python-gitlab Documentation

Release 0.21.2

Gauvain Pocentek, Mika Mäenpää

Aug 11, 2017

Contents

Contents:

CHAPTER 1

Installation

python-gitlab is compatible with python 2 and 3.

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.

Configuration

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.

Content

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

```
[global]
default = somewhere
ssl_verify = true
timeout = 5
api_version = 3

[somewhere]
url = https://some.whe.re
private_token = vTbFeqJYCY3sibBP7BZM
api_version = 4

[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` or `False`	Verify the SSL certificate. Set to `False` if your SSL certificate is auto-signed.
`timeout`	Integer	Number of seconds to wait for an answer before failing.

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

Table 2.2: GitLab server options

Option	Description
`url`	URL for the GitLab server
`private_token`	Your user token. Login/password is not supported.
`api_version`	API version to use (`3` or `4`), defaults to `3`
`http_username`	Username for optional HTTP authentication
`http_password`	Password for optional HTTP authentication

CLI

Objects and actions

The `gitlab` command expects two mandatory arguments. This 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
```

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.

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

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

Example:

```
$ gitlab -v -g elsewhere -c /tmp/gl.cfg project list
```

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 or by username:

```
$ gitlab user get --id 3
$ gitlab user get-by-username --query jdoe
```

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

The `gitlab` package provides 3 base types:

- `gitlab.Gitlab` is the primary class, handling the HTTP requests. It holds the GitLab URL and authentication information.
- `gitlab.GitlabObject` is the base class for all the GitLab objects. These objects provide an abstraction for GitLab resources (projects, groups, and so on).
- `gitlab.BaseManager` is the base class for objects managers, providing the API to manipulate the resources and their attributes.

`gitlab.Gitlab` class

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

```
import gitlab

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

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

# 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.

GitLab v4 support

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

```
import gitlab

gl = gitlab.Gitlab('http://10.0.0.1', 'JVNSESs8EwWRx5yDxM5q', api_version=4)
```

Warning: The v4 support is experimental.

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. Resources are represented as gitlab.GitlabObject-derived objects.

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)
```

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:

```
project = gl.projects.get(1)
print(vars(project))
# or
print(project.__dict__)
```

Some gitlab.GitlabObject classes also provide managers to access related GitLab resources:

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

Gitlab Objects

You can update or delete an object when it exists as a GitlabObject object:

```
# 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 `GitlabObject`-derived classes provide additional methods, allowing more actions on the GitLab resources. For example:

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

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.

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)
```

Note: 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.

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')
```

CHAPTER 4

API examples

Access requests

Use `ProjectAccessRequest` and `GroupAccessRequest` objects to manipulate access requests for projects and groups. The *gitlab.Gitlab.project_accessrequests*, *gitlab.Gitlab.group_accessrequests*, `Project.accessrequests` and `Group.accessrequests` manager objects provide helper functions.

Examples

List access requests from projects and groups:

```
p_ars = gl.project_accessrequests.list(project_id=1)
g_ars = gl.group_accessrequests.list(group_id=1)
# or
p_ars = project.accessrequests.list()
g_ars = group.accessrequests.list()
```

Get a single request:

```
p_ar = gl.project_accessrequests.get(user_id, project_id=1)
g_ar = gl.group_accessrequests.get(user_id, group_id=1)
# or
p_ar = project.accessrequests.get(user_id)
g_ar = group.accessrequests.get(user_id)
```

Create an access request:

```
p_ar = gl.project_accessrequests.create({}, project_id=1)
g_ar = gl.group_accessrequests.create({}, group_id=1)
# or
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:

```
gl.project_accessrequests.delete(user_id, project_id=1)
gl.group_accessrequests.delete(user_id, group_id=1)
# or
project.accessrequests.delete(user_id)
group.accessrequests.delete(user_id)
# or
ar.delete()
```

Branches

Use `ProjectBranch` objects to manipulate repository branches.

To create `ProjectBranch` objects use the *gitlab.Gitlab.project_branches* or `Project.branches` managers.

Examples

Get the list of branches for a repository:

```
branches = gl.project_branches.list(project_id=1)
# or
branches = project.branches.list()
```

Get a single repository branch:

```
branch = gl.project_branches.get(project_id=1, id='master')
# or
branch = project.branches.get('master')
```

Create a repository branch:

```
branch = gl.project_branches.create({'branch_name': 'feature1',
                                     'ref': 'master'},
                                    project_id=1)
# or
branch = project.branches.create({'branch_name': 'feature1',
                                  'ref': 'master'})
```

Delete a repository branch:

```
gl.project_branches.delete(project_id=1, id='feature1')
# or
project.branches.delete('feature1')
# or
branch.delete()
```

Protect/unprotect a repository branch:

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

Note: By default, developers will not be able to push or merge into protected branches. This can be changed by passing `developers_can_push` or `developers_can_merge` like so: `branch.protect(developers_can_push=False, developers_can_merge=True)`

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.

- Object class: `gitlab.objects.BroadcastMessage`
- Manager object: *`gitlab.Gitlab.broadcastmessages`*

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()
```

Builds

Build triggers

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

- Object class: `ProjectTrigger`
- Manager objects: *gitlab.Gitlab.project_triggers*, `Project.triggers`

Examples

List triggers:

```
triggers = gl.project_triggers.list(project_id=1)
# or
triggers = project.triggers.list()
```

Get a trigger:

```
trigger = gl.project_triggers.get(trigger_token, project_id=1)
# or
trigger = project.triggers.get(trigger_token)
```

Create a trigger:

```
trigger = gl.project_triggers.create({}, project_id=1)
# or
trigger = project.triggers.create({})
```

Remove a trigger:

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

Build variables

You can associate variables to builds to modify the build script behavior.

- Object class: `ProjectVariable`
- Manager objects: *gitlab.Gitlab.project_variables*, `gitlab.objects.Project.variables`

Examples

List variables:

```
variables = gl.project_variables.list(project_id=1)
# or
variables = project.variables.list()
```

Get a variable:

```
var = gl.project_variables.get(var_key, project_id=1)
# or
var = project.variables.get(var_key)
```

Create a variable:

```
var = gl.project_variables.create({'key': 'key1', 'value': 'value1'},
                                  project_id=1)
# or
var = project.variables.create({'key': 'key1', 'value': 'value1'})
```

Update a variable value:

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

Remove a variable:

```
gl.project_variables.delete(var_key)
# or
project.variables.delete()
# or
var.delete()
```

Builds

Builds are associated to projects and commits. They provide information on the build that have been run, and methods to manipulate those builds.

- Object class: `ProjectBuild`
- Manager objects: *gitlab.Gitlab.project_builds*, `gitlab.objects.Project.builds`

Examples

Build are usually automatically triggered, but you can explicitly trigger a new build:

Trigger a new build on a project:

```
p = gl.projects.get(project_id)
p.trigger_build('master', trigger_token,
                {'extra_var1': 'foo', 'extra_var2': 'bar'})
```

List builds for the project:

```
builds = gl.project_builds.list(project_id=1)
# or
builds = project.builds.list()
```

To list builds for a specific commit, create a `ProjectCommit` object and use its `builds` method:

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

Get a build:

```
build = gl.project_builds.get(build_id, project_id=1)
# or
project.builds.get(build_id)
```

Get a build artifacts:

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

Mark a build artifact as kept when expiration is set:

```
build.keep_artifacts()
```

Get a build trace:

```
build.trace()
```

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

Cancel/retry a build:

```
build.cancel()
build.retry()
```

Play (trigger) a build:

```
build.play()
```

Erase a build (artifacts and trace):

```
build.erase()
```

Commits

Commits

- Object class: `ProjectCommit`
- Manager objects: *gitlab.Gitlab.project_commits*, `gitlab.objects.Project.commits`

Examples

List the commits for a project:

```
commits = gl.project_commits.list(project_id=1)
# or
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',
    'commit_message': 'blah blah blah',
    'actions': [
        {
            'action': 'create',
            'file_path': 'blah',
            'content': 'blah'
        }
    ]
}

commit = gl.project_commits.create(data, project_id=1)
# or
commit = project.commits.create(data)
```

Get a commit detail:

```
commit = gl.project_commits.get('e3d5a71b', project_id=1)
# or
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')
```

Commit comments

- Object class: `ProjectCommiComment`
- Manager objects: *gitlab.Gitlab.project_commit_comments*, `gitlab.objects.Project.commit_comments`, `gitlab.objects.ProjectCommit.comments`

Examples

Get the comments for a commit:

