Pretty Little Python Secrets Episode 2:

Making Python Packaging Simple as a Haiku

Episode 2: From packaging hell to heaven

- Why should I package a Python app
- How to package a Python app
  - Project structure (Important!)
- Uploading your Python app to Pypi
  - Using setuptools (and why setuptools is evil)
  - Twine
- Managing Dependencies
  - Manually vs Pipenv vs Poetry
- How to make life less painful and automate everything
  - Cookiecutter
  - Poetry
- Conclusions

Why should I package my Python app?

By “packaging your Python app” I mean making it available on the Python Package Repository (a.k.a Pypi)

Why should I do this?

So people can install your app with pip!

- pip3 install myawesometool

Or as we learned in PLPS Episode 1, pipx!:

- pipx install myawesometool

Instead of:

- git clone https://github.com/itsme/myawesometool && cd myawesometool
- pip3 install –r requirements.txt
- pip3 install .
- Say a little prayer

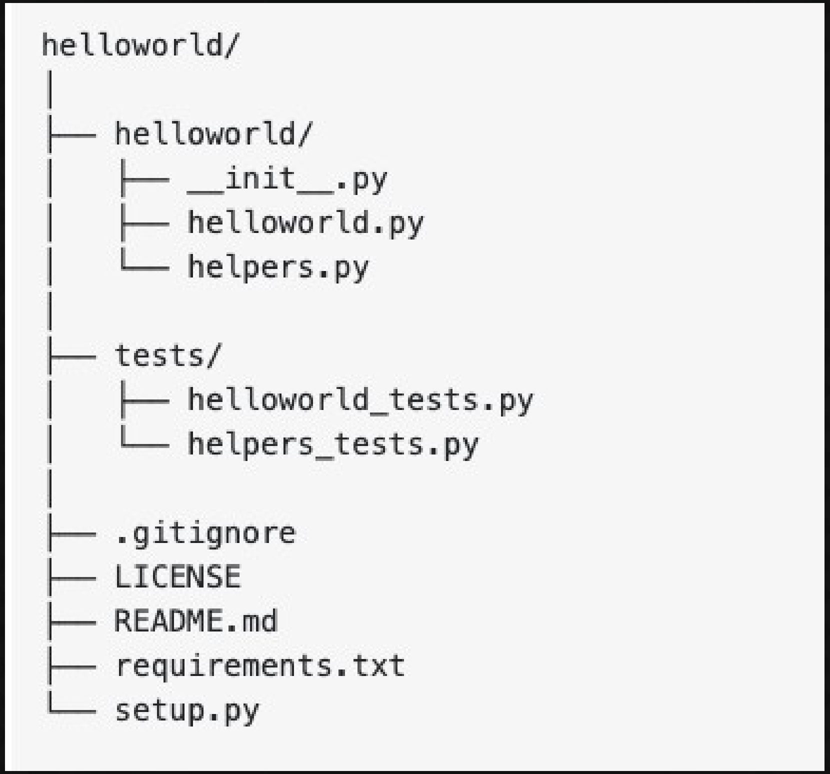
Along with a a lot of other reasons …

How to package a Python application

https://packaging.python.org/tutorials/packaging-projects/

The official docs on the subject are not practical at all in my opinion..

Project layout matters when packaging!



```
helloworld/
│
├── helloworld/
│   ├── __init__.py
│   ├── helloworld.py
│   └── helpers.py
│
├── tests/
│   ├── helloworld_tests.py
│   └── helpers_tests.py
│
├── .gitignore
├── LICENSE
├── README.md
├── requirements.txt
└── setup.py
```

