pip Python package installer

START HERE

`python -m venv .venv`
Make an environment

`pip list`
What is installed?

`source .venv/bin/activate`
Activate it

`pip freeze > requirements.txt`
Pin versions

`pip install <package>`
Install a package

`pip install -U <package>`
Upgrade a package

`pip install -r requirements.txt`
Install a list

`pip uninstall <package>`
Remove a package

Index PyPI or mirror → `resolve` → **Wheel / sdist** cached download → `build + unpack` → **site-packages** active environment → `freeze` → **requirements.txt** pinned list

CONCEPTS

Use python -m pip so pip always belongs to the interpreter you mean.

Install into a virtual environment; on system Pythons marked EXTERNALLY-MANAGED pip refuses.

Set PIP_REQUIRE_VIRTUALENV=1 (or --require-virtualenv) to block accidental global installs.

INSTALL

`pip install <package>`
Install the latest version

`pip install <package>==<version>`
Install an exact version

`pip install "<package>≥1.2,<2"`
Install within a range
Quote the specifier or the shell treats > as a redirect.

`pip install -r <file>`
Install from a requirements file

`pip install -U <package>`
Upgrade to the newest version

`pip install -e <path>`
Editable install of a local project

`pip install "<package>[<extra>]"`
Install with optional extras

`pip install <path or url>`
Install a wheel, archive or directory

`pip install git+<url>@<ref>`
Install from a Git repository

`pip install --no-deps <package>`
Skip dependencies

`pip install --dry-run <package>`
Show what would be installed

`pip install --pre <package>`
Allow pre-release versions

`pip install --force-reinstall <package>`
Reinstall even if present

`pip install --user <package>`
Install into the user site

`pip install -t <dir> <package>`
Install into a target directory

INSTALL (cont.)

`pip install --only-binary :all: <package>`
Refuse source builds

`pip install --no-build-isolation <package>`
Build with your own environment

`pip install --break-system-packages <package>`
Override EXTERNALLY-MANAGED
Can break OS-managed Python; prefer a venv.

INSPECT

`pip list`
List installed packages

`pip list --outdated`
Show packages with newer versions

`pip show <package>`
Show version, location, dependents

`pip show -f <package>`
Also list installed files

`pip freeze`
Print installed packages as pins

`pip list --not-required`
Only top-level packages

`pip list --format json`
Machine-readable list

`pip list --editable`
Only editable installs

`pip check`
Verify dependencies are consistent

`pip index versions <package>`
List versions on the index

`pip inspect`
Dump the environment as JSON

`pip debug`
Show tags and platform info

`pip hash <file>`
Compute an archive hash

REMOVE

`pip uninstall <package>`
Remove a package

`pip uninstall -y <package>`
Remove without confirmation

`pip uninstall -r <file>`
Remove everything in a file

REMOVE (cont.)

`pip freeze | xargs pip uninstall -y`
Wipe the environment
Fails on editable installs; recreating the venv is cleaner.

REQUIREMENTS

`pip freeze > requirements.txt`
Snapshot the environment

`pip install -r requirements.txt`
Restore a snapshot

`pip install -c constraints.txt -r <file>`
Cap versions without adding packages

`pip freeze --exclude-editable`
Freeze without editable installs

`pip freeze -r <file>`
Freeze in the order of a file

`pip install --require-hashes -r <file>`
Verify each hash

`pip install --group <name>`
Install a pyproject dependency group
Needs pip 25.1 or newer.

`pip lock -r <file>`
Generate a lock file
Experimental: writes pylock.toml; may change or go.

In a file: -r other.txt includes another, -e . adds the project, -c sets constraints, # comments, ; marker after a pin.

Direct dependencies go in a hand-written file; use a constraints file for the full pinned set.

INDEXES AND DOWNLOAD

`pip install -i <url> <package>`
Use another index

`pip install --extra-index-url <url> <package>`
Add an index to PyPI
Mixing indexes risks dependency confusion; prefer one trusted index.

`pip install --no-index -f <dir> <package>`
Install offline from a directory

`pip download -d <dir> -r <file>`
Fetch packages without installing

INDEXES AND DOWNLOAD (cont.)

`pip wheel -w <dir> <package>`
Build wheels into a directory

`pip download --platform <tag> --only-binary :all: <package>`
Fetch wheels for another platform

CONFIG AND CACHE

`pip config list`
Show active configuration

`pip config debug`
Show config files and values

`pip config set global.index-url <url>`
Set a default index

`pip config unset <key>`
Remove a setting

`pip cache dir`
Print the cache location

`pip cache info`
Show cache size

`pip cache purge`
Empty the cache

`pip install --no-cache-dir <package>`
Skip the cache for one run

`pip --python <interpreter> <command>`
Manage another interpreter's packages

`pip --version`
Show pip and Python versions

`python -m pip install -U pip`
Upgrade pip itself

RECIPES

New project environment

```
python3 -m venv .venv
source .venv/bin/activate
python -m pip install -U pip
pip install -r requirements.txt
```

Review and update outdated packages

```
pip list --outdated
pip install -U <package>
pip check
pip freeze > requirements.txt
```

Offline install bundle

```
pip download -d wheelhouse -r
requirements.txt
# copy wheelhouse/ to the
target machine
pip install --no-index -f
wheelhouse -r requirements.txt
```

Develop a local package

```
pip install -e ".[dev]"
pip show <package>
pip uninstall <package>
```