El Administrador de Paquetes uv: Simplificando el Desarrollo Python

8 de julio de 2025

Introducción a `uv`

Un paso adelante en la gestión de paquetes Python

- La herramienta **uv** es un instalador de paquetes que ofrece un rendimiento superior a `pip`.
- Se describe como **entre 10 y 100 veces más rápido** que `pip`.
- Su diseño prioriza la **facilidad de uso**.
- Por defecto, `uv` **requiere** el uso de un entorno virtual para instalar paquetes.

La Necesidad de Entornos Virtuales

Resolviendo conflictos de dependencias

- **Problema central**: En el desarrollo de aplicaciones Python, es común que diferentes proyectos requieran **versiones distintas** de los mismos paquetes, o incluso **versiones diferentes de Python**.
- La instalación global de paquetes puede llevar a **conflictos de dependencias**, donde la actualización o instalación de un paquete para un proyecto rompe otro.
- Los **entornos virtuales** son la solución a estos problemas.
- Un entorno virtual es un **entorno independiente** creado sobre una instalación de Python existente.
- Cada entorno virtual tiene su **propio conjunto de paquetes Python** instalados, aislados del sistema principal.
- Son **desechables** y pueden ser fácilmente eliminados y recreados.

Velocidad y Facilidad de Uso de `uv`

Una alternativa eficiente a Pip

- `uv` se destaca por su **notable velocidad**, siendo significativamente más rápido que `pip`.
- Esto se traduce en **tiempos de resolución e instalación de paquetes mucho menores**, lo que mejora la productividad del desarrollador.
- Su interfaz es **intuitiva y sencilla**, lo que facilita su adopción y uso diario.
- A diferencia de `pip`, `uv` **obliga por defecto al uso de entornos virtuales**, lo que promueve una buena práctica de desarrollo desde el inicio.

Cómo Instalar uv

Múltiples opciones para diferentes sistemas operativos

- Para **Linux o Mac**, use `curl`:

  ```
  curl -LsSf https://astral.sh/uv/install.sh | sh
  ```

- Para **Mac**, también puede usar `Homebrew`:

  ```
  brew install uv
  ```

- Para **Windows**, use `powershell`:

  ```
  powershell -c "irm https://astral.sh/uv/install
  ```

- Alternativamente, puede instalar `uv` con `pip` (si ya lo tiene):

  ```
  pip install uv
  ```

- Para **verificar** la instalación, ejecute:

  ```
  uv --version
  ```

  Lo que debería mostrar una salida como `uv 0.1.44`.

Creación de Entornos Virtuales

Configurando su entorno de proyecto

- Primero, cree un directorio para su proyecto y navegue hasta él:

```
mkdir my\_uv\_project
cd my\_uv\_project
```

- Para crear un entorno virtual con un nombre por defecto (`.venv`), ejecute:

```
uv venv
```

- Para especificar un nombre para su entorno virtual (ej., `my_env`):

```
uv venv my\_env
```

- Para crear un entorno virtual con una **versión específica de python** (ej., Python 3.11), utilice la opción `--python`:

```
uv venv 3rd\_env --python 3.11
```

Esto requiere que la versión de python esté disponible en el sistema.

Activación de un Entorno Virtual

Habilitando su entorno de proyecto

- Una vez creado el entorno virtual, debe **activarlo** para que sus paquetes estén disponibles y se aíslen del sistema.
- Para activar el entorno (usando shells compatibles con POSIX como `sh`, `bash`, o `zsh`):

```
source my\_env/bin/activate
```

- Si usa otro shell (ej., `cmd` en Windows), los scripts de activación serán diferentes (ej., `my_env\Scripts\activate.bat`).
- Para verificar que el entorno no contiene paquetes preinstalados, ejecute:

```
uv pip list
```

La salida no listará nada, confirmando el aislamiento.

Instalación de Paquetes en el Entorno Virtual

Gestionando dependencias de forma aislada

- Una vez activado, puede instalar paquetes usando `uv pip install`:

```
uv pip install pandas
```

- Para verificar la instalación de paquetes y sus dependencias, utilice:

```
uv pip list
```

- Esto demuestra el **aislamiento**: si instala `pandas==2.2.2` en un entorno y `pandas==2.1.0` en otro, cada entorno mantendrá su versión específica sin conflictos.

Cambio y Eliminación de Entornos Virtuales

Flexibilidad en la gestión de proyectos

- Para **desactivar** un entorno virtual:

```
deactivate
```

- Para **cambiar** a otro entorno, primero desactive el actual y luego active el nuevo.
- Los entornos virtuales se crean como **directorios regulares** dentro de su proyecto.
- Para **eliminar** un entorno virtual, simplemente borre su directorio:

```
rm -rf my\_second\_env
```

Limitaciones de uv

¿Cuándo buscar otras herramientas?

- `uv` está diseñado principalmente para gestionar **dependencias relacionadas con python**.
- No está pensado para instalar **paquetes a nivel de sistema** (ej., `libpq-dev` para PostgreSQL).
- `uv` **carece de capacidades de contenerización**.
- **No tiene un mecanismo de caché** para reutilizar dependencias idénticas en diferentes entornos virtuales del mismo proyecto, lo que puede ser ineficiente en proyectos grandes.

Conclusión

Simplificando el flujo de trabajo python

- `uv` es una herramienta **rápida y fácil de usar** para la gestión de dependencias y entornos virtuales en python.
- Los entornos virtuales son **esenciales** para evitar conflictos y asegurar la consistencia entre proyectos.
- Para proyectos más grandes o aquellos que requieren gestión de dependencias a nivel de sistema o capacidades de contenerización, herramientas como **Earthly** pueden ser una solución más completa.
- Earthly ofrece características similares a Docker, como **contenerización, caché** y la capacidad de instalar **dependencias a nivel de sistema** junto con las de python, optimizando el proceso de construcción.