Flit Documentation

Release 3.7.1

Thomas Kluyver

Aug 02, 2022

CONTENTS

Flit is a simple way to put Python packages and modules on PyPI. It tries to require less thought about packaging and help you avoid common mistakes. See Why use Flit? for more about how it compares to other Python packaging tools.

CHAPTER

ONE

INSTALL

```
$ python3 -m pip install flit
```

Flit requires Python 3 and therefore needs to be installed using the Python 3 version of pip.

Python 2 modules can be distributed using Flit, but need to be importable on Python 3 without errors.

CHAPTER

TWO

USAGE

Say you're writing a module `foobar` — either as a single file `foobar.py`, or as a directory — and you want to distribute it.

1. Make sure that foobar's docstring starts with a one-line summary of what the module is, and that it has a `__version__`:

   ```
   """An amazing sample package!"""

   __version__ = "0.1"
   ```

2. Install flit if you don't already have it:

   ```
   python3 -m pip install flit
   ```

3. Run `flit init` in the directory containing the module to create a `pyproject.toml` file. It will look something like this:

   ```
   [build-system]
   requires = ["flit_core >=3.2,<4"]
   build-backend = "flit_core.buildapi"

   [project]
   name = "foobar"
   authors = [{name = "Sir Robin", email = "robin@camelot.uk"}]
   dynamic = ["version", "description"]

   [project.urls]
   Home = "https://github.com/sirrobin/foobar"
   ```

   You can edit this file to add other metadata, for example to set up command line scripts. See the pyproject.toml page of the documentation.

   If you have already got a `flit.ini` file to use with older versions of Flit, convert it to `pyproject.toml` by running `python3 -m flit.tomlify`.

4. Run this command to upload your code to PyPI:

   ```
   flit publish
   ```

Once your package is published, people can install it using *pip* just like any other package. In most cases, pip will download a 'wheel' package, a standard format it knows how to install. If you specifically ask pip to install an 'sdist' package, it will install and use Flit in a temporary environment.

To install a package locally for development, run:

```
flit install [--symlink] [--python path/to/python]
```

Flit packages a single importable module or package at a time, using the import name as the name on PyPI. All sub-packages and data files within a package are included automatically.

CHAPTER

THREE

DOCUMENTATION CONTENTS

3.1 The pyproject.toml config file

This file lives next to the module or package.

Note: Older version of Flit (up to 0.11) used a flit.ini file for similar information. These files no longer work with Flit 3 and above.

Run `python3 -m flit.tomlify` to convert a `flit.ini` file to `pyproject.toml`.

3.1.1 Build system section

This tells tools like pip to build your project with flit. It's a standard defined by PEP 517. For any new project using Flit, it will look like this:

```
[build-system]
requires = ["flit_core >=3.2,<4"]
build-backend = "flit_core.buildapi"
```

Version constraints:

- For now, all packages should specify <4, so they won't be impacted by changes in the next major version.
- *New style metadata* requires `flit_core >=3.2`
- *Old style metadata* requires `flit_core >=2,<4`
- The older flit.ini file requires `flit_core <3`.
- TOML features new in version 1.0 require `flit_core >=3.4`.
- `flit_core` 3.3 is the last version supporting Python 3.4 & 3.5. Packages supporting these Python versions can only use TOML v0.5.
- Only `flit_core` 2.x can build packages on Python 2, so packages still supporting Python 2 cannot use new-style metadata (the `[project]` table).

3.1.2 New style metadata

New in version 3.2.

The new standard way to specify project metadata is in a `[project]` table, as defined by **PEP 621**. Flit works for now with either this or the older `[tool.flit.metadata]` table (*described below*), but it won't allow you to mix them.

A simple `[project]` table might look like this:

```
[project]
name = "astcheck"
authors = [
    {name = "Thomas Kluyver", email = "thomas@kluyver.me.uk"},
]
readme = "README.rst"
classifiers = [
    "License :: OSI Approved :: MIT License",
]
requires-python = ">=3.5"
dynamic = ["version", "description"]
```

The allowed fields are:

name The name your package will have on PyPI. This field is required. For Flit, this name, with any hyphens replaced by underscores, is also the default value of the import name (see *Module section* if that needs to be different).

version Version number as a string. If you want Flit to get this from a `__version__` attribute, leave it out of the TOML config and include "version" in the `dynamic` field.

description A one-line description of your project. If you want Flit to get this from the module docstring, leave it out of the TOML config and include "description" in the `dynamic` field.

readme A path (relative to the .toml file) to a file containing a longer description of your package to show on PyPI. This should be written in reStructuredText, Markdown or plain text, and the filename should have the appropriate extension (`.rst`, `.md` or `.txt`). Alternatively, `readme` can be a table with either a `file` key (a relative path) or a `text` key (literal text), and an optional `content-type` key (e.g. `text/x-rst`).

requires-python A version specifier for the versions of Python this requires, e.g. `~=3.3` or `>=3.3,<4`, which are equivalents.

license A table with either a `file` key (a relative path to a license file) or a `text` key (the license text).

authors A list of tables with `name` and `email` keys (both optional) describing the authors of the project.

maintainers Same format as authors.

keywords A list of words to help with searching for your package.

classifiers A list of Trove classifiers. Add `Private :: Do Not Upload` into the list to prevent a private package from being uploaded to PyPI by accident.

dependencies & optional-dependencies See *Dependencies*.

urls See *URLs table*.

scripts & gui-scripts See *Scripts section*.

entry-points See *Entry points sections*.

dynamic A list of field names which aren't specified here, for which Flit should find a value at build time. Only "version" and "description" are accepted.

