

Gerenciadores de pacotes Python: Indo além do pip install

Ítalo Epifânio

Quem sou eu

- Norte-rio-grandense da trombinha do elefante
- Cientista da Computação
- Machine Learning Engineer
- Entusiasta de comunidades de tecnologia open source



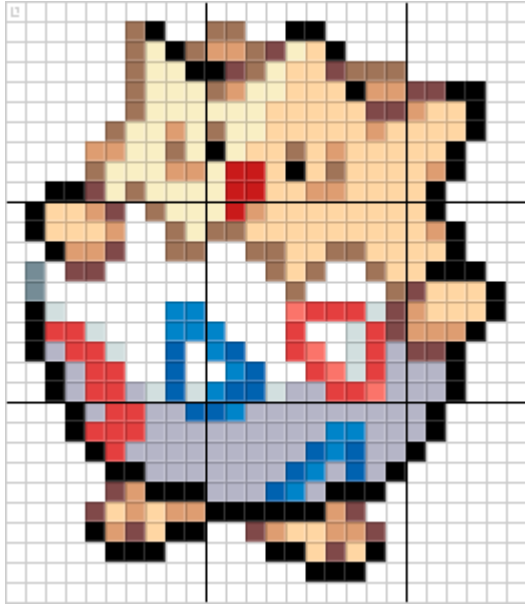
Sumário

- Pip install nem sempre basta, e escolher a ferramenta certa é desafiador
- Por que Python tem diversas ferramentas para gerenciar dependências?
- Estudo de caso na hora de escolher uma ferramenta
- Apresentação de algumas delas

O que é ferramenta e o que é pokemon?

- Hatch
- UV
- Togepy
- Pipenv
- pip-tools
- Poetry
- PDM
- Rhyhorn
- Rye
- Pyflow
- Pixi
- Mamba

