Pretty Little Python Secrets
Episode 1:

Installing Python Tools and Libraries
The *Right* way

Pilot Episode! What we're going to be tackling

How to solve common problems that security professionals encounter with Python tooling/libraries.

- How to properly manage different Python versions on the same system?
- My "Python is broken". Why is this happening?
  - The disaster that is Python Packaging
  - Apt-get & Pip. Don't cross the streams.
- Going from dependency hell to heaven
  - VirtualEnvs
  - The "Ez" buttons
- Conclusions

Pilot Episode! What we’re going to be tackling

Disclaimer:
We’re going to be focusing on the practical things/solutions from an end user perspective.

We’re not going to be talking about development specific tools (e.g. Pipenv, Poetry).

1st Circle of Hell: Managing Python Versions

- What happens if the tool needs a specific Python version?
  - Or still hasn’t been ported over to Python 3 ?

Managing Python Versions

Traditionally, you’d have to go out and manually compile the version of Python you need…

Managing Python Versions

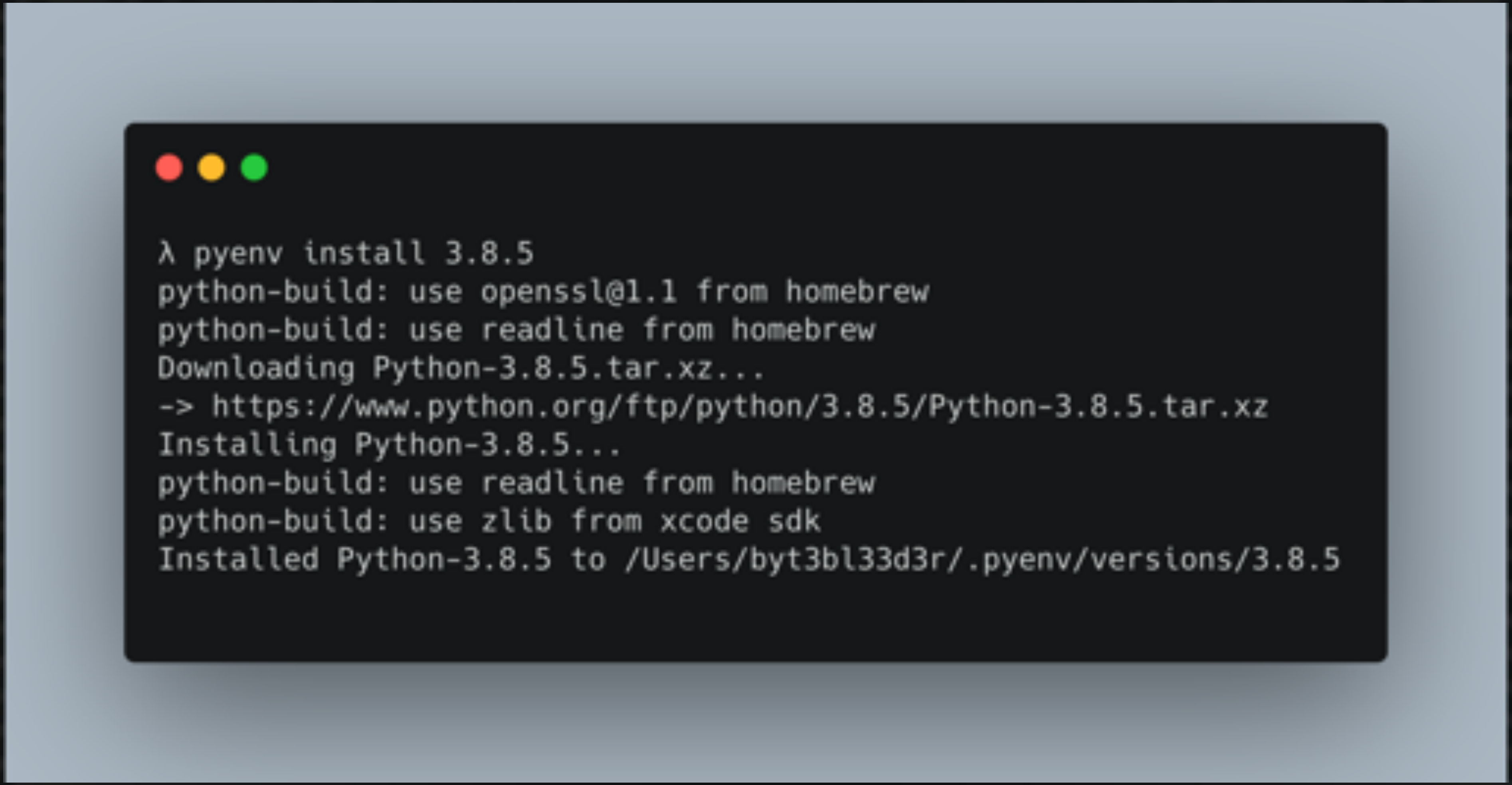
PyEnv!