```
comments = gl.project_commit_comments.list(project_id=1, commit_id='master')
# or
comments = project.commit_comments.list(commit_id='a5fe4c8')
# or
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'})
```

Commit status

- Object class: `ProjectCommitStatus`
- Manager objects: *gitlab.Gitlab.project_commit_statuses*, `gitlab.objects.Project.commit_statuses`, `gitlab.objects.ProjectCommit.statuses`

Examples

Get the statuses for a commit:

```
statuses = gl.project_commit_statuses.list(project_id=1, commit_id='master')
# or
statuses = project.commit_statuses.list(commit_id='a5fe4c8')
# or
statuses = commit.statuses.list()
```

Change the status of a commit:

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

Deploy keys

Deploy keys

Deploy keys allow read-only access to multiple projects with a single SSH key.

- Object class: `DeployKey`
- Manager object: *gitlab.Gitlab.deploykeys*

Examples

List the deploy keys:

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

Get a single deploy key:

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

Deploy keys for projects

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

- Object class: `ProjectKey`
- Manager objects: *gitlab.Gitlab.project_keys* and `Project.keys`

Examples

List keys for a project:

```
keys = gl.project_keys.list(project_id=1)
# or
keys = project.keys.list()
```

Get a single deploy key:

```
key = gl.project_keys.get(key_id, project_id=1)
# or
key = project.keys.get(key_id)
```

Create a deploy key for a project:

```
key = gl.project_keys.create({'title': 'jenkins key',
                              'key': open('/home/me/.ssh/id_rsa.pub').read()},
                             project_id=1)
# or
key = project.keys.create({'title': 'jenkins key',
                           'key': open('/home/me/.ssh/id_rsa.pub').read()})
```

Delete a deploy key for a project:

```
key = gl.project_keys.delete(key_id, project_id=1)
# or
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.keys.disable(key_id)
```

Deployments

Use `ProjectDeployment` objects to manipulate project deployments. The *gitlab.Gitlab.project_deployments*, and `Project.deployments` manager objects provide helper functions.

Examples

List deployments for a project:

```
deployments = gl.project_deployments.list(project_id=1)
# or
deployments = project.deployments.list()
```

Get a single deployment:

```
deployment = gl.project_deployments.get(deployment_id, project_id=1)
# or
deployment = project.deployments.get(deployment_id)
```

Environments

Use `ProjectEnvironment` objects to manipulate environments for projects. The *gitlab.Gitlab.project_environments* and `Project.environments` manager objects provide helper functions.

Examples

List environments for a project:

```
environments = gl.project_environments.list(project_id=1)
# or
environments = project.environments.list()
```

Get a single environment:

```
environment = gl.project_environments.get(environment_id, project_id=1)
# or
environment = project.environments.get(environment_id)
```

Create an environment for a project:

```
environment = gl.project_environments.create({'name': 'production'},
                                             project_id=1)
# or
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 = gl.project_environments.delete(environment_id, project_id=1)
# or
environment = project.environments.list(environment_id)
# or
environment.delete()
```

Groups

Groups

Use `Group` objects to manipulate groups. The *gitlab.Gitlab.groups* manager object provides helper functions.

Examples

List the groups:

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

Search groups:

```
groups = gl.groups.search('group')
```

Get a group's detail:

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

List a group's projects:

```
projects = group.projects.list()
# or
projects = gl.group_projects.list(group_id)
```

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()
```

Group members

Use `GroupMember` objects to manipulate groups. The *gitlab.Gitlab.group_members* and `Group.members` manager objects provide helper functions.

The following `Group` attributes define the supported access levels:

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

List group members:

```
members = gl.group_members.list(group_id=1)
# or
members = group.members.list()
```

Get a group member:

```
members = gl.group_members.get(member_id)
# or
members = group.members.get(member_id)
```

Add a member to the group:

```
member = gl.group_members.create({'user_id': user_id,
                                  'access_level': gitlab.GUEST_ACCESS},
                                 group_id=1)
# or
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:

```
gl.group_members.delete(member_id, group_id=1)
# or
group.members.delete(member_id)
# or
member.delete()
```

Issues

Reported issues

Use `Issues` objects to manipulate issues the authenticated user reported. The *gitlab.Gitlab.issues* manager object provides helper functions.

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'])
```

Group issues

Use `GroupIssue` objects to manipulate issues. The *gitlab.Gitlab.project_issues* and `Group.issues` manager objects provide helper functions.

Examples

List the group issues:

```
issues = gl.group_issues.list(group_id=1)
# or
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')
```

Project issues

Use `ProjectIssue` objects to manipulate issues. The *gitlab.Gitlab.project_issues* and `Project.issues` manager objects provide helper functions.

Examples

List the project issues:

```
issues = gl.project_issues.list(project_id=1)
# or
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 = gl.project_issues.get(issue_id, project_id=1)
# or
issue = project.issues.get(issue_id)
```

Create a new issue:

```
issue = gl.project_issues.create({'title': 'I have a bug',
                                  'description': 'Something useful here.'},
                                 project_id=1)
# or
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:

```
gl.project_issues.delete(issue_id, project_id=1)
# or
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()
```

Labels

Use `ProjectLabel` objects to manipulate labels for projects. The *gitlab.Gitlab.project_labels* and `Project.labels` manager objects provide helper functions.

Examples

List labels for a project:

```
labels = gl.project_labels.list(project_id=1)
# or
labels = project.labels.list()
```

Get a single label:

```
label = gl.project_labels.get(label_name, project_id=1)
# or
label = project.labels.get(label_name)
```

Create a label for a project:

```
label = gl.project_labels.create({'name': 'foo', 'color': '#8899aa'},
                                 project_id=1)
# or
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:

```
gl.project_labels.delete(label_id, project_id=1)
# or
project.labels.delete(label_id)
# or
label.delete()
```

Notification settings

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

- `NOTIFICATION_LEVEL_DISABLED`
- `NOTIFICATION_LEVEL_PARTICIPATING`
- `NOTIFICATION_LEVEL_WATCH`
- `NOTIFICATION_LEVEL_GLOBAL`
- `NOTIFICATION_LEVEL_MENTION`
- `NOTIFICATION_LEVEL_CUSTOM`

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

- Object classes: `gitlab.objects.NotificationSettings` (global), `gitlab.objects.GroupNotificationSettings` (groups) and `gitlab.objects.ProjectNotificationSettings` (projects)
- Manager objects: *`gitlab.Gitlab.notificationsettings`* (global), `gitlab.objects.Group.notificationsettings` (groups) and `gitlab.objects.Project.notificationsettings` (projects)

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()
```

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.

- Object class: `ProjectMergeRequest`
- Manager objects: *gitlab.Gitlab.project_mergerequests*, `Project.mergerequests`

Examples

List MRs for a project:

```
mrs = gl.project_mergerequests.list(project_id=1)
# or
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
- `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 = gl.project_mergerequests.list(project_id=1)
# or
mrs = project.mergerequests.list()
```

Get a single MR:

```
mr = gl.project_mergerequests.get(mr_id, project_id=1)
# or
mr = project.mergerequests.get(mr_id)
```

Create a MR:

```
mr = gl.project_mergerequests.create({'source_branch': 'cool_feature',
                                      'target_branch': 'master',
                                      'title': 'merge cool feature'},
                                     project_id=1)
# or
mr = project.mergerequests.create({'source_branch': 'cool_feature',
                                   'target_branch': 'master',
                                   'title': 'merge cool feature'})
```

Update a MR:

```
mr.description = 'New description'
mr.save()
```

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

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

Delete a MR:

```
gl.project_mergerequests.delete(mr_id, project_id=1)
# or
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()
```

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)
```

Namespaces

Use `Namespace` objects to manipulate namespaces. The *gitlab.Gitlab.namespaces* manager objects provides helper functions.

Examples

List namespaces:

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

Search namespaces:

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

Milestones

Use `ProjectMilestone` objects to manipulate milestones. The *gitlab.Gitlab.project_milestones* and `Project.milestones` manager objects provide helper functions.