Dependencies

The `dependencies` field is a list of other packages from PyPI that this package needs. Each package may be followed by a version specifier like `>=4.1`, and/or an environment marker after a semicolon. For example:

```
dependencies = [
    "requests >=2.6",
    "configparser; python_version == '2.7'",
]
```

The `[project.optional-dependencies]` table contains lists of packages needed for every optional feature. The requirements are specified in the same format as for `dependencies`. For example:

```
[project.optional-dependencies]
test = [
    "pytest >=2.7.3",
    "pytest-cov",
]
doc = ["sphinx"]
```

You can call these optional features anything you want, although `test` and `doc` are common ones. You specify them for installation in square brackets after the package name or directory, e.g. `pip install '.[test]'`.

URLs table

Your project's page on pypi.org can show a number of links. You can point people to documentation or a bug tracker, for example.

This section is called `[project.urls]` in the file. You can use any names inside it. Here it is for flit:

```
[project.urls]
Documentation = "https://flit.readthedocs.io/en/latest/"
Source = "https://github.com/pypa/flit"
```

Scripts section

This section is called `[project.scripts]` in the file. Each key and value describes a shell command to be installed along with your package. These work like setuptools 'entry points'. Here's the section for flit:

```
[project.scripts]
flit = "flit:main"
```

This will create a `flit` command, which will call the function `main()` imported from `flit`.

A similar table called `[project.gui-scripts]` defines commands which launch a GUI. This only makes a difference on Windows, where GUI scripts are run without a console.

Entry points sections

You can declare entry points using sections named `[project.entry-points.groupname]`. E.g. to provide a pygments lexer from your package:

```
[project.entry-points."pygments.lexers"]
dogelang = "dogelang.lexer:DogeLexer"
```

In each `package:name` value, the part before the colon should be an importable module name, and the latter part should be the name of an object accessible within that module. The details of what object to expose depend on the application you're extending.

If the group name contains a dot, it must be quoted (`"pygments.lexers"` above). Script entry points are defined in *scripts tables*, so you can't use the group names `console_scripts` or `gui_scripts` here.

Module section

If your package will have different names for installation and import, you should specify the install (PyPI) name in the `[project]` table (*see above*), and the import name in a `[tool.flit.module]` table:

```
[project]
name = "pynsist"
# ...

[tool.flit.module]
name = "nsist"
```

Flit looks for the source of the package by its import name. The source may be located either in the directory that holds the `pyproject.toml` file, or in a `src/` subdirectory.

3.1.3 Old style metadata

Flit's older way to specify metadata is in a `[tool.flit.metadata]` table, along with `[tool.flit.scripts]` and `[tool.flit.entrypoints]`, described below. This is still recognised for now, but you can't mix it with *New style metadata*.

There are three required fields:

module The name of the module/package, as you'd use in an import statement.

author Your name

author-email Your email address

e.g. for flit itself

```
[tool.flit.metadata]
module = "flit"
author = "Thomas Kluyver"
author-email = "thomas@kluyver.me.uk"
```

Changed in version 1.1: `home-page` was previously required.

The remaining fields are optional:

home-page A URL for the project, such as its Github repository.

requires A list of other packages from PyPI that this package needs. Each package may be followed by a version specifier like `(>=4.1)` or `>=4.1`, and/or an environment marker after a semicolon. For example:

```
requires = [
    "requests >=2.6",
    "configparser; python_version == '2.7'",
]
```

requires-extra Lists of packages needed for every optional feature. The requirements are specified in the same format as for `requires`. The requirements of the two reserved extras `test` and `doc` as well as the extra `dev` are installed by `flit install`. For example:

```
[tool.flit.metadata.requires-extra]
test = [
    "pytest >=2.7.3",
    "pytest-cov",
]
doc = ["sphinx"]
```

New in version 1.1.

description-file A path (relative to the .toml file) to a file containing a longer description of your package to show on PyPI. This should be written in reStructuredText, Markdown or plain text, and the filename should have the appropriate extension (`.rst`, `.md` or `.txt`).

classifiers A list of Trove classifiers. Add `Private :: Do Not Upload` into the list to prevent a private package from uploading on PyPI by accident.

requires-python A version specifier for the versions of Python this requires, e.g. `~=3.3` or `>=3.3,<4` which are equivalents.

dist-name If you want your package's name on PyPI to be different from the importable module name, set this to the PyPI name.

keywords Comma separated list of words to help with searching for your package.

license The name of a license, if you're using one for which there isn't a Trove classifier. It's recommended to use Trove classifiers instead of this in most cases.

maintainer, maintainer-email Like author, for if you've taken over a project from someone else.