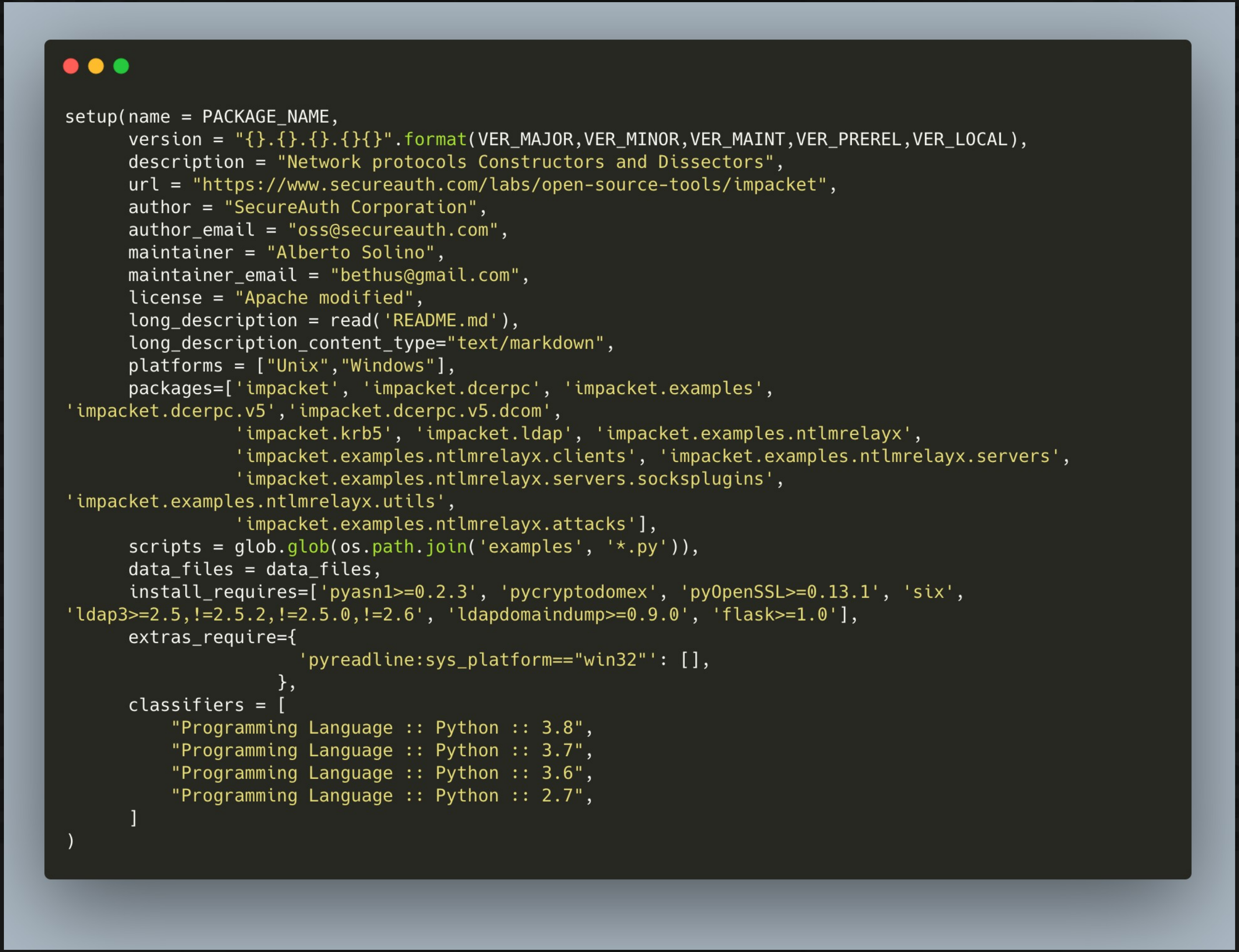
What are all these files?

- .gitignore
  - Files that won't be checked in to source control
    - Github's default Python .gitignore file: https://github.com/github/gitignore/blob/master/Python.gitignore
    - Don't check in your __pycache__ or *.pyc files!
- LICENSE
  - Your license
- README.md
  - Your projects readme in Mardown format
    - (Fun fact, if you google any Python packaging tutorial or look at the official docs, you'll see them referencing a README.rst file as well. This is because up until recently Pypi didn't support Markdown 💩)
- requirements.txt
  - Your dependencies

What are all these files? (part 2)

- MANIFEST.in
  - Here you define all the files that are *not* code file that your package depends on
  - https://python-packaging.readthedocs.io/en/latest/non-code-files.html
- Setup.py
  - Your main setup config file

