Labelord Documentation

Release 0.5

Jan Horacek

Dec 01, 2017

Contents:

This application is simple tool for working with GitHub labels. You can easily create new label, delete label, edit some labels and what is the best, replicated all labels through repositories!

This project was develop as homework for CTU FIT for lecture MI-PYT.

Whole project is programed in python and using GitHub API. This application works only with GitHub!

CHAPTER 1

How to install

Application is standard python library. You can download directly from my GitHub or you can use PyPi.

You should install application to new virtual environment. If you install library thought pip, all requirements will be installed. Otherwise you must install all requirements manualy. Requirements are located in `requirements.txt` file

1.1 Install from PyPi

Library is located at test PyPi. You can easily install with one command:

```
python -m pip install --extra-index-url https://test.pypi.org/pypi labelord_horacj10
```

1.2 Get source from GitHub

You can download source code directly from GitHub. You can find repository at my GitHub

CHAPTER 2

Basic usage

Labelord have two different parts. First part is console application. With this console you can manage labels at repositories. you can delete labels, create new labels or edit some labels. You can specify this in config file or use console arguments. But all the changes you must do manually.

Second part is web server. This server is used for accepting GitHub webhooks and based on this webhooks replicate labels between two or more repositories.

2.1 Console

The main entrypoint for console application is command **labelord**. When you run only **labelord** without any parameters, you will get help.

Note: You must have activated virtual enviroment and after that just type `python -m labelord`

Main command could have some parameters which are same for every command:

- **-c / –config [path]** - with this parameter you could specify path to config file. Default is config.cfg in current directory.
- **-t / –token [token]** - with this parameter you could specify GitHub API token

Token must be specified (in config file or by token parameter). Next you could chose from 3 commands.

2.1.1 list_repos

This command is used for get list of repositories, which you enable. This command will check config file and read all enabled repositories. After that check if you have access for this repositories and print all enabled repositories.

Hint: `python -m labelord -t my_secret_token list_repos` will show list of repositories

2.1.2 list_labels

This command is used for get list of all labels in one repository. For this command you must specify target repository. You can specify this as first argument of command. If you don't have permission for this repository or this repository don't exist, program will show error message. Repository is specified in format **user_name/repository_name**.

Hint: `python -m labelord -t my_secret_token list_labels dummyUser/HelloWorld` will show list of label in HelloWorld repository

2.1.3 run

This is main command for labels editing. With this command you can create labels, delete labels or edit some labels. This command accept many optional arguments which defining behaviour of command:

- **-t / –template-repo [repo]** - with this argument you can specify repository, which will be used as template. All definitions of labels will be get from this repo. Just read all repos in this repo.
- **-a / –all-repos** - if this flag is set, command will ignore settings in config.cfg in section repos and editing labels will be done in all user repositories
- **-q / –quiet** - if this flag is set, program print nothing to console
- **-v / –verbose** - if this flag is set, program will print more informations about run. (done commands, errors, etc.)
- **-d / –dry-run** - if this flag is set, all command wil not affect any repository. Just print some informations about commands.

Next you must specify mode. This mode is first parameter of command. You can chose from two:

- **update** - if you select this mode, all new labels will be created, labels with different color will be changed and all other labels will not change
- **replace** - if you select this mode, all new labels will be created, labels with different color will be changed and **all other labels that are not in template will be deleted!**

After run this command, summary output will be printed in command line.

Hint: `python -m labelord -t my_secret_token run -a -v update` will update labels in all my repositories based on config file and some debug info will be printed to console

---

Example of verbose output

```
[ADD][SUC] Wilson194/hello_world; label1; FFAA00
[ADD][SUC] Wilson194/hello_world; label2; CCAAFF
[ADD][SUC] Wilson194/hello_world; label3; 00BBCC
[DEL][SUC] Wilson194/hello_world; bug; fc2929
[DEL][SUC] Wilson194/hello_world; DefinitelyNotAPorn; FF0000
[DEL][SUC] Wilson194/hello_world; duplicate; cdcdcd
[DEL][SUC] Wilson194/hello_world; enhancement; 84b6eb
[DEL][SUC] Wilson194/hello_world; help wanted; 159818
[DEL][SUC] Wilson194/hello_world; invalid; e6e6e6
[DEL][SUC] Wilson194/hello_world; ItsATrap; FF0000
[DEL][SUC] Wilson194/hello_world; new fantastic label; 0e8a16
[DEL][SUC] Wilson194/hello_world; question; cc317c
[DEL][SUC] Wilson194/hello_world; Won't fix; 888888
```

```
[DEL][SUC] Wilson194/hello_world; wontfix; ffffff
[SUMMARY] 1 repo(s) updated successfully
```

Tip: first [] show command, second [] show result of this command, next is name of repository and last is name of label. Last line is summary

2.2 Server

This is part of modelu for server. You can run your own server on your computer, there is problem with forwarding packet to your computer. Much better is deploy application to some server like *PythonAnywhere <https://www.pythonanywhere.com/>*.

Run the server is very simple. Just type to console:

```
python -m labelord run_server
```

If no parameters are set, server will start in default mode at localhost at port 5000. The config file must be correctly filled. You must specify webhook_secret, github token and target repositories. When you open webpage, you will see some informations about application and all traced repositories. If you create webhook at GitHub that dirrecting request to this IP, server start replicating labels from this repo to all selected repositories.

