Python 记录文档

引言

我对 python 并不抱有好感，不过反感归反感，该用的地方还是得用的，笑。本文档主要对 python 的安装以及一些模块的简单使用做一些记录。

目录

1. 环境

Linux 一般自带 python，不过版本可能比较低。如果有网甚至有 root，那么 python 的安装会变得格外简单。不过实际工作中，公司不会给开发者提供 root 权限，一般也不会有外网，最多提供一个 pip 的代理地址。

如果根目录下的 python 版本新一些还好，如果只有 python2 之类的，那么情况会变得非常蛋疼。这时候将新版本的 python 安装到主目录成了唯一的办法。

2. python 安装

2.1 前置说明

下载 python 源码后直接安装其实是可以直接通过的，但是执行 make 时可能会有一些提示，一般会提示 ssl、tkinter 等模块未找到。其中值得注意的就是这个 ssl 模块。一般终端会提示：

Could not build the ssl module!

Python requires a OpenSSL 1.1.1 or newer

SSL 是一种加密协议，python 中经常会使用到 SSL 加密以确保安全通信。比如使用 pip 安装时，可能需要通过 HTTPS 连接到 PyPI，此时就需要 SSL 来加密通信。

其实我至今不太理解为什么通过源码安装 python 时需要额外安装一次 ssl，通过源码安装 python 是附带 pip 的，然后 pip 经常会使用到 ssl，但 ssl 又需要额外安装。至少看起来多少有些不合理。截止到 python3.12.2，我都遇到了这种情况，希望未来安装时可以再简单一些，不用这么折腾吧。

在 python3.10 之后，libressl 将不再被支持，所以这里使用 openssl 提供 ssl 模块。安装 openssl 前，需要先安装 libffi。

综上，安装前，需要至少先获取 libffi、openssl、python 三份源码。

libffi 官方地址如下，这里我使用的是 libffi-3.4.5 版本：

https://sourceware.org/libffi/

https://github.com/libffi/libffi/releases/download/v3.4.5/libffi-3.4.5.tar.gz

openssl 官方地址如下，这里我使用的是 openssl-1.1.1s 版本：

https://www.openssl.org/source/

https://www.openssl.org/source/old/1.1.1/openssl-1.1.1s.tar.gz

python3 官方地址如下，这里我使用的是 python-3.12.2 版本：

https://www.python.org/ftp/python/

https://www.python.org/ftp/python/3.12.2/Python-3.12.2.tgz

不考虑源代码的保留，基本步骤如下：

```
mkdir -p ~/tmp
mv ~/source_code_path/libffi-3.4.5.tar.gz ~/tmp
mv ~/source_code_path/openssl-1.1.1s.tar.gz ~/tmp
mv ~/source_code_path/Python-3.12.2.tgz ~/tmp
```

2.2 libffi 安装

libffi 最好和 python 安装到同目录下，否则后续可能需要额外的环境配置，安装步骤如下：

```
cd ~/tmp
tar -zxvf libffi-3.4.5.tar.gz && cd libffi-3.4.5
./configure --prefix=$HOME/.local/python-3.12.2
make -j && make install
```

2.3 openssl 安装

openssl 最好不要和 python 安装到同目录下，否则安装可能无法成功，当然也可以尝试修改 python 源码中的 Module/Setup 等方式解决。安装步骤如下：

```
cd ~/tmp
tar -zxvf openssl-1.1.1s.tar.gz && cd openssl-1.1.1s
./config --prefix=$HOME/.local/openssl-1.1.1s enable-shared
make -j && make install
```

2.4 python 安装

python 安装步骤如下：

```
cd ~/tmp
tar -zxvf Python-3.12.2.tgz && cd Python-3.12.2
export PKG_CONFIG_PATH=\
$HOME/.local/python-3.12.2/lib/pkgconfig:$PKG_CONFIG_PATH
export PKG_CONFIG_PATH=\
$HOME/.local/openssl-1.1.1s/lib/pkgconfig:$PKG_CONFIG_PATH
export LD_LIBRARY_PATH=\
$HOME/.local/python-3.12.2/lib64:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=\
$HOME/.local/openssl-1.1.1s/lib:$LD_LIBRARY_PATH
./configure --prefix=$HOME/.local/python-3.12.2 \
--with-openssl=$HOME/.local/openssl-1.1.1s \
--enable-shared
make -j && make install
```

