pyapi-gitlab Documentation

Release 0.2

Itxaka Serrano Garcia

Sep 19, 2017

Contents

pyapi-gitlab is a wrapper to access all the functions of Gitlab from our python scripts.

CHAPTER 1

How to use it

There are several optional parameters in a lot of the commands, you should check the command documentation or the command string, for example adding an user accepts up to 7 extra parameters.

First we import our library:

```
import gitlab
```

Then we need to authenticate to our Gitlab instance. There is 3 ways of doing this.

CHAPTER 2

Authenticating via user/password

First create the instance passing the gitlab server as parameter:

```
git = gitlab.Gitlab("our_gitlab_host")
```

Then call the login() method:

```
git.login("user", "password")
```

That's it, now your gitlab instance is using the private token in all the calls. You can see it in the token variable

CHAPTER 3

Authenticating via private_token

You can also authenticate via the private_token that you can get from your gitlab profile and it's easier than using user/password

Just call the instance with the parameter token:

```
git = gitlab.Gitlab("our_gitlab_host", token="mytoken")
```

CHAPTER 4

Authenticating via oAuth2 token

You can also authenticate via the oAuth token that you can get from your gitlab profile and it's easier than using user/password

Just call the instance with the parameter oauth_token:

```
git = gitlab.Gitlab("our_gitlab_host", oauth_token="mytoken")
```

CHAPTER 5

Using sudo on the functions

All API calls support using sudo (e.g. calling the API as a different user) This is accomplished by using the setsudo() method to temporarily make all requests as another user, then calling it with no args to go back to the original user:

```
>>> git = gitlab.Gitlab(host=host)
>>> git.login(user=user, password=password)
True
>>> git.currentuser()["username"]
u'root'
>>> [[u["id"], u["username"]] for u in git.getusers()]
[[1, u'root'], [9, u'sudo_user'], [10, u'NMFUQ85Y']]
>>> # lets try with sudo_user
>>> git.setsudo(9)
>>> git.currentuser()["username"]
u'sudo_user'
>>> # lets change back to the original user
>>> git.setsudo(1)
>>> git.currentuser()["username"]
u'root'
```

CHAPTER 6

Pagination

All get* functions now accept a page and per_page parameter:

```
git.getissues(page=1, per_page=40)
```

The default is to get page 1 and 20 results per page. The max value for per_page is 100.

CHAPTER 7

Getting all results

There is a getall method which will return all results for any call that accepts pagination:

```
git.getall(git.getprojects)
```

Used in loops:

```
for project in git.getall(git.getprojects):
    pass
```

Treated as a generator:

```
print ", ".join(user['username'] for user in git.getall(git.getusers, per_page=100))
```

Wrap with list() which retrieves all the elements:

```
print 'number of users: %d' % len(list(git.getall(git.getusers)))
```

Start from any page:

```
# skip the first 4400 results
len(list(git.getall(git.getusers, page=51, per_page=80)))
```

And with positional args:

```
print len(list(git.getall(git.getgroupmembers, 191, page=3, per_page=7)))
```

CHAPTER 8

API Deprecation

We are currently working on moving to new method names but don't want to break any existing code bases. In order for us to do this we will be adding a deprecation warning to the old style methods. Please find below an example of how things will change.

Code Example:

```
# The old style to get users
getusers('bob')

# the new style
get_users('bob')
```

CHAPTER 9

API doc

We are working to get back to a 1:1 translation of the Gitlab API.

Gitlab

class gitlab.**Gitlab**(*host*, *token=None*, *oauth_token=None*, *verify_ssl=True*, *auth=None*, *timeout=None*, *suppress_http_error=True*)

Gitlab class

On init we setup the token used for all the api calls and all the urls

Parameters

- **host** – host of gitlab
- **token** – token
- **verify_ssl** – Weather or not to verify the SSL cert
- **auth** – Authentication
- **timeout** – Timeout
- **suppress_http_error** – Use `False` to unsuppress `requests.exceptions.HTTPError` exceptions on failure

Returns None

acceptmergerequest(*project_id*, *mergerequest_id*, *merge_commit_message=None*)

Update an existing merge request.

Parameters

- **project_id** – ID of the project originating the merge request
- **mergerequest_id** – ID of the merge request to accept
- **merge_commit_message** – Custom merge commit message

Returns dict of the modified merge request

addcommenttocommit (*project_id*, *author*, *sha*, *path*, *line*, *note*)
Adds an inline comment to a specific commit

Parameters

- **project_id** – project id
- **author** – The author info as returned by create mergerequest
- **sha** – The name of a repository branch or tag or if not given the default branch
- **path** – The file path
- **line** – The line number
- **note** – Text of comment

Returns True or False

addcommenttomergerequest (*project_id*, *mergerequest_id*, *note*)
Add a comment to a merge request.

Parameters

- **project_id** – ID of the project originating the merge request
- **mergerequest_id** – ID of the merge request to comment on
- **note** – Text of comment

Returns True if success

adddeploykey (*project_id*, *title*, *key*)
Creates a new deploy key for a project.

