEnvironment'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectEnvironment'>.

data is a dict defining the object attributes. Available attributes are:

- project_id (required if not discovered on the parent objects)
- name (required)
- external_url (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectEnvironment*

class gitlab.v3.objects.**ProjectEvent**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canGet = 'from_list'

canUpdate = False

requiredUrlAttrs = ['project_id']

shortPrintAttr = 'target_title'

class gitlab.v3.objects.**ProjectEventManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectEvent'> objects.

<class 'gitlab.v3.objects.ProjectEvent'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectEvent'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve

- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectEvent'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectEvent*

class gitlab.v3.objects.**ProjectFile**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `file_path` (required)
- `branch_name` (required)
- `content` (required)
- `commit_message` (required)
- `encoding` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

canList = False

decode()
Returns the decoded content of the file.

Returns the decoded content.

Return type (str)

getRequiresId = False

optionalCreateAttrs = ['encoding']

requiredCreateAttrs = ['file_path', 'branch_name', 'content', 'commit_message']

requiredDeleteAttrs = ['branch_name', 'commit_message', 'file_path']

requiredGetAttrs = ['file_path', 'ref']

requiredUrlAttrs = ['project_id']

shortPrintAttr = 'file_path'

class gitlab.v3.objects.**ProjectFileManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectFile'> objects.

<class 'gitlab.v3.objects.ProjectFile'> objects can be updated.

get(***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectFile'>.

Available keys for `kwargs` are:

- `file_path` (required)
- `ref` (required)
- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectFile'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `file_path` (required)
- `branch_name` (required)
- `content` (required)
- `commit_message` (required)
- `encoding` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectFile*

class gitlab.v3.objects.**ProjectFork**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canDelete = False

canGet = False

canList = False

canUpdate = False

optionalCreateAttrs = ['namespace']

requiredUrlAttrs = ['project_id']

class gitlab.v3.objects.**ProjectForkManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectFork'> objects.

<class 'gitlab.v3.objects.ProjectFork'> objects **cannot** be updated.

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectFork'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `namespace` (optional)

Available keys for `kwargs` are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectFork*

class gitlab.v3.objects.**ProjectHook**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- url (required)
- push_events (optional)
- issues_events (optional)
- note_events (optional)
- merge_requests_events (optional)
- tag_push_events (optional)
- build_events (optional)
- enable_ssl_verification (optional)
- token (optional)
- pipeline_events (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

optionalCreateAttrs = ['push_events', 'issues_events', 'note_events', 'merge_requests_

requiredCreateAttrs = ['url']

requiredUrlAttrs = ['project_id']

shortPrintAttr = 'url'

class gitlab.v3.objects.**ProjectHookManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectHook'> objects.

<class 'gitlab.v3.objects.ProjectHook'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectHook'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectHook'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectHook'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `url` (required)
- `push_events` (optional)
- `issues_events` (optional)
- `note_events` (optional)
- `merge_requests_events` (optional)
- `tag_push_events` (optional)
- `build_events` (optional)
- `enable_ssl_verification` (optional)
- `token` (optional)
- `pipeline_events` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectHook*

class gitlab.v3.objects.**ProjectIssue**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

notes
ProjectIssueNoteManager - Manager for objects.

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `title` (optional)
- `description` (optional)
- `assignee_id` (optional)
- `milestone_id` (optional)
- `labels` (optional)
- `created_at` (optional)
- `updated_at` (optional)
- `state_event` (optional)
- `due_date` (optional)

Available keys for `kwargs` are:

- sudo (string or int): run the request as another user (requires admin permissions)

add_spent_time(***kwargs*)
Set an estimated time of work for the issue.

Raises GitlabConnectionError – If the server cannot be reached.

description_attr = 'description'

managers = (('notes', 'ProjectIssueNoteManager', [('project_id', 'project_id'), ('issu

move(*to_project_id*, ***kwargs*)
Move the issue to another project.

Raises GitlabConnectionError – If the server cannot be reached.

optionalCreateAttrs = ['description', 'assignee_id', 'milestone_id', 'labels', 'create

optionalListAttrs = ['state', 'labels', 'milestone', 'iid', 'order_by', 'sort']

optionalUpdateAttrs = ['title', 'description', 'assignee_id', 'milestone_id', 'labels'

project_id_attr = 'project_id'

requiredCreateAttrs = ['title']

requiredUrlAttrs = ['project_id']

reset_spent_time(***kwargs*)
Set an estimated time of work for the issue.

Raises GitlabConnectionError – If the server cannot be reached.

reset_time_estimate(***kwargs*)
Resets estimated time for the issue to 0 seconds.

Raises GitlabConnectionError – If the server cannot be reached.

shortPrintAttr = 'title'

subscribe(***kwargs*)
Subscribe to an issue.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabSubscribeError – If the subscription cannot be done

time_estimate(***kwargs*)
Set an estimated time of work for the issue.

Raises GitlabConnectionError – If the server cannot be reached.

time_stats(***kwargs*)
Get time stats for the issue.

Raises GitlabConnectionError – If the server cannot be reached.

todo(***kwargs*)
Create a todo for the issue.

Raises GitlabConnectionError – If the server cannot be reached.

unsubscribe(***kwargs*)
Unsubscribe an issue.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabUnsubscribeError – If the unsubscription cannot be done

class gitlab.v3.objects.**ProjectIssueManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectIssue'> objects.

<class 'gitlab.v3.objects.ProjectIssue'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectIssue'>.

Available keys for kwargs are:

- state (optional)
- labels (optional)
- milestone (optional)
- iid (optional)
- order_by (optional)
- sort (optional)
- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectIssue'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectIssue'>.

data is a dict defining the object attributes. Available attributes are:

- project_id (required if not discovered on the parent objects)
- title (required)
- description (optional)
- assignee_id (optional)
- milestone_id (optional)
- labels (optional)
- created_at (optional)
- due_date (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectIssue*

class gitlab.v3.objects.**ProjectIssueNote**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `body` (required)
- `created_at` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

canDelete = False

description_attr = 'body'

optionalCreateAttrs = ['created_at']

project_id_attr = 'project_id'

requiredCreateAttrs = ['body']

requiredUrlAttrs = ['project_id', 'issue_id']

class gitlab.v3.objects.**ProjectIssueNoteManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectIssueNote'> objects.

<class 'gitlab.v3.objects.ProjectIssueNote'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectIssueNote'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectIssueNote'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectIssueNote'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)

- `issue_id` (required if not discovered on the parent objects)
- `body` (required)
- `created_at` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectIssueNote*

class gitlab.v3.objects.**ProjectKey**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

requiredCreateAttrs = ['title', 'key']

requiredUrlAttrs = ['project_id']

class gitlab.v3.objects.**ProjectKeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectKey'> objects.

<class 'gitlab.v3.objects.ProjectKey'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectKey'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectKey'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectKey'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `title` (required)
- `key` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- sudo (string or int): run the request as another user (requires admin permissions)

disable(*key_id*)
Disable a deploy key for a project.

enable(*key_id*)
Enable a deploy key for a project.

obj_cls
alias of *ProjectKey*

class gitlab.v3.objects.**ProjectLabel**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- name (required)
- new_name (optional)
- color (optional)
- description (optional)
- priority (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

canGet = 'from_list'

idAttr = 'name'

optionalCreateAttrs = ['description', 'priority']

optionalUpdateAttrs = ['new_name', 'color', 'description', 'priority']

requiredCreateAttrs = ['name', 'color']

requiredDeleteAttrs = ['name']

requiredUpdateAttrs = ['name']

requiredUrlAttrs = ['project_id']

subscribe(***kwargs*)
Subscribe to a label.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabSubscribeError – If the subscription cannot be done

unsubscribe(***kwargs*)
Unsubscribe a label.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabUnsubscribeError – If the unsubscription cannot be done

class gitlab.v3.objects.**ProjectLabelManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectLabel'> objects.

<class 'gitlab.v3.objects.ProjectLabel'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectLabel'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectLabel'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectLabel'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `name` (required)
- `color` (required)
- `description` (optional)
- `priority` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectLabel*

class gitlab.v3.objects.**ProjectManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Project'> objects.

<class 'gitlab.v3.objects.Project'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Project'>.

Available keys for `kwargs` are:

- `search` (optional)
- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve

- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Project'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.Project'>.

`data` is a dict defining the object attributes. Available attributes are:

- `name` (required)
- `path` (optional)
- `namespace_id` (optional)
- `description` (optional)
- `issues_enabled` (optional)
- `merge_requests_enabled` (optional)
- `builds_enabled` (optional)
- `wiki_enabled` (optional)
- `snippets_enabled` (optional)
- `container_registry_enabled` (optional)
- `shared_runners_enabled` (optional)
- `public` (optional)
- `visibility_level` (optional)
- `import_url` (optional)
- `public_builds` (optional)
- `only_allow_merge_if_build_succeeds` (optional)
- `only_allow_merge_if_all_discussions_are_resolved` (optional)
- `lfs_enabled` (optional)
- `request_access_enabled` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

all(***kwargs*)
List all the projects (need admin rights).

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- ****kwargs** – Additional arguments to send to GitLab.

Returns The list of projects.

Return type *list*(gitlab.Gitlab.Project)

obj_cls
alias of *Project*

owned(***kwargs*)
List owned projects.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- ****kwargs** – Additional arguments to send to GitLab.

Returns The list of owned projects.

Return type *list*(gitlab.Gitlab.Project)

search(*query*, ***kwargs*)
Search projects by name.

API v3 only.

Note: The search is only performed on the project name (not on the namespace or the description). To perform a smarter search, use the `search` argument of the `list()` method:

```
gl.projects.list(search=your_search_string)
```

Parameters

- **query** (*str*) – The query string to send to GitLab for the search.
- **all** (*bool*) – If True, return all the items, without pagination
- ****kwargs** – Additional arguments to send to GitLab.

Returns A list of matching projects.

Return type *list*(gitlab.Gitlab.Project)

starred(***kwargs*)
List starred projects.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- ****kwargs** – Additional arguments to send to GitLab.

Returns The list of starred projects.

Return type *list*(gitlab.Gitlab.Project)

class gitlab.v3.objects.**ProjectMember**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `access_level` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

optionalCreateAttrs = ['expires_at']

requiredCreateAttrs = ['access_level', 'user_id']

requiredUpdateAttrs = ['access_level']

requiredUrlAttrs = ['project_id']

shortPrintAttr = 'username'

class gitlab.v3.objects.**ProjectMemberManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectMember'> objects.

<class 'gitlab.v3.objects.ProjectMember'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectMember'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectMember'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectMember'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `access_level` (required)
- `user_id` (required)
- `expires_at` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectMember*

class gitlab.v3.objects.**ProjectMergeRequest**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

notes
ProjectMergeRequestNoteManager - Manager for objects.

diffs
ProjectMergeRequestDiffManager - Manager for objects.

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `target_branch` (optional)
- `assignee_id` (optional)
- `title` (optional)
- `description` (optional)
- `state_event` (optional)
- `labels` (optional)
- `milestone_id` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

add_spent_time(***kwargs*)
Set an estimated time of work for the merge request.

Raises `GitlabConnectionError` – If the server cannot be reached.

cancel_merge_when_build_succeeds(***kwargs*)
Cancel merge when build succeeds.

changes(***kwargs*)
List the merge request changes.

Returns List of changes

Return type *list* (dict)

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabListError` – If the server fails to perform the request.

closes_issues(***kwargs*)
List issues closed by the MR.

Returns List of closed issues

Return type *list* (*ProjectIssue*)

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabGetError` – If the server fails to perform the request.

commits(***kwargs*)
List the merge request commits.

Returns List of commits

Return type *list* (*ProjectCommit*)

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabListError` – If the server fails to perform the request.

managers = (('notes', 'ProjectMergeRequestNoteManager', [('project_id', 'project_id'),

merge(*merge_commit_message=None*, *should_remove_source_branch=False*, *merge_when_build_succeeds=False*, ***kwargs*)

Accept the merge request.

Parameters

- **merge_commit_message** (*bool*) – Commit message
- **should_remove_source_branch** (*bool*) – If True, removes the source branch
- **merge_when_build_succeeds** (*bool*) – Wait for the build to succeed, then merge

Returns The updated MR

Return type *ProjectMergeRequest*

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabMRForbiddenError` – If the user doesn't have permission to close thr MR
- `GitlabMRClosedError` – If the MR is already closed

optionalCreateAttrs = ['assignee_id', 'description', 'target_project_id', 'labels', 'm

optionalListAttrs = ['iid', 'state', 'order_by', 'sort']

optionalUpdateAttrs = ['target_branch', 'assignee_id', 'title', 'description', 'state_

requiredCreateAttrs = ['source_branch', 'target_branch', 'title']

requiredUrlAttrs = ['project_id']

reset_spent_time(***kwargs*)

Set an estimated time of work for the merge request.

