SecureDrop Debian Packaging Guide Documentation

Release 0.1

Kushal Das

Oct 31, 2018

Contents:

Note: This is a live documentation, means it will be changed regularly as we update the guide.

CHAPTER 1

Notes to the Python developer

Even before we start discussing Debian packaging, first we are going to look into a few guideline for the project development based on Python.

All new projects will be maintained for Debian Stretch, it means all of code has to run on `Python 3.5.3`. This means we have to make sure we do not use any later language feature or any module which is dependent on the latest Python version. `Python 3.5.3` is our runtime dependency.

But, this does not mean that we can not use tools which are dependent on the latest version of Python. For example, Black requires at least Python 3.6, and we can easily have it on the developer's system (say using a container or inside of a vm), and make sure that we format our code using Black. But, this is not a runtime dependency.

1.1 Example project git repository

We will use the whosaysthat repository as the example project for the rest of this documentation.

1.2 Check security implication before using any new Python module

Before you start using any new Python module from PyPI, make sure you get a sign-off from the rest of the team. You should look for the number of maintainers, and active issue-tracker and other users of any module before start using it for a project.

1.3 Use Pipenv for development environment setup

Use pipenv tool to maintain your project's dependencies.

For building the final Debian packages, we need every Python dependency to be installable from the PyPI from source. We need this so that we can track the sha265sums of the source tarballs.

Do not add any of the following dependencies in *Pipfile*, use Debian packages from Debian.

- PyQt5

Create the virtualenv using the following command

```
$ pipenv install -d --site-packages
```

1.4 Debian package names for dependencies

Use the following for PyQy5

- python3-pyqt5
- python3-pyqt5.qtsvg

1.5 Writing your setup.py

The `setup.py` is key point deciding how the project will get installed. If you want to know about all the available options, this guide will help you to find the details.

You can start from this *example setup.py <https://github.com/kushaldas/whosaysthat/blob/master/setup.py>_*.

- Follow semver guide for the version number of the project.
- Add a correct LICENSE file in the project repository.
- All command line tools should be marked clearly as console entry points
- Add a `__main__.py` as required in the package/module so that it can be invoked as `python3 -m modulename`.
- Remember that `pipenv install -e .` will enable the console scripts while you are developing the application.
- Do not install or copy any configuration file or other data using setup.py, we will use Debian package for the same.

1.6 MANIFEST.in file

All of the files which should be installed into the end user system, should be mentioned in the `MANIFEST.in` file. Here is one example of our example project.

Remember to add *requirements-build.txt* and *requirement.txt* to the manifest file.

1.7 Releasing a project

`python3 setup.py sdist` command will create a tarball of the package under `dist/` directory. Please verify the STDOUT output of the above mentioned command as it will tell you what all files where put inside of the tar file. For any modification, update the `MANIFEST.in` file accordingly.

The maintainer can then sign and release the tarball in a pre-defined location.

Remember to read the whole guide first before release any application for this workflow.

CHAPTER 2

Debian packaging 101

In this chapter we will package our example project into a Debian package and learn about the process and different tools and files involved.

2.1 Install the required package build tools

We will need the required build tools in the system.

```
$ sudo apt-get install build-essential devscripts dh-python python3-all python3-
↪setuptools dh-make
```

2.2 Get the source tarball

Clone the example project anywhere you like, and create an release tarball using the following command.

```
$ git clone https://github.com/kushaldas/whosaysthat.git
Cloning into 'whosaysthat'...
remote: Counting objects: 49, done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 49 (delta 15), reused 44 (delta 11), pack-reused 0
Unpacking objects: 100% (49/49), done.

$ cd whosaysthat
$ $ python3 setup.py sdist
/usr/lib/python3.5/distutils/dist.py:261: UserWarning: Unknown distribution option:
↪'long_description_content_type'
warnings.warn(msg)
running sdist
running egg_info
creating whosaysthat.egg-info
```

(continues on next page)

