Python打包和环境管理方案研究

Shell
2025 年 9 月 20 日

大纲

- 虚拟环境使用演示
  - venv
  - virtualenv
  - pyenv
  - poetry
  - uv
- 虚拟环境原理解析
  - PATH法自制虚拟环境
  - Python系统目录结构
  - stdlib的决定机制
  - stdlib的影响因素
  - 最小Python home
  - 虚拟环境的简单实现和改进
  - site.py的核心逻辑
  - 验证venv的pyvenv.cfg
  - 验证空pyvenv.cfg
  - 验证pyvenv.cfg和bin下面的符号链接错位
  - 实验：亲手制作一个小的虚拟环境
- Python包管理系统
  - 包数据库
  - sdist安装格式和wheel安装格式
  - pip包管理器
  - 其他包管理器

Python虚拟环境主要实现

- venv：内置包，用起来很简单。
- virtualenv：扩展包，功能更强一些。
- pyenv：主要管理不同版本的Python，不直接管理虚拟环境。
- poetry：主要管理依赖。
- uv：rust实现，主要管理依赖。

venv

- 建立虚拟环境: python3 -m venv venv1
- 激活: source venv1/bin/activate
- 反激活: deactivate
- 直接用: venv1/bin/python3 -m pip list

```
shell@dev:~/var$ python3 -m venv venv1
shell@dev:~/var$ source venv1/bin/activate
(venv1) shell@dev:~/var$ pip3 list
Package Version
------- -------
pip     25.1.1
(venv1) shell@dev:~/var$ deactivate
shell@dev:~/var$ venv1/bin/python3 -m pip list
Package Version
------- -------
pip     25.1.1
```

virtualenv

- 建立虚拟环境: `virtualenv virtualenv1`
- 激活: `source virtualenv1/bin/activate`
- 反激活: `deactivate`
- 直接用: `virtualenv1/bin/python3 -m pip list`
- 和venv的比较: 功能更多一些

```
shell@dev:~/var$ virtualenv virtualenv1
created virtual environment CPython3.13.5.final.0-64 in 453ms
  creator CPython3Posix(dest=/home/shell/var/virtualenv1, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, via=copy, app_data_dir=/home/shell/.local/share/virtualenv)
    added seed packages: pip==25.2
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
shell@dev:~/var$ source virtualenv1/bin/activate
(virtualenv1) shell@dev:~/var$ pip list
Package Version
------- -------
pip     25.2
(virtualenv1) shell@dev:~/var$ deactivate
shell@dev:~/var$ virtualenv1/bin/python3 -m pip list
Package Version
------- -------
pip     25.2
```

pyenv

- 别和pyvenv搞混了。
- 列出环境: `pyenv install -l`
- 安装环境: `pyenv install 3.10.1`
- shell环境整合: `pyenv init`, 然后看说明。
- 激活: `pyenv shell 3.10.1`

```
shell@dev:~/var$ pyenv install 3.10.1
Downloading Python-3.10.1.tar.xz...
-> https://www.python.org/ftp/python/3.10.1/Python-3.10.1.tar.xz
Installing Python-3.10.1...
Installed Python-3.10.1 to /home/shell/.pyenv/versions/3.10.1
```

pyenv配合virtualenv建立虚拟环境

- 先做pyenv，并安装环境。
- `pip3 install --user virtualenv`
- `virtualenv virtualenv2`
- 随后和普通virtualenv建立出来的环境无区别。
- 注意：后续即便是从环境中移除pyenv相关设定（删除pyenv init让你加的内容，再开一个新的shell），此时建立出来的virtualenv2仍然可以启动正确的python版本。

poetry

- 建立项目: poetry new poetry-demo
- 添加依赖: poetry add pendulum
- 列出依赖:poetry show --tree
- 虚拟环境建立在~/.cache/pypoetry/virtualenvs/下面，每个项目一个。注意干掉项目的时候不自动删除。
- 使用虚拟环境: poetry run python your_script.py
- 激活:eval $(poetry env activate)
- 安装依赖:poetry install
- 打包:poetry build
- 列出python环境:poetry python list

```
shell@dev:~/var/poetry-demo$ poetry add pendulum
Creating virtualenv poetry-demo-0d-t6cYr-py3.13 in /home/shell/.cache/pypoetry/virtualenvs
Using version ^3.1.0 for pendulum

Updating dependencies
Resolving dependencies... (1.3s)

Package operations: 4 installs, 0 updates, 0 removals

  - Installing six (1.17.0)
  - Installing python-dateutil (2.9.0.post0)
  - Installing tzdata (2025.2)
  - Installing pendulum (3.1.0)

Writing lock file
```

