Pipenv Python env and dependencies

START HERE

`pipenv install`
Create env, install from Pipfile

`pipenv install <pkg>`
Add a package

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

`pipenv run <cmd>`
Run inside the env

`pipenv shell`
Activate the env

`pipenv lock`
Write Pipfile.lock

`pipenv graph`
Show dependency tree

`pipenv --venv`
Where is the env?

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

CONCEPTS

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

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

Prefer pipenv run over activating: one command, no shell state to forget.

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 --categories <name> <pkg>`
Add to a custom category

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

`pipenv install --skip-lock`
Install without re-locking

`pipenv install --dry-run`
Show what would be installed

INSTALL AND REMOVE (cont.)

`pipenv uninstall <pkg>`
Remove a package

`pipenv uninstall --all-dev`
Remove all dev packages

`pipenv uninstall --all`
Remove every 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 lock --pre`
Allow pre-releases when locking

`pipenv lock --keep-outdated`
Lock without bumping existing pins

`pipenv sync`
Install exactly what the lock says
Never re-locks; use it in CI and on servers.

`pipenv sync --dev`
Sync including dev packages

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Fail if the lock is out of date

LOCK AND SYNC (cont.)

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

`pipenv verify`
Check the lock matches the Pipfile

`pipenv install --system --deploy`
Install the lock into system Python

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
Leave it with exit; there is no deactivate for this subshell.

`pipenv scripts`
List the Pipfile shortcuts

`pipenv open <module>`
Open a module in your editor

INSPECT

`pipenv graph`
Show the dependency tree

INSPECT (cont.)

`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 update --dry-run`
Preview an update

`pipenv upgrade <pkg>`
Re-lock one package only

`pipenv check`
Scan packages for known vulnerabilities
Deprecated in favour of pipenv audit; check the installed version.

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

ENVIRONMENT (cont.)

`pipenv --rm`
Delete the virtualenv (deprecated)
Newer versions prefer pipenv remove.

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

`PIPENV_PIPFILE=<path>`
Use a Pipfile elsewhere

EXPORT

`pipenv requirements`
Print requirements.txt from the lock

`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"
```

PIPFILE (cont.)

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.

Pipenv will offer to install a missing [requires] Python if pyenv or asdf is present.

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
```