(continued from previous page)

```
writing whosaysthat.egg-info/PKG-INFO
writing dependency_links to whosaysthat.egg-info/dependency_links.txt
writing entry points to whosaysthat.egg-info/entry_points.txt
writing top-level names to whosaysthat.egg-info/top_level.txt
writing requirements to whosaysthat.egg-info/requires.txt
writing manifest file 'whosaysthat.egg-info/SOURCES.txt'
reading manifest file 'whosaysthat.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'whosaysthat.egg-info/SOURCES.txt'
warning: sdist: standard file not found: should have one of README, README.rst,␣
↪README.txt

running check
creating whosaysthat-0.0.1
creating whosaysthat-0.0.1/configs
creating whosaysthat-0.0.1/data
creating whosaysthat-0.0.1/mike
creating whosaysthat-0.0.1/whosaysthat
creating whosaysthat-0.0.1/whosaysthat.egg-info
copying files to whosaysthat-0.0.1...
copying LICENSE -> whosaysthat-0.0.1
copying MANIFEST.in -> whosaysthat-0.0.1
copying README.md -> whosaysthat-0.0.1
copying setup.py -> whosaysthat-0.0.1
copying whatismyip -> whosaysthat-0.0.1
copying configs/whosaysthat.json -> whosaysthat-0.0.1/configs
copying data/1.txt -> whosaysthat-0.0.1/data
copying data/2.txt -> whosaysthat-0.0.1/data
copying mike/__init__.py -> whosaysthat-0.0.1/mike
copying mike/__main__.py -> whosaysthat-0.0.1/mike
copying whosaysthat/__init__.py -> whosaysthat-0.0.1/whosaysthat
copying whosaysthat/__main__.py -> whosaysthat-0.0.1/whosaysthat
copying whosaysthat.egg-info/PKG-INFO -> whosaysthat-0.0.1/whosaysthat.egg-info
copying whosaysthat.egg-info/SOURCES.txt -> whosaysthat-0.0.1/whosaysthat.egg-info
copying whosaysthat.egg-info/dependency_links.txt -> whosaysthat-0.0.1/whosaysthat.
↪egg-info
copying whosaysthat.egg-info/entry_points.txt -> whosaysthat-0.0.1/whosaysthat.egg-
↪info
copying whosaysthat.egg-info/requires.txt -> whosaysthat-0.0.1/whosaysthat.egg-info
copying whosaysthat.egg-info/top_level.txt -> whosaysthat-0.0.1/whosaysthat.egg-info
Writing whosaysthat-0.0.1/setup.cfg
creating dist
Creating tar archive
removing 'whosaysthat-0.0.1' (and everything under it)
$ ls dist/
whosaysthat-0.0.1.tar.gz
```

As you can see, the final output of the above command is a tarball inside of the `dist/` directory.

2.3 Working directory for the packaging

We will use the the `~/packaging/` as the working directory to build all the packages. Create this directory in your system.

Warning: Do not build any package as root user.

2.4 Install the build dependencies of the package itself

In this step we should install the build dependency of our package itself. As we use `requests` module in the example project, we will just install that from Debian repository.

```
$ sudo apt-get install python3-requests
```