注意:poetry run which python和poetry run bash，然后which python的输出不一致。

uv

- 建立项目：`uv init uv1`
- 安装python环境：`uv python install 3.10.18`
- 列出python环境：`uv python list`
- 添加依赖：`uv add requests`
- 列出依赖：`uv tree`
- 虚拟环境在`.venv`下面，干掉项目时会一并删除。
- 使用虚拟环境：`uv run your_script.py`
- `uv run which python`和`uv run bash`，然后`which python`的输出一致。
- 激活：没有直接指令，试试`source .venv/bin/activate`
- 安装依赖：`uv sync`
- 打包：`uv build`

```
shell@dev:~/var/uv1$ uv tree
Resolved 6 packages in 1ms
uv1 v0.1.0
└── requests v2.32.5
    ├── certifi v2025.8.3
    ├── charset-normalizer v3.4.3
    ├── idna v3.10
    └── urllib3 v2.5.0
```

注意目录下的`.python-version`文件。

uv

```
shell@dev:~/var$ uv python list
cpython-3.14.0rc2-linux-x86_64-gnu                <download available>
cpython-3.14.0rc2+freethreaded-linux-x86_64-gnu   <download available>
cpython-3.13.7-linux-x86_64-gnu                   <download available>
cpython-3.13.7+freethreaded-linux-x86_64-gnu      <download available>
cpython-3.13.5-linux-x86_64-gnu                   /usr/bin/python3.13
cpython-3.13.5-linux-x86_64-gnu                   /usr/bin/python3 -> python3.13
cpython-3.12.11-linux-x86_64-gnu                  /home/shell/.local/bin/python3.12 -> /home/shell/.local/share/uv/python/cpython-3.12.11-linux-x86_64-gnu/bin/python3.12
cpython-3.12.11-linux-x86_64-gnu                  /home/shell/.local/share/uv/python/cpython-3.12.11-linux-x86_64-gnu/bin/python3.12
cpython-3.11.13-linux-x86_64-gnu                  <download available>
cpython-3.10.18-linux-x86_64-gnu                  /home/shell/.local/bin/python3.10 -> /home/shell/.local/share/uv/python/cpython-3.10.18-linux-x86_64-gnu/bin/python3.10
cpython-3.10.18-linux-x86_64-gnu                  /home/shell/.local/share/uv/python/cpython-3.10.18-linux-x86_64-gnu/bin/python3.10
cpython-3.9.23-linux-x86_64-gnu                   <download available>
cpython-3.8.20-linux-x86_64-gnu                   <download available>
pypy-3.11.13-linux-x86_64-gnu                     <download available>
pypy-3.10.16-linux-x86_64-gnu                     <download available>
pypy-3.9.19-linux-x86_64-gnu                      <download available>
pypy-3.8.16-linux-x86_64-gnu                      <download available>
graalpy-3.11.0-linux-x86_64-gnu                   <download available>
graalpy-3.10.0-linux-x86_64-gnu                   <download available>
graalpy-3.8.5-linux-x86_64-gnu                    <download available>
```

参考和引用

- https://stackoverflow.com/questions/41573587/what-is-the-difference-between-venv-pyvenv-pyenv-virtualenv-virtualenvwrappe
- https://docs.python.org/3/library/venv.html
- https://virtualenv.pypa.io/en/latest/
- https://github.com/pyenv/pyenv
- https://python-poetry.org/docs/basic-usage/
- https://python-poetry.org/docs/cli/
- https://docs.astral.sh/uv/guides/projects/
- https://peps.python.org/pep-0518/ https://peps.pythonlang.cn/pep-0518/

虚拟环境原理解析

- PATH法自制虚拟环境
- Python系统目录结构
- stdlib的决定机制
- stdlib的影响因素
- 最小Python home
- 虚拟环境的简单实现和改进
- site.py的核心逻辑
- 验证venv的pyvenv.cfg
- 验证空pyvenv.cfg
- 验证pyvenv.cfg和bin下面的符号链接错位
- 实验：亲手制作一个小的虚拟环境

