dennis Documentation

Release 0.9

Will Kahn-Greene

Feb 17, 2017

Contents

Dennis is a set of utilities for working with PO files to ease development and improve quality. Translate POT files to find problems with localization in your code. Lint PO files for common problems like variable formatting, mismatched HTML, missing variables, etc.

It includes the following subcommands:

- **lint**: Lints PO and POT files for problems including errors that can cause your production system to crash and problems in strings that can lead to poor translations.

  The system allows for defining other variable formats.

- **status**: Get a high-level status of a PO file including a list of unstranslated strings.

- **translate**: Translates strings in PO files into something else! Comes with an HTML extractor (tokenizes strings so that only the text is translated) and a bunch of translations like Pirate!.

  This is helpful for l10n testing, development, finding unicode/layout problems, amazing your friends, hilarious April 1st shenanigans, etc.

  Specify the tokenizer/transform pipeline you want to use that combines things. Zombie? Sure! Shouty Zombie? Ok! Manic shouty Dubstep? Bring it on!

  This also works on strings passed in as command line arguments and as stdin—it doesn't have to be a PO file or in a PO format format. For example, Dennis uses Dennis to translate all Dennis commit messages into Pirate!. That's how cool Dennis is!

Quick start

Install:

```
$ pip install dennis
$ pip install blessings  # Optional for prettier output
```

Lint a PO file for problems:

```
$ dennis-cmd lint locale/fr/LC_MESSAGES/messages.po
```

Lint all your PO files for errors:

```
$ dennis-cmd lint --errorsonly locale/
```

Lint a POT file for problems:

```
$ dennis-cmd lint locale/templates/LC_MESSAGES/messages.pot
```

Translate a PO file in place into Pirate!:

```
$ dennis-cmd translate --pipeline=html,pirate \
    locale/xx/LC_MESSAGES/messages.po
```

Get help:

```
$ dennis-cmd
```

Project details

Code http://github.com/willkg/dennis

Documentation http://dennis.rtfd.org/

Issue tracker https://github.com/willkg/dennis/issues

License BSD 3-clause; see LICENSE file

Donate gittip

User guide

What's new in Dennis

- *Version 0.9.0: February 2nd, 2017*
- *Version 0.8.0: January 3rd, 2017*
- *Version 0.7.0: October 2nd, 2015*
- *Version 0.6.1: December 18th, 2014*
- *Version 0.6: December 16th, 2014*
- *Version 0.5: August 24th, 2014*
- *Version 0.4.3: August 1st, 2014*
- *Version 0.4.2: May 13th, 2014*
- *Version 0.4.1: May 9th, 2014*
- *Version 0.4: May 1st, 2014*
- *Version 0.3.11: April 16th, 2014*
- *Version 0.3.10: October 25th, 2013*
- *Version 0.3.9: October 17th, 2013*
- *Version 0.3.8: October 16th, 2013*
- *Version 0.3.7: October 15th, 2013*
- *Version 0.3.6: September 19th, 2013*
- *Version 0.3.5: September 17th, 2013*
- *Version 0.3.4: July 30th, 2013*
- *Version 0.3.3: July 29th, 2013*
- *Version 0.3.2: July 23rd, 2013*
- *Version 0.3.1: July 15th, 2013*
- *Version 0.3: July 8th, 2013*

Version 0.9.0: February 2nd, 2017

- dfc0b68 Remove th' django shims

  This is a backwards-incompatible change. If you need them back, then you should pull the code and add it to your project.

  Why remove them? They weren't tested in any way, possibly didn't work with new Django versions and weren't really supported.

- 033d88d Remove click varrsion pinnin'

- 9241ff7 Fix filename printin'

Version 0.8.0: January 3rd, 2017

- b0705f4 Clean up pytest code and drop Python 2.6 bits

- d27790b Fix th' --varformat flag to alloww nay formats (#83)

- 3bf0929 Switch travis sudo flag

- 990c842 Update requirements

- 0d00ad4 Add travis supparrt

- 528fcc1 Fix support for Python 3.5 (Thanks John Vandenberg!)

