Installing packages using pip and virtual environments

This guide discusses how to install packages using pip and a virtual environment manager: either venv for Python 3 or virtualenv for Python 2. These are the lowest-level tools for managing Python packages and are recommended if higher-level tools do not suit your needs.

Note: This doc uses the term **package** to refer to a Distribution Package which is different from a Import Package that which is used to import modules in your Python source code.

Installing pip

pip is the reference Python package manager. It's used to install and update packages. You'll need to make sure you have the latest version of pip installed.

Windows

The Python installers for Windows include pip. You should be able to access pip using:

```
py -m pip --version
pip 9.0.1 from c:\python36\lib\site-packages (Python 3.6.1)
```

You can make sure that pip is up-to-date by running:

```
py -m pip install --upgrade pip
```

Linux and macOS

Debian and most other distributions include a python-pip package, if you want to use the Linux distribution-provided versions of pip see Installing pip/setuptools/wheel with Linux Package Managers.

You can also install pip yourself to ensure you have the latest version. It's recommended to use the system pip to bootstrap a user installation of pip:

v: latest

```
python3 -m pip install --user --upgrade pip
```

Afterwards, you should have the newest pip installed in your user site:

```
python3 -m pip --version
pip 9.0.1 from $HOME/.local/lib/python3.6/site-packages (python 3.6)
```

Installing virtualenv

Note: If you are using Python 3.3 or newer, the `venv` module is the preferred way to create and manage virtual environments. venv is included in the Python standard library and requires no additional installation. If you are using venv, you may skip this section.

virtualenv is used to manage Python packages for different projects. Using virtualenv allows you to avoid installing Python packages globally which could break system tools or other projects. You can install virtualenv using pip.

On macOS and Linux:

```
python3 -m pip install --user virtualenv
```

On Windows:

```
py -m pip install --user virtualenv
```

Creating a virtual environment

venv (for Python 3) and virtualenv (for Python 2) allow you to manage separate package installations for different projects. They essentially allow you to create a “virtual” isolated Python installation and install packages into that virtual installation. When you switch projects, you can simply create a new virtual environment and not have to worry about breaking the packages installed in the other environments. It is always recommended to use a virtual environment while developing Python applications.

To create a virtual environment, go to your project’s directory and run venv. If you are using Python 2, replace `venv` with `virtualenv` in the below commands.

On macOS and Linux:

```
python3 -m venv env
```

On Windows:

```
py -m venv env
```

The second argument is the location to create the virtual environment. Generally, you can just create this in your project and call it `env`.

venv will create a virtual Python installation in the `env` folder.

> **Note:** You should exclude your virtual environment directory from your version control system using `.gitignore` or similar.

Activating a virtual environment

Before you can start installing or using packages in your virtual environment you'll need to *activate* it. Activating a virtual environment will put the virtual environment-specific `python` and `pip` executables into your shell's `PATH`.

On macOS and Linux:

```
source env/bin/activate
```

On Windows:

```
.\env\Scripts\activate
```

You can confirm you're in the virtual environment by checking the location of your Python interpreter, it should point to the `env` directory.

On macOS and Linux:

```
which python
.../env/bin/python
```

On Windows:

```
where python
.../env/bin/python.exe
```

As long as your virtual environment is activated pip will install packages into that specific environment and you'll be able to import and use packages in your Python application.

Leaving the virtual environment

If you want to switch projects or otherwise leave your virtual environment, simply run:

```
deactivate
```

If you want to re-enter the virtual environment just follow the same instructions above about activating a virtual environment. There's no need to re-create the virtual environment.

Installing packages

Now that you're in your virtual environment you can install packages. Let's install the Requests library from the Python Package Index (PyPI):

```
pip install requests
```

pip should download requests and all of its dependencies and install them:

```
Collecting requests
  Using cached requests-2.18.4-py2.py3-none-any.whl
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Using cached chardet-3.0.4-py2.py3-none-any.whl
Collecting urllib3<1.23,>=1.21.1 (from requests)
  Using cached urllib3-1.22-py2.py3-none-any.whl
Collecting certifi>=2017.4.17 (from requests)
  Using cached certifi-2017.7.27.1-py2.py3-none-any.whl
Collecting idna<2.7,>=2.5 (from requests)
  Using cached idna-2.6-py2.py3-none-any.whl
Installing collected packages: chardet, urllib3, certifi, idna, requests
Successfully installed certifi-2017.7.27.1 chardet-3.0.4 idna-2.6 requests-2.18.4 urllib
```

Installing specific versions

pip allows you to specify which version of a package to install using version specifiers. For example, to install a specific version of `requests`:

```
pip install requests==2.18.4
```

To install the latest `2.x` release of requests:

```
pip install requests>=2.0.0,<3.0.0
```

To install pre-release versions of packages, use the `--pre` flag:

```
pip install --pre requests
```

Installing extras

Some packages have optional extras. You can tell pip to install these by specifying the extra in brackets:

```
pip install requests[security]
```

Installing from source

pip can install a package directly from source, for example:

```
cd google-auth
pip install .
```

Additionally, pip can install packages from source in development mode, meaning that changes to the source directory will immediately affect the installed package without needing to re-install:

```
pip install --editable .
```

Installing from version control systems

pip can install packages directly from their version control system. For example, you can install directly from a git repository:

```
git+https://github.com/GoogleCloudPlatform/google-auth-library-python.git#egg=google-aut
```

For more information on supported version control systems and syntax, see pip's documentation on VCS Support.

Installing from local archives

If you have a local copy of a Distribution Package's archive (a zip, wheel, or tar file) you can install it directly with pip:

```
pip install requests-2.18.4.tar.gz
```

If you have a directory containing archives of multiple packages, you can tell pip to look for packages there and not to use the Python Package Index (PyPI) at all:

```
pip install --no-index --find-links=/local/dir/ requests
```

This is useful if you are installing packages on a system with limited connectivity or if you want to strictly control the origin of distribution packages.

Using other package indexes

If you want to download packages from a different index than the Python Package Index (PyPI), you can use the `--index-url` flag:

```
pip install --index-url http://index.example.com/simple/ SomeProject
```

If you want to allow packages from both the Python Package Index (PyPI) and a separate index, you can use the `--extra-index-url` flag instead:

```
pip install --extra-index-url http://index.example.com/simple/ SomeProject
```

Upgrading packages

pip can upgrade packages in-place using the `--upgrade` flag. For example, to install the latest version of `requests` and all of its dependencies:

```
pip install --upgrade requests
```

Using requirements files

Instead of installing packages individually, pip allows you to declare all dependencies in a Requirements File. For example you could create a `requirements.txt` file containing:

```
requests==2.18.4
google-auth==1.1.0
```

And tell pip to install all of the packages in this file using the `-r` flag:

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

Freezing dependencies

Pip can export a list of all installed packages and their versions using the `freeze` command:

```
pip freeze
```

Which will output a list of package specifiers such as:

```
cachetools==2.0.1
certifi==2017.7.27.1
chardet==3.0.4
google-auth==1.1.1
idna==2.6
pyasn1==0.3.6
pyasn1-modules==0.1.4
requests==2.18.4
rsa==3.4.2
six==1.11.0
urllib3==1.22
```

This is useful for creating Requirements Files that can re-create the exact versions of all packages installed in an environment.