2.5 Extracting our source tarball

```
$ cp dist/whosaysthat-0.0.1.tar.gz ~/packaging
$ cd ~/packaging
$ $ tar -xvf whosaysthat-0.0.1.tar.gz
whosaysthat-0.0.1/
whosaysthat-0.0.1/setup.py
whosaysthat-0.0.1/configs/
whosaysthat-0.0.1/configs/whosaysthat.json
whosaysthat-0.0.1/PKG-INFO
whosaysthat-0.0.1/mike/
whosaysthat-0.0.1/mike/__init__.py
whosaysthat-0.0.1/mike/__main__.py
whosaysthat-0.0.1/LICENSE
whosaysthat-0.0.1/whosaysthat.egg-info/
whosaysthat-0.0.1/whosaysthat.egg-info/PKG-INFO
whosaysthat-0.0.1/whosaysthat.egg-info/top_level.txt
whosaysthat-0.0.1/whosaysthat.egg-info/requires.txt
whosaysthat-0.0.1/whosaysthat.egg-info/entry_points.txt
whosaysthat-0.0.1/whosaysthat.egg-info/SOURCES.txt
whosaysthat-0.0.1/whosaysthat.egg-info/dependency_links.txt
whosaysthat-0.0.1/data/
whosaysthat-0.0.1/data/2.txt
whosaysthat-0.0.1/data/1.txt
whosaysthat-0.0.1/whosaysthat/
whosaysthat-0.0.1/whosaysthat/__init__.py
whosaysthat-0.0.1/whosaysthat/__main__.py
whosaysthat-0.0.1/README.md
whosaysthat-0.0.1/MANIFEST.in
whosaysthat-0.0.1/setup.cfg
whosaysthat-0.0.1/whatismyip
$ cd whosaysthat-0.0.1/
```

In the above commands, we extracted the tarball and cd into the source directory.

2.6 Add the packager details in your ~/.bashrc

Edit and add the following lines to reflect the right name and email address and add it to your `~/.bashrc` file. Remember to source the file.

```
DEBEMAIL="kushal@freedom.press"
DEBFULLNAME="Kushal Das"
export DEBEMAIL DEBFULLNAME
```

2.7 Create the initial packaging file

```
$ dh_make -f ../whosaysthat-0.0.1.tar.gz
Type of package: (single, indep, library, python)
[s/i/l/p]?
Email-Address       : kushal@freedom.press
License             : blank
Package Name        : whosaysthat
Maintainer Name     : Kushal Das
Version             : 0.0.1
Package Type        : python
Date                : Mon, 17 Sep 2018 19:51:02 -0400
Are the details correct? [Y/n/q]
Please respond with "yes" or "no" (or "y" or "n")
pth
Done. Please edit the files in the debian/ subdirectory now.
```

Note: remember that you will have to do this only for building the package for the first time.

After this we will have a new `debian` directory inside of the current directory. This directory has a lot of new files required for the packaging work.

```
$ tree debian/
debian/
├── changelog
├── compat
├── control
├── copyright
├── manpage.1.ex
├── manpage.sgml.ex
├── manpage.xml.ex
├── menu.ex
├── postinst.ex
├── postrm.ex
├── preinst.ex
├── prerm.ex
├── README.Debian
├── README.source
├── rules
├── source
│   ├── format
│   └── options
├── watch.ex
├── whosaysthat.cron.d.ex
├── whosaysthat.default.ex
├── whosaysthat.doc-base.EX
└── whosaysthat-docs.docs

1 directory, 22 files
```

2.8 Editing the control file

Our first step is to edit the `control` file and update it with the required information.

```
Source: whosaysthat
Section: unknown
Priority: optional
Maintainer: Kushal Das <kushal@freedom.press>
Build-Depends: debhelper (>= 9), dh-python, python3-all, python3-setuptools
Standards-Version: 3.9.8
Homepage: https://github.com/freedomofpress/yourpackage
X-Python-Version: >= 2.6
X-Python3-Version: >= 3.5

Package: whosaysthat
Architecture: all
Depends: ${python3:Depends}, python3-requests, ${misc:Depends}
Description: This is our example tool
This package installs the library for Python 3.
```

Import points to remember for this file.

- Double check the Build-Depends lines
- Add all the Debian packages this package is depending on the `Depends` line
- Please make sure to add all the native libraries this package is dependent on

are in the `Depends` line.

2.9 Editing the copyright file

The `debian/copyright` is an important file which tracks the copyright details of the different files inside of the package.

2.10 Editing the changelog

This is a *must have* file for the package. Below is an example. #1234 is the release ticket in our project's github.

```
whosaysthat (0.0.1-1) unstable; urgency=medium

* Initial release (Closes: #1234)

-- Kushal Das <kushal@freedom.press>  Mon, 17 Sep 2018 19:51:02 -0400
```

Note: Please update this file with new entries everytime you rebuild the package with any kind of change.