Raises `GitlabConnectionError` – If the server cannot be reached.

reset_time_estimate(***kwargs*)

Resets estimated time for the merge request to 0 seconds.

Raises `GitlabConnectionError` – If the server cannot be reached.

subscribe(***kwargs*)

Subscribe to a MR.

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabSubscribeError` – If the subscription cannot be done

time_estimate(***kwargs*)

Set an estimated time of work for the merge request.

Raises `GitlabConnectionError` – If the server cannot be reached.

time_stats(***kwargs*)

Get time stats for the merge request.

Raises GitlabConnectionError – If the server cannot be reached.

todo(**kwargs)
Create a todo for the merge request.

Raises GitlabConnectionError – If the server cannot be reached.

unsubscribe(**kwargs)
Unsubscribe a MR.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabUnsubscribeError – If the unsubscription cannot be done

class gitlab.v3.objects.**ProjectMergeRequestDiff**(*gl*, *data=None*, **kwargs)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canDelete = False

canUpdate = False

requiredUrlAttrs = ['project_id', 'merge_request_id']

class gitlab.v3.objects.**ProjectMergeRequestDiffManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectMergeRequestDiff'> objects.

<class 'gitlab.v3.objects.ProjectMergeRequestDiff'> objects **cannot** be updated.

list(**kwargs)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectMergeRequestDiff'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, **kwargs)
Get a single object of type <class 'gitlab.v3.objects.ProjectMergeRequestDiff'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectMergeRequestDiff*

class gitlab.v3.objects.**ProjectMergeRequestManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectMergeRequest'> objects.

<class 'gitlab.v3.objects.ProjectMergeRequest'> objects can be updated.

list(**kwargs)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectMergeRequest'>.

Available keys for kwargs are:

- `iid` (optional)
- `state` (optional)
- `order_by` (optional)
- `sort` (optional)
- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectMergeRequest'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectMergeRequest'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `source_branch` (required)
- `target_branch` (required)
- `title` (required)
- `assignee_id` (optional)
- `description` (optional)
- `target_project_id` (optional)
- `labels` (optional)
- `milestone_id` (optional)
- `remove_source_branch` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectMergeRequest*

class gitlab.v3.objects.**ProjectMergeRequestNote**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `body` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

requiredCreateAttrs = ['body']

requiredUrlAttrs = ['project_id', 'merge_request_id']

class gitlab.v3.objects.**ProjectMergeRequestNoteManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectMergeRequestNote'> objects.

<class 'gitlab.v3.objects.ProjectMergeRequestNote'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectMergeRequestNote'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectMergeRequestNote'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectMergeRequestNote'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `merge_request_id` (required if not discovered on the parent objects)
- `body` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectMergeRequestNote*

class gitlab.v3.objects.**ProjectMilestone**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `title` (optional)
- `description` (optional)

- due_date (optional)
- start_date (optional)
- state_event (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

canDelete = False

issues(***kwargs*)

merge_requests(***kwargs*)

List the merge requests related to this milestone

Returns List of merge requests

Return type *list* (*ProjectMergeRequest*)

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabListError – If the server fails to perform the request.

optionalCreateAttrs = ['description', 'due_date', 'start_date', 'state_event']

optionalListAttrs = ['iid', 'state']

optionalUpdateAttrs = ['title', 'description', 'due_date', 'start_date', 'state_event'

requiredCreateAttrs = ['title']

requiredUrlAttrs = ['project_id']

shortPrintAttr = 'title'

class gitlab.v3.objects.**ProjectMilestoneManager**(*gl*, *parent=None*, *args=[]*)

Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectMilestone'> objects.

<class 'gitlab.v3.objects.ProjectMilestone'> objects can be updated.

list(***kwargs*)

Returns a list of objects of type <class 'gitlab.v3.objects.ProjectMilestone'>.

Available keys for kwargs are:

- iid (optional)
- state (optional)
- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)

Get a single object of type <class 'gitlab.v3.objects.ProjectMilestone'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectMilestone'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `title` (required)
- `description` (optional)
- `due_date` (optional)
- `start_date` (optional)
- `state_event` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectMilestone*

class gitlab.v3.objects.**ProjectNote**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canDelete = False

canUpdate = False

requiredCreateAttrs = ['body']

requiredUrlAttrs = ['project_id']

class gitlab.v3.objects.**ProjectNoteManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectNote'> objects.

<class 'gitlab.v3.objects.ProjectNote'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectNote'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectNote'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectNote'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)

- `body` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectNote*

class gitlab.v3.objects.**ProjectNotificationSettings**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.v3.objects.NotificationSettings*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `level` (optional)
- `notification_email` (optional)
- `new_note` (optional)
- `new_issue` (optional)
- `reopen_issue` (optional)
- `close_issue` (optional)
- `reassign_issue` (optional)
- `new_merge_request` (optional)
- `reopen_merge_request` (optional)
- `close_merge_request` (optional)
- `reassign_merge_request` (optional)
- `merge_merge_request` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

requiredUrlAttrs = ['project_id']

class gitlab.v3.objects.**ProjectNotificationSettingsManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectNotificationSettings'> objects.

<class 'gitlab.v3.objects.ProjectNotificationSettings'> objects can be updated.

get(***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectNotificationSettings'>.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectNotificationSettings*

class gitlab.v3.objects.**ProjectPipeline**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canDelete = False

canUpdate = False

cancel(***kwargs*)
Cancel builds in a pipeline.

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabPipelineCancelError` – If the retry cannot be done.

requiredCreateAttrs = ['ref']

requiredUrlAttrs = ['project_id']

retry(***kwargs*)
Retries failed builds in a pipeline.

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabPipelineRetryError` – If the retry cannot be done.

class gitlab.v3.objects.**ProjectPipelineManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectPipeline'> objects.

<class 'gitlab.v3.objects.ProjectPipeline'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectPipeline'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectPipeline'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectPipeline'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `ref` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectPipeline*

class gitlab.v3.objects.**ProjectRunner**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

requiredCreateAttrs = ['runner_id']

class gitlab.v3.objects.**ProjectRunnerManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectRunner'> objects.

<class 'gitlab.v3.objects.ProjectRunner'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectRunner'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectRunner'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectRunner'>.

`data` is a dict defining the object attributes. Available attributes are:

- `runner_id` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectRunner*

class gitlab.v3.objects.**ProjectService**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

canCreate = False

canList = False

getRequiresId = False

requiredUrlAttrs = ['project_id', 'service_name']

class gitlab.v3.objects.**ProjectServiceManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectService'> objects.

<class 'gitlab.v3.objects.ProjectService'> objects can be updated.

get(***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectService'>.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

available(***kwargs*)
List the services known by python-gitlab.

Returns The list of service code names.

Return type *list* (str)

obj_cls
alias of *ProjectService*

class gitlab.v3.objects.**ProjectSnippet**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

notes
ProjectSnippetNoteManager - Manager for objects.

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `title` (optional)
- `file_name` (optional)
- `code` (optional)
- `visibility_level` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

content(*streamed=False*, *action=None*, *chunk_size=1024*, ***kwargs*)
Return the raw content of a snippet.

Parameters

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data.
- **chunk_size** (*int*) – Size of each chunk.

Returns The snippet content

Return type str

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabGetError` – If the server fails to perform the request.

```
managers = (('notes', 'ProjectSnippetNoteManager', [('project_id', 'project_id'), ('sn
optionalCreateAttrs = ['lifetime', 'visibility_level']
optionalUpdateAttrs = ['title', 'file_name', 'code', 'visibility_level']
requiredCreateAttrs = ['title', 'file_name', 'code']
requiredUrlAttrs = ['project_id']
shortPrintAttr = 'title'
```

class gitlab.v3.objects.**ProjectSnippetManager**(*gl*, *parent=None*, *args=[]*)

Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectSnippet'> objects.

<class 'gitlab.v3.objects.ProjectSnippet'> objects can be updated.

list(***kwargs*)

Returns a list of objects of type <class 'gitlab.v3.objects.ProjectSnippet'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)

Get a single object of type <class 'gitlab.v3.objects.ProjectSnippet'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)

Create an object of type <class 'gitlab.v3.objects.ProjectSnippet'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `title` (required)
- `file_name` (required)
- `code` (required)
- `lifetime` (optional)
- `visibility_level` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)

Delete the object with ID `id`.

Available keys for `kwargs` are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectSnippet*

class gitlab.v3.objects.**ProjectSnippetNote**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canDelete = False

canUpdate = False

requiredCreateAttrs = ['body']

requiredUrlAttrs = ['project_id', 'snippet_id']

class gitlab.v3.objects.**ProjectSnippetNoteManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectSnippetNote'> objects.

<class 'gitlab.v3.objects.ProjectSnippetNote'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectSnippetNote'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectSnippetNote'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectSnippetNote'>.

data is a dict defining the object attributes. Available attributes are:

- project_id (required if not discovered on the parent objects)
- snippet_id (required if not discovered on the parent objects)
- body (required)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectSnippetNote*

class gitlab.v3.objects.**ProjectTag**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canGet = 'from_list'

canUpdate = False

idAttr = 'name'

optionalCreateAttrs = ['message']

requiredCreateAttrs = ['tag_name', 'ref']

requiredUrlAttrs = ['project_id']

set_release_description(*description*)
Set the release notes on the tag.

If the release doesn't exist yet, it will be created. If it already exists, its description will be updated.

Parameters **description** (*str*) – Description of the release.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabCreateError – If the server fails to create the release.
- GitlabUpdateError – If the server fails to update the release.

shortPrintAttr = 'name'

class gitlab.v3.objects.**ProjectTagManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectTag'> objects.

<class 'gitlab.v3.objects.ProjectTag'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectTag'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectTag'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectTag'>.

data is a dict defining the object attributes. Available attributes are:

- project_id (required if not discovered on the parent objects)
- tag_name (required)
- ref (required)
- message (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectTag*

class gitlab.v3.objects.**ProjectTagRelease**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `description` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

canDelete = False

canList = False

requiredCreateAttrs = ['description']

requiredUrlAttrs = ['project_id', 'tag_name']

shortPrintAttr = 'description'

class gitlab.v3.objects.**ProjectTrigger**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

idAttr = 'token'

requiredUrlAttrs = ['project_id']

class gitlab.v3.objects.**ProjectTriggerManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectTrigger'> objects.

<class 'gitlab.v3.objects.ProjectTrigger'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectTrigger'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectTrigger'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectTrigger'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectTrigger*

class gitlab.v3.objects.**ProjectVariable**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `key` (required)
- `value` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

idAttr = 'key'

requiredCreateAttrs = ['key', 'value']

requiredUrlAttrs = ['project_id']

class gitlab.v3.objects.**ProjectVariableManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.ProjectVariable'> objects.

<class 'gitlab.v3.objects.ProjectVariable'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.ProjectVariable'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.ProjectVariable'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.ProjectVariable'>.

`data` is a dict defining the object attributes. Available attributes are:

- `project_id` (required if not discovered on the parent objects)
- `key` (required)
- `value` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *ProjectVariable*

class gitlab.v3.objects.**Runner**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `description` (optional)
- `active` (optional)
- `tag_list` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

canCreate = False

optionalListAttrs = ['scope']

optionalUpdateAttrs = ['description', 'active', 'tag_list']

class gitlab.v3.objects.**RunnerManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Runner'> objects.

<class 'gitlab.v3.objects.Runner'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Runner'>.

Available keys for `kwargs` are:

- `scope` (optional)
- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Runner'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

all(*scope=None*, ***kwargs*)
List all the runners.

Parameters **scope** (*str*) – The scope of runners to show, one of: specific, shared, active, paused, online

Returns a list of runners matching the scope.

Return type *list*(*Runner*)

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabListError` – If the resource cannot be found

obj_cls
alias of *Runner*

class gitlab.v3.objects.**SidekiqManager**(*gl*)
Bases: `object`

Manager for the Sidekiq methods.

This manager doesn't actually manage objects but provides helper fonction for the sidekiq metrics API.

compound_metrics(***kwargs*)
Returns all available metrics and statistics.

job_stats(***kwargs*)
Returns statistics about the jobs performed.

process_metrics(***kwargs*)
Returns the registred sidekiq workers.

queue_metrics(***kwargs*)
Returns the registred queues information.

class gitlab.v3.objects.**Snippet**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- `title` (optional)
- `file_name` (optional)
- `content` (optional)
- `visibility_level` (optional)

Available keys for `kwargs` are:

- sudo (string or int): run the request as another user (requires admin permissions)

optionalCreateAttrs = ['lifetime', 'visibility_level']

optionalUpdateAttrs = ['title', 'file_name', 'content', 'visibility_level']

raw(*streamed=False*, *action=None*, *chunk_size=1024*, ***kwargs*)
Return the raw content of a snippet.

