# dennis Documentation

***Release 0.6***

**Will Kahn-Greene**

December 16, 2014

Contents

Dennis is a command line tool for translating and linting .po files.

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. For example, Dennis uses Dennis to translate all Dennis commit messages into Pirate!. That's how cool Dennis is!

CHAPTER 1

# 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
```

CHAPTER 2

# 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

CHAPTER 3

# User guide

## 3.1 What's new in Dennis



### 3.1.1 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)

### 3.1.2 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)

### 3.1.3 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)

### 3.1.4 Version 0.4.2: May 13th, 2014

**Changes**

- 06e4b6d Fix utf8_args decoratorr

### 3.1.5 Version 0.4.1: May 9th, 2014

**Changes**

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

### 3.1.6 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)

### 3.1.7 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

### 3.1.8 Version 0.3.10: October 25th, 2013

**Changes**

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

### 3.1.9 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)

### 3.1.10 Version 0.3.8: October 16th, 2013

**Changes**

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

### 3.1.11 Version 0.3.7: October 15th, 2013

**Changes**

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

### 3.1.12 Version 0.3.6: September 19th, 2013

**Changes**

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

### 3.1.13 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

### 3.1.14 Version 0.3.4: July 30th, 2013

**Changes**

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

### 3.1.15 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)

### 3.1.16 Version 0.3.2: July 23rd, 2013

**Changes**

- c778532 Add haha transform
- e41bca8 Add –errorsonly flag to linter (#16)
- 759352d Fix UnicodeEncodeErrors wit' translate

### 3.1.17 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

### 3.1.18 Version 0.3: July 8th, 2013

**Changes**

- Initial writing. Yay!

What happened to 0.1 and 0.2? I skipped them.

## 3.2 Translating!

### 3.2.1 Help

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

This lists available transforms, variable formats and other options.

### 3.2.2 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 --vars=pysprintf,pyformat
```

---

**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*.

### 3.2.3 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.

### 3.2.4 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:

  «»

## 3.3 Linting!

### 3.3.1 Help

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

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

### 3.3.2 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
```

### 3.3.3 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"
```

### 3.3.4 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

These are for `.po` files:

| Code | Description |
|---|---|
| E101 | Malformed variable missing type<br>Only checks pythonpercent variables.<br>Example (Python):<br>`Error: malformed variables: %(count)`<br>`msgid: "%(count)s view"`<br>`msgstr: "%(count) view"`<br>`>>> "%(count) view" % {"count": 5}`<br>`Traceback (most recent call last):`<br>`  File "<stdin>", line 1, in <module>`<br>`ValueError: unsupported format character 'v' (0x76`<br>`>>>`<br>`Traceback (most recent call last):`<br>`  File "<stdin>", line 1, in <module>`<br>`ValueError: unsupported format character 'v' (0x76` |
| E102 | Malformed variable missing right curly-brace<br>For example `{foo` with missing `}`.<br>Only checks pythonformat variables.<br>Example (Python):<br>`Error: malformed variables: {foo bar baz`<br>`msgid: "{foo} bar baz"`<br>`msgstr: "{foo bar baz"`<br>`>>> "{foo bar baz".format(foo="some thing")`<br>`Traceback (most recent call last):`<br>`  File "<stdin>", line 1, in <module>`<br>`ValueError: unmatched '{' in format`<br>`>>>`<br>`Traceback (most recent call last):`<br>`  File "<stdin>", line 1, in <module>`<br>`ValueError: unmatched '{' in format` |
| E103 | Malformed variable missing left curly-brace<br>For example `foo}` with missing `{`.<br>Only checks pythonformat variables.<br>Example (Python):<br>`Error: malformed variables: foo}`<br>`msgid: "{foo} bar baz"`<br>`msgstr: "foo} bar baz"`<br>`>>> "foo}".format(foo="some thing")`<br>`Traceback (most recent call last):`<br>`  File "<stdin>", line 1, in <module>`<br>`ValueError: Single '}' encountered in format strin`<br>`>>>`<br>`Traceback (most recent call last):`<br>`  File "<stdin>", line 1, in <module>`<br>`ValueError: Single '}' encountered in format strin` |
| E201 | Invalid variables in translated string<br>There are formatting variable tokens in the *translated* string that aren't in the original string.<br>Example:<br>`Error: mismatched: invalid variables: {helpurl}`<br>`msgid: "You can find help at {url}"`<br>`msgstr: "You can find help at {helpurl}"`<br>In this example, "helpurl" won't be in the list of variables to interpolate and this will throw a KeyError.<br>That's equivalent to this:<br>`>>> "You can find help at {helpurl}".format(url="h`<br>`Traceback (most recent call last):` |