2.11 The rules file

This is primary file which decides how the package will be built. We can just simply use the standard commands provided by our *dh* tools. For more details, please have a look at the documentation.

The following should be a good start for a *setup.py* based project.

```
#!/usr/bin/make -f
# See debhelper(7) (uncomment to enable)
# output every command that modifies files on the build system.
#export DH_VERBOSE = 1

export PYBUILD_NAME=whosaysthat

%:
        dh $@ --with python2,python3 --buildsystem=pybuild
```

2.12 Copying extra files to different directories

We should a new `debian/packagename.install` file for the same. For our example package, we will only install the data files under `/usr/share/whosaysthat` directory.

```
data/1.txt usr/share/whosaysthat/data/1.txt
data/2.txt usr/share/whosaysthat/data/2.txt
```

2.13 Building the package

```
$ dpkg-buildpackage -us -uc
```

This command will build the package in `~/packaging` directory.

CHAPTER 3

Making your project ready for using virtualenv in Debian package

For this part of the tutorial, we will add a new dependency `cryptography` to our example project, and we will also import the library inside of our source code.

3.1 Install computepipfilehash tool

```
$ pip3 install computepipfilehash --user -U
```

The above command will install computepipfilehash tool. Use `0.0.3` version for this guide.

3.2 Change the example code

First, we will move into the source directory and checkout a different branch. Ensure that `Pipfile.lock` exists in the current working directory and then:

```
cd ~/code/whosaysthat/
git checkout fancyrelease
computepipfilehash > requirements-build.txt
```

The final command above creates a `requirements-build.txt` file for the dependencies. We will use this file to build the wheels locally. Next, we should move into our development environment and create an empty directory.

Note: We will sync source tarballs and binary wheels to the localwheels directory here.

Next, we will download missing source tarballs from PyPI.

```
mkdir localwheels
pip3 download --no-binary :all: -d ./localwheels/ -r requirements-build.txt
```

Then, update the `requirements-build.txt` file with the hashes from the existing wheels.

```
computepipfilehash --update-hashes
```

Finally, we can build the missing binary wheels from the sources.

```
pip3 wheel --no-index --find-links ./localwheels/ -w ./localwheels/ -r requirements-
↪build.txt
```

Note: The *python-dateutil* package is an exception as it we have to build the wheel manually first. This is because the way the *setup.py* of the said project works.

`pip3 install wheel` and then `pip3 wheel localwheels/python-dateutil-2.7.5.tar.gz`

The above command will build the wheels in the `./localwheels/` directory. But, this will fail as some development header files are missing. We should install all external C level dependencies from the Debian repository itself. After installing the packages, we should retry to build the wheels again.

Note: Remember to commit the `requirements-build.txt` file to the git repo. This will help others to build using the same source tarballs.

```
sudo apt-get install libssl-dev libffi-dev
pip3 wheel --no-index --find-links ./localwheels/ -w ./localwheels/ -r requirements-
↪build.txt
ls ./localwheels/
asn1crypto-0.24.0-py3-none-any.whl
certifi-2018.8.24-py2.py3-none-any.whl
cffi-1.11.5-cp35-cp35m-linux_x86_64.whl
chardet-3.0.4-py2.py3-none-any.whl
cryptography-2.3.1-cp35-cp35m-linux_x86_64.whl
idna-2.7-py2.py3-none-any.whl
pycparser-2.19-py2.py3-none-any.whl
requests-2.19.1-py2.py3-none-any.whl
six-1.11.0-py2.py3-none-any.whl
urllib3-1.23-py2.py3-none-any.whl
```

3.3 Create the requirements.txt file for our wheels

As the next step, we will create the final `requirements.txt` file which will contain the details of the wheels including the hashes.

```
computepipfilehash --wheel-hashes > requirements.txt
```

3.4 Sync the local wheels into a central storage

Note: Here we will have to figure out the steps to move the wheels to a central location.

