SIDEBAR Programming

Python Environments Cheat Sheet

Sidebar

Programming · Python Environments Cheat Sheet

Creating, activating, and safely tearing down a Python virtual environment — using the built-in venv module, no extra tools required.

WHY BOTHER WITH ONE AT ALL?

`Isolation`	Each project gets its own private set of installed packages
`No version clashes`	Project A can use Django 4 while Project B uses Django 5
`A clean system Python`	Nothing you install for a project can break other tools on your machine
`Reproducibility`	A pinned package list lets anyone rebuild the exact same setup

CREATING ONE

`python -m venv myenv`	Creates a folder named `myenv` holding its own interpreter
`python3 -m venv myenv`	Use this instead on macOS/Linux if `python` isn't aliased
`Where to put it`	Inside the project folder itself — commonly named `venv` or `.venv`

ACTIVATING

`myenv\Scripts\activate.bat`	Windows, Command Prompt
`myenv\Scripts\Activate.ps1`	Windows, PowerShell (see Troubleshooting if this fails)
`source myenv/bin/activate`	macOS / Linux, bash or zsh
`(myenv)`	The prompt prefix that confirms activation actually worked

DEACTIVATING

`deactivate`	The exact same command on every platform — no arguments needed
`where python`	/
`which python`	Confirms which interpreter is actually active right now

INSTALLING & FREEZING PACKAGES

`pip install requests`	Installs only into the currently active environment
`pip freeze > requirements.txt`	Snapshots exact installed versions
`pip install -r requirements.txt`	Recreates that exact same package set elsewhere
`python -m pip install --upgrade pip`	A new venv often ships with an outdated bundled pip

SAFELY DELETING A VENV

`deactivate`	Step 1 — leave it before deleting it
`rmdir /s /q myenv`	Windows, Command Prompt
`Remove-Item -Recurse -Force myenv`	Windows, PowerShell
`rm -rf myenv`	macOS / Linux
`It's just a folder`	Deleting it never touches your actual project source files

TROUBLESHOOTING

`"running scripts is disabled"`	PowerShell only — see the fix below
`Packages installing "globally"`	You forgot to activate — check for the `(myenv)` prefix
`Wrong Python version inside it`	Delete it and recreate with the correct interpreter, e.g. `py -3.11 -m venv myenv`
`Old/broken pip errors`	`python -m pip install --upgrade pip` while activated

GOOD HABITS

`.gitignore`	Add your venv folder name — never commit it to version control
`requirements.txt`	Commit this instead, so the environment itself is always reproducible
`One venv per project`	Don't reuse a single environment across unrelated projects

CREATED IT IN THE WRONG FOLDER, OR WITH THE WRONG NAME? DELETE IT — DON'T TRY TO MOVE OR RENAME IT

A venv's own activation scripts and its `pyvenv.cfg` file have the *absolute path* to where it was created baked directly into them. Move the folder or rename it and those paths silently stop pointing anywhere real — activation can break in ways that are confusing to debug. Python's own official documentation is explicit about this: if you need a venv somewhere else, delete the misplaced one and run `python -m venv` again in the right location under the right name. It only takes a few seconds to recreate, and deleting it is completely safe — a venv holds nothing but installed packages and a copy of the interpreter, never your actual project code.

FIXING THE POWERSHELL "RUNNING SCRIPTS IS DISABLED" ERROR

Run `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser` once, in PowerShell. This only relaxes the policy for locally-created scripts under your own user account — it doesn't touch the whole system, and it isn't Python-specific. Using Command Prompt instead of PowerShell sidesteps the issue entirely, since cmd.exe doesn't block script execution this way.