



pip installs packages that other people wrote. A virtual environment (venv) is a separate folder with its own Python packages. Each project gets the package versions it needs, and one project cannot break another. Type the commands in a terminal, inside your project folder.

CREATE A VENV

Make one venv per project. .venv is the usual folder name, and code editors find a folder with that name on their own.

python -m venv .venv

makes a folder named .venv with its own Python and pip

py -m venv .venv

the same on Windows, if 'python' is not found

python3 -m venv .venv

the same on macOS and Linux

WHAT IS INSIDE .VENV

.venv\Scripts\python.exe

the venv's Python on Windows (.venv/bin/python on macOS and Linux)

.venv\Scripts\python.exe app.py

runs app.py with the venv's Python: no need to turn the venv on first

.venv\Lib\site-packages

where packages go on Windows (lib/python3.x/site-packages on macOS and Linux)

pyenv.cfg

a small text file that records which Python made the venv

TURN IT ON AND OFF

Turning a venv on is called activating it. While it is on, the prompt starts with (.venv), and python and pip mean the ones inside .venv. Activate it again in every new terminal.

.venv\Scripts\Activate.ps1

Windows PowerShell

.venv\Scripts\activate.bat

Windows cmd

source .venv/bin/activate

macOS and Linux

deactivate

turns the venv off (closing the terminal also does this)

INSTALL AND REMOVE

python -m pip install requests

installs the newest version

python -m pip install "requests==2.32.3"

installs one exact version

python -m pip install "pandas>=2.0"

version 2.0 or newer; the quotes stop the shell from treating > as 'write to a file'

python -m pip install --upgrade requests

updates requests to the newest version

python -m pip uninstall requests

removes requests (it asks y/n; add -y to skip the question)

VERSION RULES

Use these in pip install and in requirements.txt, one package per line.

"requests==2.32.3"

exactly this version

"requests>=2.31,<3"

2.31 or newer, but below 3

"requests~2.31"

2.31 or any newer 2.x version, never 3.0

"requests!=2.30.0"

any version except 2.30.0

KEEP PIP HEALTHY

python -m pip install --upgrade pip

upgrade pip itself; a new venv often has an old pip

python -m pip check

lists packages whose required packages are missing or the wrong version

python -m pip install -e .

installs your own project; your code changes then work without reinstalling

WHY PYTHON -M PIP

The pip command on its own can belong to a different Python install than the one python starts.

python -m pip install requests

runs the pip that belongs to this exact python, so the package goes where your code will look for it

python -m pip --version

shows which Python folder pip installs into

SEE WHAT IS INSTALLED

Commit requirements.txt to git. Anyone can then rebuild the same environment.

python -m pip list

lists every package and its version

python -m pip show requests

shows the version, the folder and the packages it needs

python -m pip freeze > requirements.txt

saves the exact versions to a file for other people

python -m pip install -r requirements.txt

installs every package listed in requirements.txt

requests==2.32.3

pandas==2.2.3

what requirements.txt looks like: one package per line

A NEW PROJECT, START TO FINISH

python -m venv .venv

source .venv/bin/activate

python -m pip install requests

python -m pip freeze > requirements.txt

create, turn on (on Windows, use the Activate line from above), install, save the list

git clone <project>

python -m venv .venv

source .venv/bin/activate

python -m pip install -r requirements.txt

someone else's project: make your own venv, then install from their list

WHICH PYTHON IS RUNNING? (FROM PYTHON)

import sys

import importlib.metadata

print(sys.prefix != sys.base_prefix)

True inside a virtual environment

print(sys.executable)

the full path of the Python running this code

print(importlib.metadata.version("pip"))

the installed version of a package, as a string

importlib.metadata.version("no-such-pkg")

→ raises PackageNotFoundError

raised when the package is not installed in this Python

FROM THE TERMINAL

where python

Windows cmd: lists every python it finds; the first one listed is the one that runs

which python

macOS and Linux: shows the python that runs

DELETE AND START AGAIN

Deactivate first. A venv holds nothing you wrote, so you can always rebuild it: create, activate, then pip install -r requirements.txt.

Remove-Item -Recurse -Force .venv

Windows PowerShell

rmdir /s /q .venv

Windows cmd

rm -rf .venv

macOS and Linux

OTHER TOOLS YOU MAY SEE

uv venv

uv pip install requests

uv: a faster tool that does the same jobs as venv and pip

pipx install black

pipx installs command-line tools; each tool gets its own hidden venv

WATCH OUT

pip install requests

before activating: the package goes into the main Python, and your project then fails with ModuleNotFoundError

git add .venv

never commit .venv: add .venv/ to .gitignore and commit requirements.txt

error: externally-managed-environment

macOS and Linux protect the system Python: create a venv and install there

sudo pip install requests

changes files the system needs; use a venv instead

pip install x

python app.py

pip and python can come from two different Python installs. Always type python -m pip

running scripts is disabled

PowerShell blocks Activate.ps1. Run this once, then try again: Set-ExecutionPolicy -Scope CurrentUser RemoteSigned

mv .venv env

a venv stops working after a move or rename: delete it and create it again