labelx Documentation

Release 2.3.2

Dalwar Hossain

Jan 28, 2023

Contents:

1 Overview

`labelx` is a label generator for GitLab project issues and merge requests. Often project owner or maintainer needs to create different issue labels for multiple projects and it takes time and it's a monotonous work. `labelx` can simplify that task of creating labels.

1.1 Features

- Show package information
- Create labels for GitLab projects
- Create badges for GitLab projects

2 Installation

Labelx (`labelx`) requires Python 3.6, 3.7 or 3.8. If you do not already have a Python environment configured on your computer, please see the Python (https://www.python.org) page for instructions on installing Python environment.

Note: if you are on Windows and want to install optional packages (e.g., scipy) then you will need to install a python distribution such as Anaconda (https://www.anaconda.com), Enthought Canopy (https://www.enthought.com/product/canopy) or Pyzo (https://www.pyzo.org). If you use one of these Python distributions, please refer to their online documentation.

Assuming that the default python environment is already configured on your computer and you intend to install `labelx` inside of it. To create and work with Python virtual environments, please follow instructions on venv (https://docs.python.org/3/library/venv.html) and virtual environments (http://docs.python-guide.org/en/latest/dev/virtualenvs/)

To start the installation process, please make sure the latest version of `pip` (python3 package manager) is installed. If `pip` is not installed, please refer to the Pip documentation (https://pip.pypa.io/en/stable/installing/) and install `pip` first.

2.1 Linux/macOS

Install the latest release of `labelx` with `pip`:

```
pip install labelx
```

To upgrade to a newer version use the `--upgrade` flag:

```
pip install --upgrade labelx
```

If system wide installation is not possible for permission reasons, use `--user` flag to install `labelx` for current user

```
pip install --user labelx
```

2.2 Windows

`labelx` should support `windows cmd` out of the box. But if you face any issues, please refer to the `hint` below -

---

Hint: Windows terminal (cmd/power shell) doesn't support all the unicode codecs and To get the best results - please use a terminal emulator like, cmder [Download Cmder (http://cmder.net/)] or ConEmu [Download ConEmu (https://conemu.github.io/)]. Please use <*xterm*> color scheme from *settings* menu, for the best visual representation of the program.

Considering `python3` is installed and `pip` is configured.

Open Cmder/ConEmu and Type:

```
pip install labelx
```

Or

```
python3 setup.py install
```

This command should install `labelx` with all the required dependencies.

2.3 Install from Github

Alternatively, `labelx` can be installed manually by downloading the current version from GitHub (https://github.com/dalwar23/labelx) or PyPI (https://pypi.org/project/labelx/). To install a downloaded versions, please unpack it in a preferred directory and run the following commands at the top level of the directory:

```
pip install .
```

or run the following:

```
python3 setup install
```

2.4 Dependencies

This package requires a configuration file in either `.yaml` or `yml` format. The look up priority for the configuration file is as following-

1. <user_home_directory>/.config/<package_name>/config.yaml (`Window/Linux/MacOS`)
2. <current_working_directory>/<package_name>/config.yaml (`Windows/Linux/MacOS`)
3. /etc/<package_name>/config.yaml (`Linux/MacOS`)

If `config.yaml` doesn't exists in one of these locations, the program will NOT run. So, to create the configuration file, please use -

Windows

Windows system by default doesn't allow creation of `.` prefixed directory from GUI, so use the following -

- Open *cmd* and change the directory to the `home` folder of the user
- Run `mkdir .config` (if the folder doesn't exist)
- Run `cd .config`
- Run `mkdir labelx`

Now that the `.` prefixed directory is created, use the GUI to add a file in `labelx` directory named `config.yaml`. Once the file is created, open the file and add the following lines according to your settings -

```
---
login:
  host: <gitlab_server>
  protocol: <https>/<http>
  token: <secret_access_token_from_gitlab_profile>
```

Linux/MacOS

- Open a terminal and `cd` into the home directory or any other directory form the above dependency list.
- Run `mkdir -p .config/labelx`
- Run `cd .config/labelx/`
- Run `nano config.yaml`
- Add the above lines into the file and save it

3 Tutorials

Tip: Always remember, when in doubt use `--help`.