- 700490d Add Travis CI testing (#88)

- a59a9bb Fix translation o' plurals (#79)

- aca57f6 Fix false positive wit' InvalidVarsLintRule (#78)

- 581f230 Add note about --check-headerr flag to recipes

- bce6308 Fix PythonBraceFormat regexp to handle spaces

- 7016ee4 Add .cache to .gitignore

- 05a4e14 Collapse th' whitespace in text in th' html transform

- 75fa600 Prepare fer 0.8 development

- 070808d Add additional dev-relat'd packages

Version 0.7.0: October 2nd, 2015

- 2d27a8c Add lint rule fer bad format charactarrs (#68)

- 2bc1053 Missin' python-format variables is an errorrr (#57)

- 8e178c7 Fix notype test to handle more cases (#63)

- 90ab98d Implement rule exclusion (#60)

- caabbbb Rewrite rule spec varrification to work correctly (#61)

- 7e2f7f0 Add --showfuzzy to status command (#64)

- 02ea679 Add untranslat'd word counts to status command (#55)

- 713288b Change Varr to Format and use gettext names (#48)

- c3c49b9 Handle th' standalone } case (#56)

Version 0.6.1: December 18th, 2014

Changes

- 39208bd Rewrite poentry block parserr (#54)
- b21d553 Handle HTMLParseErrorr in MismatchedHTMLLintRule (#53)
- e719886 Update lintin' wit' warnings we add'd in 0.6
- 6312d33 Tweak a bunch o' project summary language

Version 0.6: December 16th, 2014

Notes

- Adds click as a dependency.
- Adds a line reporter for the linter so Dennis can be used as a linting plugin.
- Adds line numbers to lint output so you can more easily find the problematic strings.

Changes

- 9f9f42b Add exception handlin' text (#41)
- fd15fe8 Add double transform (#44)
- dcd3e7f Fix th' Django command shims
- 1606887 Rewrite command line code wit' click (#51)
- c34d77e Fix pyflakes issues
- 905ce05 Change "dennis" to "Dennis"
- a6d49a8 Nix bin/dennis-cmd for setuptools console entrypoint
- be9c867 Merge lint and linttemplate commands (#50)
- 92f2037 Add line numbers to output and line reporter (#47)
- 37cad18 Showw entire poentry in linttemplate (#46)

Version 0.5: August 24th, 2014

Changes

- 8dddfb7 Add MismatchedHTMLLintRule (#36)
- b31c094 Minorr code cleanup
- 9353f5b Fix bugs when runnin' wit' Python 3
- 4883e52 Add template linterr (#39)

Version 0.4.3: August 1st, 2014

Changes

- ead33d3 Add UnchangedLintRule (#36)
- fde6d9a Add BlankLintRule (#36)

- 73b1f35 Fix W202 regarding missing variables in pluralistic strings (#38)

Version 0.4.2: May 13th, 2014

Changes

- 06e4b6d Fix utf8_args decoratorr

Version 0.4.1: May 9th, 2014

Changes

- 831af1a Fix lint output regarding UnicodeEncodeErrors (#37)

Version 0.4: May 1st, 2014

Changes

- Tweak Python 3 support
- c42b7e8 Overhaul linter for finer-grained linting
- 3e1cc1d Add extracted-comment-based lint rule ignoring so you can easily ignore false positives on a string-by-string basis (#34)

Version 0.3.11: April 16th, 2014

Changes

- 0c2e5a9 Fix foo} with missing right curly brace (#33)
- Python 3 support (#30)
- 6f60b00 Add reverse transform

Version 0.3.10: October 25th, 2013

Changes

- f874578 Add status command
- 8b99cfe Add zombie transform
- fb319e3 Fix lint command to handle multiple files

Version 0.3.9: October 17th, 2013

Changes

- 3852fac Mediocre tweak to betterr handle urlencodin' (#27)
- 3c65a1d Don't considerr %% a valid Python variable (#28)

Version 0.3.8: October 16th, 2013

Changes

- ac9edf0 Fix problem identifyin' mismatch'd errors in plurals (#25) (Thanks Mike!)

