snippet-fmt

Release 0.1.5

Format and validate code snippets in reStructuredText files.

Dominic Davis-Foster

May 15, 2024

Contents

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install snippet-fmt --user
```

1.2 from GitHub

```
$ python3 -m pip install git+https://github.com/python-formate/snippet-fmt@master --user
```

Chapter

TWO

Documentation

2.1 Usage

2.1.1 Command Line

Reformat code snippets in the given reStructuredText files.

```
snippet-fmt [OPTIONS] [FILENAME]...
```

Options

-c, --config-file <config_file>
The path to the TOML configuration file to use.

Default pyproject.toml

-e, --exclude <PATTERN>
Patterns for files to exclude from formatting.

-v, --verbose
Show verbose output.

--colour, --no-colour
Whether to use coloured output.

-T, --traceback
Show the complete traceback on error.

--diff
Show a diff of changes made

Arguments

FILENAME
Optional argument(s). Default None

2.1.2 As a `pre-commit` hook

`snippet-fmt` can also be used as a pre-commit hook. To do so, add the following to your .pre-commit-config.yaml file:

```
- repo: https://github.com/python-formate/snippet-fmt
  rev: 0.1.5
  hooks:
  - id: snippet-fmt
    args:
    - --verbose
```

The `args` option can be used to provide the command line arguments shown above. By default `snippet-fmt` is run with `--verbose --diff`

2.2 Configuration

`snippet-fmt` is configured using the `pyproject.toml` file in the root of your project (alongside `setup.py`, `tox.ini` etc.). The file uses the TOML syntax, with the configuration in the `[tool.snippet-fmt]` table.

The table can contain two keys: *`languages`* and *`directives`*

Alternatively, the *`-c / --config-file`* option can be used to point to a different TOML file. The layout is the same except the table `[snippet-fmt]` rather than `[tool.snippet.fmt]`.

`languages`
: This is a table of tables giving languages to check and reformat.

  These correspond to the value after `.. code-block::`, preserving case.

  For example, the following codeblock has a value of `'TOML'`:

  ```
  .. code-block:: TOML

      key = "value"
  ```

  Each language has a corresponding check / reformat function, which is determined from the lowercased form of the language name. This allows certain code blocks in a language to be excluded from formatting by using a different case, such as using `TOML` for most code blocks and `toml` for ones which shouldn't be reformatted.

  The currently supported languages (matched case insensitively) are:

  - JSON
  - INI
  - TOML
  - Python / Python3

  Each sub table contains options specific to that language (and capitalisation). The exact options vary, but each has a `reformat` option which defaults to `False`. If set to `True` the code snippets in that language will be reformatted, otherwise they will only be syntax checked.

  By default all languages are enabled for checks only.

directives
: The directive types to reformat, such as `'code-block'` for `.. code-block::`.

  The values are case sensitive.

  Defaults to `['code', 'code-block', 'sourcecode']`.

2.2.3 Supported Languages

The following languages are supported by `snippet-fmt`:

Python / Python3

Reformatting Python files with formate.

Options

`python.`**`reformat`**
: **Type:** Boolean
  Default: False

  If set to `true` the code blocks matching this language and capitalisation will be reformatted, otherwise they will only be syntax checked.

`python.`**`config-file`**
: **Type:** String
  Default: formate.toml

  The TOML file containing the configuration for formate.

JSON

Syntax checking and reformatting of JSON files, using Python's `json` module.

Options

`json.`**`reformat`**
: **Type:** Boolean
  Default: False

  If set to `true` the code blocks matching this language and capitalisation will be reformatted, otherwise they will only be syntax checked.

`json.`**`ensure_ascii`**
: **Type:** Boolean
  Default: false

  If `true`, the output is guaranteed to have all incoming non-ASCII characters escaped. If `false` (the default), these characters will be output as-is.

`json.`**`allow_nan`**

Type: Boolean

Default: true

If `true` (the default), then `NaN`, `Infinity`, and `-Infinity` will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise an error will be raised when attepting to reformat files containing such floats.

Note: JSON snippets containing `NaN` etc. when this option is `false` and `reformat` is also `false` will pass, as this check only takes place durinh reformatting.

`json.`**`sort_keys`**

Type: Boolean

Default: false

If `true` then the keys will be sorted alphabetically.

`json.`**`indent`**

Type: Integer or string

If `indent` is a non-negative integer or string, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0, negative, or "" will only insert newlines. If not specified a compact representation will be used. Using a positive integer indent indents that many spaces per level. If indent is a string (such as "t"), that string is used to indent each level.

`json.`**`separators`**

Type: Array of string

A 2-element array of `[item_separator, key_separator]`. The default is `(', ', ': ')` if `indent` is unspecified and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

TOML

Syntax checking and reformatting of TOML files using the toml library.

Note: This only supports TOML version 0.5.0.

Options

`toml.`**`reformat`**

Type: Boolean

Default: False

If set to `true` the code blocks matching this language and capitalisation will be reformatted, otherwise they will only be syntax checked.

INI

Syntax checking and reformatting of INI files, using Python's `configparser` module.

Options

`ini.`**`reformat`**

Type: Boolean
Default: False

If set to `true` the code blocks matching this language and capitalisation will be reformatted, otherwise they will only be syntax checked.

2.2.4 Example

