

# Installing Packages

This section covers the basics of how to install Python [packages](#).

It's important to note that the term “package” in this context is being used as a synonym for a [distribution](#) (i.e. a bundle of software to be installed), not to refer to the kind of [package](#) that you import in your Python source code (i.e. a container of modules). It is common in the Python community to refer to a [distribution](#) using the term “package”. Using the term “distribution” is often not preferred, because it can easily be confused with a Linux distribution, or another larger software distribution like Python itself.

## Contents

- [Requirements for Installing Packages](#)
  - [Install pip, setuptools, and wheel](#)
  - [Optionally, Create a virtual environment](#)
- [Creating Virtual Environments](#)
- [Use pip for Installing](#)
- [Installing from PyPI](#)
- [Source Distributions vs Wheels](#)
- [Upgrading packages](#)
- [Installing to the User Site](#)
- [Requirements files](#)
- [Installing from VCS](#)
- [Installing from other Indexes](#)
- [Installing from a local src tree](#)
- [Installing from local archives](#)
- [Installing from other sources](#)
- [Installing Prereleases](#)
- [Installing Setuptools “Extras”](#)

## Requirements for Installing Packages

This section describes the steps to follow before installing other Python packages.

### Install pip, setuptools, and wheel

- If you have Python 2  $\geq 2.7.9$  or Python 3  $\geq 3.4$  installed from [python.org](#), you will already have [pip](#) and [setuptools](#), but will need to upgrade to the latest version:

On Linux or macOS:

```
pip install -U pip setuptools
```

On Windows:

```
python -m pip install -U pip setuptools
```

 v: latest ▼

- If you're using a Python install on Linux that's managed by the system package manager (e.g "yum", "apt-get" etc...), and you want to use the system package manager to install or upgrade pip, then see [Installing pip/setuptools/wheel with Linux Package Managers](#)
- Otherwise:
  - Securely Download [get-pip.py](#) [1]
  - Run `python get-pip.py`. [2] This will install or upgrade pip. Additionally, it will install [setuptools](#) and [wheel](#) if they're not installed already.

**Warning:** Be cautious if you're using a Python install that's managed by your operating system or another package manager. `get-pip.py` does not coordinate with those tools, and may leave your system in an inconsistent state. You can use `python get-pip.py --prefix=/usr/local/` to install in `/usr/local` which is designed for locally-installed software.

## Optionally, Create a virtual environment

See [section below](#) for details, but here's the basic commands:

Using [virtualenv](#):

```
pip install virtualenv
virtualenv <DIR>
source <DIR>/bin/activate
```

Using [venv](#): [3]

```
python3 -m venv <DIR>
source <DIR>/bin/activate
```

## Creating Virtual Environments

Python "Virtual Environments" allow Python [packages](#) to be installed in an isolated location for a particular application, rather than being installed globally.

Imagine you have an application that needs version 1 of LibFoo, but another application requires version 2. How can you use both these applications? If you install everything into `/usr/lib/python2.7/site-packages` (or whatever your platform's standard location is), it's easy to end up in a situation where you unintentionally upgrade an application that shouldn't be upgraded.

Or more generally, what if you want to install an application and leave it be? If an application works, any change in its libraries or the versions of those libraries can break the application.

Also, what if you can't install [packages](#) into the global site-packages directory? For instance, on a shared host.

In all these cases, virtual environments can help you. They have their own installation directories and they don't share libraries with other virtual environments.

Currently, there are two viable tools for creating Python virtual environments:

- [venv](#) is available by default in Python 3.3 and later, and installs [pip](#) and [setuptools](#) into created virtual environments in Python 3.4 and later.
- [virtualenv](#) needs to be installed separately, but supports Python 2.6+ and Python 3.3+, and [pip](#), [setuptools](#) and [wheel](#) are always installed into created virtual environments by default (regardless of Python version).

The basic usage is like so:

Using [virtualenv](#):

```
virtualenv <DIR>  
source <DIR>/bin/activate
```

Using [venv](#):

```
python3 -m venv <DIR>  
source <DIR>/bin/activate
```

For more information, see the [virtualenv](#) docs or the [venv](#) docs.

## Use pip for Installing

[pip](#) is the recommended installer. Below, we'll cover the most common usage scenarios. For more detail, see the [pip docs](#), which includes a complete [Reference Guide](#).

There are a few cases where you might want to use [easy\\_install](#) instead of pip. For details, see [pip vs easy\\_install](#).

