autopilot Documentation

Release 0.3.0

José Luis Lafuente

February 04, 2016

> There should be one– and preferably only one –obvious way to do it.
>
> —The Zen of Python, by Tim Peters

There are too many options to create and distribute a Python package. Probably that's why many people hate package distribution in Python:

Autopilot tries to provide good defaults, avoiding you from repetitive tasks and bikeshedding.

CHAPTER 1

Paradox of choice

There are many choices when you create a Python package, but many of them are choices on things that don't really matter. Just make the choice once and move on. Conventions are good for things like choose where to store the version number of your project. Nobody cares about it and don't thinking about it makes you more productive. Autopilot makes this choices for you, and this way can easily automate your package release. Of course, the choices are not written in stone and can change over time, but the main idea is to change them only if there is a good reason.

There is a talk from Yehuda Katz about defining conventions, which I really recommend:

Link to youtube

CHAPTER 2

On giants' shoulders

These resources were an inspiration for autopilot:

- Python Packaging User Guide
- Blog post from Ionel Cristian Mărie
- Zest.realeaser

Contents:

2.1 Introduction

2.1.1 Requisites

autopilot requires Python >= 3.5

2.1.2 Installation

```
pip install autopilot
```

2.1.3 Basic usage

Autopilot provides 2 commands, `new` and `release`. You can call them from the command line:

```
$ ap new
$ ap release
```

`ap new` command creates a new project. You can pass optionally the new project name:

```
$ ap new cool-project
```

`ap release` creates a new release for a project. If you want to upload it to a PyPI server, you need to create a configuration file to tell autopilot which are your username and password. See next section for more info.

2.1.4 Configuration

We choose the yaml format for the configuration. This is the default configuration file:

```
    author:
      name: ''            # Take it from git config if empty
      email: ''           # Take it from git config if empty

    new_project:
      default_dir: ''    # Uses current working directory
      license: gplv2
      commit: true        # Do the initial commit?

    editor: ''            # Use $EDITOR, fallback to vim

    release:
      upload: PyPI
      push: true

    pypi_servers:
      PyPI:
        user: ''
        passeval: ''     # Executable command
        url: 'https://pypi.python.org/pypi'
      PyPI Test:
        user: ''
        passeval: ''     # Executable command
        url: 'https://testpypi.python.org/pypi'
```

For some options, if they are empty, autopilot will try to fill the data with information from your system, see comments in the default configuration file.

It is possible to override the default configuration. To do it, create a file at `XDG_CONFIG_HOME/autopilot/config.yml` (or `~/.config/autopilot/config.yml` if `XDG_CONFIG_HOME` is not defined). You can override just some of the options. For every option, the logic is to look for the option in the user configuration file, if it's not there, search in autopilot default configuration, and if the option has an empty value, try to get the value from the system.

An example of a custom configuration file:

```
new_project:
  license: mit

editor: vim -R

release:
  upload: nope
  push: false

pypi_servers:

  pypi:
    user: my_pypi_user
    passeval: pass pypi

  local devpi:
    user: devpi_user
    passeval: pass devpi_local
    url: 'http://localhost:8080'
```

As you see, is possible to add more servers to `pypi_servers`. In this example, a new PyPI server (`local devpi`) would be added to the list of options on the UI.

2.1.5 Version handling

According to PEP 396, the version should be available in the `__version__` attribute, should be a string, and have only one source and all the others should be derived from it.

When you do a new release, with the `ap release` command, autopilot will generate two new version numbers, the number for the release you are doing, and the next development version.

For the new release number, autopilot just remove the pre-release part from the version number. For example, if your current version is `1.0.0a1` or `1.0.0.dev0`, the next version number will be `1.0.0`.

For the next development version number, autopilot just increments the minor number of the release and adds a *.dev0*. For example, if you are going to release version `1.0.0`, your next development version will be `1.0.1.dev0`.

The new version numbers generated by autopilot are just a suggestion, you can always change the version numbers through the user interface. If you update the version number manually, autopilot will check that the number is a valid number according to PEP 440

Also notice that when you update your release version number, autopilot will automatically update the next development version number.