PATH法自制虚拟环境

通过设定PATH，实现最简单的虚拟环境。

所有脚本存放在单一目录下。激活时设定PATH，反激活时移除设定。

```
shell@dev:~/var$ mkdir uenv
shell@dev:~/var$ cd !$
cd uenv
shell@dev:~/var/uenv$ echo 'echo "hello, world"' > hello.sh
shell@dev:~/var/uenv$ chmod +x hello.sh
shell@dev:~/var/uenv$ export PATH=`pwd`:$PATH
shell@dev:~/var/uenv$ echo "$PATH"
/home/shell/var/uenv:/home/shell/.local/bin:/home/shell/bin:/usr/share/pyenv/shims:
/usr/share/pyenv/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
shell@dev:~/var/uenv$ cd ..
shell@dev:~/var$ hello.sh
hello, world
```

Python是否能够使用PATH法自制虚拟环境？

一个简单实验：

mkdir -p bin && cp /usr/bin/python3 bin/

docker run -it --rm -v $PWD/bin/:/root/ debian:13-slim bash

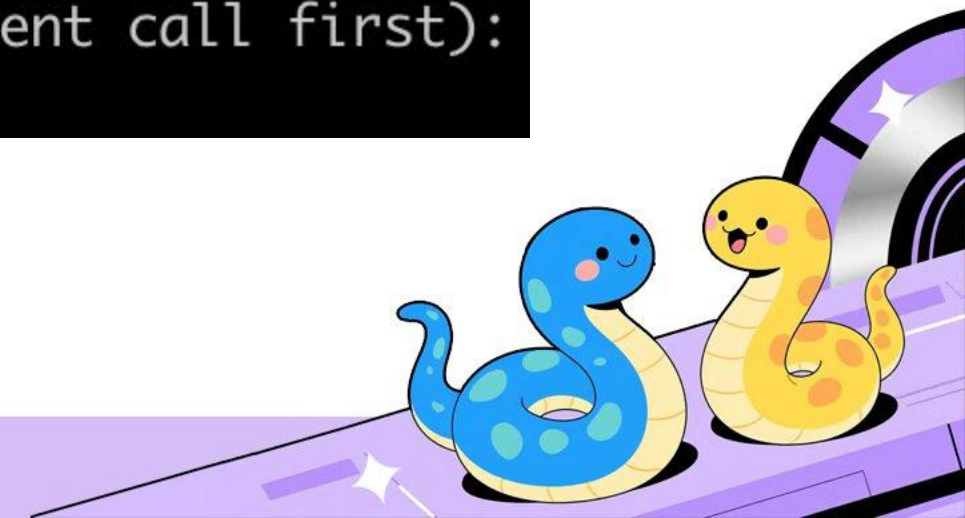
```
shell@dev:~/var$ mkdir -p bin && cp /usr/bin/python3 bin/
shell@dev:~/var$ docker run -it --rm -v $PWD/bin/:/root/ debian:13-slim bash
root@da700cd08c95:/# /root/python3
/root/python3: error while loading shared libraries: libexpat.so.1: cannot open shared object
file: No such file or directory
```

如果补上依赖呢？

apt-get update && apt-get install -y libexpat1

```
root@aadec8686f2f:/# /root/python3
Could not find platform independent libraries <prefix>
Could not find platform dependent libraries <exec_prefix>
Fatal Python error: Failed to import encodings module
Python runtime state: core initialized
ModuleNotFoundError: No module named 'encodings'

Current thread 0x00007f8c09c3c100 (most recent call first):
  <no Python frame>
```



Python的系统目录结构

- bin文件夹下
  - 二进制，主要就是python主程序，部分是安装包时带来的。
  - 脚本，包括pydoc3，pygettext3，pdb3，还有包括安装包时装进去的。
- 依赖库，一般是lib文件夹下
  - 二进制，有一些python实现使用动态编译。
  - stdlib，包括os.py，encoding。
- site-packages，一般分三种情况
  - 系统site-packages，例如debian下为`/usr/lib/python3.11/dist-packages/`。
  - 用户site-packages，debian下默认为`~/.local/lib/python3.11/site-packages/`。
  - 虚拟环境。
- 系统里最低需要有stdlib，才能启动（原理后述）。
- 可以通过sys.path查看路径设定，但是启动时，sys.path时怎么决定的？
  - 其实这是两个问题。stdlib怎么决定的，和site-packages怎么决定的。