Parameters

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data.
- **chunk_size** (*int*) – Size of each chunk.

Returns The snippet content.

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the server fails to perform the request.

requiredCreateAttrs = ['title', 'file_name', 'content']

shortPrintAttr = 'title'

class gitlab.v3.objects.**SnippetManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Snippet'> objects.

<class 'gitlab.v3.objects.Snippet'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Snippet'>.

Available keys for kwargs are:

- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Snippet'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.Snippet'>.

data is a dict defining the object attributes. Available attributes are:

- title (required)
- file_name (required)
- content (required)

- `lifetime` (optional)
- `visibility_level` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *Snippet*

public(***kwargs*)
List all the public snippets.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- ****kwargs** – Additional arguments to send to GitLab.

Returns The list of snippets.

Return type *list*(gitlab.Gitlab.Snippet)

class gitlab.v3.objects.**Team**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

managers = (('members', 'TeamMemberManager', [('team_id', 'id')]), ('projects', 'TeamP

requiredCreateAttrs = ['name', 'path']

shortPrintAttr = 'name'

class gitlab.v3.objects.**TeamManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Team'> objects.

<class 'gitlab.v3.objects.Team'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Team'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Team'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.Team'>.

`data` is a dict defining the object attributes. Available attributes are:

- `name` (required)
- `path` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *Team*

class gitlab.v3.objects.**TeamMember**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

requiredCreateAttrs = ['access_level']

requiredUrlAttrs = ['teamd_id']

shortPrintAttr = 'username'

class gitlab.v3.objects.**TeamMemberManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.TeamMember'> objects.

<class 'gitlab.v3.objects.TeamMember'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.TeamMember'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.TeamMember'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.TeamMember'>.

`data` is a dict defining the object attributes. Available attributes are:

- `teamd_id` (required if not discovered on the parent objects)
- `access_level` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *TeamMember*

class gitlab.v3.objects.**TeamProject**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

requiredCreateAttrs = ['greatest_access_level']

requiredUrlAttrs = ['team_id']

shortPrintAttr = 'name'

class gitlab.v3.objects.**TeamProjectManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.TeamProject'> objects.

<class 'gitlab.v3.objects.TeamProject'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.TeamProject'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.TeamProject'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.TeamProject'>.

`data` is a dict defining the object attributes. Available attributes are:

- `team_id` (required if not discovered on the parent objects)
- `greatest_access_level` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- sudo (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *TeamProject*

class gitlab.v3.objects.**Todo**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canCreate = False

canGet = 'from_list'

canUpdate = False

optionalListAttrs = ['action', 'author_id', 'project_id', 'state', 'type']

class gitlab.v3.objects.**TodoManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.Todo'> objects.

<class 'gitlab.v3.objects.Todo'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.Todo'>.

Available keys for kwargs are:

- action (optional)
- author_id (optional)
- project_id (optional)
- state (optional)
- type (optional)
- per_page (int): number of item per page. May be limited by the server.
- page (int): page to retrieve
- all (bool): iterate over all the pages and return all the entries
- sudo (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.Todo'> using its id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID id.

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

delete_all(***kwargs*)
Mark all the todos as done.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabDeleteError – If the resource cannot be found

Returns The number of todos maked done.

obj_cls
alias of *Todo*

class gitlab.v3.objects.**User**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

emails
UserEmailManager - Manager for objects.

keys
UserKeyManager - Manager for objects.

projects
UserProjectManager - Manager for objects.

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

- email (required)
- username (required)
- name (required)
- password (optional)
- skype (optional)
- linkedin (optional)
- twitter (optional)
- projects_limit (optional)
- extern_uid (optional)
- provider (optional)
- bio (optional)
- admin (optional)
- can_create_group (optional)
- website_url (optional)
- confirm (optional)
- external (optional)
- organization (optional)
- location (optional)

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

block(***kwargs*)
Blocks the user.

managers = (('emails', 'UserEmailManager', [('user_id', 'id')]), ('keys', 'UserKeyMana

optionalCreateAttrs = ['password', 'reset_password', 'skype', 'linkedin', 'twitter', '

optionalUpdateAttrs = ['password', 'skype', 'linkedin', 'twitter', 'projects_limit', '

requiredCreateAttrs = ['email', 'username', 'name']

requiredUpdateAttrs = ['email', 'username', 'name']

shortPrintAttr = 'username'

unblock(***kwargs*)
Unblocks the user.

class gitlab.v3.objects.**UserEmail**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canUpdate = False

requiredCreateAttrs = ['email']

requiredUrlAttrs = ['user_id']

shortPrintAttr = 'email'

class gitlab.v3.objects.**UserEmailManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.UserEmail'> objects.

<class 'gitlab.v3.objects.UserEmail'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.UserEmail'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.UserEmail'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.UserEmail'>.

`data` is a dict defining the object attributes. Available attributes are:

- `user_id` (required if not discovered on the parent objects)
- `email` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *UserEmail*

class gitlab.v3.objects.**UserKey**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canGet = 'from_list'

canUpdate = False

requiredCreateAttrs = ['title', 'key']

requiredUrlAttrs = ['user_id']

class gitlab.v3.objects.**UserKeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.UserKey'> objects.

<class 'gitlab.v3.objects.UserKey'> objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.UserKey'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.UserKey'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.UserKey'>.

`data` is a dict defining the object attributes. Available attributes are:

- `user_id` (required if not discovered on the parent objects)
- `title` (required)
- `key` (required)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *UserKey*

class gitlab.v3.objects.**UserManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.User'> objects.

<class 'gitlab.v3.objects.User'> objects can be updated.

list(***kwargs*)
Returns a list of objects of type <class 'gitlab.v3.objects.User'>.

Available keys for `kwargs` are:

- `per_page` (int): number of item per page. May be limited by the server.
- `page` (int): page to retrieve
- `all` (bool): iterate over all the pages and return all the entries
- `sudo` (string or int): run the request as another user (requires admin permissions)

get(*id*, ***kwargs*)
Get a single object of type <class 'gitlab.v3.objects.User'> using its `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.User'>.

`data` is a dict defining the object attributes. Available attributes are:

- `email` (required)
- `username` (required)
- `name` (required)
- `password` (optional)
- `reset_password` (optional)
- `skype` (optional)
- `linkedin` (optional)
- `twitter` (optional)
- `projects_limit` (optional)
- `extern_uid` (optional)
- `provider` (optional)
- `bio` (optional)
- `admin` (optional)
- `can_create_group` (optional)
- `website_url` (optional)
- `confirm` (optional)
- `external` (optional)
- `organization` (optional)
- `location` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

delete(*id*, ***kwargs*)
Delete the object with ID `id`.

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

get_by_username(*username*, ***kwargs*)
Get a user by its username.

Parameters

- **username** (*str*) – The name of the user.
- ****kwargs** – Additional arguments to send to GitLab.

Returns The matching user.

Return type *User*

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabGetError` – If the server fails to perform the request.

obj_cls
alias of *User*

search(*query*, ***kwargs*)
Search users.

Parameters

- **query** (*str*) – The query string to send to GitLab for the search.
- **all** (*bool*) – If True, return all the items, without pagination
- ****kwargs** – Additional arguments to send to GitLab.

Returns A list of matching users.

Return type *list*(*User*)

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabListError` – If the server fails to perform the request.

class gitlab.v3.objects.**UserProject**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

canDelete = False

canGet = False

canList = False

canUpdate = False

optionalCreateAttrs = ['default_branch', 'issues_enabled', 'wall_enabled', 'merge_requ

requiredCreateAttrs = ['name']

requiredUrlAttrs = ['user_id']

class gitlab.v3.objects.**UserProjectManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for <class 'gitlab.v3.objects.UserProject'> objects.

<class 'gitlab.v3.objects.UserProject'> objects **cannot** be updated.

create(*data*, ***kwargs*)
Create an object of type <class 'gitlab.v3.objects.UserProject'>.

`data` is a dict defining the object attributes. Available attributes are:

- `user_id` (required if not discovered on the parent objects)
- `name` (required)
- `default_branch` (optional)
- `issues_enabled` (optional)
- `wall_enabled` (optional)
- `merge_requests_enabled` (optional)
- `wiki_enabled` (optional)
- `snippets_enabled` (optional)
- `public` (optional)
- `visibility_level` (optional)
- `description` (optional)
- `builds_enabled` (optional)
- `public_builds` (optional)
- `import_url` (optional)
- `only_allow_merge_if_build_succeeds` (optional)

Available keys for `kwargs` are:

- `sudo` (string or int): run the request as another user (requires admin permissions)

obj_cls
alias of *UserProject*

Module contents

6.1.2 gitlab.v4 package

Submodules

gitlab.v4.objects module

class gitlab.v4.objects.**ApplicationSettings**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ApplicationSettingsManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetWithoutIdMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**BroadcastMessage**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**BroadcastMessageManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**CurrentUser**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**CurrentUserEmail**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**CurrentUserEmailManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**CurrentUserGPGKey**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**CurrentUserGPGKeyManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**CurrentUserKey**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**CurrentUserKeyManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**CurrentUserManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetWithoutIdMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**DeployKey**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**DeployKeyManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Dockerfile**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**DockerfileManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Event**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**EventManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.ListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Feature**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**FeatureManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.ListMixin*, *gitlab.base.RESTManager*

set(**args*, ***kwargs*)
Create or update the object.

Parameters

- **name** (*str*) – The value to set for the object

- **value** (*bool/int*) – The value to set for the object
- **feature_group** (*str*) – A feature group name
- **user** (*str*) – A GitLab username
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabSetError – If an error occured

Returns The created/updated attribute

Return type obj

class gitlab.v4.objects.**Gitignore**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**GitignoreManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Gitlabciyml**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**GitlabciymlManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Group**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

transfer_project(**args*, ***kwargs*)
Transfer a project to this group.

Parameters

- **to_project_id** (*int*) – ID of the project to transfer
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTransferProjectError – If the project could not be transfered

class gitlab.v4.objects.**GroupAccessRequest**(*manager*, *attrs*)
Bases: *gitlab.mixins.AccessRequestMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**GroupAccessRequestManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupCustomAttribute**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**GroupCustomAttributeManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.SetMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupIssue**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**GroupIssueManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupMember**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**GroupMemberManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupMergeRequest**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**GroupMergeRequestManager**(*gl*, *parent=None*)
Bases: *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupMilestone**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

issues(**args*, ***kwargs*)
List issues related to this milestone.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the list could not be retrieved

Returns The list of issues

Return type *RESTObjectList*

merge_requests(**args*, ***kwargs*)
List the merge requests related to this milestone.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the list could not be retrieved

Returns The list of merge requests

Return type *RESTObjectList*

class gitlab.v4.objects.**GroupMilestoneManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupNotificationSettings**(*manager*, *attrs*)
Bases: *gitlab.v4.objects.NotificationSettings*

class gitlab.v4.objects.**GroupNotificationSettingsManager**(*gl*, *parent=None*)
Bases: *gitlab.v4.objects.NotificationSettingsManager*

class gitlab.v4.objects.**GroupProject**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**GroupProjectManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupSubgroup**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**GroupSubgroupManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**GroupVariable**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**GroupVariableManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Hook**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**HookManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Issue**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**IssueManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**License**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**LicenseManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Namespace**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**NamespaceManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**NotificationSettings**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**NotificationSettingsManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetWithoutIdMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**PagesDomain**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**PagesDomainManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.ListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Project**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

archive(**args*, ***kwargs*)
Archive a project.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the server failed to perform the request

create_fork_relation(**args*, ***kwargs*)
Create a forked from/to relation between existing projects.

Parameters

- **forked_from_id** (*int*) – The ID of the project that was forked from
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the relation could not be created

delete_fork_relation(**args*, ***kwargs*)
Delete a forked relation between existing projects.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabDeleteError – If the server failed to perform the request

housekeeping(**args*, ***kwargs*)
Start the housekeeping task.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabHousekeepingError – If the server failed to perform the request

repository_archive(**args*, ***kwargs*)
Return a tarball of the repository.

Parameters

- **sha** (*str*) – ID of the commit (default branch by default)

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment
- **action** (*callable*) – Callable responsible of dealing with chunk of data
- **chunk_size** (*int*) – Size of each chunk
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the server failed to perform the request

Returns The binary data of the archive

Return type str

repository_blob(**args*, ***kwargs*)
Return a file by blob SHA.

Parameters

- **sha** (*str*) – ID of the blob
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server failed to perform the request

Returns The blob content and metadata

Return type dict

repository_compare(**args*, ***kwargs*)
Return a diff between two branches/commits.

Parameters

- **from** (*str*) – Source branch/SHA
- **to** (*str*) – Destination branch/SHA
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server failed to perform the request

Returns The diff

Return type str

repository_contributors(**args*, ***kwargs*)
Return a list of contributors for the project.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)

- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server failed to perform the request

Returns The contributors

Return type *list*

repository_raw_blob(**args*, ***kwargs*)
Return the raw file contents for a blob.