You can change some behaviour of server with parameters:

- **-h / –host [ip]** - with this parameter, you can change ip of created server
- **-p / –port [port]** - with this parameter, you can change port of created server
- **-d / –debug** - with this parameter you enable debug mode of Flask, which writing some debug informations

CHAPTER 3

GitHub

This application communicate with GitHub all the time. For this purpose labelord using GitHub API. When you want to communicate with GitHub API, you must authenticate session.

3.1 Secret

GitHub secret is for authenticate communication with GitHub API. This is something like password so never tell him anyone. You can create new secret at GitHub setting:

Select settings in menu

Select developer option

Click on **Generate new token**

Generate new token

After generate token, save this token! After continue to next step, you will never found this token again! This token you must fill in to application (config or parameter)

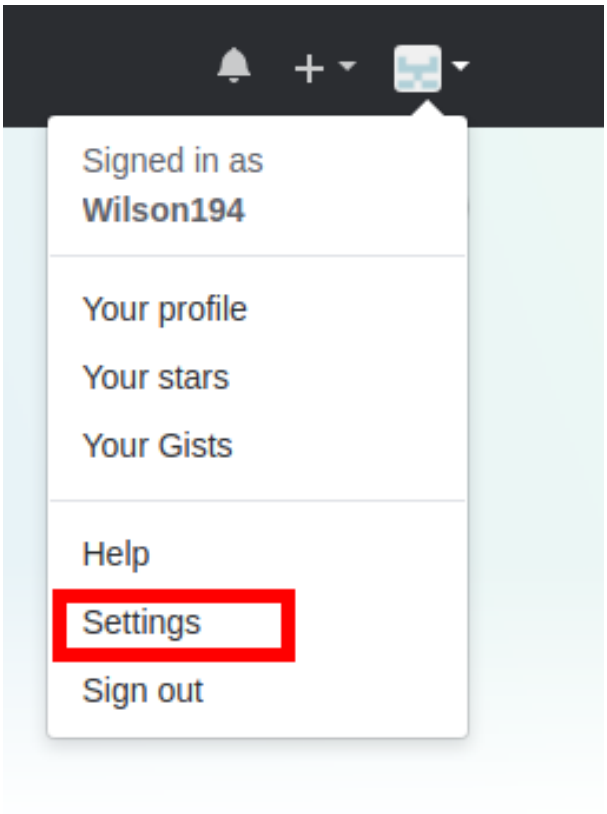
3.2 Webhook

GitHub API can create some request to your web after action. This is used for some feedback actions depends on GitHub actions. You can send this notifications to some web server, and parse this requests. For out project you could use webhook for replicating labels from one repository to another repositories. We just need to set up source repository, to send some requests to out server with running labelord application (server part). You can enable webhook for as many repositories as you want, but all webhooks must have same secret!

You must enable this feature in GitHub setting of repository. Webhook is not for whole account but only for one repo!

Click on **settings** in repository, where you want to create webhook.

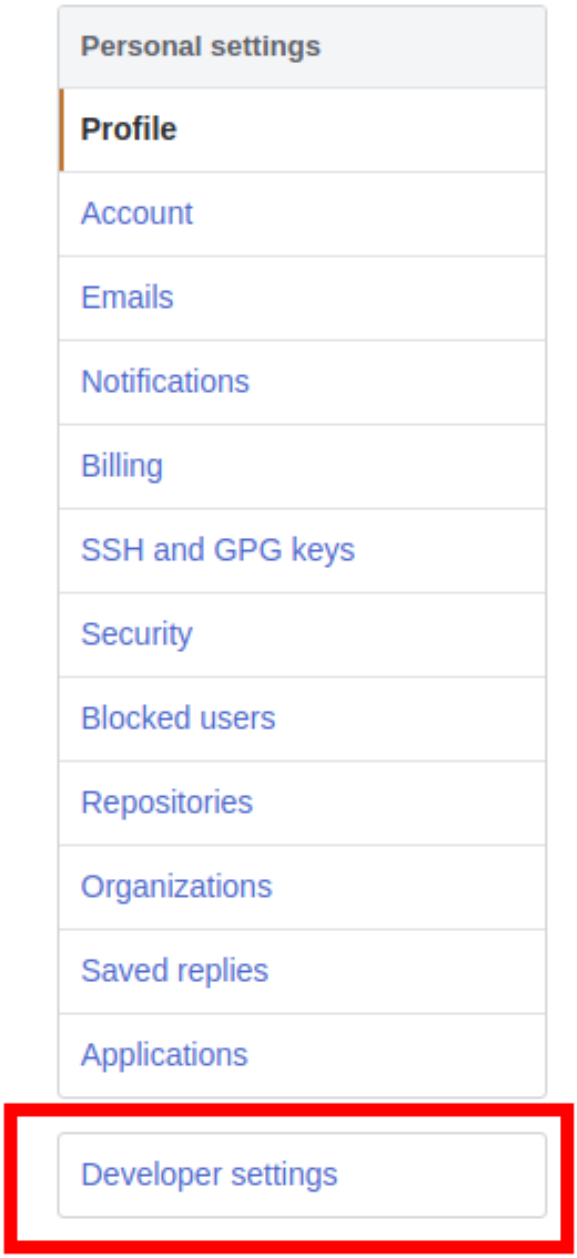
Select option **Webhooks** and in that section click on **Add webhook**

