**Pipenv**

* **What is a Package Manager and why should you use it?**

A package manager is a programming language’s tool to create project environments and easily import external dependencies. You don’t have to reinvent the wheel and are able to make the most of the tools at your disposal. You can simply re-use available packages outside of your built-in scope. When working on a project or library, you may “package” your project and publish it for others.  
  
Examples of Package Managers  
Python - pipenv, pip  
Java - maven, gradle  
JavaScript - npm, yarn  
PHP - composer  
Ruby - bundler  
  
**Pipenv**

* What is Pipenv

pipenv is just a package management tool for Python. It aims to bring the best of all packaging worlds (bundler, composer, npm, cargo, yarn, etc.) to the Python world.  
  
<https://dev.to/yukinagae/your-first-guide-to-getting-started-with-pipenv-50bn>

* Why you should use it

It automatically creates and manages a virtualenv for your projects, as well as adds/removes packages from your Pipfile as you install/uninstall packages. It also generates the ever-important Pipfile.lock, which is used to produce deterministic builds.  
  
Pipenv is primarily meant to provide users and developers of applications with an easy method to setup a working environment.   
Pipenv ships with package management and virtual environment support, so you can use one tool to install, uninstall, track, and document your dependencies and to create, use, and organize your virtual environments. When you start a project with it, Pipenv will automatically create a virtual environment for that project if you aren't already using one.  
  
The problems that Pipenv seeks to solve are multi-faceted:

1. You no longer need to use pip and virtualenv separately. They work together.
2. 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.
3. Hashes are used everywhere, always. Security. Automatically expose security vulnerabilities.
4. Strongly encourage the use of the latest versions of dependencies to minimize security risks arising from outdated components.
5. Give you insight into your dependency graph (e.g. $ pipenv graph).
6. Streamline development workflow by loading .env files.

<https://opensource.com/article/18/2/why-python-devs-should-use-pipenv>

* What are some best practices
* What are the commands

To install pipenv

pip install pipenv

To install packages to use in your project(eg django), change into the directory for your project.

pipenv install django

To generate a **Pipfile.lock** file, run:

pipenv lock

You can also run Python scripts with Pipenv. To run a top-level Python script called **hello.py**, run:

pipenv run python hello.py

To start a shell (also activates the virtual environment), run:

pipenv shell

If you would like to convert a project that currently uses a **requirements.txt** file to use Pipenv, install Pipenv and run:

pipenv install requirements.txt

To check your current directory

pipenv --where

Videos  
<https://www.youtube.com/watch?v=tRmmjlVHzno>  
<https://www.youtube.com/watch?v=6Qmnh5C4Pmo>