# **pipenv快速入门**

<https://www.jianshu.com/p/dcbb01e8c78a>

学过Python的同学应该都了解pip这个工具，我们用pip绝大部分的第三方库都可以用pip来安装，用起来很方便。但是如果我们要把项目部署到服务器上面的话，就稍微有些麻烦了，因为还需要在服务器上用pip安装这些包，假如项目中用到很多包的话，一个个安装会很麻烦，而且没有通用性。Java上的maven、gradle，NodeJS的npm这些工具就不存在这个问题，它们有一个或多个的专门的依赖文件来管理这些包。pipenv就是这样一个类似的工具，可以帮助我们管理Python和第三方库的版本。

## 安装

安装pipenv很简单，用pip命令就可以安装。

pip install pipenv

将来需要更新pipenv的时候，运行：

pip install --user --upgrade pipenv

### 首次运行

如果是第一次在项目中运行pipenv命令的话，会在项目中创建一个名为Pipfile的文件，文件内容类似下面这样。用过maven、gradle等工具的同学对此应该熟悉，相信不用我解释其中的含义。

[[source]]

url = "https://pypi.org/simple"

verify\_ssl = true

name = "pypi"

[packages]

requests-html = "\*"

[dev-packages]

[requires]

python\_version = "3.7"

如果运行过install、update等命令的话，还会创建一个Pipfile.lock文件，类似npm中的lock文件。这两个文件就是pipenv用于管理第三方库的配置文件，如果同时使用版本控制软件的话，需要将它们也加入进去。

## 常用命令

### 安装

例如，我想在项目中安装requests这个包，运行：

pipenv install requests

如果需要指定具体版本号，可以这样：

pipenv install requests==2.13.0

如果是第一次运行pipenv的话，会先创建Pipfile文件，否则会修改Pipfile`文件。

该命令还有一个常用参数-d或--dev，用于安装仅供开发使用的包。

### 卸载

相应的还有命令来卸载第三方包，该命令还有两个参数--all和--all-dev用于卸载所有包和所有开发包。

pipenv uninstall requests

### 更新

查看所有需要更新的包：

pipenv update --outdated

更新所有包：

pipenv update

更新指定的包：

pipenv update <包名>

### 从requirements.txt导入

如果项目中有requirements.txt文件，pipenv会在安装的时候自动导入。如果需要导入其他位置的requirements.txt，可以用下面的命令：

pipenv install -r path/to/requirements.txt

### 指定Python版本

pipenv会创建虚拟Python环境，并在其中用pip安装所有包。如果要指定Python版本，可以用下面的命令，三种版本号都支持：

pipenv --python 3

pipenv --python 3.6

pipenv --python 2.7.14

如果不指定版本号，pipenv会使用系统默认的Python版本。需要注意，这里指定的Python必须是系统已经安装的、可以在环境变量中搜索到的版本号，如果指定未安装的版本，会提示错误。

### 运行命令

用下面的命令可以启动一个在虚拟环境中的shell：

pipenv shell

如果不想启动shell，而是直接在虚拟环境中执行命令，可以使用run：

pipenv run python --version

## 高级用法

一开始我文档没看全，然后用pipenv的时候发现有一些问题，后来我发现官方文档还有一部分高级内容也很重要，所以再来补充一下。当然如果有需要的话还是得看[原文](https://pipenv.readthedocs.io/en/latest/" \t "_blank)。

### 导出requirements.txt

用下面的命令就可以将Pipfile和Pipfile.lock文件里面的包导出为requirements.txt文件。

pipenv lock -r

如果只想导出开发用的包，可以添加--dev参数：

pipenv lock -r --dev

### 自动安装Python

pipenv只能搜索系统中已经安装的Python版本，对于未安装的版本，会提示错误。但是如果你同时安装了pyenv的话，pipenv会自动发现pyenv，然后直接询问你是否要安装。这样一来，原来的工作流程是：用pyenv安装某个Python->用virtualenv或venv创建虚拟环境->用pip从requirements.txt中安装包->将来可能还要更新包。现在完全可以用pipenv一两条命令解决，真的是非常方便。

### 自动加载.env文件

.env文件可以设置一些环境变量，在程序开发的时候模拟环境变量。pipenv也可以自动加载.env文件。

$ cat .env

HELLO=WORLD⏎

$ pipenv run python

Loading .env environment variables…

Python 2.7.13 (default, Jul 18 2017, 09:17:00)

[GCC 4.2.1 Compatible Apple LLVM 8.1.0 (clang-802.0.42)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>> import os

>>> os.environ['HELLO']

'WORLD'

### 环境变量支持

在Pipfile中也可以引用环境变量的值，格式为${MY\_ENVAR}或$MY\_ENVAR，在Windows系统中还支持%MY\_ENVAR%。

[[source]]

url = "https://${PYPI\_USERNAME}:${PYPI\_PASSWORD}@my\_private\_repo.example.com/simple"

verify\_ssl = true

name = "pypi"

[dev-packages]

[packages]

requests = {version="\*", index="home"}

maya = {version="\*", index="pypi"}

records = "\*"

### 自定义虚拟环境路径

很多工具遵循Linux开发习惯，将东西全存在用户目录中，在Linux中可能没啥，但是在Windows下可能有人不喜欢把这些东西放在用户目录。当然pipenv也可以自定义，只需要设置或修改WORKON\_HOME环境变量的值即可。

如果设置了PIPENV\_VENV\_IN\_PROJECT环境变量，pipenv会把虚拟环境放在项目目录的.venv目录下。

### 配置pipenv

pipenv还有一些配置，都是使用环境变量配置的，由于配置项比较多，这里就不介绍了，直接看[官方文档](https://pipenv.readthedocs.io/en/latest/advanced/" \l "configuration-with-environment-variables" \t "_blank)好了。

### 从setup.py安装

pipenv也可以从setup.py安装：

pipenv install -e .

那么为什么不全用pipenv来安装呢？官方文档这里为我们做出了解释：项目可以分为两种，程序和库，对于程序来说应该使用pipenv，而对于库来说则是在setup.py中安装。详细解释说实话我没太看懂，大意就是抽象依赖和具体依赖，还有一个责任分配的问题

作者：乐百川  
链接：https://www.jianshu.com/p/dcbb01e8c78a  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。