shaarli-client Documentation

Release 0.5.0

The Shaarli Community

Oct 09, 2022

User Documentation

1 Installation **3**
- 1.1 From the Python Package Index (PyPI) . . . 3
- 1.2 From the source code . . . 3

2 Configuration **5**
- 2.1 Example . . . 5

3 Usage **7**
- 3.1 Getting help . . . 7
- 3.2 Examples . . . 8

4 Change Log **13**
- 4.1 v0.5.0 - 2022-07-26 . . . 13
- 4.2 v0.4.1 - 2021-05-13 . . . 13
- 4.3 v0.4.0 - 2020-01-09 . . . 14
- 4.4 v0.3.0 - 2019-02-23 . . . 14
- 4.5 v0.2.0 - 2017-04-09 . . . 14
- 4.6 v0.1.0 - 2017-03-12 . . . 15

5 Testing **17**
- 5.1 Environment and requirements . . . 17
- 5.2 Tools . . . 17
- 5.3 Running the tests . . . 18

6 Releasing **19**
- 6.1 Environment and requirements . . . 19
- 6.2 PyPI and TestPyPI configuration . . . 19
- 6.3 Releasing `shaarli-client` . . . 20

7 GnuPG **21**
- 7.1 Introduction . . . 21
- 7.2 Generate a GPG key . . . 22

8 Indices and tables **25**

Command-line interface (CLI) to interact with a Shaarli instance.

CHAPTER 1

Installation

`shaarli-client` is compatible with Python 3.4 and above and has been tested on Linux.

1.1 From the Python Package Index (PyPI)

The preferred way of installing `shaarli-client` is within a Python virtualenv; you might want to use a wrapper such as virtualenvwrapper or pew for convenience.

Here is an example using a Python 3.5 interpreter:

```
# create a new 'shaarli' virtualenv
$ python3 -m venv ~/.virtualenvs/shaarli

# activate the 'shaarli' virtualenv
$ source ~/.virtualenvs/shaarli/bin/activate

# install shaarli-client
(shaarli) $ pip install shaarli-client

# check which packages have been installed
$ pip freeze
PyJWT==1.4.2
requests==2.13.0
requests-jwt==0.4
shaarli-client==0.1.0
```

1.2 From the source code

To get `shaarli-client` sources and install it in a new virtualenv:

```
# fetch the sources
$ git clone https://github.com/shaarli/python-shaarli-client
$ cd python-shaarli-client

# create and activate a new 'shaarli' virtualenv
$ python3 -m venv ~/.virtualenvs/shaarli
$ source ~/.virtualenvs/shaarli/bin/activate

# build and install shaarli-client
(shaarli) $ python setup.py install

# check which packages have been installed
$ pip freeze
PyJWT==1.4.2
requests==2.13.0
requests-jwt==0.4
shaarli-client==0.1.0
```

You can also use `pip` to install directly from the git repository:

```
$ python3 -m venv ~/.virtualenvs/shaarli
$ source ~/.virtualenvs/shaarli/bin/activate
(shaarli) $ pip3 install git+https://github.com/virtualtam/python-shaarli-
↪client@master # or any other branch/tag
```

CHAPTER 2

Configuration

`shaarli-client` loads information about Shaarli instances from a configuration file, located at:

- `~/.config/shaarli/client.ini` (recommended)
- `~/.shaarli_client.ini`
- `shaarli_client.ini` (in the current directory)
- user-specified location, using the `-c`/`--config` flag

Several Shaarli instances can be configured:

`[shaarli]` the default instance

`[shaarli:<my-other-instance>]` an additional instance that can be selected by passing the `-i` flag: `$ shaarli -i my-other-instance get-info`

2.1 Example

```
[shaarli]
url = https://host.tld/shaarli
secret = s3kr37!

[shaarli:shaaplin]
url = https://shaarli.shaapl.in
secret = m0d3rn71m3s

[shaarli:dev]
url = http://localhost/shaarli
secret = asdf1234
```

CHAPTER 3

Usage

Once installed, `shaarli-client` provides the `shaarli` command, which allows to interact with a Shaarli instance's REST API.

