venv-update Documentation

Release 3.1.1

Buck Evan

Oct 19, 2018

Contents

Issues | Github | CI | PyPI

Release v3.1.1 (*Installation*)

CHAPTER 1

`venv-update` in detail

venv-update is a small script whose job is to idempotently ensure the existence and correctness of a project's virtualenv, and synchronize it with its requirements.

We like to call venv-update from our Makefiles to create and maintain a virtualenv. It does the following:

- Ensures a virtualenv exists at the specified location with the specified Python version, and that it is valid. It will create or recreate a virtualenv as necessary to ensure that one venv-update invocation is all that's needed.
- Calculates the difference in packages derived from the `requirements.txt` files and the installed packages. Packages will be uninstalled, upgraded, or installed as necessary.

  The goal is that venv-update will put you in the same state as if you wipe away your virtualenv and rebuild it with `pip install`, but much more quickly.
- Takes advantage of `pip-faster` for package installation (see below) to avoid network access and rebuilding packages as much as possible.

For reference, a project with 250 dependencies which are all pinned can run a no-op venv-update in ~2 seconds with no network access. The running time when changes are needed is dominated by the time it takes to download and install packages, but is generally quite fast (on the order of ~10 seconds).

1.1 Customizing the install command

If you don't like pip-faster, for whatever reason, `venv-update` provides sufficient control to use "plain-old" pip, or any other command for that matter.

To tell venv-update to install and run pip rather than pip-faster:

```
venv-update install-command= pip install --upgrade bootstrap-deps= 'pip>8'
```

CHAPTER 2

`pip-faster` in detail

By design `pip-faster` maintains the interface of pip, and should only have a few desirable behavior differences, listed below.

1. `pip-faster` adds optional "prune" capability to the `pip install` subcommand. `pip-faster install --prune` will *uninstall* any installed packages that are not required (as arguments to the same install command). This is used by default in *venv-update* to implement reproducible builds.
2. We've taken great care to reduce the number of round-trips to PyPI, which makes up the majority of time spent on what should be a no-op update. For example, if you're installing a specific version of a package which we already have cached, there's no need to talk to PyPI, but vanilla pip will.
3. Packages are downloaded and wheeled before installation (if they aren't available from PyPI as wheels). If the virtualenv needs to be rebuilt, or you use the same requirement in another project, the wheel can be reused. This greatly speeds up installation of projects like lxml or numpy which have a slow-to-compile binary component.

   Mainline pip recently added this feature (in pip 7.0, 2015-05-21). We plan to merge, but this isn't currently an urgent work item; all of our use cases are satisfied. However, patches *are* welcome.
4. `pip-faster` will refuse to install package versions which conflict (we generally consider this a feature); stock pip, on the other hand, will happily install conflicting packages. Similarly, pip-faster detects circular dependencies and unsatisfied dependencies and throws an error where stock pip would not.

2.1 Installation

You can `pip install venv-update` to get `pip-faster`, the same way you would any other Python tool, but if you're using *venv-update* it's not necessary to install pip-faster; the venv-update script will install the correct version inside your virtualenv for you.

2.1.1 Internal PyPI Servers

Under linux, performance will be much better if you use an internal PyPI server instead of the public PyPI.

Besides the potentially lesser latency, an internal PyPI server allows for uploading binary wheels compiled for Linux. Unlike OS X or Windows, installing projects like lxml on Linux is normally extremely slow since they will need to be compiled during every installation.

pip-faster improves this by only compiling on the first installation for each user (this is also the default behavior for pip >= 7), but this doesn't help for the first run.

Using an internal PyPI server which allows uploading of Linux wheels can improve speed greatly. Unfortunately, these wheels are guaranteed compatible only with the same Linux distribution they were compiled on, so this only works if your developers work in very homogeneous environments.

For both venv-update and pip-faster, you can specify an index server by setting the `$PIP_INDEX_URL` environment variable (or `$PIP_EXTRA_INDEX_URL` if you want to supplement but not replace the default PyPI). For pip-faster you can also use `-i` or `-e`, just like in regular pip.

2.1.2 Benchmarks

You can find the set of scripts used to derive these numbers at: https://github.com/Yelp/venv-update/tree/master/benchmark

```
benchmark: installing plone and its dependencies (260 packages)
last run: 2016-02-24

    pip 8.0.2:
        cold:
            4m37.612s
            4m39.762s
            4m39.717s
        noop:
            0m6.890s
            0m7.112s
            0m7.436s
        warm:
            0m44.684s
            0m44.614s
            0m43.272s


    pip-faster:
        cold:
            4m16.163s
            4m21.282s
            4m14.038s
        noop:
            0m2.399s
            0m2.389s
            0m2.335s
        warm:
            0m30.410s
            0m21.303s
            0m21.323s
```

CHAPTER 3

Introduction

venv-update is an MIT-Licensed tool to quickly and exactly synchronize a large python project's virtualenv with its requirements.

This project ships as two separable components: `pip-faster` and `venv-update`.

Both are designed for use on large projects with hundreds of requirements and are used daily by Yelp engineers.

CHAPTER 4

Why?

Generating a repeatable build of a virtualenv has many edge cases. If a requirement is removed, it should be uninstalled when the virtualenv is updated. If the version of python has changed, the only reliable solution is to re-build the virtualenv from scratch. Initially, this was exactly how we implemented updates of our virtualenv, but it slowed things down terribly. `venv-update` handles all of these edge cases and more, without completely starting from scratch (in the usual case).

