drytoml

Release 0.2.8

Pablo Woolvett

Feb 15, 2021

CONTENTS

Keep your toml configuration centralized and personalizable.

Through `drytoml`, TOML files can have references to any content from another TOML file. References work with relative/absolute paths and urls, and can point to whole files, a specific table, or in general anything reachable by a sequence of `getitem` calls, like `["tool", "poetry", "source", 0, "url"]`. `drytoml` takes care of transcluding the content for you.

Inspired by flakehell and nitpick, the main motivation behind `drytoml` is to have several projects sharing a common, centralized configuration defining codestyles, but still allowing granular control when required.

This is a deliberate departure from nitpick, which works as a linter instead, ensuring your local files have the right content.

CHAPTER

ONE

USAGE

`drytoml` has two main usages:

1. Use a file as a reference and "build" the resulting one:

```toml
# contents of pyproject.dry.toml
...
[tool.black]
__extends = "../../common/black.toml"
target-version = ['py37']
include = '\.pyi?$'
...
```

```toml
# contents of ../../common/black.toml
[tool.black]
line-length = 100
```

```
$ dry export --file=pyproject.dry.toml > pyproject.toml
```

```toml
# contents of pyproject.toml
[tool.black]
line-length = 100
target-version = ['py37']
include = '\.pyi?$'
```

2. Use included wrappers, allowing you to use references in your current configuration without changing any file:

   Consider the original `pyproject.dry.toml` from the example above, an alternative usage for `drytoml` is shown next. Instead of this:

```
$ black .
All done!
14 files left unchanged.
```

   You would run this:

```
$ dry black
reformatted /path/to/cwd/file_with_potentially_long_lines.py
reformatted /path/to/cwd/another_file_with_potentially_long_lines.py
All done!
2 files reformatted, 12 files left unchanged.
```

   What just happened? `drytoml` comes with a set of wrappers which

   1. Create a transcluded temporary file, equivalent to the resulting `pyproject.toml` in the example above

2. Configure the wrapped tool (`black` in this case) to use the temporary file
3. Run `black`, and get rid of the file on exit.

For the moment, the following wrappers are available (more to come, contributions are welcome):

- [x] black
- [x] isort
- [x] flakehell, flake8helled *

In the works:

- [ ] docformatter
- [ ] pytest

1.1 Notes

Although the flakehell project was archived, we're keeping a fork alive from here, availabe as flakeheaven in pypi.

CHAPTER

TWO

SETUP

2.1 Prerequisites

- A compatible python >3.6.9
- [recommended] virtualenv
- A recent `pip`

2.2 Install

Install as usual, with `pip`, `poetry`, etc:

- `pip install drytoml`
- `poetry add drytoml` (probably you'll want `poetry add --dev drytoml` instead)

CHAPTER

THREE

USAGE

For any command , run `--help` to find out flags and usage. Some of the most common are listed below:

- Use any of the provided wrappers as a subcommand, eg `dry black` instead of `black`.
- Use `dry -q export` and redirect to a file, to generate a new file with transcluded contents
- Use `dry cache` to manage the cache for remote references.

CHAPTER

FOUR

FAQ

Q: I want to use a different key to trigger transclusions

A: In cli mode (using the `dry` command), you can pass the `--key` flagcli, to change it. In api mode (from python code), initialize `drytoml.parser.Parser` using a custom value for the `extend_key` kwarg.

Q: I changed a referenced toml upstream (eg in github) but still get the same result.

A: Run `dry cache clear --help` to see available options.

CHAPTER

FIVE

CONTRIBUTE

Start by creating an issue, forking the project and creating a Pull Request.

5.1 Setting up the development environment

If you have docker, the easiest way is to use the provided devcontainer inside vscode, which already contains everything pre-installed. You must open the cloned directory using the remote-containers extension. Just run `poetry shell` or prepend any command with `poetry run` to ensure commands are run inside the virtual environment.

Alternatively, you can use poetry: `poetry install -E dev`

The next steps assume you have already activated the venv.

5.2 Committing

- Commits in every branch must adhere to Conventional Commits. Releases are done automatically and require conventional commits messages compliance.
- To validate commits, you can install the pre-commit hook

```
pre-commit install --hook-type commit-msg
```

- With venv activated, you can commit using `cz commit` instead of `git commit` to ensure compliance.