Pokemons



Togepi



Rhyhorn

Ferramentas

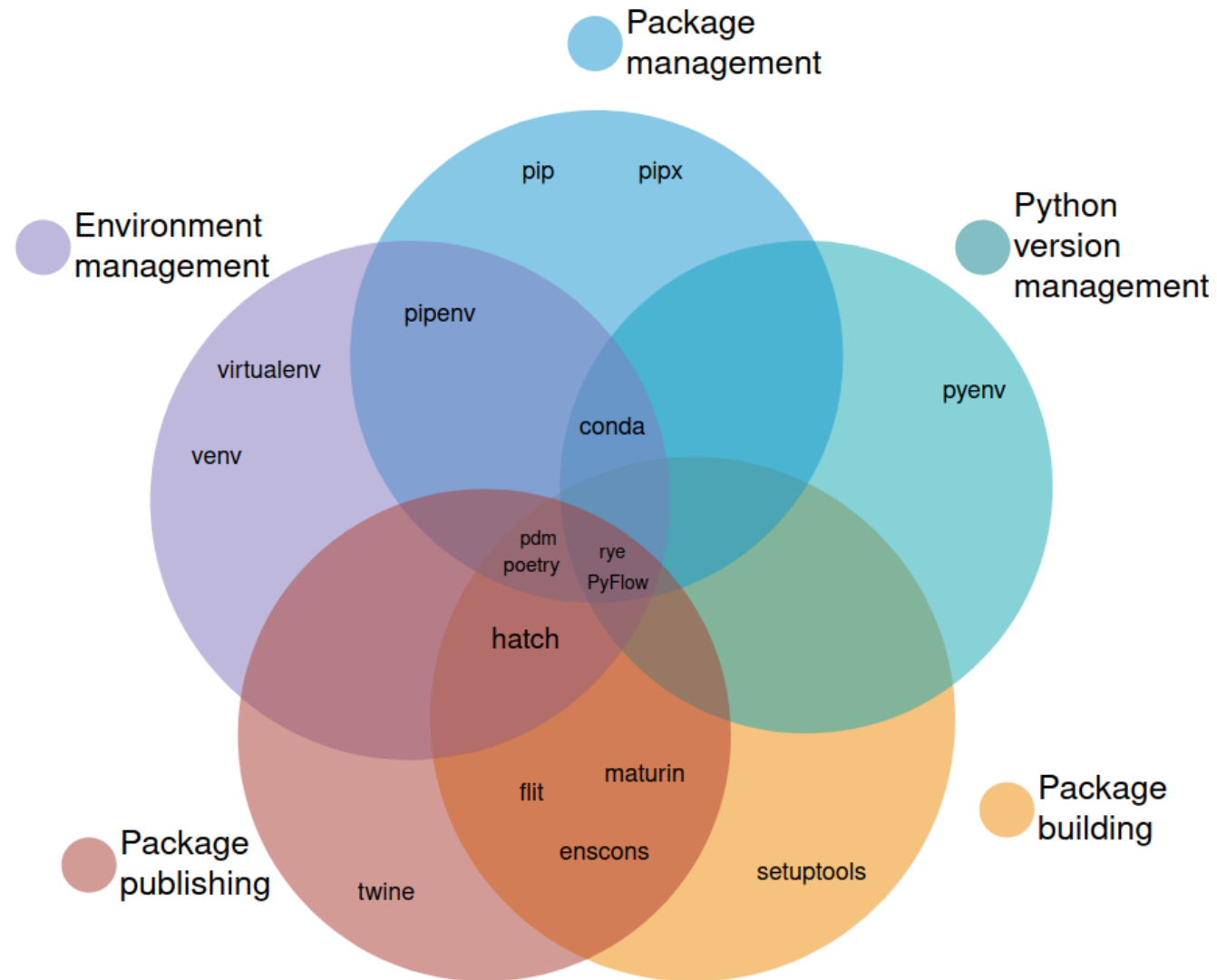


Imagem do blog post da [Anna Lena Popkes](#) blog post

Ferramentas

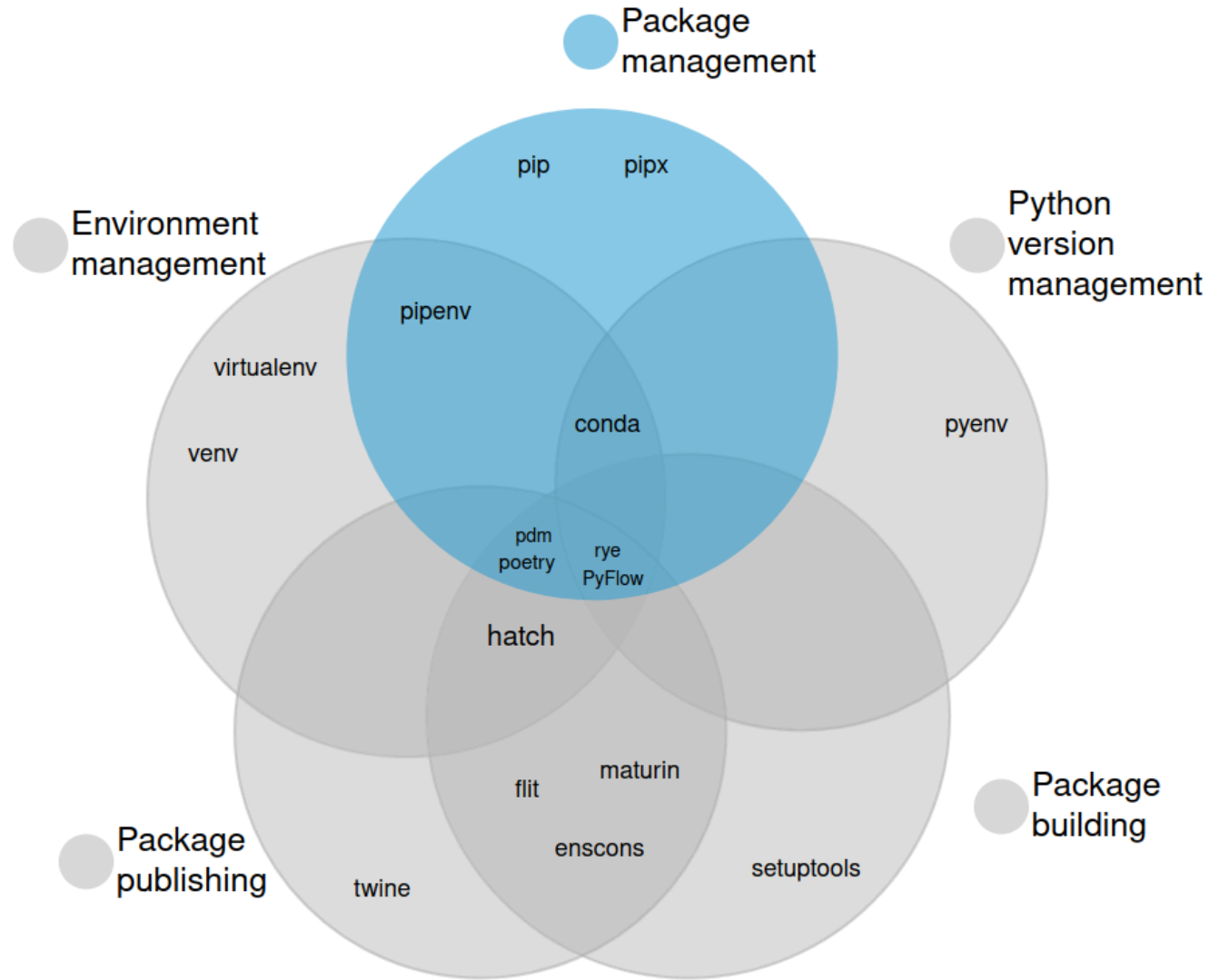


Diagrama de Venn

Muitas opções, um só caminho



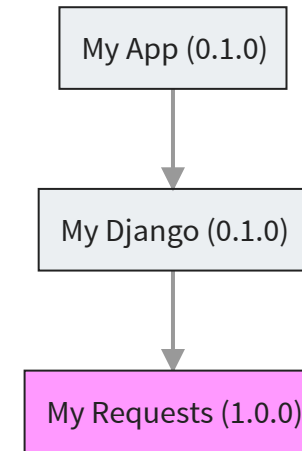
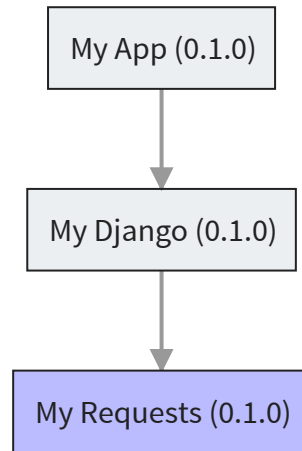
Eu só quero instalar meu pacote e suas dependências!

Pip

- Gerenciador comum ao Python, já vem com a linguagem
- `pip install algum-pacote-do-pypi`
- `pip install -e .[dev]`
- `pip install -r requirements.txt`

Pip

Quem resolve as dependências das minhas dependências?



- `cd my_app`
- `pip install -r requirements`
- Bump my_requests e mude o código

Pip

Como garantir a reprodutibilidade do meu app e garantir o meu sextou sem incidentes?

Lock files

- Arquivos que armazenam todas as dependências
 - Garante reprodutibilidade
