¡Adiós pip!

Abrazando Poetry como nuevo gestor de paquetes

Juanjo Salvador
Backend Developer
Cathedral Software

Juanjo Salvador

Backend Developer @ Cathedral Software

> Python Dev especializado en Django, fotógrafo aficionado, usuario de Linux, y wikipedista en mis ratos libres.

@Linuxneitor

@jsalvador@mastodon.social

TL;DR

- ¿Qué es Poetry?
- De 0 a Producción
- Configuración del proyecto
- Extra: Migrar nuestro proyecto desde pip
- Preguntas y respuestas

Illustrations by Pixeltrue on icons8

¿Qué es Poetry?

¿Por qué Poetry? Y otras preguntas existenciales

Dependencias

Gestiona las dependencias de tu paquete organizadas por bloques (producción, desarrollo, test...)

Publicar paquetes

Publica tus paquetes en PyPI o cualquier otro repositorio de forma sencilla y prácticamente automatizada.

Entornos aislados

Poetry por defecto utiliza un entorno virtual por proyecto, pero permite utilizar alguno existente si fuese necesario.

Builds

Crea builds y wheels de tu paquete listos para su distribución, con un solo comando.

Seguimiento

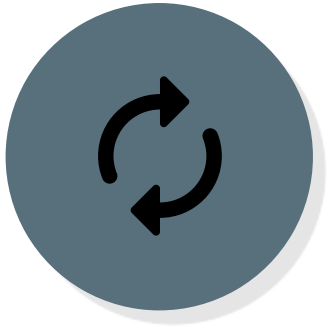
Mantén al día las dependencias de tu software, evitando con ello vulnerabilidades y agujeros de seguridad.

Comandos

Define tus propios comandos (tests, linters, estilos...) dentro del propio proyecto.

Fácil de utilizar

Si bien Poetry viene con un montón de posibilidades de serie, su utilización carece de demasiada curva de aprendizaje.

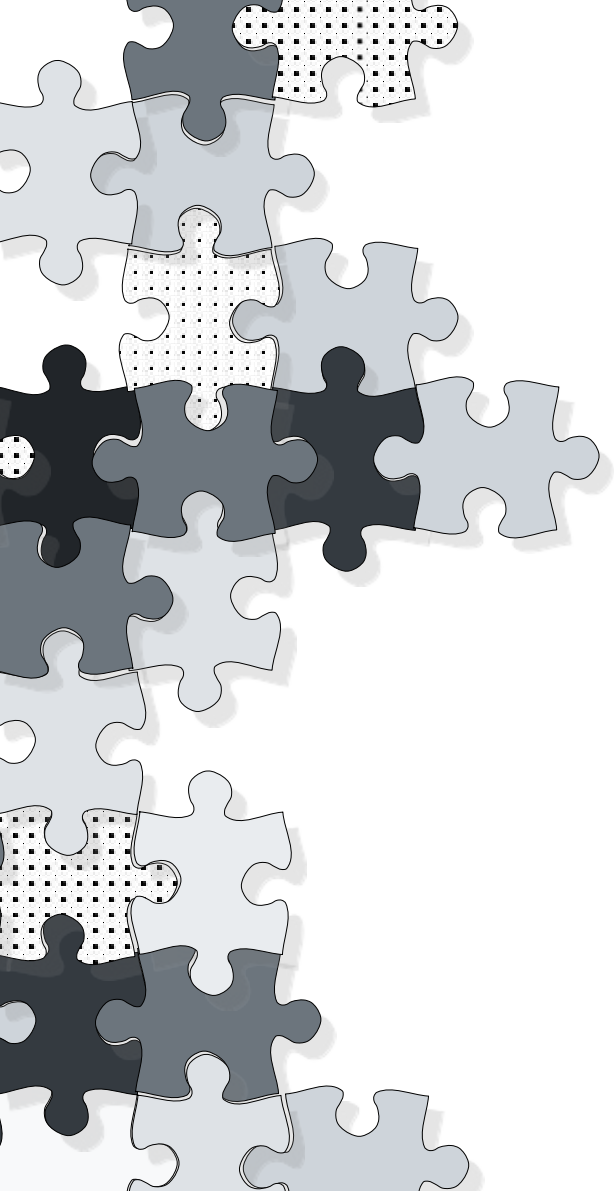


Extensible

Una particularidad de Poetry, es la posibilidad de extender sus funciones mediante plugins, propios o de terceros.

Resolutivo

Poetry siempre buscará una solución fácil para resolver las dependencias de nuestro paquete.



> If the implementation is hard to explain, it's a bad idea.
> If the implementation is easy to explain, it may be a good idea.
>
> - Zen de Python

De cero a Producción

Un repaso por las pautas y la configuración

01 Inicializa

Inicia un nuevo proyecto de Poetry, generando todos los directorios necesarios.