3.1 Getting help

To see the options use `--help` flag after the command.

```
labelx --help
```

This should output the help information on screen and it looks something similar as below -

```
Usage: labelx [OPTIONS] COMMAND [ARGS]...

GitLab label creator control panel

Options:
  --debug    Turns on DEBUG mode.  [default: False]
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  create-badges  Create badges for project.
  create-labels  Create labels for issues and merge requests.
  pkg-info       Shows package information.
```

To checkout individual options for any command use `--help` flag.

```
labelx pkg-info --help
```

This command should show all the available `arguments` for `info` sub-command.

```
Usage: labelx pkg-info [OPTIONS]

  Prints information about the package

Options:
  --help  Show this message and exit.
```

3.2 Turning on debug

Sometimes program runs into ERROR and there are not enough data shown on screen to determine the cause of the ERROR. For getting `verbose` output of all the actions done by the program simply `turn on` the `debug` mode with `--debug` flag. By default it's turned off.

To turn on `debug` mode -

```
labelx --debug create-labels [OPTIONS] [ARGS]
```

Also you can turn on the `debug` mode at sub-command level by using `--debug` flag after the sub-command

```
labelx create-labels [OPTIONS] [ARGS] --debug
```

OR like this -

```
labelx create-labels --debug [OPTIONS] [ARGS]
```

3.3 Creating Labels

To create `default` labels, use the following command -

```
labelx create-labels -p [gitlab project id]
```

To create group labels use `-g` flag

```
labelx create-labels -g [gitlab group id]
```

Example

```
labelx create-labels -p 12345
```

This command should run the program with `preset` labels and create these labels in the project mentioned. Output should be something similar -

(output is from version 2.1.1)

```
+--------------------------------------------------+
|                     labelx                       |
+--------------------------------------------------+
| about: GitLab label/badge creator                |
| author: Dalwar Hossain (dalwar23@pm.me)          |
| version: 2.1.1                                   |
| license: GNU General Public License v3           |
| documentation: https://labelx.readthedocs.io/    |
+--------------------------------------------------+

[*] Initializing.....
[*] Please use 'labelx --help' to see all available options
------------------------------------ [labelx] ----------------------------------
↪-----
[$] Creating label - [Bug] ..... DONE
[$] Creating label - [Done] ..... DONE
[$] Creating label - [Feature Upgrade] ..... DONE
[$] Creating label - [Fixed] ..... DONE
[$] Creating label - [New Feature Request] ..... DONE
[$] Creating label - [On Hold] ..... DONE
[$] Creating label - [P1] ..... DONE
[$] Creating label - [P2] ..... DONE
[$] Creating label - [P3] ..... DONE
[$] Creating label - [Planned] ..... DONE
```

(continues on next page)

(continued from previous page)

```
[$] Creating label - [Source Code Refactoring] ..... DONE
[$] Creating label - [Testing] ..... DONE
[$] Creating label - [WIP] ..... DONE
[$] Creating label - [Won't Fix] ..... DONE
------------------------------------ Goodbye! ----------------------------------
↪-----
```

3.4 Using user defined labels

Note: When using user defined labels, the builtin labels will be ignored.

To create user defined labels, use `-f` or `--labels` option with an argument of a valid `yaml` file path that contains label information. E.g.

```
labelx create-labels -p 1234 -f ~/labels.yaml
```

OR

```
labelx create-labels -g 23 --labels ~/my/vey/nested/file/path/to/labels.yaml
```

Example labels.yaml file

```yaml
Test1:
  color: "#FF0000"
  description: A test label
  description_html: ''
  text_color: "#FFFFFF"
  subscribed: false
  priority: 0
  is_project_label: true
Test2:
  color: "#0033CC"
  description: Second test label
  description_html: ''
  text_color: "#FFFFFF"
  subscribed: false
  priority: 1
  is_project_label: true
Test3:
  color: "#AD4363"
  description: Third test label
  description_html: ''
  text_color: "#FFFFFF"
  subscribed: false
  priority:
  is_project_label: true
```

3.5 Creating Badges

To create `default` badges, use the following command -

```
labelx create-badges -p [gitlab project id]
```

To create `group` badges use `-g` flag with numeric group id.

```
labelx create-badges -g [gitlab group id]
```

Note: Group badges require `owner` permission for the group in context.