3.5 Work on Debian packaging

Then, we will create a new source tarball for our project and also copy the wheels.

```
python3 setup.py sdist
cp dist/whosaysthat-0.0.2.tar.gz ~/packaging/
cd ~/packaging/
tar -xvf whosaysthat-0.0.2.tar.gz
cd whosaysthat-0.0.2/
cp -r ~/code/whosaysthat/localwheels .
```

Now, we will create the files required for our packaging manually, including the `debian` directory. We will also install `dh-virtualenv` package.

```
$ mkdir debian
$ sudo apt-get install dh-virtualenv
```

3.6 Create the compatibility file

```
$ echo "9" > debian/compat
```

3.7 Create the control file

Add the following text to the `debian/control` file.

```
Source: whosaysthat
Section: unknown
Priority: optional
Maintainer: Kushal Das <kushal@freedom.press>
Build-Depends: debhelper (>= 9), dh-python, python3-all, python3-setuptools, dh-
↪virtualenv
Standards-Version: 3.9.8
Homepage: https://github.com/freedomofpress/yourpackage
X-Python3-Version: >= 3.5

Package: whosaysthat
Architecture: all
Depends: ${python3:Depends}, ${misc:Depends}
Description: This is our example tool
 This package installs the library for Python 3.
```

If we know any library we are dependent on (written in C), we should explicitly mention that in the `Depends:` line above.

3.8 Create the triggers file

To keep our virtualenv in sync with the host Python, let us create a `debian/whosaysthat.triggers` file. The standard name for this is `debian/packagename.triggers`.

```
# Register interest in Python interpreter changes (Python 2 for now); and
# don't make the Python package dependent on the virtualenv package
# processing (noawait)
interest-noawait /usr/bin/python3.5

# Also provide a symbolic trigger for all dh-virtualenv packages
interest dh-virtualenv-interpreter-update
```

3.9 Update the changelog file

First, we will copy the existing changelog file. Then, we will use `dch` tool to update the entry there.

```
$ cp ../whosaysthat-0.0.1/debian/changelog debian/
$ dch
```

This will open up your favorite editor, update and save the file.

Note: You will have to install *devscripts* package in Debian for the *dch* command.

3.10 Create the install file

This is same as in the last time. Add the following in the `debian/whosaysthat.install` file.

```
data/1.txt usr/share/whosaysthat/data/1.txt
data/2.txt usr/share/whosaysthat/data/2.txt
```

3.11 Create a links file

dh-virtualenv tool will create a virtualenv under `/opt/venvs`, in our example, this will be `/opt/venvs/whosaysthat` directory, and the console entry point based executables will be installed in the bin directory there. So, we should create links to those commands from `/usr/bin`.

Add the following in the `debian/whosaysthat.links` file.

```
opt/venvs/whosaysthat/bin/whatismyip usr/bin/whatismyip
opt/venvs/whosaysthat/bin/whoisthebest usr/bin/whoisthebest
```

3.12 Export environment variables to use the local wheels

```
$ export DH_PIP_EXTRA_ARGS="--require-hashes --no-index --find-links=./localwheels"
```

This will make *dh-virtualenv* to use our wheels instead of downloading them from PyPI.

3.13 The final rules file

Add the following text to the `debian/rules` file.

```
#!/usr/bin/make -f

%:
        dh $@ --with python-virtualenv --python /usr/bin/python3.5 --setuptools
```

Note: If you copy paste the above example, then remember to use a TAB instead of 8 spaces :)

Remember, for a package with dependent system *site-packages*, means packages which depends on Python modules from Debian world, the above will need modification.

```
#!/usr/bin/make -f

%:
    dh $@ --with python-virtualenv

override_dh_virtualenv:
    dh_virtualenv --python /usr/bin/python3.5 --setuptools -S
```

3.14 Let us build the package

```
$ dpkg-buildpackage -us -uc
```

This should create the Debian package in the parent directory.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search