Parameters

- **project_id** – project id
- **title** – title of the key
- **key** – the key itself

Returns true if success, false if not

addgroupmember (*group_id*, *user_id*, *access_level*)
Adds a project member to a project

Parameters

- **user_id** – user id
- **access_level** – access level, see gitlab help to know more

Returns True if success

addldapgrouplink (*group_id*, *cn*, *group_access*, *provider*)
Add LDAP group link

Parameters

- **id** – The ID of a group
- **cn** – The CN of a LDAP group
- **group_access** – Minimum access level for members of the LDAP group
- **provider** – LDAP provider for the LDAP group (when using several providers)

Returns True if success

addprojecthook(*project_id*, *url*, *push=False*, *issues=False*, *merge_requests=False*, *tag_push=False*)
add a hook to a project

Parameters

- **project_id** – project id
- **url** – url of the hook

Returns True if success

addprojectmember(*project_id*, *user_id*, *access_level*)
Adds a project member to a project

Parameters

- **project_id** – project id
- **user_id** – user id
- **access_level** – access level, see gitlab help to know more

Returns True if success

addsshkey(*title*, *key*)
Add a new ssh key for the current user

Parameters

- **title** – title of the new key
- **key** – the key itself

Returns true if added, false if it didn't add it (it could be because the name or key already exists)

addsshkeyuser(*user_id*, *title*, *key*)
Add a new ssh key for the user identified by id

Parameters

- **user_id** – id of the user to add the key to
- **title** – title of the new key
- **key** – the key itself

Returns true if added, false if it didn't add it (it could be because the name or key already exists)

addsystemhook(*url*)
Add a system hook

Parameters **url** – url of the hook

Returns True if success

blockuser(*user_id*, ***kwargs*)
Block a user.

Parameters

- **user_id** – id of the user to change
- **kwargs** – Any param the the Gitlab API supports

Returns Dict of the user

compare_branches_tags_commits (*project_id*, *from_id*, *to_id*)
Compare branches, tags or commits

Parameters

- **project_id** – The ID of a project
- **from_id** – the commit sha or branch name
- **to_id** – the commit sha or branch name

Returns commit list and diff between two branches tags or commits provided by name

Raise HttpError: If invalid response returned

createbranch (*project_id*, *branch*, *ref*)
Create branch from commit SHA or existing branch

Parameters

- **project_id** – The ID of a project
- **branch** – The name of the branch
- **ref** – Create branch from commit SHA or existing branch

Returns True if success, False if not

createfile (*project_id*, *file_path*, *branch_name*, *encoding*, *content*, *commit_message*)
Creates a new file in the repository

Parameters

- **project_id** – project id
- **file_path** – Full path to new file. Ex. lib/class.rb
- **branch_name** – The name of branch
- **content** – File content
- **commit_message** – Commit message

Returns true if success, false if not

createfork (*project_id*)
Forks a project into the user namespace of the authenticated user.

Parameters **project_id** – Project ID to fork

Returns True if succeed

createforkrelation (*project_id*, *from_project_id*)
Create a fork relation.

This DO NOT create a fork but only adds a link as fork the relation between 2 repositories

Parameters

- **project_id** – project id
- **from_project_id** – from id

Returns true if success

creategroup (*name*, *path*, ***kwargs*)
Creates a new group

Parameters

- **name** – The name of the group
- **path** – The path for the group
- **kwargs** – Any param the the Gitlab API supports

Returns dict of the new group

createissue(*project_id*, *title*, ***kwargs*)
Create a new issue

Parameters

- **project_id** – project id
- **title** – title of the issue

Returns dict with the issue created

createissuewallnote(*project_id*, *issue_id*, *content*)
Create a new note

Parameters

- **project_id** – Project ID
- **issue_id** – Issue ID
- **content** – Contents

Returns Json or False

createlabel(*project_id*, *name*, *color*)
Creates a new label for given repository with given name and color.

Parameters

- **project_id** – The ID of a project
- **name** – The name of the label
- **color** – Color of the label given in 6-digit hex notation with leading '#' sign (e.g. #FFAABB)

Returns

createmergerequest(*project_id*, *sourcebranch*, *targetbranch*, *title*, *target_project_id=None*, *assignee_id=None*)
Create a new merge request.

Parameters

- **project_id** – ID of the project originating the merge request
- **sourcebranch** – name of the branch to merge from
- **targetbranch** – name of the branch to merge to
- **title** – Title of the merge request
- **assignee_id** – Assignee user ID

Returns dict of the new merge request

createmergerequestewallnote(*project_id*, *merge_request_id*, *content*)
Create a new note

Parameters

- **project_id** – Project ID

- **merge_request_id** – Merger Request ID
- **content** – Content

Returns Json or False

createmilestone(*project_id*, *title*, ***kwargs*)
Create a new milestone

Parameters

- **project_id** – project id
- **title** – title
- **description** – description
- **due_date** – due date
- **sudo** – do the request as another user

Returns dict of the new issue

createproject(*name*, ***kwargs*)
Creates a new project owned by the authenticated user.