Version 0.3.7: October 15th, 2013

Changes

- 5787e12 Add dubstep translator
- fd90046 Add shims so you can easily use with django

Version 0.3.6: September 19th, 2013

Changes

- 56b7372 Fix false positives like (%(count)s) in malformed lint rule. (#21) (Thanks Kumar!)

Version 0.3.5: September 17th, 2013

Changes

- b432e1b Fix rules default – Running the linter with the default set of rules will now include malformed variable linting.
- 72083f9 Improve detect missing } with python vars
- b8f3776 Improve linting docs – It includes a list of lint rules and what they do.
- 6d9bac5 Detect missing } in Python formatting vars (#20) (Thanks Kumar!)
- 1a10c35 Fix detection of malformed formatting token at end of string

Version 0.3.4: July 30th, 2013

Changes

- 8a1d4a8 Make sure to lint only translated non-fuzzy strings

Version 0.3.3: July 29th, 2013

Backwards-incompatible changes

- cf668a3 Rename var_types to just var

  If you were doing something like:

  ```
  $ dennis-cmd lint --types=python ...
  $ dennis-cmd translate --types=python ...
  ```

  that `--types` argument is now `--vars`.

Changes

- 952245c Tweak lint output to betterr do errorsonly

- cc63144 Fix lint output issues

- 6ee94a3 Overhaul linter to support multiple lint rules (#18)

Version 0.3.2: July 23rd, 2013

Changes

- c778532 Add haha transform

- e41bca8 Add –errorsonly flag to linter (#16)

- 759352d Fix UnicodeEncodeErrors wit' translate

Version 0.3.1: July 15th, 2013

Changes

- c600064 Handle invalid .po files (#10)

- 52f81f9 Fix lint output so it's utf-8 (#11) (Thanks Mike!)

- 7da6add Tweak translator to allow for translating stdin (#13)

- a5e3556 Add empty, xxx, anglequote and shouty transforms

- 8cb1f2a Add redacted transform

- Documentation

- Bug fixes

Version 0.3: July 8th, 2013

Changes

- Initial writing. Yay!

What happened to 0.1 and 0.2? I skipped them.

Translating!

Help

```
$ dennis-cmd translate --help
```

This lists available transforms, variable formats and other options.

Summary

Dennis can translate the strings in your PO file. For example, this does the default which extracts text from HTML strings and translates that text into Pirate:

```
$ dennis-cmd translate messages.po
```

Note: This translates the `messages.po` file in-place. If you don't want that, then copy the file and translate the copy.

You can also translate strings on the command line:

```
$ dennis-cmd translate -s "Dennis is my friend"
```

You can translate stuff from stdin:

```
$ echo "Dennis can see the future" | dennis-cmd translate -
```

You can change the pipeline to use one of many exciting transforms. For example, this is extra-piraty and shouty!:

```
$ dennis-cmd translate --pipeline=pirate,pirate,shouty \
    -s "Dennis can make hard boiled eggs boil faster"
```

Dennis can translate around variable tokens in strings. By default, it translates around Python variable forms. You can specify other variable formats to translate around:

```
$ denis-cmd translate --varformat=python-format,python-brace-format
```

Note: The infrastructure is there for handling other variable formats, but only Python formats have been coded. Help me add additional formats that are used in your gettext strings!

For help and a list of variable formats and transforms, do this:

```
$ dennis-cmd translate
```

For more about pipelines, see *Pipelines*.

For more about other transforms, see *Transforms*.

For more about extending Dennis to do dirty things, see *API*.