3.1 Getting help

The `-h` and `--help` flags allow to display help for any command or sub-command:

```
$ shaarli -h

usage: shaarli [-h] [-c CONFIG] [-i INSTANCE] [-u URL] [-s SECRET]
               [-f {json,pprint,text}] [-o OUTFILE] [--insecure]
               {get-info,get-links,post-link,put-link,get-tags,get-tag,put-tag,delete-
↪tag,delete-link}
               ...
positional arguments:
  {get-info,get-links,post-link,put-link,get-tags,get-tag,put-tag,delete-tag,delete-
↪link}
                        REST API endpoint
    get-info            Get information about this instance
    get-links           Get a collection of links ordered by creation date
    post-link           Create a new link or note
    put-link            Update an existing link or note
    get-tags            Get all tags
    get-tag             Get a single tag
    put-tag             Rename an existing tag
    delete-tag          Delete a tag from every link where it is used
    delete-link         Delete a link

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        Configuration file
```

(continues on next page)

(continued from previous page)

```
  -i INSTANCE, --instance INSTANCE
                        Shaarli instance (configuration alias)
  -u URL, --url URL     Shaarli instance URL
  -s SECRET, --secret SECRET
                        API secret
  -f {json,pprint,text}, --format {json,pprint,text}
                        Output formatting
  -o OUTFILE, --outfile OUTFILE
                        File to save the program output to
  --insecure            Bypass API SSL/TLS certificate verification
```

```
$ shaarli get-links -h

usage: shaarli get-links [-h] [--limit LIMIT] [--offset OFFSET]
                         [--searchtags SEARCHTAGS [SEARCHTAGS ...]]
                         [--searchterm SEARCHTERM [SEARCHTERM ...]]
                         [--visibility {all,private,public}]

optional arguments:
  -h, --help            show this help message and exit
  --limit LIMIT         Number of links to retrieve or 'all'
  --offset OFFSET       Offset from which to start listing links
  --searchtags SEARCHTAGS [SEARCHTAGS ...]
                        List of tags
  --searchterm SEARCHTERM [SEARCHTERM ...]
                        Search terms across all links fields
  --visibility {all,private,public}
                        Filter links by visibility
```

3.2 Examples

3.2.1 General syntax

```
$ shaarli <global arguments> <endpoint> <endpoint arguments>
```

Note: The following examples assume a *Configuration* file is used

3.2.2 GET info

```
$ shaarli get-info
```

```
{
    "global_counter": 1502,
    "private_counter": 5,
    "settings": {
        "default_private_links": false,
        "enabled_plugins": [
            "markdown",
```

(continues on next page)

(continued from previous page)


```
            "archiveorg"
        ],
        "header_link": "?",
        "timezone": "Europe/Paris",
        "title": "Yay!"
    }
}
```


3.2.3 GET links

```
$ shaarli get-links --searchtags super hero
```


```
[
    {
        "created": "2015-02-22T15:14:41+00:00",
        "description": "",
        "id": 486,
        "private": false,
        "shorturl": null,
        "tags": [
            "wtf",
            "kitsch",
            "super",
            "hero",
            "spider",
            "man",
            "parody"
        ],
        "title": "Italian Spiderman",
        "updated": "2017-03-10T19:53:34+01:00",
        "url": "https://vimeo.com/42254051"
    },
    {
        "created": "2014-06-14T09:13:36+00:00",
        "description": "",
        "id": 970,
        "private": false,
        "shorturl": null,
        "tags": [
            "super",
            "hero",
            "comics",
            "spider",
            "man",
            "costume",
            "vintage"
        ],
        "title": "Here's Every Costume Spider-Man Has Ever Worn",
        "updated": "2017-03-10T19:53:34+01:00",
        "url": "http://mashable.com/2014/05/01/spider-man-costume"
    }
]
```

3.2.4 POST link

```
$ shaarli post-link --url https://w3c.github.io/activitypub/
```


```
{
    "created": "2018-06-04T20:35:12+00:00",
    "description": "",
    "id": 3252,
    "private": false,
    "shorturl": "kMkHHQ",
    "tags": [],
    "title": "https://w3c.github.io/activitypub/",
    "updated": "",
    "url": "https://w3c.github.io/activitypub/"
}
```


3.2.5 PUT link

```
shaarli put-link --private 3252
```


```
{
    "created": "2018-06-04T20:35:12+00:00",
    "description": "",
    "id": 3252,
    "private": true,
    "shorturl": "kMkHHQ",
    "tags": [],
    "title": "?kMkHHQ",
    "updated": "2018-06-04T21:57:44+00:00",
    "url": "http://aaron.localdomain/~virtualtam/shaarli/?kMkHHQ"
}
```


