

Python uv & uvx Cheatsheet (with Conda Comparison)

What is uv?

uv is an extremely fast Python package and project manager, written in Rust. It's a drop-in replacement for pip, pip-tools, pipx, poetry, pyenv, virtualenv, and more.

uv vs Conda: Quick Comparison

Feature	Conda	uv
Speed	Slower (can take minutes)	10-100x faster (seconds)
Package Source	Conda-forge, defaults, PyPI	PyPI primarily
Non-Python Packages	Yes (R, Julia, system libs)	No (Python only)
Environment Management	Built-in	Built-in
Python Installation	Yes	Yes
Lock Files	environment.yml	uv.lock (automatic)
Disk Space	Larger (duplicates packages)	Smaller (global cache)
Scientific Packages	Optimized binaries	Standard PyPI wheels

When to Use Which?

- **Use Conda:** When you need non-Python packages, CUDA libraries, or specific scientific computing optimizations
- **Use uv:** For pure Python projects, faster workflows, modern web development, better dependency resolution

Installation

```
bash
```

On macOS and Linux

```
curl -Lsf https://astral.sh/uv/install.sh | sh
```

On Windows

```
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"
```

Using pip

```
pip install uv
```

Using Homebrew

```
brew install uv
```

Core uv Commands

Python Management

```
bash
```

Install Python

```
uv python install 3.12
```

```
uv python install 3.11.7
```

List available Python versions

```
uv python list
```

Find installed Python

```
uv python find
```

Pin Python version for project

```
uv python pin 3.12
```

Project Management

```
bash
```

```
# Initialize a new project
```

```
uv init my-project
```

```
cd my-project
```

```
# Initialize in current directory
```

```
uv init
```

```
# Create project with specific Python version
```

```
uv init --python 3.12
```

```
# Add dependencies
```

```
uv add requests pandas numpy
```

```
uv add "fastapi>=0.104"
```

```
# Add dev dependencies
```

```
uv add --dev pytest black ruff
```

```
# Remove dependencies
```

```
uv remove requests
```

```
# Sync dependencies (install from pyproject.toml)
```

```
uv sync
```

```
# Update dependencies
```

```
uv lock --upgrade
```

```
uv sync
```

Virtual Environment Management

```
bash
```

Create virtual environment

```
uv venv
```

Create with specific Python version

```
uv venv --python 3.12
```

Activate virtual environment

On Unix/macOS:

```
source .venv/bin/activate
```

On Windows:

```
.venv\Scripts\activate
```

Run command in virtual environment (without activating)

```
uv run python script.py
```

```
uv run pytest
```

Package Management (pip replacement)

```
bash
```

```
# Install packages
uv pip install requests numpy pandas
uv pip install -r requirements.txt

# Install in editable mode
uv pip install -e .

# List installed packages
uv pip list

# Show package info
uv pip show requests

# Freeze dependencies
uv pip freeze > requirements.txt

# Uninstall packages
uv pip uninstall requests

# Compile requirements
uv pip compile requirements.in -o requirements.txt
```

uvx Commands (pipx replacement)

uvx is for running and installing Python CLI tools in isolated environments.

Running Tools (Without Installation)

```
bash
```

Run tools without installing

uvx ruff check

uvx black --version

uvx mypy script.py

Run specific version

uvx ruff@0.1.0 check

Run from git

uvx --from git+https://github.com/user/repo tool-name

Installing Tools Globally

bash

Install tool globally

uv tool **install** ruff

uv tool **install** black

uv tool **install** pipenv

Install specific version

uv tool **install** ruff==0.1.0

List installed tools

uv tool list

Update tool

uv tool update ruff

Uninstall tool

uv tool uninstall ruff

Run installed tool

ruff check *# After installation, use directly*

Running Streamlit Apps with uv

Method 1: Direct Project Setup

bash

```
# Create new project
uv init streamlit-app
cd streamlit-app

# Add Streamlit as dependency
uv add streamlit pandas plotly

# Create your app
echo 'import streamlit as st
import pandas as pd

st.title("My Streamlit App")
st.write("Hello, World!")

# Add your code here
df = pd.DataFrame({"col1": [1, 2, 3], "col2": [4, 5, 6]})
st.dataframe(df)' > app.py

# Run the Streamlit app
uv run streamlit run app.py

# Or with specific port
uv run streamlit run app.py --server.port 8502
```