Here was the metadata section from flit using the older style:

```
[tool.flit.metadata]
module="flit"
author="Thomas Kluyver"
author-email="thomas@kluyver.me.uk"
home-page="https://github.com/pypa/flit"
requires=[
    "flit_core >=2.2.0",
    "requests",
    "docutils",
    "tomli",
    "tomli-w",
]
requires-python=">=3.6"
description-file="README.rst"
classifiers=[
```

(continues on next page)

(continued from previous page)

```
    "Intended Audience :: Developers",
    "License :: OSI Approved :: BSD License",
    "Programming Language :: Python :: 3",
    "Topic :: Software Development :: Libraries :: Python Modules",
]
```

URLs subsection

Your project's page on pypi.org can show a number of links, in addition to the `home-page` URL described above. You can point people to documentation or a bug tracker, for example.

This section is called `[tool.flit.metadata.urls]` in the file. You can use any names inside it. Here it is for flit:

```
[tool.flit.metadata.urls]
Documentation = "https://flit.readthedocs.io/en/latest/"
```

New in version 1.0.

Scripts section

A `[tool.flit.scripts]` table can be used along with `[tool.flit.metadata]`. It is in the same format as the newer `[project.scripts]` table *described above*.

Entry points sections

`[tool.flit.entrypoints]` tables can be used along with `[tool.flit.metadata]`. They are in the same format as the newer `[project.entry-points]` tables *described above*.

3.1.4 Sdist section

New in version 2.0.

When you use *flit build* or *flit publish*, Flit builds an sdist (source distribution) tarball containing the files that are checked into version control (git or mercurial). If you want more control, or it doesn't recognise your version control system, you can give lists of paths or glob patterns as `include` and `exclude` in this section. For example:

```
[tool.flit.sdist]
include = ["doc/"]
exclude = ["doc/*.html"]
```

These paths:

- Always use `/` as a separator (POSIX style)
- Must be relative paths from the directory containing `pyproject.toml`
- Cannot go outside that directory (no `../` paths)
- Cannot contain control characters or `<>:"\\`
- Cannot use recursive glob patterns (`**/`)
- Can refer to directories, in which case they include everything under the directory, including subdirectories

- Should match the case of the files they refer to, as case-insensitive matching is platform dependent

Exclusions have priority over inclusions.

Note: If you are not using *flit build* but `flit_core` via another build frontend, Flit doesn't doesn't check the VCS for files to include but instead builds a 'minimal' sdist (which includes the files necessary to build a wheel). You'll have to adapt your inclusion/exclusion rules to achieve the same result as you'd get with *flit build*.

3.1.5 External data section

New in version 3.7.

Data files which your code will use should go inside the Python package folder. Flit will package these with no special configuration.

However, sometimes it's useful to package external files for system integration, such as man pages or files defining a Jupyter extension. To do this, arrange the files within a directory such as `data`, next to your `pyproject.toml` file, and add a section like this:

```
[tool.flit.external-data]
directory = "data"
```

Paths within this directory are typically installed to corresponding paths under a prefix (such as a virtualenv directory). E.g. you might save a man page for a script as `(data)/share/man/man1/foo.1`.

Whether these files are detected by the systems they're meant to integrate with depends on how your package is installed and how those systems are configured. For instance, installing in a virtualenv usually doesn't affect anything outside that environment. Don't rely on these files being picked up unless you have close control of how the package will be installed.

If you install a package with `flit install --symlink`, a symlink is made for each file in the external data directory. Otherwise (including development installs with `pip install -e`), these files are copied to their destination, so changes here won't take effect until you reinstall the package.

Note: For users coming from setuptools: external data corresponds to setuptools' `data_files` parameter, although setuptools offers more flexibility.

3.2 Flit command line interface

All operations use the `flit` command, followed by one of a number of subcommands.

3.2.1 Common options

-f <path>, **--ini-file** <path>

Path to a config file specifying the module to build. The default is `pyproject.toml`.

--version

Show the version of Flit in use.

--help

Show help on the command-line interface.

--debug

Show more detailed logs about what flit is doing.

3.2.2 `flit build`

Build a wheel and an sdist (tarball) from the package.

--format <format>

Limit to building either `wheel` or `sdist`.

--setup-py

Generate a `setup.py` file in the sdist, so it can be installed by older versions of pip.

--no-setup-py

Don't generate a setup.py file in the sdist. This is the default. An sdist built without this will only work with tools that support PEP 517, but the wheel will still be usable by any compatible tool.

Changed in version 3.5: Generating `setup.py` disabled by default.

3.2.3 `flit publish`

Build a wheel and an sdist (tarball) from the package, and upload them to PyPI or another repository.

--format <format>

Limit to publishing either `wheel` or `sdist`. You should normally publish the two formats together.

--setup-py

Generate a `setup.py` file in the sdist, so it can be installed by older versions of pip.

--no-setup-py

Don't generate a setup.py file in the sdist. This is the default. An sdist built without this will only work with tools that support PEP 517, but the wheel will still be usable by any compatible tool.

Changed in version 3.5: Generating `setup.py` disabled by default.

--repository <repository>

Name of a repository to upload packages to. Should match a section in `~/.pypirc`. The default is `pypi`.

--pypirc <pypirc>

The .pypirc config file to be used. The default is `~/.pypirc`.

See also:

Controlling package uploads

3.2.4 flit install

Install the package on your system.

By default, the package is installed to the same Python environment that Flit itself is installed in; use *--python* or *FLIT_INSTALL_PYTHON* to override this.