Examples

List the milestones for a project:

```
milestones = gl.project_milestones.list(project_id=1)
# or
milestones = project.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`

```
milestones = gl.project_milestones.list(project_id=1, state='closed')
# or
milestones = project.milestones.list(state='closed')
```

Get a single milestone:

```
milestone = gl.project_milestones.get(milestone_id, project_id=1)
# or
milestone = project.milestones.get(milestone_id)
```

Create a milestone:

```
milestone = gl.project_milestones.create({'title': '1.0'}, project_id=1)
# or
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()
```

Projects

Use `Project` objects to manipulate projects. The *`gitlab.Gitlab.projects`* manager objects provides helper functions.

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 = gl.user_projects.create({'name': 'project',
                                        'user_id': alice.id})
```

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 = gl.project_forks.create({}, project_id=1)
# or
fork = project.forks.create({})

# fork to a specific namespace
fork = gl.project_forks.create({'namespace': 'myteam'}, project_id=1)
```

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.

Repository

The following examples show how you can manipulate the project code repository.

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 of a file for a commit:

```
file_content = p.repository_blob('master', 'README.rst')
```

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 i in commit:
    print(result.commits)

# get the diffs
for file_diff in commit.diffs:
    print(file_diff)
```

Get a list of contributors for the repository:

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

Files

The following examples show how you can manipulate the project files.

Get a file:

```
f = gl.project_files.get(file_path='README.rst', ref='master',
                         project_id=1)
# or
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:

```
f = gl.project_files.create({'file_path': 'testfile',
                             'branch_name': 'master',
                             'content': file_content,
                             'commit_message': 'Create testfile'},
                            project_id=1)
# or
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_name='master', commit_message='Update testfile')

# 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_name='master', commit_message='Update testfile', encoding='base64')
```

Delete a file:

```
gl.project_files.delete({'file_path': 'testfile',
                         'branch_name': 'master',
                         'commit_message': 'Delete testfile'},
                        project_id=1)
# or
project.files.delete({'file_path': 'testfile',
                      'branch_name': 'master',
                      'commit_message': 'Delete testfile'})
# or
f.delete(commit_message='Delete testfile')
```

Tags

Use `ProjectTag` objects to manipulate tags. The *gitlab.Gitlab.project_tags* and `Project.tags` manager objects provide helper functions.

List the project tags:

```
tags = gl.project_tags.list(project_id=1)
# or
tags = project.tags.list()
```

Get a tag:

```
tag = gl.project_tags.list('1.0', project_id=1)
# or
tags = project.tags.list('1.0')
```

Create a tag:

```
tag = gl.project_tags.create({'tag_name': '1.0', 'ref': 'master'},
                             project_id=1)
# or
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:

```
gl.project_tags.delete('1.0', project_id=1)
# or
project.tags.delete('1.0')
# or
tag.delete()
```

Snippets

Use `ProjectSnippet` objects to manipulate snippets. The *gitlab.Gitlab.project_snippets* and `Project.snippets` manager objects provide helper functions.

List the project snippets:

```
snippets = gl.project_snippets.list(project_id=1)
# or
snippets = project.snippets.list()
```

Get a snippet:

```
snippet = gl.project_snippets.list(snippet_id, project_id=1)
# or
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 = gl.project_snippets.create({'title': 'sample 1',
                                      'file_name': 'foo.py',
                                      'code': 'import gitlab',
                                      'visibility_level':
                                      gitlab.VISIBILITY_PRIVATE},
                                     project_id=1)
# or
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:

```
gl.project_snippets.delete(snippet_id, project_id=1)
# or
project.snippets.delete(snippet_id)
# or
snippet.delete()
```

Notes

You can manipulate notes (comments) on the following resources:

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

List the notes for a resource:

```
i_notes = gl.project_issue_notes.list(project_id=1, issue_id=2)
mr_notes = gl.project_mergerequest_notes.list(project_id=1, merge_request_id=2)
s_notes = gl.project_snippet_notes.list(project_id=1, snippet_id=2)
# or
i_notes = issue.notes.list()
mr_notes = mr.notes.list()
s_notes = snippet.notes.list()
```

Get a note for a resource:

```
i_notes = gl.project_issue_notes.get(note_id, project_id=1, issue_id=2)
mr_notes = gl.project_mergerequest_notes.get(note_id, project_id=1,
                                             merge_request_id=2)
s_notes = gl.project_snippet_notes.get(note_id, project_id=1, snippet_id=2)
```

```
# or
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 = gl.project_issue_notes.create({'body': 'note content'},
                                       project_id=1, issue_id=2)
mr_note = gl.project_mergerequest_notes.create({'body': 'note content'}
                                               project_id=1,
                                               merge_request_id=2)
s_note = gl.project_snippet_notes.create({'body': 'note content'},
                                         project_id=1, snippet_id=2)
# or
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()
```

Events

Use `ProjectEvent` objects to manipulate events. The *gitlab.Gitlab.project_events* and `Project.events` manager objects provide helper functions.

List the project events:

```
gl.project_events.list(project_id=1)
# or
project.events.list()
```

Team members

Use `ProjectMember` objects to manipulate projects members. The *gitlab.Gitlab.project_members* and `Project.members` manager objects provide helper functions.

List the project members:

```
members = gl.project_members.list()
# or
members = project.members.list()
```

Search project members matching a query string:

```
members = gl.project_members.list(query='foo')
# or
members = project.members.list(query='bar')
```

Get a single project member:

```
member = gl.project_members.get(1)
# or
member = project.members.get(1)
```

Add a project member:

```
member = gl.project_members.create({'user_id': user.id, 'access_level':
                                    gitlab.DEVELOPER_ACCESS},
                                   project_id=1)
# or
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:

```
gl.project_members.delete(user.id, project_id=1)
# or
project.members.delete(user.id)
# or
member.delete()
```

Share the project with a group:

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

Hooks

Use `ProjectHook` objects to manipulate projects hooks. The *gitlab.Gitlab.project_hooks* and `Project.hooks` manager objects provide helper functions.

List the project hooks:

```
hooks = gl.project_hooks.list(project_id=1)
# or
hooks = project.hooks.list()
```

Get a project hook:

```
hook = gl.project_hooks.get(1, project_id=1)
# or
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:

```
gl.project_hooks.delete(1, project_id=1)
# or
project.hooks.delete(1)
# or
hook.delete()
```

Pipelines

Use `ProjectPipeline` objects to manipulate projects pipelines. The *gitlab.Gitlab.project_pipelines* and `Project.services` manager objects provide helper functions.

List pipelines for a project:

```
pipelines = gl.project_pipelines.list(project_id=1)
# or
pipelines = project.pipelines.list()
```

Get a pipeline for a project:

```
pipeline = gl.project_pipelines.get(pipeline_id, project_id=1)
# or
pipeline = project.pipelines.get(pipeline_id)
```

Retry the failed builds for a pipeline:

```
pipeline.retry()
```

Cancel builds in a pipeline:

```
pipeline.cancel()
```

Create a pipeline for a particular reference:

```
pipeline = gl.project_pipelines.create({'project_id': 1, 'ref': 'master'})
# or
pipeline = project.pipelines.create({'ref': 'master'})
```

Services

Use `ProjectService` objects to manipulate projects services. The *gitlab.Gitlab.project_services* and `Project.services` manager objects provide helper functions.

Get a service:

```
service = gl.project_services.get(service_name='asana', project_id=1)
# or
service = project.services.get(service_name='asana', project_id=1)
# display it's 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()
```

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.

Get the list of existing boards for a project:

```
boards = gl.project_boards.list(project_id=1)
# or
boards = project.boards.list()
```

Get a single board for a project:

```
board = gl.project_boards.get(board_id, project_id=1)
# or
board = project.boards.get(board_id)
```

Boards have lists of issues. Each list is defined by a `ProjectLabel` and a position in the board.

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. Note that getting the label ID is broken at the moment (see https://gitlab.com/gitlab-org/gitlab-ce/issues/23448):

```
# 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 insert 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()
```

Runners

Runners are external process 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.

Global runners (admin)

- Object class: `Runner`
- Manager objects: *`gitlab.Gitlab.runners`*

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()
```

Project runners

- Object class: `ProjectRunner`
- Manager objects: *gitlab.Gitlab.runners*, `gitlab.Gitlab.Project.runners`

Examples

List the runners for a project:

```
runners = gl.project_runners.list(project_id=1)
# or
runners = project.runners.list()
```

Enable a specific runner for a project:

```
p_runner = gl.project_runners.create({'runner_id': runner.id}, project_id=1)
# or
p_runner = project.runners.create({'runner_id': runner.id})
```

Disable a specific runner for a project:

```
gl.project_runners.delete(runner.id)
# or
project.runners.delete(runner.id)
```

Settings

Use `ApplicationSettings` objects to manipulate Gitlab settings. The *gitlab.Gitlab.settings* manager object provides helper functions.

