python GitLab Management

Jun 21, 2020

Contents:

CHAPTER 1

Python Gitlab Management

Gitlab-management is python module that enables GitLab group configuration from code. By design it's intended to be setup to run on a schedule.

1.1 How to Use

GitLab-Management can be used from the command line or imported as a python module.

Feature wise this module only process labels (create/add). over time features will be added. please see the milestones, issue and merge requests at the home page.

For please refer to the docs.

1.1.1 Python Module

To install run `pip install gitlab-management`. from there you will be able to use and extend the module as you see fit.

1.1.2 CLI

Gitlab-management can be used via cli with `gitlab-management`, there is no requirement to prefix with `python3 -m`, as the command is registered on install. To view all available options, use switch `-h` or `--help`

1.1.3 Config File

The configuration file for this module (`config.yml`) is a yml formated file that is required to be in the directory that the command is ran from.

The layout of the yml file

```yaml
Group:
    Labels:
        -
            Group: Example1
            Name: Bug
            Description: "Items that are bugs or bug related"
            Color: "#FF0000"

        -
            Group:
                - Example1
                - Example2
            Name: Feature
            Description: "Items that are feature related"
            Color: "#00FF00"
```

`Group.Labels` is a `list` of `dict` for each label that is to be created add a new `dict` to the `list` under `Group.Labels`

`Group.Labels.#.Group` can be a single string which is the name of an existing group that the user has access to as maintainer. `Group.Labels.#.Group` can also be a `list` of group names that the label will be added to.

1.1.4 Docker container

further help can be found in the documentation.

The docker images have been desigend to run the module directly as a command, where you only have to specify the `CLI` arguments.

images are tagged for ease of idententification in accordance with the table below:

Tag	repo branch	When created
`{Version number}`	Git Tag	every git tag will create it's version
`dev`	development	when a merge to development branch occurs or a git tag is created on a branch not master.
`stable`	master	when a merge to master branch occurs

1.2 Issues, Feature Requests and Bugs

If an issue or bug is found within the package (i.e. exception), please create an issue ticket using the applicable issue template available at the time of ticket creation. If you would like to request a feature and are unable to contribute, please create an issue using the feature issue template.

1.3 Contributing

Contribution guide can be viewed in the repo.

1.4 Licence

The package licence can be viewed in the repo

CHAPTER 2

Help

For assistance in using this module please see the sections below.

2.1 Command Line Interface (CLI)

`gitlab-management` can be run from the command line. to enable functionality when running the command, use arguments. `-h` will display help.

`-h` displays the following

```
GitLab-Management Help
To run this module the following options are available. There is a short and long␣
↪version of each option.

Example: python3 gitlab-management -T {GitlabAuthToken}

-H    --Host      (Optional) GitLab host to connect to. include http(s)://. Default:␣
↪https://gitlab.com
-T    --Token     (Mandatory) GitLab Private token for authentication.
-l    --labels    Process configuration group.labels.

-v    --verbose   (Optional) Verbose command output.

-h    --Help      used to display this help text.
```

When running the module from the cli, exit codes are returned to denote what has occured. exit code `0` is always returned on success.

Please see cli class for more details.

2.1.1 See Also

- cli Class

2.2 Configuration

The configuration file for gitlab-management is called `config.yml`.

2.2.1 Config.yml

the config file type is yaml. for `gitlab-management` to process, it must be valid yaml.

Example config file layout

```
Group:
    Labels:
        -
            Group: "ExampleGroup1"
            Name: "Stage::Planning"
            Description: "example description for the label"
            Color: "#FF0000"

        -
            Group:
                - "ExampleGroup1"
                - "ExampleGroup2"
            Name: "Stage::RequireFeedback"
            Description: "example description for the label"
            Color: "#FF0000"
```

2.3 Docker

The module has been built into a docker container that can be found on docker hub. by design the container does not need a command to be specified at runtime, as the `ENTRYPOINT` has been set to `gitlab-management`. Therefore you only need to specify the arguments for `gitlab-management`. If you fail to specify any arguments in the `docker run` command, gitlab-management help will be displayed

2.3.1 Building

To Build, execute the following:

```
python3 setup.py sdist bdist_wheel
docker build . --no-cache -t nofusscomputing/gitlab-management:$(cat dist/version)-
↪alpine
```

2.3.2 Running

To run the docker image, execute to following:

```
WORKING_DIR=/Config && docker run -v $PWD:$WORKING_DIR -w $WORKING_DIR␣
↪nofusscomputing/gitlab-management:$(cat dist/version)-alpine {Arguments}
```

Tip: Note: substitute `{Arguments}` with the switches for gitlab-management, i.e. `-h` for help.