Parameters

- **name** – new project name
- **path** – custom repository name for new project. By default generated based on name
- **namespace_id** – namespace for the new project (defaults to user)
- **description** – short project description
- **issues_enabled** –
- **merge_requests_enabled** –
- **wiki_enabled** –
- **snippets_enabled** –
- **public** – if true same as setting visibility_level = 20
- **visibility_level** –
- **sudo** –
- **import_url** –

Returns

createprojectuser(*user_id*, *name*, ***kwargs*)
Creates a new project owned by the specified user. Available only for admins.

Parameters

- **user_id** – user_id of owner
- **name** – new project name
- **description** – short project description
- **default_branch** – 'master' by default
- **issues_enabled** –
- **merge_requests_enabled** –

- **wiki_enabled** –
- **snippets_enabled** –
- **public** – if true same as setting visibility_level = 20
- **visibility_level** –
- **import_url** –
- **sudo** –

Returns

createrepositorytag(*project_id*, *tag_name*, *ref*, *message=None*)
Creates new tag in the repository that points to the supplied ref

Parameters

- **project_id** – project id
- **tag_name** – tag
- **ref** – sha1 of the commit or branch to tag
- **message** – message

Returns dict

createsnippet(*project_id*, *title*, *file_name*, *code*, *visibility_level=0*)
Creates an snippet

Parameters

- **project_id** – project id to create the snippet under
- **title** – title of the snippet
- **file_name** – filename for the snippet
- **code** – content of the snippet
- **visibility_level** – snippets can be either private (0), internal(10) or public(20)

Returns True if correct, false if failed

createsnippetewallnote(*project_id*, *snippet_id*, *content*)
Create a new note

Parameters

- **project_id** – Project ID
- **snippet_id** – Snippet ID
- **content** – Content

Returns Json or False

createuser(*name*, *username*, *password*, *email*, ***kwargs*)
Create a user

Parameters

- **name** – Obligatory
- **username** – Obligatory
- **password** – Obligatory

- **email** – Obligatory
- **kwargs** – Any param the the Gitlab API supports

Returns True if the user was created,false if it wasn't(already exists)

currentuser()
Returns the current user parameters. The current user is linked to the secret token