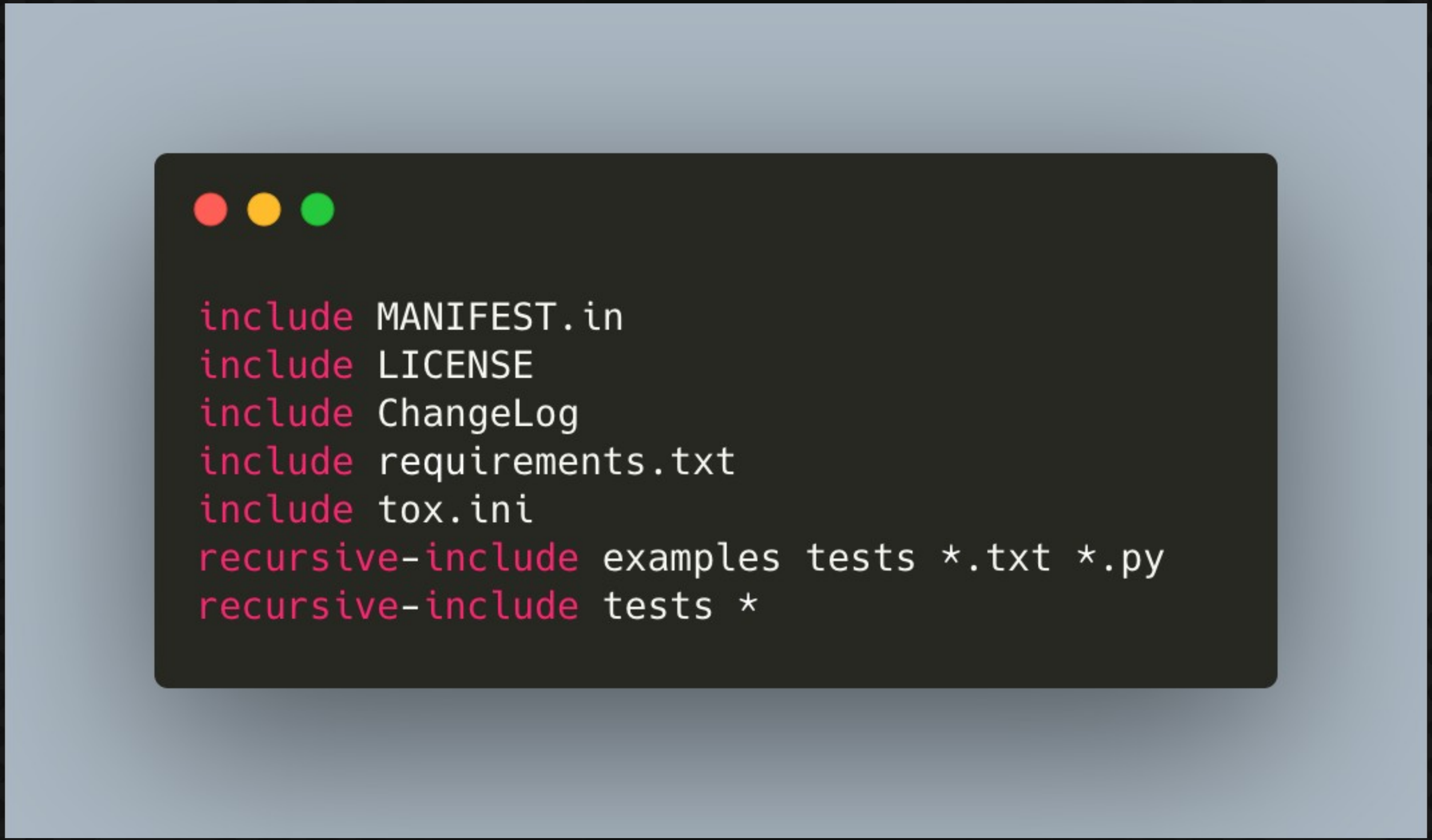
Whats inside a setup.py file?



```
setup(name = PACKAGE_NAME,
      version = "{}.{}.{}.{}{}".format(VER_MAJOR,VER_MINOR,VER_MAINT,VER_PREREL,VER_LOCAL),
      description = "Network protocols Constructors and Dissectors",
      url = "https://www.secureauth.com/labs/open-source-tools/impacket",
      author = "SecureAuth Corporation",
      author_email = "oss@secureauth.com",
      maintainer = "Alberto Solino",
      maintainer_email = "bethus@gmail.com",
      license = "Apache modified",
      long_description = read('README.md'),
      long_description_content_type="text/markdown",
      platforms = ["Unix","Windows"],
      packages=['impacket', 'impacket.dcerpc', 'impacket.examples',
'impacket.dcerpc.v5','impacket.dcerpc.v5.dcom',
                'impacket.krb5', 'impacket.ldap', 'impacket.examples.ntlmrelayx',
                'impacket.examples.ntlmrelayx.clients', 'impacket.examples.ntlmrelayx.servers',
                'impacket.examples.ntlmrelayx.servers.socksplugins',
'impacket.examples.ntlmrelayx.utils',
                'impacket.examples.ntlmrelayx.attacks'],
      scripts = glob.glob(os.path.join('examples', '*.py')),
      data_files = data_files,
      install_requires=['pyasn1>=0.2.3', 'pycryptodomex', 'pyOpenSSL>=0.13.1', 'six',
'ldap3>=2.5,!=2.5.2,!=2.5.0,!=2.6', 'ldapdomaindump>=0.9.0', 'flask>=1.0'],
      extras_require={
                      'pyreadline:sys_platform=="win32"': [],
                    },
      classifiers = [
          "Programming Language :: Python :: 3.8",
          "Programming Language :: Python :: 3.7",
          "Programming Language :: Python :: 3.6",
          "Programming Language :: Python :: 2.7",
      ]
)
```

https://github.com/SecureAuthCorp/impacket/blob/master/setup.py

Whats inside a MANIFEST.in file?



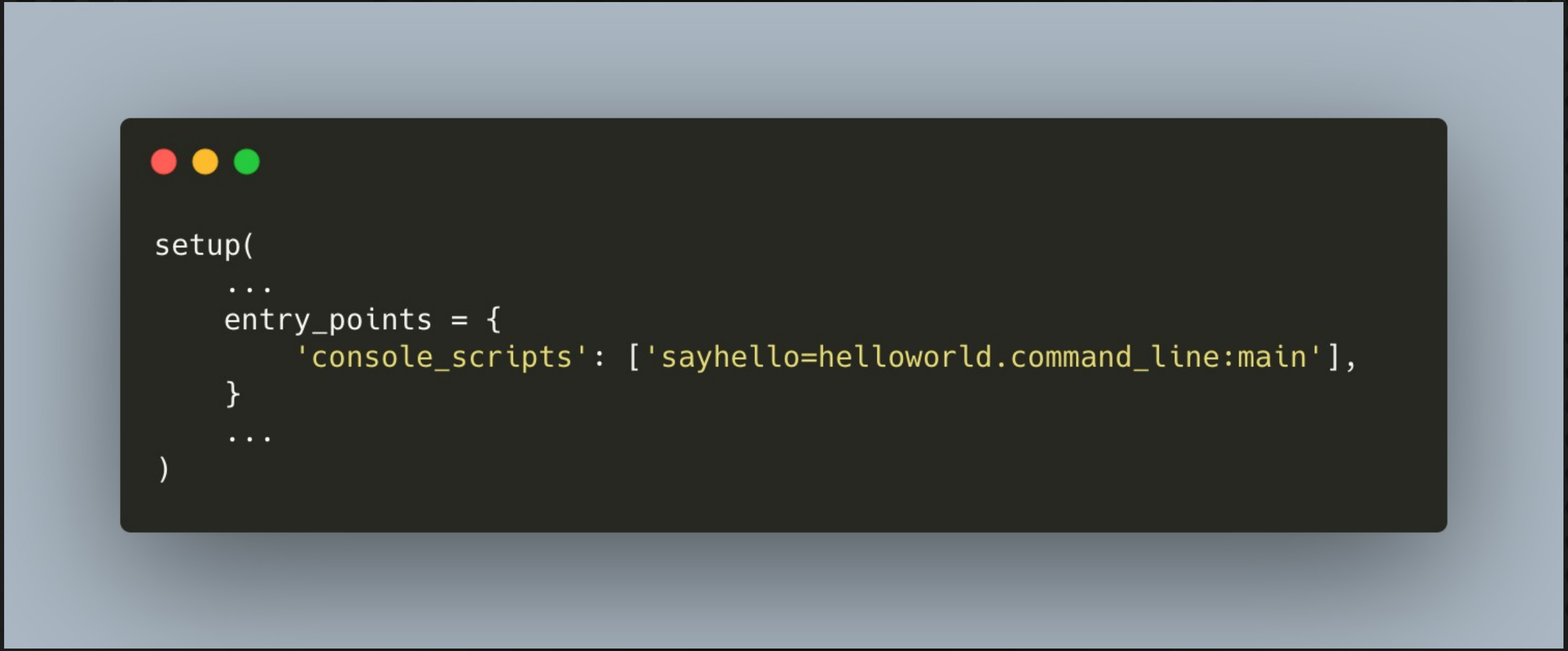
```
include MANIFEST.in
include LICENSE
include ChangeLog
include requirements.txt
include tox.ini
recursive-include examples tests *.txt *.py
recursive-include tests *
```

