What is the best way to install Python packages?

What is the best way to install Python packages in Ubuntu 11? I am a recent convert to Ubuntu and want to learn best practices.

For context, I am looking to install the tweeststream package, but I did not see it in my Synaptic package manager. Also, I am very new to programming, but I usually can follow along with code samples.

python

edited Aug 19 '16 at 21:50
Peter Mortensen
1,051 2 10 16

asked Jan 13 '12 at 0:29
Btibert3
290 1 5 8

See also askubuntu.com/questions/21027/easy-install-pip-or-apt-get – Warrick Sep 26 '12 at 12:16

4 Answers

I think best way for you would be to install Python packaging system like "python-pip". You can install it with Synaptic or Ubuntu Software Center.

Pip will allow you to easy install and uninstall Python packages, simply as `pip install package`

In your case it would be something like this from terminal:

```
sudo pip install tweeststream
```

answered Jan 13 '12 at 2:00
zetah
5,219 5 35 63

3 This is fine for simple packages, but not so good for larger packages e.g. numpy – hayd Oct 25 '13 at 18:03

@hayd: why? Is that related to askubuntu.com/questions/595366/… – naught101 Mar 11 '15 at 3:57

2 @naught101 I think I probably meant scipy, not sure if it's related... looks like pip is too late in the python path. I strongly recommend using anaconda/conda, it's vastly superior to pip IMO. – hayd Mar 12 '15 at 2:47

2 IMHO using `sudo pip <anything>` should be used very carefuly. Since `tweetstream` isn't in the Ubuntu Software Center, I recommend using a virtualenv. tweetstream's `setup.py` requires `anyjson` which *is* an Ubuntu Package. Not installing tweetstream in a `virtualenv` may cause other Ubuntu apps dependent on `anyjson` to fail. Troubleshoot hard. – Mark Mikofski Sep 30 '15 at 16:43

5 To repeat: don't use `sudo pip` on Ubuntu. @MarkMikofski: you don't need virtualenv (unless you want it for some specific reason). `python -mpip install --user package-name` can install the package `packafe-name` for the current user. If there are complex (large C extensions) dependencies; you could install them using `apt-get` , to avoid installing build dependencies unless necessary. – jfs Feb 23 '16 at 20:18

updated: 2017-03-27: PEP 513 - `manylinux` binaries for PyPI

updated: 2016-08-19: Continuum Anaconda Option

This is somewhat a duplicate of easy_install/pip or apt-get.

For *global* Python packages, use either the Ubuntu Software Center, apt, apt-get or synaptic

Ubuntu uses Python for many important functions, therefore interfering with Python can corrupt your OS. This is the main reason I never use `pip` on my Ubuntu system, but instead I use either Ubuntu Software Center, synaptic, `apt-get`, or the newer just `apt`, which all by default install packages from the Ubuntu repository. These packages are tested, usually pre-compiled so they install faster and ultimately designed for Ubuntu. In addition all required dependencies are also installed and a log of installs is maintained so they can be rolled back. I think most packages have corresponding Launchpad repos so you can file issues.

Another reason to use either Ubuntu packages is that sometimes these Python packages have different names depending on where you downloaded them from. Python-chardet is an example of a package which at one time was named one thing on PyPI and another thing in the Ubuntu repository. Therefore doing something like `pip install requests` will not realize that chardet is already installed in your system because the Ubuntu version has a different name, and consequently install a new version which will corrupt your system in a minor insignificant way but still why would you do that.

In general you only want to install trusted code into your OS. So **you should be *nervous*** about typing `$ sudo pip <anything-could-be-very-bad>`.

Lastly some things are just easier to install using either Ubuntu packages. For example if you try `pip install numpy` to install numpy & scipy unless you have already installed gfortran, atlas-dev, blas-dev and lapack-dev, you will see an endless stream of compile errors. However, installing numpy & scipy through the Ubuntu repository is as easy as...

```
$ sudo apt-get install python-numpy python-scipy
```

You are in luck, because you are using Ubuntu, one of the most widely supported and oft updated distributions existing. Most likely every Python package you will need is in the Ubuntu repository, and probably already installed on your machine. And every 6 months, a new cycle of packages will be released with the latest distribution of Ubuntu.

