Table Of Contents

virtualenv

- 讨论区
- 勘误

Contents

- virtualenv
  - 安装
  - 用处
    - 命令
    - 环境变量和配置文件
    - Windows下注意事项
    - PyPy支持
  - 创建自己的启动脚本
    - 启动脚本范例
    - 激活脚本
    - **--system-site-packages** 参数
    - 不使用Virtualenv下的 **bin/python**
    - 重定位隔离环境
    - **--extra-search-dir** 参数
  - 与可替代品的比较
  - 贡献力量
    - 运行测试
  - 相关文档和链接
  - 现状和许可

Next topic

Changes & News

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

安装

运行 `pip install virtualenv` 即可安装virtualenv，想用 最新开发版 就运行 `pip install virtualenv==dev` 。

还可以用 `easy_install` 安装，即使是没有安装任何Python包管理器，也可以直接获取 virtualenv.py 并运行 `python virtualenv.py` ，效果一样。

用处

`virtualenv` 用来创建隔离的Python环境。

处理python环境的多版本和模块依赖，以及相应的权限是一个很常见的问题。比如，你有个应用使用的是LibFoo V1.0，但另一个应用却要用到LibFoo V2.0。 如何处理呢？如果把所有模块都安装到 `/usr/lib/python2.7/site-packages` (或是你本机python默认的模块安装目录)，那你极有可能无意中升级一些不该升级的模块。

更普遍的是，就算你成功安装了某个应用，那么接下来又会怎样？只要它开始运行了，那么只要其所依赖的模块发生任何改动，亦或升级，都可能打断该应用。

这还没完，要是你无法在 `site-packages` 目录下安装模块呢？比如共享主机。

上述这几种场合都适用 `virtualenv` 。它会创建一个拥有独立安装目录的python环境，该隔离环境不会与其他virtualenv环境共享模块（可选择是否访问全局库目录）。

一般用法是:

```
$ python virtualenv.py ENV
```

在已安装virtualenv的情况下，可以直接运行 `virtualenv ENV` 。

该操作会创建 `ENV/lib/pythonX.X/site-packages` 目录和 `ENV/bin/python` ，前者用来存放要安装的模块，后者就是隔离环境的Python解释器。在virtualenv环境下使用此解释器（包括以 `#!/path/to/ENV/bin/python` 开头的脚本）时，使用的都是隔离环境下的模块。

该操作还在隔离环境下安装了 Setuptools 或 distribute 。要用Distribue取代setuptools的话，只要运行:

```
$ python virtualenv.py --distribute ENV
```

设置环境变量 VIRTUALENV_USE_DISTRIBUTE 也能达到同样目的。

新的virtualenv还包含了 pip 包管理器，可以直接用 `ENV/bin/pip` 安装第三方模块。

命令

用法:

```
$ virtualenv [OPTIONS] DEST_DIR
```

选项:

`--version`
显示当前版本号。

`-h, --help`
显示帮助信息。

`-v, --verbose`
显示详细信息。

`-q, --quiet`
不显示详细信息。

`-p PYTHON_EXE, --python=PYTHON_EXE`
指定所用的python解析器的版本，比如 `--python=python2.5` 就使用2.5版本的解析器创建新的隔离环境。 默认使用的是当前系统安装(/usr/bin/python)的python解析器

`--clear`
清空非root用户的安装，并重头开始创建隔离环境。

`--no-site-packages`
令隔离环境不能访问系统全局的site-packages目录。

`--system-site-packages`
令隔离环境可以访问系统全局的site-packages目录。

`--unzip-setuptools`
安装时解压Setuptools或Distribute

`--relocatable`

重定位某个已存在的隔离环境。使用该选项将修正脚本并令所有.pth文件使用相当路径。

`--distribute`

使用Distribute代替Setuptools，也可设置环境变量VIRTUALENV_DISTRIBUTE达到同样效要。

`--extra-search-dir=SEARCH_DIRS`

用于查找setuptools/distribute/pip发布包的目录。可以添加任意数量的–extra-search-dir路径。

`--never-download`

禁止从网上下载任何数据。此时，如果在本地搜索发布包失败，virtualenv就会报错。

`--prompt==PROMPT`

定义隔离环境的命令行前缀。

环境变量和配置文件

virtualenv既可以通过命令行配置，比如 `--distribute` ，也可以用下面两种方式配置:

