Flit Documentation

Release 0.11.4

Thomas Kluyver

Sep 17, 2017

Contents

Flit is a simple way to put Python packages and modules on PyPI.

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. Create a file `flit.ini` next to the module. It should look like this:

```
[metadata]
module=foobar
author=Sir Robin
author-email=robin@camelot.uk
home-page=http://github.com/sirrobin/foobar

# If you want command line scripts, this is how to declare them.
# If not, you can leave this section out completely.
[scripts]
# foobar:main means the script will do: from foobar import main; main()
foobar=foobar:main
```

   You can use `flit init` to easily create a basic `flit.ini` file for your package.

   Besides the details shown above, there are other fields you can add—see the flit.ini page of the docs.

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

```
pip install flit
```

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

```
flit publish
```

If your package is not registered on PyPI yet, flit will try to register it for you during the upload step.

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 subpackages and data files within a package are included automatically.

Flit requires Python 3, but you can use it to distribute modules for Python 2, so long as they can be imported on Python 3.

See Flit's documentation for more information.

Contents:

CHAPTER 1

The flit.ini config file

This file lives next to the module or package.

Metadata section

There are four 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

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

e.g. for flit itself

```
[metadata]
module=flit
author=Thomas Kluyver
author-email=thomas@kluyver.me.uk
home-page=https://github.com/takluyver/flit
```

The remaining fields are optional:

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

```
requires = requests (>=2.6)
      configparser; python_version == '2.7'
```

dev-requires Packages that are required for development. This field is in the same format as `requires`.

These are not (yet) encoded in the wheel, but are used when doing `flit install`.

description-file A path (relative to the .ini file) to a file containing a longer description of your package to show on PyPI. This should be written in reStructuredText, if your long description is not valid reStructuredText, a warning will be printed, and it will be interpreted as plain text on PyPI.

classifiers A list of Trove classifiers, one per line, indented.

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 Space 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's the full example from flit itself:

```
[metadata]
author=Thomas Kluyver
author-email=thomas@kluyver.me.uk
home-page=https://github.com/takluyver/flit
requires=requests
requires-python= >=3
description-file=README.rst
classifiers=Intended Audience :: Developers
    License :: OSI Approved :: BSD License
    Programming Language :: Python :: 3
    Topic :: Software Development :: Libraries :: Python Modules
```

Scripts section

Each key and value in this describes a shell command to be installed along with your package. These work like setuptools 'entry points'. Here's the section for flit:

```
[scripts]
flit = flit:main
```

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

CHAPTER 2

Specifying entry points

The most common use of entry points is the `console_scripts` section for installing system commands. You can specify these in the *Scripts section* of `flit.ini`.

If you need other entry points, e.g. to distribute a plugin for an application, you should store these in an `entry_points.txt` file next to `flit.ini`. The format is like this:

```
[group]
name1=package.module:func
name2=package:obj

# e.g.
[calculator.plugins]
romannumerals=romancalc:init
```

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 you need to name the entry points file something else, you can tell flit its name by adding a `entry-points-file` key in the `[metadata]` section of `flit.ini`.

CHAPTER 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:

Using .pypirc

You can create or edit a config file in your home directory, `~/.pypirc`. 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 --repository testpypi publish
```

If you don't use this option, Flit will use the server called `pypi` in the config file. If that doesn't exist, it uses

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 = ...`)

Using environment variables

FLIT_USERNAME
FLIT_PASSWORD
FLIT_INDEX_URL
New in version 0.11.

Set a username, password, and index URL for uploading packages.

Environment variables take precedence over the config file, except if you use the `--repository` option to explicitly pick a server from the config file.

CHAPTER 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 are a couple of caveats, however:

First, 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 an environment variable:

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

Zip files also record the permission bits on a file. Checking out a repository on computers with different umasks can result in different permissions - a file that has mode `644` on Ubuntu may have `664` on Fedora. If you're concerned about this, normalise the permissions before using flit. Normalisation might be added in a future version.

CHAPTER 5

Release history

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.

Version 0.11.3

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

Version 0.11.2

- A couple of behaviour changes when uploading to warehouse.

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.

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.

Version 0.10

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

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.

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.

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`.

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`.

Version 0.5

- A new `flit init` command to quickly define the essential basic metadata for a package.
- Support for *Specifying 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.

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 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.

There are other environment variables to control *uploading packages*. and *reproducible builds*.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index