labelord Documentation

Release 0.5

Lenka Stejskalova

Dec 03, 2017

Contents:

CHAPTER 1

INTRO

This is a command-line application with Flask web interface. It allows to manage labels from GitHub issues. Goal of project is to make it easy to manage labels for multiple repositories and not to create same labels for all repositories.

Application can list all repositories or all labels of issues in given repository. Application can update or replace labels with labels from another source. Finally application can run a Flask web interface.

Application is *installed* as Python module. For examples how to run commands see *USAGE* or *TEST*.

Application need a GitHub API token for connection to GitHub. This token can be used to access the GitHub API and allows you to see some details of repositories. Manual for obtaining this token is in section *GitHub API token*. Web interface need a webhook for connection to GitHub. Manual for obtaining this webhook is in section *GitHub webhook secret*.

Part of application can be set by configuration file or options. Config file is writen in INI format. See more details in *Configuration file*. Details about options are in *Options*.

CHAPTER 2

INSTALL

Labelord application is install as Python module. Module can be downloaded from PyPi `https://test.pypi.org/project/labelord-stejsle1/` or from GitHub repository `https://github.com/stejsle1/labelord/releases` as latest release.

After you download it you can install it.

Move to folder with extract files. To make a module from file run:

```
python3 setup.py sdist
```

It will make a dist folder with distribution. To install module run:

```
python3 -m pip install dist/Labelord-Version.tar
```

Command will automatically install required modules.

Now the Labelord module is install and you can run application:

```
# For help type:
python3 -m labelord
# or
labelord
```

For running commands type (for more information ses *USAGE*):

```
python3 -m labelord [command options]
# or
labelord [command options]
```

For details about configuration see *CONFIG*.

CHAPTER 3

USAGE

If Labelord module is *installed* you can start to use it.

All commands can be run by typing:

```
python3 -m labelord [command options]
# or
labelord [command options]
```

All commands and its options are explained below.

3.1 List repos

This `command` print out all repositories according to given token.

```
list_repos
```

This code end with error code because no GitHub token was inserted.

If you want to set a configuration file, type:

```
list_repos -c YourConfigFile.cfg
# or
list_repos --config YourConfigFile.cfg
```

Token can be also insert via option:

```
list_repos -t YourToken
# or
list_repos --token YourToken
```

Warning: Be sure your token can't be copied or saved in some way by unauthorized person. Secure your confidential data.

3.2 List labels

This `command` print out all label from given repository.

```
list_labels MyNick/MyRepository
```

This code end with error code because no GitHub token was inserted.

If you want to set a configuration file, type:

```
list_labels -c YourConfigFile.cfg MyNick/MyRepository
# or
list_labels --config YourConfigFile.cfg MyNick/MyRepository
```

Token can be also insert via option:

```
list_labels -t YourToken MyNick/MyRepository
# or
list_labels --token YourToken MyNick/MyRepository
```

Warning: Be sure your token can't be copied or saved in some way by unauthorized person. Secure your confidential data.

3.3 Run

This `command` run a label changes. To run it you must decided if you want to *update* labels or *replace* labels.

Token can be provided same ways as in commands `list_repos` and `list_labels` - by config file and by option. To save space thid documentation will be working with configfile-way.

Warning: Be sure your token can't be copied or saved in some way by unauthorized person. Secure your confidential data.

3.3.1 Update

To update label means if your changing repository missing some label, then label will be inserted. If you have some labels which are not in template repository, this labels will be keeped in. Same labels are keep in.

To update labels run:

```
run update -c YourConfigFile.cfg
```

If repositories and labels or template repository are set in config file, then this command will make changes fine.

If repositories are not set in config file, you can set to run this changes to all repositories from command `list_repos` by typing option `--all_repos`:

```
run update -c YourConfigFile.cfg --all-repos
# or
run update -c YourConfigFile.cfg -a
```

If nor labels nor template repository are not set in config file, you can set template repository by option `--template-repo`:

```
run update -c YourConfigFile.cfg --template-repo YourRepo
# or
run update -c YourConfigFile.cfg -r
```

If you don't want to make change and run it to check what would be changed, run dry_run:

```
run update -c YourConfigFile.cfg --dry-run
# or
run update -c YourConfigFile.cfg -d
```

Command can be run in verbose or quiet mode:

```
run update -c YourConfigFile.cfg --verbose
# or
run update -c YourConfigFile.cfg -v

run update -c YourConfigFile.cfg --quiet
# or
run update -c YourConfigFile.cfg -q
```

3.3.2 Replace

To replace label means if your changing repository missing some label, then label will be inserted. If you have some labels which are not in template repository, this labels will be **deleted**. Same labels are keep in.

To replace labels run:

```
run replace -c YourConfigFile.cfg
```

If repositories and labels or template repository are set in config file, then this command will make changes fine.

If repositories are not set in config file, you can set to run this changes to all repositories from command `list_repos` by typing option `--all_repos`:

```
run replace -c YourConfigFile.cfg --all-repos
# or
run replace -c YourConfigFile.cfg -a
```

If nor labels nor template repository are not set in config file, you can set template repository by option `--template-repo`:

```
run replace -c YourConfigFile.cfg --template-repo YourRepo
# or
run replace -c YourConfigFile.cfg -r
```

If you don't want to make change and run it to check what would be changed, run dry_run:

```
run replace -c YourConfigFile.cfg --dry-run
# or
run replace -c YourConfigFile.cfg -d
```

Command can be run in verbose or quiet mode:

```
run replace -c YourConfigFile.cfg --verbose
# or
run replace -c YourConfigFile.cfg -v

run replace -c YourConfigFile.cfg --quiet
# or
run replace -c YourConfigFile.cfg -q
```

3.4 Run server

This `command` run a server - Flask web interface. This command need to has set an envitonment variable `LABELORD_CONFIG`. This variable contains config file name with configuration (token, webhook secret, ...).