- https://github.com/pyenv/pyenv

- Automatically compiles Python
- Manages the Path and environment settings for you

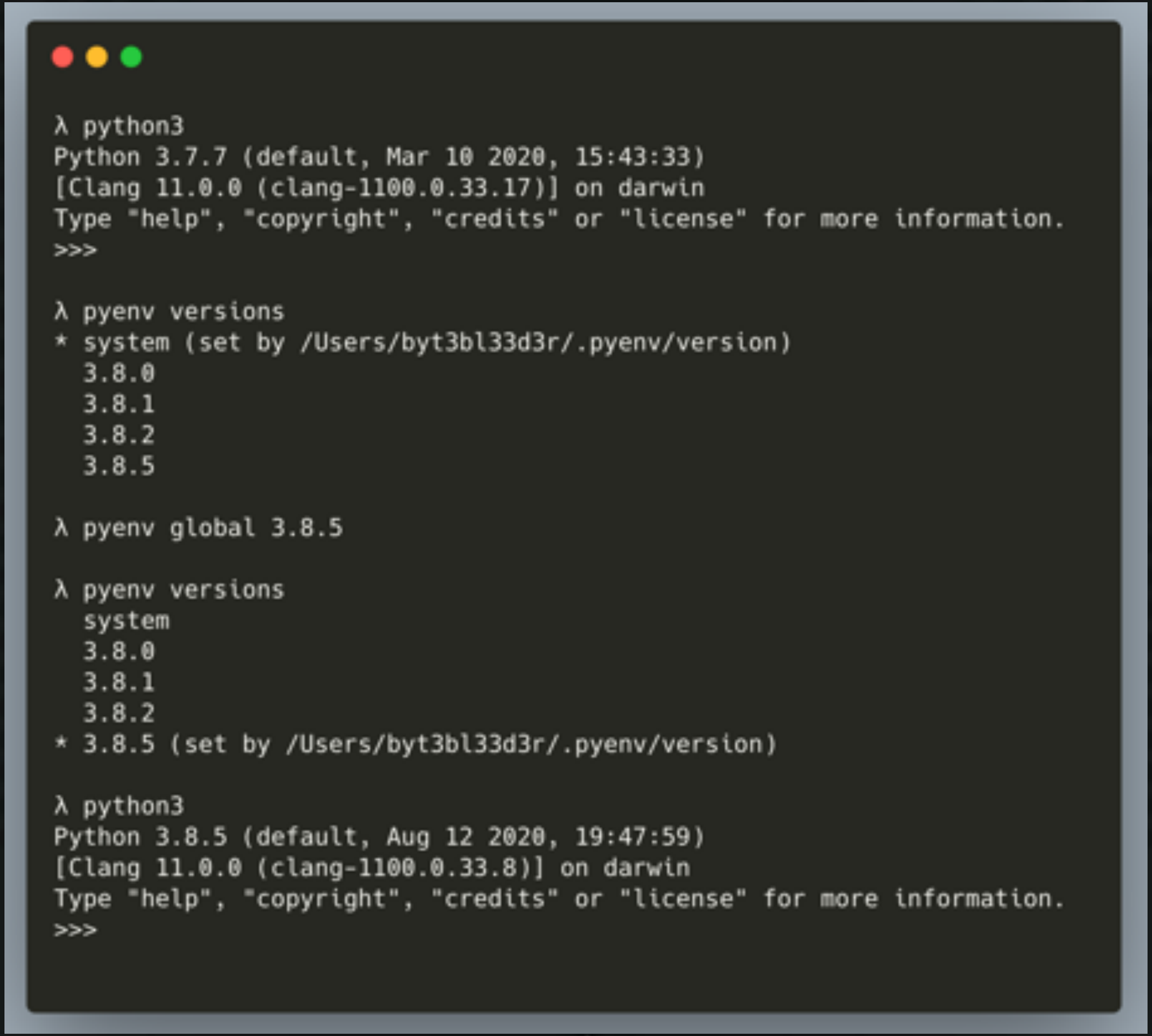
- See the following for installation instructions:
    - brew install pyenv (MacOS)
    - https://github.com/pyenv/pyenv/wiki/Common-build-problems
    - https://github.com/pyenv/pyenv-installer

