Packaging and Distributing

Create, package and distribute your own python application

José Antonio Perdiguero López
http://www.perdy.top
https://github.com/PeRDy
https://www.linkedin.com/in/perdy
@ perdy.hh@gmail.com

September 24, 2017

Lead Developer @ Piksel, Machine Learning Team

Index

Introduction

Why should I distribute my application?

- Given enough eyeballs, all bugs are shallow.
- Upstream improvements.
- Force multiplier.
- Modular.
- Great advertising.
- Attract talent.
- Stand on the shoulders of giants.
- Best technical interview possible.
- Show your code.

[1] https://opensource.com/life/15/12/why-open-source

Versioning

Version schema

A normal version must be denoted by `X.Y.Z` where `X`, `Y` and `Z` are positive integers. `X` represents the major version, `Y` the minor version and `Z` the patch version. Version `1.0.0` defines the public API.

Version upgrade

Given a version number, increment:

Major version when you make incompatible API changes. Reset minor and patch version to `0`.

Minor version when you add functionality in a backwards-compatible manner. Reset patch version to `0`.

Patch version when you make backwards-compatible bug fixes.

[1] http://semver.org/

Tools I

Prospector

Static code analysis using different tools.

https://github.com/landscapeio/prospector

Tools I

Prospector

Static code analysis using different tools.
https://github.com/landscapeio/prospector

Sphinx

Create documentation for your project.
https://github.com/sphinx-doc/sphinx

Tools I

Prospector

Static code analysis using different tools.
https://github.com/landscapeio/prospector

Sphinx

Create documentation for your project.
https://github.com/sphinx-doc/sphinx

Bumpversion

Utility to upgrade your project version.
https://github.com/peritus/bumpversion

Tools I

Prospector

Static code analysis using different tools.
https://github.com/landscapeio/prospector

Sphinx

Create documentation for your project.
https://github.com/sphinx-doc/sphinx

Bumpversion

Utility to upgrade your project version.
https://github.com/peritus/bumpversion

Pre-commit

Utility that does some checks before git commits.
https://github.com/pre-commit/pre-commit

Tools II

Tox

Run your tests using many different python interpreters.
https://github.com/tox-dev/tox

Tools II

Tox

Run your tests using many different python interpreters.
https://github.com/tox-dev/tox

Cookiecutter

Application that creates project skeletons using Jinja templates.
https://github.com/audreyr/cookiecutter

Tools II

Tox

Run your tests using many different python interpreters.
https://github.com/tox-dev/tox

Cookiecutter

Application that creates project skeletons using Jinja templates.
https://github.com/audreyr/cookiecutter

Cookiecutter Template

Cookiecutter template for Python packages.
https://github.com/PeRDy/cookiecutter-python-package

Tools II

Tox

Run your tests using many different python interpreters.
https://github.com/tox-dev/tox

Cookiecutter

Application that creates project skeletons using Jinja templates.
https://github.com/audreyr/cookiecutter

Cookiecutter Template

Cookiecutter template for Python packages.
https://github.com/PeRDy/cookiecutter-python-package

Clinner

Utility to create powerful Command Line Interfaces with a few lines.
https://github.com/PeRDy/clinner

Services

PyPI

Python Package Index, the main repository of python software.
https://pypi.python.org/pypi

Services

PyPI

Python Package Index, the main repository of python software.
https://pypi.python.org/pypi

GitHub

Repositories for your source code. https://github.com

Services

PyPI

Python Package Index, the main repository of python software.
https://pypi.python.org/pypi

GitHub

Repositories for your source code. https://github.com

Travis

Continuous Integration service. https://travis-ci.org/

Services

PyPI

Python Package Index, the main repository of python software.
https://pypi.python.org/pypi

GitHub

Repositories for your source code. https://github.com

Travis

Continuous Integration service. https://travis-ci.org/

Codecov

Keeps the changes of test coverage of your code. https://codecov.io

Services

PyPI

Python Package Index, the main repository of python software.
https://pypi.python.org/pypi

GitHub

Repositories for your source code. https://github.com

Travis

Continuous Integration service. https://travis-ci.org/

Codecov

Keeps the changes of test coverage of your code. https://codecov.io

ReadTheDocs

Stores and serves documentation for your project. https://readthedocs.io

Creation

Storing the project

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner **Repository name**

PeRDy / foo

Great repository names are short and memorable. Need inspiration? How about **furry-adventure**.