stdlib的决定机制

- 最简单使用PYTHONHOME环境变量
  - 能用，但是low
- Python自己是怎么决定的？
  - `strace bin/python3 a.py 2>&1 | grep newfstatat`
  - 是通过执行路径逐步查找的
- 严格来说，决定过程是getpath
  - https://github.com/python/cpython/blob/main/PC/readme.txt
  - https://github.com/python/cpython/blob/main/Modules/getpath.c
  - https://github.com/python/cpython/blob/main/Modules/getpath.py
- getpath的载入和运行非常奇特。getpath.py是在python主程序的编译时，被编译成字节码，包含到可执行文件内部的。同时，在代码内部也不能引用一般的模块（因为stdlib的路径尚未确定）。
  - Python的类型系统已经就绪。
  - Python会把一些帮助函数和变量，注入到代码内部，用于计算。
  - 具体看getpath.py的注释。

stdlib的影响因素

- 当前路径
- 可执行文件路径
- 环境变量
- 还有一个是内置在python中的默认prefix，编译时决定
- pyvenv.cfg（后详述）

以上每个目录，都会搜索STDLIB_LANDMARKS。Linux下即是os.py或os.pyc。全路径为lib/python3.13/os.py（如果是python3.13的话）。找不到会向上级目录搜索。这个过程，基本符合strace的输出。

因此，将python3放置在某个目录的bin/下，将so放在同目录lib/下，stdlib放在lib/{python version}/，并且包含os.py。这个目录即可作为python的home目录，成功执行。具体应用的就是“可执行文件路径”这条。

实验：Python home

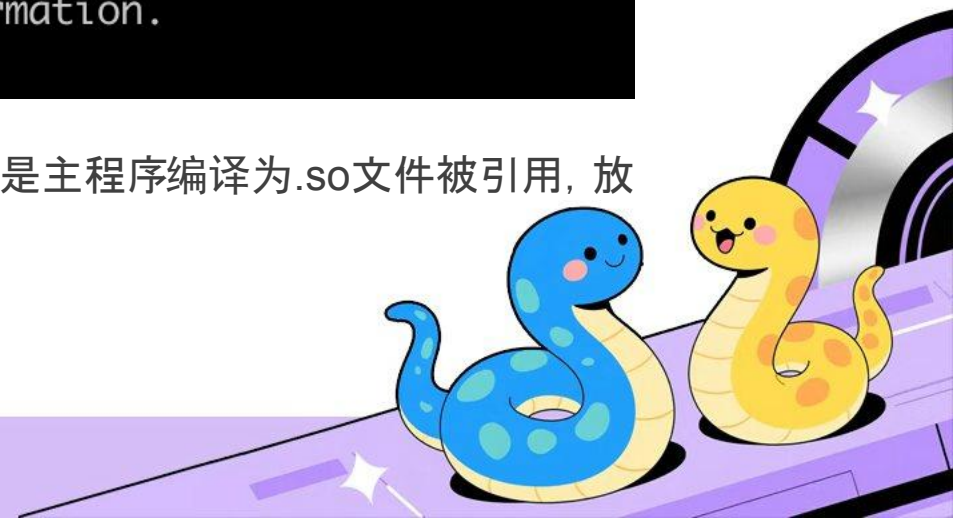
docker run -it --rm -v $PWD:/root/ debian:13-slim bash