3.2.6 GET tags

```
$ shaarli get-tags --limit 5
```


```
[
    {
        "name": "bananas",
        "occurrences": 312
    },
    {
        "name": "snakes",
        "occurrences": 247
    },
    {
        "name": "ladders",
        "occurrences": 240
    },
    {
        "name": "submarines",
        "occurrences": 48
```


(continues on next page)

(continued from previous page)

```
    },
    {
        "name": "yellow",
        "occurrences": 27
    }
]
```

3.2.7 GET tag

```
$ shaarli get-tag bananas
```

```
{
    "name": "bananas",
    "occurrences": 312
}
```

3.2.8 PUT tag

```
$ shaarli put-tag w4c --name w3c
```

```
{
    "name": "w3c",
    "occurrences": 5
}
```

3.2.9 New lines/line breaks

If you need to include line breaks in your descriptions, use a literal newline `\n` and *$'...'* around the description:

```
$ shaarli post-link --url https://example.com/ --description $'One\nword\nper\nline'.
```

3.2.10 NOT (minus) operator

It is required to pass all values to *–searchtags* as a quoted string:

```
$ shaarli get-links --searchtags "video -idontwantthistag"
```

The value passed to –searchtags must not start with a dash, a workaround is to start the string with a space:

```
$ shaarli get-links --searchtags " -idontwantthistag -northisone"
```

CHAPTER 4

Change Log

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

4.1 v0.5.0 - 2022-07-26

Added:

- Add `delete-link` command (delete a link by ID)

Changed:

- Update test tooling and documentation

Fixed:

- Fix `--insecure` option for non-GET requests

Security:

- Update PyJWT to 2.4.0

4.2 v0.4.1 - 2021-05-13

Added:

- Add support for Python 3.7, 3.8 and 3.9

Changed:

- Bump project and test requirements
- Update test tooling and documentation

Removed:

- Drop support for Python 3.4 and 3.5

Security:

- Rework JWT usage without the unmaintained requests-jwt library

4.3 v0.4.0 - 2020-01-09

Added:

- CLI:
  - Add support for `--insecure` option (bypass SSL certificate verification)

4.4 v0.3.0 - 2019-02-23

Added:

- CLI:
  - Add support for endpoint resource(s)
- REST API client:
  - `PUT api/v1/links/<LINK_ID>`

Fixed:

- Use requests-jwt < 0.5
- Fix *POST /link* endpoint name

4.5 v0.2.0 - 2017-04-09

Added:

- Add client parameter checks and error handling
- Read instance information from a configuration file
- REST API client:
  - `POST api/v1/links`

Changed:

- CLI:
  - rename `--output` to `--format`
  - default to 'pprint' output format
  - improve endpoint-specific parser argument generation
  - improve exception handling and logging

4.6 v0.1.0 - 2017-03-12

Added:

- Python project structure
- Packaging metadata
- Code quality checking (lint)
- Test coverage
- Sphinx documentation:
  - user - installation, usage
  - developer - testing, releasing
- Makefile
- Tox configuration
- Travis CI configuration
- REST API client:
  - `GET /api/v1/info`
  - `GET /api/v1/links`

CHAPTER 5

Testing

See also:

- *Installation*

5.1 Environment and requirements

Tox is used to manage test virtualenvs, and is the only tool needed to run static analysis and unitary tests, as it will create the appropriate testing virtualenvs on-the-fly.

```
(shaarli) $ pip install -r requirements/ci.txt
```

Nevertheless, in case you want to install *test*, *development* and *documentation* dependencies, e.g. for editor integration or local debugging:

```
(shaarli) $ pip install -r requirements/dev.txt
```

5.2 Tools

The documentation is written in reStructuredText, using the Sphinx generator.

Coding style is checked using tools provided by the Python Code Quality Authority:

- isort: check import ordering and formatting
- pycodestyle: Python syntax and coding style (see PEP8)
- pydocstyle: docstring formatting (see PEP257)
- pylint: syntax checking using predefined heuristics

Tests are run using the pytest test framework/harness, with the following plugins:

- pytest-pylint: pylint integration

- pytest-cov: coverage integration

5.3 Running the tests

To renew test virtualenvs, run all tests and generate the documentation:

```
$ tox -r
```

To run specific tests without renewing the corresponding virtualenvs:

```
$ tox -e py34 -e py36
```

To run specific tests and renew the corresponding virtualenv:

```
$ tox -r py35
```

CHAPTER 6

Releasing

Reference:

- Python Packaging User Guide
  - Packaging and Distributing Projects
- TestPyPI Configuration

6.1 Environment and requirements

twine is used to register Python projects to PyPI and upload release artifacts:

- `PKG-INFO`: project description and metadata defined in `setup.py`
- `sdist`: source distribution tarball
- `wheel`: binary release that can be platform- and interpreter- dependent

Development libraries need to be installed to build the project and upload artifacts (see *Testing*):

```
(shaarli) $ pip install -r requirements/dev.txt
```

6.2 PyPI and TestPyPI configuration

Danger: Once uploaded, artifacts cannot be overwritten. If something goes wrong while releasing artifacts, you will need to bump the release version code and issue a new release.