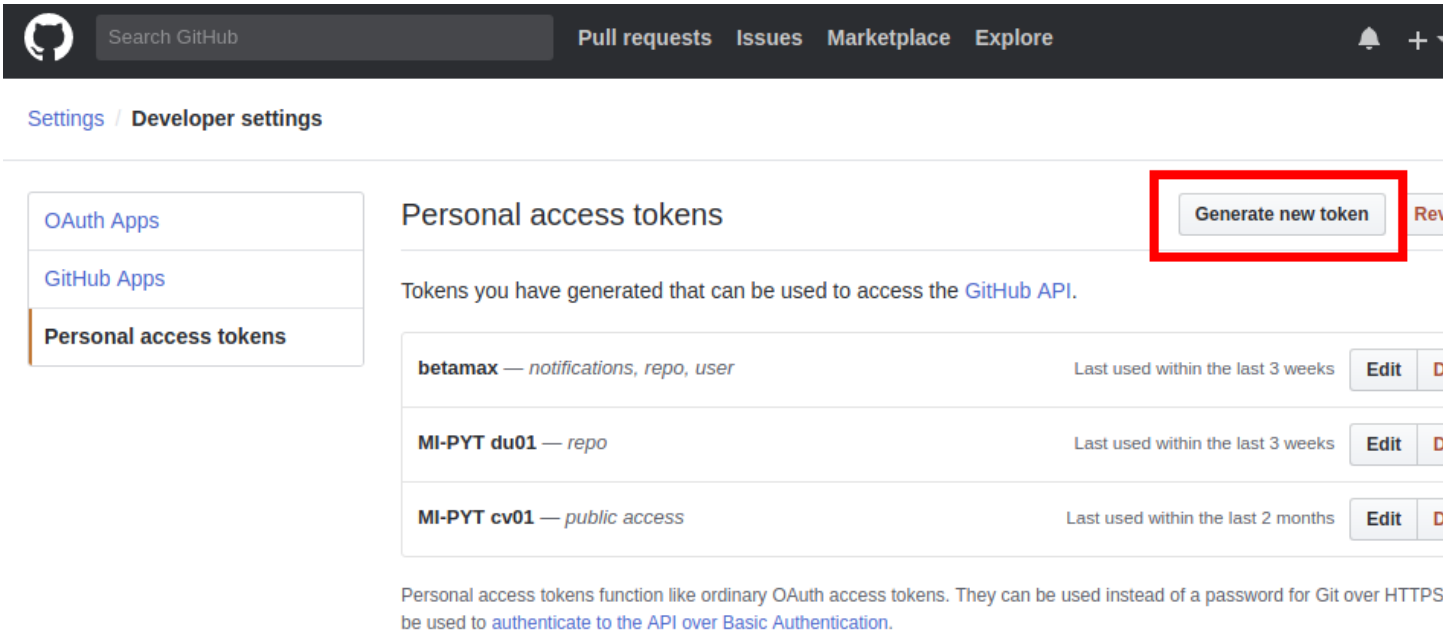


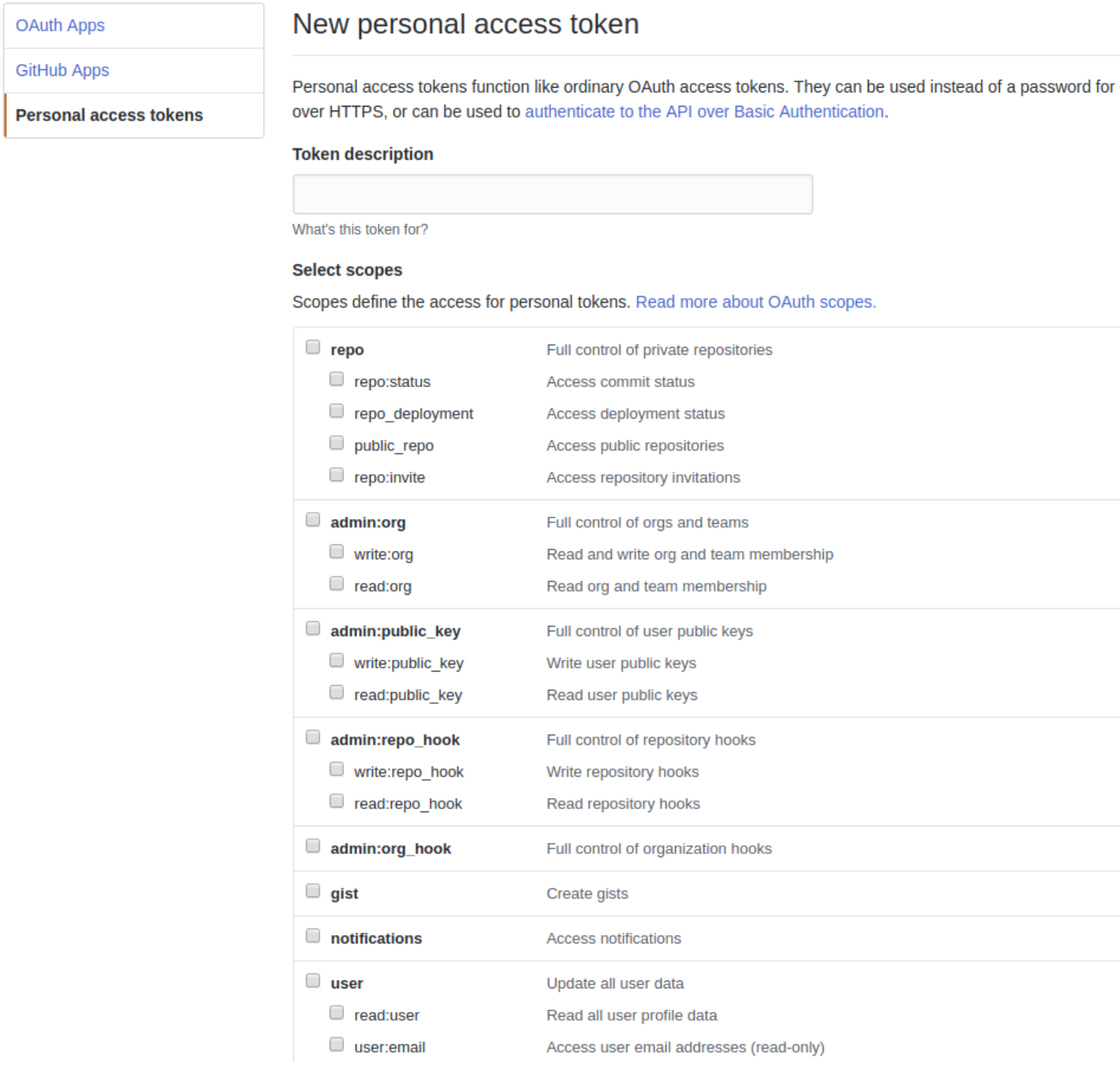
Now you are in form for creating new webhook. There you must fill informations about your server. First you must specify **Payload URL**. This is url, where all requests will be send. This should be your server address.

