Pipenv Python env and dependencies

CONCEPTS

> Commit both Pipfile and Pipfile.lock; the lock file makes installs repeatable.

Pipfile what you want
↓ `pipenv lock`
Pipfile.lock exact pins + hashes
↓ `pipenv sync`
Virtualenv installed packages
↓ `pipenv run`
Your command runs in the env

> Pipenv finds your project by looking for the Pipfile in the current or a parent folder.

INSTALL AND REMOVE

`pipenv install`
Install everything from Pipfile

`pipenv install <pkg>`
Add a package and lock

`pipenv install <pkg>==<ver>`
Add with a version pin

`pipenv install --dev <pkg>`
Add a development package

`pipenv install -r requirements.txt`
Import a requirements file

`pipenv uninstall <pkg>`
Remove a package

`pipenv clean`
Uninstall packages not in the lock

LOCK AND SYNC

`pipenv lock`
Resolve and write Pipfile.lock

`pipenv lock --clear`
Lock with a fresh resolver cache

`pipenv sync`
Install exactly what the lock says

`pipenv sync --dev`
Sync including dev packages

`pipenv install --deploy`
Fail if the lock is out of date

`pipenv install --ignore-pipfile`
Install from the lock only

`pipenv verify`
Check the lock matches the Pipfile

RUN AND ACTIVATE

`pipenv run <cmd>`
Run a command in the env

`pipenv run python <file>`
Run a script with the env's Python

`pipenv run <script> [args]`
Run a Pipfile [scripts] shortcut

`pipenv shell`
Spawn a shell with the env active

`pipenv scripts`
List the Pipfile shortcuts

INSPECT

`pipenv graph`
Show the dependency tree

`pipenv graph --reverse`
Who depends on each package?

`pipenv graph --json-tree`
Dependency tree as JSON

`pipenv update`
Update and re-lock all packages

`pipenv update <pkg>`
Update one package and its deps

`pipenv update --outdated`
List what would change

`pipenv check`
Scan packages for known vulnerabilities

ENVIRONMENT

`pipenv --venv`
Print the virtualenv path

`pipenv --py`
Print the env's Python path

`pipenv --where`
Print the project folder

`pipenv --python <ver>`
Create the env with a Python version

`pipenv --rm`
Delete the virtualenv (deprecated)

`PIPENV_VENV_IN_PROJECT=1`
Keep the env in ./.venv

EXPORT

`pipenv requirements`
Print requirements.txt from the lock

EXPORT (cont.)

`pipenv requirements --dev`
Include dev packages

`pipenv requirements --dev-only`
Only dev packages

`pipenv requirements --hash`
Include package hashes

PIPFILE

Minimal Pipfile

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
requests = "*"

[dev-packages]
pytest = "*"

[requires]
python_version = "3.12"
```

Script shortcuts

```
[scripts]
start = "python app.py"
test = {cmd = "pytest -q"}
serve = {call =
"app:main('dev')"}
```

> A .env file in the project is loaded automatically by pipenv run and pipenv shell.

RECIPES

New project

```
mkdir app && cd app
pipenv --python 3.12
pipenv install requests
pipenv install --dev pytest
pipenv run pytest
```

Convert a requirements.txt project

```
pipenv install -r
requirements.txt
pipenv lock
pipenv graph
```

CI or Docker build

```
pip install pipenv
pipenv install --deploy --ignore-
pipfile
pipenv run python -m app
```

Export for tools that need requirements.txt

```
pipenv requirements >
requirements.txt
pipenv requirements --dev-only >
requirements-dev.txt
```