Transforms

Dennis currently supports the following transforms:

name	description
anglequote	Encloses string in unicode angle quotes.
double	Doubles all vowels in a string.
dubstep	Translates text into dubstep. It's an experience.
empty	Returns empty strings.
haha	Adds haha! before sentences in a string.
pirate	Translates text into Pirate!
redacted	Redacts everything.
reverse	Reverses strings.
shouty	Translates into all caps.
xxx	Adds xxx before and after lines in a string.
zombie	Zombie.

Aditionally, there's the html transform which extracts the bits to be translated, but doesn't do any translation itself:

---

name	description
html	Tokenizes HTML bits so only text is translated.

anglequote

The anglequote transform adds unicode angle quotes to the beginning and end of strings. This helps to make sure your code handles unicode strings and also some layout issues like when strings are cut off or overlapping.

double

The double transform doubles all vowels in the string.

dubstep

The dubstep transform is an experience.

empty

The empty transform nixes the string.

OMG! Why?!

This is helpful for building .pot files from .po files. Also, it's sort of interesting to see a ui with no text in it.

haha

Haha! Adds "Haha!" before sentences in a string. Haha! The exclamation point is a non-ASCII character, so this is both fun and tests unicode handling!

pirate

The Pirate! translation has the following properties:

1. it's longer than the English equivalent (tests layout issues)
2. it's different than the English equivalent (tests missing gettext calls)
3. every string ends up with a non-ascii character (tests unicode handling)
4. looks close enough to the English equivalent that you can quickly figure out what's wrong (doesn't test your reading comprehension)

redacted

Xxx xxxxxxxx xxxxxxxxx xxxxxxx xxxxxxxxxx.

reverse

.LTR rof lufpleh semitemos si hcihw sgnirts sesreveR

---

shouty

THE SHOUTY TRANSFORM MAKES THINGS IN ALL ASCII UPPERCASE. SHOUTY SHOUTY SHOUTY.

xxx

The xxx transform wraps all lines in strings with xxx.

zombie

ThHA zHRmbARHA HGMZanRZZRHRMZm HGNMMZnRZ HGHAZBHG ARnHGHR zHRmbARHA!

html

The html transform extracts strings from HTML to be translated. This includes any TEXT nodes as well as the text in alt and title attributes.

Pipelines

A pipeline consists of one or more transforms connected together. The output of one transform is the input of the next transform.

Each transform takes an iterable of Tokens and outputs an iterable of Tokens. In this way, you can build your pipeline however you like. For more on this and how to build your own transforms, see *API*.

Sample string: "Dennis can make your dreams come true."

Example pipelines:

- `pirate`

  Translates into Pirate!

  Sample string:

  ```
  <b>Dennis can make yerr dreams come true.</b> ye scalleywag!
  ```

  Note that this isn't extracting HTML, so it just considers that whole thing a single string.

- `shouty,pirate`

  Capitalizes everything in the string (including the html) then runs that through pirate.

  Sample string:

  ```
  <B>DENNIS CAN MAKE YOUR DREAMS COME TRUE.</B> ye scalleywag!
  ```

  Note that this isn't extracting HTML, so it just considers that whole thing a single string.

- `html,pirate,pirate,pirate,shouty`

  Extracts text from HTML to be translated, runs it through pirate multiple times, then runs it through shouty which results in an extra Piraty shouty string

  Sample string:

```
<b>DENNIS CAN MAKE YARRRRR DREAMS COME TRUE PREPARE TO BE BOARD'D!
YE LANDLUBBARRS! MATEY!.</b>
```

- `empty,anglequote`

  Woah—where'd the words go? It's like a ghost-town of a ui.

  Sample string:

  ```
  «»
  ```

Linting!

Help

```
$ dennis-cmd lint --help
```

This will list the available lint rules and codes, the variable formats and any additional options available.

Summary

Dennis can lint your translated PO files for Python formatting token issues:

```
$ dennis-cmd lint messages.po
```

This runs multiple lint rules on all the strings in the PO file generating a list of errors and a list of warnings.

