
reqwire Documentation

Release 0.2.1

David Gidwani

Mar 20, 2021

Contents

1 Quickstart	3
2 Primer	5
3 User Guide	7
4 API Documentation	11
5 Release Notes	15
Python Module Index	19

reqwire wires up your Python dependencies with `pip-tools`.

Contents

- *Quickstart*
 - *Installation*
 - *Usage*

1.1 Installation

reqwire supports all versions of Python **above 2.7**. The recommended way to install **reqwire** is with `pip`:

```
$ pip install reqwire
```

Source code is available on [GitHub](#).

1.2 Usage

1. Run `reqwire init` in the working directory of your Python project. This will scaffold out a requirements directory:

```
requirements/  
- lck  
- src  
  - main.in  
  - qa.in  
  - test.in
```

2. To add requirements during development, use `reqwire add [-t <tag name>] <requirement>`.

For example, `reqwire add -t qa flake8` will:

- Resolve the latest version of flake8 (e.g. `flake8==3.2.1`).
- Add `flake8==3.2.1` to `requirements/src/qa.in`.

3. To compile tags, use `reqwire build -t <tag name>`.

To quickly compile *all* tags, use `reqwire build -a`.

Contents

- *Primer*
 - *Required Reading*
 - *State of Python Package Management*
 - *The Case for Tooling*

2.1 Required Reading

(Pun absolutely intended.)

- Unless you're already familiar with [pip-tools](#), be sure to read Vincent Driessen's article on [Better Package Management](#).
- After drinking the [pip-tools](#) kool-aid, read Kenneth Reitz's article on [A Better Pip Workflow™](#).
- If by now you're unconvinced that your project could benefit in terms of maintainability and build determinism by adding a separate `requirements.txt` for top-level dependencies, then what are you still doing here?

2.2 State of Python Package Management

Although new and [improved](#) ways of managing Python requirements are on the horizon, the current standard of including and maintaining at least one `requirements.txt` file probably isn't going anywhere.

Third-party tools like [pip-tools](#) aren't perfect solutions, but until an officially sanctioned alternative is implemented and released, Python package maintainers are going to have to decide the workflow that suits their needs.

2.3 The Case for Tooling

Some would argue that installing additional tools to manage package requirements adds unnecessary overhead to the development process. This argument certainly has merit, but it should be noted that `pip` itself was an external tool until it came installed with Python after version **2.7.9**. And while `Pipfile` looks incredibly promising, it might be safe to assume that it wouldn't be included in a Python distribution in the *immediate* future. Oh, and `virtualenv` remains an external tool, although it was included in the *standard library* through [PEP 405](#) (Python 3.3+).

Whether or not extra tooling for managing a project's requirements is necessary is up to the author(s), but historically, as well as in other spaces, managing software dependencies has frequently involved more than one tool.

Contents

- *User Guide*
 - *Directory Structure*
 - *Packaging and Version Control*
 - *Tag Organization*
 - *Command Reference*
 - * *reqwire add*
 - * *reqwire build*
 - * *reqwire init*
 - * *reqwire remove*

3.1 Directory Structure

Reqwire introduces a **requirements base directory** to the root of your project, and two subdirectories: a **source** directory and a **build** directory. By default, the source directory is named *src*, and the build directory is named *lck* (for lock, as built requirements.txt files are analogous to the lock files of many other package managers).

The names for all three of these directories are configurable, by passing the `-d|--directory`, `--source-directory`, and `--build-directory` options, respectively, to reqwire, **before** any commands. For example:

```
$ reqwire -d req --source-directory=_src --build-directory=_build build -a
# ...or...
$ export REQWIRE_DIR_BASE=req
```

```
$ export REQWIRE_DIR_SOURCE=_src
$ export REQWIRE_DIR_BUILD=_build
$ reqwire build -a
```

3.2 Packaging and Version Control

Ideally, the requirements directory should be located in a project's root directory, and both source and build directories added to version control. Depending on the project and target audience, it might make sense to copy or symlink the primary, built requirements tag (usually `main`) to a `requirements.txt` file in the project root.

If you're distributing a Python package, it might be useful to include the *build directory* in your `MANIFEST.in` file. This can be simply achieved with `graft`:

```
graft requirements/lck
```

Take care to ensure that the built requirements directory is not ignored by `.gitignore`, `.hgignore`, etc. This should not be a problem if using the default build directory name (`lck`).