Returns a list with the current user properties

delete(*uri*, *default_response=None*)
Call DELETE on the Gitlab server

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.delete('/users/5')
```

Parameters

- **uri** – String with the URI you wish to delete
- **default_response** – Return value if JSONDecodeError

Returns Dictionary containing response data

Raise HttpError: If invalid response returned

delete_project(*id*)
Delete a project from the Gitlab server

Gitlab currently returns a Boolean True if the deleted and as such we return an empty Dictionary

Parameters **id** – The ID of the project or NAMESPACE/PROJECT_NAME

Returns Dictionary

Raise HttpError: If invalid response returned

delete_repository_tag(*project_id*, *tag_name*)
Deletes a tag of a repository with given name.

Parameters

- **project_id** – The ID of a project
- **tag_name** – The name of a tag

Returns Dictionary containing delete tag

Raise HttpError: If invalid response returned

delete_user(*user*)
Deletes a user. Available only for administrators. This is an idempotent function, calling this function for a non-existent user id still returns a status code 200 OK. The JSON response differs if the user was actually deleted or not. In the former the user is returned and in the latter not.

Parameters **user** – The ID of the user

Returns Empty Dict

Raise HttpError: If invalid response returned

deletebranch(*project_id*, *branch*)
Delete branch by name

Parameters

- **project_id** – The ID of a project
- **branch** – The name of the branch

Returns True if success, False if not

deletedeploykey(*project_id*, *key_id*)
Delete a deploy key from a project

Parameters

- **project_id** – project id
- **key_id** – key id to delete

Returns true if success, false if not

deletefile(*project_id*, *file_path*, *branch_name*, *commit_message*)
Deletes existing file in the repository

Parameters

- **project_id** – project id
- **file_path** – Full path to new file. Ex. lib/class.rb
- **branch_name** – The name of branch
- **commit_message** – Commit message

Returns true if success, false if not

deletegitlabciservice(*project_id*, *token*, *project_url*)
Delete GitLab CI service settings

Parameters

- **project_id** – Project ID
- **token** – Token
- **project_url** – Project URL

Returns true if success, false if not

deletegroup(*group_id*)
Deletes an group by ID

Parameters **group_id** – id of the group to delete

Returns True if it deleted, False if it couldn't. False could happen for several reasons, but there isn't a good way of differentiating them

deletegroupmember(*group_id*, *user_id*)
Delete a group member

Parameters

- **group_id** – group id to remove the member from
- **user_id** – user id

Returns always true

deletelabel(*project_id*, *name*)
Deletes a label given by its name.

Parameters

- **project_id** – The ID of a project
- **name** – The name of the label

Returns True if succeed

deleteldapgrouplink(*group_id*, *cn*, *provider=None*)
Deletes a LDAP group link (for a specific LDAP provider if given)

Parameters

- **group_id** – The ID of a group
- **cn** – The CN of a LDAP group
- **provider** – Name of a LDAP provider

Returns True if success

deleteproject(**args*, ***kwargs*)
Delete a project

> **Warning:** Warning this is being deprecated please use *gitlab.Gitlab.delete_project()*

Parameters **project_id** (*int*) – project id

Returns always true

deleteprojecthook(*project_id*, *hook_id*)
Delete a project hook

Parameters

- **project_id** – project id
- **hook_id** – hook id

Returns True if success

deleteprojectmember(*project_id*, *user_id*)
Delete a project member

Parameters

- **project_id** – project id
- **user_id** – user id

Returns always true

deletesnippet(*project_id*, *snippet_id*)
Deletes a given snippet

Parameters

- **project_id** – project_id
- **snippet_id** – snippet id

Returns True if success

deletesshkey(*key_id*)
Deletes an sshkey for the current user identified by id

Parameters **key_id** – the id of the key

Returns False if it didn't delete it, True if it was deleted

deletesystemhook(*hook_id*)
Delete a project hook

Parameters **hook_id** – hook id

Returns True if success

deleteuser(**args*, ***kwargs*)
Deletes a user. Available only for administrators. This is an idempotent function, calling this function for a non-existent user id still returns a status code 200 OK. The JSON response differs if the user was actually deleted or not. In the former the user is returned and in the latter not.

Warning: Warning this is being deprecated please use *gitlab.Gitlab.delete_user()*

Parameters **user_id** – The ID of the user

Returns True if it deleted, False if it couldn't

editgroupmember(*group_id*, *user_id*, *access_level*)
Edit user access level in a group

Parameters

- **group_id** – group id
- **user_id** – user id
- **access_level** – access level, see gitlab help to know more

Returns True if success

editissue(*project_id*, *issue_id*, ***kwargs*)
Edit an existing issue data

Parameters

- **project_id** – project id
- **issue_id** – issue id

Returns true if success

editlabel(*project_id*, *name*, *new_name=None*, *color=None*)
Updates an existing label with new name or now color. At least one parameter is required, to update the label.

Parameters

- **project_id** – The ID of a project
- **name** – The name of the label

Returns True if succeed

editmilestone(*project_id*, *milestone_id*, ***kwargs*)
Edit an existing milestone

Parameters

- **project_id** – project id

- **milestone_id** – milestone id
- **title** – title
- **description** – description
- **due_date** – due date
- **state_event** – state
- **sudo** – do the request as another user

Returns dict with the modified milestone

editproject(*project_id*, ***kwargs*)
Edit an existing project.

Parameters

- **name** – new project name
- **path** – custom repository name for new project. By default generated based on name
- **default_branch** – they default branch
- **description** – short project description
- **issues_enabled** –
- **merge_requests_enabled** –
- **wiki_enabled** –
- **snippets_enabled** –
- **public** – if true same as setting visibility_level = 20
- **visibility_level** –

Returns

editprojecthook(*project_id*, *hook_id*, *url*, *push=False*, *issues=False*, *merge_requests=False*, *tag_push=False*)
edit an existing hook from a project

Parameters

- **id** – project id
- **hook_id** – hook id
- **url** – the new url

Returns True if success

editprojectmember(*project_id*, *user_id*, *access_level*)
Edit a project member

Parameters

- **project_id** – project id
- **user_id** – user id
- **access_level** – access level

Returns True if success

edituser(*user_id*, ***kwargs*)
Edits an user data.

Parameters

- **user_id** – id of the user to change
- **kwargs** – Any param the the Gitlab API supports

Returns Dict of the user

enable_deploy_key (*project*, *key_id*)
Enables a deploy key for a project.

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.enable_deploy_key(15, 5)
```

Parameters

- **project** – The ID or URL-encoded path of the project owned by the authenticated user
- **key_id** – The ID of the deploy key

Returns A dictionary containing deploy key details

Raise HttpError: If invalid response returned

get (*uri*, *default_response=None*, ***kwargs*)
Call GET on the Gitlab server

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.get('/users/5')
```

Parameters

- **uri** – String with the URI for the endpoint to GET from
- **default_response** – Return value if JSONDecodeError
- **kwargs** – Key word arguments to use as GET arguments

Returns Dictionary containing response data

Raise HttpError: If invalid response returned

get_all_deploy_keys ()
Get a list of all deploy keys across all projects of the GitLab instance. This endpoint requires admin access.

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.get_all_deploy_keys()
```

Returns List of Dictionaries containing all deploy keys

Raise HttpError: If invalid response returned

get_project (*project*)
Get info for a project identified by id or namespace/project_name

Parameters **project** – The ID or URL-encoded path of the project

Returns Dictionary containing the Project

Raise HttpError: If invalid response returned

get_users (*search=None*, *page=1*, *per_page=20*, ***kwargs*)
Returns a list of users from the Gitlab server

Parameters

- **search** – Optional search query
- **page** – Page number (default: 1)
- **per_page** – Number of items to list per page (default: 20, max: 100)

Returns List of Dictionaries containing users

Raise HttpError if invalid response returned

getall (*fn*, *page=None*, **args*, ***kwargs*)
Auto-iterate over the paginated results of various methods of the API. Pass the GitLabAPI method as the first argument, followed by the other parameters as normal. Include *page* to determine first page to poll. Remaining kwargs are passed on to the called method, including *per_page*.