Parameters

- **sha** (*str*) – ID of the blob
- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment
- **action** (*callable*) – Callable responsible of dealing with chunk of data
- **chunk_size** (*int*) – Size of each chunk
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server failed to perform the request

Returns The blob content if streamed is False, None otherwise

Return type str

repository_tree(**args*, ***kwargs*)
Return a list of files in the repository.

Parameters

- **path** (*str*) – Path of the top folder (/ by default)
- **ref** (*str*) – Reference to a commit or branch
- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server failed to perform the request

Returns The representation of the tree

Return type *list*

share (**args*, ***kwargs*)
Share the project with a group.

Parameters

- **group_id** (*int*) – ID of the group.
- **group_access** (*int*) – Access level for the group.
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the server failed to perform the request

star (**args*, ***kwargs*)
Star a project.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the server failed to perform the request

trigger_pipeline (**args*, ***kwargs*)
Trigger a CI build.

See https://gitlab.com/help/ci/triggers/README.md#trigger-a-build

Parameters

- **ref** (*str*) – Commit to build; can be a branch name or a tag
- **token** (*str*) – The trigger token
- **variables** (*dict*) – Variables passed to the build script
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the server failed to perform the request

unarchive (**args*, ***kwargs*)
Unarchive a project.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabDeleteError – If the server failed to perform the request

unstar (**args*, ***kwargs*)
Unstar a project.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabDeleteError – If the server failed to perform the request

upload(**args*, ***kwargs*)
Upload the specified file into the project.

Note: Either `filedata` or `filepath` *MUST* be specified.

Parameters

- **filename** (*str*) – The name of the file being uploaded
- **filedata** (*bytes*) – The raw data of the file being uploaded
- **filepath** (*str*) – The path to a local file to upload (optional)

Raises

- `GitlabConnectionError` – If the server cannot be reached
- `GitlabUploadError` – If the file upload fails
- `GitlabUploadError` – If `filedata` and `filepath` are not specified
- `GitlabUploadError` – If both `filedata` and `filepath` are specified

Returns

A `dict` with the keys:

- `alt` - The alternate text for the upload
- `url` - The direct url to the uploaded file
- `markdown` - Markdown for the uploaded file

Return type dict

class gitlab.v4.objects.**ProjectAccessRequest**(*manager*, *attrs*)
Bases: *gitlab.mixins.AccessRequestMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectAccessRequestManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectBoard**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectBoardList**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectBoardListManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectBoardManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectBranch**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

protect(**args*, ***kwargs*)
Protect the branch.

Parameters

- **developers_can_push** (*bool*) – Set to True if developers are allowed to push to the branch
- **developers_can_merge** (*bool*) – Set to True if developers are allowed to merge to the branch
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabProtectError – If the branch could not be protected

unprotect(**args*, ***kwargs*)
Unprotect the branch.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabProtectError – If the branch could not be unprotected

class gitlab.v4.objects.**ProjectBranchManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectCommit**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

cherry_pick(**args*, ***kwargs*)
Cherry-pick a commit into a branch.

Parameters

- **branch** (*str*) – Name of target branch
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCherryPickError – If the cherry-pick could not be performed

diff(**args*, ***kwargs*)
Generate the commit diff.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the diff could not be retrieved

Returns The changes done in this commit

Return type *list*

class gitlab.v4.objects.**ProjectCommitComment**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectCommitCommentManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.ListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectCommitManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.CreateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectCommitStatus**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectCommitStatusManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.base.RESTManager*

create(*data*, ***kwargs*)
Create a new object.

Parameters

- **data** (*dict*) – Parameters to send to the server to create the resource
- ****kwargs** – Extra data to send to the Gitlab server (e.g. sudo or 'ref_name', 'stage', 'name', 'all'.

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the server cannot perform the request

Returns

A new instance of the manage object class build with the data sent by the server

Return type *RESTObject*

class gitlab.v4.objects.**ProjectCustomAttribute**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectCustomAttributeManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.SetMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectDeployment**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectDeploymentManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectEnvironment**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectEnvironmentManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectEvent**(*manager*, *attrs*)
Bases: *gitlab.v4.objects.Event*

class gitlab.v4.objects.**ProjectEventManager**(*gl*, *parent=None*)
Bases: *gitlab.v4.objects.EventManager*

class gitlab.v4.objects.**ProjectFile**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

decode()
Returns the decoded content of the file.

Returns the decoded content.

Return type (str)

delete(*branch*, *commit_message*, ***kwargs*)
Delete the file from the server.

Parameters

- **branch** (*str*) – Branch from which the file will be removed
- **commit_message** (*str*) – Commit message for the deletion
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabDeleteError – If the server cannot perform the request

save(*branch*, *commit_message*, ***kwargs*)
Save the changes made to the file to the server.

The object is updated to match what the server returns.

Parameters

- **branch** (*str*) – Branch in which the file will be updated
- **commit_message** (*str*) – Message to send with the commit
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabUpdateError – If the server cannot perform the request

class gitlab.v4.objects.**ProjectFileManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

create(**args*, ***kwargs*)
Create a new object.

Parameters

- **data** (*dict*) – parameters to send to the server to create the resource
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns

a new instance of the managed object class built with the data sent by the server

Return type *RESTObject*

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the server cannot perform the request

delete (**args*, ***kwargs*)
Delete a file on the server.

Parameters

- **file_path** (*str*) – Path of the file to remove
- **branch** (*str*) – Branch from which the file will be removed
- **commit_message** (*str*) – Commit message for the deletion
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabDeleteError – If the server cannot perform the request

get (**args*, ***kwargs*)
Retrieve a single file.

Parameters

- **file_path** (*str*) – Path of the file to retrieve
- **ref** (*str*) – Name of the branch, tag or commit
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the file could not be retrieved

Returns The generated RESTObject

Return type object

raw (**args*, ***kwargs*)
Return the content of a file for a commit.

Parameters

- **ref** (*str*) – ID of the commit
- **filepath** (*str*) – Path of the file to return
- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment
- **action** (*callable*) – Callable responsible of dealing with chunk of data
- **chunk_size** (*int*) – Size of each chunk
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the file could not be retrieved

Returns The file content

Return type str

update (**args*, ***kwargs*)
Update an object on the server.

Parameters

- **id** – ID of the object to update (can be None if not required)
- **new_data** – the update data for the object
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The new object data (*not* a RESTObject)

Return type dict

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabUpdateError – If the server cannot perform the request

class gitlab.v4.objects.**ProjectFork**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectForkManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CreateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectHook**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectHookManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectIssue**(*manager*, *attrs*)
Bases: *gitlab.mixins.SubscribableMixin*, *gitlab.mixins.TodoMixin*, *gitlab.mixins.TimeTrackingMixin*, *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

move(**args*, ***kwargs*)
Move the issue to another project.

Parameters

- **to_project_id** (*int*) – ID of the target project
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabUpdateError – If the issue could not be moved

user_agent_detail(**args*, ***kwargs*)
Get user agent detail.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the detail could not be retrieved

class gitlab.v4.objects.**ProjectIssueAwardEmoji**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectIssueAwardEmojiManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectIssueManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectIssueNote**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectIssueNoteAwardEmoji**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectIssueNoteAwardEmojiManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectIssueNoteManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectJob**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

artifacts(**args*, ***kwargs*)
Get the job artifacts.

Parameters

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment
- **action** (*callable*) – Callable responsible of dealing with chunk of data
- **chunk_size** (*int*) – Size of each chunk
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the artifacts could not be retrieved

Returns The artifacts if *streamed* is False, None otherwise.

Return type str

cancel(**args*, ***kwargs*)
Cancel the job.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabJobCancelError – If the job could not be canceled

erase(**args*, ***kwargs*)
Erase the job (remove job artifacts and trace).

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabJobEraseError – If the job could not be erased

keep_artifacts(**args*, ***kwargs*)
Prevent artifacts from being deleted when expiration is set.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the request could not be performed

play(**args*, ***kwargs*)
Trigger a job explicitly.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabJobPlayError – If the job could not be triggered

retry(**args*, ***kwargs*)
Retry the job.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabJobRetryError – If the job could not be retried

trace(**args*, ***kwargs*)
Get the job trace.

Parameters

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment
- **action** (*callable*) – Callable responsible of dealing with chunk of data
- **chunk_size** (*int*) – Size of each chunk
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the artifacts could not be retrieved

Returns The trace

Return type str

class gitlab.v4.objects.**ProjectJobManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectKey**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectKeyManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

enable(**args*, ***kwargs*)
Enable a deploy key for a project.

Parameters

- **key_id** (*int*) – The ID of the key to enable

- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabProjectDeployKeyError – If the key could not be enabled

class gitlab.v4.objects.**ProjectLabel**(*manager*, *attrs*)
Bases: *gitlab.mixins.SubscribableMixin*, *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

save(**args*, ***kwargs*)
Saves the changes made to the object to the server.

The object is updated to match what the server returns.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct.
- GitlabUpdateError – If the server cannot perform the request.

class gitlab.v4.objects.**ProjectLabelManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

delete(**args*, ***kwargs*)
Delete a Label on the server.

Parameters

- **name** – The name of the label
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct.
- GitlabDeleteError – If the server cannot perform the request.

class gitlab.v4.objects.**ProjectManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectMember**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectMemberManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectMergeRequest**(*manager*, *attrs*)
Bases: *gitlab.mixins.SubscribableMixin*, *gitlab.mixins.TodoMixin*, *gitlab.mixins.TimeTrackingMixin*, *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

cancel_merge_when_pipeline_succeeds(**args*, ***kwargs*)
Cancel merge when the pipeline succeeds.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct

- GitlabMROnBuildSuccessError – If the server could not handle the request

changes (**args*, ***kwargs*)
List the merge request changes.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the list could not be retrieved

Returns List of changes

Return type *RESTObjectList*

closes_issues (**args*, ***kwargs*)
List issues that will close on merge."

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the list could not be retrieved

Returns List of issues

Return type *RESTObjectList*

commits (**args*, ***kwargs*)
List the merge request commits.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the list could not be retrieved

Returns The list of commits

Return type *RESTObjectList*

merge(**args*, ***kwargs*)
Accept the merge request.

Parameters

- **merge_commit_message** (*bool*) – Commit message
- **should_remove_source_branch** (*bool*) – If True, removes the source branch
- **merge_when_pipeline_succeeds** (*bool*) – Wait for the build to succeed, then merge
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabMRClosedError – If the merge failed

participants(**args*, ***kwargs*)
List the merge request participants.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the list could not be retrieved

Returns The list of participants

Return type *RESTObjectList*

class gitlab.v4.objects.**ProjectMergeRequestAwardEmoji**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectMergeRequestAwardEmojiManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectMergeRequestDiff**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectMergeRequestDiffManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectMergeRequestManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectMergeRequestNote**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectMergeRequestNoteAwardEmoji**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectMergeRequestNoteAwardEmojiManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectMergeRequestNoteManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectMilestone**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

issues(**args*, ***kwargs*)
List issues related to this milestone.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the list could not be retrieved

Returns The list of issues

Return type *RESTObjectList*

merge_requests(**args*, ***kwargs*)
List the merge requests related to this milestone.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the list could not be retrieved

Returns The list of merge requests

Return type *RESTObjectList*

class gitlab.v4.objects.**ProjectMilestoneManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectNote**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectNoteManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectNotificationSettings**(*manager*, *attrs*)
Bases: *gitlab.v4.objects.NotificationSettings*

class gitlab.v4.objects.**ProjectNotificationSettingsManager**(*gl*, *parent=None*)
Bases: *gitlab.v4.objects.NotificationSettingsManager*

class gitlab.v4.objects.**ProjectPagesDomain**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectPagesDomainManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectPipeline**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

cancel(**args*, ***kwargs*)
Cancel the job.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabPipelineCancelError – If the request failed

retry(**args*, ***kwargs*)
Retry the job.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabPipelineRetryError – If the request failed

class gitlab.v4.objects.**ProjectPipelineJob**(*manager*, *attrs*)
Bases: *gitlab.v4.objects.ProjectJob*

class gitlab.v4.objects.**ProjectPipelineJobManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectPipelineJobsManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.ListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectPipelineManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.CreateMixin*, *gitlab.base.RESTManager*

create(*data*, ***kwargs*)
Creates a new object.

Parameters

- **data** (*dict*) – Parameters to send to the server to create the resource
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct

- GitlabCreateError – If the server cannot perform the request

Returns

A new instance of the managed object class build with the data sent by the server

Return type *RESTObject*

class gitlab.v4.objects.**ProjectPipelineSchedule**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

take_ownership(**args*, ***kwargs*)
Update the owner of a pipeline schedule.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabOwnershipError – If the request failed

class gitlab.v4.objects.**ProjectPipelineScheduleManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectPipelineScheduleVariable**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectPipelineScheduleVariableManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CreateMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectProtectedBranch**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectProtectedBranchManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectRunner**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectRunnerManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectService**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectServiceManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

available(**args*, ***kwargs*)
List the services known by python-gitlab.

