history Documentation

Release 0.1.0

Donald Stufft

Sep 27, 2017

Contents

history generates history/changelog files for a Project

CHAPTER 1

Installation

You can install history with `pip`:

```
$ pip install history
```

Development

As an open source project, history welcomes contributions of all forms. The sections below will help you get started.

File bugs and feature requests on our issue tracker on GitHub. If it is a bug check out what to put in your bug report.

Getting started

Working on history requires the installation of a small number of development dependencies. These are listed in `dev-requirements.txt` and they can be installed in a virtualenv using pip. Once you've installed the dependencies, install history in `editable` mode. For example:

```
$ # Create a virtualenv and activate it
$ pip install --requirement dev-requirements.txt
$ pip install --editable .
```

You are now ready to run the tests and build the documentation.

Running tests

history unit tests are found in the `tests/` directory and are designed to be run using pytest. pytest will discover the tests automatically, so all you have to do is:

```
$ py.test
...
62746 passed in 220.43 seconds
```

This runs the tests with the default Python interpreter.

You can also verify that the tests pass on other supported Python interpreters. For this we use tox, which will automatically create a virtualenv for each supported Python version and run the tests. For example:

```
$ tox
...
ERROR:   py26: InterpreterNotFound: python2.6
 py27: commands succeeded
ERROR:   pypy: InterpreterNotFound: pypy
ERROR:   py32: InterpreterNotFound: python3.2
 py33: commands succeeded
 docs: commands succeeded
 pep8: commands succeeded
```

You may not have all the required Python versions installed, in which case you will see one or more `InterpreterNotFound` errors.

Building documentation

history documentation is stored in the `docs/` directory. It is written in reStructured Text and rendered using Sphinx.

Use tox to build the documentation. For example:

```
$ tox -e docs
...
docs: commands succeeded
congratulations :)
```

The HTML documentation index can now be found at `docs/_build/html/index.html`.

Submitting patches

- Always make a new branch for your work.
- Patches should be small to facilitate easier review. Studies have shown that review quality falls off as patch size grows. Sometimes this will result in many small PRs to land a single large feature.
- Larger changes should be discussed in a ticket before submission.
- New features and significant bug fixes should be documented in the *Changelog*.

If you believe you've identified a security issue in history, please follow the directions on the *security page*.

Code

When in doubt, refer to **PEP 8** for Python code. You can check if your code meets our automated requirements by running `flake8` against it. If you've installed the development requirements this will automatically use our configuration. You can also run the `tox` job with `tox -e pep8`.

Write comments as complete sentences.

Every code file must start with the boilerplate notice of the Apache License. Additionally, every Python code file must contain

```
from __future__ import absolute_import, division, print_function
```

Tests

All code changes must be accompanied by unit tests with 100% code coverage (as measured by the combined metrics across our build matrix).

Documentation

All features should be documented with prose in the `docs` section.

When referring to a hypothetical individual (such as "a person receiving an encrypted message") use gender neutral pronouns (they/them/their).

Docstrings are typically only used when writing abstract classes, but should be written like this if required:

```
def some_function(some_arg):
    """
    Does some things.

    :param some_arg: Some argument.
    """
```

So, specifically:

- Always use three double quotes.
- Put the three double quotes on their own line.
- No blank line at the end.
- Use Sphinx parameter/attribute documentation syntax.

Reviewing and merging patches

Everyone is encouraged to review open pull requests. We only ask that you try and think carefully, ask questions and are excellent to one another. Code review is our opportunity to share knowledge, design ideas and make friends.

When reviewing a patch try to keep each of these concepts in mind:

Architecture

- Is the proposed change being made in the correct place?

Intent

- What is the change being proposed?
- Do we want this feature or is the bug they're fixing really a bug?

Implementation

- Does the change do what the author claims?
- Are there sufficient tests?
- Has it been documented?
- Will this change introduce new bugs?

Grammar and style

These are small things that are not caught by the automated style checkers.

- Does a variable need a better name?
- Should this be a keyword argument?

Security

We take the security of history seriously. If you believe you've identified a security issue in it, please report it to `donald@stufft.io`. Message may be encrypted with PGP using key fingerprint `7C6B 7C5D 5E2B 6356 A926 F04F 6E3C BCE9 3372 DCFA` (this public key is available from most commonly-used key servers).

Once you've submitted an issue via email, you should receive an acknowledgment within 48 hours, and depending on the action to be taken, you may receive further follow-up emails.

Changelog

Index

P