3.3 Tag Organization

The purpose of **tags** in reqwire is to provide logical separation of package requirements based on the environment they target. For instance, `Sphinx` is likely only needed when building documentation, and not at runtime. `pytest` and `pytest` plugins are only required in a continuous integration (CI) environment, and so on.

Traditionally, you would use tools like `tox`, and end up maintaining requirements in more than one location, and likely not bother to pin versions or declare sub-dependencies. reqwire makes it convenient for package maintainers to quickly generate concrete, first-level requirements, which should hopefully encourage best practices across all environments.

3.4 Command Reference

3.4.1 reqwire add

- `reqwire add [specifier]...`

Installs packages to the local environment and updates one or more tagged requirement source files.

If no other parameters are given, this command will...

- Resolve the latest version of the provided package(s), unless a pinned version is provided.
- Install the package with **pip**.
- Add the requirement to the `main` tag.

- `reqwire add -b [specifier]...`

Calls `reqwire build` for each tag provided (or `main` if no tags were provided).

- `reqwire add -e [path/url]`

Adds an editable project.

- `reqwire add [-t <tag name>]... [specifier]...`

Saves packages to the specified requirement tag(s).

- `require add --no-install [specifier]...`

Skips package installation.

- `require add --no-resolve-canonical-names [specifier]...`

By default, `require` will search the Python package index for an exact match of package names, and use the canonical name (i.e. casing) for each specifier.

Passing this flag results in the user-provided package being saved to requirement source files.

- `require add --no-resolve-versions [specifier]...`

By default, `require` will resolve the latest version for each specifier provided.

Passing this flag allows for adding non-pinned packages to requirement source files. In most cases, this is not recommended even though the resulting requirement lock files will resolve to latest versions anyway.

- `require add --pre [specifier]...`

Includes prerelease versions when resolving versions.

3.4.2 `require build`

- `require build -a`

Builds all tags.

- `require build -t TAG`

Builds one or more tags.

- `require build -a -- [pip-compile options]...`

Passes all additional options and arguments to **pip-compile**.

For instance, to build requirements with hashes:

```
$ require build -a -- --generate-hashes
```

3.4.3 `require init`

- `require init`

Scaffolds a requirements directory in the current directory.

- `require init -f`

Scaffolds a requirements directory and overwrites any default tag names, and ignores pre-existing directories.

- `require init --index-url=INDEX_URL`

Changes the base URL written to requirement source files.

- `require init -t TAG`

Creates the given tag names as requirement source files.

If not provided, the tags `docs`, `main`, `qa`, and `test` will get created.

- `require init --extra-index-url INDEX_URL`

Adds `extra-index-url` options to requirement source files.

3.4.4 reqwire remove

- `reqwire remove [specifier]...`
Removes the provided package name(s) from the main requirement source file.
- `reqwire remove -t TAG [specifier]...`
Removes the provided package name(s) from one or more tagged requirement source files.

Contents

- *API Documentation*
 - *require.helpers*
 - * *require.helpers.cli*
 - * *require.helpers.requirements*
 - *require.config*
 - *require.errors*
 - *require.scaffold*

4.1 require.helpers

Various helper utilities.

4.1.1 require.helpers.cli

Helpers for command-line applications.

class `require.helpers.cli.ConsoleWriter` (*verbose=True*)
Facilitates writing formatted, informational messages to a TTY.

echo (*message, *args, **kwargs*)
Wraps `click.echo()`.

Parameters

- **message** – The message to write to stdout.

- ***args** – Used to format message.
- ****kwargs** – Used to format message.

error (*message*, *args, **kwargs)

Prints an error message.

Parameters

- **message** – The message to write to stdout.
- ***args** – Used to format message.
- ****kwargs** – Used to format message.

fatal (*message*, *args, **kwargs)

Prints a fatal message.

Parameters

- **message** – The message to write to stdout.
- ***args** – Used to format message.
- ****kwargs** – Used to format message.

info (*message*, *args, **kwargs)

Prints an informational message.

Parameters

- **message** – The message to write to stdout.
- ***args** – Used to format message.
- ****kwargs** – Used to format message.

warn (*message*, *args, **kwargs)

Prints a warning message.

Parameters

- **message** – The message to write to stdout.
- ***args** – Used to format message.
- ****kwargs** – Used to format message.

warning (*message*, *args, **kwargs)

Prints a warning message.

Parameters

- **message** – The message to write to stdout.
- ***args** – Used to format message.
- ****kwargs** – Used to format message.

`require.helpers.cli.emojize` (*message*, **kwargs)

Wrapper around `emoji.emojize()` for Windows compatibility.

