dh-virtualenv Documentation

Release 0.8

Spotify AB

March 07, 2016

1 What is dh-virtualenv 3

2 Changelog 5
2.1 0.8 5
2.2 0.7 5
2.3 0.6 5

3 Tutorial 7
3.1 Step 1: Install dh-virtualenv 7
3.2 Step 2: Setup the Debian packaging 7
3.3 Step 3: Build your project 9

4 Building packages with dh-virtualenv 11
4.1 Simple usecase 11
4.2 Command line options 11
4.3 Advanced usage 12

5 Indices and tables 13

Contents:

CHAPTER 1

What is dh-virtualenv

`dh-virtualenv` is a tool that aims to combine Debian packaging with self-contained virtualenv based Python deployments. To do this, the package extends debhelper's sequence by providing a new command in sequence, `dh_virtualenv`, which effectively replaces following commands from the sequence:

- `dh_auto_install`
- `dh_python2`
- `dh_pycentral`
- `dh_pysupport`

In the sequence the `dh_virtualenv` is inserted right after `dh_perl`.

CHAPTER 2

Changelog

Following list contains most notable changes by version. For full list consult the git history of the project.

2.1 0.8

- Support for running triggers upon host interpreter update. This new feature makes it possible to upgrade the host Python interpreter and avoid breakage of all the virtualenvs installed with virtualenv. For usage, see the the Tutorial. Huge thanks to Jürgen Hermann for implementing this long wanted feature!
- Add support for the built-in `venv` module. Thanks to Petri Lehtinen!
- Allow custom `pip` flags to be passed via the `--extra-pip-args` flag. Thanks to @labeneator for the feature.

2.2 0.7

- **Backwards incompatible** Support running tests. This change breaks builds that use distutils. For those cases a flag `--no-test` needs to be passed.
- Add tutorial to documentation
- Don't crash on debbuild parameters `-i` and `-a`
- Support custom source directory (debhelper's flag `-D`)

2.3 0.6

First public release of *dh-virtualenv*

CHAPTER 3

Tutorial

This tutorial will guide you through setting up your first project using *dh-virtualenv*. Having some knowledge on how Debian packages work won't hurt, but it is not necessary a mandatory requirement. You also need some basic build tools, so it is recommended to install *build-essential* and *devscripts* packages.

3.1 Step 1: Install dh-virtualenv

In order to use it, you need to install the *dh-virtualenv*. If you run Debian Jessie (testing), Debian Sid (unstable) or Ubuntu 14.04 LTS (Trusty), you can install *dh-virtualenv* simply with *apt-get*:

```
sudo apt-get install dh-virtualenv
```

For other systems the only way is to build and install it yourself. Steps to do that, after you have cloned the repository are:

```
sudo apt-get install devscripts python-virtualenv git equivs # Install needed packages
git clone https://github.com/spotify/dh-virtualenv.git       # Clone Git repository
cd dh-virtualenv                                             # Move into the repository
sudo mk-build-deps -ri                                       # This will install build dependencies
dpkg-buildpackage -us -uc -b                                 # Build the *dh-virtualenv* package

# and finally, install it (you might have to solve some
# dependencies when doing this):
sudo dpkg -i ../dh-virtualenv_<version>.deb
```

3.2 Step 2: Setup the Debian packaging

Grab your favourite Python project you want to use *dh-virtualenv* with and set it up. Only requirement is that your project has a somewhat sane `setup.py` and requirements listed in a `requirements.txt` file. Note however that the defining requirements is not mandatory.

Next you need to define the Debian packaging for your software. To do this, create a directory called `debian` in the project root.

To be able to build a debian package, a few files are needed. First, we need to define the compatibility level of the project. For this, do:

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

The 9 is a magic number for latest compatibility level, but we don't need to worry about that. Next we need a file that tells what our project is about, a file called `control`. Enter a following `debian/control` file:

```
Source: my-awesome-python-software
Section: python
Priority: extra
Maintainer: Matt Maintainer <matt@example.com>
Build-Depends: debhelper (>= 9), python, dh-virtualenv (>= 0.8)
Standards-Version: 3.9.5

Package: my-awesome-python-software
Architecture: any
Pre-Depends: dpkg (>= 1.16.1), python2.7-minimal | python2.6-minimal, ${misc:Pre-Depends}
Depends: ${python:Depends}, ${misc:Depends}
Description: really neat package!
 second line can contain extra information about it.
```

The `control` file is used to define the build dependencies, so if you are building a package that requires for example `lxml`, make sure you define `libxml2-dev` in *Build-Depends* etc.

Depends in the lower section is used to define run-time dependencies. Following the example above, in case of lxml you would add `libxml2` in to the *Depends* field.

To help keeping your installed virtualenv in sync with the host's Python interpreter in case of updates, create a file named `debian/«pkgname».triggers`, where `«pkgname»` is what you named your package in the `control` file. It triggers a special script whenever the Python binary changes; don't worry, that script is provided by `dh-virtualenv` automatically.

```
# 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/python2.6
interest-noawait /usr/bin/python2.7

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

Note that if you provide a custom `postinst` script with your package, then don't forget to put the `#DEBHELPER#` marker into it, else the trigger script will be missing.

Next, we need a changelog file. It is basically a documentation of changes in your package plus the source for version number for Debian package builder. Here's a short sample changelog to be entered in `debian/changelog`:

```
my-awesome-python-software (0.1-1) unstable; urgency=low

  * Initial public release

 -- Matt Maintainer <matt@example.com>  Fri, 01 Nov 2013 17:00:00 +0200
```

You don't need to create this file by hand, a handy tool called `dch` exists for entering new changelog entries.

Now, last bit is left, which is the `debian/rules` file. This file is basically a Makefile that Debian uses to build the package. Content for that is fairly straightforward:

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

%:
	dh $@ --with python-virtualenv
```

And there we go, debianization of your new package is ready!