/root/bin/python3

export LD_LIBRARY_PATH=/root/lib/

```
shell@dev:~/.pyenv/versions/3.10.1$ docker run -it --rm -v $PWD:/root/ debian:13-slim bash
root@ed1961aaea72:/# /root/bin/python3
/root/bin/python3: error while loading shared libraries: libpython3.10.so.1.0: cannot open
shared object file: No such file or directory
root@ed1961aaea72:/# export LD_LIBRARY_PATH=/root/lib/
root@ed1961aaea72:/# /root/bin/python3
Python 3.10.1 (main, Sep  7 2025, 01:59:54) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

注意：这个版本是用pyenv装出来的，是python的官方编译。不依赖libexpat1，但是主程序编译为.so文件被引用，放置在lib/下面。所以需要补一个LD_LIBRARY_PATH的环境变量。



最小Python home

逐步添加stdlib里的库，直到何时，Python主程序才可以正常启动？

略过过程直接说结果：最小需要有以下文件。
lib/python3.11/
lib/python3.11/os.py
lib/python3.11/encodings
lib/python3.11/encodings/aliases.py
lib/python3.11/encodings/__init__.py
lib/python3.11/encodings/utf_8.py
lib/python3.11/dist-packages
lib/python3.11/lib-dynload

其中的dist-packages是针对debian版编译。如果是标准版编译，则是site-packages。（原因后面说）
同样可以用docker验证，过程略。

虚拟环境的简单实现和改进

在有了上述准备知识后，我们可以很简单的设计如下的“虚拟环境”：

1. 设定一个路径，为虚拟环境基本路径。
2. python主程序放置在基本路径的bin/文件夹下。
3. so放置在lib/文件夹下。
4. stdlib放置在lib/{python version}/文件夹下。里面再放置以下内容。
    a. site-packages/dist-packages
    b. lib-dynload
5. 激活时将<基本路径/bin>加入PATH变量。
6. 反激活时移除。

我们就可以得到最简单的虚拟环境了。

但这个环境有个缺点：安装包和python主环境不分离。如果要打造多套虚拟环境，每个环境都要带一遍python主程序+stdlib。空间利用率太低，管理复杂。

虚拟环境的改进

虚拟环境的核心改进，就是site-packages和stdlib+bin所在的home目录分离，形成二级结构。从而达成多个虚拟环境共享一个python的效果。
而要达成这点，就要研究site-packages是怎么决定的。

- https://github.com/python/cpython/blob/main/Lib/site.py
- site.py是通过frozen import机制，直接打包在可执行文件内部在，不从硬盘加载。
- 通过python -v可以观察到加载过程。

site.py的核心逻辑

跳过其他逻辑判断，和我们有关的逻辑只有一点。如果pyvenv.cfg文件，存在于sys.executable上方的某个目录，那么sys.prefix将设定为此处，且此处是虚拟环境。此处相对的lib/{python version}/site-packages/会被添加到sys.path中。如果pyvenv.cfg中有include-system-site-packages，且为false，那么相对的，home目录的lib/{python version}/site-packages/就不会被加到搜索路径中。如果不包含这个设定，或者设定为其他值，那么系统的site-packages也会被搜索。

题外话：Debian为什么搜索的是 dist-packages？

- 这里可以下载到Debian的打包源码：https://packages.debian.org/source/trixie/python3.13
- 其中有个补丁：debian/patches/distutils-install-layout.diff
- 补丁修改了site.py的业务逻辑。
- 在安装后的文件中，亦可见此修改：/usr/lib/python3.13/site.py
- 同时，Debian在编译时默认设定了prefix，由此增加了一个搜索路径。/usr/lib/python3/dist-packages/。这一路径是用于安装Debian中众多的python3-*安装包，而不需要针对每一个python版本单独打一个安装包。

事情的麻烦之处在于，这个例子表明，我们的研究针对不同版本，不同系统，甚至相同版本相同系统的不同编译，结论可能全然不同。因此大部分结论只具有参考意义。有意义的是分析方法，以及熟悉虚拟环境的一些约定。

先写一个工具脚本

```
import sys

for name in ['_home', 'executable', '_stdlib_dir', 'platlibdir', 'prefix', 'exec_prefix', 'base_prefix', 'path']:
    if hasattr(sys, name):
        print(f'{name}: {getattr(sys, name)}')
```

验证venv的pyvenv.cfg

```
home = /usr/bin
include-system-site-packages = false
version = 3.13.5
executable = /usr/bin/python3.13
command = /usr/bin/python3 -m venv /home/shell/var/venv1
```

验证venv的pyvenv.cfg

```
shell@dev:~/var$ venv1/bin/python3 a.py
_home: /usr/bin
executable: /home/shell/var/venv1/bin/python3
_stdlib_dir: /usr/lib/python3.13
platlibdir: lib
prefix: /home/shell/var/venv1
exec_prefix: /home/shell/var/venv1
base_prefix: /usr
path: ['/home/shell/var', '/usr/lib/python313.zip', '/usr/lib/python3.13', '/usr/lib/python3.13/lib-dynload',
'/home/shell/var/venv1/lib/python3.13/site-packages', '/home/shell/.local/lib/python3.13/site-packages', '/usr
/local/lib/python3.13/dist-packages', '/usr/lib/python3/dist-packages']
shell@dev:~/var$ venv1/bin/python3 -m pip list | grep pendulum
pendulum                    3.1.0
shell@dev:~/var$ mv venv1/pyvenv.cfg ./
shell@dev:~/var$ venv1/bin/python3 a.py
_home: None
executable: /home/shell/var/venv1/bin/python3
_stdlib_dir: /usr/lib/python3.13
platlibdir: lib
prefix: /usr
exec_prefix: /usr
base_prefix: /usr
path: ['/home/shell/var', '/usr/lib/python313.zip', '/usr/lib/python3.13', '/usr/lib/python3.13/lib-dynload',
'/home/shell/.local/lib/python3.13/site-packages', '/usr/local/lib/python3.13/dist-packages', '/usr/lib/python
3/dist-packages']
shell@dev:~/var$ venv1/bin/python3 -m pip list | grep pendulum
```

验证venv的pyvenv.cfg

- pyvenv.cfg没了，虚拟环境里的site-packages就从sys.path里消失了，所安装的包也没了。

验证空pyvenv.cfg