2.5 环境变量添加

因为 python 库和可执行文件安装在~/.local/python-3.12.2/目录下，所以添加环境变量是必要的，步骤如下：

```
vim ~/.bash_profile
export PATH=$HOME/.local/python-3.12.2/bin:$PATH
export LD_LIBRARY_PATH=\
$HOME/.local/python-3.12.2/lib:\
$HOME/.local/python-3.12.2/lib64:\
$HOME/.local/openssl-1.1.1s/lib:$LD_LIBRARY_PATH
source ~/.bash_profile
```

2.6 安装完成

一般完成上述流程 python 的安装就完成了。使用如下命令查看 python 是否安装成功。

```
which python3
python3 --version
```

最好再通过 pip 安装一些模块检验 ssl 是否存在问题，如果都没有问题，就可以将~/tmp 目录删除了。

3. 模块安装

3.1 网络超时

使用 pip 安装模块时，可能会遇到 socket.timeout 报错；也可能是 hash 值不匹配报错：

ERROR: THESE PACKAGES DO NOT MATCH THE HASHES FROM THE REQUIREMENTS FILE。

在 pip 命令后添加--default-timeout 和--upgrade 参数，一般可以解决此问题：

pip3 --default-timeout=6666 install --upgrade paramiko

3.2 --user 参数

在 pip 命令后添加--user 参数，默认情况下，模块会被安装在~/.local/目录中，包括 bin、lib 等等。

在 pip 命令后不添加--user 参数，默认情况下，模块会被安装在 python-3.12.2 目录下，模块的 bin 目录和 python-3.12.2/bin 目录共用；模块的 lib 安装在 python-3.12.2/lib/python3.12/site-packages 下。

4. pyinstaller

4.1 引言

python 直接运行源文件实际上已经非常方便了，那么为什么还要用第三方工具将源代码进行打包呢？原因是多方面的。比如出于对源代码的保密，要求仅提供一个可执行的包；又或者当我的某个脚本需要在其它环境中运行，别的环境中没有我需要 python 配置。

因此，一些对 python 源码打包的工具应运而生，比如 pyinstaller、cxfreeze 等等。这里以 pyinstaller 为例。通过以下命令安装 pyinstaller：

pip install pyinstaller

4.2 使用

pyinstaller 官方使用文档地址如下：

https://pyinstaller.org/en/stable/usage.html

关于使用 pyinstaller 命令时的参数，实际上官方文档已经说明的非常清楚了，这里记录一些常用的参数选项。

(1) --distpath <dist_dir>

指定打包后程序的存放目录，不指定默认为./dist

(2) --clean

pyinstaller 在打包时会产生一系列缓存文件，如果希望打包前删除上一次的缓存文件，可以使用此参数。

(3) -F/--onefile

只生成一个可执行文件。不指定的话默认生成一个目录，其中包含很多文件。

(4) -n binary_file_name/--name binary_file_name

指定可执行文件的文件名，不指定默认使用打包时的第一个脚本文件名。

(5) --add-data=<file_source_path>:<file_packed_dest_dir>

添加其它数据文件。例如脚本中需要对./image/img.png 图片文件做处理。那么默认情况下 img.png 文件是不会被打包到程序中的。此时需要按如下方式指定打包参数：

--add-data=./image/img.png:./image/

添加 img.png 后，打包完成运行时不一定找得到 img.png 文件，因为脚本代码中可能使用的是相对路径，而实际使用 pyinstaller 打包过程中，pyinstaller 会

创建一个临时目录，此时通过--add-data 指定的文件将无法被添加。所以，若脚本中要对其它数据文件做处理，尽量使用这些数据文件的绝对路径。

--add-data 参数可多次使用。Linux 中，file_source_path 和 file_packed_dest_dir 之间用符号:分隔；Windows 中，file_source_path 和 file_packed_dest_dir 之间用符号;分隔。

(6) --add-binary=<lib_source_path>:<lib_packed_dest_dir>

打包时添加二进制文件。一些动态库在打包时可能不会被引用到程序中，此选项可以手动添加二进制文件。它的用法和--add-data 类似。lib_packed_dest_dir 一般为模块名称。比如添加 zbar 模块，那么 lib_packed_dest_dir 一般就指定./zbar。