Wait, but that's ugly and hard to read! If you install `blessings`, it comes colorized and really easy to parse. All hail blessings!

Dennis can also lint your POT files and point out issues that can result in poor translations. You can lint POT files like this:

```
$ dennis-cmd lint messages.pot
```

Skipping rules string-by-string

In the extracted comment, you can tell Dennis to ignore lint rules (or all lint rules).

Ignore everything:

```
#. dennis-ignore: *
msgid "German makes up 10% of our visitor base"
msgstr "A német a látogatóbázisunk 10%-át teszi ki"
```

Ignore specific rules (comma-separated):

```
#. dennis-ignore: E101,E102,E103
msgid "German makes up 10% of our visitor base"
msgstr "A német a látogatóbázisunk 10%-át teszi ki"
```

Ignore everything, but note the beginning of the line is ignored by Dennis so you can tell localizers to ignore the ignore thing:

```
#. localizers--ignore this comment. dennis-ignore: *
msgid "German makes up 10% of our visitor base"
msgstr "A német a látogatóbázisunk 10%-át teszi ki"
```

Warnings and Errors

What's a warning?

Warnings indicate the translated string is either outdated or a poor translation, but the string is fine in the sense that it won't kick up an error in production.

For example, say the original string has a variable, but the translated string doesn't use that variable.

That's not great and probably means the translated string needs to be updated, but it won't throw an error in production.

What's an error?

Errors indicate problems with the translated string that will cause an error to be thrown. These should get fixed pronto.

For example, when the translated string has a Python variable that's not in the original string. When this string is interpolated, it will kick up a Python error. That causes the software to die, users to be unhappy, tires to go flat, people to work on weekends, mass hysteria, etc. No one likes that. I don't like that. You probably don't like that, either.

Table of Warnings and errors

PO files

Code	Description

E101	Malformed variable missing type
	Only checks python-format variables.
	Example (Python):
	`Error: malformed variables: %(count)`
	`msgid: "%(count)s view"`
	`msgstr: "%(count) view"`
	`>>> "%(count) view" % {"count": 5}`
	`Traceback (most recent call last):`
	`  File "<stdin>", line 1, in <module>`
	`ValueError: unsupported format character 'v' (0x76`
	`>>>`

E102	Malformed variable missing right curly-brace
	For example `{foo` with missing `}`.
	Only checks python-brace-format variables.
	Example (Python):
	`Error: malformed variables: {foo bar baz`
	`msgid: "{foo} bar baz"`
	`msgstr: "{foo bar baz"`
	`>>> "{foo bar baz".format(foo="some thing")`
	`Traceback (most recent call last):`
	`  File "<stdin>", line 1, in <module>`
	`ValueError: unmatched '{' in format`
	`>>>`

E103	Malformed variable missing left curly-brace
	For example `foo}` with missing `{`.
	Only checks python-brace-format variables.
	Example (Python):
	`Error: malformed variables: foo}`
	`msgid: "{foo} bar baz"`
	`msgstr: "foo} bar baz"`
	`>>> "foo}".format(foo="some thing")`
	`Traceback (most recent call last):`
	`  File "<stdin>", line 1, in <module>`
	`ValueError: Single '}' encountered in format strin`
	`>>>`

E104	Bad format character
	For example `%a`.
	Only checks python-format variables.
	Example (Python):
	`Error: bad format character: %a`
	`msgid: "%s foo"`
	`msgstr: "%a FOO"`
	`>>> "%a" % (1, 2, 3)`
	`Traceback (most recent call last):`
	`  File "<stdin>", line 1, in <module>`
	`ValueError: unsupported format character 'a' (0x61`
	`>>>`

E201	Invalid variables in translated string
	There are formatting variable tokens in the *translated* string that aren't in the original string.
	Example:
	`Error: mismatched: invalid variables: {helpurl}`
	`msgid: "You can find help at {url}"`