5.3 Running checks

You can look at the different actions defined in `.github/workflows`. There are three ways to check your code:

- Remotely, by pushing to an open PR. You'll se the results in the PR page.
- Manually, executing the check from inside a venv

  For example, to generate the documentation, check `.github/workflows/docs`. To run the `Build with Sphinx` step:

```
sphinx-build docs/src docs/build
```

  Or to run pytest, from `.github/workflows/tests.yml`:

```
sphinx-build docs/src docs/build
```

... etc

- Locally, with act (Already installed in the devcontainer)

For example, to emulate a PR run for the tests workflow:

```
act -W .github/workflows/tests.yml pull_request
```

CHAPTER

SIX

TODO

Check out current development here

CHAPTER

SEVEN

TABLE OF CONTENTS

7.1 Why drytoml?

7.1.1 A bit of History

We wanted to have a single source of truth where code style and convetions live. Because of its nature, it should be an evolving thing, as we found more edge cases, or decided to use another framkework to solve a specific problem.

Using tools like cookiecutter and nitpick cover part of the solution, which is why we deceided to develop drytoml.

7.1.2 Driving principles

- Use `.toml` as configuration file, with `pyproject.toml` as default, unless specified.
- Allow inheritance (transclusion) of configurations both from path and from urls.
- Allow as much overriding / customization as possible.
- Enable update of references, but disable them by default.

7.2 API Documentation

Information

This documentation was autogenerated from sphinx-apidoc from the docstrings

7.2.1 drytoml package

DRY Toml - Inheritance and centralization with toml files.

Subpackages

drytoml.app package

Cli application for drytoml.

drytoml.app.**main**()
Execute the cli application.

Returns The result of the wrapped command

drytoml.app.**setup_log**(*argv: Optional[List[str]]*) → List[str]
Control verbosity via logging level using "-q/-v" as flags.

Parameters **argv** – If not set, use sys.argv. For each "-v" or "–verbose", increase the log level verbosity. If it contains a "-q", or a "–quiet", set lefel to *logging.CRITICAL*.

Returns Unparsed, remanining arguments.

Submodules

drytoml.app.cache module

Manage drytoml's internal cache.

This module contains the Cache class, which allows fire to execute any method (bound, static, or classmethod) as sub-command from the cli.

class drytoml.app.cache.**Cache**
Bases: object

Manage drytoml's internal cache.

classmethod clear(*force: bool = False*, *name: str = ''*) → Dict[Union[pathlib.Path, str], str]
Clear drytoml's cache.

Parameters

- **force** – Clear without asking.
- **name** – If set, only clear a specific cache element.

Returns Contents of the cache after clearing it.

static show() → Dict[Union[pathlib.Path, str], str]
Show drytoml's cache contents.

Returns Locations -> weight (in kb) mapping

drytoml.app.explain module

This module contains the *explain* command and its required utilities.

drytoml.app.explain.**explain**(*file='pyproject.toml'*, *key='__extends'*)
Show steps for toml transclusion.

Parameters

- **file** – TOML file to interpolate values.
- **key** – Name too look for inside the file to activate interpolation.

Example

```
>>> explain("isort.toml", "base")
```

drytoml.app.export module

This module contains the *export* command and its required utilities.

drytoml.app.export.**export** (*file='pyproject.toml'*, *key='__extends'*) → str
Generate resulting TOML after transclusion.

Parameters

- **file** – TOML file to transclude values.
- **key** – Name too look for inside the file to activate interpolation.

Returns The transcluded toml.

Example

```
>>> toml = export("isort.toml", "base")
```

drytoml.app.wrappers module

Third-party commands enabled through drytoml.

class drytoml.app.wrappers.**Cli** (*configs: List[str]*)
Bases: *drytoml.app.wrappers.Wrapper*

Call another script, configuring it with specific cli flag.

cfg: str

pre_call () → None
Prepare sys.argv to contain the configuration flag and file.

virtual: IO[str]

class drytoml.app.wrappers.**Env** (*env: Union[str, List[str]]*)
Bases: *drytoml.app.wrappers.Wrapper*

Call another script, configuring it with an environment variable.

cfg: str

pre_import ()
Configure env var before callback import.

virtual: IO[str]

class drytoml.app.wrappers.**Wrapper**
Bases: object

Common skeleton for third-party wrapper commands.

cfg: str

pre_call()
Execute custom processing done before callback execut.

pre_import()
Execute custom processing done before callback import.

tmp_dump()
Yield a temporary file with the configuration toml contents.

Yields Temporary file with the configuration toml contents

virtual: IO[str]

drytoml.app.wrappers.**black**()
Execute black, configured with custom setting cli flag.

drytoml.app.wrappers.**check**()
Execute all formatters and linters, sequentially.

drytoml.app.wrappers.**flake8helled**()
Execute flake8helled, configured with custom env var.

drytoml.app.wrappers.**flakehell**()
Execute flakehell, configured with custom env var.

drytoml.app.wrappers.**import_callable**(*string: str*) → Callable
Import a module from a string using colon syntax.