Next there is **Content type** option. There must be specified json format, because our application using only json format. Next option option is **secret**. This is something like your password. This secret you must fill in config file under option **webhook_secret** under GitHub section. Last item is types of events, you could select **send me everything** or just enable what you want, but you must enable all actions about labels!

Click on **Add webhook** and you are ready for replicating labels from your repository! Hurayyyy!


Personal settings
Profile
Account
Emails
Notifications
Billing
SSH and GPG keys
Security
Blocked users
Repositories
Organizations
Saved replies
Applications
Developer settings


Search GitHub
Pull requests
Issues
Marketplace
Explore
Settings / Developer settings
OAuth Apps
GitHub Apps
Personal access tokens
Personal access tokens
Generate new token
Tokens you have generated that can be used to access the GitHub API.
betamax — notifications, repo, user
Last used within the last 3 weeks
Edit
MI-PYT du01 — repo
Last used within the last 3 weeks
Edit
MI-PYT cv01 — public access
Last used within the last 2 months
Edit
Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS
be used to authenticate to the API over Basic Authentication.



OAuth Apps
GitHub Apps
Personal access tokens

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for over HTTPS, or can be used to authenticate to the API over Basic Authentication.

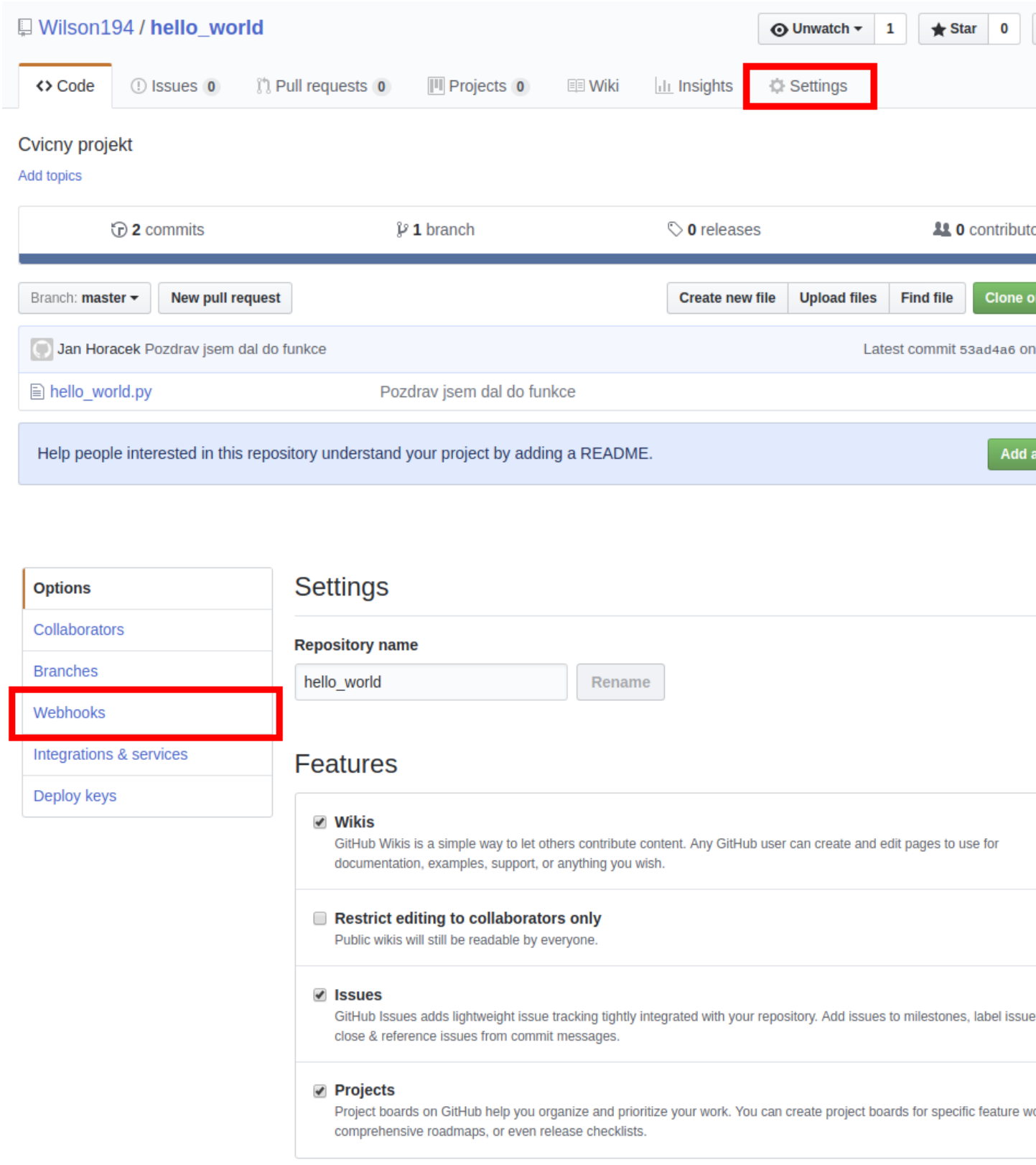
Token description

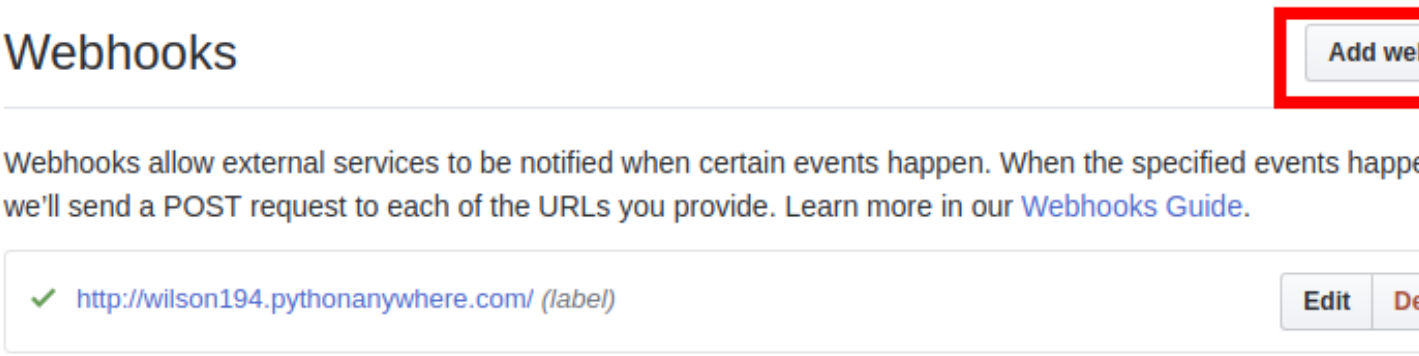
What's this token for?

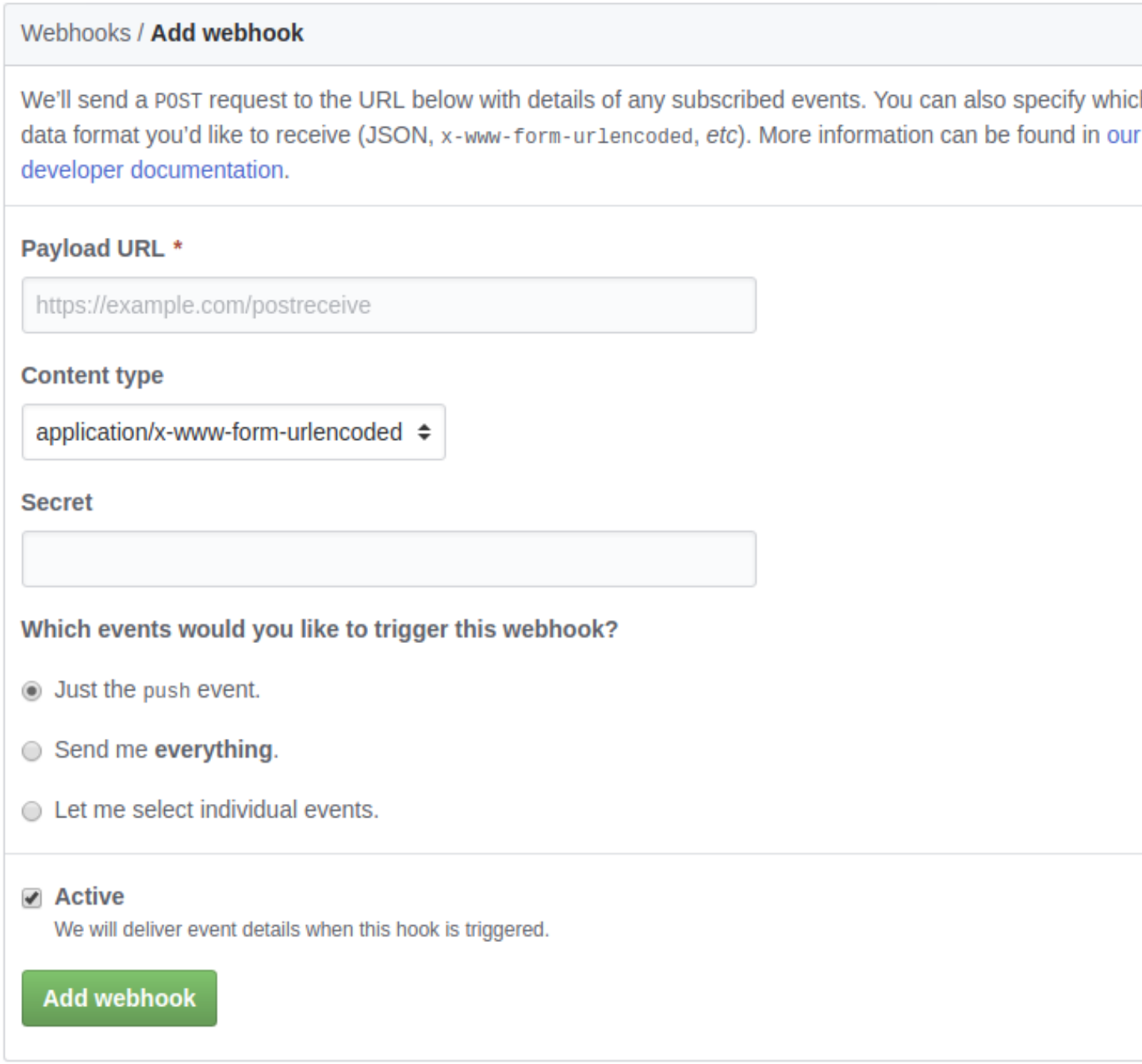
Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