If you don't have permission to modify the environment (e.g. the system Python on Linux), Flit may do a user install instead. Use the *--user* or *--env* flags to force this one way or the other, rather than letting Flit guess.

-s, --symlink

Symlink the module into site-packages rather than copying it, so that you can test changes without reinstalling the module.

--pth-file

Create a `.pth` file in site-packages rather than copying the module, so you can test changes without reinstalling. This is a less elegant alternative to `--symlink`, but it works on Windows, which typically doesn't allow symlinks.

--deps <dependency option>

Which dependencies to install. One of `all`, `production`, `develop`, or `none`. `all` and `develop` install the extras `test`, `doc`, and `dev`. Default `all`.

--extras <extra[,extra,...]>

Which named extra features to install dependencies for. Specify `all` to install all optional dependencies, or a comma-separated list of extras. Default depends on `--deps`.

--user

Do a user-local installation. This is the default if flit is not in a virtualenv or conda env (if the environment's library directory is read-only and `site.ENABLE_USER_SITE` is true).

--env

Install into the environment - the opposite of *--user*. This is the default in a virtualenv or conda env (if the environment's library directory is writable or `site.ENABLE_USER_SITE` is false).

--python <path to python>

Install for another Python, identified by the path of the python executable. Using this option, you can install a module for Python 2, for instance. See *FLIT_INSTALL_PYTHON* if this option is not given.

Changed in version 2.1: Added *FLIT_INSTALL_PYTHON* and use its value over the Python running Flit when an explicit *--python* option is not given.

Note: Flit calls pip to do the installation. You can set any of pip's options using its environment variables.

When you use the *--symlink* or *--pth-file* options, pip is used to install dependencies. Otherwise, Flit builds a wheel and then calls pip to install that.

3.2.5 flit init

Create a new `pyproject.toml` config file by prompting for information about the module in the current directory.

3.2.6 Environment variables

FLIT_NO_NETWORK

New in version 0.10.

Setting this to any non-empty value will stop flit from making network connections (unless you explicitly ask to upload a package). This is intended for downstream packagers, so if you use this, it's up to you to ensure any necessary dependencies are installed.

FLIT_ROOT_INSTALL

By default, `flit install` will fail when run as root on POSIX systems, because installing Python modules systemwide is not recommended. Setting this to any non-empty value allows installation as root. It has no effect on Windows.

FLIT_USERNAME

FLIT_PASSWORD

FLIT_INDEX_URL

New in version 0.11.

Set a username, password, and index URL for uploading packages. See *uploading packages with environment variables* for more information.

FLIT_ALLOW_INVALID

New in version 0.13.

Setting this to any non-empty value tells Flit to continue if it detects invalid metadata, instead of failing with an error. Problems will still be reported in the logs, but won't cause Flit to stop.

If the metadata is invalid, uploading the package to PyPI may fail. This environment variable provides an escape hatch in case Flit incorrectly rejects your valid metadata. If you need to use it and you believe your metadata is valid, please open an issue.

FLIT_INSTALL_PYTHON

New in version 2.1.

Set a default Python interpreter for *flit install* to use when `--python` is not specified. The value can be either an absolute path, or a command name (which will be found in `PATH`). If this is unset or empty, the module is installed for the copy of Python that is running Flit.

SOURCE_DATE_EPOCH

To make reproducible builds, set this to a timestamp as a number of seconds since the start of the year 1970 in UTC, and document the value you used. On Unix systems, you can get a value for the current time by running:

```
date +%s
```

See also:

The SOURCE_DATE_EPOCH specification

3.3 Controlling package uploads

The command `flit publish` will upload your package to a package index server. The default settings let you upload to PyPI, the default Python Package Index, with a single user account.

If you want to upload to other servers, or with more than one user account, or upload packages from a continuous integration job, you can configure Flit in two main ways:

3.3.1 Using .pypirc

You can create or edit a config file in your home directory, `~/.pypirc` that will be used by default or you can specify a custom location. This is also used by other Python tools such as twine.

For instance, to upload a package to the Test PyPI server instead of the normal PyPI, use a config file looking like this:

```
[distutils]
index-servers =
   pypi
   testpypi

[pypi]
repository = https://upload.pypi.org/legacy/
username = sirrobin  # Replace with your PyPI username

[testpypi]
repository = https://test.pypi.org/legacy/
username = sirrobin  # Replace with your TestPyPI username
```

You can select an index server from this config file with the *--repository* option:

```
flit publish --repository testpypi
```

If you don't use this option, Flit will use the server called `pypi` in the config file. If that doesn't exist, it uploads to PyPI at `https://upload.pypi.org/legacy/` by default.

If you publish a package and you don't have a `.pypirc` file, Flit will create it to store your username.

Flit tries to store your password securely using the keyring library. If keyring is not installed, Flit will ask for your password for each upload. Alternatively, you can also manually add your password to the `.pypirc` file (`password = ...`)

3.3.2 Using environment variables

You can specify a server to upload to with *FLIT_INDEX_URL*, and pass credentials with *FLIT_USERNAME* and *FLIT_PASSWORD*. Environment variables take precedence over the config file, except if you use the *--repository* option to explicitly pick a server from the config file.

This can make it easier to automate uploads, for example to release packages from a continuous integration job.