https://github.com/SecureAuthCorp/impacket/blob/master/MANIFEST.in

Exposing Command Line Applications

You can use the `scripts` or `entry_points` argument in the setup function in the setup.py file…

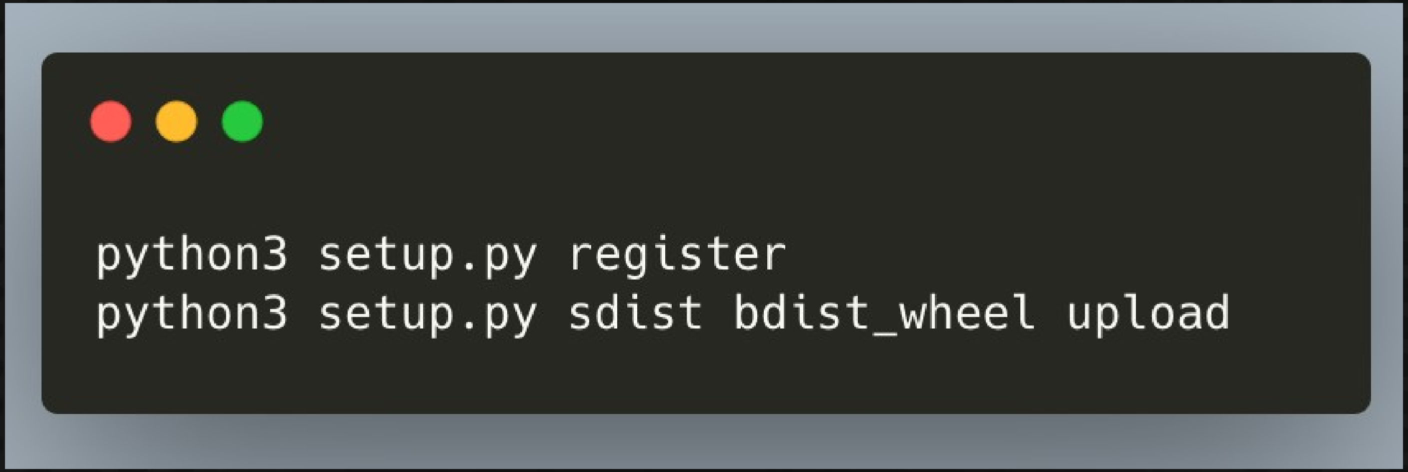
- `entry_points` is generally preferred over `scripts`



```
setup(
    ...
    entry_points = {
        'console_scripts': ['sayhello=helloworld.command_line:main'],
    }
    ...
)
```

Ok so, how do I upload the dang thing to Pypi?

Well if you were to read a bunch of StackOverflow posts and the quasi-legit official docs here: https://python-packaging.readthedocs.io/en/latest/minimal.html#publishing-on-pypi



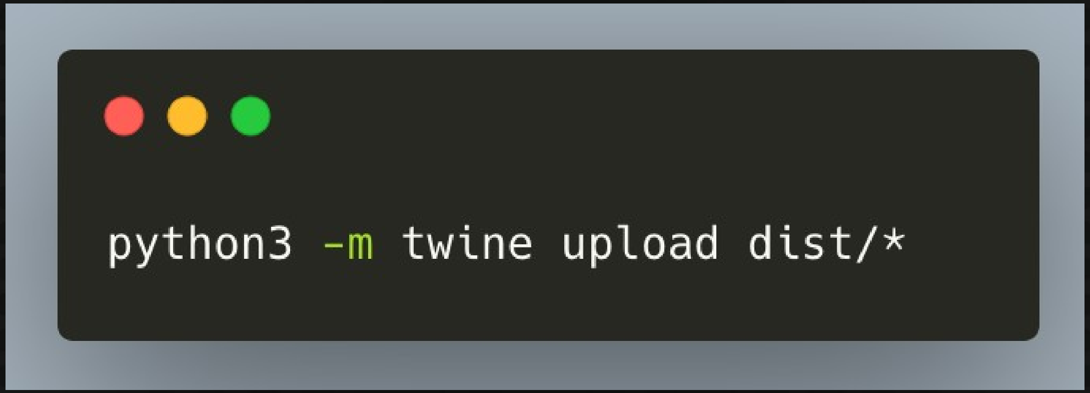
```
python3 setup.py register
python3 setup.py sdist bdist_wheel upload
```

⚠ Not secure! Do not *ever* do this! ⚠

GO TO SECURITY JAIL
BONK
imgflip.com

Ok so, how do I upload the dang thing to Pypi?