repo	Full control of private repositories
repo:status	Access commit status
repo_deployment	Access deployment status
public_repo	Access public repositories
repo:invite	Access repository invitations
admin:org	Full control of orgs and teams
write:org	Read and write org and team membership
read:org	Read org and team membership
admin:public_key	Full control of user public keys
write:public_key	Write user public keys
read:public_key	Read user public keys
admin:repo_hook	Full control of repository hooks
write:repo_hook	Write repository hooks
read:repo_hook	Read repository hooks
admin:org_hook	Full control of organization hooks
gist	Create gists
notifications	Access notifications
user	Update all user data
read:user	Read all user profile data
user:email	Access user email addresses (read-only)


Wilson194 / hello_world
Unwatch 1
Star 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Cvicny projekt
Add topics
2 commits
1 branch
0 releases
0 contributo
Branch: master
New pull request
Create new file
Upload files
Find file
Clone o
Jan Horacek Pozdrav jsem dal do funkce
Latest commit 53ad4a6 on
hello_world.py
Pozdrav jsem dal do funkce
Help people interested in this repository understand your project by adding a README.
Add a
Options
Collaborators
Branches
Webhooks
Integrations & services
Deploy keys
Settings
Repository name
hello_world
Rename
Features
Wikis
GitHub Wikis is a simple way to let others contribute content. Any GitHub user can create and edit pages to use for documentation, examples, support, or anything you wish.
Restrict editing to collaborators only
Public wikis will still be readable by everyone.
Issues
GitHub Issues adds lightweight issue tracking tightly integrated with your repository. Add issues to milestones, label issue
close & reference issues from commit messages.
Projects
Project boards on GitHub help you organize and prioritize your work. You can create project boards for specific feature wo
comprehensive roadmaps, or even release checklists.


Webhooks
Add we
Webhooks allow external services to be notified when certain events happen. When the specified events happe
we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.
http://wilson194.pythonanywhere.com/ (label)
Edit
De


Webhooks / Add webhook
We'll send a POST request to the URL below with details of any subscribed events. You can also specify whic
data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our
developer documentation.
Payload URL *
https://example.com/postreceive
Content type
application/x-www-form-urlencoded
Secret
Which events would you like to trigger this webhook?
Just the push event.
Send me everything.
Let me select individual events.
Active
We will deliver event details when this hook is triggered.
Add webhook

CHAPTER 4

Config file

All behaviour could be defined in config file. Example of config file is located in GitHub.

```
[github]
token = ThereShouldBeToken
webhook_secret = S3cret!


[labels]
bug = FF0000
Last year = 23FB89
duplicate = cccccc

[repos]
DummyUser/HelloWorld = yes
DummyUser/labelLord = no

[other]
template-repo = DummyUser/repo3
```

4.1 [github]

In this section you can specify information about GitHub authorization.

4.1.1 token

This option is for specify GitHub token. Where you can find GitHub token is specified in GitHub section.

4.1.2 webhook_secret

This option is for specify GitHub webhook secret. Where you can find webhook_secret and what is webhook you can find in section GitHub.

4.2 [labels]

This section is for specify custom labels. That should be created in repositories. The syntax is very simple:

name = color

Name could have spaces. color is hexadecimal RGB value of color. You can specify as many labels as you want. All labels will be created.

4.3 [repos]

In this section you can specify repositories, in which all operations should be done. The syntax is very simple:

user/repository = yes

or

user/repository = no

- **user** - name of owner of repository
- **repository** - name of repository
- **yes/no** - specify, if this repository should be enable for editing.

4.4 [other]

In this section are located all other settings

4.4.1 template-repo

This option is for specify source repository. From this repository will be loaded all labels and will be used as template. You can specify only one source repository.

CHAPTER 5

Module documentation

5.1 Module github

class labelord.github.**Label**(*name*, *color*)
Bases: object

Class that handle one github label.

Variables

- **name** (*str*) – Name of label
- **color** (*str*) – Color of label in hexadecimal format

class labelord.github.**LabelUpdater**(*session*, *config*, *runConfig*)
Bases: object

Class that handle all communication with GitHub api.