```
[tool.snippet-fmt]
directives = [ "code", "code-block", "sourcecode",]

[tool.snippet-fmt.languages.python]
reformat = true
config-file = "pyproject.toml"

[tool.snippet-fmt.languages.TOML]
reformat = true

[tool.snippet-fmt.languages.toml]

[tool.snippet-fmt.languages.ini]
```

This will:

- look at `.. code::`, `.. code-block::` and `.. sourcecode::` directives for `python`, `TOML`, `toml`, and `ini`.
- Code blocks marked `python` and `TOML` will be reformatted.
- Code blocks marked `toml` and `ini` will only be checked for valid syntax.
- formate is configured to take its configuration from `pyproject.toml`.

2.3 Changelog

2.3.1 v0.1.4

Fixed typo in the regular expression preventing single line code blocks from matching.

2.3.2 v0.1.3

Ensure indentation is preserved with nested directives.

2.3.3 v0.1.2

Correctly handle indentation containing mixed tabs and spaces.

2.3.4 v0.1.1

Corrected filetypes in `.pre-commit-hooks.yaml`.

2.3.5 v0.1.0

Initial Release

2.4 Downloading source code

The `snippet-fmt` source code is available on GitHub, and can be accessed from the following URL: `https://github.com/python-formate/snippet-fmt`

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/python-formate/snippet-fmt
```

```
Cloning into 'snippet-fmt'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

2.4.1 Building from source

The recommended way to build `snippet-fmt` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

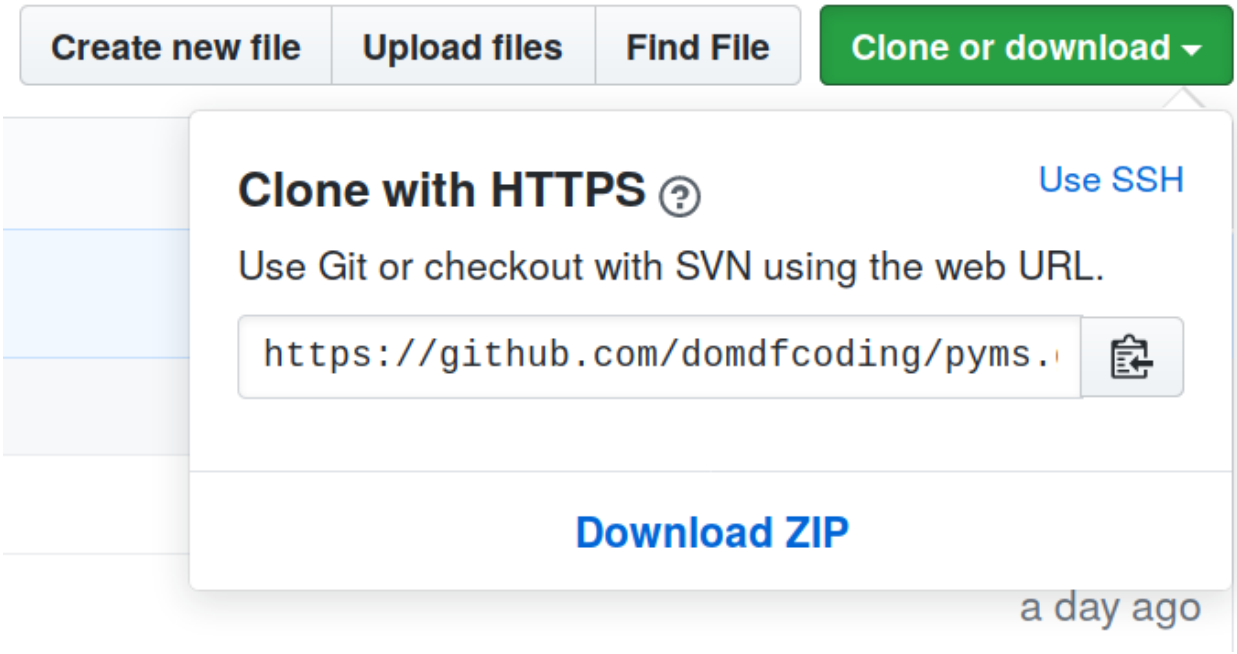

Fig. 1: Downloading a 'zip' file of the source code

2.5 License

`snippet-fmt` is licensed under the MIT License

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

See more information on choosealicense.com ⇒

Chapter

THREE

API Reference

3.1 snippet_fmt

Format and validate code snippets in reStructuredText files.

Classes:

CodeBlockError(offset, exc)	Represents an exception raised when parsing and reformatting a code block.
RSTReformatter(filename, config)	Reformat code snippets in a reStructuredText file.

Functions:

reformat_file(filename, config[, colour])	Reformat the given reStructuredText file, and show the diff if changes were made.

namedtuple CodeBlockError(*offset*, *exc*)

Bases: NamedTuple

Represents an exception raised when parsing and reformatting a code block.

Fields

0) **offset** (int) – The character offset where the exception was raised.
1) **exc** (Exception) – The exception itself.

__repr__()

Return a nicely formatted representation string

class RSTReformatter(*filename*, *config*)

Bases: object

Reformat code snippets in a reStructuredText file.

Parameters

- **filename** (Union[str, Path, PathLike]) – The filename to reformat.
