Python2 & Python coexist on Linux

为什么需要两个版本的 Python

在前一阵子，一直在忙着将公司的测试用例从 Python2 迁到 Python3, 前几天才终于完成了，所有三百多个测试用例终于能在我本地的 Python3 环境下跑通了。
改别人的东西很煎熬，也学到不少东西。
接下来的任务就是把改好的用例部署在 Jenkins 服务器上，需要安装 Python3 环境，**但不能影响现有的 Python2 下的运行环境**。
于是就有了两个版本并存的需求。
然后开始折腾。

安装 Python3

- 安装依赖环境

```
$ sudo yum -y install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel gdbm-devel db4-devel libpcap-devel xz-devel
```

- 下载最新 Python3

```
$ sudo wget https://www.python.org/ftp/python/3.7.1/Python-3.7.1.tgz
```

- 解压
  创建安装目录：

```
$ sudo mkdir -p /usr/local/python3
```

  解压：

```
$ sudo tar -zxvf Python-3.7.1.tgz
```

- 编译安装
  进入解压后的目录，编译安装：

```
$ cd Python-3.7.1
$ ./configure --prefix=/usr/local/python3   # /usr/local/python3 为安装目录
$ sudo make
$ sudo make install
```

~~使用 `sudo make && make install` 命令会安装失败，不知道为什么~~

- 建立 Python3 软链

```
$ sudo ln -s /usr/local/python3/bin/python3 /usr/bin/python3
```

- ~~将 /usr/local/python3/bin 加入 PATH~~, 实际上我没有做这一步

```
$ vim ~/.bash_profile
$ .bash_profile
$ Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
$ User specific environment and startup programs
PATH=$PATH:$HOME/bin:/usr/local/python3/bin
export PATH
```

做到这里，双环境其实是已经装完，只要将 Python2 软链到 Python2 的安装目录；Python3 软链到 Python3 的安装目录，使用 Python2 & Python3 命令即可实现 python 环境的切换。

```
[apps@vm167 ~]$ python2 --version
Python 2.7.14
[apps@vm167 ~]$ python3.7 --version
Python 3.7.1
```

可以参考软链的创建、删除和更新
但是并不能满足 robotframework 在 Jenkins 上的双环境的运行需求，甚至还直接把默认 python 环境改成 Python3, 造成了现存在 Python2 的用例不能运行... 瑟瑟发抖，赶紧去查怎么改回 Python2.

切换 python2 和 python3

修改别名

首先先来看一下我们的默认Python版本

```
$ python --version
Python 2.7.6
```

然后我们修改一下别名

```
$ alias python='/usr/bin/python3'
$ python --version
```

```
Python 3.4.3  # 版本已经改变
```

/usr/bin/python3 这个路径是怎么找到的呢？

> 一般来说，软件的二进制文件都可以在 /usr/bin 或 /usr/local/bin (这个优先级高一点)找到。当然如果你是Debian系的Linux，可以这么找(前提是你已经安装了Python3)：

> 上面的别名修改只是暂时性的，重开一个窗口后配置就不见了。如果要使每个窗口都使用这个别名，可以编辑 ~/.bashrc (如果你是别的shell的话，就不是这个文件，如zsh是 ~/.zshrc)，把alias配置写入文件。

修改别名优点是足够简单，但是切换不灵活。

链接文件

在 /usr/bin 中建立一个链接文件指向Python3。

```
$ ln -s python /usr/bin/python3
$ python --version
Python 3.4.3
```

跟修改别名一样，默认环境只能有一个

采用 **update-alternatives** 切换版本

update-alternatives 是 Debian 提供的一个工具(非Debian系的就不用看了)，原理类似于上面一个办法，也是通过链接的方式，但是其切换的过程非常方便。
首先看一下update-alternatives的帮助信息：

```
$ update-alternatives --help
用法：update-alternatives [<选项> ...] <命令>

命令：
  --install <链接> <名称> <路径> <优先级>
    [--slave <链接> <名称> <路径>] ...
                            在系统中加入一组候选项。
  --remove <名称> <路径>    从 <名称> 替换组中去除 <路径> 项。
  --remove-all <名称>       从替换系统中删除 <名称> 替换组。
  --auto <名称>             将 <名称> 的主链接切换到自动模式。
  --display <名称>          显示关于 <名称> 替换组的信息。
  --query <名称>            机器可读版的 --display <名称>.
  --list <名称>             列出 <名称> 替换组中所有的可用候选项。
  --get-selections          列出主要候选项名称以及它们的状态。
  --set-selections          从标准输入中读入候选项的状态。
  --config <名称>           列出 <名称> 替换组中的可选项，并就使用其中
                            哪一个，征询用户的意见。
  --set <名称> <路径>       将 <路径> 设置为 <名称> 的候选项。
  --all                     对所有可选项一一调用 --config 命令。
```

```
<链接> 是指向 /etc/alternatives/<名称> 的符号链接。
    (如 /usr/bin/pager)
<名称> 是该链接替换组的主控名。
    (如 pager)
<路径> 是候选项目标文件的位置。
    (如 /usr/bin/less)
<优先级> 是一个整数，在自动模式下，这个数字越高的选项，其优先级也就越高。

选项：
  --altdir <目录>           改变候选项目录。
  --admindir <目录>         设置 statoverride 文件的目录。
  --log <文件>              改变日志文件。
  --force                   就算没有通过自检，也强制执行操作。
  --skip-auto               在自动模式中跳过设置正确候选项的提示
                            (只与 --config 有关)
  --verbose                 启用详细输出。
  --quiet                   安静模式，输出尽可能少的信息。不显示输出信息。
  --help                    显示本帮助信息。
  --version                 显示版本信息。
```

