Pip: PyPI Package Manager

All Versions

Examples

Install Packages

To install the latest version of a package named SomePackage :

```
$ pip install SomePackage
```

To install a specific version of a package:

```
$ pip install SomePackage==1.0.4
```

To specify a minimum version to install for a package:

```
$ pip install SomePackage>=1.0.4
```

If commands shows permission denied error on Linux/Unix then use sudo with the commands

---

Install from requirements files

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

Each line of the requirements file indicates something to be installed, and like arguments to pip install, Details on the format of the files are here: Requirements File Format .

After install the package you can check it using freeze command:

```
$ pip freeze
```

To list all packages installed using `pip`

To list installed packages:

```
$ pip list
# example output
docutils (0.9.1)
Jinja2 (2.6)
Pygments (1.5)
Sphinx (1.1.2)
```

To list outdated packages, and show the latest version available:

```
$ pip list --outdated
# example output
docutils (Current: 0.9.1 Latest: 0.10)
Sphinx (Current: 1.1.2 Latest: 1.1.3)
```

Upgrade Packages

Running

```
$ pip install --upgrade SomePackage
```

will upgrade package SomePackage and all its dependencies. Also, pip automatically removes older version of the package before upgrade.

To upgrade pip itself, do

```
$ pip install --upgrade pip
```

on Unix or

```
$ python -m pip install --upgrade pip
```

on Windows machines.

Create a requirements.txt file of all packages on the system

pip assists in creating requirements.txt files by providing the freeze option.

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

This will save a list of all packages and their version installed on the system to a file named requirements.txt in the current folder.

Uninstall Packages

To uninstall a package:

```
$ pip uninstall SomePackage
```

Updating all outdated packages on Linux

pip doesn't current contain a flag to allow a user to update all outdated packages in one shot. However, this can be accomplished by piping commands together in a Linux environment:

```
pip list --outdated --local | grep -v '^\-e' | cut -d = -f 1  | xargs -n1 pip install -U
```

This command takes all packages in the local virtualenv and checks if they are outdated. From that list, it gets the package name and then pipes that to a pip install -U command. At the end of this process, all local packages should be updated.

Updating all outdated packages on Windows

pip doesn't current contain a flag to allow a user to update all outdated packages in one shot. However, this can be accomplished by piping commands together in a Windows environment:

```
for /F "delims= " %i in ('pip list --outdated --local') do pip install -U %i
```

This command takes all packages in the local virtualenv and checks if they are outdated. From that list, it gets the package name and then pipes that to a pip install -U command. At the end of this process, all local packages should be updated.

Create a requirements.txt file of packages only in the current virtualenv

pip assists in creating requirements.txt files by providing the freeze option.

```
pip freeze --local > requirements.txt
```

The --local parameter will only output a list of packages and versions that are installed locally to a virtualenv. Global packages will not be listed.

Using a certain Python version with pip

If you have both Python 3 and Python 2 installed, you can specify which version of Python you would like pip to use. This is useful when packages only support Python 2 or 3 or when you wish to test with both.

If you want to install packages for Python 2, run either:

```
pip install [package]
```

or:

```
pip2 install [package]
```

If you would like to install packages for Python 3, do:

```
pip3 install [package]
```

You can also invoke installation of a package to a specific python installation with:

```
\path\to\that\python.exe -m pip install some_package # on Windows OR
/usr/bin/python25 -m pip install some_package # on OS-X/Linux
```

On OS-X/Linux/Unix platforms it is important to be aware of the distinction between the system version of python, (which upgrading make render your system inoperable), and the user version(s) of python. You **may** , *depending on which you are trying to upgrade* , need to prefix these commands with `sudo` and input a password.

Likewise on Windows some python installations, especially those that are a part of another package, can end up installed in system directories - those you will have to upgrade from a command window running in Admin mode - if you find that it looks like you need to do this it is a **very** good idea to check which python installation you are trying to upgrade with a command such as `python -c"import sys;print(sys.path);"` or `py -3.5 -c"import sys;print(sys.path);"` you can also check which pip you are trying to run with `pip --version`

On Windows, if you have both python 2 and python 3 installed, and on your path and your python 3 is greater than 3.4 then you will probably also have the python launcher `py` on your system path. You can then do tricks like:

```
py -3 -m pip install -U some_package # Install/Upgrade some_package to the latest python 3
py -3.3 -m pip install -U some_package # Install/Upgrade some_package to python 3.3 if present
py -2 -m pip install -U some_package # Install/Upgrade some_package to the latest python 2 - 64
py -2.7-32 -m pip install -U some_package # Install/Upgrade some_package to python 2.7 - 32 bit
```

If you are running & maintaining multiple versions of python I would strongly recommend reading up about the python `virtualenv` or `venv` virtual enviroments which allow you to isolate both the version of python and which packages are present.