Chapter 3. User guide

POT files

Code	Description

W500	Hard to read variable name
	There are a series of letters and numbers which are hard to distinguish from one another: o, O, 0, l, 1. It's not uncommon for a hard-working translator to misread and use the wrong character.
	Example:
	`Warning: hardtoread: hard to read variable name "l`
	`msgid: "Title: {l}"d help at {url}"`
	`msgstr: ""`

W501	One character variable name Using a one character variable name doesn't give enough context to the translator about what's being put in that variable. Example: `Warning: onechar: one character variable name: "{t` `msgid: "{t}	{c}"` `msgstr: ""`

W502	Multiple unnamed variables
	Having one unnamed variable is ok since it's not order-dependent. However, having more than one unnamed variable means those variabes must occur in an order specified outside of the string. This creates problems with RTL languages and any other language that might need to change the order of the variables to create a translation that makes sense.
	Example:
	`Warning: multiple variables with no name`
	`msgid: "%s replies to %s"`
	`msgstr: ""`

Statusing!

Help

```
$ dennis-cmd status --help
```

Summary

Sometimes you just want to see the high-level status of a PO file and also maybe see the list of untranslated strings.

You can do that with Dennis:

```
$ dennis-cmd status messages.po
```

This will spit out riveting PO metadata and strings statistics like the total number of translated strings, untranslated strings, fuzzy strings and percentage translated.

Additionally, you can tell Dennis to show you all the untranslated strings:

```
$ dennis-cmd status --showuntranslated messages.po
```

Now you can verify that translation has been completed on a PO file without reading through the PO file.

API

Dennis is actually a library with a commandline interface frontend. You can use the library without using the command line at all.

This could be useful for linting strings on a translation system, etc.

This documentation needs to be written, but I'm going to wait until the core stabilizies.

Recipes

- *linting all your PO files*
- *linting your POT file*
- *translate PO file to find problems with localization*
- *selective MO compiling*
- *commit-msg git hook*
- *convert your web page into Pirate for April fools day*

linting all your PO files

Sometimes it's good to just get a look at all the PO files and make sure they're ok:

```
$ dennis-cmd lint locale/
```

It'll give you a summary at the end.

linting your POT file

Linting your POT file can reduce the number of issues that translators will stumble over. It's good to lint it before you push new strings to translate:

```
$ dennis-cmd lint locale/templates/LC_MESSAGES/messages.pot
```

translate PO file to find problems with localization

Use the Dennis translator to find l10n issues in your application without having to bother your translators.

For example, this translates strings into Pirate for a web application written in Python:

```
1  #!/bin/bash
2
3  mkdir -p locale/xx/LC_MESSAGES
4  cp locale/templates/LC_MESSAGES/messages.pot \
5      locale/xx/LC_MESSAGES/messages.po
6
7  dennis-cmd translate \
8      --pipeline=html,pirate \
9      --varsformat=python-format,python-brace-format \
10     locale/xx/LC_MESSAGES/messages.po
```

Now view your application with the xx locale and behold it's Piratey wonderfulness!

There are a variety of transforms which have different properties and suss out different kinds of localization problems.

selective MO compiling

The Dennis linter returns an exit code of 1 if the file(s) it's linting have errors. You can trivially use this to selectively compile PO files into MO files iff they are error free [1]:

```
1  #!/bin/bash
2
3  for pofile in `find locale/ -name "*.po"`
4  do
5      dir=`dirname "$pofile"`
6      stem=`basename "$pofile" .po`
7
8      dennis-cmd lint --errorsonly "$pofile"
9      if [ $? -ne 0 ]
10     then
11         echo "ERRORZ!!! Not compiling $pofile!"
12
13     else
14         msgfmt -o "${dir}/${stem}.mo" "$pofile"
15     fi
16 done
```

commit-msg git hook

You can automatically translate all future commit messages for your git project by creating a commit-msg hook like this:

```
1  #!/bin/bash
2
3  # Pipe the contents of the commit message file through dennis to
4  # a temp file, then copy it back.
5  (cat < $1 | dennis-cmd translate - > $1.tmp) && mv $1.tmp $1
```