Variables

- **session** (*Session*) – authenticated connection session with GitHub
- **config** (*Config*) – Config object with loaded labelord config
- **runConfig** (*dict*) – dictionary with all arguments from console

add_label(*repository*, *label*)
Add label to repository, if there is some error, increase error counter

Parameters

- **repository** (str) – target repository
- **label** (*Label*) – class Label with new label

Return type None

Returns None

get_source_labels(*repository*)
Return source labels (how target repository should have) First use -t/–template_repo (even it was empty) Next try template-repo from config file from others section Last one is labels section in config file

Parameters **repository** (str) – repository from -t/–template-repo parameter

Return type list

Returns list of Label classes

get_target_repositories()
Get list of target repositories, where update will be used First check parameter -a/–all_repos (use all accesable repos) Second try list of repos from repos section in config file If not found, quit with exit code 7

Return type list

Returns list of repos

Raises **SystemExit** – if not repos found

print_log(*repository*, *operationType*, *label*, *error*)
Print one log line

Parameters

- **repository** (str) – Repository specification
- **operationType** (str) – Operation type (UPD,DEL,ADD)
- **label** (*Label*) – Label class with new label
- **error** (str) – error message, if there war some error

Return type None

Returns None

print_summary()
Print summary line based on -q/–quit and -v/–verbose parameters Each of them have another format If there were some errors, quit with exit code 10

Return type None

Returns None

Raises **SystemExit** – if there is some error

remove_label(*repository*, *label*)
Remove label from repository, if there is some error, increase error counter

Parameters

- **repository** (str) – target repository
- **label** (*Label*) – class Label, which should be deleted

Return type None

Returns None

update_label(*repository*, *label*, *oldLabel=None*)
Update existing label in repository, if there is some error, increase error counter

Parameters

- **repository** (str) – target repository

- **label** (*Label*) – class Label with new Label
- **oldLabel** (Optional[*Label*]) – class Label with old label, if change only case in name of label

Return type None

Returns None

update_labels(*newLabels*, *targetRepositories*)
Change labels in given repositories

Parameters

- **newLabels** (list) – list of new labels
- **targetRepositories** (list) – list of target repositories

Return type None

Returns None

class labelord.github.**MyAuth**(*token: str*)
Bases: requests.auth.AuthBase

Class for authentication of requests

Variables **token** (*str*) – authentication token for GitHub

set_token(*token*)
Set token to class auth

Parameters **token** (*str*) – GitHub token

labelord.github.**find_label**(*label*, *iterable*)
Find class Label in list of Label classes

- it -> founded old label
- **type -> type of match**
  - 0 -> not found
  - 1 -> name found, other color
  - 2 -> name found, same color
  - 3 -> name different case

Parameters

- **label** (*Label*) – searched label
- **iterable** (list) – list of labels

Return type tuple

Returns tuple (it, type)

labelord.github.**get_list_labels**(*session*, *repository*, *exitProgram=True*)
Get list of labels in given repository

Parameters

- **session** (Session) – authenticated session
- **repository** (str) – target repository

- **exitProgram** (bool) – if False, program will not quit if repository not found, only return False

Return type Union[list, bool]

Returns list of Labels / False

labelord.github.**get_list_repos**(*session*)
Get list of available repos

Parameters **session** (Session) – authenticated session

Return type list

Returns list of available repos

labelord.github.**load_config**(*cfg*)
Load .cfg file and parse to Config object

Parameters **cfg** (str) – path to config file

Return type ConfigParser

Returns config file Object

labelord.github.**load_token**(*config*, *token*)
Load token from config file, if token given by parameter, just return this string

Parameters

- **config** (ConfigParser) – config file object
- **token** (str) – token string

Return type str

Returns token

Raises **SystemExit** – if no token found

labelord.github.**validate_response**(*response*, *exitProgram=True*)
Validate response by status code

- 401 - code 4
- 404 - code 5
- response.ok == False - code 10

Parameters

- **response** (Response) – response object from session
- **exitProgram** (bool) – if false, quit will be suppressed

Return type int

Returns error code

Raises **SystemExit** – if exitProgram is tru

5.2 Module labelord

5.2.1 labelord

Main program group of click

param ctx context for object passing

param config path to config file, default config.cfg

param token GITHUB token

return None

```
labelord [OPTIONS] COMMAND [ARGS]...
```

Options

-c, --config <config>
Specify path to config file.

-t, --token <token>
Token for GitHub API.

--version
Show the version and exit.

Environment variables

LABELORD_CONFIG
Provide a default for *-c*

GITHUB_TOKEN
Provide a default for *-t*

list_labels

List of labels from repository :param ctx: context :param repository: name if repository :return: None

```
labelord list_labels [OPTIONS] REPOSITORY
```

Arguments

REPOSITORY
Required argument

list_repos

Command for repositories list :param ctx: context :return: None

```
labelord list_repos [OPTIONS]
```

run

Main program for copy labels and update labels

param ctx context

param sourceRepository source repository for source labels

param allRepos flag if all repos should be updated

param mode mode -> update / replace

param quiet flag for quit mode

param verbose flag for verbose mode

param dryRun flag for dryRun mode

return None

```
labelord run [OPTIONS] MODE
```

Options

-t, --template-repo <sourceRepository>
Source repository for labels

-a, --all-repos
Use all available repository

-q, --quiet
Quit print, no output to console

-v, --verbose
Debug info to console

-d, --dry-run
Run only testing instation, no changes at repos

Arguments

MODE
Required argument

run_server

Run server command :param ctx: context from click :param hostname: hostname for flask run :param port: port for flask run :param debug: enable debug mode :return: None

```
labelord run_server [OPTIONS]
```

Options

-h, --host <hostname>
Host name for start server

-p, --port <port>
Port for start server

-d, --debug
Enable flask server debug mode

5.3 Module server

class labelord.server.**LabelordWeb**(**args*, ***kwargs*)
Bases: flask.app.Flask