Method 2: Using Existing Project

```
bash
```



```
# In existing project directory
```

```
uv add streamlit
```

```
# Run your existing app.py
```

```
uv run streamlit run app.py
```

Method 3: Quick Run with uvx (Temporary)

```
bash
```

```
# Quick run without project setup
```

```
uvx --with streamlit --with pandas streamlit run app.py
```

```
# With multiple dependencies
```

```
uvx --with streamlit --with plotly --with numpy streamlit run dashboard.py
```

Method 4: Install Streamlit Globally

```
bash
```

```
# Install streamlit as a tool
```

```
uv tool install streamlit
```

```
# Then run directly
```

```
streamlit run app.py
```

Common Workflows

Starting a New Data Science Project

```
bash
```

```
# Initialize project
uv init data-analysis --python 3.12
cd data-analysis

# Add common dependencies
uv add pandas numpy matplotlib seaborn jupyter scikit-learn

# Add dev tools
uv add --dev black ruff pytest ipython

# Run Jupyter
uv run jupyter lab
```

Setting Up a Web API Project

```
bash

# Initialize
uv init my-api --python 3.11
cd my-api

# Add dependencies
uv add fastapi uvicorn[standard] sqlalchemy pydantic

# Add dev dependencies
uv add --dev pytest httpx black ruff

# Run the server
uv run uvicorn main:app --reload
```

CI/CD Integration

```
bash
```

```
# Install exact dependencies from lock file
```

```
uv sync --frozen
```

```
# Run tests
```

```
uv run pytest
```

```
# Run linting
```

```
uv run ruff check .
```

```
uv run black --check .
```

Advanced Features

Using Different Package Indexes

```
bash
```

```
# Add extra index URL
```

```
uv add torch --index-url https://download.pytorch.org/whl/cpu
```

```
# Use different index
```

```
uv pip install --index-url https://test.pypi.org/simple/ package-name
```

Working with Private Packages

```
bash
```

```
# Install from private git repo
```

```
uv add git+ssh://git@github.com/org/private-repo.git
```

```
# Install from private PyPI
```

```
uv add package-name --index-url https://private.pypi.org/simple/
```

Scripts in pyproject.toml

```
toml
```

```
# In pyproject.toml
```

```
[project.scripts]
```

```
my-cli = "my_package.cli:main"
```

```
[tool.uv.scripts]
```

```
dev = "uvicorn main:app --reload"
```

```
test = "pytest tests/ -v"
```

```
lint = ["ruff check .", "black --check ."]
```

```
bash
```

```
# Run scripts
```

```
uv run dev
```

```
uv run test
```

```
uv run lint
```

Environment Variables

```
bash
```

```
# Set UV cache directory  
export UV_CACHE_DIR=/path/to/cache  
  
# Disable cache  
export UV_NO_CACHE=1  
  
# Set concurrent downloads  
export UV_CONCURRENT_DOWNLOADS=10  
  
# Quiet mode  
export UV_QUIET=1
```

Tips & Best Practices

1. **Speed:** uv is 10-100x faster than pip for most operations
2. **Lock Files:** Always commit `uv.lock` for reproducible builds
3. **Python Versions:** uv can manage Python installations, no need for pyenv
4. **Cache:** uv uses a global cache, saving disk space across projects
5. **Compatibility:** uv reads and respects `requirements.txt` and `pyproject.toml`
6. **Scripts:** Use `uv run` instead of activating venv for scripts
7. **Tools:** Use `uvx` for one-off tool runs, `uv tool install` for frequent use

Quick Command Reference

Task	Command
Init project	<code>uv init project-name</code>
Add package	<code>uv add package-name</code>
Install deps	<code>uv sync</code>
Run script	<code>uv run python script.py</code>
Run tool once	<code>uvx tool-name</code>
Install tool	<code>uv tool install tool-name</code>
Create venv	<code>uv venv</code>
List packages	<code>uv pip list</code>
Update deps	<code>uv lock --upgrade && uv sync</code>
Run Streamlit	<code>uv run streamlit run app.py</code>

Migrating from Other Tools

From Conda/Mamba (Detailed Comparison)

