

【pipenv】快速入门，超全面的pipenv教程！（教你如何快速创建python虚拟环境！）

2022-09-11 13:13 1543阅读 0赞

 Infineon

Automotive product quality

Download

文章目录

- 一、pipenv的快速入门
 - - 0、快速上手
 - 1、安装
 - 2、卸载
 - 3、跟新包
 - 4、首次运行
 - 5、删除虚拟环境
 - 6、与虚拟环境相关的常用命令
 - - (0) 展示环境中已安装的包
 - (1) 包的安装
 - (2) 包的卸载
 - (3) 包的更新
 - (4) 从requirements.txt导入环境
 - (5) 指定Python版本
 - (6) 运行命令
 - (7) 锁定版本
 - (8) 环境变量管理
- 二、高级用法
 - - 1、导出requirements.txt
 - 2、自动安装Python
 - 3、环境变量支持
 - 4、自定义虚拟环境路径
 - 5、配置pipenv
 - 6、从setup.py安装
- 三、联动pyinstaller的步骤
 - - 1、使用步骤
 - 2、国内源
 - 3、生成、安装requirements.txt
 - - (1) 生成方法一
 - (2) 万能生成方法
 - (3) 安装方法
- 四、遇到的问题
 - - 1、遇到依赖包无法安装的问题
 - 2、'gbk' 格式的问题
 - 3、pipenv install的时候报错
- Reference
 - 写在前面：由于pyinstaller每次打包的时候过大，经常几百兆，所以在此学习pipenv，并记录相关笔记，写为博客。
 - 作用：pipenv可以帮助我们管理Python和第三方库的版本。

- 参数

Usage: pipenv [OPTIONS] COMMAND [ARGS]...

Options:

- update 升级 pipenv, pip 到最新.
- where 输出项目的目录信息.
- venv 输出 virtualenv 的目录信息.
- py 输出 Python 解析器的路径.
- envs 输出环境变量的设置.
- rm 删除当前 virtualenv.
- bare Minimal output.
- completion Output completion (to be evald).
- man 显示使用手册.
- three / —two 使用 Python 3/2 来创建 virtualenv
- python TEXT 直接指定 Python 解析器.
- site-packages 拷贝系统 site-packages 到 virtualenv.
- jumbotron An easter egg, effectively.
- version 显示版本信息并退出.
- h, —help 显示当前信息并退出.

Commands:

check 检查安全漏洞和反对 PEP 508 标记在Pipfile提供.

graph 显示当前依赖关系图信息.

install 安装提供的包, 并加入 Pipfile 的依赖清单中

lock 生成 Pipfile.lock.

open 在编辑器(vim)查看一个特定模块.

run 在 virtualenv 中执行命令.

shell 切换到 virtualenv 中.

uninstall 删除提供的包, 并清理 Pipfile 的依赖清单中.

update 卸载当前所以依赖, 然后安装最新包

版权声明: 本文为CSDN博主「UMBL」的原创文章, 遵循CC 4.0 BY-SA版权协议, 转载请附上原文出处链接及本声明。
原文链接: https://blog.csdn.net/weixin_29799175/article/details/113581544

一、pipenv的快速入门

0、快速上手

- 使用pipenv的操作步骤

1. cmd中输入命令 `pip install pipenv` 安装pipenv
2. cmd切换路径到需要建立虚拟环境的项目目录下
3. 初始化特定版本的环境 `pipenv --python 3.6` (可选, 如果不初始, 则跟随系统)
4. 输入命令 `pipenv install` 开始安装虚拟环境
5. 安装完毕后输入命令 `pipenv shell` 进入虚拟环境
6. `pipenv install xxx` 安装相关依赖包
7. `pipenv graph` 查看目前按照的依赖包

1、安装

- 用pip命令就可以安装pipenv。

```
pip install pipenv
```

2、卸载

```
1. pip uninstall pipenv
```

3、跟新包

- 需要更新pipenv的时候，运行：

```
pip install --user --upgrade pipenv
```

4、首次运行

```
1. pipenv install
```

- 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。
- 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。
- Pipfile：用于保存项目的python版本、依赖包等相关信息

- Pipfile.lock：用于对Pipfile的锁定

```
[[source]]
url = "https://pypi.org/simple "
verify_ssl = true
name = "pypi"

[packages]
requests-html = "*"

[dev-packages]

[requires]
python_version = "3.7"
```

5、删除虚拟环境

```
1. pipenv --rm
```

6、与虚拟环境相关的常用命令

```
pipenv install:
```

- 若项目目录中虚拟环境未创建且无Pipfile文件，将安装虚拟环境并创建Pipfile文件
- 若项目目录中虚拟环境未创建且有Pipfile文件，将根据Pipfile文件来安装相应python版本和依赖包
- 若项目目录中虚拟环境已创建且有Pipfile文件，将根据Pipfile文件来安装依赖包

```
pipenv install xx: 安装python包
pipenv uninstall xx: 卸载python包
pipenv shell: 进入虚拟环境(项目目录下)
exit: 退出虚拟环境
pipenv graph: 显示包依赖关系
pipenv --venv: 显示虚拟环境安装路径
```

(0) 展示环境中已安装的包

```
1. pipenv graph
```

- 能够更清晰地展示现有依赖包。

(1) 包的安装

- 例如，我想在项目中安装requests这个包，运行：

```
pipenv install request
```

- 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。

(2) 包的卸载

- 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
```

(3) 包的更新

- 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>
```

(4) 从requirements.txt导入环境

- 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
```

(5) 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。

(6) 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：