Use Twine!
https://twine.readthedocs.io/



```
python3 -m twine upload dist/*
```

This actually is secure ✓

Wow that’s a lot of stuff…

Yeah I agree... Also a few issues with this entire process:

1. You have to install twine manually...
   - pip install twine

2. How do I keep my dependencies specified in requirements.txt in sync with the ones defined in my setup.py file?
   - You don’t 💩

3. How do I manage dependencies in the first place?
   - Next section!

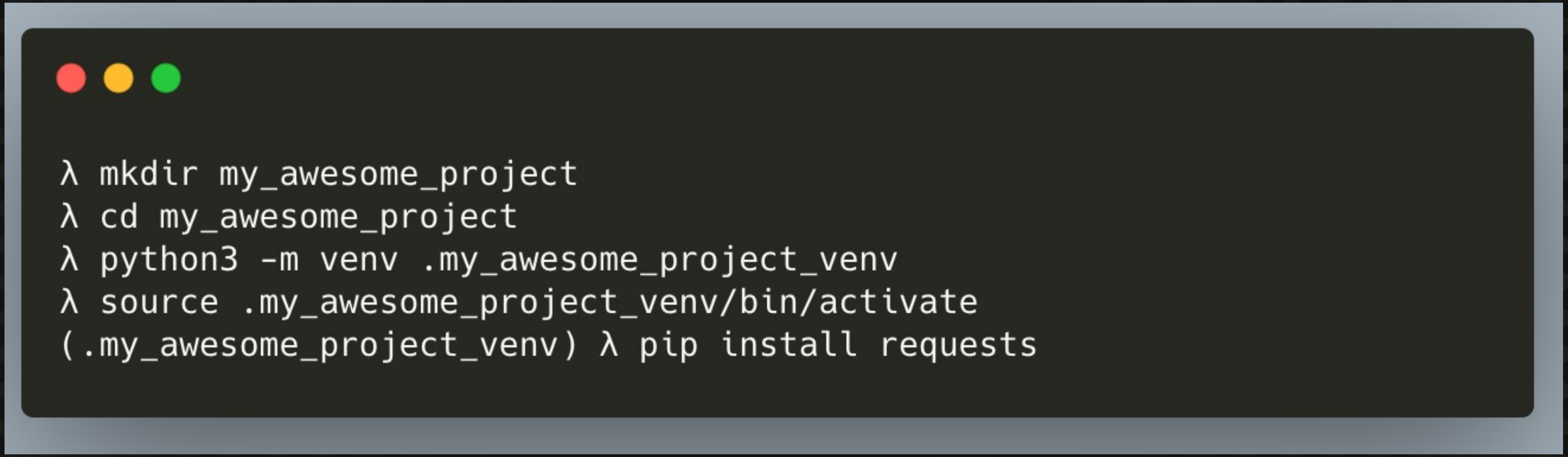
4. How do I actually go about setting up a sane dev environment
   - Good question (next section)

How do I setup a development environment in the first place?

How do I manage dependencies In the first place?

The old-school/manual way…

1. Setup a Virtualenv! (Isolate, Isolate, Isolate)
   - We talked about virtualenvs in Episode 1



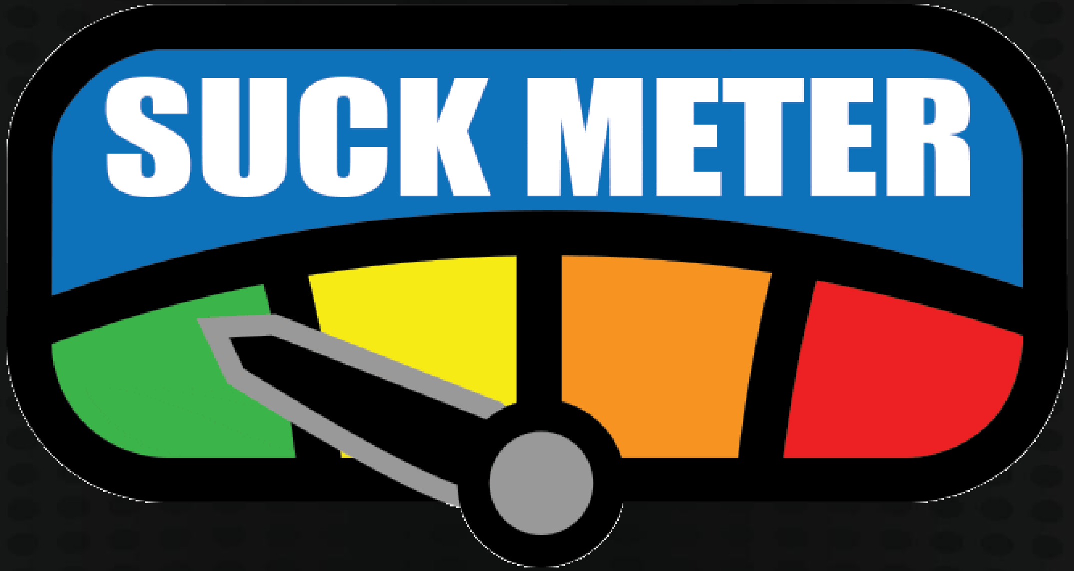
```
λ mkdir my_awesome_project
λ cd my_awesome_project
λ python3 -m venv .my_awesome_project_venv
λ source .my_awesome_project_venv/bin/activate
(.my_awesome_project_venv) λ pip install requests
```