2.4 Labels

Labels can be managed via gitlab-management python module.

Within the config file config.yml, the labels are an array (`dict` in python) under path `Group.Labels`. The layout file must be a valid yaml.

> **Tip:** *Labels are cached to keep GitLab API queries as low as possible. A groups labels will only be requested from the GitLab API, `once!!`*

Example config file layout for labels

```
Group:
    Labels:
        -
            Group: "ExampleGroup1"
            Name: "Stage::Planning"
            Description: "example description for the label"
            Color: "#FF0000"

        -
            Group:
                - "ExampleGroup1"
                - "ExampleGroup2"
            Name: "Stage::RequireFeedback"
            Description: "example description for the label"
            Color: "#FF0000"
```

2.4.1 Keys

- **Group.Labels.#.Group**: can be a `string` for a single group or `array` of string if you wan't more than one group to have the label.
- **Group.Labels.#.Name**: The name that you want the label to be.
- **Group.Labels.#.Description**: The description you want attached to the label.
- **Group.Labels.#.Color**: a hex representation of the color. in html color notation.

In the above example, `ExampleGroup1` would get both labels (`Stage::Planning` and `Stage::RequireFeedback`), whereas `ExampleGroup2` will only get label `Stage::RequireFeedback`

2.4.2 See Also

- gitlab-management.ProcessConfigLabels() method
- gitlab-management.GetLabelByName() method
- **Feature** - Labels now cached (issue #8)

2.5 Unit Testing

Unit testing is done on all classes and methods. Unit testing has been broken down into the following categories

1. Unit

2. Function
3. Integration

Depending on what item is being tested will depend on where the testing code will go.

2.5.1 Unit

The Unit test is the most basic of all of the tests. It's intent is to check the following items:

1. Method exists `{Classname}.globals`
2. all Inputs
3. The return values
4. Exceptions
    - Unexpected
    - Expected exceptions are thrown

whilst unit testing, all other methods or classes that the tested method is calling are to be mocked.

2.5.2 Function

Function testing is intended to test interoperability between classes and methods, including any user input(s). Function testing is to be limited to required dependancies.

The only items that are to be mocked dueing function testing are any methods that require interaction outside of the tested class methods, including dependant modules. *i.e. a http request.*

2.5.3 Integration

Integration testing is designed to confirm the classes, dependent modules and any required external services work as they were designed. If all tests upto and including this stage of testing pass, it is assumed that the module is ready for release.

CHAPTER 3

gitlab_management

3.1 gitlab_management package

3.1.1 Submodules

gitlab_management.base module

class gitlab_management.base.**GitlabManagement**(*GitLab_URL: str*, *GitLab_PrivateToken: str*, *Authenticate: bool = True*)

Bases: object

No Fuss Computing's Gitlab Config Management python module.

Config

CreateGroupLabel(*Group: gitlab.v4.objects.Group*, *Name: str*, *Description: str*, *Color: str*) → bool

Create a group label within the specified group

Parameters

- **Group** (*gitlab.v4.objects.Group*) – A gitlab group Object
- **Name** (*str*) – The name of the label to create
- **Description** (*str*) – The description for the label. the description will be prefixed with [Managed Label] to denote this script created the label.
- **Color** (*str*) – the colour that the label shoud be in format '#RRGGBB'

Raises

- Exception – Generic catch all Exception is returned, however the exception will be printed to the console
- gitlab.GitlabHttpError – A http error occured, the output should denote what the issue is

- `gitlab.GitlabCreateError` – a check for '409: label exists', as no attempt should be made to create a label when the check has already been done.

> **Warning:** if any other status is returned, then an issue should be raised.

Returns if the string was successfully created, true is returned. false will be returned when there was an exception

Return type bool

See also:

***Dependent Methods*:** *GetLabelByName Output*

`DesiredOutputLevel`

`GetConfig``()` → bool

Read all of the *config.yml* config file to an object in this class.