Installing packages not yet on pip as wheels

Many, pure python, packages are not yet available on the Python Package Index as wheels but still install fine. However, some packages on Windows give the dreaded vcvarsall.bat not found error.

The problem is that the package that you are trying to install contains a C or C++ extension and is not *currently* available as a pre-built wheel from the python package index, *pypi* , and on windows you do not have the tool chain needed to build such items.

The simplest answer is to go to Christoph Gohlke's excellent site and locate the **appropriate** version of the libraries that you need. By appropriate in the package name a **-cp *NN*-** has to match your version of python, i.e. if you are using windows 32 bit python *even on win64* the name must include **-win32-** and if using the 64 bit python it must include **-win_amd64-** and then the python version must match, i.e. for Python 34 the filename **must** include **-cp *34*-** , etc. this is basically the magic that pip does for you on the pypi site.

Alternatively, you need to get the appropriate windows development kit for the version of python that you are using, the headers for any library that the package you are trying to build interfaces to, possibly the python headers for the version of python, etc.

Python 2.7 used Visual Studio 2008, Python 3.3 and 3.4 used Visual Studio 2010, and Python 3.5+ uses Visual Studio 2015.

- Install " Visual C++ Compiler Package for Python 2.7 ", which is available from Microsoft's website **or**
- Install " Windows SDK for Windows 7 and .NET Framework 4 " (v7.1), which is available from Microsoft's website **or**
- Install Visual Studio 2015 Community Edition , *(or any later version, when these are released)* , **ensuring you select the options to install C & C++ support** *no longer the default - I am told that this can take up to **8 hours*** to download and install so make **sure** that those options are set on the first try.

Then you *may need* to locate the header files, *at the matching revision* for any libraries that your desired package links to and download those to an appropriate locations.

Finally you can let pip do your build - of course if the package has dependencies that you don't yet have you may also need to find the header files for them as well.

Alternatives: It is also worth looking out, *both on pypi or Christop's site* , for any slightly earlier version of the package that you are looking for that is either pure python or pre-built for your platform and python version and possibly using those, if found, until your package does become available. Likewise if you are using the very latest version of python you may find that it takes the package maintainers a little time to catch up so for projects that really **need** a specific package you may have to use a slightly older python for the moment. You can also check the packages source site to see if there is a forked version that is available pre-built or as pure python and searching for alternative packages that provide the functionality that you require but are available - one example that springs to mind is the Pillow , *actively maintained* , drop in replacement for PIL *currently not updated in 6 years and not available for python 3* .

Afterword , I would encourage anybody who is having this problem to go to the bug tracker for the package and add to, or raise if there isn't one already, a ticket **politely** requesting that the package maintainers provide a wheel on pypi for your specific combination of platform and python, if this is done then normally things will get better with time, some package maintainers don't realise that they have missed a given combination that people may be using.

Syntax

```
pip <command> [options] where <command> is one of:
 install
  Install packages

 uninstall
  Uninstall packages

 freeze
  Output installed packages in requirements format

 list
  List installed packages

 show
  Show information about installed packages

 search
  Search PyPI for packages

 wheel
  Build wheels from your requirements

 zip
  Zip individual packages (deprecated)

 unzip
  Unzip individual packages (deprecated)

 bundle
  Create pybundles (deprecated)

 help
  Show help for commands
```

Parameters

Remarks

Sometimes, `pip` will perfom a manual compilation of native code. On Linux python will automatically choose an available C compiler on your system. Refer to the table below for the required Visual Studio/Visual C++ version on Windows (newer versions will not work.).

Python Version	Visual Studio Version	Visual C++ Version
2.6 - 3.2	Visual Studio 2008	Visual C++ 9.0
3.3 - 3.4	Visual Studio 2010	Visual C++ 10.0
3.5	Visual Studio 2015	Visual C++ 14.0

Source: wiki.python.org