```
shell@dev:~/var$ virtualenv1/bin/python3 a.py
_home: /usr/bin
executable: /home/shell/var/virtualenv1/bin/python3
_stdlib_dir: /usr/lib/python3.13
platlibdir: lib
prefix: /home/shell/var/virtualenv1
exec_prefix: /home/shell/var/virtualenv1
base_prefix: /usr
path: ['/home/shell/var', '/usr/lib/python313.zip', '/usr/lib/python3.13', '/usr/lib/python3.13/lib-dynload', '
/home/shell/var/virtualenv1/lib/python3.13/site-packages']
shell@dev:~/var$ mv virtualenv1/pyvenv.cfg ./
shell@dev:~/var$ touch virtualenv1/pyvenv.cfg
shell@dev:~/var$ virtualenv1/bin/python3 a.py
_home: None
executable: /home/shell/var/virtualenv1/bin/python3
_stdlib_dir: /usr/lib/python3.13
platlibdir: lib
prefix: /home/shell/var/virtualenv1
exec_prefix: /home/shell/var/virtualenv1
base_prefix: /usr
path: ['/home/shell/var', '/usr/lib/python313.zip', '/usr/lib/python3.13', '/usr/lib/python3.13/lib-dynload', '
/home/shell/var/virtualenv1/lib/python3.13/site-packages', '/home/shell/.local/lib/python3.13/site-packages', '
/usr/local/lib/python3.13/dist-packages', '/usr/lib/python3/dist-packages']
```

验证空pyvenv.cfg

- 空的pyvenv.cfg，虚拟环境的site-packages未消失。
- 由于include-system-site-packages不存在，所以系统的site-packages还在sys.path中。（虽然实际是dist-packages）
- 由于pyvenv.cfg里没有home，所以sys._home变量为None。

验证pyvenv.cfg和bin下面的符号 链接错位

```
shell@dev:~/var$ ls -l uv2/.venv/bin/python
lrwxrwxrwx 1 shell shell 81 Sep 11 00:45 uv2/.venv/bin/python -> /home/shell/.local/share/uv/python/cpython
-3.13.7-linux-x86_64-gnu/bin/python3.13
shell@dev:~/var$ cat uv2/.venv/pyvenv.cfg | grep home
home = /usr/bin
shell@dev:~/var$ uv2/.venv/bin/python a.py
_home: /usr/bin
executable: /home/shell/var/uv2/.venv/bin/python
_stdlib_dir: /usr/lib/python3.13
platlibdir: lib
prefix: /home/shell/var/uv2/.venv
exec_prefix: /home/shell/var/uv2/.venv
base_prefix: /usr
path: ['/home/shell/var', '/usr/lib/python313.zip', '/usr/lib/python3.13', '/usr/lib/python3.13/lib-dynload
', '/home/shell/var/uv2/.venv/lib/python3.13/site-packages']
```

验证pyvenv.cfg和bin下面的符号 链接错位

- bin下面的符号连接，和pyvenv.cfg中的home变量，两者设定可以错位。（实际上，符号连接指向的python为uv安装的3.13.7。home变量写的是debian系统里的python，版本号为3.13.5。两者只要大小版本号相同，getpath和site两大搜索逻辑就会互相找到对方的标志，兼容互换。因为标志为lib/{python version}/，不看patch version。）
- 两者错位的情况下，home变量的设定最终生效。

实验：亲手制作一个小的虚 拟环境

制作虚拟home

1. 建立myhome/目录。
2. 建立myhome/bin/目录。
3. 将某个版本python的bin文件，复制到myhome/bin/。
4. 建立myhome/lib/目录。
5. 将某个版本的lib/{python version}，复制到myhome/lib/。
6. 上面可能带了myhome/lib/{python version}/lib-dynload。如果没有，请建立。
7. 上述步骤实施完毕后，myhome/bin/python应当已经可以直接执行脚本。

制作虚拟环境

1. 建立myenv/目录。
2. 建立myenv/bin/目录。
3. 建立符号链接，从myenv/bin/python，指向myhome/bin/python。
4. 建立pyvenv.cfg，内容可以为空。
5. 建立myenv/lib/{python version}/site-packages目录。
6. 执行myenv/bin/python3 -m ensurepip，安装pip。
7. myenv/lib/{python version}/site-packages下应当出现pip目录。
8. myenv/bin/python3 -m pip list应当可以看到所有site-packages下的包。

实验：亲手制作一个小的虚 拟环境

