枫恋寒 216

Pipenv – 超好用的 Python 包管理工具

python 更新于 2018-06-26 • 约 13 分钟

pipenv 是什么

pipenv 是 python 官方推荐的包管理工具，集成了 virtualenv、pyenv 和 pip 三者的功能于一身，类似于 php 中的 composer。

我们知道，为了方便管理 python 的虚拟环境和库，通常使用较多的是 virtualenv 、pyenv 和 pip，但是他们不够好用或者说不够偷懒。于是 requests 的作者 Kenneth Reitz 开发了用于创建和管理 python 虚拟环境的工具 —- pipenv。

它能够自动为项目创建和管理虚拟环境，从 Pipfile 文件中添加或者删除包，同时生成 Pipfile.lock 文件来锁定安装包的版本和依赖信息，避免构建错误。

pipenv 主要解决了以下问题：

- 不用再单独使用 virtualenv、pyenv 和 pip 了，现在它们结合到了一起。
- 不用再维护 requirement.txt 了，使用 Pipfile 和 Pipfile.lock 来代替。
- 可以在开发环境使用多个 python 版本。
- 在安装的 pyenv 条件下，可以自动安装需要的 python 版本。
- 安全，广泛地使用 Hash 校验，能够自动曝露安全漏洞。
- 随时查看图形化的依赖关系。

安装 pipenv

由于我的开发环境一直都是 Mac 笔记本，所以这里只介绍在 Mac 环境如何安装好了。

使用 pip 安装

```
$ pip install --user pipenv
```

这个命令在用户级别（非系统全局）下安装 pipenv。如果安装后 shell 提示找不到 pipenv 命令，你需要添加当前 Python 用户主目录的 bin 目录到 PATH 环境变量。如果你不知道 Python 用户主目录在哪里，用下面的命令来查看：

```
$ python -m site --user-base
```

你会看到类似下面的输出

```
/Users/liyafeng/Library/Python/3.6
```

使用 brew 安装

Mac 下使用 brew 安装软件应该是最方便的了，推荐使用：

```
brew install pipenv
```

升级 pipenv：

```
brew upgrade pipenv
```

shell 自动补齐

Linux or Mac 环境下，bash下如果能自动命令补全岂不是更好？请把如下语句追加到`.bashrc`或者`.zshrc`即可：

```
eval "$(pipenv --completion)"
```

常用命令

pipenv 具有的选项：

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

Options:
  --where           显示项目文件所在路径
  --venv            显示虚拟环境实际文件所在路径
  --py              显示虚拟环境Python解释器所在路径
  --envs            显示虚拟环境的选项变量
  --rm              删除虚拟环境
  --bare            最小化输出
  --completion      完整输出
  --man             显示帮助页面
  --three / --two   使用Python 3/2创建虚拟环境（注意本机已安装的Python版本）
  --python TEXT     指定某个Python版本作为虚拟环境的安装源
  --site-packages   附带安装原Python解释器中的第三方库
  --jumbotron       An easter egg, effectively.
  --version         版本信息
  -h, --help        帮助信息
```

pipenv 可使用的命令参数：

```
Commands:
  check       检查安全漏洞
  graph       显示当前依赖关系图信息
  install     安装虚拟环境或者第三方库
  lock        锁定并生成Pipfile.lock文件
  open        在编辑器中查看一个库
  run         在虚拟环境中运行命令
  shell       进入虚拟环境
  uninstall   卸载一个库
  update      卸载当前所有的包，并安装它们的最新版本
```

一些例子：

```
Usage Examples:
   Create a new project using Python 3.6, specifically:
   $ pipenv --python 3.6

   Install all dependencies for a project (including dev):
   $ pipenv install --dev

   Create a lockfile containing pre-releases:
   $ pipenv lock --pre

   Show a graph of your installed dependencies:
   $ pipenv graph

   Check your installed dependencies for security vulnerabilities:
   $ pipenv check

   Install a local setup.py into your virtual environment/Pipfile:
   $ pipenv install -e .

