1. What is Pipenv?

Pipenv is a tool that aims to bring the best of all packaging worlds to the Python world. It automatically creates and manage a virtualenv for our projects, as well as add/removes packages from our Pipfile as our install/uninstall packages.

Pipenv is primarily meant to provide users and developers of application with an easy method to setup a working environment.

1.1 What is Pip?

A packages contains all the files we need for a module. And modules are Python code libraries we can include in our project. **PIP** is a package manager for Python packages, or modules if we like.

1.2 What is the Virtual Environment?

A **virtual environment** is a Python environment such that the Python interpreter, libraries and scripts installed into it are is isolated from those installed in other virtual environments. A virtual environment is a directory tree which contains Python executable files and other files which indicate that is a virtual environment.

2. How to use Pipenv?

Step 1 : Install pipenv packages

```
CLI
$ pip install pipenv
```

Step 2 : Move to designated folder

```
CLI
$ cd my_practice/
```

Step 3 : Creating virtual environment and installing packages

When we use the pipenv command, it first checks whether a virtual environment exists in the Project. If the virtual environment is not installed, then the virtual environment will be installed in the project.

```
CLI
$ pipenv install requests

Creating a virtualenv for this project...
Pipfile: /home/users/my_practice/Pipfile
Using /usr/bin/python3.8 (3.8.10) to create virtualenv...
  Creating virtual environment...

✔ Successfully created virtual environment!
Virtualenv location: /home/users/.local/share/virtualenvs/my_practice-qrBNbtx-
Creating a Pipfile for this project...
Installing requests...

Pipfile.lock not found, creating...
Locking [packages] dependencies...
Building requirements...
Resolving dependencies...
✔ Success!
```

Pipfile

Pipfile basically just tells us which packages we have installed and what version of that packages we're using.

The [package] section specify what we install with its version. If we didn't specify a specific version of package, then it will be stored in package = "*" format. However, this can also be risky because we don't want to take any changes of updates breaking our project(This will be protected by Pipfile.lock).

- source : Denote where we're downloading packages from
- packages : Installed packages = Version(* if we didn't specify a specific version.)
- dev-packages : Specify installed packages for development requirements
- requires : Specify which version of Python that we're using. We can change version by changing to wanted version.

Pipfile.lock

Pipfile.lock is another file that we're not supposed to touch. It's just a file that gets generated and produces deterministic builds. This file is a generated file that has more detailed information about our current environment.

This file contains the dependencies that are installed when we installed the library. And also it has the exact version of each of these packages that were installed.

[illegible]

The dev-packages is for development requirements. So stuff like linters, unit test libraries, etc. All that is not needed on the user's machine.

To install a package as dev-requirement add -d or --dev to install command.

```
CLI
$ pipenv install pytest -d

Installing pytest...
Adding pytest to Pipfile's [dev-packages]...
✔ Installation Succeeded
Pipfile.lock (fbd99e) out of date, updating to (9e60ab)...
Locking [packages] dependencies...
Building requirements...
Resolving dependencies...
✔ Success!
Locking [dev-packages] dependencies...
Building requirements...
Resolving dependencies...
✔ Success!
Updated Pipfile.lock (bb86d5e8cd294cb34e55b4d26f021d673efacad14a065a789827a38d599e60a
b)!
Installing dependencies from Pipfile.lock (9e60ab)...
Installing dependencies from Pipfile.lock (9e60ab)...
To activate this project's virtualenv, run pipenv shell.
```

Step 4 : Activate virtual environment

```
CLI
# Activate
$ pipenv shell

# Deactivate
$ exit
```

Step 5 : Run the project file

```
CLI
# Way1
$ pipenv shell
$ python script.py

# Way2
$ pipenv run python script.py
```

Step 6 : Removing virtual environments and packages

```
CLI
# Removing installed packages
$ pipenv uninstall pytest

# Removing virtual environment
$ pipenv --rm
```

3. Pipenv features

- **pipenv --venv** : pipenv --venv returns the location of virtual environment directory.
- **pipenv graph** : pipenv graph is only showing the graph of development dependencies, and leaving out the regular dependencies.
- **pipenv check** : Check security vulnerabilites.
- **pipenv lock** : Set lockfile - before deployment.

Source from :

- https://pipenv.pypa.io/en/latest/
- https://docs.python.org/3/library/venv.html#venv-def