2. pip install requests

3. pip freeze > requirements.txt

Problems

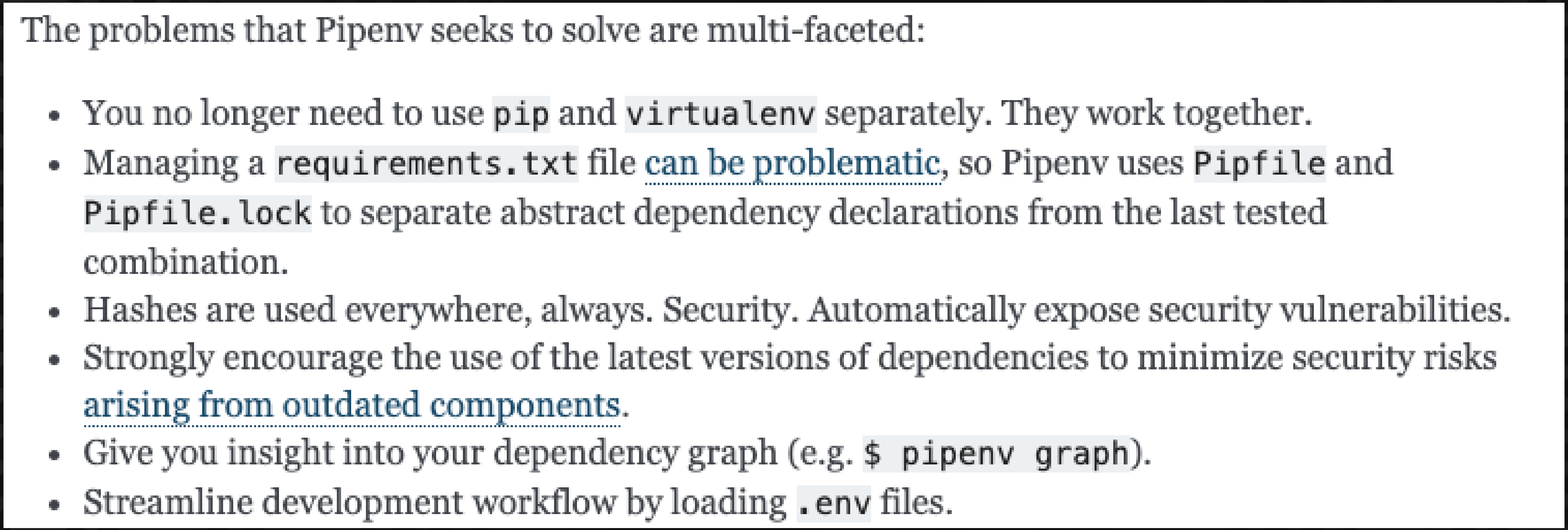
- Setting up a virtualenv for every single new project is a PITA
- Switching to the virtualenv manually is a major PITA
- You have to remember to run `pip freeze > requirements.txt` every time you add a dependency
- Managing the requirements.txt file is problematic when it comes to pinning dependencies. Additionally, `pip freeze` doesn’t hash your dependencies automatically, meaning you can potentially install a package that’s been tampered with (or even backdoored).



BLACK HILLS
Information Security
• 2008 •

Pipenv! Solves everything???