Conda Command	uv Equivalent	Notes
<code>conda create -n myenv python=3.11</code>	<code>uv init myproject --python 3.11</code>	uv creates project-based envs
<code>conda activate myenv</code>	<code>source .venv/bin/activate</code> or just use <code>uv run</code>	uv run doesn't require activation
<code>conda install numpy pandas</code>	<code>uv add numpy pandas</code>	Adds to project and installs
<code>conda install -c conda-forge scikit-learn</code>	<code>uv add scikit-learn</code>	uv uses PyPI by default
<code>conda env list</code>	<code>uv python list</code> (for Python versions)	Different paradigm - uv is project-based
<code>conda list</code>	<code>uv pip list</code>	Lists packages in current env
<code>conda update pandas</code>	<code>uv add pandas --upgrade</code>	Updates specific package
<code>conda env export > environment.yml</code>	<code>uv pip freeze</code> or use <code>uv.lock</code>	uv.lock is automatic
<code>conda env create -f environment.yml</code>	<code>uv sync</code>	Installs from uv.lock/pyproject.toml
<code>conda clean -a</code>	Not needed	uv has efficient global cache
<code>conda config --add channels</code>	<code>--index-url</code> flag	Different index system
<code>conda run python script.py</code>	<code>uv run python script.py</code>	Similar functionality

Conda to uv Workflow Example

Conda workflow:

```
bash
```

Create environment

```
conda create -n ml-project python=3.11
```

```
conda activate ml-project
```

Install packages

```
conda install numpy pandas scikit-learn
```

```
conda install -c conda-forge xgboost lightgbm
```

Install from pip when needed

```
pip install streamlit
```

Save environment

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

Run scripts

```
python train.py
```

```
streamlit run app.py
```

Equivalent uv workflow:

```
bash
```



```
# Create project
uv init ml-project --python 3.11
cd ml-project

# Install packages (all from PyPI)
uv add numpy pandas scikit-learn xgboost lightgbm streamlit

# No activation needed - just run
uv run python train.py
uv run streamlit run app.py

# Dependencies are auto-locked in uv.lock
# To recreate: just run 'uv sync'
```

Key Differences for Conda Users

1. Project-based vs Global Environments

- Conda: Named environments accessible globally
- uv: Project-specific environments (like node_modules)

2. Package Sources

- Conda: Multiple channels (defaults, conda-forge, bioconda)
- uv: PyPI by default (can add other indexes)

3. Activation

- Conda: Requires `conda activate`
- uv: Use `uv run` or activate traditionally

4. Non-Python Dependencies

- Conda: Can install system libraries, compilers, R, etc.
- uv: Python packages only

5. Scientific Stack

- Conda: MKL-optimized numpy, specific CUDA versions
- uv: Standard PyPI wheels (though often identical performance)

Common Conda Patterns in uv

Data Science Setup:

```
bash

# Conda approach
conda create -n ds python=3.11
conda activate ds
conda install jupyter numpy pandas matplotlib seaborn scikit-learn

# uv approach
uv init ds-project --python 3.11
cd ds-project
uv add jupyter numpy pandas matplotlib seaborn scikit-learn
uv run jupyter lab
```

GPU/CUDA Projects:

```
bash

# Conda (better for CUDA)
conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia

# uv (need to specify index)
uv add torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
```

Managing Python Versions:

```
bash

# Conda
conda create -n py39 python=3.9
conda create -n py311 python=3.11

# uv
uv python install 3.9 3.11 # Install multiple versions
uv init --python 3.9 project1
uv init --python 3.11 project2
```

Tips for Conda Users Switching to uv

1. **Think in projects, not environments** - Each project has its own dependencies
2. **Use `uv run` liberally** - No need to activate environments
3. **PyPI has most packages** - Even scientific packages are well-maintained on PyPI now
4. **Lock files are automatic** - No need to manually export environments
5. **It's MUCH faster** - Package resolution and installation takes seconds, not minutes
6. **Keep Conda for special cases** - Complex scientific stacks, R integration, or specific CUDA needs

When to Keep Using Conda

- **Mixed language projects** (Python + R + Julia)
- **Specific CUDA/cuDNN versions** for deep learning
- **Bioinformatics pipelines** requiring specific tools
- **System libraries** that aren't on PyPI
- **HPC environments** with module systems

From pip/venv

bash

Instead of: python -m venv .venv

uv venv

Instead of: pip install -r requirements.txt

uv pip [install](#) -r requirements.txt

Or better: uv add -r requirements.txt

From pipx

bash

Instead of: pipx run black

uvx black

Instead of: pipx install black

uv tool [install](#) black

From poetry

bash

Instead of: poetry init

uv init

Instead of: poetry add requests

uv add requests

Instead of: poetry install

uv sync

Instead of: poetry run python script.py

uv run python script.py