Analysis as Applications: Quick introduction to lockfiles

Matthew Feickert

(@HEPfeickert)

Reproducibility is a hard problem in science

Lorena A. **Barba group**

Barbagroup reproducibility syllabus

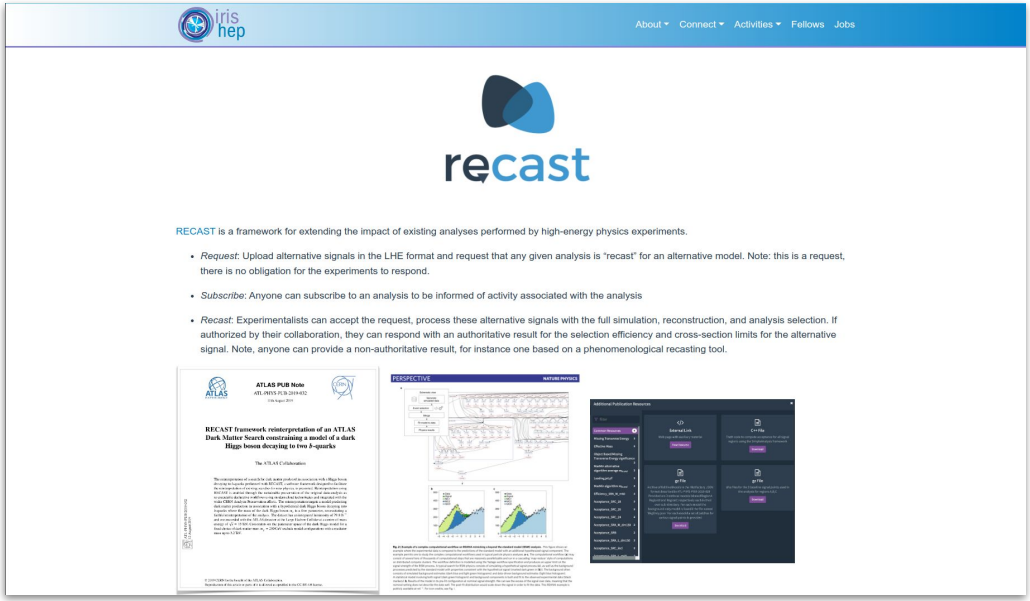
Lockheed P-80A airplane (1946). Credit: NASA Commons. —A reminder to test your code.

Posted on 10.31.2016

Also published on the Medium publication ***"Hacker Noon."***

After my short piece, **"A hard road to reproducibility,"** appeared in *Science*, I received several emails and Twitter mentions asking for more specific tips—both about tools and documents we use in the group to train the team about reproducibility.

> In answer to popular demand, then, I have collected here what we could call the **"Barba-group Reproducibility Syllabus."**



Your environment is part of your science.

Treat your analysis like a Python application.

There are problems with what I’m about to show.

But this is to get thinking.

Lock files are a thing

I’m running out of time, so won’t iterate here (conda-lock, pdm, poetry, pipenv, pip-tools). Not sure what to do? Here’s a “get going and do something” introduction to lock files with pip-tools.

```
# In a virtual environment of course
$ python -m pip install pip-tools
```

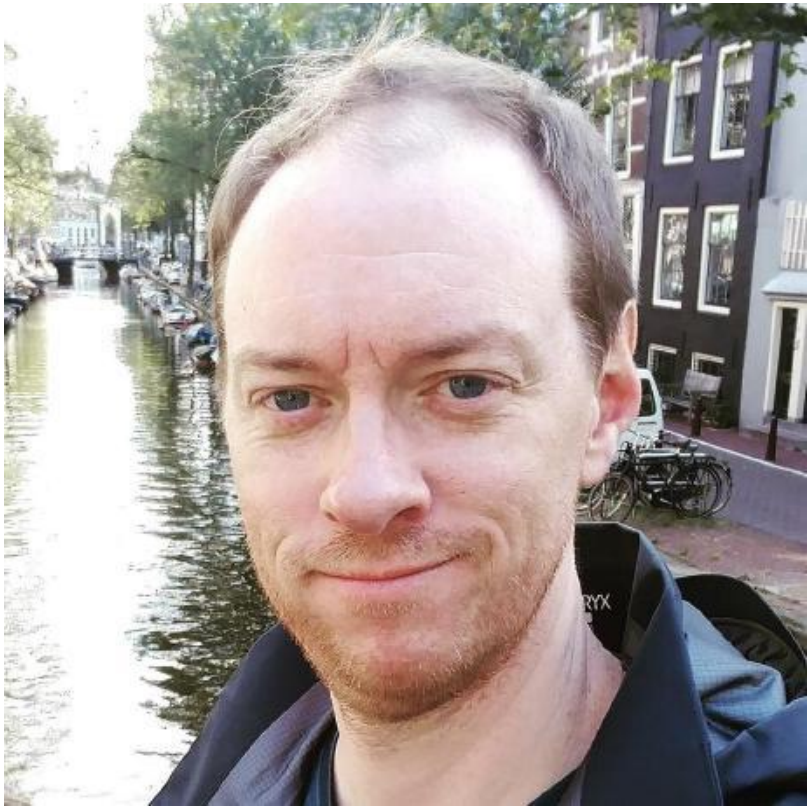
high level `requirements.txt`

```
$ cat requirements.txt

numpy

matplotlib
```

Compile high level dependencies into a lock file

```
$ pip-compile \
    --generate-hashes \
    --output-file requirements.lock \
    requirements.txt
```

pip-secure-install with pip-tools lock files



README.md

pip-secure-install

A GitHub action to have pip install from a requirements file as securely as possible.

