undebt

Release

Oct 19, 2016

Contents

This is the documentation for Undebt. See below for a list of all of the articles hosted here.

CHAPTER 1

Undebt: Command Line Interface

1.1 Install it

```
$ git clone https://github.com/Yelp/undebt.git
$ cd undebt
$ pip install .
```

1.2 Read it

```
$ undebt --help
usage: undebt [-h] --pattern MODULE [--verbose] [--dry-run]
              [FILE [FILE...]]

positional arguments:
  FILE [FILE...]
                        paths to files to be modified (if not passed uses
                        stdin)

optional arguments:
  -h, --help                  show this help message and exit
  --pattern MODULE, -p MODULE
                              pattern definition modules
  --verbose, -v
  --dry-run, -d               only print to stdout; do not overwrite files
```

1.3 Try it out

```
$ undebt -p undebt.examples.method_to_function ./tests/inputs/method_to_function_input.txt
$ git diff
diff --git a/tests/inputs/method_to_function_input.txt b/tests/inputs/method_to_function_input.txt
index f268ab9..7681c63 100644
--- a/tests/inputs/method_to_function_input.txt
+++ b/tests/inputs/method_to_function_input.txt
@@ -1,13 +1,14 @@
+from function_lives_here import function
 something before code pattern
 @decorator([Class])
```

```
 def some_function(self, abc, xyz):
     """herp the derp while also derping and herping"""
     cde = fgh(self.l)
     ijk = cde.herp(
-        opq_foo=FOO(abc).method()
+        opq_foo=function(FOO(abc))
     )['str']
     lmn = cde.herp(
-        opq_foo=FOO(xyz).method()
+        opq_foo=function(FOO(xyz))
     )['str']
 bla bla bla
     for str_data in derp_data['data']['strs']:
@@ -16,8 +17,8 @@ bla bla bla
             rst.uvw(
                 CTA_BUSINESS_PLATFORM_DISABLED_LOG,
                 "derp {derp_foo} herp {herp_foo}".format(
-                    derp_foo=FOO(derp_foo).method(),
-                    herp_foo=FOO(herp_foo).method(),
+                    derp_foo=function(FOO(derp_foo)),
+                    herp_foo=function(FOO(herp_foo)),
                 ),
             )
 something after code pattern
```

1.4 Tips and Tricks

Most of these will make use of xargs

1.4.1 Using with `grep`/`git grep` to find files

```
grep -l <search-text> **/*.css | xargs undebt -p <pattern-module>

# Use git grep if you only want to search tracked files
git grep -l <search-text> | xargs undebt -p <pattern-module>
```

1.4.2 Using `find` to limit to a particular extension

```
find -name '*.js' | xargs grep -l <search-text> | xargs undebt -p <pattern-module>
```

1.4.3 Using `xargs` to work in parallel

`xargs` takes a `-P` flag, which specifies the maximum number of processes to use.

```
git grep -l <search-text> | xargs -P <numprocs> undebt -p <pattern-module>
```

CHAPTER 2

Undebt: Pattern Files

Undebt requires a pattern file that describes what to replace and how to replace it. There are two different ways to write pattern files: basic style, and advanced style. Unless you know you need multi-pass parsing, you should use basic style by default.

2.1 Basic Style

If you don't know what style you should be using, you should be using basic style. When writing a basic style pattern, you must define the following names in your pattern file:

- `grammar` defines what pattern you want to replace, and must be a pyparsing grammar object.
- `replace` is a function of one argument, the tokens produced by `grammar`, that returns the string they should be replaced with, or `None` to do nothing (this is the single-argument form—multi-argument is also allowed as documented *below*).
- *(optional)* `extra` can be set to a string that will be added to the beginning (after the standard Python header) of any file in which there's at least one match for `grammar` and in which `extra` does not already appear in the header (this feature is commonly used for adding in imports).

That sounds complicated, but it's actually very simple. To start learning more, it's recommended you check out Undebt's example patterns and pattern utilities.

2.2 Advanced Style

Unlike basic style, advanced style allows you to use custom multi-pass parsing—if that's not something you need, you should use basic style. When writing an advanced style pattern, you need only define one name:

- `patterns` is a list of `(grammar, replace)` tuples, where each tuple in the list is only run if the previous one succeeded

If `patterns` is defined, Undebt will ignore any definitions of `grammar`, `replace`, and `extra`. Instead, all of that information should go into the `patterns` list.