https://pipenv.pypa.io/en/latest/

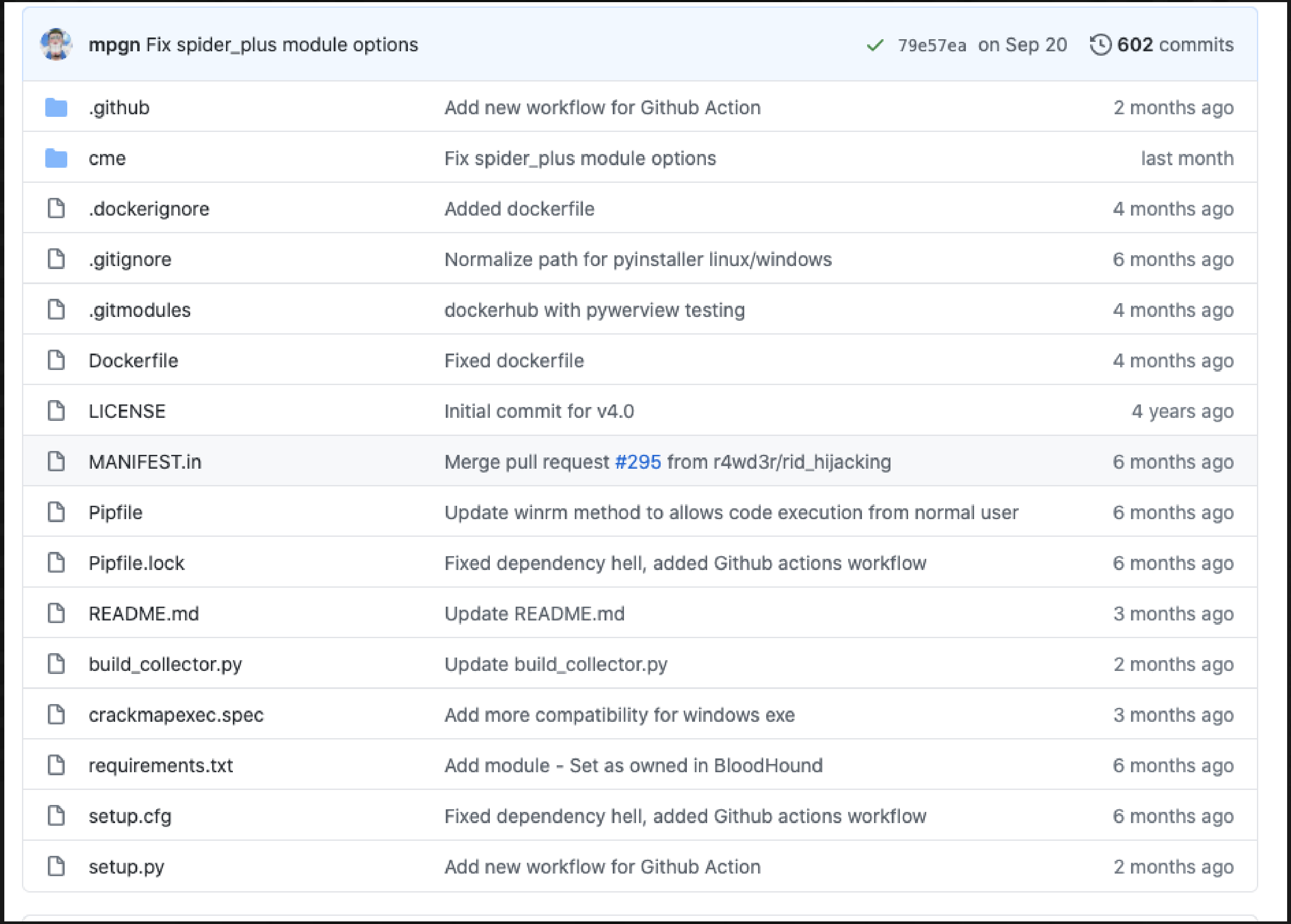

The problems that Pipenv seeks to solve are multi-faceted:

- You no longer need to use `pip` and `virtualenv` separately. They work together.
- Managing a `requirements.txt` file can be problematic, so Pipenv uses `Pipfile` and `Pipfile.lock` to separate abstract dependency declarations from the last tested combination.
- Hashes are used everywhere, always. Security. Automatically expose security vulnerabilities.
- Strongly encourage the use of the latest versions of dependencies to minimize security risks arising from outdated components.
- Give you insight into your dependency graph (e.g. `$ pipenv graph`).
- Streamline development workflow by loading `.env` files.

Pipenv solves problems, but brings in new ones…

- You’ve now added an additional 2 more files (Pipfile, Pipefile.lock) to your ever growing list of files needed for packaging your app.
- Dependency resolution is slow as hell.
- You still have to use Twine to actually upload the package to Pypi. Pipenv only manages dependencies for you.
- When you uninstall a dependency with Pipenv, it doesn’t actually uninstall the dependency.
- It has an insane amount of bugs when it comes to dependency resolution