 - Acelera a resolução das dependências (instalação mais rápida)
 - Aumenta a segurança da aplicação (supply chain)
 - Aumenta as chances de um sextou

Lock files

- Como?
 - Pin de versões dos pacotes e os pacotes dos pacotes...
 - Artifact hashing para checar se o hash do download atual é o mesmo do arquivo .lock

Lock files

```
1 # This file is automatically @generated by Poetry 2.1.2 and should not be c
2
3 [[package]]
4 name = "my-django"
5 version = "0.1.0"
6 description = ""
7 optional = false
8 python-versions = ">=3.11"
9 groups = ["main"]
10 files = []
11 develop = false
12
13 [package.dependencies]
14 my-requests = {path = "/home/italo/open/avaliando-gerenciadores-de-pacotes-
15
16 [package.source]
17 type = "directory"
18 url = "../my_django"
19
```

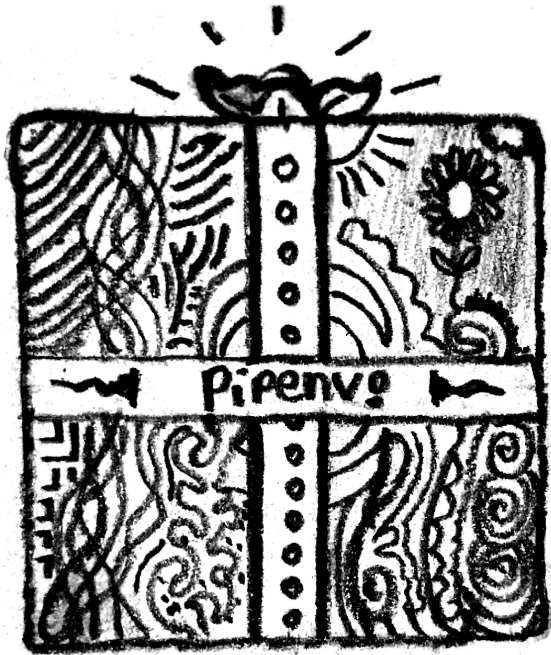
Posso usar `pip freeze > requirements.txt`?