If you are 100% confident that the package will not interfere with your Ubuntu system in any way, then you can install it using pip and Ubuntu is nice enough to keep these packages separate from the distro packages by placing the distro packages in a folder called `dist-packages/`. Ubuntu repository has both pip, virtualenv and setuptools. However, I second Wojciech's suggestion to use virtualenv.

For *personal* Python projects use pip and wheel in a virtualenv

If you need the latest version, or the module is not in the Ubuntu repository then start a virtualenv and use pip to install the package. Although pip and setuptools have merged, IMO pip is preferred over easy-install or distutils, because it will always wait until the package is completely downloaded and built before it copies it into your file system, and it makes upgrading or uninstalling a breeze. In a lot of ways it is similar to apt-get, in that it generally handles dependencies well. However you ~~will~~ *may* have to handle some dependencies yourself, *but since PEP 513 was adopted there are now `manylinux` binaries at the Python Package Index (PyPI) for popular Linux distros like Ubuntu and Fedora.* ~~for example as mentioned above for NumPy and SciPy make sure you have installed gfortran, atlas-dev, blas-dev and lapack-dev from the Ubuntu repository~~ For example, both NumPy and SciPy are now distributed for Ubuntu as `manylinux` wheels **by default** using OpenBLAS instead of ATLAS. You can still build them from source by using the pip options `--no-use-wheel` or `--no-binary <format control>`.

```
~$ sudo apt-get install gfortran libblas-dev liblapack-dev libatlas-dev python-
virtualenv
~$ mkdir ~/.venvs
~$ virtualenv ~/.venvs/my_py_proj
~$ source ~/.venvs/my_py_proj/bin/activate
~(my_py_proj)$ pip install --no-use-wheel numpy scipy
```

See the next section, "You're not in `sudoers`", below for installing updated versions of pip, setuptools, virtualenv or wheels to your *personal* profile using the `--user` installation scheme with pip. You can use this to update pip for your personal use as J.F. Sebastian indicated in his comment to another answer. *NOTE: the `-m` is really only necessary on MS Windows when updating pip.*

```
python -m pip install --user pip setuptools wheel virtualenv
```

Newer versions of pip automatically cache wheels, so the following is only useful for older versions of pip. Since you may end up installing these many times, consider using wheel with pip to create a wheelhouse. Wheel is already included in `virtualenv` since v13.0.0 therefore if your version of `virtualenv` is too old, you may need to install wheel first.

```
~(my_py_proj)$ pip isntall wheel  # only for virtualenv < v13.0.0
~(my_py_proj)$ pip wheel --no-use-wheel numpy scipy
```

This will create binary wheel files in `<cwd>/wheelhouse` , use `-d` to specify a different directory. Now if you start another virtualenv and you need the same packages you've already built them and you can install them form your wheelhouse using `pip install --find-links=<fullpath>/wheelhouse`

Read Installing Python Modules in the Python documentation and Get Packages on the Python Package Index main page. Also pip, virtualenv and wheel.

If you're not in `sudoers` and `virtualenv` isn't installed.

Another option to using a virtual environment, or if you are using a Linux share without root privileges, then using either the `--user` or `--home=<wherever-you-want>` Python installation schemes with Python's `distutils` will install packages to the value of `site.USERBASE` or to wherever you want. Newer versions of pip also have a `--user` option. *Do not use `sudo` !*

```
pip install --user virtualenv
```

If your Linux version of pip is too old, then you can pass setup options using `--install-option` which is useful for passing custom options to some `setup.py` scripts for some packages that build extensions, such as setting the `PREFIX` . You may need to just extract the distribution and use `distutils` to install the package the old-school way by typing `python setup install [options]` . Reading some of the install documentation and the `distutils` documentation may help.

Python is nice enough to add `site.USERBASE` to your `PYTHONPATH` ahead of anything else, so the changes will only effect you. A popular location for `--home` is `~/.local` . See the Python module installation guide for the exact file structure and specifically where your site-packages are. *Note*: if you use the `--home` installation scheme then you may need to add it to the `PYTHONPATH` environment variable using `export` in your `.bashrc` , `.bash_profile` or in your shell for your localized pacakges to be available in Python.

