Virtual Environments & Package Management

The Problem

```
Global Python (System)
├── pandas==1.5.0  ← Project A needs this
├── numpy==1.24.0
└── requests==2.28.0

Install pandas==2.0.0 for Project B
→ OVERWRITES 1.5.0
→ Project A breaks

Solution: Isolated environments per project
```

Why Virtual Environments?

- **Dependency Isolation**: Each project has own packages
- **Version Control**: Different versions per project
- **Reproducibility**: Same setup everywhere
- **No Global Pollution**: System Python stays clean

Tool Comparison

Tool	Speed	Lock Files	Auto venv	Best For
UV	Fast	Yes	Yes	2024+ (Recommended)
pip+venv	Slow	No	No	Legacy/Simple
Poetry	Medium	Yes	Yes	Publishing packages

Tool	Speed	Lock Files	Auto venv	Best For
Conda	Slow	Yes	Yes	Data science (multi-lang)
Pipenv	Very Slow	Yes	Yes	Avoid

UV (Recommended)

Installation

```
# Install UV
curl -LsSf https://astral.sh/uv/install.sh | sh
# or
pip install uv

# Verify
uv --version
```

Basic Workflow

```
# Create new project
uv init my-project
cd my-project

# Creates:
# ├── pyproject.toml
# ├── .python-version
# ├── README.md
# └── hello.py

# Add packages
uv add pandas numpy requests

# Add dev dependencies
uv add --dev pytest black
```

```
# Install all dependencies
uv sync

# Run scripts (auto-activates venv)
uv run python script.py
uv run pytest

# Update dependencies
uv lock --upgrade

# Remove package
uv remove pandas
```

Installing from requirements.txt

```
# Convert old project to UV
uv pip install -r requirements.txt
uv pip freeze > requirements.txt  # If needed
```

Key Files

```
pyproject.toml  → What you want (pandas>=2.0)
uv.lock         → What you get (pandas==2.0.3 + all subdeps)
.venv/          → Virtual environment (auto-created)
```

No Manual Activation Needed

```
# Old way (pip+venv)
source .venv/bin/activate  # Forget this
python script.py
```

```
# UV way
uv run python script.py    # Just works
```

pip + venv (Traditional)

```
# Create venv
python -m venv .venv

# Activate
source .venv/bin/activate       # Mac/Linux
.venv\Scripts\activate          # Windows

# Install packages
pip install pandas numpy

# Save dependencies
pip freeze > requirements.txt

# Install from requirements
pip install -r requirements.txt

# Deactivate
deactivate
```

Conda

```
# Create environment
conda create -n myenv python=3.10

# Activate
conda activate myenv

# Install packages
conda install pandas numpy scipy
```

```
# Export environment
conda env export > environment.yml

# Create from file
conda env create -f environment.yml

# Deactivate
conda deactivate

# List environments
conda env list

# Remove environment
conda env remove -n myenv
```

Poetry

```
# Install Poetry
curl -sSL https://install.python-poetry.org | python3 -

# Initialize project
poetry init

# Add packages
poetry add pandas
poetry add --group dev pytest

# Install all
poetry install

# Run scripts
poetry run python script.py

# Update lock file
```

```
poetry lock

# Export requirements
poetry export -f requirements.txt > requirements.txt
```

Decision Tree

```
Existing project?
├─ YES → Use what it uses
└─ NO → New project
   │
   ├─ Multi-language (Python + R)?
   │  └─ YES → Conda
   │
   ├─ Publishing to PyPI?
   │  └─ YES → Poetry or UV
   │
   ├─ Corporate (pip required)?
   │  └─ YES → pip + venv
   │
   └─ Everything else?
      └─ UV (fastest, modern)
```

Project Structure

```
my-project/
├── .venv/              # Virtual environment (auto)
├── pyproject.toml      # Dependencies + metadata
├── uv.lock             # Locked versions
├── .gitignore          # Ignore .venv/
├── README.md
├── src/
│   └── main.py
```

```
└── tests/
    └── test_main.py
```

.gitignore for Environments

```
# Virtual environments
.venv/
venv/
ENV/
env/

# Python
__pycache__/
*.pyc
*.pyo

# Packages
*.egg-info/
dist/
build/

# UV
.python-version

# Poetry
poetry.lock  # Commit this for apps, ignore for libraries
```

Common Workflows

Start New Project (UV)

```
uv init my-app
cd my-app
```

```
uv add pandas numpy
uv run python main.py
```

Clone Existing Project

```
git clone <repo>
cd repo

# If using UV
uv sync

# If using pip
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt

# If using Poetry
poetry install

# If using Conda
conda env create -f environment.yml
```

Add New Dependency

```
# UV
uv add requests

# pip
pip install requests
pip freeze > requirements.txt

# Poetry
poetry add requests
```

```
# Conda
conda install requests
```

VSCode Integration

Select Python Interpreter

```
Ctrl+Shift+P → "Python: Select Interpreter"
Choose: .venv/bin/python
```

Jupyter Kernel

```
# Install ipykernel in venv
uv add ipykernel

# Register kernel
uv run python -m ipykernel install --user --name=myproject

# In VSCode:
# Open .ipynb → Select Kernel → "myproject"
```

Multiple Terminals

```
# Each terminal can use different venv
# With UV, just use:
uv run python script.py  # Auto-activates correct venv
```

Lock Files Explained

```
pyproject.toml (Your Intent):
dependencies = ["pandas>=2.0"]

uv.lock (Reality):
```

```
pandas==2.0.3
├── numpy==1.24.4
├── python-dateutil==2.8.2
│   └── six==1.16.0
└── pytz==2023.3

Lock = Exact versions for reproducibility
```

Common Issues

Wrong Python Version

```
# UV: Specify version
uv venv --python 3.10

# pip: Use specific Python
python3.10 -m venv .venv

# Conda: Specify in create
conda create -n myenv python=3.10
```

Packages Not Found

```
# Check you're in venv
which python  # Should show .venv/bin/python

# UV: Auto-handles this
uv run python script.py
```

Multiple venvs Conflict

```
# Rule: One venv per project
# Don't activate multiple venvs
```

```
# With UV: No manual activation needed
```

Best Practices

- **One venv per project** - Never share between projects
- **Commit lock files** - uv.lock, poetry.lock (for apps)
- **Ignore .venv/** - Never commit virtual environment
- **Document dependencies** - requirements.txt or pyproject.toml
- **Pin versions in production** - Use lock files
- **Use UV for new projects** - Fastest, modern approach

Quick Reference

Task	UV	pip+venv	Poetry	Conda
Create	`uv init`	`python -m venv .venv`	`poetry init`	`conda create -n name`
Activate	Auto	`source .venv/bin/activate`	Auto	`conda activate name`
Add package	`uv add pkg`	`pip install pkg`	`poetry add pkg`	`conda install pkg`
Run script	`uv run python x.py`	`python x.py`	`poetry run python x.py`	`python x.py`
Lock deps	`uv lock`	`pip freeze > req.txt`	`poetry lock`	`conda env export`

Migration Paths

pip → UV

```
# In existing project
uv pip install -r requirements.txt
uv add <packages from requirements>
# Delete requirements.txt, use pyproject.toml
```

Conda → UV

```
# Export conda packages
conda list --export > packages.txt
# Install with UV
uv add <packages>
```

Tips

- Always use `uv run` - never manually activate with UV
- Check `which python` if packages not found
- Delete `.venv/` and recreate if corrupted
- Use `-dev` flag for dev-only dependencies
- Keep system Python clean - never install packages globally
- Read lock files to understand dependency tree