파이썬 초심자를 위한 PIP 그리고 Virtualenv 소개

Dan Kim Follow
May 30, 2018 · 13 min read

이 글은 **A non-magical introduction to Pip and Virtualenv for Python beginners**를 번역한 글입니다.
파이썬의 신규 개발자들이 반드시 넘어야할 산중 하나는 파이썬 패키징 생태계를 이해하는 것입니다. 이 글에서는 **Python for Programming training course**에서 다루었던 자료를 기반으로 파이썬 초심자들을 위해 **pip** 와 **virtualenv** 를 설명하고자 합니다.

전제 조건

Python for Programmers 강의는 하나 이상의 개발 언어에 익숙한 개발자들을 대상으로 한 강의였기 때문에, 이 글은 여러분이 어느 정도 수준의 기술 지식을 가지고 있다고 가정하고 있습니다. 커맨드 라인 사용에 대해 편하게 느낀다면 이 글을 쉽게 이해하는데 도움이 될것입니다. 이 글에서 예제들은 맥과 리눅스 환경에서 기본으로 사용되는 shell인 **bash** 를 사용할 예정입니다. 하지만, 사용할 명령어들이 매우 간단하기 때문에 윈도우의 PowerShell에서도 동일한 개념의 명령어를 쉽게 찾아서 사용하실 수 있을 겁니다.

PIP

이제 시작해봅시다. **pip**는 **Python Package Index** (PyPI) 저장소로부터 파이썬 패키지를 받아 설치하는 패키지 관리 도구입니다. PyPI (가끔 **The Cheeseshop** 이라고도 불리는)는 third-party 파이썬 오픈소스 패키지들을 위한 저장소입니다. Ruby에서의 RubyGems 혹은 PHP의 Packagist, Perl의 CPAN 그리고 Node.js의 NPM와 비슷하다고 생각하시면 됩니다.
사실 Python은 훨씬 더 기초적인 패키지 매니저인 easy_install을 가지고 있습니다. 여러분이 처음 Python을 설치할때 easy_install은 자동으로 설치됩니다. 하지만, 여러 측면에서 easy_install 보다 훨씬 더 우월한 pip를 사용하는것이 일반적입니다. 먼저, easy_install을 통해 pip를 다음과 같이 설치할 수 있습니다:

```
$ sudo easy_install pip
```

이제 pip를 통해 패키지들을 설치할 수 있습니다. (아래 예제에서는 **Django**를 설치해봅시다):

```
# DON'T DO THIS
$ sudo pip install django
```

여기서 우리는 Django를 시스템 전체에 (global) 설치하였습니다. 하지만, 대부분의 경우 패키지를 **글로벌하게 설치하지 않아야합니다**. 계속해서 읽어보면서 이유를 찾아보세요.

Virtualenv

virtualenv 는 아주 구체적인 문제를 해결합니다: 보통 여러개의 파이썬 프로젝트가 하나의 컴퓨터에서 충돌을 일으키지 않고 존재할 수 있도록 도와줍니다.

어떤 문제를 해결하나요?

virtualenv가 해결하는 문제를 설명하기 위해, 먼저 virtualenv가 존재하지 않는다고 가정해봅시다. 여러분은 외부 웹 서버에 HTTP 요청을 보내는 파이썬 프로그램을 작성해야하는 상황입니다.