- Remover/atualizar pacotes não garante que suas dependências serão removidas do `requirements.txt`
- Não tem capacidade de resolver problemas de diferentes versões (conflitos)
 - Pacote A -> `requests>=1.0.0`
 - Pacote B -> `requests>=1.2.0`

Se lock de dependências é tão legal por que o pip não tem um?

- 2016 - [PEP 518 – Specifying Minimum Build System Requirements for Python Projects](#) - setup.py/config.cfg -> pyproject.toml
- 2021 - [PEP 665 – A file format to list Python dependencies for reproducibility of an application](#) - Rejeitado
- 2025 - [PEP 751 – A file format to record Python dependencies for installation reproducibility](#) - Aceito

Pipenv



- Cria automaticamente `Pipfile` e `Pipfile.lock`
- Instala corretamente a versão do Python (via Pyenv)
- Automaticamente cria `virtualenv`
- Automaticamente carrega `.env`
- Não segue o padrão python `pyproject.toml`

Poetry

- “Sucessor” do Pipenv
- `poetry.lock`
- Não segue a [PEP 621 - Storing project metadata in pyproject.toml](#)
- Além das capacidades do pipenv, também publica pacotes

PDM

- Similar ao Poetry, mas segue todas as PEPs

Rye

- Pacote em fase experimental do criador do Flask

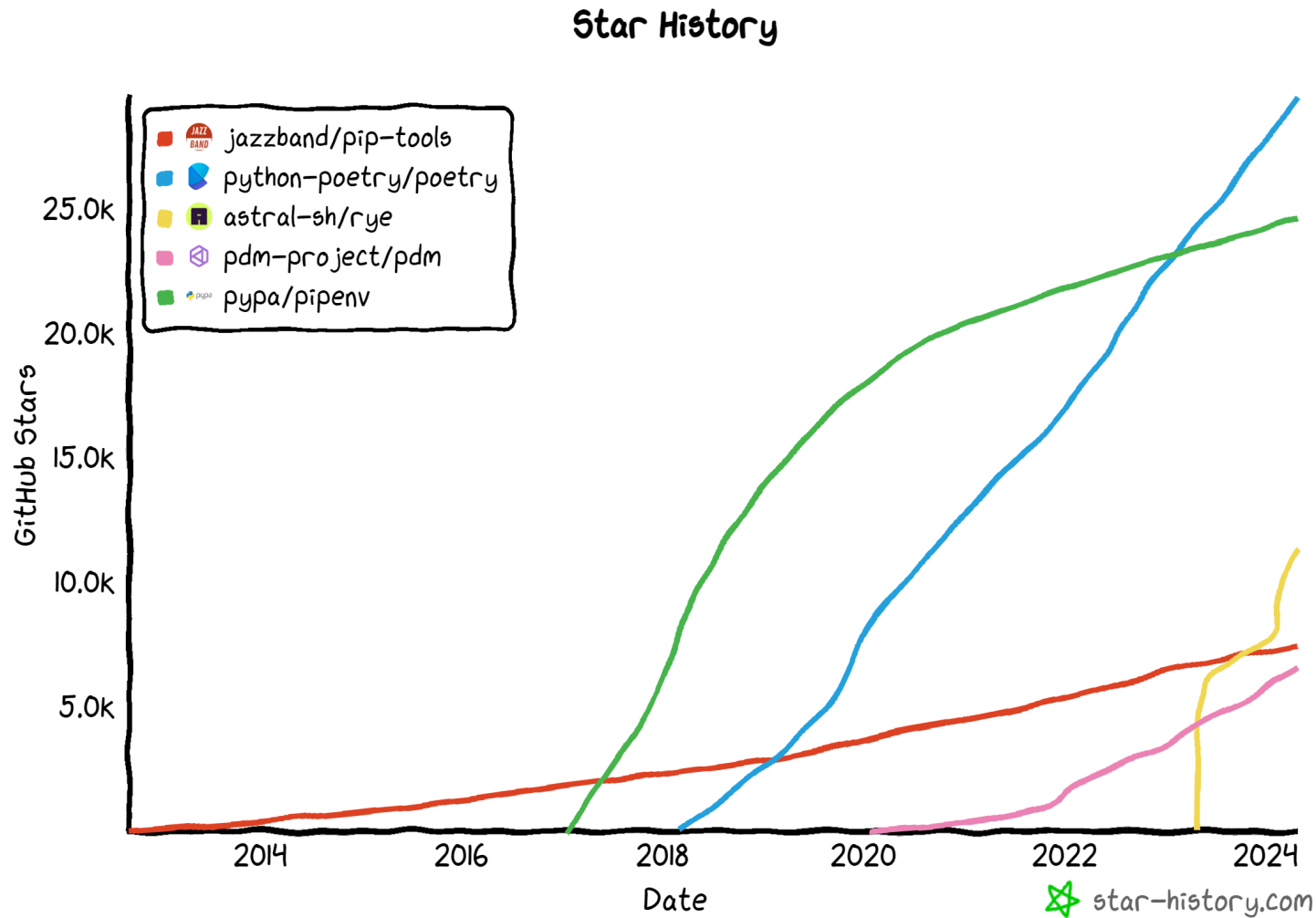
Pip-tools

- `python -m piptools compile` usa um requirements.in e gera requirements.txt com todas as dependências pinadas
- `python -m piptools sync`
- `python -m piptools update`

Comparação








- Durante as discussões do **AI & Data** levantamos algumas necessidades:
 - Queremos ter um lock (reprodutibilidade)
 - Queremos usar uma ferramenta madura
 - Preferimos seguir as PEPs do Python
 - Preferimos uma ferramenta com baixa curva de aprendizado
- Opções: pipenv, poetry e pip-tools

Comparação



Popularidade das ferramentas

Comparação

Ferramenta	cross-platform	integração com env atual	curva de aprendizado
poetry		 (padrão próprio)	Razoável
pip-tools	 (pip-compile-cross-platform)		Pequena
pipenv	 	 (Pipfile vs Pyproject.toml)	Razoável

Por que não utilizar o Pipenv?