Description (optional)

Foo project

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None**

Create repository

Create the project skeleton

Cookiecutter context

Define all variables needed by cookiecutter to properly create the project skeleton, these variables can be found in *cookiecutter.json* file.

Create the project skeleton

Cookiecutter context

Define all variables needed by cookiecutter to properly create the project skeleton, these variables can be found in *cookiecutter.json* file.

Create skeleton

Execute cookiecutter with previously defined context to create the project skeleton.

Create the project skeleton

Cookiecutter context

Define all variables needed by cookiecutter to properly create the project skeleton, these variables can be found in *cookiecutter.json* file.

Create skeleton

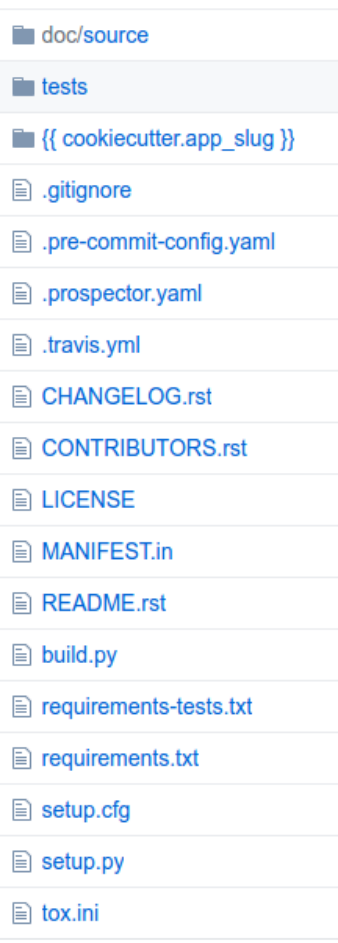
Execute cookiecutter with previously defined context to create the project skeleton.

Commit & push

Time to do your first commit and push to repository:

```
git remote add origin git@github.com:PeRDy/foo.git
git commit -a -m "Initial commit"
git push
```

Project hierarchy



Documentation folder

The place that keeps all the documentation source files as well as the doc config file.

Project hierarchy

- doc/source
- tests
- {{ cookiecutter.app_slug }}
- .gitignore
- .pre-commit-config.yaml
- .prospector.yaml
- .travis.yml
- CHANGELOG.rst
- CONTRIBUTORS.rst
- LICENSE
- MANIFEST.in
- README.rst
- build.py
- requirements-tests.txt
- requirements.txt
- setup.cfg
- setup.py
- tox.ini

Documentation folder

The place that keeps all the documentation source files as well as the doc config file.

Tests folder

All tests files are stored in a tests folder where tests collectors can gather them without problems.

Project hierarchy

- doc/source
- tests
- {{ cookiecutter.app_slug }}
- .gitignore
- .pre-commit-config.yaml
- .prospector.yaml
- .travis.yml
- CHANGELOG.rst
- CONTRIBUTORS.rst
- LICENSE
- MANIFEST.in
- README.rst
- build.py
- requirements-tests.txt
- requirements.txt
- setup.cfg
- setup.py
- tox.ini

Documentation folder

The place that keeps all the documentation source files as well as the doc config file.

Tests folder

All tests files are stored in a tests folder where tests collectors can gather them without problems.

Application folder

The application itself, the *python package* distributed, and the same that other users will import in their applications.

Project hierarchy

- doc/source
- tests
- {{ cookiecutter.app_slug }}
- .gitignore
- .pre-commit-config.yaml
- .prospector.yaml
- .travis.yml
- CHANGELOG.rst
- CONTRIBUTORS.rst
- LICENSE
- MANIFEST.in
- README.rst
- build.py
- requirements-tests.txt
- requirements.txt
- setup.cfg
- setup.py
- tox.ini

Documentation folder

The place that keeps all the documentation source files as well as the doc config file.

Tests folder

All tests files are stored in a tests folder where tests collectors can gather them without problems.

Application folder

The application itself, the *python package* distributed, and the same that other users will import in their applications.

