taggo Documentation

Release 0.13.9

Lars Solberg

Mar 08, 2018

Contents

Contents:

CHAPTER 1

taggo

Tag organizer that uses names of files and folders to create symlinks. Tags are defined by using #hashtags in the name. They can also be as many sub levels as you want, like #sub-hash-tag

note This version is a completey different version than the old (https://github.com/xeor/taggo/tree/0.2). The old version works for python 2 (but not 3). It also had config-file instead of parameters. Check out the repo if you want it..

- Free software: MIT license
- Documentation: https://taggo.readthedocs.io
- Source: https://github.com/xeor/taggo
- Issues: https://github.com/xeor/taggo/issues

1.1 Introduction

This project is in beta stage, please report bugs :)

Any questions, thoughts, bugs are very welcome!

1.2 Requirements

- 3.6+

1.3 Docker

Start the container with environment variables like *CRON_TAGGO_0* with the format * * * * *|*run* ....

- CRON_TAGGO_n where n is a number, start at 0, have as many as you want.

- We take care automaticly that only 1 of each number is running at a time. Example, if one of your job is running every minute and it takes more than a minute to finish. It wont start the 2nd time.
- The environment variable is split in 2 by a |. The first param is a cron, the 2nd is the parameters sent to the *taggo* command.

1.4 FAQ

- Why the name taggo?
  - It's a tagging tool. It does stuff with tags. What do you suggest? Tagging, taggs, tags, tag2fold... no.. Taggo!
- Why do you want to create tags with symlinks?
  - Because everyone have underestimated the power of tagging data.
  - Photo filenames are just wasted, what does DCIM1234.jpg tell you?
  - You know you miss one folder that contains all your dog pictures.
  - You sould not depend on a 3rd party program/database to manage your files/photos.

CHAPTER 2

Installation

2.1 Stable release

To install taggo, run this command in your terminal:

```
$ pip install taggo
```

If you also want to download the required packages to run all metadata-addons, install taggo like this.

```
$ pip install taggo[all]
```

This is the preferred method to install taggo, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for taggo can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/xeor/taggo
```

Or download the tarball:

```
$ curl  -OL https://github.com/xeor/taggo/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

Taggo is most usefull if you use it on the commandline

Files and folders example layout:

```
root@4c95ee980234:/# find /data/
/data/
/data/2016
/data/2016/best taco #recipes-dinner.txt
/data/2016/python snippets #projects-programming-python.txt
/data/2017
/data/2017/#traveling-london
/data/2017/#traveling-london/places to visit.txt
/data/2017/#traveling-london/airplane tickets.pdf
/data/2015
/data/2015/chocolate cake #recipes-cake.txt
/data/2015/truffle chicken #recipes-dinner.txt
/data/2015/#important note.txt
```

3.1 Making symlinks

You can now run taggo to create symlinks to the tagged files:

```
root@4c95ee980234:/# pip install taggo
root@4c95ee980234:/# taggo run data tags
root@4c95ee980234:/# find tags/
tags/
tags/traveling
tags/traveling/london
tags/traveling/london/2017_#traveling-london - #traveling-london
tags/recipes
tags/recipes/cake
tags/recipes/cake/2015 - chocolate cake #recipes-cake.txt
tags/recipes/dinner
```

```
tags/recipes/dinner/2016 - best taco #recipes-dinner.txt
tags/recipes/dinner/2015 - truffle chicken #recipes-dinner.txt
tags/important
tags/important/2015 - #important note.txt
tags/projects
tags/projects/programming
tags/projects/programming/python
tags/projects/programming/python/2016 - python snippets #projects-programming-python.
↪txt
```

notice that we have created a folder hieracy based on your tags with symlinks pointing to the correct files.

3.1.1 cli options (run)

–metadata-addon, –metadata-default

Add extra-data to use in filters or symlink-name. Use metadata-addon multiple times to add multiple.