I mean, look at this madness



mpgn Fix spider_plus module options ✓ 79e57ea on Sep 20 602 commits

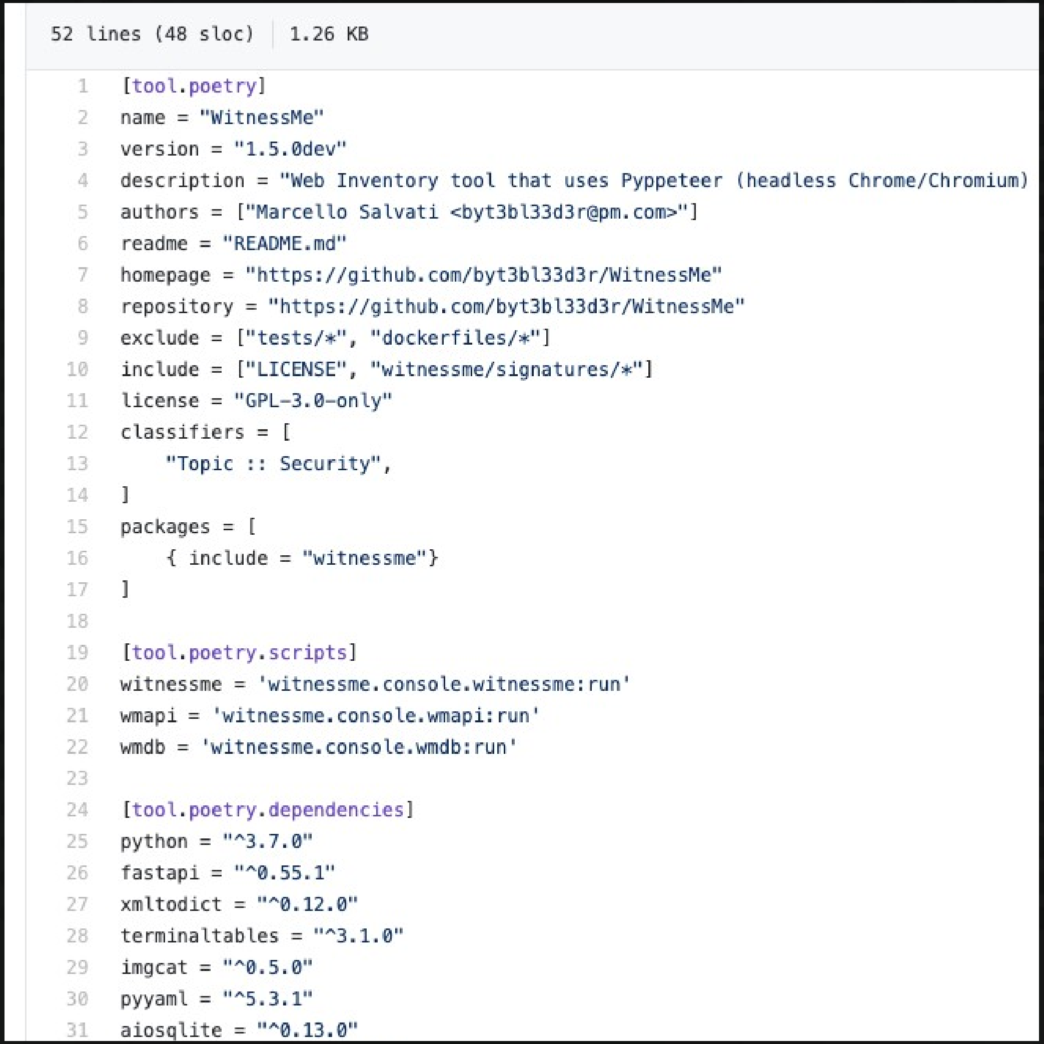
.github	Add new workflow for Github Action	2 months ago
cme	Fix spider_plus module options	last month
.dockerignore	Added dockerfile	4 months ago
.gitignore	Normalize path for pyinstaller linux/windows	6 months ago
.gitmodules	dockerhub with pywerview testing	4 months ago
Dockerfile	Fixed dockerfile	4 months ago
LICENSE	Initial commit for v4.0	4 years ago
MANIFEST.in	Merge pull request #295 from r4wd3r/rid_hijacking	6 months ago
Pipfile	Update winrm method to allows code execution from normal user	6 months ago
Pipfile.lock	Fixed dependency hell, added Github actions workflow	6 months ago
README.md	Update README.md	3 months ago
build_collector.py	Update build_collector.py	2 months ago
crackmapexec.spec	Add more compatibility for windows exe	3 months ago
requirements.txt	Add module - Set as owned in BloodHound	6 months ago
setup.cfg	Fixed dependency hell, added Github actions workflow	6 months ago
setup.py	Add new workflow for Github Action	2 months ago

Poetry to the rescue…

https://python-poetry.org/

- Does everything that Pipenv does only 100 times better.
- Condenses *everything* down to a single pyproject.toml file
  - No more setup.py, setup.cfg, Manifest.in, Pipfile, pipfile.lock, requirements.txt
- Handles packaging *and* dependencies.
- You can also use it to upload your package to Pypi.

Pyproject.toml file



```
52 lines (48 sloc)  1.26 KB
```

```toml
[tool.poetry]
name = "WitnessMe"
version = "1.5.0dev"
description = "Web Inventory tool that uses Pyppeteer (headless Chrome/Chromium)
authors = ["Marcello Salvati <byt3bl33d3r@pm.com>"]
readme = "README.md"
homepage = "https://github.com/byt3bl33d3r/WitnessMe"
repository = "https://github.com/byt3bl33d3r/WitnessMe"
exclude = ["tests/*", "dockerfiles/*"]
include = ["LICENSE", "witnessme/signatures/*"]
license = "GPL-3.0-only"
classifiers = [
    "Topic :: Security",
]
packages = [
    { include = "witnessme"}
]

[tool.poetry.scripts]
witnessme = 'witnessme.console.witnessme:run'
wmapi = 'witnessme.console.wmapi:run'
wmdb = 'witnessme.console.wmdb:run'

[tool.poetry.dependencies]
python = "^3.7.0"
fastapi = "^0.55.1"
xmltodict = "^0.12.0"
terminaltables = "^3.1.0"
imgcat = "^0.5.0"
pyyaml = "^5.3.1"
aiosqlite = "^0.13.0"
```

https://github.com/byt3bl33d3r/WitnessMe/blob/master/pyproject.toml

Cookiecutter

Tired of manually having to setup the same folder structure over and over again for each new Python app?

Cookiecutter!!

https://github.com/cookiecutter/cookiecutter

"A command-line utility that creates projects from ***cookiecutters*** *(project templates), e.g. creating a Python package project from a Python package project template"*

My Cookiecutter project template

The most minimalistic and modern Python project setup I’ve come up with so far.

https://github.com/byt3bl33d3r/pythoncookie

Conclusion

Python Packaging is still a bit of a mess but getting better.

TL;DR I recommend using Poetry for Python dependency management and packaging.

It makes your life easier.