Root files

Files that keeps in the root directory are usually:

- Tools configuration.
- Services configuration.
- Build scripts.
- Metadata.

Relevant files I

Manifest

This file, `MANIFEST.in`, with own syntax[1] defines the directories and files that will be included in the distributable package.

[1] https://docs.python.org/3/distutils/commandref.htmlsdist-cmd

Relevant files I

Manifest

This file, `MANIFEST.in`, with own syntax[1] defines the directories and files that will be included in the distributable package.

Requirements

List all requirements of your project, that are added as dependencies when installed. Usually requirements are splitted in two files:
`requirements.txt` for real dependencies and
`requirements-tests.txt` for dependencies necessaries to test the project.

[1] https://docs.python.org/3/distutils/commandref.htmlsdist-cmd

Relevant files I

Manifest

This file, `MANIFEST.in`, with own syntax[1] defines the directories and files that will be included in the distributable package.

Requirements

List all requirements of your project, that are added as dependencies when installed. Usually requirements are splitted in two files:
`requirements.txt` for real dependencies and
`requirements-tests.txt` for dependencies necessaries to test the project.

Metadata

Metadata files: `README.rst`, `CONTRIBUTORS.rst`, `CHANGELOG.rst` and `LICENSE`.

[1] https://docs.python.org/3/distutils/commandref.htmlsdist-cmd

Relevant files II

Tools and Services config

Configuration files for tools and services: `setup.cfg`, `.pre-commit-config.yaml`, `.prospector.yaml`, `.travis.yml`, `.gitignore`.

Relevant files II

Tools and Services config

Configuration files for tools and services: `setup.cfg`, `.pre-commit-config.yaml`, `.prospector.yaml`, `.travis.yml`, `.gitignore`.

Setup

Main file that defines how the project will be packaged, gather metadata from other files and provides an interface to create distributable packages.

Relevant files II

Tools and Services config

Configuration files for tools and services: `setup.cfg`, `.pre-commit-config.yaml`, `.prospector.yaml`, `.travis.yml`, `.gitignore`.

Setup

Main file that defines how the project will be packaged, gather metadata from other files and provides an interface to create distributable packages.

Tox

Tox file, `tox.ini`, defines the environments and commands that tox executes. In this case, defines an environment for each python version that should be tested, another for run lint tools and the last one for compile documentation.

Relevant files III

Build

The build file, `build.py`, is the entrypoint for everything related to build, including *testing*, *packaging* and *distributing*. This is a command line application using *Clinner* that provides a set of utility commands such as:

- Run tests and code coverage.
- Run lint.
- Run tox.
- Create documentation.
- Upgrade version, create package and upload to pypi.

Packaging

Test your application

Development

Run tests while developing using pytest.

```
python build.py pytest
```

Test your application

Development

Run tests while developing using pytest.

```
python build.py pytest
```

Multi-environment

Once development is done, run tests against all different interpreters supported to check compatibility.

```
python build.py tox
```

Test your application

Development

Run tests while developing using pytest.

```
python build.py pytest
```

Multi-environment

Once development is done, run tests against all different interpreters supported to check compatibility.

```
python build.py tox
```

Continuous Integration

Development is done, tests pass in every environment, so code can be uploaded to repository safely. Once a commit is done:

Travis run tests in every environment and will notify in case any test didn't pass. When all tests pass,

Codecov records current code coverage. In the same commit,

ReadTheDocs gets the code, build docs and updates the project's doc page.

Package types

Egg

Source distribution.

```
python setup.py sdist
```

Package types

Egg

Source distribution.

```
python setup.py sdist
```

Wheel

Built and binary distribution.

```
python setup.py bdist_wheel
```

Builder

```
[ ⓥclinner ]( ~/Desarrollo/clinner )[ ᚠ:master 1730ade ]──────────────────( @perdy-xps )
❯❯❯ python build.py -h                                                  +1483 0:55:16
usage: build.py [-h] [-s SETTINGS] [-q] [--dry-run]
                {pytest,prospector,sphinx,tox,dist} ...

optional arguments:
  -h, --help            show this help message and exit
  -s SETTINGS, --settings SETTINGS
                        Module or object with Clinner settings in format
                        "package.module[:Object]"
  -q, --quiet           Quiet mode. No standard output other than executed
                        application
  --dry-run             Dry run. Skip commands execution, useful to check
                        which commands will be executed and execution order

Commands:
  {pytest,prospector,sphinx,tox,dist}
    pytest              Run unit tests
    prospector          Run prospector lint
    sphinx              Sphinx doc
    tox                 Run tox
    dist                Bump version, create package and upload it
```

Distributing

Register the application

PyPI account

Create a PyPI[2] account. Configure `.pypirc` file with PyPI credentials.