Currently you can define:

- stat
- filetype
- exif
- md5

Use *–metadata-default* to define defaults if some of the addons is not producing data on everything. Example *–metadata-default key_should_exist=value*

–auto-cleanup

Run cleanup (see own command) after we are done

–filter, –filter-mode

Filters are checked in the order of how expensive the metadata is to calculate.. If a filter have a match, and there are no more filter to check. We will include or skip the file acordonly..

We do some string to object convertion, so if you define surtain strings, they behave specially. Example:

- –filter 'value=None': Matches if value is not defined.

Filters are split up with the *key*, an *operator*, then a *value*. A filter can example be *file-ext=jpeg*, *file-ext__contains=jpeg,png,gif*. We add *exact* as a operator if it is not specified. You specify an operator like above where *contains* is the operator.

Valid operators are:

- *exact*: The default (==)
- *neq*: Not equal (!=)
- *contains*: Value must be a comma-separated list of items to match against.
- *icontains*: Same as contains, but doesnt care about case
- *startswith*, *istartswith*, *endswith*, *iendswith*: Selfexplain

- *gt*, *gte*, *lt*, *lte*: GreaterThan, GreaterThanEqueal, LessThan, LessThanEqual. Values must be numbers!
- *regex*: Checks using python re.match()

Include (–filter-mode=include) are using logical AND. In other words, every –filter you define must match in order for it to be included. Exclude used logical OR. So, if any of the exclude filter matches. It will be excluded.

If you are using a filter that we don't have data on, example *–filter non_existing=abc*, we will ignore it.

–filter-query

If you install *jmespath*, you can use *–filter-query*. This is a very powerfull json query-language (http://jmespath.org/). See what the pros and cons are on the comparison of –filter-query and –filter below.

Examples * –filter-query='tag.original == *test*' * –filter-query='contains(paths.*, *archive*) && "file-ext" == *jpg*'

–symlink-name, –symlink-name-file, –symlink-name-folder

Let say we have

- src folder: tests/test_files/
- a tagged file: tests/test_files/tagged/folders/i-3 #Hollydays-Christmas.jpg
- dst folder: temp

Then the template tags will become

- tag[original]: 'Hollydays-Christmas'
- tag[as-folders]: 'Hollydays/Christmas'
- tag-param[original]: If your tag was in the format #tag(param here), this would be "param here"
- rel_folders: tagged_folders

** Which is a _ separated list of folders from the file, all the way up to the dst folder ** It will be set to "root" if there are no list of relative paths ** We will not include the tagged folder itself if we this is a tagged folder.

- basename: i-3 #Hollydays-Christmas.jpg

** Name of the file

- paths[0]: folders
- paths[1]: tagged
- paths[2]:
- file-ext (only on –symlink-name-file): .jpg
- md5 (only on –symlink-name-file, need –metadata-addon md5): d41d8cd98f00b204e9800998ecf8427e
- stat (need –metadata-addon stat)

** This makes a bunch of file/folder stats available (using the python os.stat) function. Use –debug to see what you have. ** We will also make a _iso version of atime, ctime and mtime with iso8601 of the value ** As well as _year, _month and _day

- exif (only on –symlink-name-file, need –metadata-addon exif, and python package *piexif* installed)

** You should set the template-keys you depends on with default-values using –metadata-default, or you might easiely get errors ** These will be available (flat) *** **exif_**...

Example

- –symlink-name, like *–symlink-name "{tag[as-folders]}/{basename}"*
- –symlink-name-file "{md5}" –symlink-name-folder "{tag[as-folders]}/{basename}" –metadata-addon md5

Note that if you want to use *–symlink-name-file* or *–symlink-name-folder*, both needs to be defined. Else *–symlink-name* is used.

3.2 Difference between *–filter* and *–filter-query*

TLDR: If possible, use *–filter* for speed :)

The longer story, is that –filter knows what you are filtering on, before it completes all the metadata-addon calculations. This is because the correct filter gets checked after each metadata calculation. Example, when the *stat* addon is done, the *stat* filters are checked. If the filter dictates that it should skip the file, no more metadata calculation is done for that file. This is usefull and can save you some time. However, there are some big cons using the –filter:

- You wont be able to filter on data that is not flat. Example, there are no way to filter on *paths[]...*
- It is not that powerfull, and doing logical AND, OR, NOT and such are a pain.