---

[1] Dennis doesn't lint the Plural-Forms headers, so it's still possible for the resulting .mo file to have errors. You can check for errors in the headers by passing the --check-header flag to msgfmt.

```
6
7   # We always exit 0 even if the dennis-cmd fails. If the dennis-cmd
8   # fails, you get your original commit message. No one likes it when
9   # shenanigans break your stuff for realz.
10  exit 0;
```

convert your web page into Pirate for April fools day

The Dennis translator can take content from stdin. Translate entire HTML pages:

```
1   #!/bin/bash
2
3   (cat < "$1" | dennis-cmd translate --pipeline=html,pirate -) > "pirate_$1"
```

Or show how you really feel about April fools day on the Internet:

```
#!/bin/bash

(cat < "$1" | dennis-cmd translate --pipeline=html,haha -) > "haha_$1"
```

Project guide

Hacking on Dennis

This covers setting up Dennis to hack on. If you're interested in using Dennis, but not hacking on it, then this probably isn't going to be interesting to you.

Install Dennis and dependencies for development

1. Clone the repository from https://github.com/willkg/dennis/

2. Create a virtual environment

3. Run: `python setup.py develop`

4. Install some other bits: `pip install -r requirements-dev.txt`

This should get you up and running.

Helping out

The non-exhaustive list of things to do are in the issue tracker.

If you want to write some code or fix a bug or add some docs or in some way contribute to Dennis, please do so using the following process:

1. Tell me what you're planning to do before you do it. Preferably in a comment in the issue tracker on a relevant issue. If not as a comment, then email me.

   After I've been informed and given approval, continue!

2. Create a new branch for your changes.

3. Make your changes!

4. Update documentation or write new documentation for the changes you've made.

5. Update the tests or write new tests for the changes you've made.

6. Submit a patch.

   To make it easier for me to maintain Dennis, all changes should either:

   (a) be submitted as a pull request in Github, or

   (b) emailed to me as an appropriately formatted patch

If you don't know how to do either, then maybe you can find someone to help you out.

That's it!

Anyone who contributes code, tests or docs (in other words, has a git commit with their name on it) get added to the CONTRIBUTORS file. Yay!

Tests

Tests are in `tests/`. We use py.test as the test runner.

To run the tests, do:

```
$ py.test
```

Please write tests for changes you make.

Documentation

Documentation is in `docs/`. We use Sphinx as the documentation generator.

To build the docs, do:

```
$ cd docs/
$ make html
```

Please make changes to the documentation as required by the changes you make.

Release howto

1. Check out master tip.

2. Check to make sure `setup.py` and `requirements-dev.txt` files have correct versions of requirements.

3. Update version number in `dennis/__init__.py`

   (a) Set `__version__` to something like `0.4`.

   (b) Set `__releasedate__` to something like `20120731`.

4. Update `CHANGELOG`. Usually, I do something like:

   ```
   git log --oneline v0.4..HEAD
   ```

   replacing `v0.4` with the most recent tag. Then I copy and paste that, remove uninteresting lines and add a header.

5. Verify correctness.

   (a) Run the tests with `tox`.

   (b) Review the `README.rst`.

   (c) Build the docs and review them.

6. Tag the release:

   ```
   git tag -a v0.4
   ```

   Copy the last section of the `CHANGELOG` into the tag commit message.

7. Push everything:

```
git push --tags official master
```

8. Update PyPI:

```
rm -rf dist/*
python setup.py sdist bdist_wheel
twine upload dist/*
```

9. Announce the release.

License

Copyright (c) 2013-2014 Will Kahn-Greene All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, IN-CIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSI-NESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CON-TRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAM-AGE.

Contributors

- Will Kahn-Greene
- Mike Cooper
- James Socol
- Ricky Rosario
- Dave Dash
- Kumar McMillan
- John Vandenberg

Indices and tables

- genindex
- modindex
- search