Parameters

- **fn** – Actual method to call
- **page** – Optional, page number to start at, defaults to 1
- **args** – Positional arguments to actual method
- **kwargs** – Keyword arguments to actual method

Returns Yields each item in the result until exhausted, and then implicit StopIteration; or no elements if error

getbranch (*project_id*, *branch*)
List one branch from a project

Parameters

- **project_id** – project id
- **branch** – branch id

Returns the branch

getbranches (*project_id*)
List all the branches from a project

Parameters **project_id** – project id

Returns the branches

getcontributors (*project_id*, *page=1*, *per_page=20*)
Get repository contributors list

Parameters

- **project_id** – The ID of a project
- **page** – Page number
- **per_page** – Records per page

Returns list of contributors or False

getdeploykey (*project_id*, *key_id*)
Get a single key.

Parameters

- **project_id** – project id
- **key_id** – key id

Returns the key in a dict if success, false if not

getdeploykeys(*project_id*)
Get a list of a project's deploy keys.

Parameters **project_id** – project id

Returns the keys in a dictionary if success, false if not

getfile(*project_id*, *file_path*, *ref*)
Allows you to receive information about file in repository like name, size, content. Note that file content is Base64 encoded.

Parameters

- **project_id** – project_id
- **file_path** – Full path to file. Ex. lib/class.rb
- **ref** – The name of branch, tag or commit

Returns

getfilearchive(*project_id*, *filepath=None*)
Get an archive of the repository

Parameters

- **project_id** – project id
- **filepath** – path to save the file to

Returns True if the file was saved to the filepath

getgroupmembers(*group_id*, *page=1*, *per_page=20*)
Lists the members of a given group id

Parameters

- **group_id** – the group id
- **page** – which page to return (default is 1)
- **per_page** – number of items to return per page (default is 20)

Returns the group's members

getgroups(*group_id=None*, *page=1*, *per_page=20*)
Retrieve group information

Parameters

- **group_id** – Specify a group. Otherwise, all groups are returned
- **page** – Page Number
- **per_page** – Records Per Page

Returns list of groups

getissues(*page=1*, *per_page=20*)
Return a global list of issues for your user.

Parameters

- **page** – Page number
- **per_page** – Records per page

Returns list of issues

getissuewallnote(*project_id*, *issue_id*, *note_id*)
Get one note from the wall of the issue

Parameters

- **project_id** – Project ID
- **issue_id** – Issue ID
- **note_id** – Note ID

Returns Json or False

getissuewallnotes(*project_id*, *issue_id*, *page=1*, *per_page=20*)
Get the notes from the wall of a issue

Parameters

- **project_id** – Project ID
- **issue_id** – Issue ID
- **page** – Page Number
- **per_page** – Records per page

getlabels(*project_id*)
Get all labels for given project.

Parameters **project_id** – The ID of a project

Returns list of the labels

getmergerequest(*project_id*, *mergerequest_id*)
Get information about a specific merge request.

Parameters

- **project_id** – ID of the project
- **mergerequest_id** – ID of the merge request

Returns dict of the merge request

getmergerequestchanges(*project_id*, *mergerequest_id*)
Get changes of a merge request.

Parameters

- **project_id** – ID of the project
- **mergerequest_id** – ID of the merge request

Returns information about the merge request including files and changes

getmergerequestcomments(*project_id*, *mergerequest_id*, *page=1*, *per_page=20*)
Get comments of a merge request.

Parameters

- **project_id** – ID of the project

- **mergerequest_id** – ID of the merge request
- **page** – Page number
- **per_page** – Records per page

Returns list of the comments

getmergerequests(*project_id*, *page=1*, *per_page=20*, *state=None*)
Get all the merge requests for a project.

Parameters

- **project_id** – ID of the project to retrieve merge requests for
- **page** – Page Number
- **per_page** – Records per page
- **state** – Passes merge request state to filter them by it

Returns list with all the merge requests

getmergerequestwallnote(*project_id*, *merge_request_id*, *note_id*)
Get one note from the wall of the merge request

Parameters

- **project_id** – Project ID
- **merge_request_id** – Merger Request ID
- **note_id** – Note ID

Returns Json or False

getmergerequestwallnotes(*project_id*, *merge_request_id*, *page=1*, *per_page=20*)
Get the notes from the wall of a merge request

Parameters

- **project_id** – Project ID
- **merge_request_id** – Merger Request ID
- **page** – Page number
- **per_page** – Records per page

Returns Json or False

getmilestone(*project_id*, *milestone_id*)
Get an specific milestone

Parameters

- **project_id** – project id
- **milestone_id** – milestone id

Returns dict with the new milestone

getmilestoneissues(*project_id*, *milestone_id*, *page=1*, *per_page=20*)
Get the issues associated with a milestone