As an example, you can replicate the behavior of the basic style `extra` by doing the following:

```
from undebt.pattern.lang.python import HEADER

@tokens_as_list(assert_len=1)
def extra_replace(tokens):
    if extra not in tokens[0]:
```

```
        return tokens[0] + extra + "\n"
    else:
        return None

patterns.append((HEADER, extra_replace))
```

Or equivalently but more succinctly:

```
from undebt.pattern.interface import get_pattern_for_extra

patterns.append(
    get_pattern_for_extra(
        extra
    )
)
```

2.3 Multi-Argument Replace

In both styles, when writing a `replace` function, it is sometimes useful to have access to the parsing location in the file and/or the text of the original file. If your `replace` function takes two arguments, it will be passed `location, tokens`, and for three arguments, it will get `text, location, tokens`. This will work even if you are using one of the `tokens_as_list` or `tokens_as_dict` decorators.

CHAPTER 3

Undebt: Examples

The undebt.examples package contains various example pattern files. These example patterns can either simply be used as they are to make use of the transformation they describe, or used as templates to build your own pattern files.

3.1 undebt.examples.nl_at_eof

(Source)

A toy example to add a new line (`"\n"`) to the end of files that lack one.

Example of:

- use of the `tokens_as_list` decorator to define a `replace` function with assert checks
- negative lookahead using the ~ operator
- match any character with `ANY_CHAR`
- match the end of a file with `END_OF_FILE`

3.2 undebt.examples.dbl_quote_docstring

(Source)

Changes all `'''` strings that can be changed to `"""` strings.

Example of:

- return `None` from `replace` to do nothing
- match a `'''` string using `TRIPLE_SGL_QUOTE_STRING`

3.3 undebt.examples.class_inherit_object

(Source)

Changes classes that inherit from nothing to inherit from `object`, which makes sure they behave as Python 3 new-style classes instead of Python 2 old-style classes.

Example of:

- `Optional` to optionally match something

- `.suppress` method to prevent an object from appearing in the parsed tokens
- `Keyword` to match an individual word
- `INDENT` to match the beginning of a line and any leading whitespace
- `NAME` to match any variable name

3.4 undebt.examples.hex_to_bitshift

(Source)

Replaces hex flags with bitshift flags.

Example of:

- `Literal` to match a specific literal
- `Combine` to match a series of tokens without any whitespace in-between
- `Word` to match a word made up of a set of characters

3.5 undebt.examples.exec_function

(Source)

Changes instances of the Python 2 style `exec code in globals, locals` exec statement to the universal Python style `exec(code, globals, locals)` (which will work on Python 2.7 and Python 3).

Example of:

- using `tokens_as_list` to assert multiple possible token list lengths
- `ATOM` to match a Python atom

3.6 undebt.examples.attribute_to_function

(Source)

Transforms uses of `.attribute` into calls to `function`, and adds `from function_lives_here import function` whenever an instance of `function` is added.

Example of:

- use of `extra` to add an import statement
- multiple possible patterns using the `|` operator
- `ZeroOrMore` to match any number of a pattern
- `PARENS, BRACKETS` to match anything inside matching parentheses and brackets
- `ATOM_BASE` to match a trailerless Python atom

3.7 undebt.examples.method_to_function

(Source)

Slightly more complicated version of `attribute_to_function` that finds a method call instead of an attribute access, and makes sure that method call is not on `self`.

3.8 undebt.examples.sqla_count

(Source)

Transforms inefficient SQL alchemy `.count()` queries into more efficient `.scalar()` queries that don't create a sub query.

Example of:

- use of the `tokens_as_dict` decorator to define a `replace` function with assert checks
- grammar element function calling to label tokens in the resulting `tokens_as_dict` dictionary
- using `leading_whitespace` and `trailing_whitespace` to extract whitespace in a `replace` function

3.9 undebt.examples.remove_unused_import

(Source)

Removes `from function_lives_here import function` if `function` does not appear anywhere else in the file.

Example of:

- using a multi-argument `replace` function
- using `HEADER` to analyze the header of a Python file

3.10 undebt.examples.contextlib_nested

(Source)

Transforms uses of `contextlib.nested` into multiple clauses in a `with` statement. Respects usage with `as` and without `as`.

Example of:

- using `tokens_as_dict` to assert multiple possible dictionary keys
- `EXPR` to match a Python expression
- `COMMA_IND, LPAREN_IND, IND_RPAREN` to match optional indentation at particular points

3.11 undebt.examples.remove_needless_u_specifier

(Source)

In files where `from __future__ import unicode_literals` appears, removes unnecessary `u` before strings.

Example of:

- an advanced style pattern file making use of multi-pass parsing
- using `in_string` to determine if the match location is inside of a string
- `originalTextFor` to make grammar elements parse to the original text that matched them
- `STRING` to match any valid string

3.12 undebt.examples.swift

(Source)

Transforms uses of `if let where` from Swift 2.2 to the updated syntax in Swift 3.0.