Raises

- `yaml.YAMLError` – any error with the yml, will return the text output
- `Exception` – Generic catch all exception. if this exception occurs, please log an issue ticket.

Returns returns success/failure on reading the config and adding to the class object *self.Config*

Return type bool

See also:

***Dependent Methods*:** *Output*

Help config.yml

`GetGroupByName`(*GroupName: str*) → gitlab.v4.objects.Group

Find and return a group by name.

Workflow 1. Find all groups that the current user (the one authorized in this module) has maintainer access to 2. check if the group has been cached, if not cach the groups for subsequent use so that additional API calls don't have to be made. 3. create an empty Labels group in the group object so that if a request for group labels is made, this can be checked for *None* if they haven't been fetched before. 4. iterate through all returned groups until the group is found that matches the string.

Parameters **`GroupName`** (*str*) – description

Raises `Exception` – Generic catch all exception. if this method returns an exception please log an issue.

ToDo: add proper error checking and exception checking.

Returns

- *gitlab.v4.objects.Group* – The Group that matches the search string will be returned.
- *None* – Returned if nothing found

See also:

***Dependent Methods*:** *Output*

issue #8 Feature: Groups now cached

GetLabelByName(*Group: gitlab.v4.objects.Group*, *LabelName: str*) → gitlab.v4.objects.GroupLabel
Finds a label by human readable name and returns a *gitlab.v4.objects.GroupLabel* object.

Workflow: 1. Check if the labels for the group have been cached, if not cache them. 1. iterate through each label in the group until the *LabelName* matches. 2. return the label as a *gitlab.v4.objects.GroupLabel*

Parameters

- **Group** (*gitlab.v4.objects.Group*) – The group that is being searched.
- **LabelName** (*str*) – The group name to search for.

Raises `Exception` – Generic catch all exception. if this method returns an exception please log an issue.

Returns

- *gitlab.v4.objects.GroupLabel* – returns the object that has a name that matches what was searched for.
- *None* – Only returned if no group is found.

See also:

***Dependent Methods*:** ProcessConfigLabels *Output*

Help Labels

issue #8 Feature: Labels now cached

GitlabLoginAuthenticate(*URL: str*, *PrivateToken: str*, *Authenticate: bool = True*) → bool
Establish the Gitlab instance to connect to and authenticate.

Parameters

- **URL** (*str*) – The url of the gitlab instance to connect to.
- **PrivateToken** (*str*) – The private token of the user that will be used to authenticate against the gitlab instance.

Raises

- `gitlab.GitlabAuthenticationError` – Returns text output of the failed authentication issue that occured when attemping to authenticate.
- `Exception` – Generic catch all exception.

Returns Returns *bool* to denote success/failure of the connection and authentication.

Return type bool

See also:

***Dependent Methods*:** *Output*

GitlabObjectCache
Cache the objects fetched via the GitLab API

See also:

***Dependent Methods*:** Nil

issue #8 Feature: Groups now cached

GitlabSession

Output(*OutputLevel: gitlab_management.base.GitlabManagement.OutputSeverity*, *OutputMessage: str*) → None

Method to output commands to the console.

Parameters

- **OutputLevel** (GitlabManagement.OutputSeverity *(Enum)*) – The output leval that the message is categorised as.
- **OutputMessage** (*str*) – The text to output.

Raises Exception – None raised. # ToDo: do proper error and exception handling.

Returns This method does not require output as it is part of the error handling of the application.

Return type None

See also:

***Dependent Methods*:** Nil

class OutputSeverity

Bases: enum.Enum

An enumeration.

Alert = 1

Critical = 2

Debug = 7

Emergency = 0

Error = 3

Informational = 6

Notice = 5

Warning = 4

ProcessConfigLabels(*ConfigGroups: list*) → bool

Process the provided configuration labels.

load the labels from the *config* file and create for each group that has been specified for the label.

The array that is passed to the function is processed as follows.

1. iterates through list of labels
2. finds the group id that the label is for
3. confirms that label attributes are in the config file Group, Name, Description and color.
4. creates the label in each group that the label is intended for. *CreateGroupLabel()* does the check to see if it exists before creating it.