Warning: Storing a password in an environment variable is convenient, but it's easy to accidentally leak it. Look out for scripts that helpfully print all environment variables for debugging, and remember that other scripts and libraries you run in that environment have access to your password.

3.4 Reproducible builds

New in version 0.8.

Wheels built by flit are reproducible: if you build from the same source code, you should be able to make wheels that are exactly identical, byte for byte. This is useful for verifying software. For more details, see reproducible-builds.org.

There is a caveat, however: wheels (which are zip files) include the modification timestamp from each file. This will probably be different on each computer, because it indicates when your local copy of the file was written, not when it was changed in version control. These timestamps can be overridden by the environment variable *SOURCE_DATE_EPOCH*.

```
SOURCE_DATE_EPOCH=$(date +%s)
flit publish
# Record the value of SOURCE_DATE_EPOCH in release notes for reproduction
```

Changed in version 0.12: Normalising permission bits

Flit normalises the permission bits of files copied into a wheel to either 755 (executable) or 644. This means that a file is readable by all users and writable only by the user who owns it.

The most popular version control systems only track the executable bit, so checking out the same repository on systems with different umasks (e.g. Debian and Fedora) produces files with different permissions. With Flit 0.11 and earlier, this difference would produce non-identical wheels.

3.5 Why use Flit?

Make the easy things easy and the hard things possible is an old motto from the Perl community. Flit is entirely focused on the *easy things* part of that, and leaves the hard things up to other tools.

Specifically, the easy things are pure Python packages with no build steps (neither compiling C code, nor bundling Javascript, etc.). The vast majority of packages on PyPI are like this: plain Python code, with maybe some static data files like icons included.

It's easy to underestimate the challenges involved in distributing and installing code, because it seems like you just need to copy some files into the right place. There's a whole lot of metadata and tooling that has to work together around that fundamental step. But with the right tooling, a developer who wants to release their code doesn't need to know about most of that.

What, specifically, does Flit make easy?

- `flit init` helps you set up the information Flit needs about your package.
- Subpackages are automatically included: you only need to specify the top-level package.
- Data files within a package directory are automatically included. Missing data files has been a common packaging mistake with other tools.
- The version number is taken from your package's `__version__` attribute, so that always matches the version that tools like pip see.
- `flit publish` uploads a package to PyPI, so you don't need a separate tool to do this.

Setuptools, the most common tool for Python packaging, now has shortcuts for many of the same things. But it has to stay compatible with projects published many years ago, which limits what it can do by default.

Flit also has some support for *reproducible builds*, a feature which some people care about.

There have been many other efforts to improve the user experience of Python packaging, such as pbr, but before Flit, these tended to build on setuptools and distutils. That was a pragmatic decision, but it's hard to build something

radically different on top of those libraries. The existence of Flit spurred the development of new standards, like **PEP 518** and **PEP 517**, which are now used by other packaging tools such as Poetry and Enscons.

3.5.1 Other options

If your package needs a build step, you won't be able to use Flit. Setuptools is the de-facto standard, but newer tools such as Enscons also cover this case.

Flit also doesn't help you manage dependencies: you have to add them to `pyproject.toml` by hand. Tools like Poetry and Pipenv have features which help add and update dependencies on other packages.

3.6 Bootstrapping

Flit is itself packaged using Flit, as are some foundational packaging tools such as `pep517`. So where can you start if you need to install everything from source?

Note: For most users, `pip` handles all this automatically. You should only need to deal with this if you're building things entirely from scratch, such as putting Python packages into another package format.

The key piece is `flit_core`. This is a package which can build itself using nothing except Python and the standard library. From an unpacked source archive, you can make a wheel by running:

```
python -m flit_core.wheel
```

And then you can install this wheel with the `bootstrap_install.py` script included in the sdist (or by unzipping it to the correct directory):

```
# Install to site-packages for this Python:
python bootstrap_install.py dist/flit_core-*.whl

# Install somewhere else:
python bootstrap_install.py --installdir /path/to/site-packages dist/flit_core-*.whl
```

As of version 3.6, flit_core bundles the `tomli` TOML parser, to avoid a dependency cycle. If you need to unbundle it, you will need to special-case installing flit_core and/or tomli to get around that cycle.

After `flit_core`, I recommend that you get installer set up. You can use `python -m flit_core.wheel` again to make a wheel, and then use installer itself (from the source directory) to install it.

After that, you probably want to get build and its dependencies installed as the goal of the bootstrapping phase. You can then use `build` to create wheels of any other Python packages, and `installer` to install them.

3.7 Developing Flit

To get a development installation of Flit itself:

```
git clone https://github.com/pypa/flit.git
cd flit
python3 -m pip install docutils requests toml
python3 bootstrap_dev.py
```

This links Flit into the current Python environment, so you can make changes and try them without having to reinstall each time.

3.7.1 Testing

To run the tests in separate environments for each available Python version:

```
tox
```

tox has many options.

To run the tests in your current environment, run:

```
pytest
```

3.8 Release history

3.8.1 Unreleased

- Add support for recursive globbing (**) in sdist includes and excludes (PR #550).

3.8.2 Version 3.7.1

- Fix building packages which need execution to get the version number, and have a relative import in `__init__.py` (PR #531).

