Installing Python Modules

Email: distutils-sig@python.org

As a popular open source development project, Python has an active supporting community of contributors and users that also make their software available for other Python developers to use under open source license terms.

This allows Python users to share and collaborate effectively, benefiting from the solutions others have already created to common (and sometimes even rare!) problems, as well as potentially contributing their own solutions to the common pool.

This guide covers the installation part of the process. For a guide to creating and sharing your own Python projects, refer to the distribution guide.

> **Note:** For corporate and other institutional users, be aware that many organisations have their own policies around using and contributing to open source software. Please take such policies into account when making use of the distribution and installation tools provided with Python.

Key terms

- `pip` is the preferred installer program. Starting with Python 3.4, it is included by default with the Python binary installers.
- A *virtual environment* is a semi-isolated Python environment that allows packages to be installed for use by a particular application, rather than being installed system wide.
- `venv` is the standard tool for creating virtual environments, and has been part of Python since Python 3.3. Starting with Python 3.4, it defaults to installing `pip` into all created virtual environments.
- `virtualenv` is a third party alternative (and predecessor) to `venv`. It allows virtual environments to be used on versions of Python prior to 3.4, which either don't provide `venv` at all, or aren't able to automatically install `pip` into created environments.
- The Python Packaging Index is a public repository of open source licensed packages made available for use by other Python users.
- the Python Packaging Authority are the group of developers and documentation authors responsible for the maintenance and evolution of the standard packaging tools and the associated metadata and file format standards. They maintain a variety of tools, documentation, and issue trackers on both GitHub and BitBucket.
- `distutils` is the original build and distribution system first added to the Python standard library in 1998. While direct use of `distutils` is being phased

out, it still laid the foundation for the current packaging and distribution infrastructure, and it not only remains part of the standard library, but its name lives on in other ways (such as the name of the mailing list used to coordinate Python packaging standards development).

Deprecated since version 3.6: `pyvenv` was the recommended tool for creating virtual environments for Python 3.3 and 3.4, and is deprecated in Python 3.6.

Changed in version 3.5: The use of `venv` is now recommended for creating virtual environments.

See also: Python Packaging User Guide: Creating and using virtual environments

Basic usage

The standard packaging tools are all designed to be used from the command line.

The following command will install the latest version of a module and its dependencies from the Python Packaging Index:

```
python -m pip install SomePackage
```

Note: For POSIX users (including Mac OS X and Linux users), the examples in this guide assume the use of a virtual environment.

For Windows users, the examples in this guide assume that the option to adjust the system PATH environment variable was selected when installing Python.

It's also possible to specify an exact or minimum version directly on the command line. When using comparator operators such as `>`, `<` or some other special character which get interpreted by shell, the package name and the version should be enclosed within double quotes:

```
python -m pip install SomePackage==1.0.4    # specific version
python -m pip install "SomePackage>=1.0.4"  # minimum version
```

Normally, if a suitable module is already installed, attempting to install it again will have no effect. Upgrading existing modules must be requested explicitly:

```
python -m pip install --upgrade SomePackage
```

More information and resources regarding `pip` and its capabilities can be found in the Python Packaging User Guide.

Creation of virtual environments is done through the `venv` module. Installing packages into an active virtual environment uses the commands shown above.

> **See also:** Python Packaging User Guide: Installing Python Distribution Packages

How do I …?

These are quick answers or links for some common tasks.

… install `pip` in versions of Python prior to Python 3.4?

Python only started bundling `pip` with Python 3.4. For earlier versions, `pip` needs to be "bootstrapped" as described in the Python Packaging User Guide.

> **See also:** Python Packaging User Guide: Requirements for Installing Packages

… install packages just for the current user?

Passing the `--user` option to `python -m pip install` will install a package just for the current user, rather than for all users of the system.

… install scientific Python packages?

A number of scientific Python packages have complex binary dependencies, and aren't currently easy to install using `pip` directly. At this point in time, it will often be easier for users to install these packages by other means rather than attempting to install them with `pip`.

> **See also:** Python Packaging User Guide: Installing Scientific Packages

… work with multiple versions of Python installed in parallel?

On Linux, Mac OS X, and other POSIX systems, use the versioned Python commands in combination with the `-m` switch to run the appropriate copy of `pip`:

```
python2   -m pip install SomePackage  # default Python 2
python2.7 -m pip install SomePackage  # specifically Python 2.7
python3   -m pip install SomePackage  # default Python 3
python3.4 -m pip install SomePackage  # specifically Python 3.4
```

Appropriately versioned `pip` commands may also be available.

On Windows, use the `py` Python launcher in combination with the `-m` switch:

```
py -2   -m pip install SomePackage  # default Python 2
py -2.7 -m pip install SomePackage  # specifically Python 2.7
py -3   -m pip install SomePackage  # default Python 3
py -3.4 -m pip install SomePackage  # specifically Python 3.4
```

Common installation issues

Installing into the system Python on Linux

On Linux systems, a Python installation will typically be included as part of the distribution. Installing into this Python installation requires root access to the system, and may interfere with the operation of the system package manager and other components of the system if a component is unexpectedly upgraded using `pip`.

On such systems, it is often better to use a virtual environment or a per-user installation when installing packages with `pip`.

Pip not installed

It is possible that `pip` does not get installed by default. One potential fix is:

```
python -m ensurepip --default-pip
```

There are also additional resources for installing pip.

Installing binary extensions

Python has typically relied heavily on source based distribution, with end users being expected to compile extension modules from source as part of the installation process.

With the introduction of support for the binary `wheel` format, and the ability to publish wheels for at least Windows and Mac OS X through the Python Packaging Index, this problem is expected to diminish over time, as users are more regularly able to install pre-built extensions rather than needing to build them themselves.

Some of the solutions for installing scientific software that are not yet available as pre-built `wheel` files may also help with obtaining other binary extensions without needing to build them locally.

See also: Python Packaging User Guide: Binary Extensions