- Já foi **abandonado uma vez** (credibilidade)
- Resolução de conflitos já foi um problema no passado (talvez ainda seja lento)
- Não segue a estrutura `pyproject.toml` sugerida pelas PEPs
- Além de instalar os pacotes cria a `virtualenv` (mais do que precisamos)

Por que não utilizar o pip-tools?

- Problemas de geração de lock files cross-platform:
 - Cada plataforma precisaria de um requirements.in diferente

Poetry

- Não segue a estrutura pyproject.toml sugerida pelas PEPs
- Além de instalar os pacotes, criar a virtualenv, também publica os pacotes (mais do que precisamos)
- No time de AI & Data não usaremos a virtualenv, continuaríamos usando pyenv, venv e rodaríamos o `poetry install`, `poetry add some-dep`, `poetry update`

Conda e mamba

- Criados para projetos de data science
- Gerenciam além da instalação dos pacotes python, como dependencias do sistema (ex. CUDA)
- Existe a possibilidade de usar [conda + poetry](#)
- Conda e mamba não tem .lock por default, usam [conda-lock](#)
- Mamba é mais rápido na resolução de pacotes e uso com docker

Uso das ferramentas

- `pip-compile --generate-hashes --output-file=requirements.txt requirements.in`
- `poetry add requests`
- `uv pip un/install requests`
 - `uv pip compile --generate-hashes pyproject.toml -o requirements.txt`
 - `uv pip compile --generate-hashes requirements.in -o requirements.txt`

Conclusão

- Lock de dependências em Python ainda é uma questão em aberto, com PEP encaminhada
- Dentre as ferramentas investigadas: Poetry e pip-tools são soluções maduras que atendem nossos requisitos
- uv tem sido experimentada nos últimos meses e sua velocidade vem intrigando o time a iniciar sua adoção

Referências

- [An unbiased evaluation of environment management and packaging tools](#)
- [Como poetry usa .lock cross platform](#)
- [Python packaging user guide](#)
- [PEP 665 lock file rejeitada](#)
- [PEP 751 – A file format to record Python dependencies for installation reproducibility](#)
- [Faster conda install](#)



Github com apresentação