- 环境变量

  命令行的每个参数都以 `VIRTUALENV_<UPPER_NAME>` 的格式对应一个环境变量。 转换变量名过程中，除了将命令行参数大写外，还要把 (`'-'`) 替换为 (`'_'`)。

  举个例子，要自动安装Distribute取代默认的setuptools，可以这样设置环境变量:

```
$ export VIRTUALENV_USE_DISTRIBUTE=true
$ python virtualenv.py ENV
```

等同于在命令行直接使用参数:

```
$ python virtualenv.py --distribute ENV
```

有时要重复输入多个命令行参数，比如 `--extra-search-dir` 。变成环境变量时，要用空格隔开多个参数值，例如:

```
$ export VIRTUALENV_EXTRA_SEARCH_DIR="/path/to/dists /path/to/other/
$ virtualenv ENV
```

等同于:

```
$ python virtualenv.py --extra-search-dir=/path/to/dists --extra-sea
```

- 配置文件

virtualenv还能通过标准ini文件进行配置。在Unix和Mac OS X中是 `$HOME/.virtualenv/virtualenv.ini` ，在Windows下是 `%HOME%\\virtualenv\\virtualenv.ini` 。

配置项名称就是命令行参数的名称。例如，参数 `--distribute` 在ini文件如下:

```
[virtualenv]
distribute = true
```

象 `--extra-search-dir` 这样的多值命令行参数，在ini文件中要用断行将多个值隔开:

```
[virtualenv]
extra-search-dir =
    /path/to/dists
    /path/to/other/dists
```

`virtualenv --help` 可以查看完整的参数列表。

Windows下注意事项

在Windows下路径会与*nix下略有不同:脚本和可执行文件在Windows下位于 `ENV\Scripts\` 下，而非 `ENV/bin/` ， 模块也会安装在 `ENV\Lib\` 下，而非 `ENV/lib/` 。

要在某个含有空格的目录下面创建virtualenv环境，就要安装 win32api 。

PyPy支持

从1.5版开始，virtualenv开始支持 PyPy 。 >=1.5版的virtualenv支持PyPy1.4和1.4.1，>=1.6.1版的virtualenv支持PyPy1,5。

创建自己的启动脚本

Wrongway提示：该段一般情况下初学者用不到，所以刚接触virtualenv的朋友不要在此节投放过多精力。Virtualenv的文档讲解顺序是有点问题。

创建隔离环境时，virtualenv不会执行额外操作。但开发者有时会想在安装隔离环境后运行某个脚本。例如用脚本安装某个web应用。

要创建上述脚本，需要调用 `virtualenv.create_bootstrap_script(extra_text)` ，将后续操作写入到生成的启动脚本，以下是从docstring中生成的文档：

启动脚本与一般脚本无异，只是多了三个extend_parser, adjust_options, after_install三个钩子方法。

create_bootstrap_script返回一个可定制的，能做为启动脚本的字符串(当然，该字符串后面要写回到磁盘文件中)。 这个字符串是一个标准的virtualenv.py脚本，用户可以自行添加内容（所加内容必须是python代码）。

如果定义了下列方法，运行脚本时就会被调用:

`extend_parser(optparse_parser)`:

可以在解析器optparse_parser中添加或删除参数。

`adjust_options(options, args)`:

调整options，或改变args（如果要接收各种不同的参数，一定要在最后将 `args` 修改为 `[DEST_DIR]` ）

`after_install(options, home_dir)`:

在所有代码和模块安装完之后，就会调用该方法。这可能是用户最喜欢的方法，例如下:

```
def after_install(options, home_dir):
    if sys.platform == 'win32':
        bin = 'Scripts'
    else:
        bin = 'bin'
    subprocess.call([join(home_dir, bin, 'easy_install'),
                     'MyPackage'])
    subprocess.call([join(home_dir, bin, 'my-package-script'),
                     'setup', home_dir])
```

上述例子会安装一个包，并运行包内的setup脚本

wrongway在这里强调：上述三个方法并不是独立方法，而是一段代码字符串！！也就是extra_text的内容。有点象javascript下的 eval('.....代码字符串.....')