Emoji are not well supported under Windows. This function not only checks `sys.platform`, but the file `/proc/version` as well to prevent “*emojification*” on the Windows Subsystem for Linux (WSL, otherwise known as Ubuntu on Windows).

Parameters

- **message** – The format string. See `emoji.emojize()` for more information. Any emoji placeholders will be removed if Windows or WSL are detected.
- ****kwargs** – Passed to `emoji.emojize()`.

4.1.2 reqwire.helpers.requirements

4.2 reqwire.config

Provides configuration and configuration defaults.

4.3 reqwire.errors

Provides custom exception classes.

exception `reqwire.errors.IndexUrlMismatchError`
Indicates a conflict between CLI and requirement source file.

exception `reqwire.errors.RequireError`
Base class for all exceptions thrown by reqwire.

4.4 reqwire.scaffold

Contents

- *Release Notes*
 - 0.2.1 (7/26/2017)
 - 0.2.0 (4/18/2017)
 - 0.1.8 (2/27/2017)
 - 0.1.7 (1/9/2017)
 - 0.1.6 (1/7/17)
 - 0.1.5 (1/7/17)
 - 0.1.2 - 0.1.4 (12/23/16)
 - 0.1.1 (12/22/16)
 - 0.1 (12/22/16)
 - 0.1a3 (12/11/16)
 - 0.1a2 (12/3/16)
 - 0.1a1 (12/1/16)

5.1 0.2.1 (7/26/2017)

- Bugfix release. Fixes #17.

5.2 0.2.0 (4/18/2017)

- Merged PR #14 by [ticosax](#), drops support for pip-tools <1.9.0.

5.3 0.1.8 (2/27/2017)

- Allow preservation of top-level dependencies (#12).

5.4 0.1.7 (1/9/2017)

Bugfix release, thanks to contributions from [ticosax](#).

- Merged PR #6 by [ticosax](#), fixes infinite recursion in `require.helpers.requirements.HashableInstallRequirement.from_line()`.
- Merged PR #8 by [ticosax](#). fixes `--no-resolve-versions`.
- Merged PR #9 by [ticosax](#), removes `--pin/--no-pin` flag in favor of `--no-resolve-versions`.

5.5 0.1.6 (1/7/17)

- Fixed support for installing/adding editable projects from VCS that use `setuptools_scm`.

5.6 0.1.5 (1/7/17)

- Initial support for adding editable requirements.
- Minor bugfixes.
- Added typing as a dependency for Python 2.7 to setup script.

5.7 0.1.2 - 0.1.4 (12/23/16)

- Various fixes for initialization and Python 2 compatibility.

5.8 0.1.1 (12/22/16)

- Fixed `require init`, uses user-defined source and build directory names.

5.9 0.1 (12/22/16)

- Corrected package setup to include `sh` as an installation dependency.
- Updated `MANIFEST.in` to include additional files in distribution.

- Made source and build directories configurable through command-line and environment variables.
- File headers now include modelines for Vim and Sublime Text (via `STEMacsModelines`).
- Added `require remove` command.

5.10 0.1a3 (12/11/16)

- Adding requirements no longer includes requirements from nested requirement files (and possibly constraint files).
- Added initial unit tests.
- Added `--pre` option to the **add** command, allowing prerelease versions of packages to be installed and added to requirements.
- Added `-b|--build` option to the **add** command, which invokes the **build** command upon successfully adding packages.
- Added initial documentation.

5.11 0.1a2 (12/3/16)

- Fixed support for Python 2.7.
- Nonexistent requirement directories are now handled gracefully.

5.12 0.1a1 (12/1/16)

- Initial alpha release, includes the **add**, **build**, and **init** commands.

r

`reqwire.config`, 13
`reqwire.errors`, 13
`reqwire.helpers`, 11
`reqwire.helpers.cli`, 11

C

ConsoleWriter (class in require.helpers.cli), 11

E

echo() (require.helpers.cli.ConsoleWriter method), 11

emojize() (in module require.helpers.cli), 12

error() (require.helpers.cli.ConsoleWriter method), 12

F

fatal() (require.helpers.cli.ConsoleWriter method), 12

I

IndexUrlMismatchError, 13

info() (require.helpers.cli.ConsoleWriter method), 12

R

require.config (module), 13

require.errors (module), 13

require.helpers (module), 11

require.helpers.cli (module), 11

RequireError, 13

W

warn() (require.helpers.cli.ConsoleWriter method), 12

warning() (require.helpers.cli.ConsoleWriter method),
12