Parameters **string** – String of the form *package.module:object*

Returns The imported module

drytoml.app.wrappers.**isort**()
Execute isort, configured with custom setting cli flag.

drytoml.app.wrappers.**pylint**()
Execute pylint, configured with custom setting cli flag.

Submodules

drytoml.locate module

Utilities to simplify deep *getitem* calls.

drytoml.locate.**deep_del**(*document*, *final*, **breadcrumbs*)
Delete content located deep within a data structure.

Parameters

- **document** – Where to remove the content from.
- **final** – Last key required to locate the element to be deleted.
- **breadcrumbs** – The path to walk from the container root up to the parent ob the object to be deleted.

Examples

Examples should be written in doctest format, and should illustrate how to use the function.

```
>>> container = {
...     "foo": [
...         {},
...         {"bar": "delete_me"}
...     ]
... }
{'foo': [{}, {'bar': 'delete_me'}]}
>>> deep_del(container, "bar", ["foo", 1])
>>> container
{'foo': [{}, {}]}
```

drytoml.locate.**deep_find**(*container: Union[str, list, dict]*, *extend_key: str*, *breadcrumbs: Optional[List[str]] = None*) → Generator[List[tomlkit.items.Key], type, None]

Walk a data structure to yield all occurences of a key.

Parameters

- **container** – Where to look for the key.
- **extend_key** – The key to look for.
- **breadcrumbs** – The sequence of walked keys to the current position.

Returns The container type

Yields

Tuple containing (*breadcrumbs*, *content*) where *extend_key* was found.

drytoml.merge module

Utilities and logic for handling inter-toml merges.

class drytoml.merge.**TomlMerger**(*container: tomlkit.container.Container*, *parser*)

Bases: object

Encapsulate toml merging strategies and procedures.

build_subparser(*value: tomlkit.items.Item*)

Construct child parser from specific content.

Parameters **value** – The content of the TOML data to be parsed.

Returns The instantiated child parser.

merge_dict_like(*dct: Dict[tomlkit.items.Key, tomlkit.items.Item]*, *breadcrumbs: List[tomlkit.items.Key]*)

Merge a dict-like object into a specific container position.

Parameters

- **dct** – The incoming data.
- **breadcrumbs** – Location of the parent container for the incoming data merge.

merge_list_like(*values: List[tomlkit.items.Item]*, *breadcrumbs: List[tomlkit.items.Key]*)

Merge sequence of values in a specific position of the container.

Parameters

- **values** – Incoming data to be merged one by one, in reversed order.
- **breadcrumbs** – Location of the parent container for the incoming value merge.

merge_simple(*value: tomlkit.items.Item*, *breadcrumbs: List[tomlkit.items.Key]*)
Merge a value in a specific position of the container.

Parameters

- **value** – Incoming data to be merged.
- **breadcrumbs** – Location of the parent container for the incoming value merge.

drytoml.merge.**deep_extend**(*current: Union[tomlkit.items.Array, tomlkit.items.AoT]*, *incoming: Union[tomlkit.items.Array, tomlkit.items.AoT]*) → Union[tomlkit.items.Array, tomlkit.items.AoT]
Extend a container with another's contents.

Parameters

- **current** – Container to extend (in-place).
- **incoming** – Container to extend with.

Returns The recevied container, modified in-place.

drytoml.merge.**deep_merge**(*current: tomlkit.items.Item*, *incoming: tomlkit.items.Item*) → tomlkit.items.Item
Merge two items using a type-dependent strategy.

Parameters

- **current** – Item to merge into.
- **incoming** – Item to merge from.

Raises **NotImplementedError** – Unable to merge received current and incoming item given their types.

Returns The current Item, after merging in-place.

drytoml.merge.**merge_targeted**(*document: tomlkit.container.Container*, *incoming: tomlkit.container.Container*, *breadcrumbs: List[Union[str, int]]*) → tomlkit.toml_document.TOMLDocument
Merge specific path contents from an incoming contianer into another.

Parameters

- **document** – The container to store the merge result.
- **incoming** – The source of the incoming data.
- **breadcrumbs** – Location of the incoming contend.

Returns The *document*, after merging in-place.

drytoml.parser module

Additional Source to transclude tomlkit with URL and files.

class drytoml.parser.**Parser**(*string: str*, *extend_key='__extends'*, *reference: Optional[Union[str, pathlib.Path,* drytoml.types.Url*]] = None*, *level=0*)

Bases: tomlkit.parser.Parser

Extend tomlkit parser to allow transclusion.

classmethod factory(*reference: Union[str,* drytoml.types.Url*, pathlib.Path]*, *extend_key='__extends'*, *parent_reference: Optional[Union[str, pathlib.Path,* drytoml.types.Url*]] = None*, *level=0*)

Instantiate a parser from url, string, or path.