启动脚本范例

这有个具体的例子:

```
import virtualenv, textwrap
output = virtualenv.create_bootstrap_script(textwrap.dedent("""
import os, subprocess
def after_install(options, home_dir):
    etc = join(home_dir, 'etc')
    if not os.path.exists(etc):
        os.makedirs(etc)
    subprocess.call([join(home_dir, 'bin', 'easy_install'),
                     'BlogApplication'])
    subprocess.call([join(home_dir, 'bin', 'paster'),
                     'make-config', 'BlogApplication',
                     join(etc, 'blog.ini')])
    subprocess.call([join(home_dir, 'bin', 'paster'),
                     'setup-app', join(etc, 'blog.ini')])
"""))
f = open('blog-bootstrap.py', 'w').write(output)
```

这还有一个例子 点击 。

激活脚本

刚创建的隔离环境下会有一个 `bin/activate` 命令行脚本。在Windows下，激活脚本要在CMD.exe或Powershell.exe中使用。

在Posix系统(*nix/BSD)中，用法如下:

```
$ source bin/activate
```

该操作会将当前 `$PATH` 指向隔离环境下的 `bin/` 目录。之所以要用source是因为它要改变当前shell环境。仅仅就是一行命令，就这么简单。 如果直接运行隔离环境下的脚本或是python解释器（比如 `path/to/env/bin/pip` or `/path/to/env/bin/python script.py` ），那都没必要使用激活脚本。

输入 `deactivate` 就能退出已激活的隔离环境，也就是取消对当前 `$PATH` 所做的修改。

`activate` 脚本会修改当前shell命令行提示符，以提示当前激活的是哪个隔离环境。这是挺有用的， 不过要是想自定义的提示符，只要在运行 `activate` 前将 `VIRTUAL_ENV_DISABLE_PROMPT` 设为你想要的提示(不能为空字符串)。

在Windows下只须如此（*nix用户此处就不用看了，包括下面的注意也不用看了）:

```
> \path\to\env\Scripts\activate
```

输入 *deactivate* 就能退出隔离环境。

视你用的shell不同（CMD.exe或Powershell.exe），Windows会使用activate.bat或activate.ps1来激活隔离环境。 如果使用的是Powershell，那么以下几点值得注意。

注意(说真的，开发python还是在*nix下好，真的真的真的！):

```
使用Powershell时，运行 ``activate`` 脚本取决于`执行策略`_ 。但在Windows7下，默认
这就意味着象 ``activate`` 这样的脚本是不能直接运行的。但稍微设置一下即可。
```

降低执行策略，改为 ``AllSigned``, 这就意味着本机所有已通过数字签名的脚本都获许运行
由于virtualenv作者之一Jannis Leidel的数字签名已被核准，允许运行。那么只要以管理员

```
PS C:\> Set-ExecutionPolicy AllSigned
```

接下来运行脚本时会提示是否信任该签名::

```
PS C:\> virtualenv .\foo
New python executable in C:\foo\Scripts\python.exe
Installing setuptools.............done.
Installing pip...................done.
PS C:\> .\foo\scripts\activate

Do you want to run software from this untrusted publisher?
File C:\foo\scripts\activate.ps1 is published by E=jannis@leidel.info
CN=Jannis Leidel, L=Berlin, S=Berlin, C=DE, Description=581796-Gh7xfJ
and is not trusted on your system. Only run scripts from trusted publ
[V] Never run  [D] Do not run  [R] Run once  [A] Always run  [?] Help
(default is "D"):A
(foo) PS C:\>
```

如果选择了 ``[A] Always Run``, 该证书就会添加到当前帐户下的受信任发布者名单中，而且
如果选择了 ``[R] Run Once``, 该脚本会立即运行，但之后每次使用都会重新出现信任提示并
高级用户可以将该证书添加到当前计算机的受信任发布者名单中，这样所有用户都可以使用该脚本

此外，还可以进一步降低执行策略，允行未验证的本地脚本运行::

```
PS C:\> Set-ExecutionPolicy RemoteSigned
```

因为对任何一个virtualenv环境而言， ``activate.ps1`` 都是一个本地脚本而非远程脚本

`--system-site-packages` 参数

`virtualenv --system-site-packages ENV` 创建的隔离环境能直接引用 `/usr/lib/python2.7/site-packages` (即是本机全局site-packages路径)中的模块。

只在拥有全局site-packages目录的读写权限，并且你的应用要依赖其中的模块的情况下，该参数会很有用。其他情况下没必要使用该参数。

不使用Virtualenv下的 bin/python

某些情况下，我们无法或是不想使用由virtualenv创建的Python解释器。 比如，在 mod_python 或 mod_wsgi 下，只能用唯一一个Python解释器。(wrongway补充，不过uwsgi是可以使用多个python解释器的)