02 Instala dependencias

Añade los paquetes extras que vas a necesitar, por ejemplo, Django y Django REST Framework

03 Configura

Establece dependencias solo para desarrollo, test, etc, así como otros parámetros.

04 Añade tu código

Una vez llegados a este punto, podemos empezar a trabajar :-)

Inicialización y dependencias

```
→  MonkeyIslandAPI git:(feat/poetry) ✗ poetry init

This command will guide you through creating your pyproject.toml config.

Package name [monkeyislandapi]:
Version [0.1.0]:
Description []:  Monkey Island REST API for development, testing and learning purposes.
Author [Juanjo Salvador <juanjosalvador@netc.eu>, n to skip]:
License []:  MIT
Compatible Python versions [^3.11]:  3.9
```

```
Would you like to define your main dependencies interactively? (yes/no) [yes] yes
You can specify a package in the following forms:
  - A single name (requests): this will search for matches on PyPI
  - A name and a constraint (requests@^2.23.0)
  - A git url (git+https://github.com/python-poetry/poetry.git)
  - A git url with a revision (git+https://github.com/python-poetry/poetry.git#develop)
  - A file path (../my-package/my-package.whl)
  - A directory (../my-package/)
  - A url (https://example.com/packages/my-package-0.1.0.tar.gz)

Package to add or search for (leave blank to skip): Django
Found 20 packages matching Django
Showing the first 10 matches

Enter package # to add, or the complete package name if it is not listed []:
 [ 0] Django
 [ 1] django-503
 [ 2] django-filebrowser-django13
 [ 3] django-tracking-analyzer-django2
 [ 4] django-jchart-django3-uvm
 [ 5] django-totalsum-admin-django3
 [ 6] django-debug-toolbar-django13
 [ 7] django-suit-redactor-django2
 [ 8] django-django_csv_exports
 [ 9] django-thumborize
 [ 10]
 > 0
Enter the version constraint to require (or leave blank to use the latest version):
Using version ^4.2.2 for Django
```