Returns The list of service code names.

Return type *list* (str)

get(*id*, ***kwargs*)
Retrieve a single object.

Parameters

- **id** (*int or str*) – ID of the object to retrieve
- **lazy** (*bool*) – If True, don't request the server, but create a shallow object giving access to the managers. This is useful if you want to avoid useless calls to the API.
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The generated RESTObject.

Return type object

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server cannot perform the request

update(*id=None*, *new_data={}*, ***kwargs*)
Update an object on the server.

Parameters

- **id** – ID of the object to update (can be None if not required)
- **new_data** – the update data for the object
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The new object data (*not* a RESTObject)

Return type dict

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabUpdateError – If the server cannot perform the request

class gitlab.v4.objects.**ProjectSnippet**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

content(**args*, ***kwargs*)
Return the content of a snippet.

Parameters

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data
- **chunk_size** (*int*) – Size of each chunk
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the content could not be retrieved

Returns The snippet content

Return type str

class gitlab.v4.objects.**ProjectSnippetAwardEmoji**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectSnippetAwardEmojiManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectSnippetManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectSnippetNote**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectSnippetNoteAwardEmoji**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectSnippetNoteAwardEmojiManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectSnippetNoteManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectTag**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

set_release_description(**args*, ***kwargs*)
Set the release notes on the tag.

If the release doesn't exist yet, it will be created. If it already exists, its description will be updated.

Parameters

- **description** (*str*) – Description of the release.
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the server fails to create the release
- GitlabUpdateError – If the server fails to update the release

class gitlab.v4.objects.**ProjectTagManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectTrigger**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

take_ownership(**args*, ***kwargs*)
Update the owner of a trigger.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabOwnershipError – If the request failed

class gitlab.v4.objects.**ProjectTriggerManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectUser**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectUserManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.ListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectVariable**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectVariableManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**ProjectWiki**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**ProjectWikiManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**Runner**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**RunnerManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

all(**args*, ***kwargs*)
List all the runners.

Parameters

- **scope** (*str*) – The scope of runners to show, one of: specific, shared, active, paused, online
- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the server failed to perform the request

Returns a list of runners matching the scope.

Return type *list*(*Runner*)

class gitlab.v4.objects.**SidekiqManager**(*gl*, *parent=None*)
Bases: *gitlab.base.RESTManager*

Manager for the Sidekiq methods.

This manager doesn't actually manage objects but provides helper fonction for the sidekiq metrics API.

compound_metrics(**args*, ***kwargs*)
Return all available metrics and statistics.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- `GitlabAuthenticationError` – If authentication is not correct
- `GitlabGetError` – If the information couldn't be retrieved

Returns All available Sidekiq metrics and statistics

Return type dict

job_stats(**args*, ***kwargs*)

Return statistics about the jobs performed.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- `GitlabAuthenticationError` – If authentication is not correct
- `GitlabGetError` – If the information couldn't be retrieved

Returns Statistics about the Sidekiq jobs performed

Return type dict

process_metrics(**args*, ***kwargs*)

Return the registred sidekiq workers.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- `GitlabAuthenticationError` – If authentication is not correct
- `GitlabGetError` – If the information couldn't be retrieved

Returns Information about the register Sidekiq worker

Return type dict

queue_metrics(**args*, ***kwargs*)

Return the registred queues information.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- `GitlabAuthenticationError` – If authentication is not correct
- `GitlabGetError` – If the information couldn't be retrieved

Returns Information about the Sidekiq queues

Return type dict

class gitlab.v4.objects.**Snippet**(*manager*, *attrs*)

Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

content(**args*, ***kwargs*)

Return the content of a snippet.

Parameters

- **streamed** (*bool*) – If True the data will be processed by chunks of *chunk_size* and each chunk is passed to *action* for treatment.
- **action** (*callable*) – Callable responsible of dealing with chunk of data

- **chunk_size** (*int*) – Size of each chunk
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the content could not be retrieved

Returns The snippet content

Return type str

class gitlab.v4.objects.**SnippetManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

public(**args*, ***kwargs*)
List all the public snippets.

Parameters

- **all** (*bool*) – If True the returned object will be a list
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises GitlabListError – If the list could not be retrieved

Returns A generator for the snippets list

Return type *RESTObjectList*

class gitlab.v4.objects.**Todo**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

mark_as_done(**args*, ***kwargs*)
Mark the todo as done.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTodoError – If the server failed to perform the request

class gitlab.v4.objects.**TodoManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

mark_all_as_done(**args*, ***kwargs*)
Mark all the todos as done.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTodoError – If the server failed to perform the request

Returns The number of todos maked done

Return type int

class gitlab.v4.objects.**User**(*manager*, *attrs*)
Bases: *gitlab.mixins.SaveMixin*, *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

block(**args*, ***kwargs*)
Block the user.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabBlockError – If the user could not be blocked

Returns Whether the user status has been changed

Return type bool

unblock(**args*, ***kwargs*)
Unblock the user.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabUnblockError – If the user could not be unblocked

Returns Whether the user status has been changed

Return type bool

class gitlab.v4.objects.**UserActivities**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**UserActivitiesManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.ListMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**UserCustomAttribute**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**UserCustomAttributeManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.SetMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**UserEmail**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**UserEmailManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**UserEvent**(*manager*, *attrs*)
Bases: *gitlab.v4.objects.Event*

class gitlab.v4.objects.**UserEventManager**(*gl*, *parent=None*)
Bases: *gitlab.v4.objects.EventManager*

class gitlab.v4.objects.**UserGPGKey**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**UserGPGKeyManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.RetrieveMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**UserImpersonationToken**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**UserImpersonationTokenManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.NoUpdateMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**UserKey**(*manager*, *attrs*)
Bases: *gitlab.mixins.ObjectDeleteMixin*, *gitlab.base.RESTObject*

class gitlab.v4.objects.**UserKeyManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.GetFromListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**UserManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.CRUDMixin*, *gitlab.base.RESTManager*

class gitlab.v4.objects.**UserProject**(*manager*, *attrs*)
Bases: *gitlab.base.RESTObject*

class gitlab.v4.objects.**UserProjectManager**(*gl*, *parent=None*)
Bases: *gitlab.mixins.ListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.base.RESTManager*

list(***kwargs*)
Retrieve a list of objects.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The list of objects, or a generator if *as_list* is False

Return type *list*

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabListError – If the server cannot perform the request

Module contents

6.2 Submodules

6.3 gitlab.base module

class gitlab.base.**BaseManager**(*gl*, *parent=None*, *args=[]*)
Bases: object

Base manager class for API operations.

Managers provide method to manage GitLab API objects, such as retrieval, listing, creation.

Inherited class must define the obj_cls attribute.

obj_cls
class – class of objects wrapped by this manager.

create(*data*, ***kwargs*)
Create a new object of class *obj_cls*.

Parameters

- **data** (*dict*) – The parameters to send to the GitLab server to create the object. Required and optional arguments are defined in the *requiredCreateAttrs* and *optionalCreateAttrs* of the *obj_cls* class.
- ****kwargs** – Additional arguments to send to GitLab.

Returns A newly create *obj_cls* object.

Return type object

Raises

- NotImplementedError – If objects cannot be created.
- GitlabCreateError – If the server fails to perform the request.

delete(*id*, ***kwargs*)
Delete a GitLab object.

Parameters **id** – ID of the object to delete.

Raises

- NotImplementedError – If objects cannot be deleted.
- GitlabDeleteError – If the server fails to perform the request.

get(*id=None*, ***kwargs*)
Get a GitLab object.

Parameters

- **id** – ID of the object to retrieve.
- ****kwargs** – Additional arguments to send to GitLab.

Returns An object of class *obj_cls*.

Return type object

Raises

- NotImplementedError – If objects cannot be retrieved.
- GitlabGetError – If the server fails to perform the request.

list(***kwargs*)
Get a list of GitLab objects.

Parameters ****kwargs** – Additional arguments to send to GitLab.

Returns A list of *obj_cls* objects.

Return type *list*[object]

Raises

- NotImplementedError – If objects cannot be listed.
- GitlabListError – If the server fails to perform the request.

obj_cls = None

class gitlab.base.**GitlabObject**(*gl*, *data=None*, ***kwargs*)
Bases: object

Base class for all classes that interface with GitLab.

save(***kwargs*)
Send the modified object to the GitLab server. The following attributes are sent:

Available keys for kwargs are:

- sudo (string or int): run the request as another user (requires admin permissions)

as_dict()
Dump the object as a dict.

canCreate = True
Tells if GitLab-api allows creation of new objects.

canDelete = True
Tells if GitLab-api allows deleting object.

canGet = True
Tells if GitLab-api allows retrieving single objects.

canList = True
Tells if GitLab-api allows listing of objects.

canUpdate = True
Tells if GitLab-api allows updating object.

classmethod create(*gl*, *data*, ***kwargs*)
Create an object.

Parameters

- **gl** (gitlab.Gitlab) – Gitlab object referencing the GitLab server.
- **data** (*dict*) – The data used to define the object.

Returns The new object.

Return type object

Raises

- NotImplementedError – If objects can't be created.
- GitlabCreateError – If the server cannot perform the request.

delete(***kwargs*)

display(*pretty*)

classmethod get(*gl*, *id*, ***kwargs*)
Retrieve a single object.

Parameters

- **gl** (gitlab.Gitlab) – Gitlab object referencing the GitLab server.
- **id** (*int or str*) – ID of the object to retrieve.

Returns The found GitLab object.

Return type object

Raises

- `NotImplementedError` – If objects can't be retrieved.
- `GitlabGetError` – If the server cannot perform the request.

getRequiresId = True
Whether the object ID is required in the GET url.

gitlab = None
(gitlab.Gitlab) – Gitlab connection.

idAttr = 'id'
Name of the identifier of an object.

json()
Dump the object as json.

Returns The json string.

Return type str

classmethod list(*gl*, ***kwargs*)
Retrieve a list of objects from GitLab.

Parameters

- **gl** (*gitlab.Gitlab*) – Gitlab object referencing the GitLab server.
- **per_page** (*int*) – Maximum number of items to return.
- **page** (*int*) – ID of the page to return when using pagination.

Returns A list of objects.

Return type *list*[object]

Raises

- `NotImplementedError` – If objects can't be listed.
- `GitlabListError` – If the server cannot perform the request.

managers = []
List of managers to create.

optionalCreateAttrs = []
Attributes that are optional when creating a new object.

optionalGetAttrs = []
Attributes that are optional when retrieving single object.

optionalListAttrs = []
Attributes that are optional when retrieving list of objects.

optionalUpdateAttrs = []
Attributes that are optional when updating an object.

pretty_print(*depth=0*)
Print the object on the standard output (verbose).

Parameters **depth** (*int*) – Used internaly for recursive call.

requiredCreateAttrs = []
Attributes that are required when creating a new object.

requiredDeleteAttrs = []
Attributes that are required when deleting object.

requiredGetAttrs = []
Attributes that are required when retrieving single object.

requiredListAttrs = []
Attributes that are required when retrieving list of objects.

requiredUpdateAttrs = []
Attributes that are required when updating an object.

requiredUrlAttrs = []
Attributes that are required for constructing url.

save(***kwargs*)

shortPrintAttr = None
Attribute to use as ID when displaying the object.

short_print(*depth=0*)
Print the object on the standard output (verbose).

Parameters **depth** (*int*) – Used internaly for recursive call.

class gitlab.base.**RESTManager**(*gl*, *parent=None*)
Bases: object

Base class for CRUD operations on objects.

Derivated class must define _path and _obj_cls.

_path: Base URL path on which requests will be sent (e.g. '/projects') _obj_cls: The class of objects that will be created

parent_attrs

path

class gitlab.base.**RESTObject**(*manager*, *attrs*)
Bases: object

Represents an object built from server data.

It holds the attributes know from the server, and the updated attributes in another. This allows smart updates, if the object allows it.

You can redefine _id_attr in child classes to specify which attribute must be used as uniq ID. None means that the object can be updated without ID in the url.

attributes

get_id()
Returns the id of the resource.

class gitlab.base.**RESTObjectList**(*manager*, *obj_cls*, *_list*)
Bases: object

Generator object representing a list of RESTObject's.

This generator uses the Gitlab pagination system to fetch new data when required.

Note: you should not instanciate such objects, they are returned by calls to RESTManager.list()

Parameters

- **manager** – Manager to attach to the created objects

- **obj_cls** – Type of objects to create from the json data
- **_list** – A GitlabList object

current_page
The current page number.

next()

next_page
The next page number.