幸运的是，这相当简单。只要用指定的Python解释器来 安装 应用包即可。 但要使用这些模块，就得更正路径。有一个脚本可以用来更正路径，如下这般设置环境:

```
activate_this = '/path/to/env/bin/activate_this.py'
execfile(activate_this, dict(__file__=activate_this))
```

上述操作会更改 `sys.path` 和 `sys.prefix` ，但使用的仍是当前Python解释器。在隔离环境中 会先寻找 `sys.path` 下的内容再寻找全局路径。不过全局路径始终是可以访问的（无论隔离环境是否是由 `--system-site-packages` 创建的）。 而且，上述操作不会影响其他隔离环境，也不会更正在此之前已经引用的模块。所以，在处理web请求时才激活环境往往是无效的，应该尽可能早的激活环境和更正路径，而不是在处理请求时才开始处理。

重定位隔离环境

注意: –relocatable参数带有一定的实验性，可能还有一些尚未发现的问题。而且该参数也不能在Windows下使用。

一般情况下，隔离环境都绑定在某个特定路径下。这也就意味着不能通过仅仅是移动或拷贝目录到另一台计算机上而迁移隔离环境。 这时可以使用–relocatable来重定位隔离环境:

```
$ virtualenv --relocatable ENV
```

该参数会根据相对路径生成某些setuptools或distribute文件，然后再运行 `activate_this.py` 更改所有的脚本， 而不是通过改变python解释器软链接的指向来重置环境。

注意: 安装 任何 包之后，都要再次重定位环境。只要你将某个隔离环境迁移了，那么每安装一个新的包之后，都要再运行一遍 `virtualenv --relocatable` 。

要认识到，该参数 不能做到 真正的跨平台 。虽然我们可以移动相关目录，但仅仅能用于类似的计算机之间。 一些已知的环境差异，仍会导致不兼容：

- 不同版本的Python
- 不同平台使用不同的内部编码，比如一台用UCS2,另一台用UCS4
- Linux和Windows
- Intel和ARM
- 某些包依赖系统的C库，而C库在不同平台下有所差异(不同的版本或不同的文件系统下的所在位置)。

使用重定位参数创建新隔离环境时，会默认使用 `--system-site-packages` 参数。

`--extra-search-dir` 参数

创建新的隔离环境时，virtualenv会安装setuptools,distribute或是pip包管理器。一般情况下，它们都会从 Python Package Index (PyPI) 中寻找并安装最新的包。但在一些特定情况下，我们并不希望如此。例如， 你在部署virtualenv时既不想从网上下载，也不想从PyPI中获取包。

做为替代方案，可以让setuptools，distribute或是pip搜寻文件系统，让virtualenv使用本地发

行包而不是从网上下载。 只要象下面这样传入一个或多个 `--extra-search-dir` 参数就能使用该特性:

```
$ virtualenv --extra-search-dir=/path/to/distributions ENV
```

`/path/to/distributions` 路径指向某个包含setuptools/distribute/pip发行包的目录。Setuptools发行包必须是 `.egg` 文件，distribute和pip发行包则是 *.tar.gz* 原代码压缩包。

如果本地路径没有找到相应的发布包，virtualenv还是会从网上下载。

要想确保不会从网上下载任何发行包，就使用 `--never-download` 参数，如下:

```
$ virtualenv --extra-search-dir=/path/to/distributions --never-download E
```

这样，virtualenv不会从网上下载任何发行包。而只搜索本地发行包，如果没有找到要安装的包，就返回状态码1。 virtualenv会按照如下顺序搜索发行包位置:

1. 当前目录
2. virtualenv.py所在目录
3. virtualenv.py所在目录下的 `virtualenv_support` 目录
4. 如果实际运行的脚本名并不是virtualenv.py (换句话说，就是你的自定义启动脚本)，会搜索实际安装的virtualenv.py所在目录下的 ``virtualenv_support`` 目录。

与可替代品的比较

下面几个替代品也可以创建隔离环境:

- `workingenv` (建议不考虑workingenv) 是virtualenv的前身。它使用全局环境的Python解释器，但要靠设置 `$PYTHONPATH` 来激活环境。因此在运行隔离环境以外的Python脚本时，出现很多问题（比如，象全局环境下的 `hg` 或 `bzr` ）。而且它与Setuptools也有很多冲突。