   Use a lower-level pip command:
   $ pipenv run pip freeze
```

pipenv 使用过程

创建环境，安装指定 python 的版本信息：

```
mkdir new_env & cd new_env
pipenv install     // pipenv install --three
```

如果指定了 `--two` 或者 `--three` 选项参数，则会使用 python2 或者 python3 的版本安装，否则将使用默认的 python 版本来安装。当然也可以指定准确的版本信息：

```
$ pipenv install --python 3
$ pipenv install --python 3.6
$ pipenv install --python 2.7.14
```

pipenv 会自动扫描系统寻找合适的版本信息，如果找不到的话，同时又安装了 pyenv 的话，则会自动调用 pyenv 下载对应版本的 python， 否则会报错。

这时候在当前 `new_env` 环境下生成 `Pipfile` 和 `Pipfile.lock` 两个环境初始化文件。

进入|退出环境：

进入环境：

```
pipenv shell
```

退出环境：

```
exit //或者 ctrl+d
```

安装第三方包：

这里我们测试安装 `urllib3` 包好了：

```
pipenv install urllib3
```

此时，Pipfile 里有最新安装的包文件的信息，如名称、版本等。用来在重新安装项目依赖或与他人共享项目时，你可以用 Pipfile 来跟踪项目依赖。

Pipfile 是用来替代原来的 requirements.txt 的，内容类似下面这样。source 部分用来设置仓库地址，packages 部分用来指定项目依赖的包，dev-packages 部分用来指定开发环境需要的包，这样分开便于管理。

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

[packages]
"urllib3" = "*"

[dev-packages]

[requires]
python_version = "3.6"
```

Pipfile.lock 则包含你的系统信息，所有已安装包的依赖包及其版本信息，以及所有安装包及其依赖包的 Hash 校验信息。

```
$ Pipfile.lock
{
    "_meta": {
        "hash": {
            "sha256": "af58f3510cb613d4d9241128f9a0ceb9bb936ad907543e23ad8317011dcb6715"
        },
        "pipfile-spec": 6,
        "requires": {
            "python_version": "3.6"
        },
        "sources": [
            {
                "name": "pypi",
                "url": "https://pypi.org/simple",
                "verify_ssl": true
            }
        ]
    },
    "default": {
         "urllib3": {
            "hashes": [
                "sha256:a68ac5e15e76e7e5dd2b8f94007233e01effe3e50e8daddf69acfd81cb686baf",
                "sha256:b5725a0bd4ba422ab0e66e89e030c806576753ea3ee08554382c14e685d117b5"
            ],
            "index": "pypi",
            "version": "==1.23"
```

现在安装另一个包，再次查看这两个文件的内容。你会发现 Pipfile 现在包含两个安装包了，Pipfile.lock 也包含了所有已安装包的依赖包及其版本信息，以及所有安装包及其依赖包的 Hash 校验信息。每次你安装新的依赖包，这两个文件都会自动更新。

安装指定版本包：

```
pipenv install urllib3==1.22
```

安装开发环境下的包：

加 `--dev` 表示包括 Pipfile 的 dev-packages 中的依赖。

```
pipenv install httpie --dev
```

卸载第三方包：

```
pipenv uninstall urllib3 //或者 pipenv uninstall --all
```

更新安装包：

```
pipenv update urllib3
```

更新所有包：

```
pipenv update
```

这个命令会删除所有软件包然后重新安装最新的版本。

查看虚拟环境目录：

```
$ pipenv --venv
/Users/liyafeng/.local/share/virtualenvs/new_env-UVLdq9CB
```

最后的虚拟环境目录是以当前环境 `new_env` 作为目录开头的。

查看项目根目录：

```
$ pipenv --where
/Users/liyafeng/Documents/www/pythondemo/new_env
```

检查软件包的完整性

你是否担心已安装的软件包有没有安全漏洞？没关系，pipenv 可以帮你检查，运行下面的命令：

```
$  pipenv check
Checking PEP 508 requirements…
Passed!
Checking installed package safety…
All good!
```

上面的命令根据 Pipfile 里的 PEP 508 标记检查安全漏洞。

查看依赖树

```
$ pipenv graph
httpie==0.9.9
  - Pygments [required: >=2.1.3, installed: 2.2.0]
  - requests [required: >=2.11.0, installed: 2.19.1]
    - certifi [required: >=2017.4.17, installed: 2018.4.16]
    - chardet [required: <3.1.0,>=3.0.2, installed: 3.0.4]
    - idna [required: <2.8,>=2.5, installed: 2.7]
    - urllib3 [required: >=1.21.1,<1.24, installed: 1.23]
```

锁定版本

更新 lock 文件锁定当前环境的依赖版本

```
pipenv lock
```