(7) --icon <ico_file>

一般用于 Windows、MacOs，用于添加可执行文件的图标。

(8) --noconsole

一般用于 Windows、MacOs，可执行文件运行时将不会产生黑框窗口。

5. trash-cli

5.1 引言

试问在 Linux 下删除文件或目录要怎么做？当然是使用 rm 命令，于是顺手敲击键盘，输入 rm -rf <file_dir>。一般-r 和-f 这两个参数足以，脚本中可能加个-v 打印删除的文件信息。一开始我甚至不曾了解-r 和-f 的意思，只知道要删除目录，那就加一个-r。

直到有一天，我的某个目录被我不小心删没了，还没有 git。输入删除命令后按下回车那一刹那我拼命地敲击 ctrl+c，然而为时已晚。后续经历的风雨就不想多回忆了。实际上-f 表示--force，即强制删除；-r 表示--recursive，即递归删除。加上这两个参数，尤其是-f，表示删除文件的保险没有了，删除任何文件时，只要有权限，都直接删除，不会做出提示。那时候有人告诉我在 rm 命令后加上-i 参数会保险一些，-i 表示--interactive，即删除时向用户确认。不过在此之前我都没有听说过-i 这个参数，估计以后也经常懒得用。

事后一开始我是想自己搞一个类似 Windows 的回收站的，不过实际想来并不轻松，文件重名、文件的恢复等都还是比较麻烦的。不过为这种东西头疼的肯定不止我一个，于是 trash-cli 工具应运而生。

这是一款开源的文件删除工具，官方地址如下：

https://github.com/andreafrancia/trash-cli

这款工具很好地解决了 Linux 下，执行 rm 命令后，文件一去不复返的问题。trash-cli 是用 python 开发的，绝大多数的 Linux 发行版都提供这款软件，如果没有 root 权限问题也不大，通过以下命令安装 trash-cli：

pip install trash-cli

5.2 使用

trash-cli 的基本使用也非常简单，常用命令如下：

(1) trash-put

将文件或目录放入回收站。

(2) trash-empty

清空回收站，终端会询问是否清空。

(3) trash-list

列出回收站中的文件或目录。

(4) trash-restore

恢复回收站中的文件或目录。

(5) trash-rm

删除回收站中的目录或文件

一般被删除的文件位于~/.local/share/Trash 下，并且不需要担心文件重名问题，trash-cli 会自动处理。使用 trash-restore 恢复文件时，trash-cli 会询问具体恢复哪个文件，防止重名文件恢复混淆。恢复后的文件和删除前是完全一样的。

5.3 配置

如果每次删除文件都要输入 trash-put，那还是不现实的，删文件如此频繁的操作要用这么长的命令，那还是放弃得了。工具开发时，开发人员肯定也意识到了这个问题。所以 trash-cli 比较友好的地方就在于它的命令参数是完美适配 rm 命令的，比如 rm 的-i、-v 等参数 trash-cli 也有。

所以可以在环境初始化中添加如下语句：

```
alias rm="trash-put"
```

此时执行 rm 后，文件便进入了回收站。还有一点是，trash-put 删除目录时是不需要加-r 参数的，不过即使习惯性的加了也没什么问题。添加以上环境变量后，若想真正的使用 rm 命令而不是 trash-put，可以在 rm 命令前加上符号\或者使用 unalias rm 解除对 rm 命令的修改。此后，时不时运行一下 trash-empty 命令清空一下回收站就可以了。

值得一提的是，执行脚本或可执行程序等，子进程不会继承父进程的 alias，alias 是 shell 的一个命令，并不是一种环境变量。通常子进程会继承父进程的环境变量，但不会继承父进程的 alias。所以在 shell 中执行 rm 命令，或在 Makefile 中编写伪目标 clean 等情况下，还是要多加小心的。

6. setuptools

6.1 引言

假设在编写 python 代码时，需要用到一个自定义模块中的函数，若此时模块文件不在当前目录下，那么单纯的 import 是搜索不到模块路径的。通常的方法是通过以下方式指定模块的搜索路径：

sys.path.append(selfModulePath)