3.3 Step 3: Build your project

Now you can just build your project by running `dpkg-buildpackage -us -uc`. Enjoy your newly baked *dh-virtualenv* backed project! :)

CHAPTER 4

Building packages with dh-virtualenv

Building packages with *dh-virtualenv* is relatively easy to start with but it also supports lot of customization to fit in your general needs.

By default, *dh-virtualenv* installs your packages under `/usr/share/python/<packagename>`. The package name is provided by the `debian/control` file. Exporting the variable `DH_VIRTUALENV_INSTALL_ROOT=</your/custom/install/dir>` will set the install path for your package if you don't wish to use the default path.

4.1 Simple usecase

To signal debhelper to use *dh-virtualenv* for building your package, you need to pass `--with python-virtualenv` to debhelper sequencer.

In a nutshell, the simplest `debian/rules` file to build using *dh-virtualenv* looks like this:

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

%:
	dh $@ --with python-virtualenv
```

However, the tool makes a few assumptions of your project's structure:

- For installing requirements, you need to have a file called `requirements.txt` in the root directory of your project. The requirements file is not mandatory.
- The project must have a `setup.py` file in the root of the project. Sequencer will run `setup.py install` to install the package inside the virtualenv.

After these are place, you can just build the package with your favorite tool!

4.2 Command line options

To change the default behavior the `dh_virtualenv` command accepts a few command line options:

-p `<package>`, **--package** `<package>`
    Act on the package named *<package>*

-N `<package>`, **--no-package** `<package>`
    Do not act on the specified package

-v, --verbose
Turn on verbose mode. This has a few effects: it sets root logger level to `DEBUG` and passes verbose flag to `pip` when installing packages. This can also be provided using the standard `DH_VERBOSE` environment variable.

--extra-index-url `<url>`
Use extra index url *<url>* when running `pip` to install packages. This can be provided multiple times to pass multiple URLs to `pip`. This is useful if you for example have a private Python Package Index.

--preinstall `<package>`
Package to install before processing the requirements. This flag can be used to provide a package that is installed by `pip` before processing requirements file. This is handy if you need to install for example a custom setup script or other packages needed to parse `setup.py`. This flag can be provided multiple times to pass multiple packages for pre-install.

--pypi-url `<URL>`
Base URL of the PyPI server. This flag can be used to pass in a custom URL to a PyPI mirror. It's useful if you for example have an internal mirror of the PyPI or you run a special instance that only exposes selected packages of PyPI. If this is not provided, the default will be whatever `pip` uses as default (usually `http://pypi.python.org/simple`).

--extra-pip-arg `<PIP ARG>`
Extra parameters to pass to the pip executable. This is useful if you need to change the behaviour of pip during the packaging process. You can use this flag multiple times to pass in different pip flags. As an example passing in –extra-pip-arg "–no-compile" to the override_dh_virtualenv section of the debian/rules file will disable the generation of pyc files.

--setuptools
Use setuptools instead of distribute in the virtualenv

--no-test
Skip running `python setup.py test` after dependencies and the package is installed. This is useful if the Python code is packaged using distutils and not setuptools.

--python `<path>`
Use a specific Python interpreter found in `path` as the interpreter for the virtualenv. Default is to use the system default, usually `/usr/bin/python`.

--builtin-venv
Enable the use of the build-in `venv` module, i.e. use `python -m venv` to create the virtualenv. For this to work, requires Python 3.4 or later to be used, e.g. by using the option `--python /usr/bin/python3.4`. (Python 3.3 has the `venv` module, but virtualenvs created with Python 3.3 are not bootstrapped with setuptools or pip.)

4.3 Advanced usage

To provide command line options to `dh_virtualenv` sequence the override mechanism of the debhelper is the best tool.

Following `debian/rules` will provide *http://example.com* as additional Python Package Index URI:

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

%:
	dh $@ --with python-virtualenv

override_dh_virtualenv:
	dh_virtualenv --extra-index-url http://example.com
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Symbols

–builtin-venv
    command line option, 12
–extra-index-url <url>
    command line option, 12
–extra-pip-arg <PIP ARG>
    command line option, 12
–no-test
    command line option, 12
–preinstall <package>
    command line option, 12
–pypi-url <URL>
    command line option, 12
–python <path>
    command line option, 12
–setuptools
    command line option, 12
-N <package>, –no-package <package>
    command line option, 11
-p <package>, –package <package>
    command line option, 11
-v, –verbose
    command line option, 11

C

command line option
    –builtin-venv, 12
    –extra-index-url <url>, 12
    –extra-pip-arg <PIP ARG>, 12
    –no-test, 12
    –preinstall <package>, 12
    –pypi-url <URL>, 12
    –python <path>, 12
    –setuptools, 12
    -N <package>, –no-package <package>, 11
    -p <package>, –package <package>, 11
    -v, –verbose, 11