Using Pipenv

Virtual Environments

When you start a new project, you'll want it to be completely separate from other projects in terms of the libraries it has installed.

If you start working today on a project using version `x` of a library, and in two years time you work on another project using version `y` of a library, you may not want to update. If version `y` includes some breaking changes from version `x`, your old project would stop working.

This is why it's interesting have separate environments for each project, so that the changes and updates to libraries one depends on does no affect other projects.

Installing new libraries

Python installs new packages using `pip`. You can install packages in your system Python installation easily using `pip`:

```
pip install requests
```

This will install the `requests` library in the Python that is linked to `pip`. By default on Mac OS X, this is a Python version that comes installed in the computer, Python 2.7.

If you've installed another Python version, it'll be that version.

If you have multiple Python versions installed, you'll have to be more specific:

- `python` will refer to the first Python version you installed.
- `python3` will refer to the version of Python 3.x that you have installed, **if you have only one version of Python 3 and one of Python 2**.
- `python3.7` will refer to Python3.7 if you have two or more versions of Python 3 installed.

Virtual environments

In order to separate projects into different environments with different libraries, we use an extremely popular library called `virtualenv`.

You can create a new virtual environment—which is just a copy of the Python version you use to create it—using one of the following commands (depending on the Python program as above):

```
virtualenv venv
virtualenv venv --python=python3
virtualenv venv --python=python3.7
```

This creates a new folder called `venv` inside your current folder, which is a copy of Python without any libraries installed.

Now we can activate this virtual environment, which then means any libraries we install using `pip` will go in the virtual environment; and when we run the `python` program that'll be the version installed in the virtual environment:

```
source venv/bin/activate
```

Or on Windows:

```
.\venv\Scripts\activate.bat
```

Pipenv

Pipenv brings together `pip`, `virtualenv`, and a dependency management system.

Instead of creating a virtualenv, activating it, and installing libraries, we can just install a library using Pipenv. Make sure you don't have a virtual environment created locally by deleting it if you have created it.

```
rm -rf venv
```

Then, let's install Pipenv. You'll have to use:

- `pip` if you only have a single version of Python installed;
- `pip3` if you have a version of Python3 and one version of Python2 installed; or
- `pip3.7` if you have two or more versions of Python3 (and one of them is Python3.7).

```
pip install pipenv
```

Then, we can install the first library using Pipenv:

```
pipenv install requests
```

This creates a new virtual environment (in a different folder in your computer) and installs the library in it.

In addition, it also creates two new files in your project folder:

- `Pipfile`; and
- `Pipfile.lock`.

The Pipfile file

After installing the `requests` library as above, the `Pipenv` file will look like this:

```
[[source]]

url = “https://pypi.python.org/simple”
verify_ssl = true
name = “pypi”


[packages]

requests = “*”

[dev-packages]
```

This file is divided into sections: `[[source]]`, `[packages]`, and `[dev-packages]`.

The most interest one here is the `[packages]` section, which has all the dependencies of our project—everything our project depends on. These libraries here are installed from the `url` defined in the `[[source]]` section.

Here we specify the `requests` library, and the version we want is `*`; any version is acceptable. By default, we will always install the latest version of any package.

However, when you share the project with someone else, you'll share both the `Pipfile` (which tells them what to install), and the `Pipfile.lock` (which tells them which versions of things to install.

Let's talk about `Pipfile.lock`!

The Pipfile.lock file

This file looks more complicated. Something like this:

```
{
    “_meta”: {
        “hash”: {
            “sha256”:
“a0e63f8a0d1e3df046dc19b3ffbaaedfa151afc12af5a5b960ae7393952f8679”
        },
        “host-environment-markers”: {
            “implementation_name”: “cpython”,
            “implementation_version”: “3.7.0b1”,
            “os_name”: “posix”,
            “platform_machine”: “x86_64”,
            “platform_python_implementation”: “CPython”,
            “platform_release”: “17.4.0”,
            “platform_system”: “Darwin”,
            “platform_version”: “Darwin Kernel Version 17.4.0: Sun Dec 17 09:19:54
PST 2017; root:xnu-4570.41.2~1/RELEASE_X86_64”,
            “python_full_version”: “3.7.0b1”,
            “python_version”: “3.7”,
            “sys_platform”: “darwin”
        },
        “pipfile-spec”: 6,
        “requires”: {},
        “sources”: [
            {
                “name”: “pypi”,
                “url”: “https://pypi.python.org/simple”,
                “verify_ssl”: true
            }
        ]
    },
    “default”: {
        “certifi”: {
            “hashes”: [
“sha256:14131608ad2fd56836d33a71ee60fa1c82bc9d2c8d98b7bdbc631fe1b3cd1296”,
“sha256:edbc3f203427eef571f79a7692bb160a2b0f7ccaa31953e99bd17e307cf63f7d”
            ],
            “version”: “==2018.1.18”
        },
        “chardet”: {
            “hashes”: [
“sha256:fc323ffcaeaed0e0a02bf4d117757b98aed530d9ed4531e3e15460124c106691”,
“sha256:84ab92ed1c4d4f16916e05906b6b75a6c0fb5db821cc65e70cbd64a3e2a5eaae”
            ],
            “version”: “==3.0.4”
        },
        “idna”: {
            “hashes”: [
“sha256:8c7309c718f94b3a625cb648ace320157ad16ff131ae0af362c9f21b80ef6ec4”,
“sha256:2c6a5de3089009e3da7c5dde64a141dbc8551d5b7f6cf4ed7c2568d0cc520a8f”
            ],
            “version”: “==2.6”
        },
        “requests”: {
            “hashes”: [
“sha256:6a1b267aa90cac58ac3a765d067950e7dbbf75b1da07e895d1f594193a40a38b”,
“sha256:9c443e7324ba5b85070c4a818ade28bfabedf16ea10206da1132edaa6dda237e”
            ],
            “version”: “==2.18.4”
        },
        “urllib3”: {
            “hashes”: [
“sha256:06330f386d6e4b195fbfc736b297f58c5a892e4440e54d294d7004e3a9bbea1b”,
"sha256:cc44da8e1145637334317feebd728bd869a35285b93cbb4cca2577da7e62db4f"
            ],
            "version": "==1.22"
        }
    },
    "develop": {}
}
```

This defines a lot more information, and this file is used to "lock" the versions of the libraries you have installed to a specific point in time.

If you give the `Pipfile` and the `Pipfile.lock` to someone else, they'll end up with exactly the same versions of libraries as you.

Each library in the `Pipfile.lock` comes with two things:

- `version`; and
- `hashes`.

The `version` is the version of the package you want. Normally this is the only thing you need in order to retrieve the correct package from the source `url` (defined in the `Pipfile`).

However, it is possible that someone might swap out a version of a library for something else—a security issue to say the least!

That is where the hashes come into play. The hashes are numbers generated from the library itself. If the library changes at all, the numbers generated will change. So if you try to install the same version as earlier on but the hashes are different, you'll get an error. Pipenv will make sure it's all safe!

Security is a difficult topic, but the entire Python community is agreeing that having these lock files is a great idea. Other programming communities, like JavaScript, is also using similar techniques for securing their dependency management. It's the way to go!

— Jose and the Teclado team.