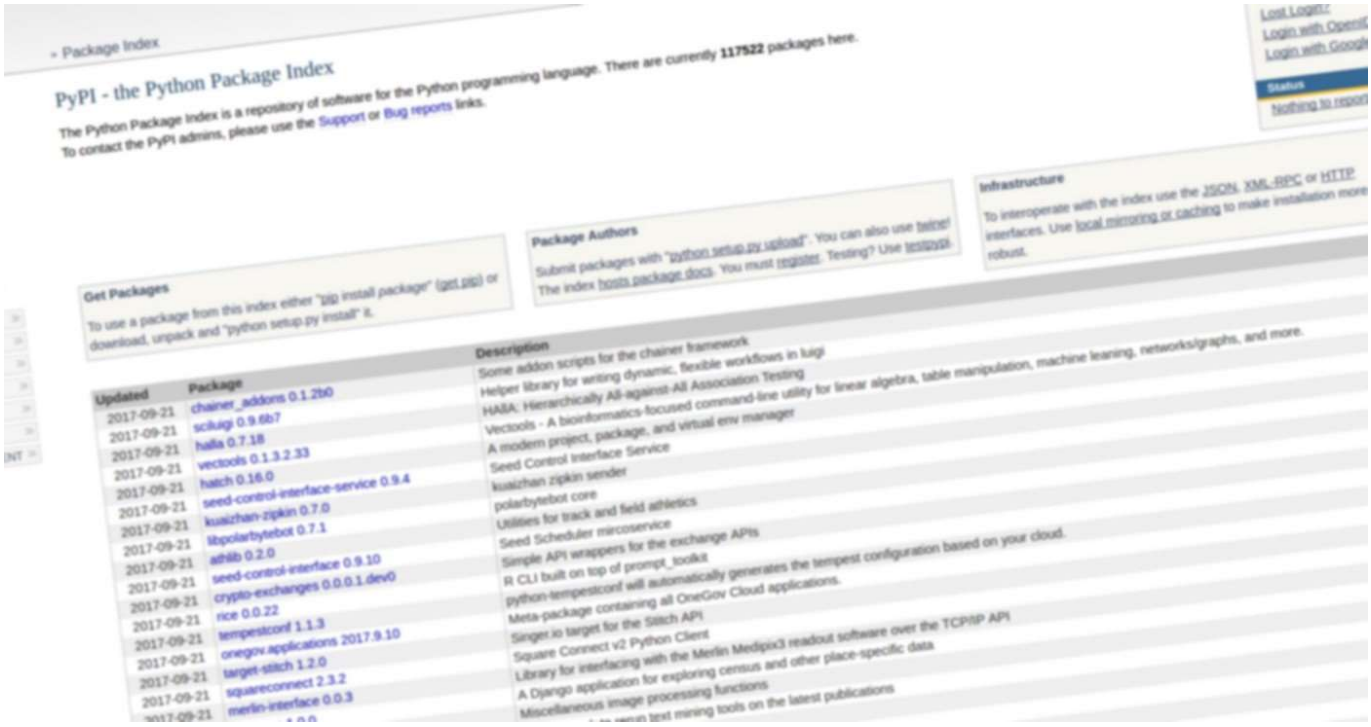
Patrick Porto

302 Followers About Follow

Gerenciando dependências do Python com pipenv

Patrick Porto Jan 27, 2017 · 3 min read



O pipenv é uma ferramenta completa para o gerenciamento de dependências em projetos Python, unindo Pipfile, pip e virtualenv em uma única ferramenta. Ele cria automaticamente o virtualenv e gerencia as suas dependências.

As principais vantagens de utilizar pipenv são:

- Possibilita a verificação de versão do python ou sistema operacional.
- Adiciona no Pipfile as dependências instaladas e retira as desinstaladas automaticamente.
- Encontra o Pipfile recursivamente dentro do projeto, permitindo a instalação de dependências à partir de qualquer localização dentro do projeto.

Instalação do pipenv

A instalação do pipenv é realizada através do pip:

```
pip install pipenv==7.4.3
```

Criando seu ambiente

O primeiro passo que você precisa para utilizar o pipenv é criar um ambiente.

```
pipenv --python 3.6
```

Instalação de dependências

O pipenv permite separar as dependências por tipo de ambiente, como produção ou desenvolvimento. Para instalar uma dependência no ambiente padrão, só precisamos executar o seguinte comando:

```
pipenv install Flask
```

Caso o objetivo seja instalar uma dependência para o ambiente de desenvolvimento, instale através do seguinte comando:

```
pipenv install pytest --dev
```

automaticamente para a virtualenv na localização em que o comando foi executado, além dois arquivos importantes: `Pipfile` e `Pipfile.lock`.

Observação: É recomendável que a pasta `.venv` seja ignorada pelo controle de versão.

Ativação do virtualenv

Nós precisamos ativar o virtualenv para utilizar as bibliotecas instaladas pelo `pipenv` através do seguinte comando:

```
pipenv shell
```

Caso você queira executar algum comando dentro do virtualenv sem precisar ativar o virtualenv, utilize o seguinte comando:

```
pipenv run <comando>
```

Exemplo:

```
pipenv run python main.py
```

Estrutura do Pipfile

O Pipfile é o ponto de partida para os projetos com pipenv. Ele descreve a url do repositório, as dependências de produção e desenvolvimento, além de informações como versão do Python e do sistema operacional. A estrutura do Pipfile é como a seguite:

```
[[source]]
url = "https://pypi.python.org/simple"
verify_ssl = true
```

```
[packages]
requests = "*"

[requires]
python_version = "3.4"
```

As dependências são adicionadas em `[packages]` e, quando é utilizado o argumento `--dev`, em `[dev-packages]`.

O Pipfile possui suporte a especificação de versão conforme a PEP 440. Por exemplo, para especificar uma versão do Django superior a 1.10 seria necessária a seguinte definição:

```
[packages]
Django = ">1.10"
```

Verificando o ambiente

O Pipfile proporciona a verificação do ambiente e informa se ela atende aos requisitos de versão do Python ou do sistema operacional. Essas definições são de acordo com a PEP 508.

Caso o nosso projeto apenas possa ser utilizado em ambientes Linux e que tenha Python 2.7, podemos definir da seguinte forma no Pipfile:

```
[requires]
python_version = '2.7'
platform_system = 'Linux'
```

A verificação do ambiente para o projeto é feita através do seguinte comando:

```
pipenv check
```

```
Checking PEP 508 requirements…
Passed!
```

Caso o ambiente não atenda aos requisitos, aparece uma mensagem de erro como a seguir:

```
AssertionError: Specifier u'platform_system' does not match u'Linux'.
```

Conclusão

O Pipfile surgiu como um substituto definitivo para o arquivo requirements.txt. Ele é recomendado oficialmente como ferramenta de gerenciamento de pacotes e tem demonstrado que é a melhor coisa depois do surgimento do pip.

Python Pip Pipenv