If None, the current page is the last.

per_page
The number of items per page.

prev_page
The next page number.

If None, the current page is the last.

total
The total number of items.

total_pages
The total number of pages.

class gitlab.base.**jsonEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)
Bases: json.encoder.JSONEncoder

default(*obj*)

6.4 gitlab.cli module

gitlab.cli.**cls_to_what**(*cls*)

gitlab.cli.**die**(*msg*, *e=None*)

gitlab.cli.**main**()

gitlab.cli.**register_custom_action**(*cls_names*, *mandatory=()*, *optional=()*)

gitlab.cli.**what_to_cls**(*what*)

6.5 gitlab.config module

exception gitlab.config.**ConfigError**
Bases: exceptions.Exception

class gitlab.config.**GitlabConfigParser**(*gitlab_id=None*, *config_files=None*)
Bases: object

exception gitlab.config.**GitlabDataError**
Bases: *gitlab.config.ConfigError*

exception gitlab.config.**GitlabIDError**
Bases: *gitlab.config.ConfigError*

6.6 gitlab.const module

6.7 gitlab.exceptions module

exception gitlab.exceptions.**GitlabAttachFileError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabAuthenticationError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabError*

exception gitlab.exceptions.**GitlabBlockError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabBuildCancelError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabCancelError*

exception gitlab.exceptions.**GitlabBuildEraseError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabRetryError*

exception gitlab.exceptions.**GitlabBuildPlayError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabRetryError*

exception gitlab.exceptions.**GitlabBuildRetryError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabRetryError*

exception gitlab.exceptions.**GitlabCancelError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabCherryPickError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabConnectionError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabError*

exception gitlab.exceptions.**GitlabCreateError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabDeleteError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: exceptions.Exception

exception gitlab.exceptions.**GitlabGetError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabHousekeepingError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabHttpError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabError*

exception gitlab.exceptions.**GitlabJobCancelError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabCancelError*

exception gitlab.exceptions.**GitlabJobEraseError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabRetryError*

exception gitlab.exceptions.**GitlabJobPlayError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabRetryError*

exception gitlab.exceptions.**GitlabJobRetryError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabRetryError*

exception gitlab.exceptions.**GitlabListError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabMRClosedError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabMRForbiddenError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabMROnBuildSuccessError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabOperationError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabError*

exception gitlab.exceptions.**GitlabOwnershipError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabParsingError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabError*

exception gitlab.exceptions.**GitlabPipelineCancelError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabCancelError*

exception gitlab.exceptions.**GitlabPipelineRetryError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabRetryError*

exception gitlab.exceptions.**GitlabProjectDeployKeyError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabProtectError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabRetryError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabSetError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabSubscribeError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabTimeTrackingError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabTodoError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabTransferProjectError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabUnblockError**(*error_message=''*, *response_code=None*, *response_body=None*)

Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabUnsubscribeError**(*error_message=''*, *response_code=None*, *response_body=None*)
Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabUpdateError**(*error_message=''*, *response_code=None*, *response_body=None*)
Bases: *gitlab.exceptions.GitlabOperationError*

exception gitlab.exceptions.**GitlabUploadError**(*error_message=''*, *response_code=None*, *response_body=None*)
Bases: *gitlab.exceptions.GitlabOperationError*

gitlab.exceptions.**on_http_error**(*error*)
Manage GitlabHttpError exceptions.

This decorator function can be used to catch GitlabHttpError exceptions raise specialized exceptions instead.

Parameters **error** (*Exception*) – The exception type to raise – must inherit from GitlabError

gitlab.exceptions.**raise_error_from_response**(*response*, *error*, *expected_code=200*)
Tries to parse gitlab error message from response and raises error.

Do nothing if the response status is the expected one.

If response status code is 401, raises instead GitlabAuthenticationError.

Parameters

- **response** – requests response object
- **error** – Error-class or dict {return-code => class} of possible error class to raise. Should be inherited from GitLabError

6.8 gitlab.mixins module

class gitlab.mixins.**AccessRequestMixin**
Bases: object

approve(**args*, ***kwargs*)
Approve an access request.

Parameters

- **access_level** (*int*) – The access level for the user
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabUpdateError – If the server fails to perform the request

class gitlab.mixins.**CRUDMixin**
Bases: *gitlab.mixins.GetMixin*, *gitlab.mixins.ListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.UpdateMixin*, *gitlab.mixins.DeleteMixin*

class gitlab.mixins.**CreateMixin**
Bases: object

create(**args*, ***kwargs*)
Create a new object.

Parameters

- **data** (*dict*) – parameters to send to the server to create the resource
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns

a new instance of the managed object class built with the data sent by the server

Return type *RESTObject*

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabCreateError – If the server cannot perform the request

get_create_attrs()

Return the required and optional arguments.

Returns

2 items: list of required arguments and list of optional arguments for creation (in that order)

Return type tuple

class gitlab.mixins.**DeleteMixin**

Bases: object

delete(**args*, ***kwargs*)

Delete an object on the server.

Parameters

- **id** – ID of the object to delete
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabDeleteError – If the server cannot perform the request

class gitlab.mixins.**GetFromListMixin**

Bases: *gitlab.mixins.ListMixin*

get(*id*, ***kwargs*)

Retrieve a single object.

Parameters

- **id** (*int or str*) – ID of the object to retrieve
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The generated RESTObject

Return type object

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server cannot perform the request

class gitlab.mixins.**GetMixin**
Bases: object

get(**args*, ***kwargs*)
Retrieve a single object.

Parameters

- **id** (*int or str*) – ID of the object to retrieve
- **lazy** (*bool*) – If True, don't request the server, but create a shallow object giving access to the managers. This is useful if you want to avoid useless calls to the API.
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The generated RESTObject.

Return type object

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server cannot perform the request

class gitlab.mixins.**GetWithoutIdMixin**
Bases: object

get(**args*, ***kwargs*)
Retrieve a single object.

Parameters ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The generated RESTObject

Return type object

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabGetError – If the server cannot perform the request

class gitlab.mixins.**ListMixin**
Bases: object

list(**args*, ***kwargs*)
Retrieve a list of objects.

Parameters

- **all** (*bool*) – If True, return all the items, without pagination
- **per_page** (*int*) – Number of items to retrieve per request
- **page** (*int*) – ID of the page to return (starts with page 1)
- **as_list** (*bool*) – If set to False and no pagination option is defined, return a generator instead of a list
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The list of objects, or a generator if *as_list* is False

Return type *list*

Raises

- GitlabAuthenticationError – If authentication is not correct

- GitlabListError – If the server cannot perform the request

class gitlab.mixins.**NoUpdateMixin**
Bases: *gitlab.mixins.GetMixin*, *gitlab.mixins.ListMixin*, *gitlab.mixins.CreateMixin*, *gitlab.mixins.DeleteMixin*

class gitlab.mixins.**ObjectDeleteMixin**
Bases: object

Mixin for RESTObject's that can be deleted.

delete(***kwargs*)
Delete the object from the server.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabDeleteError – If the server cannot perform the request

class gitlab.mixins.**RetrieveMixin**
Bases: *gitlab.mixins.ListMixin*, *gitlab.mixins.GetMixin*

class gitlab.mixins.**SaveMixin**
Bases: object

Mixin for RESTObject's that can be updated.

save(***kwargs*)
Save the changes made to the object to the server.

The object is updated to match what the server returns.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raise: GitlabAuthenticationError: If authentication is not correct GitlabUpdateError: If the server cannot perform the request

class gitlab.mixins.**SetMixin**
Bases: object

set(**args*, ***kwargs*)
Create or update the object.

Parameters

- **key** (*str*) – The key of the object to create/update
- **value** (*str*) – The value to set for the object
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabSetError – If an error occured

Returns The created/updated attribute

Return type obj

class gitlab.mixins.**SubscribableMixin**
Bases: object

subscribe(**args*, ***kwargs*)
Subscribe to the object notifications.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabSubscribeError – If the subscription cannot be done

unsubscribe(**args*, ***kwargs*)
Unsubscribe from the object notifications.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabUnsubscribeError – If the unsubscription cannot be done

class gitlab.mixins.**TimeTrackingMixin**
Bases: object

add_spent_time(**args*, ***kwargs*)
Add time spent working on the object.

Parameters

- **duration** (*str*) – Duration in human format (e.g. 3h30)
- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTimeTrackingError – If the time tracking update cannot be done

reset_spent_time(**args*, ***kwargs*)
Resets the time spent working on the object.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTimeTrackingError – If the time tracking update cannot be done

reset_time_estimate(**args*, ***kwargs*)
Resets estimated time for the object to 0 seconds.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTimeTrackingError – If the time tracking update cannot be done

time_estimate(**args*, ***kwargs*)
Set an estimated time of work for the object.

Parameters

- **duration** (*str*) – Duration in human format (e.g. 3h30)

- ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTimeTrackingError – If the time tracking update cannot be done

time_stats(**args*, ***kwargs*)
Get time stats for the object.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTimeTrackingError – If the time tracking update cannot be done

class gitlab.mixins.**TodoMixin**
Bases: object

todo(**args*, ***kwargs*)
Create a todo associated to the object.

Parameters ****kwargs** – Extra options to send to the server (e.g. sudo)

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabTodoError – If the todo cannot be set

class gitlab.mixins.**UpdateMixin**
Bases: object

get_update_attrs()
Return the required and optional arguments.

Returns

2 items: list of required arguments and list of optional arguments for update (in that order)

Return type tuple

update(**args*, ***kwargs*)
Update an object on the server.

Parameters

- **id** – ID of the object to update (can be None if not required)
- **new_data** – the update data for the object
- ****kwargs** – Extra options to send to the Gitlab server (e.g. sudo)

Returns The new object data (*not* a RESTObject)

Return type dict

Raises

- GitlabAuthenticationError – If authentication is not correct
- GitlabUpdateError – If the server cannot perform the request

6.9 gitlab.utils module

gitlab.utils.**response_content**(*response*, *streamed*, *action*, *chunk_size*)

6.10 Module contents

Wrapper for the GitLab API.

class gitlab.**Gitlab**(*url*, *private_token=None*, *oauth_token=None*, *email=None*, *password=None*, *ssl_verify=True*, *http_username=None*, *http_password=None*, *timeout=None*, *api_version='4'*, *session=None*)

Bases: object

Represents a GitLab server connection.

Parameters

- **url** (*str*) – The URL of the GitLab server.
- **private_token** (*str*) – The user private token
- **oauth_token** (*str*) – An oauth token
- **email** (*str*) – The user email or login.
- **password** (*str*) – The user password (associated with email).
- **ssl_verify** (*bool|str*) – Whether SSL certificates should be validated. If the value is a string, it is the path to a CA file used for certificate validation.
- **timeout** (*float*) – Timeout to use for requests to the GitLab server.
- **http_username** (*str*) – Username for HTTP authentication
- **http_password** (*str*) – Password for HTTP authentication
- **api_version** (*str*) – Gitlab API version to use (3 or 4)

api_version

auth()

Performs an authentication.

Uses either the private token, or the email/password pair.

The *user* attribute will hold a *gitlab.objects.CurrentUser* object on success.

create(*obj*, ***kwargs*)

Create an object on the GitLab server.

The object class and attributes define the request to be made on the GitLab server.

Parameters

- **obj** (*object*) – The object to create.
- ****kwargs** – Additional arguments to send to GitLab.

Returns

A json representation of the object as returned by the GitLab server

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabCreateError – If the server fails to perform the request.

delete(*obj*, *id=None*, ***kwargs*)
Delete an object on the GitLab server.

Parameters

- **obj** (*object or id*) – The object, or the class of the object to delete. If it is the class, the id of the object must be specified as the *id* arguments.
- **id** – ID of the object to remove. Required if *obj* is a class.
- ****kwargs** – Additional arguments to send to GitLab.

Returns True if the operation succeeds.

Return type bool

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabDeleteError – If the server fails to perform the request.

email = None
The user email

enable_debug()

static from_config(*gitlab_id=None*, *config_files=None*)
Create a Gitlab connection from configuration files.

Parameters

- **gitlab_id** (*str*) – ID of the configuration section.
- **list[str]** (*config_files*) – List of paths to configuration files.

Returns A Gitlab connection.

Return type (*gitlab.Gitlab*)

Raises *gitlab.config.GitlabDataError* – If the configuration is not correct.

get(*obj_class*, *id=None*, ***kwargs*)
Request a GitLab resources.

Parameters

- **obj_class** (*object*) – The class of resource to request.
- **id** – The object ID.
- ****kwargs** – Additional arguments to send to GitLab.

Returns An object of class *obj_class*.

Return type obj_class

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the server fails to perform the request.

headers = None
Headers that will be used in request to GitLab

http_delete(*path*, ***kwargs*)
Make a PUT request to the Gitlab server.

Parameters

- **path** (*str*) – Path or full URL to query ('/projects' or 'http://whatever/v4/api/projecs')
- ****kwargs** – Extra data to make the query (e.g. sudo, per_page, page)

Returns The requests object.