2.1.6 Command line options

You can append *–help* to any command to get a description of the command.

```
$ ap --help
Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/bin/ap", line 9,
    load_entry_point('autopilot==0.3.0', 'console_scripts', 'ap')()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return get_distribution(dist).load_entry_point(group, name)
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return ep.load()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return self.resolve()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    module = __import__(self.module_name, fromlist=['__name__'], level=0)
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    from . import render
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    **license_variables
     ^

SyntaxError: invalid syntax
```

```
$ ap new --help
Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/bin/ap", line 9,
    load_entry_point('autopilot==0.3.0', 'console_scripts', 'ap')()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return get_distribution(dist).load_entry_point(group, name)
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return ep.load()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return self.resolve()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    module = __import__(self.module_name, fromlist=['__name__'], level=0)
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    from . import render
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    **license_variables
     ^
SyntaxError: invalid syntax
```

```
$ ap release --help
Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/bin/ap", line 9,
    load_entry_point('autopilot==0.3.0', 'console_scripts', 'ap')()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return get_distribution(dist).load_entry_point(group, name)
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return ep.load()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    return self.resolve()
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    module = __import__(self.module_name, fromlist=['__name__'], level=0)
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    from . import render
  File "/home/docs/checkouts/readthedocs.org/user_builds/autopilot-docs/envs/stable/lib/python3.4/sit
    **license_variables
     ^
SyntaxError: invalid syntax
```

The `--type` option is used to generate the next version number, but you can always modify this number on the ncurses interface, before the new release is created.

2.2 Choices

2.2.1 Structure

In our top level directory, we have the following folders:

- src/${project_name}

  Note: This way you get import parity, see http://blog.ionelmc.ro/2014/05/25/python-packaging/#the-structure

- docs
- tests

And the following files:

- setup.py
- README.rst
- CHANGELOG.rst
- LICENSE
- .gitignore
- requirements.txt

Note: Here you have your dev requirements

- setup.cfg

Note: This file is for external utilities configuration

- tox.ini

Note: Test both with coverage measurements and without. See http://blog.ionelmc.ro/2014/05/25/python-packaging/#tl-dr For coverage we do a *pip install -e*, but test with a normal pip install are also great.

- .travis.yml

Note: Use tox file here.

2.2.2 Project version

In your `src/${project_name}/__init__.py` file. We can extract it later using regex on our *setup.py*. But it is also possible to get the version number in our python code:

```
>>> import autopilot
>>> autopilot.__version__
'0.2.1'
```

2.2.3 Git workflow

http://endoflineblog.com/gitflow-considered-harmful

2.2.4 License

Autopilot includes some default licenses, like *GPLv2* or *MIT*. You need to choose one, which would be copied into the *LICENSE* file, at the top directory of your project. There is an special type of license, *Private*. This one doesn't add a license file (well, it creates a *LICENSE* file, but is just link to the GNU licenses website). This license also adds a classifier to your package, "Private :: Do Not Upload". PyPI will refuse to accept packages with unknown classifiers, hence we want to use it for private packages to protect ourselves from a mistake. Anyways, if we have a private devpi we still can upload the package there.

If you want to add a new license to the list of licenses, put the license file at *$XDG_CONFIG_HOME/autopilot/licenses* directory (usually *~/.config/autopilot/licenses*). Filename would be used to generate the selectable list of licenses. By default, user licenses are private, and the *Private* classifier would be added to your *setup.py*. If you want to change this behavior, the first line of your license must be like this:

```
# pypi license: License name
```

where license name must be a valid license name listed here: `license_list`

The list was extracted from PyPI list of classifiers. For the OSI licenses you can remove *'OSI Approved ::'* from the license name.

That first line, and all the empty lines after them, are not copied to your *LICENSE* file.

2.2.5 Changelog

You should maintain a changelog file (*CHANGELOG.rst*). When you do a new release, it is possible to open the changelog with a text editor, but any changes you do to the file, would be discarded. For that reason, is recommended to open the editor in read-only mode. If you use vim, you can set the editor to *vim -R* on your local autopilot configuration:

```
editor: vim -R
```

2.2.6 Files to distribute

With autopilot you can avoid to write and maintain a `MANIFEST.in` file. All files under your `src` directory would be included in the package, unless you exclude them using a `gitignore` file. Also `README.txt` (or `README`), `setup.py` (or whatever you called your setup script), and `setup.cfg` files are included by default by `setuptools` (see distutils docs)

Autopilot also adds `CHANGELOG.rst`, `LICENSE` and `.gitignore` files to the package.

Take a look to setuptools documentation if you want to know how autopilot is able to modify the files to include in the package.

2.3 Changelog for autopilot

2.3.1 0.3.0 (2016-02-04)

- Add *–version* option, which prints out the current version number
- Add release type option to *release* command. Possible values are patch, minor and major. The next version numbers are generated based on this option

2.3.2 0.2.2 (2016-02-02)

- Fix, unused import was raising an exception

2.3.3 0.2.1 (2016-01-31)

- Remove twine dependency, now we use distlib to upload the packages
- Fix, use package version on sphinx documentation
- Fix, changelog was not properly generated for new releases

2.4 Indices and tables

- genindex
- modindex
- search