3.8.3 Version 3.7

- Support for *external data files* such as man pages or Jupyter extension support files (PR #510).
- Project names are now lowercase in wheel filenames and `.dist-info` folder names, in line with the specifications (PR #498).
- Improved support for *bootstrapping* a Python environment, e.g. for downstream packagers (PR #511). `flit_core.wheel` is usable with `python -m` to create wheels before the build tool is available, and `flit_core` sdists also include a script to install itself from a wheel before installer is available.
- Use newer importlib APIs, fixing some deprecation warnings (PR #499).

3.8.4 Version 3.6

- `flit_core` now bundles the tomli TOML parser library (version 1.2.3) to avoid a circular dependency between `flit_core` and `tomli` (PR #492). This means `flit_core` now has no dependencies except Python itself, both at build time and at runtime, simplifying *bootstrapping*.

3.8.5 Version 3.5.1

- Fix development installs with `flit install --symlink` and `--pth-file`, which were broken in 3.5.0, especially for packages using a `src` folder (PR #472).

3.8.6 Version 3.5

- You can now use Flit to distribute a module or package inside a namespace package (as defined by **PEP 420**). To do this, specify the import name of the concrete, inner module you are packaging - e.g. `name = "sphinxcontrib.foo"` - either in the `[project]` table, or under `[tool.flit.module]` if you want to use a different name on PyPI (PR #468).
- Flit no longer generates a `setup.py` file in sdists (`.tar.gz` packages) by default (PR #462). Modern packaging tools don't need this. You can use the `--setup-py` flag to keep adding it for now, but this will probably be removed at some point in the future.
- Fixed how `flit init` handles authors' names with non-ASCII characters (PR #460).
- When `flit init` generates a LICENSE file, the new `pyproject.toml` now references it (PR #467).

3.8.7 Version 3.4

- Python 3.6 or above is now required, both for `flit` and `flit_core`.
- Add a `--setup-py` option to `flit build` and `flit publish`, and a warning when neither this nor `--no-setup-py` are specified (PR #431). A future version will stop generating `setup.py` files in sdists by default.
- Add support for standardised editable installs - `pip install -e` - according to **PEP 660** (PR #400).
- Add a `--pypirc` option for `flit publish` to specify an alternative path to a `.pypirc` config file describing package indexes (PR #434).
- Fix installing dependencies specified in a `[project]` table (PR #433).
- Fix building wheels when `SOURCE_DATE_EPOCH` (see *Reproducible builds*) is set to a date before 1980 (PR #448).
- Switch to using the tomli TOML parser, in common with other packaging projects (PR #438). This supports TOML version 1.0.
- Add a document on *Bootstrapping* (PR #441).

3.8.8 Version 3.3

- `PKG-INFO` files in sdists are now generated the same way as `METADATA` in wheels, fixing some issues with sdists (PR #410).
- `flit publish` now sends SHA-256 hashes, fixing uploads to GitLab package repositories (PR #416).
- The `[project]` metadata table from **PEP 621** is now fully supported and *documented*. Projects using this can now specify `requires = ["flit_core >=3.2,<4"]` in the `[build-system]` table.

3.8.9 Version 3.2

- Experimental support for specifying metadata in a `[project]` table in `pyproject.toml` as specified by **PEP 621** (PR #393). If you try using this, please specify `requires = ["flit_core >=3.2.0,<3.3"]` in the `[build-system]` table for now, in case it needs to change for the next release.
- Fix writing METADATA file with multi-line information in certain fields such as `Author` (PR #402).
- Fix building wheel when a directory such as LICENSES appears in the project root directory (PR #401).

3.8.10 Version 3.1

- Update handling of names & version numbers in wheel filenames and `.dist-info` folders in line with changes in the specs (PR #395).
- Switch from the deprecated `pytoml` package to `toml` (PR #378).
- Fix specifying backend-path in `pyproject.toml` for flit-core (as a list instead of a string).

3.8.11 Version 3.0

Breaking changes:

- Projects must now provide Flit with information in `pyproject.toml` files, not the older `flit.ini` format (PR #338).
- `flit_core` once again requires Python 3 (>=3.4). Packages that support Python 2 can still be built by `flit_core` 2.x, but can't rely on new features (PR #342).
- The deprecated `flit installfrom` command was removed (PR #334). You can use `pip install git+https://github.com/...` instead.

Features and fixes:

- Fix building sdists from a git repository with non-ASCII characters in filenames (PR #346).
- Fix identifying the version number when the code contains a subscript assignment before `__version__ =` (PR #348).
- Script entry points can now use a class method (PR #359).
- Set suitable permission bits on metadata files in wheels (PR #256).
- Fixed line endings in the `RECORD` file when installing on Windows (PR #368).
- Support for recording the source of local installations, as in **PEP 610** (PR #335).
- `flit init` will check for a README in the root of the project and automatically set it as `description-file` (PR #337).

- Pygments is not required for checking reStructuredText READMEs (PR #357).
- Packages where the version number can be recognised without executing their code don't need their dependencies installed to build, which should make them build faster (PR #361).
- Ensure the installed `RECORD` file is predictably ordered (PR #366).