[2] https://pypi.python.org/pypi
[3] https://github.com/pypa/twine

Register the application

PyPI account

Create a PyPI[2] account. Configure `.pypirc` file with PyPI credentials.

Application register

Use twine[3] with a package of your application to register it in PyPI:

```
twine register dist/project-name.whl
```

[2] https://pypi.python.org/pypi
[3] https://github.com/pypa/twine

Upload a new version

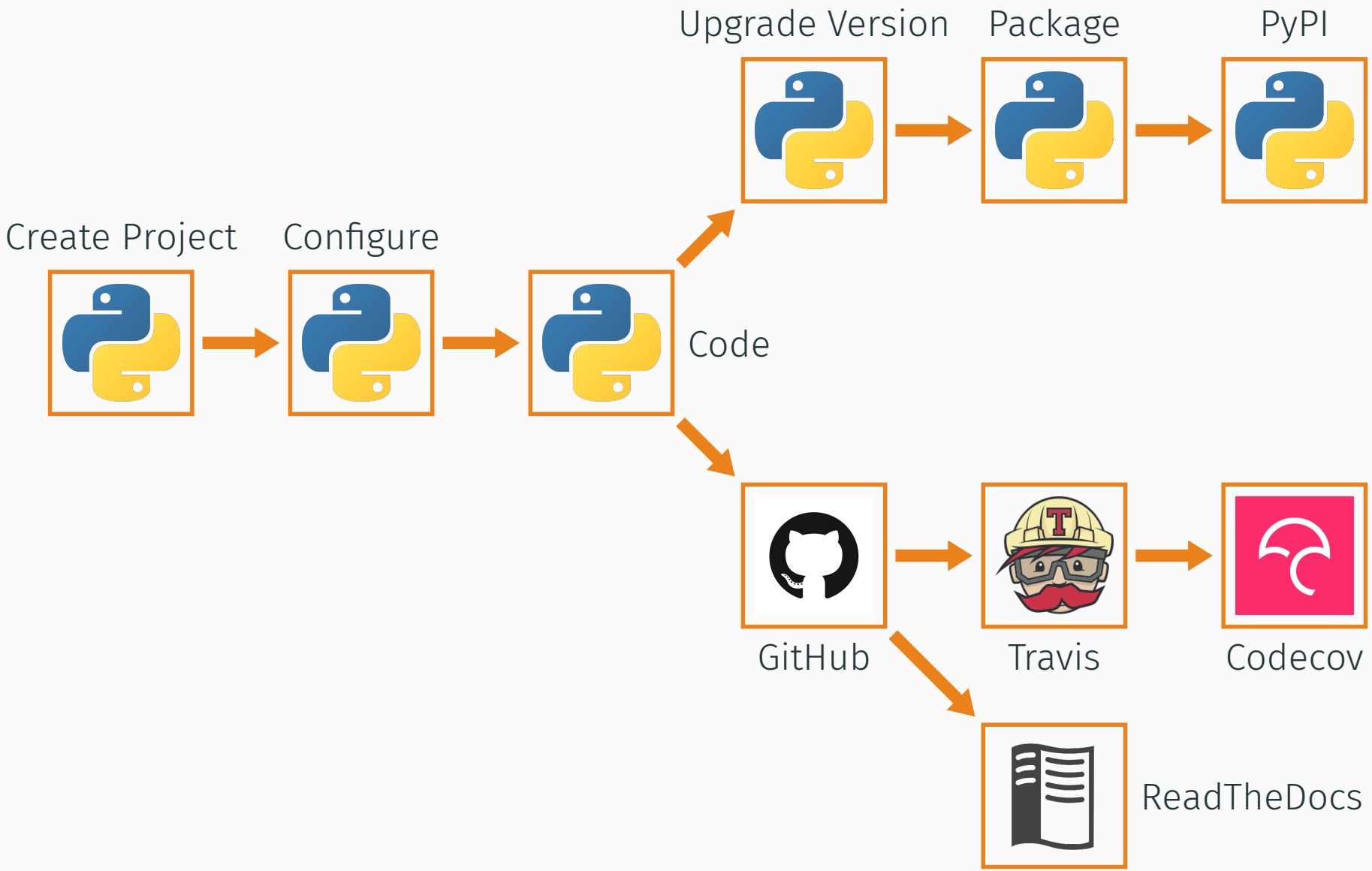
Upload packages

Use twine again to upload all packages to PyPI:

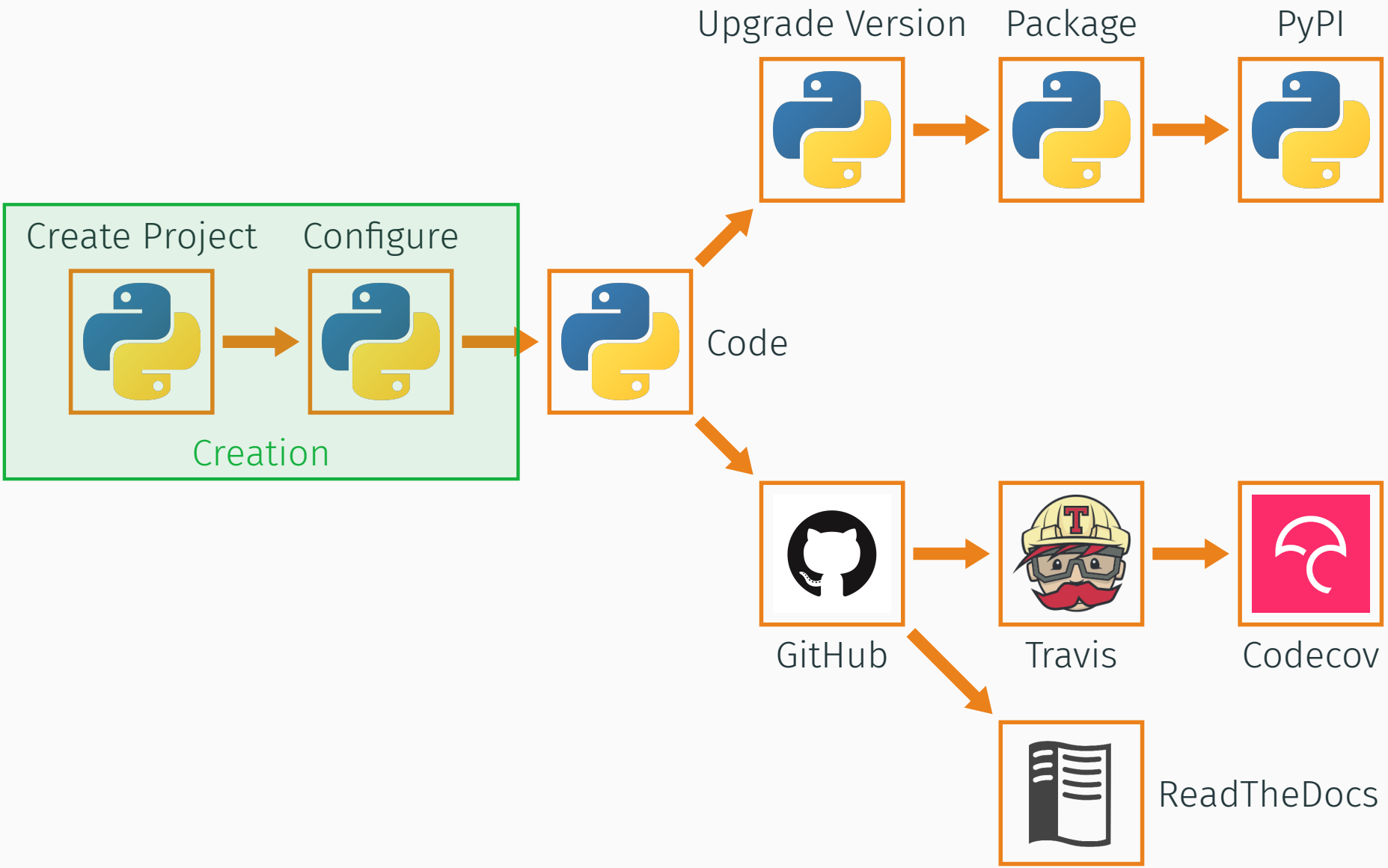
```
twine upload dist/project-name.whl
twine upload dist/project-name.tar.gz
```