Examples

Get the settings:

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

Update the settings:

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

Snippets

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

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

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()
```

System hooks

Use `Hook` objects to manipulate system hooks. The *`gitlab.Gitlab.hooks`* manager object provides helper functions.

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()
```

Templates

You can request templates for different type of files:

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

License templates

- Object class: `License`
- Manager object: *`gitlab.Gitlab.licenses`*

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)
```

.gitignore templates

- Object class: `Gitignore`
- Manager object: `gitlab.Gitlab.gitognores`

Examples

List known gitignore templates:

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

Get a gitignore template:

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

GitLab CI templates

- Object class: `Gitlabciyml`
- Manager object: *gitlab.Gitlab.gitlabciymls*

Examples

List known GitLab CI templates:

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

Get a GitLab CI template:

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

Todos

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

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()
```

Users

Use `User` objects to manipulate repository branches.

To create `User` objects use the *gitlab.Gitlab.users* manager.

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()
```

SSH keys

Use the `UserKey` objects to manage user keys.

To create `UserKey` objects use the `User.keys` or *gitlab.Gitlab.user_keys* managers.

Exemples

List SSH keys for a user:

```
keys = gl.user_keys.list(user_id=1)
# or
keys = user.keys.list()
```

Get an SSH key for a user:

```
key = gl.user_keys.list(1, user_id=1)
# or
key = user.keys.get(1)
```

Create an SSH key for a user:

```
k = gl.user_keys.create({'title': 'my_key',
                         'key': open('/home/me/.ssh/id_rsa.pub').read()},
                        user_id=2)
# or
k = user.keys.create({'title': 'my_key',
                      'key': open('/home/me/.ssh/id_rsa.pub').read()})
```

Delete an SSH key for a user:

```
gl.user_keys.delete(1, user_id=1)
# or
user.keys.delete(1)
# or
key.delete()
```

Emails

Use the `UserEmail` objects to manage user emails.

To create `UserEmail` objects use the `User.emails` or *gitlab.Gitlab.user_emails* managers.

Exemples

List emails for a user:

```
emails = gl.user_emails.list(user_id=1)
# or
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 = gl.user_emails.create({'email': 'foo@bar.com'}, user_id=2)
# or
k = user.emails.create({'email': 'foo@bar.com'})
```

Delete an email for a user:

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

Current User

Use the `CurrentUser` object to get information about the currently logged-in user.

Use the `CurrentUserKey` objects to manage user keys.

To create `CurrentUserKey` objects use the `gitlab.objects.CurrentUser.keys` manager.

Use the `CurrentUserEmail` objects to manage user emails.

To create `CurrentUserEmail` objects use the `gitlab.objects.CurrentUser.emails` manager.

Examples

Get the current user:

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

List the current user SSH keys:

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

Get a key for the current user:

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

Create a key for the current user:

```
key = gl.user.keys.create({'id': 'my_key', 'key': key_content})
```

Delete a key for the current user:

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

List the current user emails:

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

Get an email for the current user:

```
email = gl.user.emails.get(1)
```

Create an email for the current user:

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

Delete an email for the current user:

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

Sidekiq metrics

Use the `gitlab.Gitlab.sideqik` manager object to access Gitlab Sidekiq server metrics.

Examples

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

CHAPTER 5

Upgrading from python-gitlab 0.10 and earlier

`python-gitlab` 0.11 introduces new objects which make the API cleaner and easier to use. The feature set is unchanged but some methods have been deprecated in favor of the new manager objects.

Deprecated methods will be remove in a future release.

Gitlab object migration

The objects constructor methods are deprecated:

- `Hook()`
- `Project()`
- `UserProject()`
- `Group()`
- `Issue()`
- `User()`
- `Team()`

Use the new managers objects instead. For example:

```
# Deprecated syntax
p1 = gl.Project({'name': 'myCoolProject'})
p1.save()
p2 = gl.Project(id=1)
p_list = gl.Project()

# New syntax
p1 = gl.projects.create({'name': 'myCoolProject'})
p2 = gl.projects.get(1)
p_list = gl.projects.list()
```

The following methods are also deprecated:

- `search_projects()`
- `owned_projects()`
- `all_projects()`

Use the `projects` manager instead:

```
# Deprecated syntax
l1 = gl.search_projects('whatever')
l2 = gl.owned_projects()
l3 = gl.all_projects()

# New syntax
l1 = gl.projects.search('whatever')
l2 = gl.projects.owned()
l3 = gl.projects.all()
```

GitlabObject objects migration

The following constructor methods are deprecated in favor of the matching managers:

Deprecated method	Matching manager
`User.Key()`	`User.keys`
`CurrentUser.Key()`	`CurrentUser.keys`
`Group.Member()`	`Group.members`
`ProjectIssue.Note()`	`ProjectIssue.notes`
`ProjectMergeRequest.Note()`	`ProjectMergeRequest.notes`
`ProjectSnippet.Note()`	`ProjectSnippet.notes`
`Project.Branch()`	`Project.branches`
`Project.Commit()`	`Project.commits`
`Project.Event()`	`Project.events`
`Project.File()`	`Project.files`
`Project.Hook()`	`Project.hooks`
`Project.Key()`	`Project.keys`
`Project.Issue()`	`Project.issues`
`Project.Label()`	`Project.labels`
`Project.Member()`	`Project.members`
`Project.MergeRequest()`	`Project.mergerequests`
`Project.Milestone()`	`Project.milestones`
`Project.Note()`	`Project.notes`
`Project.Snippet()`	`Project.snippets`
`Project.Tag()`	`Project.tags`
`Team.Member()`	`Team.members`
`Team.Project()`	`Team.projects`

For example:

```
# Deprecated syntax
p = gl.Project(id=2)
issues = p.Issue()