如果这个自定义模块的通用性比好，许多源文件中都会用到它，那每次都通过上述的方式指定搜索路径就比较麻烦了。如果可以的话，像 sys、os 这些可以直接 import 似乎更友好一些。

setuptools 这个工具就能很好地完成上述要求。它可以自定义创建 python 模块，并将这些模块像其它模块一样安装到 site-packages 中。如此，自定义的模块用起来就可以像其它模块一样方便了。

实际上 setuptools 的功能非常丰富，也被许多其它模块作为依赖。自己使用时，通常了解一些基础功能即可。通过以下命令安装 setuptools：

pip install setuptools

6.2 使用

setuptools 是一款开源工具，github 地址如下

https://github.com/pypa/setuptools

setuptools 官方使用文档地址如下：

https://setuptools.pypa.io/en/latest/setuptools.html

6.2.1 目录层级

假设打包根目录是$HOME/python/common，这里打包根目录用$pRoot 代替。我希望打包一个名为 fileModule 的模块作用通用模块。

那么首先，需要在$pRoot 下创建一个名为 fileModule 的目录，这个目录的名称不要修改，后续 import 或者 from 导入模块时用的就是这个名称。将有关 fileModule 模块的源文件放入这个目录中，假设我这里有一个源文件 fileModule.py，那么我将其放入 fileModule 目录中。此外，还需要在 fileModule 目录中创建一个__init__.py 文件，这个文件可以是空内容，但必须要有。setuptools 会搜索同级目录中，带有__init__.py 文件的其它源文件，然后进行打包。若没有__init__.py 文件，打包后，会找不到对应的 fileModule。

其次，在$pRoot 目录下，需要创建一个名为 setup.py 的打包文件，这个文件中指定的是打包的一些信息。综上，打包根目录的层级如下：

```
$pRoot
├─── fileModule
│    ├─── fileModule.py
│    └─── __init__.py
└─── setup.py
```

6.2.2 setup.py

setup.py 文件内容如下：

```
import setuptools
setuptools.setup(
    name="fileModule",
    version="1.0",
    description="File processing",
    author="Hu_Yihua_UsadaYu",
    author_email="UsadaYu.yh@gmail.com",
    packages=setuptools.find_packages("./"),
)
```

(1) name：指定包名称，这个名称是任意，不同于上述的 fileModule 目录名称。不过最好将其和上述的目录名称设置为一样。

(2) version：指定版本号，pip install --upgrade 一般根据这个参数寻找目标版本。

(3) packages：指定源文件搜索路径，在此基础上寻找上述的__init__.py 文件。

6.2.3 command

(1) python3 setup.py sdist

上述命令会根据 setup.py 的内容以 sdist 的形式进行打包，Linux 下，打包后生成的一版是一个 tar.gz 形式的压缩文件。此时打包根目录的层级如下：

```
$pRoot
├── dist
│    └── fileModule-1.0.tar.gz
├── fileModule
│    ├── fileModule.py
│    └── __init__.py
├── fileModule.egg-info
```

```
│   ├── dependency_links.txt
│   ├── PKG-INFO
│   ├── SOURCES.txt
│   └── top_level.txt
└── setup.py
```

(2) pip install dist/fileModule-1.0.tar.gz

运行上述命令可将 fileModule 模块安装到 site-packages 中，此时 site-packages 下一般会多出两个目录，为：fileModule 和 fileModule-1.0.dist-info。可以看出这个很多其它模块的形式一样了。此时用 pip list 可以查看到 fileModule 已被安装；使用 pip show fileModule 也可以看到 setup.py 中指定的一些打包信息。

(3) from fileModule import fileModule

表示从 fileModule 目录中导入 fileModule 模块，第二个 fileModule 对应的是 fileModule.py。也就是说 site-packages 为第一级搜索目录。

6.2.4 不使用 from

上述在使用 fileModule 模块时，相比 import os 这种直接导入，多了一段 from。如果希望可以直接 import fileModule，那么在打包前的__init__.py 文件中添加如下内容即可：

```
from .fileModule import *
```

上述的符号.不要遗漏，这段内容在打包后的 site-packages 中的__init__.py 文件里添加也是有用的。