It is safer to test the release process on TestPyPI first; it provides a sandbox to experiment with project registration and upload.

6.2.1 ~/.pypirc

```
[distutils]
index-servers=
    pypi
    testpypi

[pypi]
repository = https://upload.pypi.org/legacy/
username = <PyPI username>

[testpypi]
repository = https://test.pypi.org/legacy/
username = <TestPyPI username>
password = <TestPyPI password>
```

6.3 Releasing shaarli-client

6.3.1 Checklist

- install Python dependencies
- setup PyPI and TestPyPI:
  - create an account on both servers
  - edit `~/.pypirc`
  - register the project on both servers
- get a *GnuPG* key to sign the artifacts
- double check project binaries and metadata
- tag the new release
- build and upload the release on TestPyPI
- build and upload the release on PyPI

Tip: A `Makefile` is provided for convenience, and allows to build, sign and upload artifacts on both PyPI and TestPyPI.

6.3.2 TestPyPI

```
(shaarli) $ export IDENTITY=<GPG key ID>
(shaarli) $ make test_release
```

6.3.3 PyPI

```
(shaarli) $ export IDENTITY=<GPG key ID>
(shaarli) $ make release
```

CHAPTER 7

GnuPG

7.1 Introduction

7.1.1 PGP and GPG

Gnu Privacy Guard (GnuPG) is an Open Source implementation of the Pretty Good Privacy (OpenPGP) specification. Its main purposes are digital authentication, signature and encryption.

It is often used by the FLOSS community to verify:

- Linux package signatures: Debian SecureApt, ArchLinux Master Keys
- SCM releases & maintainer identity

7.1.2 Trust

To quote Phil Pennock, the author of the SKS key server:

> You MUST understand that presence of data in the keyserver (pools) in no way connotes trust. Anyone can generate a key, with any name or email address, and upload it. All security and trust comes from evaluating security at the "object level", via PGP Web-Of-Trust signatures. This keyserver makes it possible to retrieve keys, looking them up via various indices, but the collection of keys in this public pool is KNOWN to contain malicious and fraudulent keys. It is the common expectation of server operators that users understand this and use software which, like all known common OpenPGP implementations, evaluates trust accordingly. This expectation is so common that it is not normally explicitly stated.

Trust can be gained by having your key signed by other people (and signing their keys back, too :-)), for instance during key signing parties:

- The Keysigning Party HOWTO
- Web of Trust

7.2 Generate a GPG key

- Generating a GPG key for Git tagging (StackOverflow)
- Generating a GPG key (GitHub)

7.2.1 gpg - provide identity information

```
$ gpg --gen-key

gpg (GnuPG) 2.1.6; Copyright (C) 2015 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg2 --full-gen-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Marvin the Paranoid Android
Email address: marvin@h2g2.net
You selected this USER-ID:
    "Marvin the Paranoid Android <marvin@h2g2.net>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
```

7.2.2 gpg - entropy interlude

At this point, you will:

- be prompted for a secure password to protect your key (the input method will depend on your Desktop Environment and configuration)
- be asked to use your machine's input devices (mouse, keyboard, etc.) to generate random entropy; this step *may take some time*

7.2.3 gpg - key creation confirmation

```
gpg: key A9D53A3E marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
pub   rsa2048/A9D53A3E 2015-07-31
Key fingerprint = AF2A 5381 E54B 2FD2 14C4  A9A3 0E35 ACA4 A9D5 3A3E
uid       [ultimate] Marvin the Paranoid Android <marvin@h2g2.net>
sub   rsa2048/8C0EACF1 2015-07-31
```

7.2.4 gpg - submit your public key to a PGP server

```
$ gpg --keyserver pgp.mit.edu --send-keys A9D53A3E
gpg: sending key A9D53A3E to hkp server pgp.mit.edu
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search