## Installing from PyPI

The most common usage of [pip](#) is to install from the [Python Package Index](#) using a [requirement specifier](#). Generally speaking, a requirement specifier is composed of a project name followed by an optional [version specifier](#). [PEP 440](#) contains a [full specification](#) of the currently supported specifiers. Below are some examples.

To install the latest version of "SomeProject":

```
pip install 'SomeProject'
```

To install a specific version:

```
pip install 'SomeProject==1.4'
```

 v: latest ▾

To install greater than or equal to one version and less than another:

```
pip install 'SomeProject>=1,<2'
```

To install a version that's [“compatible”](#) with a certain version: [\[4\]](#)

```
pip install 'SomeProject~=1.4.2'
```

In this case, this means to install any version “==1.4.\*” version that’s also “>=1.4.2”.

## Source Distributions vs Wheels

`pip` can install from either [Source Distributions \(sdist\)](#) or [Wheels](#), but if both are present on PyPI, `pip` will prefer a compatible [wheel](#).

[Wheels](#) are a pre-built [distribution](#) format that provides faster installation compared to [Source Distributions \(sdist\)](#), especially when a project contains compiled extensions.

If `pip` does not find a wheel to install, it will locally build a wheel and cache it for future installs, instead of rebuilding the source distribution in the future.

## Upgrading packages

Upgrade an already installed *SomeProject* to the latest from PyPI.

```
pip install --upgrade SomeProject
```

## Installing to the User Site

To install [packages](#) that are isolated to the current user, use the `--user` flag:

```
pip install --user SomeProject
```


For more information see the [User Installs](#) section from the `pip` docs.

## Requirements files

Install a list of requirements specified in a [Requirements File](#).

```
pip install -r requirements.txt
```

## Installing from VCS

Install a project from VCS in “editable” mode. For a full breakdown of the syntax `v: latest`  see section on [VCS Support](#).

```
pip install -e git+https://git.repo/some_pkg.git#egg=SomeProject # fr
pip install -e hg+https://hg.repo/some_pkg.git#egg=SomeProject # fr
pip install -e svn+svn://svn.repo/some_pkg/trunk/#egg=SomeProject # fr
pip install -e git+https://git.repo/some_pkg.git@feature#egg=SomeProject # fr
```

## Installing from other Indexes

Install from an alternate index

```
pip install --index-url http://my.package.repo/simple/ SomeProject
```

Search an additional index during install, in addition to [PyPI](#)

```
pip install --extra-index-url http://my.package.repo/simple SomeProject
```

## Installing from a local src tree

Installing from local src in [Development Mode](#), i.e. in such a way that the project appears to be installed, but yet is still editable from the src tree.

```
pip install -e <path>
```

You can also install normally from src

```
pip install <path>
```

## Installing from local archives

Install a particular source archive file.

```
pip install ./downloads/SomeProject-1.0.4.tar.gz
```

Install from a local directory containing archives (and don't check [PyPI](#))

```
pip install --no-index --find-links=file:///local/dir/ SomeProject
pip install --no-index --find-links=/local/dir/ SomeProject
pip install --no-index --find-links=relative/dir/ SomeProject
```

## Installing from other sources

To install from other data sources (for example Amazon S3 storage) you can create a helper application that presents the data in a [PEP 503](#) compliant index format, and use the `--extra-index-url` flag to direct pip to use that index.

```
./s3helper --port=7777
pip install --extra-index-url http://localhost:7777 SomeProject
```

 v: latest ▾

## Installing Prereleases

Find pre-release and development versions, in addition to stable versions. By default, pip only finds stable versions.

```
pip install --pre SomeProject
```

## Installing Setuptools “Extras”

Install [setuptools extras](#).

```
$ pip install SomePackage[PDF]
$ pip install SomePackage[PDF]==3.0
$ pip install -e .[PDF]==3.0 # editable project in current directory
```

- [1] “Secure” in this context means using a modern browser or a tool like *curl* that verifies SSL certificates when downloading from https URLs.
- [2] Depending on your platform, this may require root or Administrator access. [pip](#) is currently considering changing this by [making user installs the default behavior](#).
- [3] Beginning with Python 3.4, *venv* (a stdlib alternative to [virtualenv](#)) will create virtualenv environments with *pip* pre-installed, thereby making it an equal alternative to [virtualenv](#).
- [4] The compatible release specifier was accepted in [PEP 440](#) and support was released in [setuptools](#) v8.0 and [pip](#) v6.0