Parameters

- **project_id** – project id
- **milestone_id** – milestone id

Returns list of issues

getmilestones(*project_id*, *page=1*, *per_page=20*)
Get the milestones for a project

Parameters

- **project_id** – project id
- **page** – Page number
- **per_page** – Records per page

Returns the milestones

getnamespaces(*search=None*, *page=1*, *per_page=20*)
Return a namespace list

Parameters

- **search** – Optional search query
- **page** – Which page to return (default is 1)
- **per_page** – Number of items to return per page (default is 20)

Returns returns a list of namespaces, false if there is an error

getproject(*project_id*)
Get info for a project identified by id or namespace/project_name

Parameters **project_id** – id or namespace/project_name of the project

Returns False if not found, a dictionary if found

getprojectevents(*project_id*, *page=1*, *per_page=20*)
Get the project identified by id, events(commits)

Parameters **project_id** – id of the project

Returns False if no project with that id, a dictionary with the events if found

getprojecthook(*project_id*, *hook_id*)
Get a particular hook from a project

Parameters

- **project_id** – project id
- **hook_id** – hook id

Returns the hook

getprojecthooks(*project_id*, *page=1*, *per_page=20*)
Get all the hooks from a project

Parameters

- **project_id** – project id
- **page** – Page number
- **per_page** – Records per page

Returns the hooks

getprojectissue(*project_id*, *issue_id*)
Get an specific issue id from a project

Parameters

- **project_id** – project id
- **issue_id** – issue id

Returns the issue

getprojectissues(*project_id*, *page=1*, *per_page=20*, ***kwargs*)
Return a list of issues for project id.

Param project_id: The id for the project.

Parameters

- **page** – Page number
- **per_page** – Records per page
- **kwargs** – Extra data to send

Returns list of issues

getprojectmembers(*project_id*, *query=None*, *page=1*, *per_page=20*)
Lists the members of a given project id

Parameters

- **project_id** – the project id
- **query** – Optional search query
- **page** – Which page to return (default is 1)
- **per_page** – Number of items to return per page (default is 20)

Returns the projects members, false if there is an error

getprojectsowned(*page=1*, *per_page=20*)
Returns a dictionary of all the projects for the current user

Returns list with the repo name, description, last activity, web url, ssh url, owner and if its public

getrawblob(*project_id*, *sha1*)
Get the raw file contents for a blob by blob SHA.

Parameters

- **project_id** – The ID of a project
- **sha1** – the commit sha

Returns raw blob

getrawfile(*project_id*, *sha1*, *filepath*)
Get the raw file contents for a file by commit SHA and path.

Parameters

- **project_id** – The ID of a project
- **sha1** – The commit or branch name
- **filepath** – The path the file

Returns raw file contents

getrepositories(*project_id*, *page=1*, *per_page=20*)
Gets all repositories for a project id

Parameters

- **project_id** – project id
- **page** – Page number
- **per_page** – Records per page

Returns list of repos

getrepositorybranch(*project_id*, *branch*)
Get a single project repository branch.

Parameters

- **project_id** – project id
- **branch** – branch

Returns dict of the branch

getrepositorycommit(*project_id*, *sha1*)
Get a specific commit identified by the commit hash or name of a branch or tag.

Parameters

- **project_id** – The ID of a project
- **sha1** – The commit hash or name of a repository branch or tag

Returns dict of commit

getrepositorycommitdiff(*project_id*, *sha1*)
Get the diff of a commit in a project

Parameters

- **project_id** – The ID of a project
- **sha1** – The name of a repository branch or tag or if not given the default branch

Returns dict with the diff

getrepositorycommits(*project_id*, *ref_name=None*, *page=1*, *per_page=20*)
Get a list of repository commits in a project.

Parameters

- **project_id** – The ID of a project
- **ref_name** – The name of a repository branch or tag or if not given the default branch
- **page** – Page number
- **per_page** – Records per page

Returns list of commits

getrepositorytags(*project_id*, *page=1*, *per_page=20*)
Get a list of repository tags from a project, sorted by name in reverse alphabetical order.

Parameters

- **project_id** – project id
- **page** – Page number
- **per_page** – Records per page

Returns list with all the tags

getrepositorytree (*project_id*, ***kwargs*)
Get a list of repository files and directories in a project.

Parameters

- **project_id** – The ID of a project
- **path** – The path inside repository. Used to get contend of subdirectories
- **ref_name** – The name of a repository branch or tag or if not given the default branch

Returns dict with the tree

getsnippet (*project_id*, *snippet_id*)
Get one snippet from a project

Parameters

- **project_id** – project id to get the snippet from
- **snippet_id** – snippet id

Returns dictionary

getsnippetcontent (*project_id*, *snippet_id*)
Get raw content of a given snippet

Parameters

- **project_id** – project_id for the snippet
- **snippet_id** – snippet id

Returns the content of the snippet

getsnippets (*project_id*, *page=1*, *per_page=20*)
Get all the snippets of the project identified by project_id

Parameters **project_id** – project id to get the snippets from

Returns list of dictionaries

getsnippetwallnote (*project_id*, *snippet_id*, *note_id*)
Get one note from the wall of the snippet