Example of:

- using Undebt to transform a language that isn't Python

Note: It's possible that the 'EXPR' grammar element used won't match all Swift expressions; if you are concerned about this, you should define a custom 'EXPR' corresponding to the syntax of a Swift expression.

CHAPTER 4

Undebt: Pattern Utilities

Undebt's `undebt.pattern` package exposes various modules full of functions and grammar elements for use in writing pattern files, all documented here.

4.1 undebt.pattern.util

tokens_as_list(assert_len=None, assert_len_in=None)

Decorator used to wrap `replace` functions that converts the parsed tokens into a list. `assert_len` checks that the tokens have exactly the given length, while `assert_len_in` checks that the length of the tokens is in the provided list.

tokens_as_dict(assert_keys=None, assert_keys_in=None)

Decorator used to wrap `replace` functions that converts the parsed tokens into a dictionary, with keys assigned by calling grammar elements with the desired key as the argument. `assert_keys` checks that the keys in the token dictionary are a subset of the given keys, while `assert_keys_in` checks that the given keys are a subset of the keys in the token dictionary.

condense(item)

Modifies a grammar element to parse to a single token instead of many different tokens by concatenating the parsed tokens together.

addspace(item)

Equivalent to `condense` but also adds a space delimiter in-between the concatenated tokens.

quoted(string)

Returns a grammar element that matches a string containing `string`.

leading_whitespace(text)

Returns the whitespace at the beginning of `text`.

trailing_whitespace(text)

Returns the whitespace at the end of `text`.

in_string(location, code)

Determines if, at the given location in the code, there is an enclosing non-multiline string.

fixto(item, output)

Modifies a grammar element to always parse to the same fixed `output`.

debug(item)

Modifies a grammar element to print the tokens that it matches.

attach(item, action)

Modifies a grammar element to parse to the result of calling `action` on the tokens produced by that grammar element.

sequence(grammar, n)

Creates a grammar element that matches exactly `n` of the input grammar.

4.2 undebt.pattern.common

INDENT Matches any amount of indentation at the start of a line.

PARENS, BRACKETS, BRACES Grammar elements that match an open parenthesis / bracket / brace to the corresponding closing parenthesis / bracket / brace.

NAME Grammar element that matches a variable name.

DOTTED_NAME Grammar element to match either one or more `NAME` separated by `DOT`.

NUM Grammar element to match a number.

STRING Grammar element that matches a string.

TRIPLE_QUOTE_STRING, TRIPLE_DBL_QUOTE_STRING, TRIPLE_SGL_QUOTE_STRING Grammar elements that match different types of multi-line strings.

NL = `Literal("\n")`

DOT = `Literal(".")`

LPAREN = `Literal("(")`

RPAREN = `Literal(")")`

COMMA = `Literal(",")`

COLON = `Literal(":")`

COMMA_IND, LPAREN_IND, IND_RPAREN Same as `COMMA`, `LPAREN`, and `RPAREN`, but allow for an `INDENT` after (for `COMMA_IND` and `LPAREN_IND`) or before (for `IND_RPAREN`).

LINE_START Matches the start of a line, either after a new line, or at the start of the file.

NO_BS_NL Matches a new line not preceded by a backslash.

START_OF_FILE Grammar element that only matches at the very beginning of the file.

END_OF_FILE Grammar element that only matches at the very end of the file.

SKIP_TO_TEXT Skips parsing position to the next non-whitespace character. To see the skipped text in a token, use `originalTextFor(PREVIOUS_GRAMMAR_ELEMENT + SKIP_TO_TEXT)` where `PREVIOUS_GRAMMAR_ELEMENT` is just whatever comes before `SKIP_TO_TEXT` in your grammar.

SKIP_TO_TEXT_OR_NL Same as `SKIP_TO_TEXT`, but won't skip over new lines.

ANY_CHAR Grammar element that matches any one character, including new lines, but not non-newline whitespace. To exclude newlines, just do `~NL + ANY_CHAR`.

WHITE Normally, whitespace between grammar elements is ignored when they are added together. Put `WHITE` in-between to capture that whitespace as a token.

NL_WHITE Same as `WHITE` but also matches new lines.

4.3 undebt.pattern.lang

Contains common patterns for a variety of languages. For example, for patterns specific to the Python grammar, use `undebt.pattern.lang.python`.

4.3.1 undebt.pattern.lang.python

EXPR Matches any valid Python expression.