The *–filter-query* is using jmespath, and it have a very powerfull querylanguage. It can handle more logic, and is much more powerfull than *–filter*. However.. There are some cons:

- It depends on a 3rd party lib (*pip install jmespath*)
- The filter are checked once per file, after all metadata addons are calculated.

Both filter-types can however be combined.. So you can do a quick check using –filter, then a more advanced check later using –filter-query

3.3 Cleanup

Symlinks that are dead can be cleaned up easiely:

```
root@4c95ee980234:/# rm "/data/2016/best taco #recipes-dinner.txt"

root@4c95ee980234:/# taggo cleanup tags/
Deleting symlink /tags/recipes/dinner/2016 - best taco #recipes-dinner.txt
```

3.4 List tags

To list tags available in a source directory:

```
root@4c95ee980234:/# taggo info data/
Folder tags:
  traveling-london

File tags:
  important
  projects-programming-python
  recipes-cake
  recipes-dinner
```

3.5 Rename tags

You can also rename tags if you want them nested another way, or just got a typo:

```
root@4c95ee980234:/# taggo rename data/ traveling-london traveling-uk-london
Renaming: /data/2017/{#traveling-london -> #traveling-uk-london}

root@4c95ee980234:/# taggo cleanup tags/
Deleting symlink /tags/traveling/london/2017_#traveling-london - #traveling-london
Removing empty folder: /tags/traveling/london
Removing empty folder: /tags/traveling

root@4c95ee980234:/# taggo run data tags
```

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/xeor/taggo/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/xeor/taggo/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 :)

4.2 Get Started!

Ready to contribute? Here's how to set up *taggo* for local development.

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

```
$ git clone git@github.com:your_name_here/taggo.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 taggo
$ cd taggo/
$ 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 taggo 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.

4.3 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 3.6, and up. Check https://travis-ci.org/xeor/taggo/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Debug travis

Start a custom build (trigger build) and input something like *script: DEBUG=true pytest -k test_symlink_creation*

CHAPTER 5

Credits

5.1 Project owner

- Lars Solberg <lars.solberg@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.13.0 (2018-03-04)

- Dropping python 2.x support... Some things might end up being problematic to support. Like symlinks for directories in windows. So instead of making a bunch of hacks around functionality. It is now dropped.

6.2 0.12.0 (2018-03-03)

- Making symlink name template configurable
- Symlink collision handling
- Logs to stdout/stderr depending on message severety
- Option to output log as json
- Option to prompt/wait after each symlink. Usefull for debugging
- Lots of things around symlink-name-templates, it's now completly configurable.
- Possible to have extrainfo (used in symlink-name) from a tag parameter. Like #tag(info)
- Using powerful filters to not symlink certain files, or only symlink some files.
- Metadata-addons to use special file-info as in the symlink-name, like md5, stat, exif-data, ...
- Output data as json, if you want a logparser to use it. Single-lines..
- Configurable collision handling. If symlink already exist and points to a different file.
- Making *pip install taggo[all]* to get all metadata-addon required libs
- –auto-cleanup option in *run*
- Log different messages to stdout or stderr

6.3 0.11.0 (2018-02-20)

- Fixing up docker image

6.4 0.10.0 (2017-11-04)

- Basic docker image

6.5 0.9.0 (2017-10-21)

- Python 2.7 support

6.6 0.8.0 (2017-10-21)

- Good test coverage
- Things are mostly working
- Rename functionality
- List/info
- Much more

6.7 0.4.0 (2017-10-08)

- Started a complete rewrite, mainly focusing on using python 3.6
- Test on PyPI.. Non working version.

6.8 0.2 (2017-10-07)

- Checkpoint of the old version working only with 2.x. This checkpoint contains code from many years ago.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search