Pipenv: Python Dev Workflow for Humans

**Pipenv** is a tool that aims to bring the best of all packaging worlds (bundler, composer, npm, cargo, yarn, etc.) to the Python world. Windows is a first-class citizen, in our world.

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. For the distinction between libraries and applications and the usage of setup.py vs Pipfile to define dependencies, see [☤ Pipfile vs setup.py](https://pipenv.readthedocs.io/en/latest/advanced/#pipfile-vs-setuppy).

The problems that Pipenv seeks to solve are multi-faceted:

* You no longer need to use pip and virtualenv separately. They work together.
* Managing a requirements.txt file [can be problematic](https://www.kennethreitz.org/essays/a-better-pip-workflow), so Pipenv uses Pipfile and Pipfile.lock to separate abstract dependency declarations from the last tested combination.
* Hashes are used everywhere, always. Security. Automatically expose security vulnerabilities.
* Strongly encourage the use of the latest versions of dependencies to minimize security risks [arising from outdated components](https://www.owasp.org/index.php/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities).
* Give you insight into your dependency graph (e.g. $ pipenv graph).
* Streamline development workflow by loading .env files.

You can quickly play with Pipenv right in your browser:

## **Install Pipenv Today!**

sudo dnf install pipenv

## **Pipenv Features**

* Enables truly deterministic builds, while easily specifying only what you want.
* Generates and checks file hashes for locked dependencies.
* Automatically install required Pythons, if pyenv is available.
* Automatically finds your project home, recursively, by looking for a Pipfile.
* Automatically generates a Pipfile, if one doesn’t exist.
* Automatically creates a virtualenv in a standard location.
* Automatically adds/removes packages to a Pipfile when they are un/installed.
* Automatically loads .env files, if they exist.

The main commands are install, uninstall, and lock, which generates a Pipfile.lock. These are intended to replace $ pip install usage, as well as manual virtualenv management (to activate a virtualenv, run $ pipenv shell).

### **Basic Concepts**

* A virtualenv will automatically be created, when one doesn’t exist.
* When no parameters are passed to install, all packages [packages] specified will be installed.
* To initialize a Python 3 virtual environment, run $ pipenv --three.
* To initialize a Python 2 virtual environment, run $ pipenv --two.
* Otherwise, whatever virtualenv defaults to will be the default.

### **Other Commands**

* graph will show you a dependency graph of your installed dependencies.
* shell will spawn a shell with the virtualenv activated. This shell can be deactivated by using exit.
* run will run a given command from the virtualenv, with any arguments forwarded (e.g. $ pipenv runpython or $ pipenv run pip freeze).
* check checks for security vulnerabilities and asserts that PEP 508 requirements are being met by the current environment.