# New syntax
```

```
p = gl.projects.get(2)
issues = p.issues.list()
```

CHAPTER 6

API documentation

gitlab package

Module contents

Wrapper for the GitLab API.

class gitlab.**Gitlab**(*url*, *private_token=None*, *email=None*, *password=None*, *ssl_verify=True*, *http_username=None*, *http_password=None*, *timeout=None*, *api_version='3'*)

Bases: object

Represents a GitLab server connection.

Parameters

- **url** (*str*) – The URL of the GitLab server.
- **private_token** (*str*) – The user private token
- **email** (*str*) – The user email or login.
- **password** (*str*) – The user password (associated with email).
- **ssl_verify** (*bool*) – Whether SSL certificates should be validated.
- **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)

broadcastmessages

BroadcastMessageManager manager for BroadcastMessage objects.

deploykeys

DeployKeyManager manager for DeployKey objects.

gitignores
`GitignoreManager` manager for `Gitignore` objects.

gitlabciymls
`GitlabciymlManager` manager for `Gitlabciyml` objects.

group_accessrequests
`GroupAccessRequestManager` manager for `GroupAccessRequest` objects.

group_issues
`GroupIssueManager` manager for `GroupIssue` objects.

group_members
`GroupMemberManager` manager for `GroupMember` objects.

group_notificationsettings
`GroupNotificationSettingsManager` manager for `GroupNotificationSettings` objects.

group_project_accessrequests
`ProjectAccessRequestManager` manager for `ProjectAccessRequest` objects.

group_project_board_lists
`ProjectBoardListManager` manager for `ProjectBoardList` objects.

group_project_boards
`ProjectBoardManager` manager for `ProjectBoard` objects.

group_project_branches
`ProjectBranchManager` manager for `ProjectBranch` objects.

group_project_builds
`ProjectBuildManager` manager for `ProjectBuild` objects.

group_project_commits
`ProjectCommitManager` manager for `ProjectCommit` objects.

group_project_deployments
`ProjectDeploymentManager` manager for `ProjectDeployment` objects.

group_project_environments
`ProjectEnvironmentManager` manager for `ProjectEnvironment` objects.

group_project_events
`ProjectEventManager` manager for `ProjectEvent` objects.

group_project_files
`ProjectFileManager` manager for `ProjectFile` objects.

group_project_forks
`ProjectForkManager` manager for `ProjectFork` objects.

group_project_hooks
`ProjectHookManager` manager for `ProjectHook` objects.

group_project_issues
`ProjectIssueManager` manager for `ProjectIssue` objects.

group_project_keys
`ProjectKeyManager` manager for `ProjectKey` objects.

group_project_labels
`ProjectLabelManager` manager for `ProjectLabel` objects.

group_project_members
ProjectMemberManager manager for ProjectMember objects.

group_project_mergerequests
ProjectMergeRequestManager manager for ProjectMergeRequest objects.

group_project_milestones
ProjectMilestoneManager manager for ProjectMilestone objects.

group_project_notes
ProjectNoteManager manager for ProjectNote objects.

group_project_notificationsettings
ProjectNotificationSettingsManager manager for ProjectNotificationSettings objects.

group_project_pipelines
ProjectPipelineManager manager for ProjectPipeline objects.

group_project_runners
ProjectRunnerManager manager for ProjectRunner objects.

group_project_services
ProjectServiceManager manager for ProjectService objects.

group_project_snippets
ProjectSnippetManager manager for ProjectSnippet objects.

group_project_tags
ProjectTagManager manager for ProjectTag objects.

group_project_triggers
ProjectTriggerManager manager for ProjectTrigger objects.

group_project_variables
ProjectVariableManager manager for ProjectVariable objects.

group_projects
GroupProjectManager manager for GroupProject objects.

groups
GroupManager manager for Group objects.

hooks
HookManager manager for Hook objects.

issues
IssueManager manager for Issue objects.

keys
KeyManager manager for Key objects.

licenses
LicenseManager manager for License objects.

namespaces
NamespaceManager manager for Namespace objects.

notificationsettings
NotificationSettingsManager manager for NotificationSettings objects.

project_accessrequests
ProjectAccessRequestManager manager for ProjectAccessRequest objects.

project_board_lists
`ProjectBoardListManager` manager for `ProjectBoardList` objects.

project_boards
`ProjectBoardManager` manager for `ProjectBoard` objects.

project_branches
`ProjectBranchManager` manager for `ProjectBranch` objects.

project_builds
`ProjectBuildManager` manager for `ProjectBuild` objects.

project_commit_comments
`ProjectCommitCommentManager` manager for `ProjectCommitComment` objects.

project_commit_statuses
`ProjectCommitStatusManager` manager for `ProjectCommitStatus` objects.

project_commits
`ProjectCommitManager` manager for `ProjectCommit` objects.

project_deployments
`ProjectDeploymentManager` manager for `ProjectDeployment` objects.

project_environments
`ProjectEnvironmentManager` manager for `ProjectEnvironment` objects.

project_events
`ProjectEventManager` manager for `ProjectEvent` objects.

project_files
`ProjectFileManager` manager for `ProjectFile` objects.

project_forks
`ProjectForkManager` manager for `ProjectFork` objects.

project_hooks
`ProjectHookManager` manager for `ProjectHook` objects.

project_issue_notes
`ProjectIssueNoteManager` manager for `ProjectIssueNote` objects.

project_issues
`ProjectIssueManager` manager for `ProjectIssue` objects.

project_keys
`ProjectKeyManager` manager for `ProjectKey` objects.

project_labels
`ProjectLabelManager` manager for `ProjectLabel` objects.

project_members
`ProjectMemberManager` manager for `ProjectMember` objects.

project_mergerequest_diffs
`ProjectMergeRequestDiffManager` manager for `ProjectMergeRequestDiff` objects.

project_mergerequest_notes
`ProjectMergeRequestNoteManager` manager for `ProjectMergeRequestNote` objects.

project_mergerequests
`ProjectMergeRequestManager` manager for `ProjectMergeRequest` objects.

project_milestones
ProjectMilestoneManager manager for ProjectMilestone objects.

project_notes
ProjectNoteManager manager for ProjectNote objects.

project_notificationsettings
ProjectNotificationSettingsManager manager for ProjectNotificationSettings objects.

project_pipelines
ProjectPipelineManager manager for ProjectPipeline objects.

project_runners
ProjectRunnerManager manager for ProjectRunner objects.

project_services
ProjectServiceManager manager for ProjectService objects.

project_snippet_notes
ProjectSnippetNoteManager manager for ProjectSnippetNote objects.

project_snippets
ProjectSnippetManager manager for ProjectSnippet objects.

project_tags
ProjectTagManager manager for ProjectTag objects.

project_triggers
ProjectTriggerManager manager for ProjectTrigger objects.

project_variables
ProjectVariableManager manager for ProjectVariable objects.

projects
ProjectManager manager for Project objects.

runners
RunnerManager manager for Runner objects.

settings
ApplicationSettingsManager manager for ApplicationSettings objects.

snippets
SnippetManager manager for Snippet objects.

team_members
TeamMemberManager manager for TeamMember objects.

team_projects
TeamProjectManager manager for TeamProject objects.

teams
TeamManager manager for Team objects.

todos
TodoManager manager for Todo objects.

user_emails
UserEmailManager manager for UserEmail objects.

user_keys
UserKeyManager manager for UserKey objects.

user_projects
`UserProjectManager` manager for `UserProject` objects.

users
`UserManager` manager for `User` objects.

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.

credentials_auth()
Performs an authentication using email/password.

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

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

set_credentials(*email*, *password*)
Sets the email/login and password for authentication.

Parameters

- **email** (*str*) – The user email or login.
- **password** (*str*) – The user password.

set_token(*token*)
Sets the private token for authentication.

Parameters **token** (*str*) – The private token.

set_url(*url*)
Updates the GitLab URL.

Parameters **url** (*str*) – Base URL of the GitLab server.

ssl_verify = None
Whether SSL certificates should be validated

timeout = None
Timeout to use for requests to gitlab server

token_auth()
Performs an authentication using the private token.

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)

gitlab.base

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.**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*)

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)

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

Manager for ApplicationSettings objects.

ApplicationSettings objects can be updated.

get(***kwargs*)
Get a single object of type 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)

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

Manager for BroadcastMessage objects.

BroadcastMessage objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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*

managers = ((‘emails’, ‘CurrentUserEmailManager’, [(‘user_id’, ‘id’)]), (‘keys’, ‘CurrentUserKeyManager’, [(‘user_id’,

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

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

Manager for `CurrentUserEmail` objects.

`CurrentUserEmail` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 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 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*

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

Manager for CurrentUserKey objects.

CurrentUserKey objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 `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*

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

Manager for `DeployKey` objects.

`DeployKey` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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*

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

Manager for `Gitignore` objects.

`Gitignore` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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*

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

Manager for `Gitlabciyml` objects.

`Gitlabciyml` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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
:class:GroupAccessRequestManager - Manager for :class: objects.

members
:class:GroupMemberManager - Manager for :class: objects.

notificationsettings
:class:GroupNotificationSettingsManager - Manager for :class: objects.

projects
:class:GroupProjectManager - Manager for :class: objects.

issues
:class:GroupIssueManager - Manager for :class: 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')]), ('members', 'GroupMemberManag

transfer_project(*id*, ***kwargs*)
Transfers a project to this new groups.

Attrs: 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.

Attrs: 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.

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

Manager for GroupAccessRequest objects.

GroupAccessRequest objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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*

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

Manager for GroupIssue objects.

GroupIssue objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 Group objects.

Group objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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)

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

Manager for `GroupMember` objects.

`GroupMember` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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)

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

Manager for GroupNotificationSettings objects.

GroupNotificationSettings objects can be updated.

get(***kwargs*)
Get a single object of type 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*

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

Manager for GroupProject objects.

GroupProject objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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.**HookManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for Hook objects.

Hook objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 `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 `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.`**IssueManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `Issue` objects.

`Issue` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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.**KeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for Key objects.

Key objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type Key.

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 Key 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 Key

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

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

Manager for License objects.

License objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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*

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

Manager for Namespace objects.

Namespace objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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)

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

Manager for NotificationSettings objects.

NotificationSettings objects can be updated.

get(***kwargs*)
Get a single object of type 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
:class:ProjectAccessRequestManager - Manager for :class: objects.

boards
:class:ProjectBoardManager - Manager for :class: objects.

board_lists
:class:ProjectBoardListManager - Manager for :class: objects.

branches
:class:ProjectBranchManager - Manager for :class: objects.

builds
:class:ProjectBuildManager - Manager for :class: objects.

commits
:class:ProjectCommitManager - Manager for :class: objects.

deployments
:class:ProjectDeploymentManager - Manager for :class: objects.

environments
:class:ProjectEnvironmentManager - Manager for :class: objects.

events
:class:ProjectEventManager - Manager for :class: objects.

files
:class:ProjectFileManager - Manager for :class: objects.

forks
:class:ProjectForkManager - Manager for :class: objects.

hooks
:class:ProjectHookManager - Manager for :class: objects.

keys
:class:ProjectKeyManager - Manager for :class: objects.

issues
:class:ProjectIssueManager - Manager for :class: objects.

labels
:class:ProjectLabelManager - Manager for :class: objects.

members
:class:ProjectMemberManager - Manager for :class: objects.

mergerequests
:class:ProjectMergeRequestManager - Manager for :class: objects.

milestones
:class:ProjectMilestoneManager - Manager for :class: objects.

notes
:class:ProjectNoteManager - Manager for :class: objects.

notificationsettings
:class:ProjectNotificationSettingsManager - Manager for :class: objects.

pipelines
:class:ProjectPipelineManager - Manager for :class: objects.

runners
:class:ProjectRunnerManager - Manager for :class: objects.

services
:class:ProjectServiceManager - Manager for :class: objects.

snippets
:class:ProjectSnippetManager - Manager for :class: objects.

tags
:class:ProjectTagManager - Manager for :class: objects.