Raises GitlabHttpError – When the return code is not 2xx

http_get(*path*, *query_data={}*, *streamed=False*, ***kwargs*)
Make a GET request to the Gitlab server.

Parameters

- **path** (*str*) – Path or full URL to query ('/projects' or 'http://whatever/v4/api/projecs')
- **query_data** (*dict*) – Data to send as query parameters
- **streamed** (*bool*) – Whether the data should be streamed
- ****kwargs** – Extra data to make the query (e.g. sudo, per_page, page)

Returns A requests result object is streamed is True or the content type is not json. The parsed json data otherwise.

Raises

- GitlabHttpError – When the return code is not 2xx
- GitlabParsingError – If the json data could not be parsed

http_list(*path*, *query_data={}*, *as_list=None*, ***kwargs*)
Make a GET request to the Gitlab server for list-oriented queries.

Parameters

- **path** (*str*) – Path or full URL to query ('/projects' or 'http://whatever/v4/api/projecs')
- **query_data** (*dict*) – Data to send as query parameters
- ****kwargs** – Extra data to make the query (e.g. sudo, per_page, page, all)

Returns A list of the objects returned by the server. If *as_list* is False and no pagination-related arguments (*page*, *per_page*, *all*) are defined then a GitlabList object (generator) is returned instead. This object will make API calls when needed to fetch the next items from the server.

Return type *list*

Raises

- GitlabHttpError – When the return code is not 2xx
- GitlabParsingError – If the json data could not be parsed

http_post(*path*, *query_data={}*, *post_data={}*, *files=None*, ***kwargs*)
Make a POST request to the Gitlab server.

Parameters

- **path** (*str*) – Path or full URL to query ('/projects' or 'http://whatever/v4/api/projecs')
- **query_data** (*dict*) – Data to send as query parameters
- **post_data** (*dict*) – Data to send in the body (will be converted to json)

- ****kwargs** – Extra data to make the query (e.g. sudo, per_page, page)

Returns The parsed json returned by the server if json is return, else the raw content

Raises

- `GitlabHttpError` – When the return code is not 2xx
- `GitlabParsingError` – If the json data could not be parsed

http_put (*path*, *query_data={}*, *post_data={}*, ***kwargs*)
Make a PUT request to the Gitlab server.

Parameters

- **path** (*str*) – Path or full URL to query ('/projects' or 'http://whatever/v4/api/projecs')
- **query_data** (*dict*) – Data to send as query parameters
- **post_data** (*dict*) – Data to send in the body (will be converted to json)
- ****kwargs** – Extra data to make the query (e.g. sudo, per_page, page)

Returns The parsed json returned by the server.

Raises

- `GitlabHttpError` – When the return code is not 2xx
- `GitlabParsingError` – If the json data could not be parsed

http_request (*verb*, *path*, *query_data={}*, *post_data={}*, *streamed=False*, *files=None*, ***kwargs*)
Make an HTTP request to the Gitlab server.

Parameters

- **verb** (*str*) – The HTTP method to call ('get', 'post', 'put', 'delete')
- **path** (*str*) – Path or full URL to query ('/projects' or 'http://whatever/v4/api/projecs')
- **query_data** (*dict*) – Data to send as query parameters
- **post_data** (*dict*) – Data to send in the body (will be converted to json)
- **streamed** (*bool*) – Whether the data should be streamed
- ****kwargs** – Extra data to make the query (e.g. sudo, per_page, page)

Returns A requests result object.

Raises `GitlabHttpError` – When the return code is not 2xx

list (*obj_class*, ***kwargs*)
Request the listing of GitLab resources.

Parameters

- **obj_class** (*object*) – The class of resource to request.
- ****kwargs** – Additional arguments to send to GitLab.

Returns A list of objects of class *obj_class*.

Return type *list*(obj_class)

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabListError` – If the server fails to perform the request.

password = None
The user password (associated with email)

session = None
Create a session object for requests

ssl_verify = None
Whether SSL certificates should be validated

timeout = None
Timeout to use for requests to gitlab server

update(*obj*, ***kwargs*)
Update an object on the GitLab server.

The object class and attributes define the request to be made on the GitLab server.

Parameters

- **obj** (*object*) – The object to create.
- ****kwargs** – Additional arguments to send to GitLab.

Returns

A json representation of the object as returned by the GitLab server

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabUpdateError – If the server fails to perform the request.

version()
Returns the version and revision of the gitlab server.

Note that self.version and self.revision will be set on the gitlab object.

Returns

The server version and server revision, or ('unknown', 'unknwown') if the server doesn't support this API call (gitlab < 8.13.0)

Return type tuple (str, str)

class gitlab.**GitlabList**(*gl*, *url*, *query_data*, *get_next=True*, ***kwargs*)
Bases: object

Generator representing a list of remote objects.

The object handles the links returned by a query to the API, and will call the API again when needed.

current_page
The current page number.

next()

next_page
The next page number.

If None, the current page is the last.

per_page
The number of items per page.

prev_page
The next page number.

If None, the current page is the last.

total
The total number of items.

total_pages
The total number of pages.

CHAPTER 7

Release notes

This page describes important changes between python-gitlab releases.

7.1 Changes from 1.2 to 1.3

- `gitlab.Gitlab` objects can be used as context managers in a `with` block.

7.2 Changes from 1.1 to 1.2

- python-gitlab now respects the `*_proxy`, `REQUESTS_CA_BUNDLE` and `CURL_CA_BUNDLE` environment variables (#352)
- The following deprecated methods and objects have been removed:
  - gitlab.v3.object `Key` and `KeyManager` objects: use `DeployKey` and `DeployKeyManager` instead
  - gitlab.v3.objects.Project `archive_` and `unarchive_` methods
  - gitlab.Gitlab `credentials_auth`, `token_auth`, `set_url`, `set_token` and `set_credentials` methods. Once a Gitlab object has been created its URL and authentication information cannot be updated: create a new Gitlab object if you need to use new information
- The `todo()` method raises a `GitlabTodoError` exception on error

7.3 Changes from 1.0.2 to 1.1

- The `ProjectUser` class doesn't inherit from `User` anymore, and the `GroupProject` class doesn't inherit from `Project` anymore. The Gitlab API doesn't provide the same set of features for these objects, so python-gitlab objects shouldn't try to workaround that.

You can create `User` or `Project` objects from `ProjectUser` and `GroupProject` objects using the `id` attribute:

```
for gr_project in group.projects.list():
    # lazy object creation avoids a Gitlab API request
    project = gl.projects.get(gr_project.id, lazy=True)
    project.default_branch = 'develop'
    project.save()
```

7.4 Changes from 0.21 to 1.0.0

1.0.0 brings a stable python-gitlab API for the v4 Gitlab API. v3 is still used by default.

v4 is mostly compatible with the v3, but some important changes have been introduced. Make sure to read Switching to GtiLab API v4.

The development focus will be v4 from now on. v3 has been deprecated by GitLab and will disappear from python-gitlab at some point.

7.5 Changes from 0.20 to 0.21

- Initial support for the v4 API (experimental)

  The support for v4 is stable enough to be tested, but some features might be broken. Please report issues to https://github.com/python-gitlab/python-gitlab/issues/

  Be aware that the python-gitlab API for v4 objects might change in the next releases.

  Warning: Consider defining explicitly which API version you want to use in the configuration files or in your `gitlab.Gitlab` instances. The default will change from v3 to v4 soon.

- Several methods have been deprecated in the `gitlab.Gitlab` class:
  - `credentials_auth()` is deprecated and will be removed. Call `auth()`.
  - `token_auth()` is deprecated and will be removed. Call `auth()`.
  - `set_url()` is deprecated, create a new `Gitlab` instance if you need an updated URL.
  - `set_token()` is deprecated, use the `private_token` argument of the `Gitlab` constructor.
  - `set_credentials()` is deprecated, use the `email` and `password` arguments of the `Gitlab` constructor.
- The service listing method (`ProjectServiceManager.list()`) now returns a python list instead of a JSON string.

7.6 Changes from 0.19 to 0.20

- The `projects` attribute of `Group` objects is not a list of `Project` objects anymore. It is a Manager object giving access to `GroupProject` objects. To get the list of projects use:

```
group.projects.list()
```

Documentation: http://python-gitlab.readthedocs.io/en/stable/gl_objects/groups.html#examples

Related issue: https://github.com/python-gitlab/python-gitlab/issues/209

- The `Key` objects are deprecated in favor of the new `DeployKey` objects. They are exactly the same but the name makes more sense.

  Documentation: http://python-gitlab.readthedocs.io/en/stable/gl_objects/deploy_keys.html

  Related issue: https://github.com/python-gitlab/python-gitlab/issues/212

CHAPTER 8

ChangeLog

8.1 Version 1.3.0 - 2018-02-18

- Add support for pipeline schedules and schedule variables
- Clarify information about supported python version
- Add manager for jobs within a pipeline
- Fix wrong tag example
- Update the groups documentation
- Add support for MR participants API
- Add support for getting list of user projects
- Add Gitlab and User events support
- Make trigger_pipeline return the pipeline
- Config: support api_version in the global section
- Gitlab can be used as context manager
- Default to API v4
- Add a simplified example for streamed artifacts
- Add documentation about labels update

8.2 Version 1.2.0 - 2018-01-01

- Add mattermost service support
- Add users custom attributes support
- [doc] Fix project.triggers.create example with v4 API

- Oauth token support
- Remove deprecated objects/methods
- Rework authentication args handling
- Add support for oauth and anonymous auth in config/CLI
- Add support for impersonation tokens API
- Add support for user activities
- Update user docs with gitlab URLs
- [docs] Bad arguments in projetcs file documentation
- Add support for user_agent_detail (issues)
- Add a SetMixin
- Add support for project housekeeping
- Expected HTTP response for subscribe is 201
- Update pagination docs for ProjectCommit
- Add doc to get issue from iid
- Make todo() raise GitlabTodoError on error
- Add support for award emojis
- Update project services docs for v4
- Avoid sending empty update data to issue.save
- [docstrings] Explicitly document pagination arguments
- [docs] Add a note about password auth being removed from GitLab
- Submanagers: allow having undefined parameters
- ProjectFile.create(): don't modify the input data
- Update testing tools for /session removal
- Update groups tests
- Allow per_page to be used with generators
- Add groups listing attributes
- Add support for subgroups listing
- Add supported python versions in setup.py
- Add support for pagesdomains
- Add support for features flags
- Add support for project and group custom variables
- Add support for user/group/project filter by custom attribute
- Respect content of REQUESTS_CA_BUNDLE and *_proxy envvars

8.3 Version 1.1.0 - 2017-11-03

- Fix trigger variables in v4 API
- Make the delete() method handle / in ids
- [docs] update the file upload samples
- Tags release description: support / in tag names
- [docs] improve the labels usage documentation
- Add support for listing project users
- ProjectFileManager.create: handle / in file paths
- Change ProjectUser and GroupProject base class
- [docs] document *get_create_attrs* in the API tutorial
- Document the Gitlab session parameter
- ProjectFileManager: custom update() method
- Project: add support for printing_merge_request_link_enabled attr
- Update the ssl_verify docstring
- Add support for group milestones
- Add support for GPG keys
- Add support for wiki pages
- Update the repository_blob documentation
- Fix the CLI for objects without ID (API v4)
- Add a contributed Dockerfile
- Pagination generators: expose more information
- Module's base objects serialization
- [doc] Add sample code for client-side certificates

8.4 Version 1.0.2 - 2017-09-29

- [docs] remove example usage of submanagers
- Properly handle the labels attribute in ProjectMergeRequest
- ProjectFile: handle / in path for delete() and save()

8.5 Version 1.0.1 - 2017-09-21

- Tags can be retrieved by ID
- Add the server response in GitlabError exceptions
- Add support for project file upload
- Minor typo fix in "Switching to v4" documentation

- Fix password authentication for v4
- Fix the labels attrs on MR and issues
- Exceptions: use a proper error message
- Fix http_get method in get artifacts and job trace
- CommitStatus: *sha* is parent attribute
- Fix a couple listing calls to allow proper pagination
- Add missing doc file

8.6 Version 1.0.0 - 2017-09-08

- Support for API v4. See http://python-gitlab.readthedocs.io/en/master/switching-to-v4.html
- Support SSL verification via internal CA bundle
- Docs: Add link to gitlab docs on obtaining a token
- Added dependency injection support for Session
- Fixed repository_compare examples
- Fix changelog and release notes inclusion in sdist
- Missing expires_at in GroupMembers update
- Add lower-level methods for Gitlab()

8.7 Version 0.21.2 - 2017-06-11

- Install doc: use sudo for system commands
- [v4] Make MR work properly
- Remove extra_attrs argument from _raw_list
- [v4] Make project issues work properly
- Fix urlencode() usage (python 2/3) (#268)
- Fixed spelling mistake (#269)
- Add new event types to ProjectHook