- **config** (*SnippetFmtConfigDict*) – The snippet_fmt configuration, parsed from a TOML file (or similar).

Attributes:

config	The `formate` configuration, parsed from a TOML file (or similar).
file_to_format	The filename being reformatted, as a POSIX-style path.
filename	The filename being reformatted.

Methods:

get_diff()	Returns the diff between the original and reformatted file content.
load_extra_formatters()	Load custom formatters defined via entry points.
process_match(match)	Process a `re.Match` for a single code block.
run()	Run the reformatter.
to_file()	Write the reformatted source to the original file.
to_string()	Return the reformatted file as a string.

config
Type: *SnippetFmtConfigDict*

The `formate` configuration, parsed from a TOML file (or similar).

file_to_format
Type: `PathPlus`

The filename being reformatted, as a POSIX-style path.

filename
Type: `str`

The filename being reformatted.

get_diff()
Returns the diff between the original and reformatted file content.

Return type `str`

load_extra_formatters()
Load custom formatters defined via entry points.

process_match(*match*)
Process a `re.Match` for a single code block.

Parameters **match** (`Match[str]`)

Return type `str`

run()
Run the reformatter.

Return type `bool`

Returns Whether the file was changed.

to_file()
Write the reformatted source to the original file.

to_string()
Return the reformatted file as a string.

Return type str

reformat_file(*filename*, *config*, *colour=None*)
Reformat the given reStructuredText file, and show the diff if changes were made.

Parameters

- **filename** (Union[str, Path, PathLike]) – The filename to reformat.
- **config** (*SnippetFmtConfigDict*) – The snippet-fmt configuration, parsed from a TOML file (or similar).
- **colour** (Optional[bool]) – Whether to force coloured output on (True) or off (False). Default None.

Return type int

3.2 snippet_fmt.config

Read and parse snippet-fmt configuration.

Classes:

SnippetFmtConfigDict	typing.TypedDict representing the configuration mapping parsed from formate.toml or similar.

Functions:

load_toml(filename)	Load the snippet-fmt configuration mapping from the given TOML file.

typeddict SnippetFmtConfigDict
Bases: TypedDict

typing.TypedDict representing the configuration mapping parsed from formate.toml or similar.

Required Keys

- **languages** (Dict[str, Dict[str, Any]]) – The languages to reformat. The keys correspond to the value after .. code-block::, including matching case. The values are key: value mappings giving language-specific options. The exact options vary, but each has a reformat option which defaults to False. If set to True the code snippets in that language will be reformatted, otherwise they will only be syntax checked. For example, the following code block has a value of 'TOML': .. code-block:: rst .. code-block:: TOML key = "value" Different capitalisation (e.g. JSON vs json) can be used to apply different settings to different groups of code blocks. For example, JSON code blocks could have and indent of 2, but json blocks have no indentation.