triggers
:class:ProjectTriggerManager - Manager for :class: objects.

variables
:class:ProjectVariableManager - Manager for :class: 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.

archive_(***kwargs*)

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')]), ('boards', 'ProjectBoardManager

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.

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.

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.

unarchive_(***kwargs*)

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.

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

approve(*access_level=30*, ***kwargs*)
Approve an access request.

Attrs: 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.

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

Manager for ProjectAccessRequest objects.

ProjectAccessRequest objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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*

managers = (('lists', 'ProjectBoardListManager', [('project_id', 'project_id'), ('board_id', '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)

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

Manager for `ProjectBoardList` objects.

`ProjectBoardList` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 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 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 ProjectBoard objects.

ProjectBoard objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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*

protect(*protect=True*, ***kwargs*)
Protects the branch.

unprotect(***kwargs*)
Unprotects the branch.

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

Manager for ProjectBranch objects.

ProjectBranch objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.

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.

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 `ProjectBuild` objects.

`ProjectBuild` objects **cannot** be updated.

list (***kwargs*)
Returns a list of objects of type `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 `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.

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'), ('commit_id', 'id')]), ('statu

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

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

Manager for ProjectCommitComment objects.

ProjectCommitComment objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 ProjectCommit objects.

ProjectCommit objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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*

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

Manager for ProjectCommitStatus objects.

ProjectCommitStatus objects **cannot** be updated.

list (***kwargs*)
Returns a list of objects of type 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 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 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*

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

Manager for ProjectDeployment objects.

ProjectDeployment objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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)

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

Manager for ProjectEnvironment objects.

ProjectEnvironment objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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*

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

Manager for ProjectEvent objects.

ProjectEvent objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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)

decode()
Returns the decoded content of the file.

Returns the decoded content.

Return type (str)

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

Manager for `ProjectFile` objects.

`ProjectFile` objects can be updated.

get(***kwargs*)
Get a single object of type `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 `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*

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

Manager for `ProjectFork` objects.

`ProjectFork` objects **cannot** be updated.

create(*data*, ***kwargs*)
Create an object of type `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)

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

Manager for ProjectHook objects.

ProjectHook objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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
:class:ProjectIssueNoteManager - Manager for :class: 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.

managers = (('notes', 'ProjectIssueNoteManager', [('project_id', 'project_id'), ('issue_id', 'id')]),)

move(*to_project_id*, ***kwargs*)
Move the issue to another project.

Raises GitlabConnectionError – If the server cannot be reached.

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.

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 `ProjectIssue` objects.

`ProjectIssue` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 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)

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

Manager for ProjectIssueNote objects.

ProjectIssueNote objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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*

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

Manager for ProjectKey objects.

ProjectKey objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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)

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 `ProjectLabel` objects.

`ProjectLabel` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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 `Project` objects.

`Project` objects can be updated.

list (***kwargs*)
Returns a list of objects of type `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 `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 `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)

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

Manager for ProjectMember objects.

ProjectMember objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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
:class:ProjectMergeRequestNoteManager - Manager for :class: objects.

diffs
:class:ProjectMergeRequestDiffManager - Manager for :class: 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_request_id', 'id')]), ('d

merge(*merge_commit_message=None*, *should_remove_source_branch=False*, *merged_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
- **merged_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

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*

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

Manager for ProjectMergeRequestDiff objects.

ProjectMergeRequestDiff objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 ProjectMergeRequest objects.

ProjectMergeRequest objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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)

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

Manager for ProjectMergeRequestNote objects.

ProjectMergeRequestNote objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 `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 `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)

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.

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

Manager for ProjectMilestone objects.

ProjectMilestone objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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*

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

Manager for ProjectNote objects.

ProjectNote objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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)

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

Manager for ProjectNotificationSettings objects.

ProjectNotificationSettings objects can be updated.

get(**kwargs)
Get a single object of type 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*

cancel(**kwargs)
Cancel builds in a pipeline.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabPipelineCancelError – If the retry cannot be done.

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 ProjectPipeline objects.

ProjectPipeline objects **cannot** be updated.

list(**kwargs)
Returns a list of objects of type 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 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 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*

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

Manager for `ProjectRunner` objects.

`ProjectRunner` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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)

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

Manager for ProjectService objects.

ProjectService objects can be updated.

get(***kwargs*)
Get a single object of type 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
:class:ProjectSnippetNoteManager - Manager for :class: 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'), ('snippet_id', 'id')]),)

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

Manager for `ProjectSnippet` objects.

`ProjectSnippet` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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*

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

Manager for ProjectSnippetNote objects.

ProjectSnippetNote objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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*

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.

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

Manager for ProjectTag objects.

ProjectTag objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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)

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

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

Manager for `ProjectTrigger` objects.

`ProjectTrigger` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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)

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

Manager for `ProjectVariable` objects.

`ProjectVariable` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 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 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)

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

Manager for Runner objects.

Runner objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 `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.`**SnippetManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `Snippet` objects.

`Snippet` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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*

managers = (('members', 'TeamMemberManager', [('team_id', 'id')]), ('projects', 'TeamProjectManager', [('team_id',

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

Manager for Team objects.

Team objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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*

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

Manager for TeamMember objects.

TeamMember objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 `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 `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*

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

Manager for `TeamProject` objects.

`TeamProject` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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*

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

Manager for `Todo` objects.

`Todo` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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
:class:UserEmailManager - Manager for :class: objects.

keys
:class:UserKeyManager - Manager for :class: objects.

projects
:class:UserProjectManager - Manager for :class: 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’, ‘UserKeyManager’, [(‘user_id’, ‘id’)]), (‘project

unblock(***kwargs*)
Unblocks the user.

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

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

Manager for UserEmail objects.

UserEmail objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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*

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

Manager for UserKey objects.

UserKey objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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 `User` objects.

`User` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 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*

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

Manager for UserProject objects.

UserProject objects **cannot** be updated.

create(*data*, ***kwargs*)
Create an object of type 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*

gitlab.v4.objects module

class gitlab.v4.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)

class gitlab.v4.objects.**ApplicationSettingsManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ApplicationSettings objects.

ApplicationSettings objects can be updated.

get(***kwargs*)
Get a single object of type 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.v4.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)

class gitlab.v4.objects.**BroadcastMessageManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for BroadcastMessage objects.

BroadcastMessage objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**CurrentUser**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

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

class gitlab.v4.objects.**CurrentUserEmail**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**CurrentUserEmailManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `CurrentUserEmail` objects.

`CurrentUserEmail` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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.v4.objects.`**CurrentUserKey**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**CurrentUserKeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `CurrentUserKey` objects.

`CurrentUserKey` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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.v4.objects.`**DeployKey**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**DeployKeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for DeployKey objects.

DeployKey objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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.v4.objects.**Dockerfile**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**DockerfileManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for Dockerfile objects.

Dockerfile objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type Dockerfile.

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 Dockerfile 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 *Dockerfile*

class gitlab.v4.objects.**Gitignore**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**GitignoreManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for Gitignore objects.

