Python Package Managers for Research Project

Zesen Zhuang

SciEcon CIC

December 19, 2022

Outline

What is Package Manager and Why

A Python environment manager is a tool that allows you to create and manage separate Python environments on your system. This can be useful for several reasons:

- It allows you to have multiple versions of Python installed on your system and switch between them easily.
- It allows you to install and manage packages in a specific environment, rather than globally on your system. This can be useful for isolating different projects or environments, and for avoiding conflicts between package versions.
- It allows you to create and share reproducible environments, which can be useful for collaborating with others or for deploying Python applications.

Some Examples

Frequently used Python package manager

- pip: https://pypi.org/project/pip/
- conda: https://docs.conda.io/en/latest/
- virtualenv: https://virtualenv.pypa.io/en/latest/
- pyenv: https://github.com/pyenv/pyenv
- pipenv: https://pipenv.pypa.io/en/latest/

pip

pip is the default package manager for Python, and it is included with most Python installations.

Here are some common tasks you can perform using pip:

```
pip install numpy # install a package
pip install --upgrade numpy # upgrade a package
pip uninstall numpy # uninstall a package
pip list # list installed packages
pip freeze > requirements.txt # save installed packages to a file
pip install -r requirements.txt # install packages from a file
```

conda

conda is a package manager and environment manager developed by Anaconda, Inc. It is particularly useful for scientific computing and data science, and it can be used to create, manage, and share Python environments and packages.

Here are some common tasks you can perform using conda.

- Create a new environment
- Activate an environment
- Deactivate an environment
- Install a package
- Upgrade a package
- Uninstall a package
- List environments
- Export an environment

conda continued

Create a new environment: To create a new environment using conda, you can use the conda create command followed by the name of the environment and the packages you want to include in it. For example, to create an environment named myenv with Python 3.8 and the numpy package, you can use the following command:

```
conda create --name myenv python=3.8 numpy
```

conda continued

Activate an environment: To activate an environment, you can use the conda activate command followed by the name of the environment. For example:

```
1 conda activate myenv
```

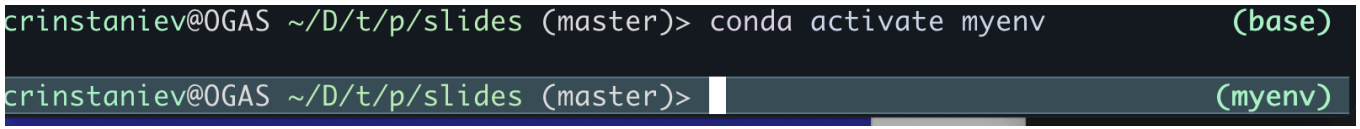

Figure: conda activated

As shown in figure above, the environment name is displayed in parentheses in the prompt.

conda continued

Deactivate an environment: To deactivate the current environment, you can use the conda deactivate command.

```
1 conda deactivate
```

conda continued

Install a package: To install a package in the current environment, you can use the conda install command followed by the name of the package. For example:

```
conda install numpy
```

Upgrade a package: To upgrade an installed package to the latest version, you can use the conda update command followed by the name of the package. For example:

```
conda update numpy
```

Uninstall a package: To uninstall a package, you can use the conda remove command followed by the name of the package. For example:

```
conda remove numpy
```

conda continued

List environments: To see a list of all the environments on your system, you can use the conda env list command.

```
conda env list
```

Export an environment: To create a YAML file that describes the current environment and its packages, you can use the conda env export command. This can be useful for creating a reproducible environment or for sharing your environment with others.

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

virtualenv

virtualenv is a tool that allows you to create isolated Python environments. It can be useful for testing and deploying Python applications, as it allows you to create environments with specific package versions and configurations.

virtualenv continued

Install virtualenv: To use virtualenv, you will first need to install it using pip. You can do this by running the following command:

```
pip install virtualenv
```

Create a new environment: To create a new environment using virtualenv, you can use the virtualenv command followed by the name of the environment. For example, to create an environment named myenv, you can use the following command:

```
virtualenv myenv
```

virtualenv continued

Activate an environment: To activate an environment, you will need to use the activate script located in the bin directory of the environment. On Windows, you can do this by running the following command:

```
myenv\Scripts\activate.bat
```

On macOS or Linux, you can do this by running the following command:

```
source myenv/bin/activate
```

Deactivate an environment: To deactivate the current environment, you can use the deactivate command.

```
deactivate
```

virtualenv continued

Install a package: To install a package in the current environment, you can use pip as usual. For example:

```
pip install numpy
```

Create a requirements file: To create a requirements file that lists all the packages and their versions installed in the current environment, you can use the pip freeze command. This can be useful for creating a reproducible environment or for sharing your dependencies with others.

```
pip freeze > requirements.txt
```

pyenv

pyenv is a tool that allows you to manage multiple Python versions and switch between them easily. It can also be used to create and manage virtual environments.

pyenv continued

Install pyenv: To use pyenv, you will first need to install it. You can do this by following the instructions on the pyenv GitHub page: `https://github.com/pyenv/pyenv#installation`
List available Python versions: To see a list of all the Python versions that are available to pyenv, you can use the pyenv install –list command.

```
pyenv install --list
```

Install a Python version: To install a specific Python version using pyenv, you can use the pyenv install command followed by the version number. For example, to install Python 3.9, you can use the following command:

```
pyenv install 3.9
```

pyenv continued

Set the global Python version: To set the global Python version, which will be used as the default Python version for all your terminal sessions, you can use the pyenv global command followed by the version number. For example:

```
pyenv global 3.9
```

Create a new virtual environment: To create a new virtual environment using pyenv, you can use the pyenv virtualenv command followed by the name of the environment and the Python version you want to use. For example, to create an environment named myenv using Python 3.9, you can use the following command:

```
pyenv virtualenv 3.9 myenv
```

pyenv continued

Create a new virtual environment: To create a new virtual environment using pyenv, you can use the pyenv virtualenv command followed by the name of the environment and the Python version you want to use. For example, to create an environment named myenv using Python 3.9, you can use the following command:

```
pyenv virtualenv 3.9 myenv
```

Activate a virtual environment: To activate a virtual environment, you can use the pyenv activate command followed by the name of the environment. For example:

```
pyenv activate myenv
```

pyenv continued

Deactivate a virtual environment: To deactivate the current virtual environment, you can use the pyenv deactivate command.

```
1 pyenv deactivate
```

pipenv

pipenv is a tool that combines pip and virtualenv into a single package, and it is designed to make it easier to manage Python environments and packages.

pipenv continued

Install pipenv: To use pipenv, you will first need to install it using pip. You can do this by running the following command:

```
pip install pipenv
```

Create a new environment: To create a new environment and install the packages specified in a Pipfile using pipenv, you can use the pipenv install command. For example:

```
pipenv install
```

pipenv continued

Install a package: To install a package and add it to the Pipfile using pipenv, you can use the pipenv install command followed by the name of the package. For example:

```
pipenv install numpy
```

Upgrade a package: To upgrade an installed package to the latest version and update the Pipfile accordingly, you can use the pipenv update command followed by the name of the package. For example:

```
pipenv update numpy
```

pipenv continued

Uninstall a package: To uninstall a package and remove it from the Pipfile, you can use the pipenv uninstall command followed by the name of the package. For example:

```
pipenv uninstall numpy
```

Activate the environment: To activate the environment created by pipenv, you can use the pipenv shell command. This will open a new shell with the environment activated.

```
pipenv shell
```

Deactivate the environment: To deactivate the current environment, you can use the exit command.

```
exit
```

Thanks for listening!