# RC Quick Byte: uv Package Manager

Research Computing
UNIVERSITY OF COLORADO BOULDER

# uv Package Manager -- Get Your Sunscreen!

Date: October 1, 2025

Instructor: Mohal Khandelwal

- Website: www.rc.colorado.edu/rc
- Documentation: https://curc.readthedocs.io
- Helpdesk: rc-help@colorado.edu
- Survey: http://tinyurl.com/curc-survey18

**Slides**
https://github.com/ResearchComputing/uv_package_manager_quick_byte

# What You'll Learn

- Introduction to uv
- Key features of uv
- Comparative analysis of uv with other tools
- Using uv on CURC

# What is uv?

- Fast, modern Python package manager and environment builder.
- Built by Astral (same team behind Ruff).
- Implemented in Rust.
- All-in-one solution
- Designed to work natively with Python's built-in venv


*Image source: [https://docs.astral.sh/uv/](https://docs.astral.sh/uv/)*

# Features of uv

- Drop-in Replacement for pip, pip-tools.
- 10-100x faster than traditional tools.
- Reliable and reproducible builds
- Memory-efficient operation, especially for large projects

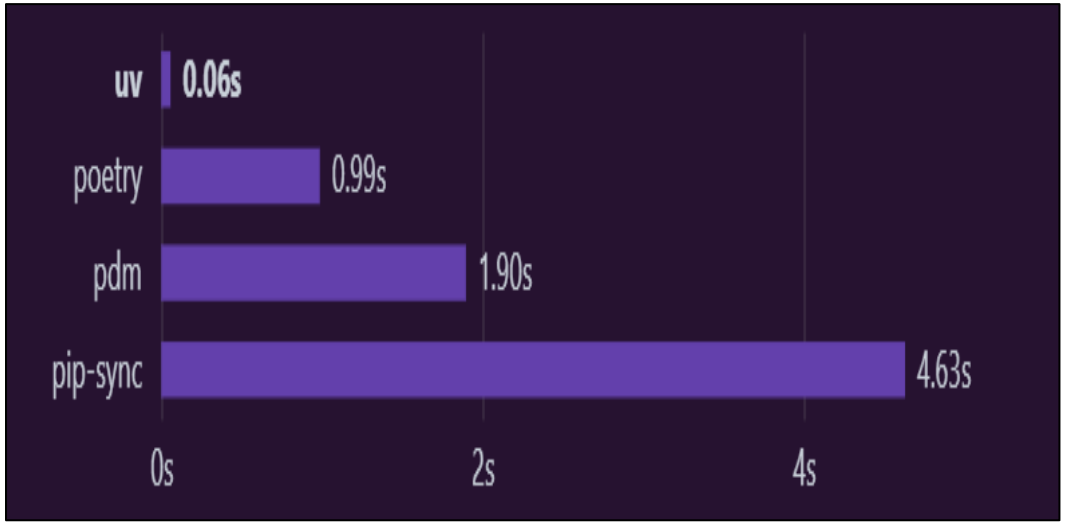


*Image source: [https://docs.astral.sh/uv/](https://docs.astral.sh/uv/)*

# uv vs. the status quo

| Feature | uv | Traditional pip + venv | conda |
|---|---|---|---|
| **Speed** | 10-100x faster than pip | Baseline | Slower than pip |
| **Memory Usage** | Very efficient | Higher | High |
| **Reproducibility** | Universal lockfile (uv sync). | Requires a separate tool (pip-tools) for locking. | YAML files (often flaky). |
| **Non-Python Packages** | No | No | Yes |
| **Dependency Resolution** | Fast, modern resolver | Basic | Comprehensive |
| **Ease of use** | Simple, single tool | Broader scope with a steeper learning curve. | Requires multiple commands (python -m venv, source, pip). |

# Getting Started with uv

- To start using uv, you must first load the module in your interactive session:

```
$ module spider uv
$ module load uv
$ uv --version
```

# Creating a uv Environment

- uv automates the creation and management of environments using venv
- venv is Python's built-in tool for creating isolated environments
- Each venv:
  - Has its own Python interpreter
  - Has its own installed packages, separate from system Python
  - Avoids dependency conflicts across projects
- It's a lightweight alternative to Conda environments

# Creating a uv Environment

- Loading the module sets a key environment variable **$UV_ENVS**.

```
$ echo $UV_ENVS
/projects/$USER/software/uv/envs
```

- To create a new environment

```
$ uv venv $UV_ENVS/myenv
$ uv venv $UV_ENVS/myenv1 --python 3.8
```

# Activating the Environment

- Packages are installed within activated environments
  - Fully compatible with PyPI
  - Uses caching to speed up repeated installs

```
$ source $UV_ENVS/myenv/bin/activate
$ uv pip install numpy
$ uv pip install pandas
```

# Useful Commands

```
uv <command> --help                    # help for specific commands
uv venv $UV_ENVS/<envname>             # create a new environment
uv pip install <packagename>           # install a package in active env
uv pip uninstall <packagename>         # uninstall a package from env
rm -rf $UV_ENVS/<envname>              # remove an environment
uv cache clean                         # clean up unused cache
uv pip freeze                          # list installed packages
deactivate                             # deactivate environment
```

Note: Be very careful with rm -rf when removing environments

# Any Questions

# Thank you!

Survey and feedback

http://tinyurl.com/curc-survey18