령어를 실행시키면 다음과 같은 일이 발생합니다.

```
$ pip install requests
 Downloading/unpacking requests
     Downloading requests-1.0.0.tar.gz (337Kb): 337Kb downloaded
     Running setup.py egg_info for package requests

 Installing collected packages: requests
     Running setup.py install for requests

     error: could not create '/Library/Python/2.7/site-
packages/requests': Permission denied
```

pip가 `/Library/Python/2.7/site-packages/requests` 안에 패키지를 설치하려고 했던 것으로 보입니다. 이 폴더는 파이썬이 알고있는 **특별한 폴더**입니다. `site-packages` 내부에 패키지가 설치되면, 파이썬 프로그램에서 이 패키지를 임포트해서 사용할 수 있게됩니다.
하지만, 위에서는 에러가 발생했습니다. 일반적으로 맥에서 **"일반 사용자"**는 `/Library` 폴더에 쓰기 권한이 없기 때문에 위와 같은 에러가 발생한것입니다. 이 에러를 고치기 위해서는 `sudo pip install requests` 명령어를 실행하면 됩니다. 여기에서 sudo는 **"슈퍼 유저"**로 명령어를 실행하라는 의미입니다.

```
$ sudo pip install requests
 Password:
 Downloading/unpacking requestss
     Running setup.py egg_info for package requests

 Installing collected packages: requests
     Running setup.py install for requests

 Successfully installed requests
 Cleaning up ...
```

이제 정상적으로 동작합니다. **python** 명령어를 실행해서 Requests 라이브러리를 임포트해봅시다:

```
>>> import requests
>>> requests.get('http://dabapps.com')
<Response [200]>
```

지금까지 우리는 pip를 통해 라이브러리를 설치하고, 이 패키지를 `import requests` 구문을 통해 파이썬 프로그램 내에서 사용해보았습니다. 시간이 지나서, PyPI에서 다른 라이브러리들을 가져와서 우리의 앱을 계속 발전시켰다고 가정해봅시다. 시간이지나 우리가 발전시킨 앱은 뛰어난 성능을 보여주면서 많은 돈을 벌 수 있었고, 사용자들은 약간의 차이가 있는 새로운 프로그램을 작성해달라는 요청을 보내고 있습니다.
사용자들의 요청에 보답하여 새로운 앱을 만들어보고자 새로운 프로젝트를 시작했습니다. 이번 앱에도 requests 라이브러리가 필요한데, 첫번째 앱을 만들었던 이후로 requests 라이브러리에는 새로운 기능이 추가된것을 확인했습니다. 이 추가된 기능이 새로 앱에서 필수적인 상황입니다. 새

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

모든 것이 좋아보이지만, 우리가 알지 못하는 사이에 재앙은 발생하고 있습니다! 업데이트 이후, 우리에게 돈을 많이 벌어다주던 기존의 프로그램을 돌려보자 프로그램이 에러가 납니다. 왜일까요? 바로 **requests 라이브러리가 변경**되었기 때문입니다. 아주 작은 변화였지만, 우리의 코드가 더 이상 이 라이브러리를 못쓰는 상황이 벌어진 것입니다. 모든 것이 망가져 버렸습니다.
물론, 새로운 **requests API**를 사용하도록 기존의 프로그램을 변경해 문제를 해결할 수 있습니다. 하지만, 시간이 소요되고 새로운 프로젝트에서 집중도를 잃게 됩니다. 보통 숙련된 파이썬 개발자는 단지 2개의 프로젝트만 가지고 있지 않습니다. 수십가지 프로젝트를 동시에 진행하고, 각 프로젝트는 수십가지의 라이브러리에 대한 의존성을 가지고 있습니다! 이 라이브러리 전체를 최신버전으로 유지하고 모든 프로젝트에서 동일한 버전의 라이브러리를 사용하도록 만드는 것은 완전 악몽입니다.

어떻게 virtualenv는 이 문제를 해결하나요?

virtualenv는 각 프로그램별로 완전히 독립적인 가상의 환경을 만들어냄으로써 이 문제를 해결합니다. 여기서 환경이란 파이썬 프로그램을 실행시키는데 필요한 모든것의 복사본을 가지고 있는 단순한 폴더입니다. 전체 파이썬 스탠다드 라이브러리 복사본, pip 설치 프로그램 복사본, 그리고 위에서 언급한 site-packages 복사본 등을 포함합니다. 여러분이 virtualenv 도구를 이용해 생성된 pip 복사본을 이용해 PyPI로 부터 패키지를 설치하면, **virtualenv 폴더 내부의 site-packages 폴더에 이를 설치**합니다. 그리고 설치된 패키지는 이전과 동일한 방법으로 파이썬 프로그램 내부에서 사용할 수 있습니다.