pyproject.toml

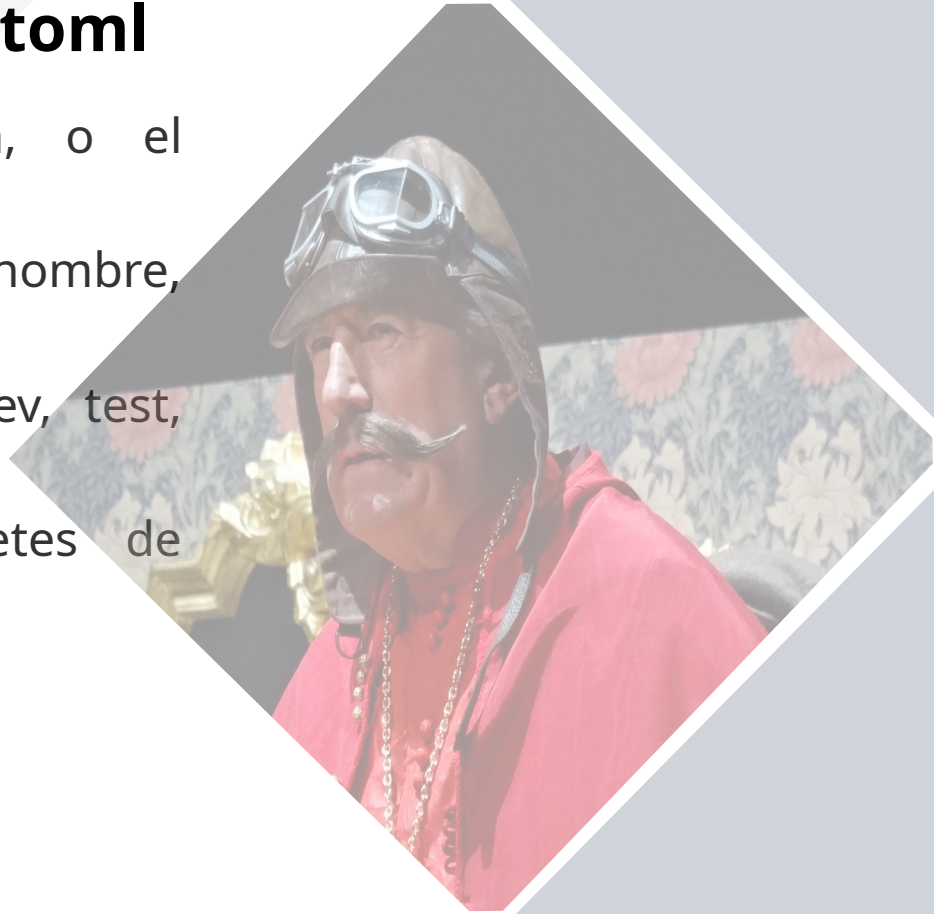
```
[tool.poetry]
name = "monkeyislandapi"
version = "0.1.0"
description = "Monkey Island REST API for development, testing and learning purposes."
authors = ["Juanjo Salvador <juanjosalvador@netc.eu>"]
license = "MIT"
readme = "README.md"

[tool.poetry.dependencies]
python = "3.9"
Django = "^4.2.2"
django-rest-framework = "^0.1.0"


[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Configuración del proyecto

Estableciendo algunas reglas en nuestro pyproject.toml

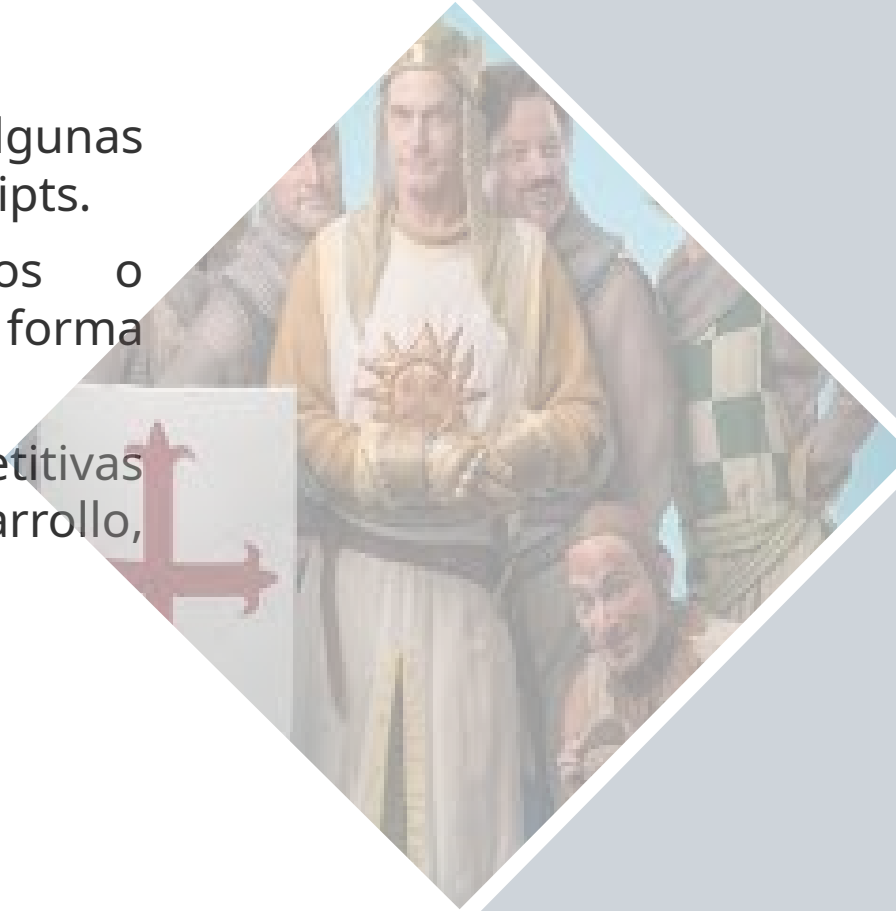
Entendiendo nuestro pyproject.toml

- Equivale al `package.json` de npm, o el `setup.py` de pip
- Definimos en él variables como el nombre, versión, descripción, licencia, etc
- Establece grupos de dependencias: dev, test, producción...
- Añade tu configuración para paquetes de terceros que lo requieran
- Añade tus propios scripts y comandos



Scripts personalizados

- Vercel ha incorporado en Poetry algunas funciones muy útiles de npm, como los scripts.
- Permite definir comandos complejos o demasiado largos para recordarlos, de forma concisa.
- Especialmente útil para tareas repetitivas (chequeos de estilo, test, servidor de desarrollo, ejecutar test...)



Extra: Migración desde pip

No te preocupes, no duele

Adaptar nuestro requirements.txt

- La sintaxis que utiliza Poetry es similar a la de pip

- Podemos añadir las dependencias copiando cada línea del requirements.txt, y realizando los siguientes cambios:

    1. == por =
    2. Entrecomillado de las versiones

Adaptar nuestro setup.py

- La mayoría de propiedades comunes se rellenan por defecto durante la creación del proyecto.
- Para exportar el resto, se declaran de la misma manera, pero con nombres equivalentes.
- La lista de equivalencias se encuentra en la propia documentación de Poetry

  https://python-poetry.org/docs/master/pyproject/

Preguntas y respuestas

Enlaces de interés

- Poetry - https://python-poetry.org
- Poetry Docs - https://python-poetry.org/docs/
- Diapositivas – https://bit.ly/OpenSouthCode_Poetry

- MonkeyIslandAPI – https://github.com/JuanjoSalvador/MonkeyIslandAPI

@Linuxneitor

@jsalvador@mastodon.social