Parameters

- **project_id** – Project ID
- **snippet_id** – Snippet ID
- **note_id** – Note ID

Returns Json or False

getsnippetwallnotes (*project_id*, *snippet_id*, *page=1*, *per_page=20*)
Get the notes from the wall of a snippet

Parameters

- **project_id** – Project ID
- **snippet_id** – Snippet ID
- **page** – Page number
- **per_page** – Records per page

Returns Json or False

getsshkey(**args*, ***kwargs*)
Get a single ssh key identified by key_id

> **Warning:** Warning this is being deprecated please use *gitlab.Gitlab.keys()*

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.getsshkeys(1)
```

Parameters **key_id** – The ID of an SSH key

Returns Dictionary containing Key data

getsshkeys()
Gets all the ssh keys for the current user

Returns a dictionary with the lists

getsystemhooks(*page=1*, *per_page=20*)
Get all system hooks

Parameters

- **page** – Page number
- **per_page** – Records per page

Returns list of hooks

getuser(*user_id*)
Get info for a user identified by id

Parameters **user_id** – id of the user

Returns False if not found, a dictionary if found

getusers(**args*, ***kwargs*)
Returns a list of users from the Gitlab server

> **Warning:** Warning this is being deprecated please use *gitlab.Gitlab.get_users()*

Parameters

- **search** – Optional search query
- **page** – Page number (default: 1)
- **per_page** – Number of items to list per page (default: 20, max: 100)

Returns returns a dictionary of the users, false if there is an error

keys(*key_id*)
Get SSH key with user by ID of an SSH key. Note only administrators can lookup SSH key with user by ID of an SSH key.

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.keys(1)
```

Parameters **key_id** – The ID of an SSH key

Returns Dictionary containing Key data

login (*email=None*, *password=None*, *user=None*)
Logs the user in and setups the header with the private token

Parameters

- **email** – Gitlab user Email
- **user** – Gitlab username
- **password** – Gitlab user password

Returns True if login successful

Raise HttpError

Raise ValueError

moveproject (*group_id*, *project_id*)
Move a given project into a given group

Parameters

- **group_id** – ID of the destination group
- **project_id** – ID of the project to be moved

Returns dict of the updated project

post (*uri*, *default_response=None*, ***kwargs*)
Call POST on the Gitlab server

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> password = 'MyTestPassword1'
>>> email = 'example@example.com'
>>> data = {'name': 'test', 'username': 'test1', 'password': password, 'email
↪': email}
>>> gitlab.post('/users/5', **data)
```

Parameters

- **uri** – String with the URI for the endpoint to POST to
- **default_response** – Return value if JSONDecodeError
- **kwargs** – Key word arguments representing the data to use in the POST

Returns Dictionary containing response data

Raise HttpError: If invalid response returned

protectbranch (*project_id*, *branch*)
Protect a branch from changes

Parameters

- **project_id** – project id
- **branch** – branch id

Returns True if success

protectrepositorybranch (*project_id*, *branch*)
Protects a single project repository branch. This is an idempotent function, protecting an already protected repository branch still returns a 200 OK status code.

Parameters

- **project_id** – project id
- **branch** – branch to protect

Returns dict with the branch

removeforkrelation (*project_id*)
Remove an existing fork relation. this DO NOT remove the fork,only the relation between them

Parameters **project_id** – project id

Returns true if success

searchproject (*search*, *page=1*, *per_page=20*)
Search for projects by name which are accessible to the authenticated user

Parameters

- **search** – Query to search for
- **page** – Page number
- **per_page** – Records per page

Returns list of results

setgitlabciservice (*project_id*, *token*, *project_url*)
Set GitLab CI service for project

Parameters

- **project_id** – project id
- **token** – CI project token
- **project_url** – CI project url

Returns true if success, false if not

setsudo (*user=None*)
Set the subsequent API calls to the user provided

Parameters **user** – User id or username to change to, None to return to the logged user

Returns Nothing

shareproject (*project_id*, *group_id*, *group_access*)
Allow to share project with group.

Parameters

- **project_id** – The ID of a project
- **group_id** – The ID of a group
- **group_access** – Level of permissions for sharing

Returns True is success

success_or_raise (*response*, *default_response=None*)
Check if request was successful or raises an HttpError

Parameters

- **response** – Response Object to check
- **default_response** – Return value if JSONDecodeError

Returns dict Dictionary containing response data

Returns bool `False` on failure when exceptions are suppressed

Raises requests.exceptions.HTTPError – If invalid response returned

testsystemhook (*hook_id*)
Test a system hook

Parameters hook_id – hook id

Returns list of hooks

unprotectbranch (*project_id*, *branch*)
Stop protecting a branch

Parameters

- **project_id** – project id
- **branch** – branch id

Returns true if success

unprotectrepositorybranch (*project_id*, *branch*)
Unprotects a single project repository branch. This is an idempotent function, unprotecting an already unprotected repository branch still returns a 200 OK status code.

Parameters

- **project_id** – project id
- **branch** – branch to unprotect

Returns dict with the branch

updatefile (*project_id*, *file_path*, *branch_name*, *content*, *commit_message*)
Updates an existing file in the repository

Parameters

- **project_id** – project id
- **file_path** – Full path to new file. Ex. lib/class.rb
- **branch_name** – The name of branch
- **content** – File content
- **commit_message** – Commit message

Returns true if success, false if not

updatemergerequest (*project_id*, *mergerequest_id*, ***kwargs*)
Update an existing merge request.

Parameters

- **project_id** – ID of the project originating the merge request
- **mergerequest_id** – ID of the merge request to update
- **sourcebranch** – name of the branch to merge from
- **targetbranch** – name of the branch to merge to

- **title** – Title of the merge request
- **assignee_id** – Assignee user ID
- **closed** – MR status. True = closed

Returns dict of the modified merge request

Base

class gitlab.base.**Base**(*host*, *token=None*, *oauth_token=None*, *verify_ssl=True*, *auth=None*, *timeout=None*, *suppress_http_error=True*)

Base class

On init we setup the token used for all the api calls and all the urls

Parameters

- **host** – host of gitlab
- **token** – token
- **verify_ssl** – Weather or not to verify the SSL cert
- **auth** – Authentication
- **timeout** – Timeout
- **suppress_http_error** – Use `False` to unsuppress `requests.exceptions.HTTPError` exceptions on failure

Returns None

delete(*uri*, *default_response=None*)

Call DELETE on the Gitlab server

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.delete('/users/5')
```