3.8.12 Version 2.3

- New projects created with *flit init* now declare that they require `flit_core >=2,<4` (PR #328). Any projects using `pyproject.toml` (not `flit.ini`) should be compatible with flit 3.x.
- Fix selecting files from a git submodule to include in an sdist (PR #324).
- Fix checking classifiers when no writeable cache directory is available (PR #319).
- Better errors when trying to install to a mis-spelled or missing Python interpreter (PR #331).
- Fix specifying `--repository` before `upload` (PR #322). Passing the option like this is deprecated, and you should now pass it after `upload`.
- Documentation improvements (PR #327, PR #318, PR #314)

3.8.13 Version 2.2

- Allow underscores in package names with Python 2 (PR #305).
- Add a `--no-setup-py` option to build sdists without a backwards-compatible `setup.py` file (PR #311).
- Fix the generated `setup.py` file for packages using a `src/` layout (PR #303).
- Fix detecting when more than one file matches the module name specified (PR #307).
- Fix installing to a venv on Windows with the `--python` option (PR #300).
- Don't echo the command in scripts installed with `--symlink` or `--pth-file` on Windows (PR #310).
- New `bootstrap_dev.py` script to set up a development installation of Flit from the repository (PR #301, PR #306).

3.8.14 Version 2.1

- Use compression when adding files to wheels.
- Added the *FLIT_INSTALL_PYTHON* environment variable (PR #295), to configure flit to always install into a Python other than the one it's running on.
- `flit_core` uses the `intreehooks` shim package to load its bootstrapping backend, until a released version of pip supports the standard `backend-path` mechanism.

3.8.15 Version 2.0

Flit 2 is a major architecture change. The `flit_core` package now provides a **PEP 517** backend for building packages, while `flit` is a *command line interface* extending that.

The build backend works on Python 2, so tools like pip should be able to install packages built with flit from source on Python 2. The `flit` command requires Python 3.5 or above. You will need to change the build-system table in your `pyproject.toml` file to look like this:

```
[build-system]
requires = ["flit_core >=2,<4"]
build-backend = "flit_core.buildapi"
```

Other changes include:

- Support for storing your code under a `src/` folder (PR #260). You don't need to change any configuration if you do this.
- Options to control what files are included in an sdist - see *Sdist section* for the details.
- Requirements can specify a URL 'direct reference', as an alternative to a version number, with the syntax defined in **PEP 440**: `requests @ https://example.com/requests-2.22.0.tar.gz`.
- Fix the shebang of scripts installed with the `--python` option and the `--symlink` flag (PR #286).
- Installing with `--deps develop` now installs normal dependencies as well as development dependencies.
- Author email is no longer required in the metadata table (PR #289).
- More error messages are now shown without a traceback (PR #254)

3.8.16 Version 1.3

- Fix for building sdists from a subdirectory in a Mercurial repository (PR #233).
- Fix for getting the docstring and version from modules defining their encoding (PR #239).
- Fix for installing packages with `flit installfrom` (PR #221).
- Packages with requirements no longer get a spurious `Provides-Extra: .none` metadata entry (#228).
- Better check of whether `python-requires` includes any Python 2 version (PR #232).
- Better check of home page URLs in `flit init` (PR #230).
- Better error message when the description file is not found (PR #234).
- Updated a help message to refer to `pyproject.toml` (PR #240).
- Improve tests of `flit init` (PR #229).

3.8.17 Version 1.2.1

- Fix for installing packages with `flit install`.
- Make `requests_download` an extra dependency, to avoid a circular build dependency. To use `flit installfrom`, you can install with `pip install flit[installfrom]`. Note that the `installfrom` subcommand is deprecated, as it will soon be possible to use pip to install Flit projects directly from a VCS URL.

3.8.18 Version 1.2

- Fixes for packages specifying `requires-extra`: sdists should now work, and environment markers can be used together with `requires-extra`.
- Fix running `flit installfrom` without a config file present in the working directory.
- The error message for a missing or empty docstring tells you what file the docstring should be in.
- Improvements to documentation on version selectors for requirements.

3.8.19 Version 1.1

- Packages can now have 'extras', specified as `requires-extra` in the *pyproject.toml file*. These are additional dependencies for optional features.
- The `home-page` metadata field is no longer required.
- Additional project URLs are now validated.
- `flit -V` is now equivalent to `flit --version`.
- Various improvements to documentation.

3.8.20 Version 1.0

- The description file may now be written in reStructuredText, Markdown or plain text. The file extension should indicate which of these formats it is (`.rst`, `.md` or `.txt`). Previously, only reStructuredText was officially supported.
- Multiple links (e.g. documentation, bug tracker) can now be specified in a new *[tool.flit.metadata.urls] section* of `pyproject.toml`.
- Dependencies are now correctly installed to the target Python when you use the `--symlink` or `--pth-file` options.
- Dependencies are only installed to the Python where Flit is running if it fails to get the docstring and version number without them.
- The commands deprecated in 0.13—`flit wheel`, `flit sdist` and `flit register`—have been removed.

Although version 1.0 sounds like a milestone, there's nothing that makes this release especially significant. It doesn't represent a step change in stability or completeness. Flit has been gradually maturing for some time, and I chose this point to end the series of 0.x version numbers.

3.8.21 Version 0.13

- Better validation of several metadata fields (`dist-name`, `requires`, `requires-python`, `home-page`), and of the version number.
- New *`FLIT_ALLOW_INVALID`* environment variable to ignore validation failures in case they go wrong.
- The list of valid classifiers is now fetched from Warehouse (https://pypi.org), rather than the older https://pypi.python.org site.
- Deprecated `flit wheel` and `flit sdist` subcommands: use *flit build*.
- Deprecated `flit register`: you can no longer register a package separately from uploading it.