어떻게 virtualenv를 설치하나요?

만약 여러분이 이미 pip를 설치하셨다면, virtualenv를 설치하는 가장 쉬운 방법은 `sudo pip install virtualenv` 명령어 입니다. pip와 virtualenv는 일반적으로 글로벌 설치가 되어야하는 유일한 패키지입니다. 이 두개를 설치하고 나면 나머지 패키지들은 가상 환경에 설치하면 되기 때문입니다.
사실, virtualenv는 pip의 복사본을 수반하고 있습니다. 그렇기에 여러분이 필요한것은 **사실상 virtualenv 뿐**입니다. virtualenv는 PyPi에서 설치하는 것이 아닌 독립적인 패키지로 설치가 가능합니다. 이런 방법은 윈도우 사용자들에게 더 쉬울 수 있습니다. **virtualenv.org** 에 있는 설치 안내를 참고해주세요.

어떻게 새로운 가상 환경을 생성할 수 있나요?

새로운 환경을 만들기 위해서는 **virtualenv**만 있으면 됩니다. 아주 간단하죠. 여러분 프로젝트의 루트 폴더로 이동한 이후, 아래와 같이 **virtualenv** 명령어로 생성하면 됩니다.

```
$ cd ~/code/myproject
$ virutalenv env
 New python executable in /env/bin/python
 Installing setuptools ..... done.
 Installing pip .... done.
```

여기에서, `env` 는 여러분의 가상 환경을 생성할 폴더의 이름입니다. 아무런 이름을 붙여도 되지만, 보통 `env` 라고 부르고 프로젝트 디렉토리 내부에 이를 만드는 것이 일반적인 관습입니다. 예를 들어, 여러분이 코드를 `~/code/projectname/` 에 보관한다고 하면, `~/code/projectname/env` 가 새롭게 만들어진 환경이라고 생각하시면 됩니다. 하지만, 여러분이 원하는 폴더 이름으로 지으셔도 상관없고, 프로젝트 폴더 외부에 위치시켜도 상관 없습니다.

> *만약 여러분이 git과 같은 버전 컨트롤 시스템을 사용하고 계신다면, env 디렉토리를 커밋에 포함시키지않는 것을 추천합니다. 반드시 .gitignore 파일에 env 디렉토리를 추가하세요.*

```
$ ls env
bin include lib
```

여러분이 가장 관심을 기울여야하는 폴더는 bin 입니다. 이 폴더는 파이썬 라이브러리의 로컬 복사본과 pip 설치 복사본이 있는 곳입니다. 이제 pip 복사본을 이용해서 vitrualenv 내부에 requests 라이브러리를 설치해봅시다.

```
$ env/bin/pip install requests
 Downloading/unpacking requests
     Downloading requests-1.1.0.tar.gz (337Kb): 337kB downloaded
     Running setup.py egg_info for package requests

 Installing collected packages: requests
     Running setup.py install for requests

 Successfully installed requests
 Cleaning up...
```

동작하는군요! 여기에서 여러분이 **sudo를 사용하지 않았다는 점**을 기억하세요. 이제 requests를 글로벌로 설치하지 않고 home 폴더내부에 설치하기 때문에, 더이상 sudo가 필요하지 않은것입니다.
이제, Python shell을 실행하기 위해 pytho 명령어를 실행하지 않고, env/bin/python 명령어를 실행하면 됩니다.

```
>>> import requests
>>> requests.get('http://dabapps.com')
<Response [200]>
```