In a large application, best practice is to "pin" versions, with requirements like `package-x==1.2.3` in order to ensure that dev, staging, test, and production will all use the same code. Currently `pip` will always reach out to PyPI to list the versions of `package-x` regardless of whether the package is already installed, or whether its wheel can be found in the local cache. `pip-faster` adds these optimizations and others.

CHAPTER 5

venv-update

A small script designed to keep a virtualenv in sync with a changing list of requirements. The contract of `venv-update` is this:

> The virtualenv state will be exactly the same as if you deleted and rebuilt it from scratch, but will get there in *much* less time.

The needs of venv-update are what drove the development of pip-faster. For more, see *venv-update in detail*.

CHAPTER 6

pip-faster

pip-faster is a drop-in replacement for pip. `pip-faster`'s contract is:

> Take the same arguments and give the same results as `pip`, just more quickly.

This is *especially* true in the case of pinned requirements (e.g. `package-x==1.2.3`). If you're also using venv-update (which we heartily recommend!), you can view pip-faster as an implementation detail. For more, see *pip-faster in detail*.

6.1 *How much* faster?

If we install plone (a large python application with more than 250 dependencies) we get these numbers:

testcase	pip v8.0.2	pip-faster	improvement
cold	4m 39s	4m 16s	8%
noop	7.11s	2.40s	**196%**
warm	44.6s	21.3s	**109%**

In the "cold" case, all caches are completely empty. In the "noop" case nothing needs to be done in order to update the virtualenv. In the "warm" case caches are fully populated, but the virtualenv has been completely deleted.

The *Benchmarks* page has more detail.

CHAPTER 7

Installation

Because `venv-update` is meant to be the entry-point for creating your virtualenv directory and installing your packages, it's not meant to be installed via pip; that would require a virtualenv to already exist!

Instead, the script is designed to be *vendored* (directly checked in) to your project. It has no dependencies other than virtualenv and the standard Python library.

```
curl -o venv-update https://raw.githubusercontent.com/Yelp/venv-update/v3.1.1/
venv_update.py
chmod +x venv-update
```

CHAPTER 8

Usage

By default, running `venv-update` will create a virtualenv named `venv` in the current directory, using `requirements.txt` in the current directory. This should be the desired default for most projects.

If you need more control, you can pass additional options to both `virtualenv` and `pip`. The command-line help gives more detail:

```
$ venv-update --help
usage: venv-update [-hV] [options]

Update a (possibly non-existent) virtualenv directory using a pip requirements
file.  When this script completes, the virtualenv directory should contain the
same packages as if it were deleted then rebuilt.

venv-update uses "trailing equal" options (e.g. venv=) to delimit groups of
(conventional, dashed) options to pass to wrapped commands (virtualenv and pip).

Options:
    venv=             parameters are passed to virtualenv
                       default: venv
    install=          options to pip-command
                       default: -r requirements.txt
    pip-command=      is run after the virtualenv directory is bootstrapped
                       default: pip-faster install --upgrade --prune
    bootstrap-deps=   dependencies to install before pip-command= is run
                       default: venv-update==3.1.1

Examples:
    # install requirements.txt to "venv"
    venv-update

    # install requirements.txt to "myenv"
    venv-update venv= myenv

    # install requirements.txt to "myenv" using Python 3.4
```

(continues on next page)

(continued from previous page)

```
    venv-update venv= -ppython3.4 myenv

    # install myreqs.txt to "venv"
    venv-update install= -r myreqs.txt

    # install requirements.txt to "venv", verbosely
    venv-update venv= venv -vvv install= -r requirements.txt -vvv

    # install requirements.txt to "venv", without pip-faster --update --prune
    venv-update pip-command= pip install

We strongly recommend that you keep the default value of pip-command= in order
to quickly and reproducibly install your requirements. You can override the
packages installed during bootstrapping, prior to pip-command=, by setting
bootstrap-deps=

Pip options are also controllable via environment variables.
See https://pip.readthedocs.org/en/stable/user_guide/#environment-variables
For example:
    PIP_INDEX_URL=https://pypi.example.com/simple venv-update

Please send issues to: https://github.com/yelp/venv-update
```

8.1 ... in your `Makefile`

venv-update is a good fit for use with make because it is idempotent and should never fail, under normal circumstances. Here's an example Makefile:

```
venv: requirements.txt
   ./bin/venv-update

.PHONY: run-some-script
run-some-script: venv
   ./venv/bin/some-script
```

8.2 ... with tox

tox is a useful tool for testing libraries against multiple versions of the Python interpreter. You can speed it up by telling it to use venv-update for dependency installation; not only will it avoid network access and prefer wheels, but it's also better at syncing a virtualenv (whereas tox will often throw out an entire virtualenv and start over).

To start using venv-update inside tox, copy the venv-update script into your project (for example, at `bin/venv-update`).

Then, apply a change like this to your `tox.ini` file:

```
  [tox]
  envlist = py27,py34
+ skipsdist = true

  [testenv]
+ venv_update =
```

(continues on next page)

(continued from previous page)

```
+     {toxinidir}/bin/venv-update \
+        venv= {envdir} \
+        install= -r {toxinidir}/requirements.txt {toxinidir}
- deps = -rrequirements.txt
  commands =
+     {[testenv]venv_update}
      py.test tests/
      pre-commit run --all-files
```

The exact changes will of course vary, but above is a general template. The two changes are: running venv-update as the first test command, and removing the list of `deps` (so that tox will never invalidate your virtualenv itself; we want to let venv-update manage that instead). The `skipsdist` avoids installing your package twice. In tox<2, it also prevents all of your packages dependencies from being installed by pip-slower.

Python Module Index

v

Index

V