Parameters **ConfigGroups** (*list*) – The labels array from the config.yml file

Raises Exception – currently is a catch all exception. if this function returns an exception, an issue needs to be raised.

Returns returns bool denoting success/failure for the processing of the labels provided.

Return type bool

See also:

***Dependent Methods*:** *GetGroupByName GetLabelByName Output*

Help configuration Labels

issue #8 Feature: Labels now cached

_Config = None

_DesiredOutputLevel = None

_GitlabObjectCache = None
Cache the objects fetched via the GitLab API

_GitlabSession = None

get_detail(*ItemName: str*)

gitlab = <module 'gitlab' from '/home/docs/checkouts/readthedocs.org/user_builds/pytho

logging = <module 'logging' from '/home/docs/.pyenv/versions/3.7.3/lib/python3.7/loggi

traceback = <module 'traceback' from '/home/docs/.pyenv/versions/3.7.3/lib/python3.7/t

gitlab_management.cli module

gitlab_management.cli.**main**()

3.1.2 Module contents

https://gitlab.com/nofusscomputing/projects/python-gitlab-management

CHAPTER 4

Contribution Guide

4.1 Developer setup

our development workflow is designed to be easy to setup. For this to occur, the following requirements are needed on the developers machine:

- Docker
- sudo access or be part of the group that can execute `docker`
- text editor *(with new lines set to ‘‘n‘‘ not ‘‘rn‘‘ as you would see in windows)*

4.2 Module

Setup the environment to use the local module being developed.

> **Note:** all commands run from the root of the repository directory.

Commands to setup build environment:

```
pip3 install virtualenv

python3 -mvirtualenv $PWD

bin/python -m pip install --upgrade --no-cache-dir pip

bin/python -m pip install --upgrade --no-cache-dir -r requirements.txt

bin/python setup.py develop install

python3 ./buildinit.py
```

test the module being developed

```
bin/gitlab_management -h
```

Clean-up the environment

```
rm -Rf build bin docs/_build gitlab_management.egg-info lib include pyvenv.cfg
```

to build the module

```
python3 setup.py sdist bdist_wheel
```

> **Note:** you must increment the version in the build script (`buildinit.py`) prior to committing your final changes to the repo.

4.3 running builds

Prior to committing to the repo, test builds need to be conducted. we have designed this to replicate gitlabs CI. Each stage of the `.gitlab-ci.yml` can be run from the command line using the following docker command

```
docker run --privileged -w $PWD -v $PWD:$PWD -v /var/run/docker.sock:/var/run/docker.
↪sock gitlab/gitlab-runner:ubuntu-v13.0.0 exec docker ${Gitlab-CI Stage} -env "CI_
↪COMMIT_BRANCH=$(git rev-parse --abbrev-ref HEAD)" --docker-privileged --docker-
↪volumes '/var/run/docker.sock:/var/run/docker.sock'
```

> **tip:** substitute `{Gitlab-CI Stage}` with one of the available stages below before running this command

Tested and confirmed `.gitlab-ci.yml` tasks as working with the above command

- **Verify**
  - PyLint
- **Unit Testing**
  - Unit Test
  - Function Test
- **package**
  - Coverage - *Not Working*
  - gitlab-management_package
- **build**
  - *- not working*
- **test**
  - *- Only usable on GitLab*
  - *- Not Tested*
  - *- Not Tested*
  - *- Not Tested*
  - *- Not Tested*
- **validate**

  - - *Not Working* **Tip:** use this command to build/test the docs

```
cd {Repo Directory}

rm -Rf build bin docs/_build gitlab_management.egg-info lib include pyvenv.
↪cfg docs/module docs/includes dist/version gitlab_management/__init__.py

CI_PROJECT_DIR=/Repository && docker run -e CI_PROJECT_DIR=$CI_PROJECT_DIR -w
↪$CI_PROJECT_DIR -v $PWD:$CI_PROJECT_DIR readthedocs/build:latest bash test/
↪validation-build-docs.sh
```

- **release**
  - - *Only usable on GitLab*
- **publish**
  - - *Only usable on GitLab*

4.3.1 Version Changes

Every change, prior to being committed to the `development` branch, must have it's version incremented. In most cases only the `{newfeature}` number will need to be incremented.

if the version is less than `1.0.0` the version number layout is `0.{newfeature}.{bugfix}`

if the version number is more than `1.0.0` the version number layout is `{majorchange}.{newfeature}.{bugfix}`. `{majorchange}` should only be incremented if the end user would be required to make a manual adjustments to use the version.