Custom class for app, extend flask.Flask

inject_session(*session*)
Inject session to current app (for testing) :type session: Session :param session: session class :rtype: None :return: None

reload_config()
Reload config from system variable. Look to system variable LABELORD_CONFIG for path to config file. Default is config.cfg Reload whole authentication and all settings from config

Return type None

Returns None

Raises **SystemExit** – if no webhook provided

set_labelord_config(*config*)
Setter for config (set ConfigParser object)

Parameters **config** (ConfigParser) – config object

Return type None

Returns None

labelord.server.**check_allowed_repo**(*repo*)
Check if repo is allowed in config

Parameters **repo** (str) – repository string

Return type bool

Returns True if allowed, False otherwise

labelord.server.**check_signature**(*msg*, *secret*, *signature*)
Check sha1 sum of GitHub request

Parameters

- **msg** (str) – body message
- **secret** (str) – secret from config
- **signature** (str) – signature from GitHub header

Return type bool

Returns True if checked, False otherwise

labelord.server.**convert_git_repo**(*text*)
Create link to GitHub repo

labelord.server.**create_label_request**(*js*)
Handle POST request for create label Create new label

Parameters **js** – json request object

Return type None

Returns None

labelord.server.**delete_label_request**(*js*)
Handle POST request for delete label Delete label

Parameters **js** – json request object

Return type None

Returns None

labelord.server.**edit_label_request**(*js*)
Handle POST edit label :param js: json request object :return: None

labelord.server.**index**()
Main page of application Show main information about settings of server (repos, basic info..)

Return type str

Returns

labelord.server.**post_request**()
Request handler for POST method This function handle all post requests to our page. Parse post request, control all authentication and call correct function from labelord

Return type str

Returns str

Raises **flaskAbort** – if come request with bad header

CHAPTER 6

Using like API

If you want to use this module as part of your application, you can. In this part of documentation i will try describe main functions, which help you integrate this module to your application.

6.1 Basic structure

Module is divided to three parts:

- labelord - which enable running this module from console
- server - which creating web server
- github - which doing all magic with GitHub API

If you want to use this module as part of your code, you will probably use only github part. I don't recommend you using labelord console as entry point to this module. It's really bad idea. Much better solution is use directly functions and classes from github part.

6.2 Setup main class

The main class `LabelUpdater` handle all action with GitHub, so you need to initialize this class. Class constructor accept tree parameters:

- **session** - this is session with correct authetication for GitHub
- **config** - loaded config file (you can use load_config function
- **runConfig - dictionary of parameters, which changing behaviour of class**
  - allRepos - bool
  - dryRun - bool
  - verbose - bool

- quiet - bool
- mode - str (update/replace)

```
import labelord.github
import requests

s = requests.Session()
cfg = labelord.github.load_config('_static/default_config.cfg')
lu = labelord.github.LabelUpdater(s,cfg,{'allRepos':True, 'mode':'update'})
print(lu.allRepos)
print(lu.mode)
```

```
True
update
```

Your session must have correct authentication. You can use created class for authentication MyAuth. This class need only github token, you can specify token in constructor and anytime you can change it with setter `set_token(token:str)`

```
import labelord.github
auth = labelord.github.MyAuth('secret_token')
auth.set_token('new_secret_token')

print(auth.token)
```

```
new_secret_token
```

New you have all initialized and you can start using API. There are several other functions that could help you:

6.2.1 get_list_repos(session)

This function will return list of all repos, which you can find with current GitHub secret

For example:

```
import labelord.github

repos = labelord.github.get_list_repos(session)
print(repos)
```

will return list of repositories. If don't specify token or something goes wrong, system will raise SystemExit

6.2.2 get_list_labels(session, repository)

This function will return list of label in one repository. You must specify session and name of repository.

For example:

```
import labelord.github

repos = labelord.github.get_list_labels(session, 'dumpUser/HelloWorld')
print(repos)
```

will return list of labels of selected repository. List is list of class Label, described later.

6.3 Label class

For save label to one object there is class Label. Constructor need name and color of label, both parameters are strings. Name could have spaces and color is hexadecimal value of RGB color (without hashtag). Be careful, color is case sensitive (same color but could create more edit actions)

Example usage of class:

```
import labelord.github

label = labelord.github.Label('Bug', 'FF0000')
label2 = labelord.github.Label('Bug', 'ff0000')
print(label.name)
print(label.color)
print(label == label2)
```

```
Bug
FF0000
False
```

Label class have some controls of color (correct color format), some you need only catch exceptions if for example user will specify labels

Example:

```
import labelord.github

try:
    label = labelord.github.Label('Bug', 'HH0000')
except ValueError as e:
    print(str(e))


try:
    label = labelord.github.Label('Bug', 'FF0000FF')
except ValueError as e:
    print(str(e))
```

```
Label color must be hexadecimal number
Label color must have 6 digit! (RGB in hexadecimal)
```

6.4 Using LabelUpdater class

Main class have some functions, which you can use. All functions are documented in module documentation. There I will only write interesting ones.

6.4.1 get_source_labels(repository)

This function accept string name of repository and return list of labels of that repository. If repository is not set, try to find source repository in config. If source repository is not in config, load labels from config. If labels not in config, function exit with error

6.4.2 get_target_repositories()

Based on parameters and config file will return list of target repositories (list of strings)

6.4.3 update_labels(newLabels,targetRepositories)

This function will update labels in target repositories. NewLabels is list of classes Label. Function connect to repository and based on mode, create/update/delete all necessary labels.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

Symbols

A

C

D

E

F

G

I

L

M

P

R

S

U

V