Parameters

- **uri** – String with the URI you wish to delete
- **default_response** – Return value if JSONDecodeError

Returns Dictionary containing response data

Raise HttpError: If invalid response returned

get(*uri*, *default_response=None*, ***kwargs*)

Call GET on the Gitlab server

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.get('/users/5')
```

Parameters

- **uri** – String with the URI for the endpoint to GET from
- **default_response** – Return value if JSONDecodeError

- **kwargs** – Key word arguments to use as GET arguments

Returns Dictionary containing response data

Raise HttpError: If invalid response returned

static getall (*fn*, *page=None*, **args*, ***kwargs*)

Auto-iterate over the paginated results of various methods of the API. Pass the GitLabAPI method as the first argument, followed by the other parameters as normal. Include *page* to determine first page to poll. Remaining kwargs are passed on to the called method, including *per_page*.

Parameters

- **fn** – Actual method to call
- **page** – Optional, page number to start at, defaults to 1
- **args** – Positional arguments to actual method
- **kwargs** – Keyword arguments to actual method

Returns Yields each item in the result until exhausted, and then implicit StopIteration; or no elements if error

post (*uri*, *default_response=None*, ***kwargs*)

Call POST on the Gitlab server

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> password = 'MyTestPassword1'
>>> email = 'example@example.com'
>>> data = {'name': 'test', 'username': 'test1', 'password': password, 'email
↪': email}
>>> gitlab.post('/users/5', **data)
```

Parameters

- **uri** – String with the URI for the endpoint to POST to
- **default_response** – Return value if JSONDecodeError
- **kwargs** – Key word arguments representing the data to use in the POST

Returns Dictionary containing response data

Raise HttpError: If invalid response returned

success_or_raise (*response*, *default_response=None*)

Check if request was successful or raises an HttpError

Parameters

- **response** – Response Object to check
- **default_response** – Return value if JSONDecodeError

Returns dict Dictionary containing response data

Returns bool `False` on failure when exceptions are suppressed

Raises **requests.exceptions.HTTPError** – If invalid response returned

Session

class gitlab.session.**Session**(*host*, *token=None*, *oauth_token=None*, *verify_ssl=True*, *auth=None*, *timeout=None*, *suppress_http_error=True*)

login(*email=None*, *password=None*, *user=None*)

Logs the user in and setups the header with the private token

Parameters

- **email** – Gitlab user Email
- **user** – Gitlab username
- **password** – Gitlab user password

Returns True if login successful

Raise HttpError

Raise ValueError

Keys

class gitlab.keys.**Keys**(*host*, *token=None*, *oauth_token=None*, *verify_ssl=True*, *auth=None*, *timeout=None*, *suppress_http_error=True*)

getsshkey(**args*, ***kwargs*)

Get a single ssh key identified by key_id

> **Warning:** Warning this is being deprecated please use *gitlab.Gitlab.keys()*

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.getsshkeys(1)
```

Parameters **key_id** – The ID of an SSH key

Returns Dictionary containing Key data

keys(*key_id*)

Get SSH key with user by ID of an SSH key. Note only administrators can lookup SSH key with user by ID of an SSH key.

```
>>> gitlab = Gitlab(host='http://localhost:10080', verify_ssl=False)
>>> gitlab.login(user='root', password='5iveL!fe')
>>> gitlab.keys(1)
```

Parameters **key_id** – The ID of an SSH key

Returns Dictionary containing Key data

CHAPTER 10

Indices and tables

- genindex
- search

Index

K

L

M

P

R

S

T

U