8.8 Version 0.21.1 - 2017-05-25

- Fix the manager name for jobs in the Project class
- Fix the docs

8.9 Version 0.21 - 2017-05-24

- Add time_stats to ProjectMergeRequest
- Update User options for creation and update (#246)
- Add milestone.merge_requests() API
- Fix docs typo (s/correspnding/corresponding/)
- Support milestone start date (#251)
- Add support for priority attribute in labels (#256)
- Add support for nested groups (#257)
- Make GroupProjectManager a subclass of ProjectManager (#255)
- Available services: return a list instead of JSON (#258)
- MR: add support for time tracking features (#248)
- Fixed repository_tree and repository_blob path encoding (#265)
- Add 'search' attribute to projects.list()
- Initial gitlab API v4 support
- Reorganise the code to handle v3 and v4 objects
- Allow 202 as delete return code
- Deprecate parameter related methods in gitlab.Gitlab

8.10 Version 0.20 - 2017-03-25

- Add time tracking support (#222)
- Improve changelog (#229, #230)
- Make sure that manager objects are never overwritten (#209)
- Include chanlog and release notes in docs
- Add DeployKey{,Manager} classes (#212)
- Add support for merge request notes deletion (#227)
- Properly handle extra args when listing with all=True (#233)
- Implement pipeline creation API (#237)
- Fix spent_time methods
- Add 'delete source branch' option when creating MR (#241)
- Provide API wrapper for cherry picking commits (#236)
- Stop listing if recursion limit is hit (#234)

8.11 Version 0.19 - 2017-02-21

- Update project.archive() docs
- Support the scope attribute in runners.list()
- Add support for project runners
- Add support for commit creation
- Fix install doc
- Add builds-email and pipelines-email services
- Deploy keys: rework enable/disable
- Document the dynamic aspect of objects
- Add pipeline_events to ProjectHook attrs
- Add due_date attribute to ProjectIssue
- Handle settings.domain_whitelist, partly
- {Project,Group}Member: support expires_at attribute

8.12 Version 0.18 - 2016-12-27

- Fix JIRA service editing for GitLab 8.14+
- Add jira_issue_transition_id to the JIRA service optional fields
- Added support for Snippets (new API in Gitlab 8.15)
- [docs] update pagination section
- [docs] artifacts example: open file in wb mode
- [CLI] ignore empty arguments
- [CLI] Fix wrong use of arguments
- [docs] Add doc for snippets
- Fix duplicated data in API docs
- Update known attributes for projects
- sudo: always use strings

8.13 Version 0.17 - 2016-12-02

- README: add badges for pypi and RTD
- Fix ProjectBuild.play (raised error on success)
- Pass kwargs to the object factory
- Add .tox to ignore to respect default tox settings
- Convert response list to single data source for iid requests
- Add support for boards API

- Add support for Gitlab.version()
- Add support for broadcast messages API
- Add support for the notification settings API
- Don't overwrite attributes returned by the server
- Fix bug when retrieving changes for merge request
- Feature: enable / disable the deploy key in a project
- Docs: add a note for python 3.5 for file content update
- ProjectHook: support the token attribute
- Rework the API documentation
- Fix docstring for http_{username,password}
- Build managers on demand on GitlabObject's
- API docs: add managers doc in GitlabObject's
- Sphinx ext: factorize the build methods
- Implement __repr__ for gitlab objects
- Add a 'report a bug' link on doc
- Remove deprecated methods
- Implement merge requests diff support
- Make the manager objects creation more dynamic
- Add support for templates API
- Add attr 'created_at' to ProjectIssueNote
- Add attr 'updated_at' to ProjectIssue
- CLI: add support for project all –all
- Add support for triggering a new build
- Rework requests arguments (support latest requests release)
- Fix *should_remove_source_branch*

8.14 Version 0.16 - 2016-10-16

- Add the ability to fork to a specific namespace
- JIRA service - add api_url to optional attributes
- Fix bug: Missing coma concatenates array values
- docs: branch protection notes
- Create a project in a group
- Add only_allow_merge_if_build_succeeds option to project objects
- Add support for –all in CLI
- Fix examples for file modification

- Use the plural merge_requests URL everywhere
- Rework travis and tox setup
- Workaround gitlab setup failure in tests
- Add ProjectBuild.erase()
- Implement ProjectBuild.play()

8.15 Version 0.15.1 - 2016-10-16

- docs: improve the pagination section
- Fix and test pagination
- 'path' is an existing gitlab attr, don't use it as method argument

8.16 Version 0.15 - 2016-08-28

- Add a basic HTTP debug method
- Run more tests in travis
- Fix fork creation documentation
- Add more API examples in docs
- Update the ApplicationSettings attributes
- Implement the todo API
- Add sidekiq metrics support
- Move the constants at the gitlab root level
- Remove methods marked as deprecated 7 months ago
- Refactor the Gitlab class
- Remove _get_list_or_object() and its tests
- Fix canGet attribute (typo)
- Remove unused ProjectTagReleaseManager class
- Add support for project services API
- Add support for project pipelines
- Add support for access requests
- Add support for project deployments

8.17 Version 0.14 - 2016-08-07

- Remove 'next_url' from kwargs before passing it to the cls constructor.
- List projects under group
- Add support for subscribe and unsubscribe in issues

- Project issue: doc and CLI for (un)subscribe
- Added support for HTTP basic authentication
- Add support for build artifacts and trace
- –title is a required argument for ProjectMilestone
- Commit status: add optional context url
- Commit status: optional get attrs
- Add support for commit comments
- Issues: add optional listing parameters
- Issues: add missing optional listing parameters
- Project issue: proper update attributes
- Add support for project-issue move
- Update ProjectLabel attributes
- Milestone: optional listing attrs
- Add support for namespaces
- Add support for label (un)subscribe
- MR: add (un)subscribe support
- Add *note_events* to project hooks attributes
- Add code examples for a bunch of resources
- Implement user emails support
- Project: add VISIBILITY_* constants
- Fix the Project.archive call
- Implement archive/unarchive for a projet
- Update ProjectSnippet attributes
- Fix ProjectMember update
- Implement sharing project with a group
- Implement CLI for project archive/unarchive/share
- Implement runners global API
- Gitlab: add managers for build-related resources
- Implement ProjectBuild.keep_artifacts
- Allow to stream the downloads when appropriate
- Groups can be updated
- Replace Snippet.Content() with a new content() method
- CLI: refactor _die()
- Improve commit statuses and comments
- Add support from listing group issues
- Added a new project attribute to enable the container registry.

- Add a contributing section in README
- Add support for global deploy key listing
- Add support for project environments
- MR: get list of changes and commits
- Fix the listing of some resources
- MR: fix updates
- Handle empty messages from server in exceptions
- MR (un)subscribe: don't fail if state doesn't change
- MR merge(): update the object

8.18 Version 0.13 - 2016-05-16

- Add support for MergeRequest validation
- MR: add support for cancel_merge_when_build_succeeds
- MR: add support for closes_issues
- Add "external" parameter for users
- Add deletion support for issues and MR
- Add missing group creation parameters
- Add a Session instance for all HTTP requests
- Enable updates on ProjectIssueNotes
- Add support for Project raw_blob
- Implement project compare
- Implement project contributors
- Drop the next_url attribute when listing
- Remove unnecessary canUpdate property from ProjectIssuesNote
- Add new optional attributes for projects
- Enable deprecation warnings for gitlab only
- Rework merge requests update
- Rework the Gitlab.delete method
- ProjectFile: file_path is required for deletion
- Rename some methods to better match the API URLs
- Deprecate the file_* methods in favor of the files manager
- Implement star/unstar for projects
- Implement list/get licenses
- Manage optional parameters for list() and get()

8.19 Version 0.12.2 - 2016-03-19

- Add new *ProjectHook* attributes
- Add support for user block/unblock
- Fix GitlabObject creation in _custom_list
- Add support for more CLI subcommands
- Add some unit tests for CLI
- Add a coverage tox env
- Define GitlabObject.as_dict() to dump object as a dict
- Define GitlabObject.__eq__() and __ne__() equivalence methods
- Define UserManager.search() to search for users
- Define UserManager.get_by_username() to get a user by username
- Implement "user search" CLI
- Improve the doc for UserManager
- CLI: implement user get-by-username
- Re-implement _custom_list in the Gitlab class
- Fix the 'invalid syntax' error on Python 3.2
- Gitlab.update(): use the proper attributes if defined

8.20 Version 0.12.1 - 2016-02-03

- Fix a broken upload to pypi

8.21 Version 0.12 - 2016-02-03

- Improve documentation
- Improve unit tests
- Improve test scripts
- Skip BaseManager attributes when encoding to JSON
- Fix the json() method for python 3
- Add Travis CI support
- Add a decode method for ProjectFile
- Make connection exceptions more explicit
- Fix ProjectLabel get and delete
- Implement ProjectMilestone.issues()
- ProjectTag supports deletion
- Implement setting release info on a tag

- Implement project triggers support
- Implement project variables support
- Add support for application settings
- Fix the 'password' requirement for User creation
- Add sudo support
- Fix project update
- Fix Project.tree()
- Add support for project builds

8.22 Version 0.11.1 - 2016-01-17

- Fix discovery of parents object attrs for managers
- Support setting commit status
- Support deletion without getting the object first
- Improve the documentation

8.23 Version 0.11 - 2016-01-09

- functional_tests.sh: support python 2 and 3
- Add a get method for GitlabObject
- CLI: Add the -g short option for –gitlab
- Provide a create method for GitlabObject's
- Rename the _created attribute _from_api
- More unit tests
- CLI: fix error when arguments are missing (python 3)
- Remove deprecated methods
- Implement managers to get access to resources
- Documentation improvements
- Add fork project support
- Deprecate the "old" Gitlab methods
- Add support for groups search

8.24 Version 0.10 - 2015-12-29

- Implement pagination for list() (#63)
- Fix url when fetching a single MergeRequest
- Add support to update MergeRequestNotes

- API: Provide a Gitlab.from_config method
- setup.py: require requests>=1 (#69)
- Fix deletion of object not using 'id' as ID (#68)
- Fix GET/POST for project files
- Make 'confirm' an optional attribute for user creation
- Python 3 compatibility fixes
- Add support for group members update (#73)

8.25 Version 0.9.2 - 2015-07-11

- CLI: fix the update and delete subcommands (#62)

8.26 Version 0.9.1 - 2015-05-15

- Fix the setup.py script

8.27 Version 0.9 - 2015-05-15

- Implement argparse libray for parsing argument on CLI
- Provide unit tests and (a few) functional tests
- Provide PEP8 tests
- Use tox to run the tests
- CLI: provide a –config-file option
- Turn the gitlab module into a proper package
- Allow projects to be updated
- Use more pythonic names for some methods
- **Deprecate some Gitlab object methods:**
  - raw* methods should never have been exposed; replace them with _raw_* methods
  - setCredentials and setToken are replaced with set_credentials and set_token
- Sphinx: don't hardcode the version in conf.py

8.28 Version 0.8 - 2014-10-26

- Better python 2.6 and python 3 support
- Timeout support in HTTP requests
- Gitlab.get() raised GitlabListError instead of GitlabGetError
- Support api-objects which don't have id in api response

- Add ProjectLabel and ProjectFile classes
- Moved url attributes to separate list
- Added list for delete attributes

8.29 Version 0.7 - 2014-08-21

- Fix license classifier in setup.py
- Fix encoding error when printing to redirected output
- Fix encoding error when updating with redirected output
- Add support for UserKey listing and deletion
- Add support for branches creation and deletion
- Support state_event in ProjectMilestone (#30)
- Support namespace/name for project id (#28)
- Fix handling of boolean values (#22)

8.30 Version 0.6 - 2014-01-16

- IDs can be unicode (#15)
- ProjectMember: constructor should not create a User object
- Add support for extra parameters when listing all projects (#12)
- Projects listing: explicitly define arguments for pagination

8.31 Version 0.5 - 2013-12-26

- Add SSH key for user
- Fix comments
- Add support for project events
- Support creation of projects for users
- Project: add methods for create/update/delete files
- Support projects listing: search, all, owned
- System hooks can't be updated
- Project.archive(): download tarball of the project
- Define new optional attributes for user creation
- Provide constants for access permissions in groups

8.32 Version 0.4 - 2013-09-26

- Fix strings encoding (Closes #6)
- Allow to get a project commit (GitLab 6.1)
- ProjectMergeRequest: fix Note() method
- Gitlab 6.1 methods: diff, blob (commit), tree, blob (project)
- Add support for Gitlab 6.1 group members

8.33 Version 0.3 - 2013-08-27

- Use PRIVATE-TOKEN header for passing the auth token
- provide a AUTHORS file
- cli: support ssl_verify config option
- Add ssl_verify option to Gitlab object. Defauls to True
- Correct url for merge requests API.

8.34 Version 0.2 - 2013-08-08

- provide a pip requirements.txt
- drop some debug statements

8.35 Version 0.1 - 2013-07-08

- Initial release

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W