```
shell@dev:~/var$ myenv/bin/python3 -m ensurepip
Looking in links: /tmp/tmpnoa9t4_6
Processing /tmp/tmpnoa9t4_6/pip-25.1.1-py3-none-any.whl
Installing collected packages: pip
Successfully installed pip-25.1.1
shell@dev:~/var$ myenv/bin/python3 -m pip list
Package           Version
----------------- -------
distlib           0.4.0
filelock          3.19.1
pip               25.1.1
ttsfrd-dependency 0.1
uv                0.8.15
virtualenv        20.34.0
shell@dev:~/var$ ls -l myenv/lib/python3.13/site-packages/
total 8
drwxrwxr-x 5 shell shell 4096 Sep 11 01:12 pip
drwxrwxr-x 3 shell shell 4096 Sep 11 01:12 pip-25.1.1.dist-info
shell@dev:~/var$
```

实验：虚拟环境改善

写如下脚本，放在myenv/bin/activate

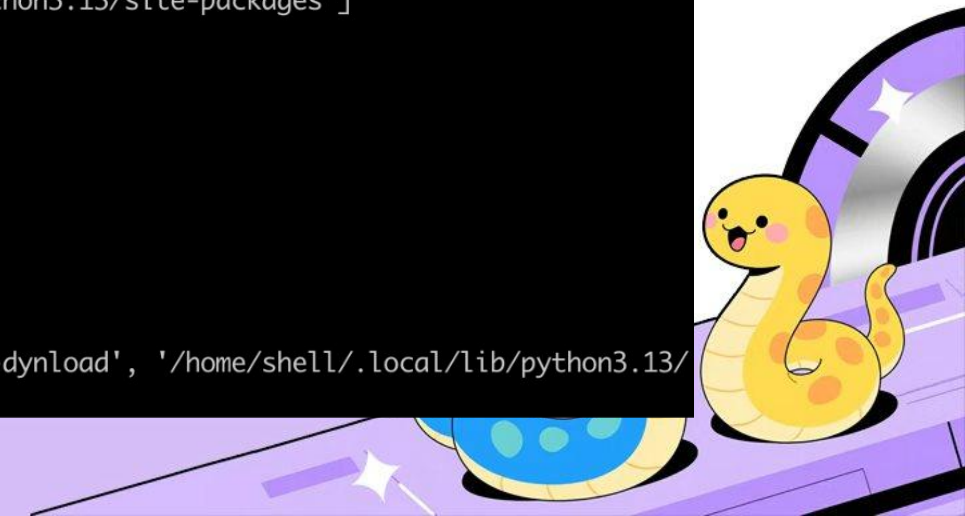
```
deactivate () {
  PATH="$OLDPATH"
  export PATH
  unset OLDPATH
  unset deactivate
}

OLDPATH="$PATH"
export OLDPATH
PATH=/home/shell/var/myenv/bin/:"$PATH"
export PATH
```

注意修改PATH里面，真实的虚拟环境路径。

实验：虚拟环境改善