To run a server, type:

```
run_server
```

In this case server will run on 127.0.0.1 on port 5000 (default values). If you want to run server on different host or port, run command:

```
run_server --host 146.13.306.124 --port 5001
# or
run_server -h 146.13.306.124 -p 5001
```

You can run server in debug mode:

```
run_server --debug
# or
run_server -d
```

This mode can be also triggered by environment variable `FLASK_DEBUG` with value `true`.

CHAPTER 4

CONFIG

Configuration can be set by *configuration file* or *options*.

4.1 Configuration file

General input to application is configuration file. This file is written in INI format. File is set by option *-c/–config*. Default value is *'./config.cfg'*.

File have to contain section *'github'*, where is placed *GitHub token* and *webhook secret*.

```
[github]
token = YourToken
webhook_secret = YourWebhook
```

Next acceptable sections are *'repos'*, *'labels'* and *'others'*. Section *'repos'* defines repositories which will be changed. In section will be list of repositories and flag *'on/off'* for making or not making changes on this repository. List of repositories can be empty.

```
[repos]
stejsle1/lab01 = on
stejsle1/labelord = on
stejsle1/wator = off
```

Section *'labels'* defines list of labels which will be insert in repository. Every line in section is a pair of name and color. List can be empty.

```
[labels]
Error = FF0000
Important = 112233
```

Section *'others'* defines a template repository. This means labels will be change to be same as in template repository.

```
[others]
template-repo = stejsle1/wator
```

4.1.1 GitHub API token

To obtain this token log in GitHub. Open a **'Personal setting'**, click to **'Developer settings'**. In section **'Personal access tokens'** click to **'Generate new token'**. It opens a new page with form. Fill a token description and mark **'public-repo'**. This option allows to make changes with repositories. Save this token to personal secret place.

> **Warning:** Never place this token to GitHub or another public place!

4.1.2 GitHub webhook secret

To be sure a request is really from GitHub there is a option to add a secret item into webhook. Then application reads a header *'X-Hub-Signature'* and checks if this secret was used to create request header. If do then request can be trusted.

To set webhook open your repository and click on **'Settings'**. On left menu click on **'Webhooks'**. On ritgh side click on **'Add webhook'**. On new page fill a form and hit **'Add webhook'**.

4.2 Options

For all settings in config file there is a option.

-c/–config [name]

Set config file.

-t/–token [token]

Set token.

-r/–template-repo [repo]

Set template repository (labels will be change to same ones as in this repository).

-a/–all-repos

Changes will be taken in all repos from request **'list_repos'**.

-d/–dry-run

Set 'dry run' (application don't make real changes).

-v/–verbose

Set verbose mode.

-q/–quiet

Set quiet mode (no output).

-h/–host [host]

Set hostname for web interface (default 127.0.0.1).

-p/–port [port]

Set port for web interface (default 5000).

-d/–debug

Set debug mode.

4.3 Environment variables

Last option to set configuration is by environment variables. There is a few variables which are supported.

GITHUB_TOKEN

Set token. Example: GITHUB_TOKEN='YourToken'

LABELORD_CONFIG

Set config file for web interface. Example: LABELORD_CONFIG='./config.cfg'

FLASK_DEBUG

Set debug mode for web interface. Exaplme: FLASK_DEBUG=true

CHAPTER 5

API

CHAPTER 6

TEST

Documentation can test some **labelord** function to see how they works.

For example it checks function *'convert_time(*string)''* for printing link to GitHub repository:

```
>>> convert_time("stejsle1/lab01")
'https://github.com/stejsle1/lab01'
```

More information about function *'convert_time(*string)''* is in *API*

Next example shows how to print messages. In case application in verbose mode add label `'Need vacation'` with color `'FFFFFF'` into repository `'stejsle1/myNewSummerRepo'`:

```
printextra(2, "stejsle1/myNewSummerRepo; Need vacation; FFFFFF", "ADD", 0)
```

it prints:

```
[ADD][SUC] stejsle1/myNewSummerRepo; Need vacation; FFFFFF
```

In case application in normal mode update label `'Unbreakable'` with color `'123456'` to color `'654321'` into repository `'stejsle1/myNewWinterRepo'` it prints nothing because application in normal mode prints **only errors** to *stderr* and **summary** to *stdout*.

So if same command ends with error `422 - Validation Failed`:

```
printextra(1, "stejsle1/myNewWinterRepo; Unbreakable; 654321; 422 - Validation Failed
↪", "UPD", 1)
```

it prints out:

```
ERROR: UPD; stejsle1/myNewWinterRepo; Unbreakable; 654321; 422 - Validation Failed
```

In normal mode module prints out a summary (with 2 errors) message:

```
>>> printextra(4+1, str(2) + ' error(s) in total, please check log above', '', 1)
SUMMARY: 2 error(s) in total, please check log above
```

Flawless case of summary report in verbose mode:

```
>>> printextra(4+2, str(3) + ' repo(s) updated successfully', '', 0)
[SUMMARY] 3 repo(s) updated successfully
```

In quiet mode application prints nothing, neither summary message.