Managing Python Versions



```
λ pyenv install 3.8.5
python-build: use openssl@1.1 from homebrew
python-build: use readline from homebrew
Downloading Python-3.8.5.tar.xz...
-> https://www.python.org/ftp/python/3.8.5/Python-3.8.5.tar.xz
Installing Python-3.8.5...
python-build: use readline from homebrew
python-build: use zlib from xcode sdk
Installed Python-3.8.5 to /Users/byt3bl33d3r/.pyenv/versions/3.8.5
```

Managing Python Versions



```
λ python3
Python 3.7.7 (default, Mar 10 2020, 15:43:33)
[Clang 11.0.0 (clang-1100.0.33.17)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>

λ pyenv versions
* system (set by /Users/byt3bl33d3r/.pyenv/version)
  3.8.0
  3.8.1
  3.8.2
  3.8.5

λ pyenv global 3.8.5

λ pyenv versions
  system
  3.8.0
  3.8.1
  3.8.2
* 3.8.5 (set by /Users/byt3bl33d3r/.pyenv/version)

λ python3
Python 3.8.5 (default, Aug 12 2020, 19:47:59)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2nd Circle of Hell: Python Dependencies

Need Tool A? it’s available in my distro’s package manager.

- apt-get install tool-a

Oh, now I need Tool B? It’s not available through my package manager!

- pip3 install tool-b

Now I need to use Tool C which I previously installed! Let me run it:

- *Traceback sounds intensify*

How do people usually solve this?

The 6 stages of Python Dependency Grief:

- Reboot
- Revert VM Snapshot
- Futz about trying to fix things for a couple of hours
- Get super frustrated
- Get high and/or drunk
- Give up.

BLACK HILLS Information Security • 2008 •

Installing Python Tools/Libraries

How to “bork your Python install” in two easy steps