Gitignore objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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.v4.objects.**Gitlabciyml**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**GitlabciymlManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for Gitlabciyml objects.

Gitlabciyml objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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.v4.objects.**Group**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

accessrequests
:class:GroupAccessRequestManager - Manager for :class: objects.

members
:class:GroupMemberManager - Manager for :class: objects.

notificationsettings
:class:GroupNotificationSettingsManager - Manager for :class: objects.

projects
:class:GroupProjectManager - Manager for :class: objects.

issues
:class:GroupIssueManager - Manager for :class: objects.

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

- `name` (optional)
- `path` (optional)
- `description` (optional)
- `visibility` (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)

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

transfer_project(*id*, ***kwargs*)
Transfers a project to this new groups.

Attrs: 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.v4.objects.`**GroupAccessRequest**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

approve(*access_level=30*, ***kwargs*)
Approve an access request.

Attrs: 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.

class `gitlab.v4.objects.`**GroupAccessRequestManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `GroupAccessRequest` objects.

`GroupAccessRequest` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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.v4.objects.`**GroupIssue**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**GroupIssueManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `GroupIssue` objects.

`GroupIssue` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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.v4.objects.`**GroupManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `Group` objects.

`Group` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `Group`.

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

- name (required)
- path (required)
- description (optional)
- visibility (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*

class gitlab.v4.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)

class gitlab.v4.objects.**GroupMemberManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for GroupMember objects.

GroupMember objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**GroupNotificationSettings**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.v4.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)

class gitlab.v4.objects.**GroupNotificationSettingsManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for GroupNotificationSettings objects.

GroupNotificationSettings objects can be updated.

get(***kwargs*)
Get a single object of type 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.v4.objects.**GroupProject**(**args*, ***kwargs*)
Bases: *gitlab.v4.objects.Project*

class gitlab.v4.objects.**GroupProjectManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.v4.objects.ProjectManager*

Manager for GroupProject objects.

GroupProject objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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.v4.objects.**HookManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for Hook objects.

Hook objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**IssueManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for Issue objects.

Issue objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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.v4.objects.**License**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**LicenseManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for License objects.

License objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 `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.v4.objects.`**Namespace**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**NamespaceManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `Namespace` objects.

`Namespace` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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.v4.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)

class `gitlab.v4.objects.`**`NotificationSettingsManager`**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `NotificationSettings` objects.

`NotificationSettings` objects can be updated.

`get`(***kwargs*)
Get a single object of type `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.v4.objects.`**`Project`**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

`accessrequests`
:class:ProjectAccessRequestManager - Manager for :class: objects.

`boards`
:class:ProjectBoardManager - Manager for :class: objects.

`board_lists`
:class:ProjectBoardListManager - Manager for :class: objects.

`branches`
:class:ProjectBranchManager - Manager for :class: objects.

`jobs`
:class:ProjectJobManager - Manager for :class: objects.

`commits`
:class:ProjectCommitManager - Manager for :class: objects.

`deployments`
:class:ProjectDeploymentManager - Manager for :class: objects.

`environments`
:class:ProjectEnvironmentManager - Manager for :class: objects.

`events`
:class:ProjectEventManager - Manager for :class: objects.

`files`
:class:ProjectFileManager - Manager for :class: objects.

`forks`
:class:ProjectForkManager - Manager for :class: objects.

hooks
:class:ProjectHookManager - Manager for :class: objects.

keys
:class:ProjectKeyManager - Manager for :class: objects.

issues
:class:ProjectIssueManager - Manager for :class: objects.

labels
:class:ProjectLabelManager - Manager for :class: objects.

members
:class:ProjectMemberManager - Manager for :class: objects.

mergerequests
:class:ProjectMergeRequestManager - Manager for :class: objects.

milestones
:class:ProjectMilestoneManager - Manager for :class: objects.

notes
:class:ProjectNoteManager - Manager for :class: objects.

notificationsettings
:class:ProjectNotificationSettingsManager - Manager for :class: objects.

pipelines
:class:ProjectPipelineManager - Manager for :class: objects.

runners
:class:ProjectRunnerManager - Manager for :class: objects.

services
:class:ProjectServiceManager - Manager for :class: objects.

snippets
:class:ProjectSnippetManager - Manager for :class: objects.

tags
:class:ProjectTagManager - Manager for :class: objects.

triggers
:class:ProjectTriggerManager - Manager for :class: objects.

variables
:class:ProjectVariableManager - Manager for :class: 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)
- `visibility` (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)

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')]), ('boards', 'ProjectBoardManager

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_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=''*, ***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

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.

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.

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_pipeline(*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.

class `gitlab.v4.objects.`**ProjectAccessRequest**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

approve(*access_level=30*, ***kwargs*)
Approve an access request.

Attrs: 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.

class `gitlab.v4.objects.`**ProjectAccessRequestManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectAccessRequest` objects.

`ProjectAccessRequest` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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.v4.objects.`**ProjectBoard**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

managers = (('lists', 'ProjectBoardListManager', [('project_id', 'project_id'), ('board_id', 'id')]),)

class `gitlab.v4.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)

class `gitlab.v4.objects.`**ProjectBoardListManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectBoardList` objects.

`ProjectBoardList` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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.v4.objects.`**ProjectBoardManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectBoard` objects.

`ProjectBoard` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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.v4.objects.`**ProjectBranch**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

protect(*protect=True*, ***kwargs*)
Protects the branch.

unprotect(***kwargs*)
Unprotects the branch.

class `gitlab.v4.objects.`**ProjectBranchManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectBranch` objects.

`ProjectBranch` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `ProjectBranch`.

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

- `project_id` (required if not discovered on the parent objects)
- `branch` (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.v4.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.

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'), ('commit_id', 'id')]), ('statu

class gitlab.v4.objects.**ProjectCommitComment**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**ProjectCommitCommentManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectCommitComment objects.

ProjectCommitComment objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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.v4.objects.**ProjectCommitManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectCommit objects.

ProjectCommit objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 `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 `ProjectCommit`.

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

- `project_id` (required if not discovered on the parent objects)
- `branch` (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.v4.objects.`**ProjectCommitStatus**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**ProjectCommitStatusManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectCommitStatus` objects.

`ProjectCommitStatus` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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.v4.objects.`**ProjectDeployment**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**ProjectDeploymentManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectDeployment` objects.

`ProjectDeployment` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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.v4.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)

class gitlab.v4.objects.**ProjectEnvironmentManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectEnvironment objects.

ProjectEnvironment objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**ProjectEvent**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**ProjectEventManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectEvent objects.

ProjectEvent objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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.v4.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` (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)

decode()
Returns the decoded content of the file.

Returns the decoded content.

Return type (str)

class gitlab.v4.objects.**ProjectFileManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectFile` objects.

`ProjectFile` objects can be updated.

get(*id*, ***kwargs*)
Get a single object of type `ProjectFile` using its `id`.

Available keys for `kwargs` are:

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

create(*data*, ***kwargs*)
Create an object of type `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` (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*

raw(*filepath*, *ref*, *streamed=False*, *action=None*, *chunk_size=1024*, ***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.

Returns The file content

Return type str

Raises

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

class `gitlab.v4.objects.`**ProjectFork**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**ProjectForkManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectFork` objects.

`ProjectFork` objects **cannot** be updated.

create(*data*, ***kwargs*)
Create an object of type `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.v4.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)
- `job_events` (optional)
- `wiki_page_events` (optional)

Available keys for `kwargs` are:

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

class `gitlab.v4.objects.`**ProjectHookManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectHook` objects.

`ProjectHook` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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)
- `job_events` (optional)
- `wiki_page_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.v4.objects.`**ProjectIssue**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

notes
:class:ProjectIssueNoteManager - Manager for :class: 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(*duration*, ***kwargs*)
Set an estimated time of work for the issue.

Parameters **duration** (*str*) – duration in human format (e.g. 3h30)

Raises `GitlabConnectionError` – If the server cannot be reached.

managers = (('notes', 'ProjectIssueNoteManager', [('project_id', 'project_id'), ('issue_iid', 'iid')]),)

move(*to_project_id*, ***kwargs*)
Move the issue to another project.

Raises `GitlabConnectionError` – If the server cannot be reached.

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.

subscribe(***kwargs*)
Subscribe to an issue.

Raises

- `GitlabConnectionError` – If the server cannot be reached.
- `GitlabSubscribeError` – If the subscription cannot be done

time_estimate(*duration*, ***kwargs*)
Set an estimated time of work for the issue.

Parameters **duration** (*str*) – duration in human format (e.g. 3h30)

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.v4.objects.**ProjectIssueManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectIssue objects.

ProjectIssue objects can be updated.

list(***kwargs*)
Returns a list of objects of type ProjectIssue.

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 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 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.v4.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)

class gitlab.v4.objects.**ProjectIssueNoteManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectIssueNote objects.

ProjectIssueNote objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 ProjectIssueNote.

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

- project_id (required if not discovered on the parent objects)
- issue_iid (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.v4.objects.**ProjectJob**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

artifacts(*streamed=False*, *action=None*, *chunk_size=1024*, ***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.

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.

cancel(***kwargs*)
Cancel the job.

erase(***kwargs*)
Erase the job (remove job 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 job explicitly.

retry(***kwargs*)
Retry the job.

trace(*streamed=False*, *action=None*, *chunk_size=1024*, ***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.