Use Continuum Anaconda Python for Personal Projects

Another relatively new option that may be available to you is Anaconda Python by Continuum. Anaconda only installs into your personal profile, *ie:* `/home/<user>/` and alters your `~/.bashrc` or `~/.bash_profile` to prepend Anaconda's path to your personal `$PATH` - **this only affects *you* - your system path is unchanged**. Therefore **you do *not* need root access or `sudo` to use Anaconda!** You can reverse this behavior by removing the new path export to Anaconda from your `~/.bashrc` or `~/.bash_profile` and then your system Python will be first again.

This is somewhat similar to the `--user` option I explained in the last section except it applies to Python as a whole and not just packages. Therefore Anaconda is **completely separate from your system Python**, it won't interfere with your system Python and only you can use or change it. Since it installs a new version of Python and all of its libraries you will need at least 200MB of room, but it is very clever about caching and managing libraries which is important for some of the cool things you can do with Anaconda.

Continuum curates a collection of Python binaries and libraries required by dependencies in an online repository called Anaconda binstar. The package manager used by Anaconda, `conda` , is used to install packages from Anaconda binstar. But `conda` can do so much more than just install packages, it also can create and manage virtual environments just like `virtualenv` . Therefore since Anaconda creates virtual environments, the `pip` package manager can be used to install packages from PyPI in an Anaconda environment without root or `sudo` . Do **not** use `sudo` with Anaconda!

Anaconda is also similar in some ways to Ruby RVM if you're familiar with that tool. Anaconda `conda` also lets you create virtual environments with ***different* versions of Python**. *EG*: `conda create -n py35sci python==3.5.2 numpy scipy matplotlib pandas statsmodels seaborn` will create a scientific/data-science stack using Python-3.5 in a new environment called `py35sci` . You can switch environments similar to `virtualenv` by sourcing the `activate` script:

```
~/Projects/myproj $ source activate py35sci
```

But wait there's more! Anaconda can also install different languages such as R for statistical programming from the `r` channel. In addition, Anaconda divides binstar into different *channels* which avoids names clashes. You can even set up your own channel to upload

package distributions using the conda build protocol. In fact there is a channel called conda-forge by conda-forge that maintains automated builds of many of the packages on PyPI.

Epilogue

There are many options for maintaining your Python projects on Linux depending on your personal needs and access. However, if there's any one thing I hope you take away from this answer is that **you should almost never need to use `sudo` to install Python packages**. The use of `sudo` should be a *smell* to you that something is amiss. You have been warned.

Good luck and happy coding!

edited Jan 27 at 16:18

answered Mar 28 '12 at 7:04

Mark Mikofski
832 8 17

I wish I'd read this before I destroyed my system's Python distribution by `sudo` ing around with `pip` . Great writeup, thanks for this information. – slhck Mar 27 '17 at 12:23

In addition to Novarchibald's addition, it is generally a good idea to create a virtual environment for your python project and install dependencies inside. This allows you to have better control over dependencies and their versions. To set up a virtual environment, enter:

```
virtualenv --no-site-packages --distribute my_python_project
```

Then, activate it with:

```
cd my_python_project
source bin/activate
```

At this point anything that you install with pip will be kept inside this virtual environment. If you want to install something globally, then you should first exit virtualenv with:

```
deactivate
```

You can find out more about virtualenv here.

answered Jan 20 '12 at 9:59

Voitek Zylinski
515 3 6

1 Alternatively: using the pip from the virtualenv bin directory (here my_python_project/bin/pip) you maintain that environment, without a need to "activate" it. Then - using any of scripts installed in the bin is using given virutalenv. "activate" is only handy to make calls to "python", "easy_install" and "pip" using virtualenv specific bin. – Jan Vlcinsky Jun 17 '13 at 11:54

Python's install docs say to use "pyvenv" to create virtual environments specific to a project; but that virtualenv is a fall back for projects using older python version, docs.python.org/3/installing/index.html?highlight=pip. – pbhj Nov 25 '14 at 1:03

2 Note that the `virtualenv` options `--no-site-packages` and `--distribute` are deprecated nowadays and have no effect any longer. – Forage Aug 8 '15 at 14:41

In addition to Zetah'a answer, command to install python-pip from terminal is:

```
sudo apt-get install python-pip
```

answered Jan 20 '12 at 8:08

Novarchibald
93 1 1 6