Conclusion

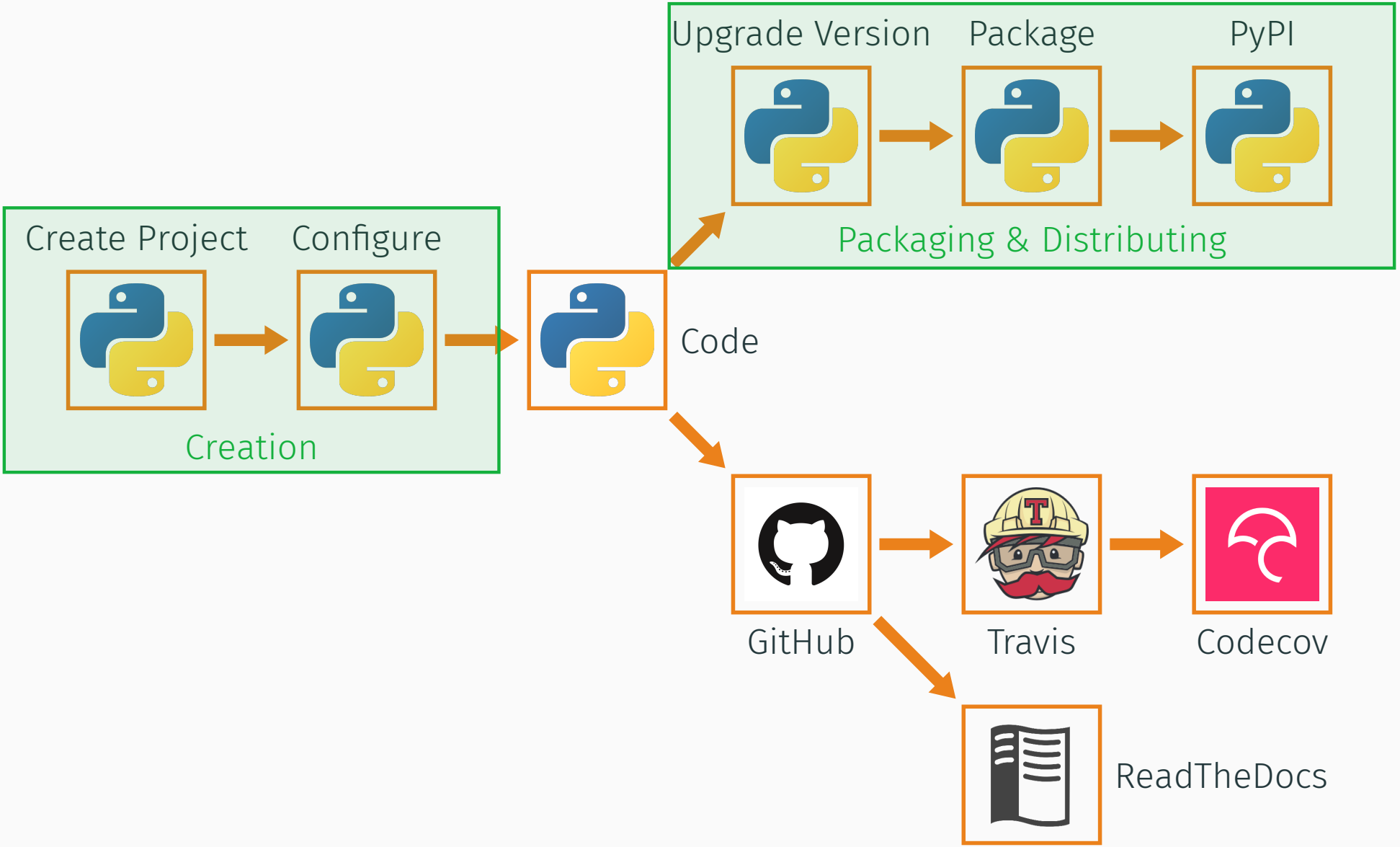
Full workflow



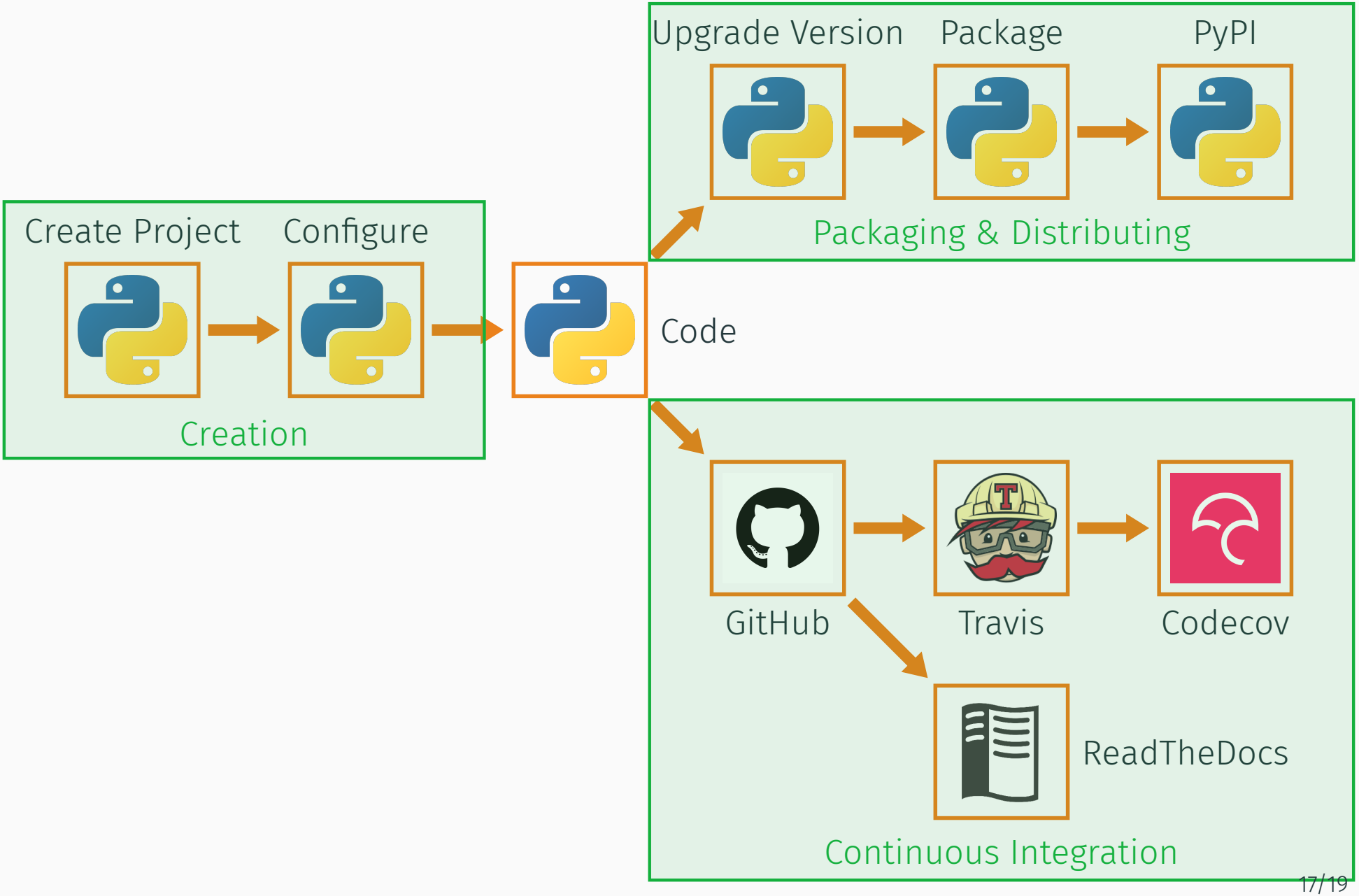
Full workflow