Inputs

`python`

The command to run Python (as `-m` is used to run pip). Defaults to `python`.

`requirements-file`

The path to the requirements file. Defaults to `requirements.txt`.

`options`

Additional command-line options to pass to pip (e.g. `--target`).

Design

A few options are turned on for pip to make sure installations are secure and reproducible:

- A requirements file must be specified to make sure all dependencies are known statically for auditing purposes (`-r`).
- No dependency resolution is done to make sure the requirements file is complete (`--no-deps`).
- All requirements must have a hash provided to make sure the files have not been tampered with (`--require-hashes`).
- Only wheels are allowed to have reproducible installs (`--only-binary :all:`).

Brett Cannon

Install with pip-secure-install

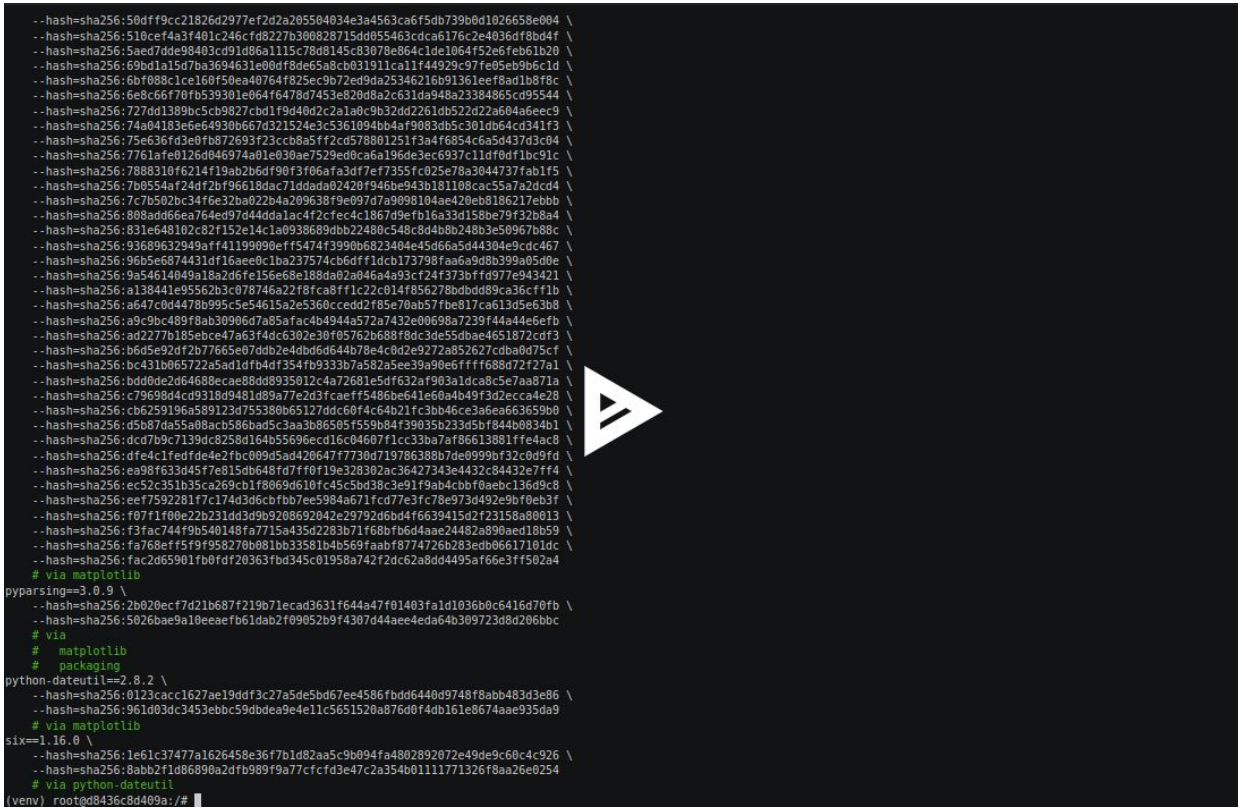
```
$ python -m pip install \
    --no-deps \
    --require-hashes \
    --only-binary :all: \
    --requirement requirements.lock
```

Pin high level `requirements.txt`

```
$ cat requirements.txt

numpy==1.23.1

matplotlib==3.5.2
```

Commit the high level `requirements.txt` and the `requirements.lock` lock file to version control

Actually there’s no time…to the asciinema!



https://asciinema.org/a/508642

Understand your environment.

Do science.

Don’t forget to be awesome!

pip-secure-install

Brett Cannon's "pip-secure-install" recipe:

Use pip flags to make installs secure and reproducible:

- A requirements file must be specified to make sure all dependencies are known statically for auditing purposes (`--requirement`).
- No dependency resolution is done to make sure the requirements file is complete (`--no-deps`).
- All requirements must have a hash provided to make sure the files have not been tampered with (`--require-hashes`).
- Only wheels are allowed to have reproducible installs (`--only-binary :all:`).