3.8.22 Version 0.12.3

- Fix building and installing packages with a `-` in the distribution name.
- Fix numbering in README.

3.8.23 Version 0.12.2

- New tool to convert `flit.ini` to `pyproject.toml`:

```
python3 -m flit.tomlify
```

- Use the PAX tar format for sdists, as specified by PEP 517.

3.8.24 Version 0.12.1

- Restore dependency on `zipfile36` backport package.
- Add some missing options to documentation of `flit install` subcommand.
- Rearrange environment variables in the docs.

3.8.25 Version 0.12

- Switch the config to `pyproject.toml` by default instead of `flit.ini`, and implement the PEP 517 API.
- A new option `--pth-file` allows for development installation on Windows (where `--symlink` usually won't work).
- Normalise file permissions in the zip file, making builds more reproducible across different systems.
- Sdists (.tar.gz packages) can now also be reproducibly built by setting *`SOURCE_DATE_EPOCH`*.
- For most modules, Flit can now extract the version number and docstring without importing it. It will still fall back to importing where getting these from the AST fails.
- `flit build` will build the wheel from the sdist, helping to ensure that files aren't left out of the sdist.
- All list fields in the INI file now ignore blank lines (`requires`, `dev-requires`, `classifiers`).
- Fix the path separator in the `RECORD` file of a wheel built on Windows.
- Some minor fixes to building reproducible wheels.
- If building a wheel fails, the temporary file created will be cleaned up.

- Various improvements to docs and README.

3.8.26 Version 0.11.4

- Explicitly open various files as UTF-8, rather than relying on locale encoding.
- Link to docs from README.
- Better test coverage, and a few minor fixes for problems revealed by tests.

3.8.27 Version 0.11.3

- Fixed a bug causing failed uploads when the password is entered in the terminal.

3.8.28 Version 0.11.2

- A couple of behaviour changes when uploading to warehouse.

3.8.29 Version 0.11.1

- Fixed a bug when you use flit to build an sdist from a subdirectory inside a VCS checkout. The VCS is now correctly detected.
- Fix the rst checker for newer versions of docutils, by upgrading the bundled copy of readme_renderer.

3.8.30 Version 0.11

- Flit can now build sdists (tarballs) and upload them to PyPI, if your code is in a git or mercurial repository. There are new commands:
  - `flit build` builds both a wheel and an sdist.
  - `flit publish` builds and uploads a wheel and an sdist.
- Smarter ways of getting the information needed for upload:
  - If you have the keyring package installed, flit can use it to store your password, rather than keeping it in plain text in `~/.pypirc`.
  - If `~/.pypirc` does not already exist, and you are prompted for your username, flit will write it into that file.
  - You can provide the information as environment variables: *FLIT_USERNAME*, *FLIT_PASSWORD* and *FLIT_INDEX_URL*. Use this to upload packages from a CI service, for instance.
- Include 'LICENSE' or 'COPYING' files in wheels.
- Fix for `flit install --symlink` inside a virtualenv.

3.8.31 Version 0.10

- Downstream packagers can use the *FLIT_NO_NETWORK* environment variable to stop flit downloading data from the network.

3.8.32 Version 0.9

- `flit install` and `flit installfrom` now take an optional `--python` argument, with the path to the Python executable you want to install it for. Using this, you can install modules to Python 2.
- Installing a module normally (without `--symlink`) builds a wheel and uses pip to install it, which should work better in some corner cases.

3.8.33 Version 0.8

- A new `flit installfrom` subcommand to install a project from a source archive, such as from Github.
- *Reproducible builds* - you can produce byte-for-byte identical wheels.
- A warning for non-canonical version numbers according to PEP 440.
- Fix for installing projects on Windows.
- Better error message when module docstring is only whitespace.

3.8.34 Version 0.7

- A new `dev-requires` field in the config file for development requirements, used when doing `flit install`.
- Added a `--deps` option for `flit install` to control which dependencies are installed.
- Flit can now be invoked with `python -m flit`.

3.8.35 Version 0.6

- `flit install` now ensures requirements specified in `flit.ini` are installed, using pip.
- If you specify a description file, flit now warns you if it's not valid reStructuredText (since invalid reStructuredText is treated as plain text on PyPI).
- Improved the error message for mis-spelled keys in `flit.ini`.

3.8.36 Version 0.5

- A new `flit init` command to quickly define the essential basic metadata for a package.
- Support for entry points.
- A new `flit register` command to register a package without uploading it, for when you want to claim a name before you're ready to release.
- Added a `--repository` option for specifying an alternative PyPI instance.
- Added a `--debug` flag to show debug-level log messages.
- Better error messages when the module docstring or `__version__` is missing.

3.8.37 Version 0.4

- Users can now specify `dist-name` in the config file if they need to use different names on PyPI and for imports.
- Classifiers are now checked against a locally cached list of valid classifiers.
- Packages can be locally installed into environments for development.
- Local installation now creates a PEP 376 `.dist-info` folder instead of `.egg-info`.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

INDEX

Symbols

E

F

P

S