Returns The trace.

Return type str

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabGetError – If the trace is not available.

class gitlab.v4.objects.**ProjectJobManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectJob objects.

ProjectJob objects **cannot** be updated.

list(**kwargs)
Returns a list of objects of type ProjectJob.

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 ProjectJob 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 *ProjectJob*

class gitlab.v4.objects.**ProjectKey**(*gl*, *data=None*, **kwargs)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**ProjectKeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectKey objects.

ProjectKey objects **cannot** be updated.

list(**kwargs)
Returns a list of objects of type 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 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 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)

enable(*key_id*)
Enable a deploy key for a project.

obj_cls
alias of *ProjectKey*

class gitlab.v4.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)

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.v4.objects.**ProjectLabelManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectLabel objects.

ProjectLabel objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 `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 `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.v4.objects.`**ProjectManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `Project` objects.

`Project` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `Project`.

Available keys for `kwargs` are:

- `search` (optional)
- `owned` (optional)
- `starred` (optional)
- `archived` (optional)
- `visibility` (optional)
- `order_by` (optional)
- `sort` (optional)
- `simple` (optional)
- `membership` (optional)

- `statistics` (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 `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 `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)
- `visibility` (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)

obj_cls
alias of *Project*

class gitlab.v4.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)

class gitlab.v4.objects.**ProjectMemberManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectMember objects.

ProjectMember objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**ProjectMergeRequest**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

notes
:class:ProjectMergeRequestNoteManager - Manager for :class: objects.

diffs
:class:ProjectMergeRequestDiffManager - Manager for :class: 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(*duration*, ***kwargs*)
Set an estimated time of work for the merge request.

Parameters **duration** (*str*) – duration in human format (e.g. 3h30)

Raises `GitlabConnectionError` – If the server cannot be reached.

cancel_merge_when_pipeline_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_request_iid', 'iid')]), ('

merge(*merge_commit_message=None*, *should_remove_source_branch=False*, *merged_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
- **merged_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

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(*duration*, ***kwargs*)

Set an estimated time of work for the merge request.

Parameters **duration** (*str*) – duration in human format (e.g. 3h30)

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.v4.objects.**ProjectMergeRequestDiff**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**ProjectMergeRequestDiffManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectMergeRequestDiff objects.

ProjectMergeRequestDiff objects **cannot** be updated.

list(**kwargs)
Returns a list of objects of type 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 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.v4.objects.**ProjectMergeRequestManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectMergeRequest objects.

ProjectMergeRequest objects can be updated.

list(**kwargs)
Returns a list of objects of type ProjectMergeRequest.

Available keys for kwargs are:

•iids (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 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 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.v4.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)

class gitlab.v4.objects.**ProjectMergeRequestNoteManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectMergeRequestNote objects.

ProjectMergeRequestNote objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `ProjectMergeRequestNote`.

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

- `project_id` (required if not discovered on the parent objects)
- `merge_request_iid` (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.v4.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)

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.

class `gitlab.v4.objects.`**`ProjectMilestoneManager`**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectMilestone` objects.

`ProjectMilestone` objects can be updated.

`list`(***kwargs*)
Returns a list of objects of type `ProjectMilestone`.

Available keys for `kwargs` are:

- `iids` (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 `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 `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.v4.objects.`**`ProjectNote`**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**ProjectNoteManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectNote objects.

ProjectNote objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**ProjectNotificationSettings**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.v4.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)

class gitlab.v4.objects.**ProjectNotificationSettingsManager**(*gl*, *parent=None*, *args=[]*)

Bases: *gitlab.base.BaseManager*

Manager for ProjectNotificationSettings objects.

ProjectNotificationSettings objects can be updated.

get(***kwargs*)

Get a single object of type 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.v4.objects.**ProjectPipeline**(*gl*, *data=None*, ***kwargs*)

Bases: *gitlab.base.GitlabObject*

cancel(***kwargs*)

Cancel builds in a pipeline.

Raises

- GitlabConnectionError – If the server cannot be reached.
- GitlabPipelineCancelError – If the retry cannot be done.

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.v4.objects.**ProjectPipelineManager**(*gl*, *parent=None*, *args=[]*)

Bases: *gitlab.base.BaseManager*

Manager for ProjectPipeline objects.

ProjectPipeline objects **cannot** be updated.

list(***kwargs*)

Returns a list of objects of type 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 `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 `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.v4.objects.`**ProjectRunner**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**ProjectRunnerManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectRunner` objects.

`ProjectRunner` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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.v4.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)

class gitlab.v4.objects.**ProjectServiceManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectService objects.

ProjectService objects can be updated.

get(***kwargs*)
Get a single object of type 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.v4.objects.**ProjectSnippet**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

notes
:class:ProjectSnippetNoteManager - Manager for :class: objects.

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

- title (optional)
- file_name (optional)
- code (optional)
- visibility (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'), ('snippet_id', 'id')]),)

class gitlab.v4.objects.**ProjectSnippetManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectSnippet objects.

ProjectSnippet objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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 (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.v4.objects.**ProjectSnippetNote**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**ProjectSnippetNoteManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectSnippetNote objects.

ProjectSnippetNote objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**ProjectTag**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

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.

class gitlab.v4.objects.**ProjectTagManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectTag objects.

ProjectTag objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.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)

class gitlab.v4.objects.**ProjectTrigger**(*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)

Available keys for kwargs are:

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

take_ownership(***kwargs*)
Update the owner of a trigger.

Raises

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

class gitlab.v4.objects.**ProjectTriggerManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for ProjectTrigger objects.

ProjectTrigger objects can be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 ProjectTrigger.

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

- project_id (required if not discovered on the parent objects)
- description (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 *ProjectTrigger*

class `gitlab.v4.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)

class `gitlab.v4.objects.`**ProjectVariableManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `ProjectVariable` objects.

`ProjectVariable` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `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.v4.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)

class `gitlab.v4.objects.`**RunnerManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `Runner` objects.

`Runner` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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.v4.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.v4.objects.`**SnippetManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `Snippet` objects.

`Snippet` objects can be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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 `Snippet`.

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

•`title` (required)

•`file_name` (required)

- `content` (required)
- `lifetime` (optional)
- `visibility` (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.v4.objects.`**Todo**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**TodoManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `Todo` objects.

`Todo` objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type `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 `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.v4.objects.`**User**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

emails
:class:UserEmailManager - Manager for :class: objects.

keys
:class:UserKeyManager - Manager for :class: objects.

projects
:class:UserProjectManager - Manager for :class: 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)
- skip_confirmation (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', 'UserKeyManager', [('user_id', 'id')]), ('project

unblock(***kwargs*)
Unblocks the user.

class gitlab.v4.objects.**UserEmail**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**UserEmailManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for UserEmail objects.

UserEmail objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**UserKey**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class gitlab.v4.objects.**UserKeyManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for UserKey objects.

UserKey objects **cannot** be updated.

list(***kwargs*)
Returns a list of objects of type 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 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 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.v4.objects.**UserManager**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for User objects.

User objects can be updated.

list(***kwargs*)
Returns a list of objects of type `User`.

Available keys for `kwargs` are:

- `active` (optional)
- `blocked` (optional)
- `username` (optional)
- `extern_uid` (optional)
- `provider` (optional)
- `external` (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 `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 `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)
- `skip_confirmation` (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)

`obj_cls`
alias of *User*

class `gitlab.v4.objects.`**`UserProject`**(*gl*, *data=None*, ***kwargs*)
Bases: *gitlab.base.GitlabObject*

class `gitlab.v4.objects.`**`UserProjectManager`**(*gl*, *parent=None*, *args=[]*)
Bases: *gitlab.base.BaseManager*

Manager for `UserProject` objects.

`UserProject` objects **cannot** be updated.

`create`(*data*, ***kwargs*)
Create an object of type `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` (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*

gitlab.exceptions module

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.**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.**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.**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*

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

CHAPTER 7

Release notes

This page describes important changes between python-gitlab releases.

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.

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

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

Version 0.21.1 - 2017-05-25

- Fix the manager name for jobs in the Project class
- Fix the docs

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

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)

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

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

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*

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()

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

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

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

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()

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

Version 0.12.1 - 2016-02-03

- Fix a broken upload to pypi

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

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

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

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)

Version 0.9.2 - 2015-07-11

- CLI: fix the update and delete subcommands (#62)

Version 0.9.1 - 2015-05-15

- Fix the setup.py script

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

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

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)

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

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

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

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.

Version 0.2 - 2013-08-08

- provide a pip requirements.txt
- drop some debug statements

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

U

V