venvs

Release 2024.5.2.dev2+g36ebc8e

Julian Berman

Apr 22, 2024

CONTENTS

1 Installation 3

2 Usage 5

3 Single-Purpose Virtualenvs 7

4 Temporary Virtualenvs 9

5 The 5 Minute Tutorial 11

6 Why don't I use `virtualenvwrapper`? 13

7 Contents 15

venvs is a tool for configuring, in a single file, a set of virtualenvs, which packages to install into each, and any binaries to make globally available from within.

CHAPTER

ONE

INSTALLATION

The usual:

```
$ pip install venvs
```

CHAPTER

TWO

USAGE

The best way to use `venvs` is by creating a file named `~/.local/share/virtualenvs/virtualenvs.toml`. Here's an example of what goes in it:

```
[virtualenv.development]
install = [
    "pudb",
    "twisted",
]
link = ["trial"]

[virtualenv.app]
install = ["$DEVELOPMENT/myapp"]
```

After creating the above, running `venvs converge` will create 2 virtualenvs, one called "development" with `pudb` and twisted installed into it and trial linked from within it onto your `PATH`, and a second called "app" installing the corresponding directory.

For a more intricate example, have a look at my own virtualenvs.toml.

That's about all you need to know. If you insist on reading further though, venvs has an older, not-very-recommended mutable interface which allows you to create virtualenvs in a central location without tracking them in a config file (or converging them). For that, usage is similar to `mkvirtualenv`, although `venvs` passes arguments directly through to `virtualenv`:

```
$ venvs create nameofvenv -- -p pypy
```

will create a virtual environment in an appropriate platform-specific data directory, or in the directory specified by `WORKON_HOME` for compatibility with `virtualenvwrapper`.

CHAPTER

THREE

SINGLE-PURPOSE VIRTUALENVS

A common use case for virtualenvs is for single-purpose installations, e.g.:

“I want to install fabric and give it its own virtualenv so that its dependencies can be independently upgraded, all while still being able to use the `fab` binary globally”.

`venvs` supports a `--link` option for this use case:

```
$ venvs create -i fabric --link fab
```

will create a virtualenv for fabric (in the same normal location), but will symlink the `fab` binary from within the virtualenv into your `~/.local/bin` directory.

(You may have heard of pipsi which is a similar tool for this use case, but with less customization than I would have liked.)

CHAPTER

FOUR

TEMPORARY VIRTUALENVS

I also find `mktmpenv` useful for quick testing. To support its use case, `venvs` currently supports a different but similar style of temporary virtualenv.

Invoking:

```
$ venv=$(venvs temporary)
```

in your shell will create (or re-create) a global temporary virtualenv, and print its `bin/` subdirectory (which in this case will be then stored in the `venv` variable). It can subsequently be used by, e.g.:

```
$ $venv/python
```

or:

```
$ $venv/pip ...
```

et cetera.

You may prefer using:

```
$ cd $(venvs temporary)
```

as your temporary venv workflow if you're into that sort of thing instead.

The global virtualenv is cleared each time you invoke `venvs temporary`. Unless you care, unlike `virtualenvwrapper`'s `mktmpenv`, there's no need to care about cleaning it up, whenever it matters for the next time, it will be cleared and overwritten.

`venvs` may support the more similar "traditional" one-use virtualenv in the future, but given that it does not activate virtualenvs by default (see below), the current recommendation for this use case would be to simply use the `virtualenv` binary directly.

CHAPTER

FIVE

THE 5 MINUTE TUTORIAL

Besides the `venvs` for named-virtualenv creation and `venvs temporary` for temporary-virtualenv creation described above:

```
$ venvs find name foo
```

will output (to standard output) the path to a virtualenv with the given name (see also `--existing-only`), and:

```
$ venvs remove foo
```

will remove it.

There are a number of other slight variants, see the `--help` information for each of the three binaries.

Real documentation to come (I hope)

CHAPTER

SIX

WHY DON'T I USE `VIRTUALENVWRAPPER`?

`virtualenvwrapper` is great! I've used it for a few years. But I've slowly settled on a much smaller subset of its functionality that I like to use. Specifically:

- I don't like activating virtualenvs.

  virtualenvs are magical and hacky enough on their own, and piling activation on top just makes things even more messy for me, especially when moving around between different projects in a shell. Some people use `cd` tricks to solve this, but I just want simplicity.

- I don't need project support.

  I've never attached a project to a virtualenv. I just use a naming convention, naming the virtualenv with the name of the repo (with simple coercion), and then using dynamic directory expansion in my shell to handle association.

Basically, I just want a thing that is managing a central repository of virtualenvs for me. So that's what `venvs` does.

CHAPTER SEVEN

CONTENTS