
pipx-in-pipx

Release 1.0.1

unknown

Oct 11, 2020

PROJECT OPERATIONS

1	Sharp Edges	3
1.1	Which Python?	3
1.2	Uninstalling	3

docs	read-the-docs	
linux	static analysis	
linux	CPython 3.7	
windows	CPython 3.7	

pipx is great for keeping your CLI tools isolated and your system Python paths clean. However, it still requires that you install *pipx itself* in your system Python.

But *pipx* is a CLI tool installed through *pip*... why not install *pipx* with *pipx*? Why not indeed!

With *pipx-in-pipx*, all you need to do is install :

```
$ pip install pipx-in-pipx
```

But wait! You say. Didn't you just say that we shouldn't install things to system Python?

Yes. What *pipx-in-pipx* actually does is slightly (but only slightly) evil. Rather than actually installing anything when you run "install", *pipx-in-pipx* instead builds a temporary virtual environment, installs *pipx* there, and then uses *that pipx* to install *pipx* in your user local space, just like any other *pipx*-installed tool.

What you end up with is a *pipx* installation that is *itself* managed by *pipx*.

SHARP EDGES

1.1 Which Python?

By default, `pipx` uses its own Python for each environment that it creates. This is whatever you used when you installed `pipx`. This is most commonly the system Python, but if you use `pipx-in-pipx` to install your `pipx`, it is the Python binary in the `pipx`-managed virtualenv for `pipx`. This, in turn, points to the Python that you used to install `pipx-in-pipx`.

This has a few notable side effects:

1. If you uninstall your `pipx`-managed `pipx` (`pipx uninstall pipx`), all of the tools that you installed using that `pipx` will stop working because their Pythons suddenly point to nothing.
 - If you do this, you can fix it by installing `pipx-in-pipx` again.
2. If you reinstall all `pipx` packages (`pipx reinstall-all`), this uninstalls your `pipx`-managed `pipx`.
 - If you do this, you can fix it by installing `pipx-in-pipx` again.
 - If you want to reinstall all other packages, tell `pipx` to ignore `pipx` (`pipx reinstall-all --skip pipx`).
 - To reinstall `pipx`, install `pipx-in-pipx` again.
3. Because all `pipx`-managed packages use the Python in the `pipx` virtualenv, you can change the Python for all packages by installing `pipx-in-pipx` again.

1.2 Uninstalling

`pipx` has a handy feature to uninstall *all* `pipx`-managed tools. Because you have now made `pipx` manage itself, running `pipx uninstall-all` *will also* uninstall `pipx`.

This is not a bug, but a feature. By installing `pipx` using `pipx-in-pipx`, you have expressed an intent that you *want* `pipx` to manage itself. If that's not what you want, this is not the tool for you.

If you at any point uninstall your `pipx`-managed `pipx`, you can simply `pip install pipx-in-pipx` again to rebuild it.

1.2.1 Versioning

pipipxx releases follow [Semantic Versioning](#).

1.2.2 Changelog

1.0.1 – 2019-10-27

Administrivia

- Rename project from pipipxx to pipx-in-pipx.

Bugfixes

- Fix Windows compatibility. [#13https://github.com/mattsb42/pipx-in-pipx/pull/13](#)

Maintenance

- Add Windows CI. [#17https://github.com/mattsb42/pipx-in-pipx/pull/17](#)

1.0.0 – 2019-05-26

Now that I have CI set up for this on at least one platform, I am comfortable saying that it is ready for use.

0.0.1b1 – 2019-05-22

Bugfixes

- Some installs in Linux were failing due to unable to find src files. All logic now in `setup.py`.
- Removed unnecessary `userpath verify` step that was causing errors. [#4https://github.com/mattsb42/pipx-in-pipx/issues/4](#)
- Removed hard requirement for Python 3.6+. Leave that for `pipx` to worry about.

0.0.1b0 – 2019-05-11

Initial MVP.