Parameters

- **reference** – Existing file/url/path with the toml contents.
- **extend_key** – kwarg to construct the parser.
- **parent_reference** – Used to parse relative paths.
- **level** – kwarg to construct the parser.

Returns Parser instantiated from received reference.

Raises **ValueError** – Attempted to intantiate a parser with a relative path as reference, without a parent reference.

classmethod from_file(*path*, *extend_key='__extends'*, *level=0*)

Instantiate a parser from file.

Parameters

- **path** – Path to an existing file with the toml contents.
- **extend_key** – kwarg to construct the parser.
- **level** – kwarg to construct the parser.

Returns Parser instantiated from received path.

classmethod from_url(*url*, *extend_key='__extends'*, *level=0*)

Instantiate a parser from url.

Parameters

- **url** – URL to an existing file with the toml contents.
- **extend_key** – kwarg to construct the parser.
- **level** – kwarg to construct the parser.

Returns Parser instantiated from received url.

parse() → tomlkit.toml_document.TOMLDocument

Parse recursively until no transclusions are required.

Returns The parsed, transcluded document.

drytoml.paths module

Common filesystem paths for drytoml.

drytoml.paths.**CACHE = PosixPath('/home/docs/.cache/drytoml')**
Location of drytoml's cache. It can be overriden by changing the XDG_CACHE_HOME env var.

drytoml.paths.**CONFIG = PosixPath('/home/docs/.config/drytoml')**
Location of drytoml's configuration files. It can be overriden by changing the XDG_CONFIG_HOME env var.

drytoml.paths.**env_or**(*xdg_env: str*, *home_subdir: Union[str, pathlib.Path]*) → pathlib.Path
Retreive path from xdg env, with home subdir as default.

Parameters

- **xdg_env** – Name of the environment variable.
- **home_subdir** – Sub-directory (inside $HOME) to use as default value if the env var is not found.

Returns Resulting path

Examples

For an exising env var: >>> env_or("XDG_CACHE_HOME", ".cache/custom") PosixPath('/home/vscode/.cache')

For an env var not present: >>> env_or("XDG_DATA_HOME", ".custom_subdir") PosixPath('/home/vscode/.custom_subdir')

drytoml.types module

Custom types and synonyms.

class drytoml.types.**Url**(*string*)
Bases: str

Avoid instantiation for non-compliant url strings.

URL_VALIDATOR = re.compile('^(?:http|ftp)s?://(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9]
Django validator.

classmethod validate(*maybe_url*) → bool
Validate url string using django regex.

Parameters **maybe_url** – Url to validate.

Returns *True* iff validation succeeds.

drytoml.utils module

Miscellaneous utilities used throughout the project.

drytoml.utils.**cached**(*func*)
Store output in drytoml's cache to use it on subsequent calls.

Parameters **func** – Function to decorate.

Returns Cached result with function result as fallback.

See also:

- *drytoml.paths.CACHE*
- *drytoml.app.cache*

drytoml.utils.**request**(*url: Union[str,* drytoml.types.Url*]*) → str
Request a *url* using a GET.

Parameters **url** – The URL to GET.

Returns Decoded content.

7.3 CHANGELOG

7.3.1 Fix

- **merge**: allow python native values to be merged as well (#32)

0.2.7 (2021-02-15)

7.3.2 Fix

- **app**: avoid forcing logger setup as it not available in pre python3.8 (#31)

0.2.6 (2021-02-15)

7.3.3 Fix

- **master-coverage**: also report coverage on releases (#30)

0.2.5 (2021-02-15)

7.3.4 Fix

- **ci**: avoid bump triggering workflow (#29)

0.2.4 (2021-02-15)

7.3.5 Fix

- **pypi**: poetry not found (#28)

0.2.3 (2021-02-15)

7.3.6 Fix

- **deploy**: replaced cz with commitizen (#27)
- **release**: bump poetry (#24)

7.3.7 fix

- **release**: use commitizen pkg as its not available for ci worker path (#26)

0.2.2 (2021-02-14)

7.3.8 Fix

- **release**: push tags from ci (#23)
- **release**: synchronize poetry and commitizen versions (#22)

0.2.1 (2021-02-14)

0.2.0 (2021-02-14)

7.3.9 Fix

- **log**: lazy logging properly implemented

0.1.2 (2021-02-13)

7.3.10 Fix

- **deps**: Replaced flakehell with flakeheaven form pypi instead of git to allow pypi publish (#15)

0.1.1 (2021-02-13)

7.3.11 Fix

- remove old issue template (#12)
- **bump**: disable simulator (#10)

0.1.0 (2021-02-13)

7.3.12 Fix

- autorelease
- **ci**: docs deployment to custom branch (#7)

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

P

R

S

T

U

V

W