pyapi-gitlab Documentation

Release 0.2

Itxaka Serrano Garcia

November 30, 2013

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 2 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

Using sudo on the functions

From version 6, gitlab accepts a sudo parameter in order to execute an order as if you were another user. This has been implemente in pyapi-gitlab on the following functions:

getusers()

createuser()

edituser()

addsshkey()

addsshkeyuser()

getprojects()

createproject()

createprojectuser()

addprojectmember()

editprojectmember()

editprojecthook()

getissues()

getprojectissues()

createissue()

editissue()

createmilestone()

editmilestone()

adddeploykey()

getgroups()

getmergerequests()

createmergerequest()

updatemergerequest()

All you need to do is add a sudo="user" parameter when calling the function like this:

```
git.createuser("name", "username", "password", "email", sudo="admin")
```

CHAPTER 5

Pagination

The following functions now accept pagination:

getusers()

getprojects()

getprojectevents()

getissues()

getprojectissues()

getgroups()

getmergerequests()

You can pass 2 parameters: page= and per_page= to them in order to get a especific page or change the results per page:

```
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 6

Users

There are several functions to manage users

Create user:

```
git.createuser("name", "username", "password", "email")
```

Delete user:

```
git.deleteuser(user_id)
```

Edit user details:

```
git.edituser(user_id)
```

Get all the users:

```
print git.getusers()
```

Get the current user:

```
print git.currentuser()
```

Get the user SSH keys:

```
for key in git.getsshkeys():
    print key
```

Get one key for the current user, specified by the key ID:

```
print git.getsshkey(key_id)
```

Add a new SSH key:

```
git.addsshkey("key name", "actual key")
```

Add a new SSH key for a specified user, identified by ID:

```
addsshkeyuser(user_id, "key name", "actual key")
```

Delete a SSH key for the current user:

```
git.deletesshkey(key_id)
```

CHAPTER 7

Projects

Get all the projects:

```
project = git.getprojects()
for proj in project:
    print proj
```

Get one project, identified by ID:

```
git.getproject(project_id)
```

Get project events:

```
git.getprojectevents(project_id)
```

Create a new project

If you are using version 6 you can pass an extra "public" argument which makes the project public.

Please note that Gitlab 5 doesn't have this option and using it will probably end in a failure while creating the project:

```
git.createproject(name, description="", default_branch="",
                  issues_enabled=0, wall_enabled=0,
                  merge_requests_enabled=0, wiki_enabled=0,
                  snippets_enabled=0, public=0)
```

List project members:

```
git.listprojectmembers(project_id)
```

Add a member to a project, access_level can be master,developer,reporter or guest:

```
git.addprojectmember(project_id, member_id, access_level)
```

Edit a project member, access_level can be master,developer,reporter or guest:

```
git.editprojectmember(id_, user_id, access_level)
```

Delete a member from a project:

```
git.deleteprojectmember(project_id, member_id)
```

Get the project Readme, you have to pass the web_url that getproject() provides:

```
git.getreadme(proj['web_url'])
```

Move a project:

```
git.moveproject(groupID, projectID)
```

CHAPTER 8

Hooks

Get all the hooks:

```
git.getprojecthooks(project_id)
```

Get one hook, identified by ID:

```
git.getprojecthook(project_id, hook_id)
```

Edit one hook:

```
git.editprojecthook(id_, hook_id, url)
```

Add a hook to a project:

```
git.addprojecthook(project_id, url_hook)
```

Delete a hook from a project:

```
git.deleteprojecthook(project_id, hook_id)
```

CHAPTER 9

Branches

Get all the branches for a project:

```
git.listbranches(1)
```

Get a specific branch for a project:

```
git.listbranch(1, "master")
```

Protect a branch:

```
git.protectbranch(1, "master")
```

Unprotect a branch:

```
git.unprotectbranch(1, "master")
```

Create a relation between two projects (The usual "forked from xxxxx"):

```
git.createforkrelation(1, 3)
```

Remove fork relation:

```
git.removeforkrelation(1)
```

CHAPTER 10

Issues

Get all the issues:

```
get.getissues()
```

Get a project issues:

```
git.getprojectissues(1)
```

Get a specified issue from a project:

```
git.getprojectissue(1,1)
```

Create an issue:

```
git.createissue(1, "pedsdfdwsdne")
```

Edit an issue, you can pass state_event="close" to close it:

```
git.editissue(1,1, title="Changing title")
```

CHAPTER 11

Milestones

Get all the milestones:

```
git.getmilestones(1)
```

Get a specific milestone from a project:

```
git.getmilestone(1,1)
```

Create a new milestone:

```
git.createmilestone(1,"New milestone")
```

Edit a milestone, you can pass state_event="closed" to close it:

```
git.editmilestone(1,1,title="Change milestone title")
```

CHAPTER 12

Deploy Keys

Get all the deployed keys for a project:

```
git.listdeploykeys(id_)
```

Get one key for a project:

```
git.listdeploykey(id_, key_id)
```

Add a key to a project:

```
git.adddeploykey(id_, title, key)
```

Delete a key from a project:

```
git.deletedeploykey(id_, key_id)
```

CHAPTER 13

Groups

Create a group:

```
def creategroup(self, name, path):
```

Get a group. If none are specified returns all the groups:

```
def getgroups(self, id_=None):
```

CHAPTER 14

Merge support

Get all the merge requests for a project:

```
git.getmergerequests(projectID, page=None, per_page=None)
```

Get information about a specific merge request:

```
git.getmergerequest(projectID, mergeRequestID)
```

Create a new merge request:

```
git.reateMergeRequest(projectID, sourceBranch, targetBranch, title, assigneeID=None)
```

Update an existing merge request:

```
git.updatemergerequest(projectID, mergeRequestID, sourceBranch=None, targetBranch=None, title=None, a
```

Add a comment to a merge request:

```
git.addcommenttomergerequest(projectID, mergeRequestID, note)
```