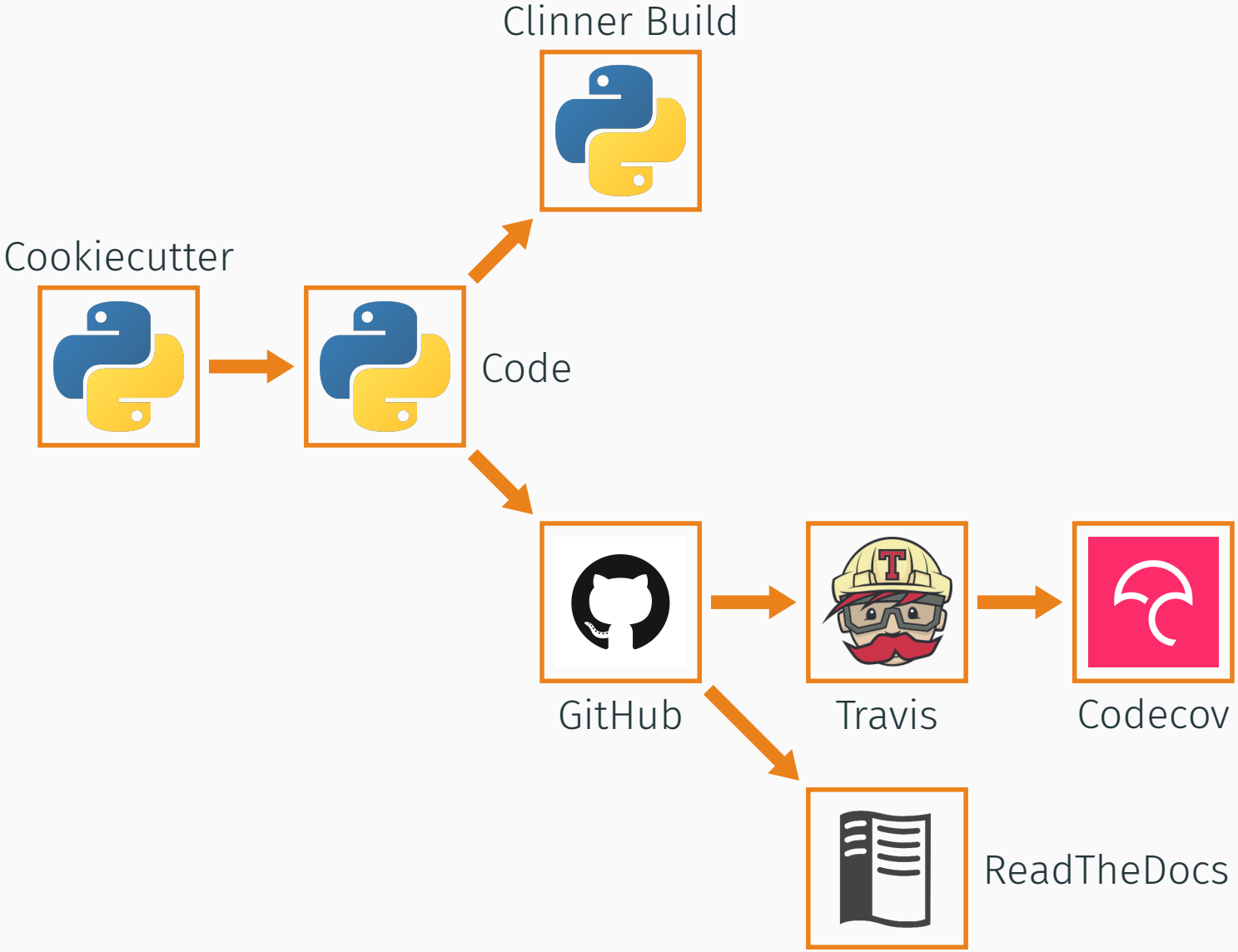
Full workflow



Full workflow



Simplified workflow I



Simplified workflow II

Workflow execution

```
cookiecutter <project_name>
...code...
python build.py dist (patch|minor|major)
git push
```

Open source your code !