```
shell@dev:~/var$ python3 a.py
_home: None
executable: /usr/bin/python3
_stdlib_dir: /usr/lib/python3.13
platlibdir: lib
prefix: /usr
exec_prefix: /usr
base_prefix: /usr
path: ['/home/shell/var', '/usr/lib/python313.zip', '/usr/lib/python3.13', '/usr/lib/python3.13/lib-dynload', '/home/shell/.local/lib/python3.13/
site-packages', '/usr/local/lib/python3.13/dist-packages', '/usr/lib/python3/dist-packages']
shell@dev:~/var$ source myenv/bin/activate
shell@dev:~/var$ python3 a.py
_home: None
executable: /home/shell/var/myenv/bin/python3
_stdlib_dir: /home/shell/var/myhome/lib/python3.13
platlibdir: lib
prefix: /home/shell/var/myenv
exec_prefix: /home/shell/var/myenv
base_prefix: /home/shell/var/myhome
path: ['/home/shell/var', '/home/shell/var/myhome/lib/python313.zip', '/home/shell/var/myhome/lib/python3.13', '/home/shell/var/myhome/lib/python
3.13/lib-dynload', '/home/shell/var/myenv/lib/python3.13/site-packages', '/home/shell/.local/lib/python3.13/site-packages']
shell@dev:~/var$ deactivate
shell@dev:~/var$ python3 a.py
_home: None
executable: /usr/bin/python3
_stdlib_dir: /usr/lib/python3.13
platlibdir: lib
prefix: /usr
exec_prefix: /usr
base_prefix: /usr
path: ['/home/shell/var', '/usr/lib/python313.zip', '/usr/lib/python3.13', '/usr/lib/python3.13/lib-dynload', '/home/shell/.local/lib/python3.13/
site-packages', '/usr/local/lib/python3.13/dist-packages', '/usr/lib/python3/dist-packages']
```



引用和参考

- https://docs.python.org/3/library/site.html
- https://docs.python.org/3/library/sys.html#sys.prefix
- https://docs.python.org/3/using/cmdline.html#envvar-PYTHONHOME
- https://github.com/python/cpython/blob/main/Lib/site.py

Python包管理系统

- 包数据库
- sdist安装格式和wheel安装格式
- pip包管理器
- 其他包管理器

包数据库

- 在site-packages下面，有很多.dist_info目录。
  - 任选其一，试着将其改名为.dist目录？
  - 该库从pip列表中消失了，但是还能正常import。
  - 包是包，包的源数据是包的源数据。
  - 也有一些包管理器使用.egg-info文件，.egg-info目录，或.egg目录。
- dist_info下包含很多包的源数据
  - METADATA：包含元数据，如PEP 345、PEP 314 和PEP 241 中所述。
  - RECORD：记录已安装文件的列表。
  - INSTALLER：记录用于安装项目的工具的名称。
  - REQUESTED：该文件的出现表示项目安装是显式请求的（即不是作为依赖项安装的）。
- 具体参考：https://peps.python.org/pep-0376/ https://peps.pythonlang.cn/pep-0376/

sdist安装格式和wheel安装格式

- 在uv虚拟环境下，执行uv build。
- 在dist目录下，可以看到tar.gz文件和whl文件。前者为sdist安装包，后者为wheel安装包。
- sdist安装包
  - 是一个tar.gz压缩包，其中包含所有源码，源数据文件。
  - sdist安装包侧重于发行源码，和具体系统无关。如果需要的话，sdist安装时需要编译，并需要编译环境支撑。
  - 具体可以参考这里：https://peps.python.org/pep-0517/ https://peps.pythonlang.cn/pep-0517/
  - 使用pip install <sdist filepath>来手工安装。

sdist安装格式和wheel安装格式

- wheel安装包
  - 是一个zip压缩包，其中包含所有编译后内容，源数据文件。
  - wheel安装包侧重发行二进制文件，和具体系统和平台相关。安装时不需要编译器支持，安装速度快得多。
  - 具体可以参考这里：https://peps.python.org/pep-0427/ https://peps.pythonlang.cn/pep-0427/
  - wheel安装包一般会在文件名里写入三元组。
    - python_tags：例如py3 for Python 3 only，cp310 for CPython 3.10。
    - abi_tags：一般是none，表示纯Python无abi依赖性。cp310 for CPython 3.10 ABI。
    - platform_tags：无平台依赖为any。manylinux1_x86_64 for Linux。
  - 具体可以参考这里：https://peps.python.org/pep-0425/ https://peps.pythonlang.cn/pep-0425/
  - 使用pip install <whl filepath>来手工安装。

pip包管理器

- 包管理工具
- 列出当前安装包信息
- 安装/删除包
- 远程下载安装（使用pypi源）
- pip自身的安装和更新
  - 使用ensurepip
  - 使用get-pip.py，具体参考此处

其他包管理器

- poetry
- uv

LOL ^_^

严格意义来说，venv和virtualenv是纯粹的虚拟环境，pyenv是纯粹的python实现编译和安装，poetry和uv是虚拟环境+包管理器+python实现管理器。

Thanks!