EXPR_LIST, EXPR_IND_LIST Matches one or more `EXPR` separated by `COMMA` for `EXPR_LIST` or `COMMA_IND` for `EXPR_IND_LIST`.

PARAM, PARAMS Matches one of (`PARAM`), or at least one of (`PARAMS`), the valid Python function parameters (`arg`, `kwarg=val`, `*args`, `**kwargs`).

ATOM Matches a single valid Python atom (that is, an expression without operators).

TRAILER, TRAILERS Matches one of (`TRAILER`), or any number of (`TRAILERS`), the valid Python trailers (attribute access, function call, indexing, etc.).

ATOM_BASE Matches an `ATOM` without any `TRAILERS` attached to it.

OP Matches any valid Python operator.

BINARY_OP Matches a valid Python binary operator.

ASSIGN_OP Matches a valid Python assignment operator.

UNARY_OP Matches a valid Python unary operator.

UNARY_OP_ATOM Matches an `ATOM` potentially preceded by unary operator(s).

HEADER Matches imports, comments, and strings at the start of a file. Used to determine where to insert the basic style `extra`.

4.4 undebt.pattern.interface

get_pattern_for_extra(extra)

Returns a `(grammar, replace)` tuple describing a pattern to insert `extra` after `undebt.pattern.python.HEADER`.

get_patterns(*pattern_modules)

Returns a list containing a advanced style `patterns` list for each pattern module in `pattern_modules`. The resulting list can be passed to `undebt.cmd.logic.process`.

4.5 undebt.cmd.logic

process(patterns, text)

Where `patterns` is a list of advanced style `patterns` lists, applies the specified patterns to the given text and returns the transformed version. Usually used in conjunction with `undebt.pattern.interface.get_patterns`.

CHAPTER 5

Undebt: Using *pyparsing*

While Undebt's parsing utilities are very helpful and provide much of the necessary functionality for writing a `grammar`, all of the objects are pyparsing objects, and thus it is often necessary and/or useful to use `pyparsing` utilities.

While the official pyparsing documentation is a great resource, most of the more advanced utilities there will usually not be necessary. This documentation is an overview of those that are most likely to be useful.

5.1 Operators

+ (And)

Adding two grammar elements produces a new grammar element that matches the first one, then the second one, with optional intervening whitespace.

| (Or)

Oring two grammar elements produces a new grammar element that attempts to match the first one, then if that fails, attempts to match the second one.

~ (Negative Lookahead)

Inverting a grammar element produces a new grammar element that produces no tokens and matches only if the inverted grammar doesn't match. Using a negative lookahead also doesn't advance the current parsing position.

^ (Match Longest)

Similar to `|`, but matches the longest of the grammar elements that match, instead of the first grammar element that matches.

5.2 Functions

Literal(str)

Creates a grammar element that matches `str` exactly.

Keyword(str)

Creates a grammar element that matches `str` only if it is surrounded by non-letters.

Optional(...)

Creates a grammar element that matches zero or one of the contained grammar element.

ZeroOrMore(...)

Creates a grammar element that matches zero or more of the contained grammar element.

OneOrMore(...)

Creates a grammar element that matches one or more of the contained grammar element.

originalTextFor(...)

Modifies a grammar element to produce only a single token that is the original text that was matched by that grammar element.

Word(charset)

Creates a grammar element that matches a word made of characters in `charset`.

SkipTo(...)

Skips parsing position to the next match for the contained objects.

Combine(...)

Forces any grammar elements added together inside of `Combine` to not match intervening whitespace and produce only a single token.

Regex(str)

Creates a grammar element that matches `str` as a regular expression.

CHAPTER 6

Undebt: Contributing

6.1 Getting Started

Undebt's development is taking place on Github, so please go ahead and fork the repository if you want to begin contributing.

You'll then want to get a local copy of the code base:

```
git clone git@github.com:<your-username>/undebt.git
```

6.2 Getting Setup

It is highly recommended that you create a virtual environment before installing the project dependencies.

You can achieve both (create a virtualenv and install dependencies) with:

```
make dev
```

6.3 Running the Tests

Undebt uses tox for testing.

You can run the entire test suite:

```
make test
```

Or, run an individual environment:

```
tox -e py35  # probably need to be virtualenv
```

Note. If you do not have the required dependencies for each Tox environment, you will receive an error. Avoid this by passing the `--skip-missing-interpreters` option.

6.4 Adding Documentation

Undebt's documentation is formatted using reStructuredText and hosted on RTD. Please try to follow the existing style and organization patterns.

You can test your contribution with:

```
make docs
```

Your new, local documentation will be available at `docs/build/html/`.