- **directives** (List[str]) – The directive types to reformat, such as 'code-block' for .. code-block::. The values are case sensitive.

load_toml(*filename*)
Load the `snippet-fmt` configuration mapping from the given TOML file.

Parameters **filename** (Union[str, Path, PathLike])

Return type *SnippetFmtConfigDict*

3.3 snippet_fmt.formatters

Formatters and syntax checkers.

Classes:

Formatter	typing.Protocol for formatter functions.

Functions:

format_ini(code, **config)	Check the syntax of, and reformat, the given INI configuration.
format_json(code, **config)	Check the syntax of, and reformat, the given JSON source.
format_python(code, **config)	Check the syntax of, and reformat, the given Python code.
format_toml(code, **config)	Check the syntax of, and reformat, the given TOML configuration.
noformat(code, **config)	A no-op formatter.

protocol Formatter
Bases: Protocol

typing.Protocol for formatter functions.

Classes that implement this protocol must have the following methods / attributes:

__call__(*code*, ***config*)
Call self as a function.

Return type str

format_ini(*code*, ***config*)
Check the syntax of, and reformat, the given INI configuration.

Parameters

- **code** (str) – The code to check.
- ****config** – The language-specific configuration.

Return type str

Returns The original code unchanged.

format_json(*code*, ***config*)
Check the syntax of, and reformat, the given JSON source.

Parameters

- **code** (str) – The code to check.
- ****config** – The language-specific configuration.

Return type str

Returns The original code unchanged.

format_python(*code*, ***config*)
Check the syntax of, and reformat, the given Python code.

Parameters

- **code** (str) – The code to check and reformat.
- ****config** – The language-specific configuration.

Return type str

Returns The reformatted code.

format_toml(*code*, ***config*)
Check the syntax of, and reformat, the given TOML configuration.

Parameters

- **code** (str) – The code to check.
- ****config** – The language-specific configuration.

Return type str

Returns The original code unchanged.

noformat(*code*, ***config*)
A no-op formatter.

Parameters

- **code** (str) – The code to check and reformat.
- ****config** – The language-specific configuration.

Return type str

Returns The original code unchanged.

Python Module Index

S

Index

Symbols

C

D

E

F

G

I

J

L

M