

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](#)
 - [Ensure you can run Python from the command line](#)
 - [Ensure you can run pip from the command line](#)
 - [Ensure pip, setuptools, and wheel are up to date](#)
 - [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.

Ensure you can run Python from the command line

Before you go any further, make sure you have Python and that the expected version is available from your command line. You can check this by running:

 [v: latest](#) ▼

```
python --version
```

You should get some output like Python 3.6.3. If you do not have Python, please install the latest 3.x version from python.org or refer to the [Installing Python](#) section of the Hitchhiker's Guide to Python.

Note: If you're a newcomer and you get an error like this:

```
>>> python --version
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'python' is not defined
```

It's because this command and other suggested commands in this tutorial are intended to be run in a *shell* (also called a *terminal* or *console*). See the Python for Beginners [getting started tutorial](#) for an introduction to using your operating system's shell and interacting with Python.

Note: If you're using an enhanced shell like IPython or the Jupyter notebook, you can run system commands like those in this tutorial by prefacing them with a `!` character:

```
In [1]: import sys
        !{sys.executable} --version
Python 3.6.3
```

It's recommended to write `{sys.executable}` rather than plain `python` in order to ensure that commands are run in the Python installation matching the currently running notebook (which may not be the same Python installation that the `python` command refers to).

Note: Due to the way most Linux distributions are handling the Python 3 migration, Linux users using the system Python without creating a virtual environment first should replace the `python` command in this tutorial with `python3` and the `pip` command with `pip3 --user`. Do *not* run any of the commands in this tutorial with `sudo`: if you get a permissions error, come back to the section on creating virtual environments, set one up, and then continue with the tutorial as written.

Ensure you can run pip from the command line

Additionally, you'll need to make sure you have [pip](#) available. You can check this by running:

```
pip --version
```

If you installed Python from source, with an installer from python.org, or via [Homebrew](#) you should already have `pip`. If you're on Linux and installed using your OS package manager, you may have to install `pip` separately, see [Installing pip/setuptools/wheel with Linux Package Managers](#).

If `pip` isn't already installed, then first try to bootstrap it from the standard library:

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

If that still doesn't allow you to run pip:

- 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.

Ensure pip, setuptools, and wheel are up to date

While pip alone is sufficient to install from pre-built binary archives, up to date copies of the setuptools and wheel projects are useful to ensure you can also install from source archives:

```
python -m pip install --upgrade pip setuptools wheel
```

Optionally, create a virtual environment



See [section below](#) for details, but here's the basic [venv](#) [3] command to use on a typical Linux system:

```
python3 -m venv tutorial_env  
source tutorial_env/bin/activate
```

This will create a new virtual environment in the `tutorial_env` subdirectory, and configure the current shell to use it as the default python environment.

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/python3.6/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  `v: latest`  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 common 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.

In both of the above cases, Windows users should not use the *source* command, but should rather run the *activate* script directly from the command shell. The use of *source* under Unix shells ensures that the virtual environment's variables are set within the current shell, and not in a subprocess (which then disappears, having no useful effect).

Managing multiple virtual environments directly can become tedious, so the [dependency management tutorial](#) introduces a higher level tool, [Pipenv](#), that automatically manages a separate virtual environment for each project and application that you work on.

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](#).

 v: latest ▼

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'
```

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

 v: latest ▼

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.

Note that the `--user` flag has no effect when inside a virtual environment - all installation commands will affect the virtual environment.

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, see pip’s section on [VCS Support](#).

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

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>
```

 [v: latest](#) ▼

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
```

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.



v: latest ▼

[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