The Future of Python Dependency Management

Kenneth Reitz
Hi.
Requests
http for humans
Requests

HTTP for Humans

```python
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers[‘content-type’]
‘application/json; charset=utf-8’
>>> r.encoding
‘utf-8’
>>> r.text
u’{"type":"User"...’
>>> r.json
{u’private_gists’: 419, u’total_private_repos’: 77, ...}
```
github.com/kennethreitz

- Requests
- OSX-GCC-Installer
- Maya
- Records
- Tablib
- httpbin.org
- Python-Guide.org
- SayThanks.io
- ‘Import This’ Podcast
- Em Keyboard
- Certifi
- Autoenv
Packaging: History
The Past...
Problems with this

- “The Cheeseshop” (e.g. PyPi) was merely an index of packages, not a sole package host.
- Packages were often hosted elsewhere.
- It was running on a single server in Sweden, serving the entire Python community.
- Its use wasn’t a fraction of what it is today, so that wasn’t a problem.
More Obvious Problems

- Very manual process; Not good for automation.
- Globally installed packages, impossible to have two versions of the same library installed.
- People often just copied things into site-packages, manually.
- Poor user experience.
Next Iteration

```
$ easy_install requests-threads
Searching for requests-threads
Best match: requests-threads 0.1.1
Processing requests_threads-0.1.1-py3.6.egg
requests-threads 0.1.1 is already the active version in easy-install.pth

Using /usr/local/lib/python3.6/site-packages/requests_threads-0.1.1-py3.6.egg
Processing dependencies for requests-threads
Finished processing dependencies for requests-threads
```
Improvements!

• Much better user experience for installation.
• Most packages were installed from PyPi.
• Easier to automate programmatically.
• But, no easy_uninstall.
Today’s World
2010 Onward...

- **Pip** became the de-facto replacement for `easy_install`, for managing packages.
- **Virtualenv** became common practice.
- Pinned **requirements.txt** files passed around.
ATTENTION PYTHON COMRADES
NEW ORDERS FROM THE MINISTRY OF PACKAGING!

SONS AND DAUGHTERS OF THE GLORIOUS
PEOPLE’S PYTHONIC REPUBLIC
USE DISTRIBUTED AND PIP
THAT IS ALL.

THIS MESSAGE HAS BEEN APPROVED BY @JEZDEZ, HIGH CHANCELLOR OF PACKAGING
Virtualenv

- Creates isolated “Python Homes” for packages to be installed in, one for each project.

- Very powerful concept, allows for extreme flexibility. Unique to the Python community.

- This is less important for Ruby, because multiple versions of Gems can be installed at the same time on the same system.
Pip: Package Manager

- Resolves, Downloads, Installs & Uninstalls Python packages from Package Indexes or arbitrary URLs.
- Utilizes `requirements.txt` files.
- Manipulates virtual environments.
This practice continues today.
Other Communities

- Node.js: yarn & npm (lockfile)
- PHP: Composer (lockfile)
- Rust: Cargo (lockfile)
- Ruby: Bundler (lockfile)
- Python: pip + virtualenv (no lockfile?)
The Problem
Venv: Downsides

• Difficult to understand abstraction layer.
• Headache for new-comers, increasing the barrier to entry.
• Very manual process, easy to automate, but unnatural to use manually.
• Tools like `virtualenv-wrapper` exist to ease this process.
requirements.txt

- $ pip freeze > requirements.txt
- A pre-flattened dependency tree is required in order to establish deterministic builds.
- Tools like pip-tools were created to ease this pain.
requirements.txt

$ cat requirements.txt

click==6.7
Flask==0.12.2
itsdangerous==0.24
Jinja2==2.10
MarkupSafe==1.0
Werkzeug==0.14.1

- Deterministic.
- Result of “pip freeze”.
- All-inclusive of transitive dependencies.
- Difficult to know “what’s going on”.
$ cat requirements.txt

Flask

- Non-deterministic.
- A representation of the actual requirements.
- Human readable/understandable.
- Does function "properly".
What you want? vs. What you need.
No Lockfile!
The Solution
The Lockfile!
Two Types of Deps...

- **What you want**: unpinned dependencies, highest level deps only (e.g. “Flask”).
- **What you need**: pinned dependencies, all-inclusive of transitive dependencies (e.g. all the things).
Two Requirements Files

- One with "what you want", e.g. unpinned dependencies, highest level deps only.
- One with "what you need", e.g. pinned dependencies, all-inclusive of transitive dependencies.
Two Requirements Files

$ cat requirements-to-freeze.txt

Flask

$ cat requirements.txt

click==6.7
Flask==0.12.2
itsdangerous==0.24
Jinja2==2.10
MarkupSafe==1.0
Werkzeug==0.14.1

See also: pip-tools (requirements.in, requirements.txt)
Not a real solution.
The Real Solution
Pipfile
Pipfile: New Standard

- **Pipfile** is the new standard, replacing **requirements.txt**, in the future.
- **TOML**, so easy to read/write manually.
- Two groups: `[packages]` & `[dev-packages]`.
- Will eventually land in `pip proper`.
Example Pipfile

$ cat Pipfile

[[source]]
url = "https://pypi.python.org/simple"
verify_ssl = true
name = "pypi"

[packages]
flask = "*"

[dev-packages]
pytest = "*"
Resulting Pipfile.lock

- JSON, so easily machine-parsable.
- Contains all transitive dependencies, pinned, with all acceptable hashes for each release.
- Two groups: “default” & “develop”. 
$ cat Pipfile.lock
{
    "_meta": {
        "hash": {
            "sha256": "bdf5339d86cd6b5cc71e6293cbd509572776e1e1957b109fe8963a9bc5bbaf41"
        },
        ...
    },
    "default": {
        "click": {
            "hashes": [
                "sha256:29f99fc6125fbc931b758dc053b3114e55c77a6e4c6c3a2674a2dc986016381d",
                "sha256:f15516df478d5a56180f80e68f206010e6d160fc39fa508b65e035fd75130b"
            ],
            "version": "==6.7"
        },
        "flask": {
            "hashes": [
                "sha256:0749df235e3ff61ac108f69ac178c9770caaeaccad2509cb762ce1f65570a8856",
                "sha256:49f44461237b69ecd901cc7ce66f3e0319b9158743dd27a2899962ab214dac1"
            ],
            "version": "==0.12.2"
        },
    }
}
Pipfile: Problems

- **Pipfile** is not yet integrated into *pip*, and it will likely take quite a long time for this to happen, due to resource constraints.

- But, you can use it today, with...
Pipenv Sales Pitch

- Officially recommended tool from python.org.
- Lets you use Pipfile/Pipfile.lock today.
- Automates away virtualenv entirely.
- Ensures deterministic builds, including hash check verification upon installation.
- Other tools: e.g. $ pipenv graph
Pipenv is the porcelain I always wanted to build for pip. It fits my brain and mostly replaces virtualenvwrapper and manual pip calls for me. Use it.

— Jannis Leidel (former pip maintainer)
Pipenv is finally an abstraction meant to engage the mind instead of merely the filesystem.

— Justin Myles Holmes
DEMO (Q&A)
Thank you!

kennethreitz.org/values