These are for `.pot` files:

| Code | Description |
|---|---|
| W500 | Hard to read variable name<br>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.<br>Example:<br>`Warning: hardtoread: hard to read variable name "l`<br>`msgid: "Title: {l}"d help at {url}"`<br>`msgstr: ""` |
| W501 | One character variable name<br>Using a one character variable name doesn't give enough context to the translator about what's being put in that variable.<br>Example:<br>`Warning: onechar: one character variable name: "{t`<br>`msgid: "{t} \| {c}"`<br>`msgstr: ""` |
| W502 | Multiple unnamed variables<br>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.<br>Example:<br>`Warning: multiple variables with no name`<br>`msgid: "%s replies to %s"`<br>`msgstr: ""` |

## 3.4 Statusing!

### 3.4.1 Help

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

### 3.4.2 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.

## 3.5 Using Dennis with Django

Dennis has some shims to make it easier to use with a Django project.

To use Dennis with Django add `dennis.django_dennis` to `INSTALLED_APPS`.

After you do that, then `lint` and `translate` subcommands are available in‘‘manage.py‘‘:

```
$ ./manage.py lint
$ ./manage.py translate
```

## 3.6 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.

## 3.7 Recipes



### 3.7.1 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.

### 3.7.2 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
```

### 3.7.3 translate .po file for l10n issue detection

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:

```
#!/bin/bash

mkdir -p locale/xx/LC_MESSAGES
cp locale/templates/LC_MESSAGES/messages.pot \
    locale/xx/LC_MESSAGES/messages.po

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

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

### 3.7.4 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:

```
#!/bin/bash

for pofile in `find locale/ -name "*.po"`
do
    dir=`dirname "$pofile"`
    stem=`basename "$pofile" .po`

    dennis-cmd lint --errorsonly "$pofile"
    if [ $? -ne 0 ]
    then
        echo "ERRORZ!!! Not compiling $pofile!"

    else
        msgfmt -o "${dir}/${stem}.mo" "$pofile"
    fi
done
```

### 3.7.5 commit-msg git hook

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

```
#!/bin/bash

# Pipe the contents of the commit message file through dennis to
# a temp file, then copy it back.
(cat < $1 | dennis-cmd translate - > $1.tmp) && mv $1.tmp $1

# We always exit 0 even if the dennis-cmd fails. If the dennis-cmd
# fails, you get your original commit message. No one likes it when
```

```
# shenanigans break your stuff for realz.
exit 0;
```

### 3.7.6 convert your web page into Pirate for April fools day

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

```
#!/bin/bash

(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"
```

CHAPTER 4

# Project guide

## 4.1 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.

### 4.1.1 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.

### 4.1.2 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!

### 4.1.3 Tests

Tests are in `dennis/tests/`. We use nose as the test runner. Further, we use some nose utilities functions.

To run the tests, do:

```
$ nosetests
```

Please write tests for changes you make.

### 4.1.4 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.

## 4.2 License

## 4.3 Contributors

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

CHAPTER 5

# Indices and tables

- *genindex*
- *modindex*
- *search*