(or if you like living that YOLO life):

- apt-get install some-tool-or-library
- pip install some-other-tool-or-library

Installing Python Tools/Libraries

DO NOT EVER EVER EVER *USE BOTH YOUR PACKAGE MANAGER AND PIP TO INSTALL PYTHON TOOLS/LIBRARIES.*

ONLY USE ONE OR THE OTHER!

Do not cross the streams!

Installing Python Tools/Libraries

This is legit the cause of 80% of your Python Issues!

But why does this happen?

apt-get install tool-a

- Tool A requires cryptography 1.5

pip install tool-b

- Tool B requires cryptography 2.0

Overwrites previous cryptography install

Unbeknownst to you, there have been breaking API changes between cryptography 1.5 and 2.0 and/or bugs!

Result? Tool-A is now broken. And everything else that depends on cryptography 1.5.

Isolate, Isolate, Isolate

Sometimes this happening is un-avoidable
(and not always your fault).
Solution?

Isolate the dependencies of each tool/library.
How does one do this?

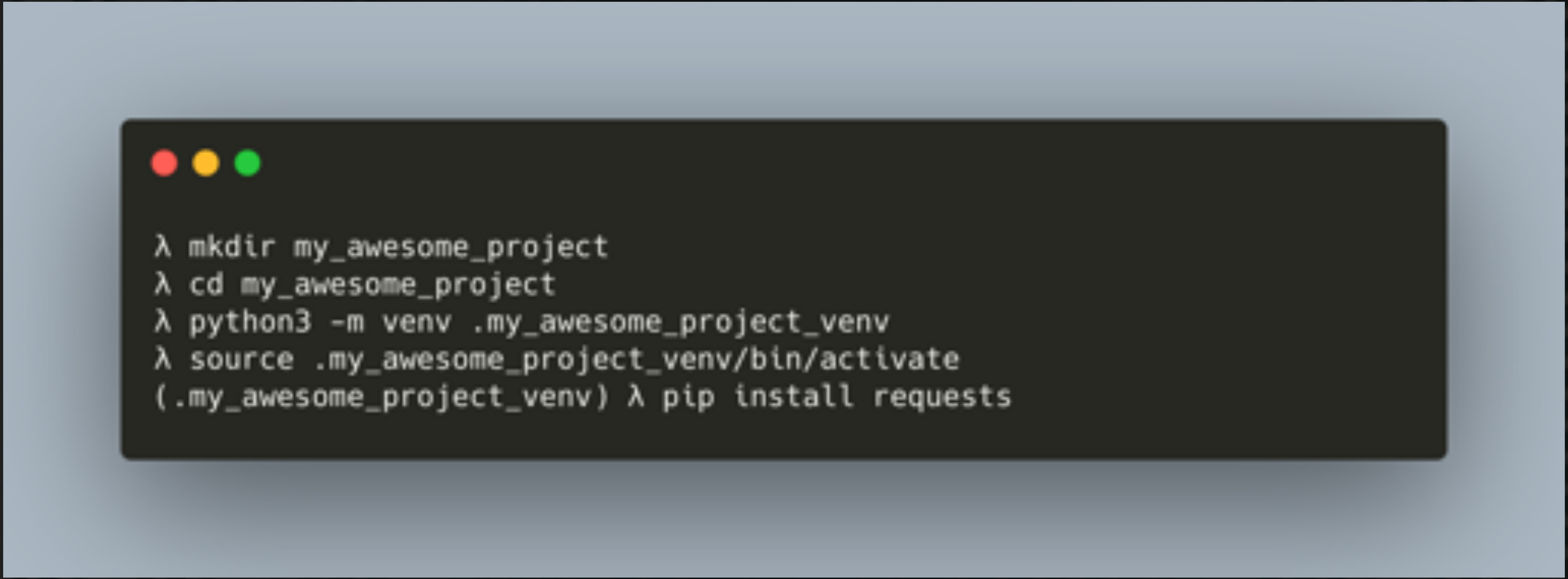
Virtualenvs! (Virtual Environments)

What are VirtualEnvs?

“The venv module provides support for creating lightweight “virtual environments” with their own site directories, optionally isolated from system site directories. Each virtual environment has its own Python binary (which matches the version of the binary that was used to create this environment) and can have its own independent set of installed Python packages in its site directories.” - https://docs.python.org/3/library/venv.html

Venv module is included with Python by default starting with version => 3.3

In practice