Example

```
labelx create-badges -p 1245
```

This command should run the program with `preset` badges and create these badges in the project mentioned. Output should be something similar -

(output is from version 2.1.1)

```
+--------------------------------------------------+
|                     labelx                       |
+--------------------------------------------------+
| about: GitLab label/badge creator                |
| author: Dalwar Hossain (dalwar23@pm.me)          |
| version: 2.1.1                                   |
| license: GNU General Public License v3           |
| documentation: https://labelx.readthedocs.io/    |
+--------------------------------------------------+

[*] Initializing.....
[*] Please use 'labelx --help' to see all available options
------------------------------------ [labelx] ----------------------------------
↪-----
[$] Creating badge - [license] ..... DONE
[$] Creating badge - [platform] ..... DONE
[$] Creating badge - [windows] ..... DONE
[$] Creating badge - [linux] ..... DONE
[$] Creating badge - [macos] ..... DONE
[$] Creating badge - [python] ..... DONE
[$] Creating badge - [style] ..... DONE
[$] Creating badge - [layout] ..... DONE
------------------------------------ Goodbye! ----------------------------------
↪-----
```

3.6 Using user defined badges

Note: When using user defined badges, the builtin badges will be ignored.

To create user defined badges, use `-f` or `--badges` option with an argument of a valid `yaml` file path that contains badge information. E.g.

```
labelx create-badges -p 143 -f ~/badges.yaml
```

OR

```
labelx create-badges -g 23 --badges ~/my/vey/nested/file/path/to/badges.yaml
```

Example badges.yaml file

```
---
license:
  link_url: "%{project_path}/LICENSE"
  image_url: "https://img.shields.io/badge/License-MIT-blue.svg?style=flat-square"
  position: 0
platform:
  link_url: "%{project_path}"
  image_url: "https://img.shields.io/badge/Platform-cc3300?style=flat-square"
  position: 1
windows:
  link_url: "%{project_path}"
  image_url: "https://img.shields.io/badge/Windows-blue?style=flat-square&
↪logo=windows"
  position: 2
linux:
  link_url: "%{project_path}"
  image_url: "https://img.shields.io/badge/Linux-333?style=flat-square&logo=linux"
  position: 3
```

Important: `%{project_path}` will translated into the project's path by `labelx` for project specific badges only. For groups, It is recommended not to use GITLABPLACEHOLDERTOKENS like `%{project_path}` or `%{project_id}`.

---

4 API References

API references are not available at this moment. In future release, it will be available.

5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/dalwar23/labelx/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

Write Documentation

labelx could always use more documentation, whether as part of the official labelx docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dalwar23/labelx/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here’s how to set up *labelx* for local development.

1. Fork the *labelx* repo on GitHub.
2. Clone your fork locally

```
$ git clone git@github.com:your_name_here/labelx.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development

```
$ mkvirtualenv labelx
$ cd labelx/
$ python setup.py develop
```

4. Create a branch for local development

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you’re done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox

```
$ flake8 labelx tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7, 3.8, 3.9 and 3.10, and for PyPy.

6 History

6.1 2.3.1 [30.03.2022]

- Fix dependency issue in setup.py Issue #13 (https://github.com/dalwar23/labelx/issues/13)

6.2 2.3.0 [30.03.2022]

- Updated documentation
- Added group level support for labels and badges
- Updated tests
- Fixed spelling mistakes
- Minor console output changes

6.3 2.2.1 [28.03.2022]

- Add group level badges and labels
- Fix version upgrade bug
- Update documentation

6.4 2.1.1 [19.06.2020]

- Fixed bug with license badge
- Updated readme and docs

6.5 2.1.0 [19.06.2020]

- Fixed bug with link_url
- Changed look and feel

6.6 1.0.5 [19.06.2020]

- Create badges with default values
- Added tests
- Fixed formatting

6.7 1.0.4 [02.04.2020]

- Create labels with default values
- Show package information
- Added more tests
- Re-newed documentation

6.8 1.0.3 (unreleased) [30.03.2020]

- Show package information
- Added test cases

7 Indices and tables

- genindex
- modindex
- search