- virtual-python 也是virtualenv的前身。它只使用软链接，因此不能在Windows上工作。而且它的链接会覆盖 标准模块和全局环境的 `site-packages` ，因此无法使用安装在全局环境下的 `site-packages` 的第三方模块

  因为virtual-python的软链接只是覆盖了全局环境下的标准模块的一部分，因此在windows上，可以用拷贝模块文件的方式来使用virtual-python。同时，它会创建一个空的 `site-packages` ，并把全局环境的 `site-packages` 指向该目录，因此更新是分别跟踪记录的（这块wrongway也不理解是什么意思，或许作者是想说要两个目录都注意要更新吧）。virtual-python也会自动安装Setuptools，从而省去了从网上手动安装这一步。

- zc.buildout 不会以上述方式创建隔离的Python环境，但它通过定义配置文件，使用非常特殊的模块，配置脚本达到了相似的效果。做为一个可定义的系统，它是非常容易复制和管理的，但是比较难以改写。 `zc.buildout` 可以安装非Python的系统（比如数据库服务器或是Apache实例）

我 强烈 建议任何人开发或部署应用时都应该上述工具中的某一款

贡献力量

参照 contributing to pip (参与PIP贡献)这篇文章，里面提及的内容同样适用于virtualenv。

Virtualenv与pip同步发行，每有新的pip发布，就意味着该捆绑新版本pip的virtualenv也发布了。

运行测试

Virtualenv 的测试案例很小，也不完整，但我们后面会完善的。

运行测试的最简单方法就是(自动处理测试依赖):

```
$ python setup.py test
```

可以使用nose运行测试的某一部分。创建一个virtualenv环境，然后安装必要的包:

```
$ pip install nose mock
```

运行nosetests:

```
$ nosetests
```

或是只测试某个文件:

```
$ nosetests tests.test_virtualenv
```

相关文档和链接

- James Gardner 编写了教程， 在virtualenv下使用Pylons 。
- 博文 workingenv已死，virtualenv当立 。
- Doug Hellmann 介绍了 virtualenv(virtualenvwrapper)命令行下流水线运行 ， 通过几个

自写的脚本，让运行多个环境变得更加容易。 他还写了 在virtualenv下运行IPython 。

- Chris Perkins 在showmedo创作了视频 使用virtualenv 。
- 在mod_wsgi下使用virtualenv 。
- 更多virtualenv周边工具 。

现状和许可

`virtualenv` 是 workingenv 的升级, 也是 virtual-python 的扩展。

`virtualenv` 由 Ian Bicking 编写，接受 Open Planning Project 赞助，由 开发小组 负责维护。 该开源遵循 MIT 协议。

Wrongway的补充：常用见法

1.创建隔离环境并安装最新的django

使用当前系统默认Python解释器安装最新的django（当前是1.4），以及django用到的mysql驱动:

```
$ mkdir myproject1
$ cd myproject1
$ virtualenv env --no-site-packages
$ source env/bin/active
$(env) pip install django
$(env) pip install mysql-python
$(env) deactive
$
```

2.创建隔离环境并安装django1.3以及一系列开发用组件

首先编辑一个.pip文件，假定为requirement.pip文件，将要用到的第三方模块名称写入:

```
Django==1.3
PIL
South
sorl-thumbnail
pylibmc
mysql-python
django-debug-toolbar
```

再在命令行运行:

```
$ mkdir myproject2
$ cd myproject2
$ virtualenv environ --no-site-packages
$ source environ/bin/active
$(environ) pip install -r requirement.pip
$(environ) deactive
$
```

3.创建Python2.7隔离环境并安装tornado

我当前环境的默认Python解析器版本是2.6，我已经安装了python2.7，现在两个python共存，但默认使用还是2.6:

```
$ mkdir myproject3
```

```
$ cd myproject3
$ virtualenv huanjing --no-site-packages --python=python2.7
$ source huanjing/bin/active
$(huanjing) pip install tornado
$(huanjing) deactive
$
```

要注意的，python2.7应该是被已设为全局可访问的，在当前命令行输入python2.7是可运行的，否则–python就要设为python2.7解释器的完整路径。

Wrongway的补充：中译版致谢

首先感谢我的妻，由于我受限于英文水平，本文很多生僻词语句子是由外语专业的老婆翻译和纠正的。在我译文的同时，她也在创作她的官案悬疑小说《大明御史传》 。

同时感谢扣扣群60158309的热心pythoner，有你们的鼓励，我才重新拾起翻译。