하지만, 너무 타이핑을 많이해야해요

virtualenv는 숨겨진 다양한 트릭들을 가지고 있습니다. env/bin/python 그리고 env/bin/pip 를 매번 입력하는 대신, 우리가 만든 가상 환경을 실행시킬 수 있습니다. source env/bin/activate 명령어를 통해서 활성화시키면 이 스크립트는 여러분의 shell의 몇몇 변수들을 일시적으로 조정합니다. 그래서, python 을 타이핑하면 글로벌 python 대신 virtualenv 내부에 있는 Python 실행파일을 얻을 수 있게 됩니다:

```
$  which python
 /usr/bin/python
$ source env/bin/activate
$ which python
 /Users/jamie/code/myproject/env/bin/python
```

이제 여러분이 env/bin/pip install requests 를 입력하는 대신에 pip install requests 라고 입력하면 글로벌이 아닌 우리의 가상환경에 라이브러리를 설치하게 됩니다. 스크립트가 터미널에 준 이 변화는 터미널이 열려있는 동안만 유지가 됩니다. 때문에 여러분이 새로운 터미널을 실행시키면 다시 source env/bin/activate 스크립트를 실행시켜야합니다. 다른 프로젝트로 이동하고 싶다면 deactivate 명령어를 통해 현재 가상환경의 사용을 종료시킬 수 있습니다. 이후, 해당 프로젝트로 이동해 source /env/bin/activate 명령어를 실행시키면 됩니다.

pip의 requrements 기능을 사용할때 virtualenv와 pip는 서로 아주 좋은 동료가 됩니다. 여러분의 각 작업 프로젝트는 각기 자기들만의 requirements.txt 파일을 가지고 있습니다. 이 파일을 가상환경에서 필요한 라이브러리들을 설치하는데 사용할 수 있습니다:

```
env/bin/pip install -r requirements.txt
```

더 자세한 정보는 pip 문서를 참고해주세요.

정리

- pip는 Python Package Index에서 패키지를 설치하는 도구입니다.
- virtualenv는 **python, pip, PyPI** 부터 설치된 라이브러리들의 복사본을 만듦으로써, 독립적인 파이썬 환경을 만들어주는 도구입니다.
- 이는 여러분이 동시에 하나의 기기에서 여러개의 프로젝트가 가지는 다른 의존성을 다룰 수 있도록 디자인되었습니다.
- 설치 관련 가이드는 **virtualeng.org**에서 찾을 수 있습니다.
- 설치 후에, env라는 가상환경 폴더를 만들고 싶다면 `virtualenv env` 명령어를 실행시키면 됩니다.
- 여러분의 각 프로젝트 마다 이러한 환경이 하나씩 필요하게 됩니다. 이 폴더들을 버전 컨트롤 시스템에서 제외하는것을 잊지 마세요.
- 가상환경에서 python과 pip를 사용하고 싶다면, `env/bin/python` 그리고 `env/bin/pip` 를 사용하면 됩니다.
- 가상환경을 `source env/bin/activate` 스크립트로 활성화 시킬 수 있고 `deactivate` 명령어를 통해 비활성화 시킬 수 있습니다. 선택적이지만 개발을 조금 더 편하게 도와줄 것입니다.

만일 여러분이 일반적인 파이썬 사용자라면 pip와 virtualenv는 서로 떨어질 수 없는 도구들일것입니다. 둘다 상대적으로 이해하기 간단하기에 이들을 명확히 이해하는 것을 추천합니다.
만약 이 글이 파이썬을 배우고 싶도록 여러분을 자극했다면 **Python for Programmers** 강의를 확인해보세요.

Python Pip Virtualenv Beginner

WRITTEN BY
Dan Kim

More From Medium

Related reads | Also tagged Virtualenv | Also tagged Python

Top Korean Games of 2018

Naom... Jan 7 ·... 21

A Virtual World for Your Project

Rojesh... May 27 ·... 39

Satellite imagery access and analysis in Python & Jupyter notebooks

Abdishak... Jul 1 · 7... 10