To adjust the version, open `buildinit.py` and edit the `__version__` property at the top of the script.

Regardless of the version until the changes are pushed to the `development` branch, it will be adjusted to be a release candidate. the appended version information will be `.rc` with a formated date i.e. `.rcYYMMDDHHMMSS`.

4.4 Documentation

We use Sphinx for ducumentation. All documentation for the project lives in the `docs/` directory within the repo. Documentation can be written in either RestructureText `.rst` or MarkDown `.md` files. Documentation for the modules however, is written within the code using `NumPy` style `DocBook` code comments. These comments are then rendered to `.rst` via the `sphinx.ext.autodoc` extension.

All Documentation is to be written in english. Since it depends on what dialect of english you speak, spelling can be different. The documentation spelling for plain speak (i.e. a sentance of text) will be Australian english.

Any additional pages are to be written in `.md` and placed in the directory `docs/pages`.

Further examples can be viewed here.

To document a property/variable add a single line comment underneath i.e.

```
DemoProperty:str = 'Property Value'
"""A single line describing the purpose of the property"""
```

4.4.1 template class documentation

The following is the template for a class. *(delete sections that are not required)*

```
class ExampleClassToDocument:
"""The summary line for a class docstring should fit on one line.

If the class has public attributes, they are not to be documented here. any␣
↪parameters the class has as part of it's ``__init__`` should be documented in this␣
↪section.

Example
-------
It should be assumed that a person using this class would have a working knowldege of␣
↪python. However, if there are any intricacies that are outside of a simple string,␣
↪this section should be used to explain how to use.

Parameters
----------
MandatoryParameter : str
    [Single Line summary]

OptionalParameter : str, optional
    [Single Line summary]

Attention
---------
Don't add an Attributes section to a class.

See Also
--------
`Dependent Methods`
    List methods alphabetically here one per line and as a hyperlink to the method.
    `Output <#gitlab_management.base.GitlabManagement.Output>`_

"""
```

4.4.2 template method documentation

The following is the template for a method. *(delete sections that are not required)*

```
def ExampleMethod(MandatoryParameter:str, OptionalParameter:str=None):
"""[one line summary]

[detailed summary formated as paragraphs]

Example
-------
Examples can be given using either the ``Example`` or ``Examples``
sections. Sections support any reStructuredText formatting, including
literal blocks::

    $ python example_numpy.py


Section breaks are created with two blank lines. Section breaks are also
implicitly created anytime a new section starts. Section bodies *may* be
indented:
```

(continues on next page)

(continued from previous page)

```
Parameters
----------
MandatoryParameter : str
    [Single Line summary]

OptionalParameter : str, optional
    [Single Line summary]

    NOTE
    ----
    Notes and other tip boxes can be included in any section, just don't forget to␣
↪indent.

Returns
-------
bool
    True if successful, False otherwise.

See Also
--------
`Dependent Methods`
    List methods alphabetically here one per line and as a hyperlink to the method.
    `Output <#gitlab_management.base.GitlabManagement.Output>`_

"""
```

4.4.3 Building

Buiding of the docs is part of the *validation* stage of the build pipeline. The doc build, must complete without any warnings to be considered a pass.

Please build the docs locally before pushing to the repo, this will ensure that they are working befor being committed. After a local build the docs can be found in the repository directory under `docs/_build/html/index.html`

Clean-up the environment prior to building the documentation, with:

```
rm -Rf build bin docs/_build gitlab_management.egg-info lib include pyvenv.cfg
```

> **Tip:** The above command contains the setting of the environmental variable `CI_PROJECT_DIR`, this is the directory where the repository is to be mounted to within the container. *don't include a trailing ''/'' slash if you change this.* The variable `$PWD` is to mount the current directory, if you are not running this command from the repo root dir, please set this to the full path of the repo.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

Symbols

A

C

D

E

G

I

L

M

N

O

P

T

W