```
λ mkdir my_awesome_project
λ cd my_awesome_project
λ python3 -m venv .my_awesome_project_venv
λ source .my_awesome_project_venv/bin/activate
(.my_awesome_project_venv) λ pip install requests
```

Ok, but that’s still a bit of finger milage…

“… Pipx is made specifically for application installation, as it adds isolation yet still makes the apps available in your shell: pipx creates an isolated environment for each application and its associated packages.”

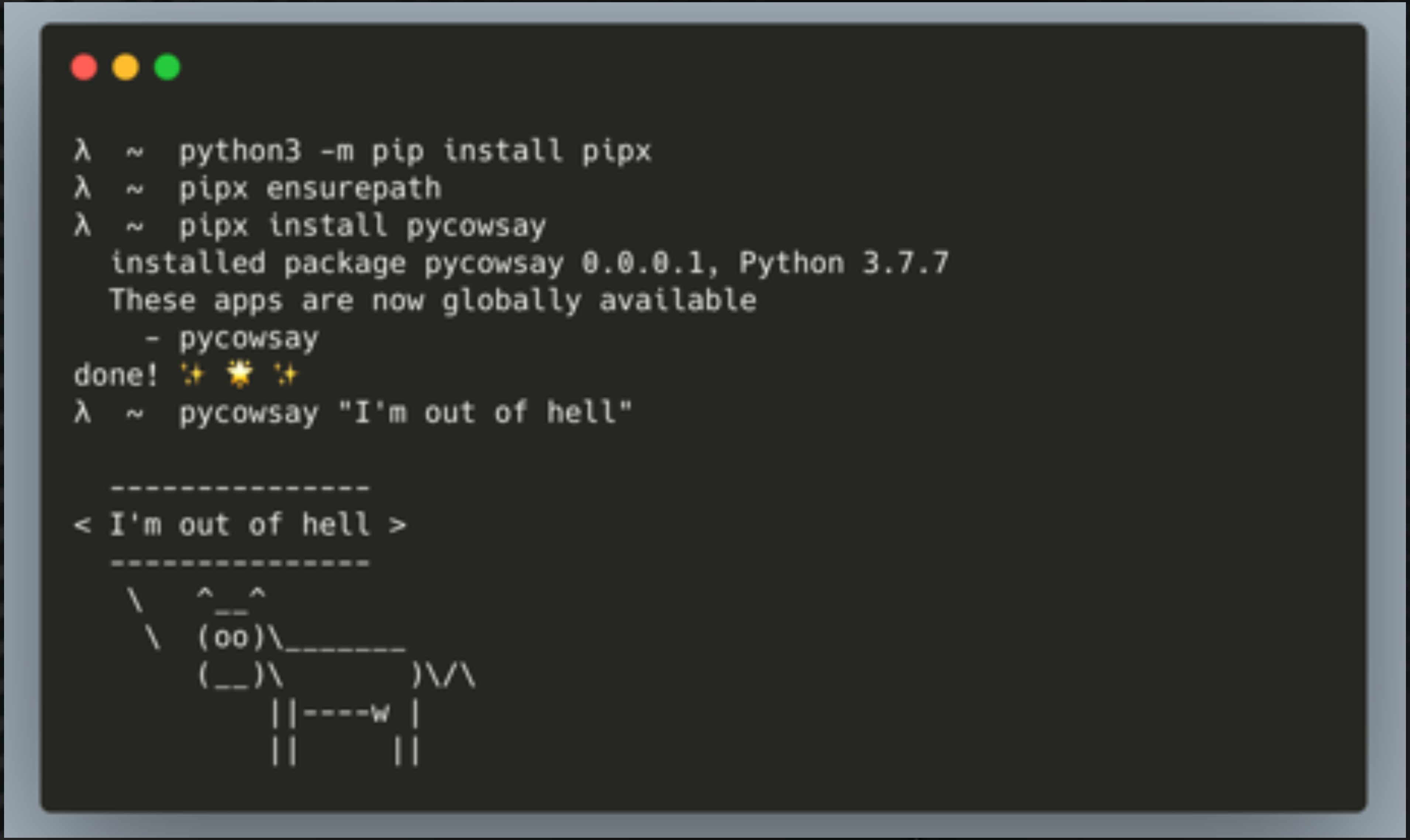
https://github.com/pipxproject/pipx

Essentially…

Basically the same as pip only that it installs everything in Isolated environments automatically for you! Yay!

*Meant only for Python Packages that expose command line tools (*not libraries or for use during development*).*

In practice



```
λ  ~  python3 -m pip install pipx
λ  ~  pipx ensurepath
λ  ~  pipx install pycowsay
  installed package pycowsay 0.0.0.1, Python 3.7.7
  These apps are now globally available
    - pycowsay
done! ✨ 🌟 ✨
λ  ~  pycowsay "I'm out of hell"