```
1. pipenv run python --version
```

(7) 锁定版本

- 更新 lock 文件锁定当前环境的依赖版本

```
pipenv lock
```

(8) 环境变量管理

- 如果你开发调试时需要配一堆环境变量，可以写到.env文件中，在pipenv shell进入虚拟环境时，它会帮你把这些环境变量加载好，非常方便。
- 例如写一个.env文件

```
echo "test=toutiao" > .env
```

- 之后pipenv shell进入虚拟环境，echo \$test就能看环境变量的值toutiao已经设置好了。

二、高级用法

1、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
```

2、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'
```

3、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 = " "*"
```

4、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PIPVENV_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。

5、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，请直接看pipenv官方文档。

6、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。详细解释说其实我没太看懂，大意就是抽象依赖和具体依赖，还有一个责任分配的问题，原文在这里。

三、联动pyinstaller的步骤

1、使用步骤

- 关键点就一个：要在虚拟环境里安装pyinstaller如果你没有在虚拟环境中安装pyinstaller，你同样可以使用pyinstaller命令，但是调用的是你系统原本的那个python编译器，内含很多关联库，导致即使在虚拟环境中，你打包的exe文件仍然非常大。
- 另外一点要注意的是：要在虚拟环境里安装好你py文件中调用的库，不然打包出来也没法正常运行。

建立虚拟环境

```
pipenv install
```

进入虚拟环境

```
pipenv shell
```

安装模块

```
pip install xxx.py里面用到的模块
```

打包的模块也要安装

```
pipenv install pyinstaller
```

备份环境 requirements.txt

```
pipenv -lock -r  
或者  
pip freeze >requirements.txt
```

开始打包

```
pyinstaller -Fw -i 8.ico ./xxx.py
```

2、国内源

1. 阿里云: <http://mirrors.aliyun.com/pypi/simple/>
2. 豆瓣: <http://pypi.douban.com/simple/>
3. 清华大学: <https://pypi.tuna.tsinghua.edu.cn/simple/>
4. 中国科学技术大学: <https://pypi.mirrors.ustc.edu.cn/simple/>

3、生成、安装requirements.txt

(1) 生成方法一

- 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
```

(2) 万能生成方法

- 在虚拟环境中使用pip生成：

```
(venv) $ pip freeze >requirements.txt
```

(3) 安装方法

- 当需要创建这个虚拟环境的完全副本，可以创建一个新的虚拟环境，并在其上运行以下命令：

```
(venv) $ pip install -r requirements.txt
```

四、遇到的问题

1、遇到依赖包无法安装的问题

- 使用其他源安装包

```
pipenv install -i http://pypi.douban.com/simple/ request
```

2、‘gbk’ 格式的问题

```
1. UnicodeDecodeError: 'gbk' codec can't decode byte 0xac in position 384: illegal multibyte sequence
```

- 可能是文件夹目录有中文的原因，把文件夹目录改为英文

3、pipenv install的时候报错

- 这里删除当前文件下的Pipfile 和 Pipfile.lock 让它自动在生成一个就可

```
Pipfile.lock (9f6bff) out of date, updating to (32fa97)...
```

```
Locking [dev-packages] dependencies...
```

```
Locking [packages] dependencies...
```

```
Locking...Building requirements...
```

```
Resolving dependencies...
```

```
Locking Failed!
```

```
[ResolutionFailure]: File "d:/anaconda/anaconda/lib/site-packages/pipenv/resolver.py", line 741, in _main
```

```
[ResolutionFailure]: resolve_packages(pre, clear, verbose, system, write, requirements_dir, packages, dev)
```

```
[ResolutionFailure]: File "d:/anaconda/anaconda/lib/site-packages/pipenv/resolver.py", line 709, in resolve_packages
```

```
1. [ResolutionFailure]: requirements_dir=requirements_dir,
2. [ResolutionFailure]: File "d:/anaconda/anaconda/lib/site-packages/pipenv/resolver.py", line 692, in resolve
3. [ResolutionFailure]: req_dir=requirements_dir
4. [ResolutionFailure]: File "d:/anaconda/anaconda/lib/site-packages/pipenv/utils.py", line 1405, in resolve_deps
5. [ResolutionFailure]: req_dir=req_dir,
6. [ResolutionFailure]: File "d:/anaconda/anaconda/lib/site-packages/pipenv/utils.py", line 1110, in actually_resolve_deps
7. [ResolutionFailure]: resolver.resolve()
8. [ResolutionFailure]: File "d:/anaconda/anaconda/lib/site-packages/pipenv/utils.py", line 835, in resolve
9. [ResolutionFailure]: raise ResolutionFailure(message=str(e))
10. [pipenv.exceptions.ResolutionFailure]: Warning: Your dependencies could not be resolved. You likely have a mismatch in :
11. First try clearing your dependency cache with $ pipenv lock --clear, then try the original command again.
12. Alternatively, you can use $ pipenv install --skip-lock to bypass this mechanism, then run $ pipenv graph to inspect tl
13. Hint: try $ pipenv lock --pre if it is a pre-release dependency.
14. ERROR: Could not find a version that matches request (from -r C:\Users\25387\AppData\Local\Temp\pipenv7ncgw_byrequireme
15. No versions found
16. Was https://pypi.org/simple reachable?
```

Reference

1. pipenv快速入门
2. Pipenv的基本使用
3. 解决pyinstaller打包后程序体积过大问题
4. virtualenv虚拟环境删除_Python虚拟环境管理神器——Pipenv
5. pipenv官方文档

[收藏](#)

[关注](#)

赞 (0)

[海报](#)

[QQ 分享](#)

[微博分享](#)



微信分享
分享