  ---------------
< I'm out of hell >
  ---------------
   \   ^__^
    \  (oo)\_______
       (__)\       )\/\
           ||----w |
           ||     ||
```

Making Python Apps Semi-Portable

Sometimes you need to make a Python app portable!

- I have Tool-A installed on machine-A with all of the dependencies nicely isolated.

- I want to run Tool-A on machine-B without going through the hassle of installing it in a virtualenv and/or using pipx.

However, it does have a compatible Python version installed.

How can I accomplish this?

ZipApps!

https://docs.python.org/3/library/zipapp.html

- They're the Jar files of the Python world!

- ZipApps by convention have the .pyz extension (but its just a zip file).

- Functionality has existed since Python 2.7, however was badly documented and somewhat hidden.

- Allows you to package a Python app in a zip file with all it's dependencies and run it !

Creating a ZipApp

https://docs.python.org/3/library/zipapp.html#creating-standalone-applications-with-zipapp

(In your application folder)

- python -m pip install -r requirements.txt --target myapp
- python –m zipapp -p “/usr/bin/env python3” myapp –m “myapp:main”

“This will produce a standalone executable (zip file), which can be run on any machine with the appropriate interpreter available…”

Run it with:

- python myapp.pyz

Creating a ZipApp

Magic!

Mo’ solutions mo’ problems…

Cause we can’t make things too easy, that would be horrible…

One of the “gotchas” with ZipApps is that they don’t work with anything that has C extensions. (Most of the useful libraries have these or depend on libraries that do, e.g. Impacket).

Shiv!

A slightly “modified” version of the ZipApp python module designed to work with libraries with C Extensions!

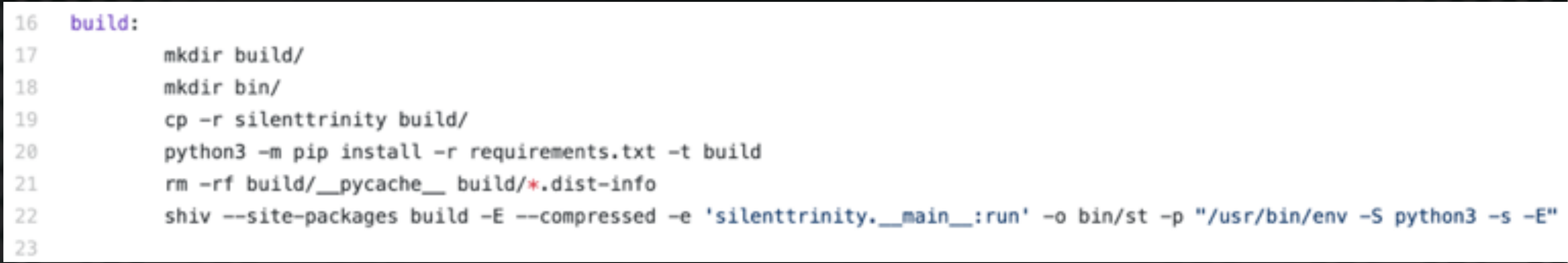
One consequence of this is that “Zipapps created with shiv are not guaranteed to be cross-compatible with other architectures…”

This can be solved using CI/CD pipelines

https://github.com/linkedin/shiv

Shiv!

SILENTTRINITY and CrackMapExec are available as Shiv ZipApps!



```
build:
        mkdir build/
        mkdir bin/
        cp -r silenttrinity build/
        python3 -m pip install -r requirements.txt -t build
        rm -rf build/__pycache__ build/*.dist-info
        shiv --site-packages build -E --compressed -e 'silenttrinity.__main__:run' -o bin/st -p "/usr/bin/env -S python3 -s -E"
```

If all else fails.. Docker !

You can quickly create isolated python environments using Docker!

Not exactly lightweight but it sure is easy!

Python in Docker “gotchas”.

- Python is not Go. People usually say the Alpine base image is the best cause it produces smaller images. This is not the case for Python.

- Stick with Python 3 images based on Debian (e.g. python:latest or python:3.8-slim-buster as of writing)

- The “bible” of Packaging Python apps in Docker:
    - https://pythonspeed.com/docker/

In practice

- git clone https://www.github.com/my-app
- cd my-app
- docker pull python
- docker run –v $(pwd):/my-app -it $IMAGE_ID /bin/bash

- This will drop you into a bash prompt and you'll have a completely isolated environment! Additionally your app will be available at /my-app.

Conclusion

Python Packaging and dependency management is slowly getting better. There still no “silver bullet” at the moment for the end user though.

This is essentially a list of things I wished someone would have told me when I started Python development as a wee boy +10 years ago.

Hopefully this will make your life a lot easier ☺