我们仅需要了解3个参数就行了

- --install <链接> <名称> <路径> <优先级> ：建立一组候选项
- --config <名称> ：配置 <名称>组中的可选项，并选择使用其中哪一个
- --remove <名称> <路径> ：从 <名称> 中去掉 <路径>选项

首先建立 python 的组，再添加 Python2 和 Python3 的可选项

```
$ sudo update-alternatives --install /usr/bin/python python /usr/bin/python2.7 2 #
添加Python2可选项，优先级为1
$ sudo update-alternatives --install /usr/bin/python python
/usr/local/python3/bin/python3 2 #添加Python3可选项，优先级为2
```

注意，这里的 /usr/bin/python 链接文件，两个可选项必须是一样的，这样这个链接文件才可以选择两个不同的可选项去链接。
这时如果我们查看 /usr/bin/python 这个文件时，会发现它已经链接到了

```
[apps@vm167 bin]$ ll python*
lrwxrwxrwx. 1 root root   24 Nov 21 17:13 python -> /etc/alternatives/python
```

此时查看 python 的可选项：

```
[apps@vm167 ~]$ update-alternatives --display python
python - status is manual.
 link currently points to /usr/bin/python2.7
/usr/bin/python2.7 - priority 1
```

```
  /usr/local/python3/bin/python3 - priority 2
  Current `best' version is /usr/local/python3/bin/python3.
```

另外关于 update-alternatives 的自动模式，在我使用的这个 linux 版本中没有遇到，暂不做讨论。如果有遇到，可以参考文末的链接。

这种方法，也只是强行把默认 python 切换回了 Python2, 使目前的依赖 Python2 的组件可以正常运行，并不能兼容使用 Python3.

virtualenvwrapper 切换版本

virtualenvwrapper 是管理 Python 虚拟环境的工具，可以很方便的为不同的项目建立独立的环境，每个项目都可以安装自己的依赖，同时也支持在不同的虚拟环境中存在不同版本的 Python。

首先安装 virtualenvwrapper, 可以选择 wget 安装或者 pip 安装

- wget安装

  ```
  $ sudo wget install virtualenvwrapper
  ```

- pip安装

  ```
  $ sudo pip install virtualenvwrapper
  ```

当你需要使用 Python2 开发项目时，建立一个 Python2 的虚拟环境：

```
$ sudo mkvirtualenv -p /usr/bin/python2 env27
```

当你需要Python3开发时：

```
$ mkvirtualenv -p /usr/bin/python3.4 env34
```

然后可以随时切换不同的虚拟环境：

```
$ workon env27  # 进入Python2环境
$ workon env34  # 进入Python3环境
```

更爽的是，你可以在进入虚拟环境的同时切换到项目目录，只需要编辑 $VIRTUAL_ENV/bin/postactivate 这个文件即可：

```

```

```
$ vim $VIRTUAL_ENV/bin/postactivate  #前提是已经进入对应的虚拟环境
```

在文件中添加切换目录的命令：

```
cd  /path/to/your/project
```

这种方法，感觉很嗨。然而 Python3 虚拟环境创建成功了，进入虚拟环境，默认还是 Python2.
接下来就是，virtualenv & virtualenvwrapper 的区别。

virtualenv & virtualenvwrapper

virtualenv

virtualenv 的作用：为单个项目创建独立的 python 虚拟环境

virtualenv 的使用：

1. 通过如下命令安装virtualenv

```
$ sudo pip install virtualenv
```

2. 创建虚拟环境：

- 创建默认虚拟环境：

  ```
  $ sudo virtualenv env27
  ```

  此命令表示创建一个名为 env27 的虚拟环境。每个虚拟环境都包含一个独立 env27/bin/python 和 env27/bin/pip, 当运行他们时 env27 做为你的运行环境。

- 创建指定解释器版本的虚拟环境：

  ```
  $ sudo virtualenv -p python2.7 env2.7 #解释器为python2.7
  $ virtualenv -p python3.4 env3.4 #解释器为python3.4
  ```

- 创建继承第三方的虚拟环境
  如果python已经安装了第三方库，你希望在新的虚拟环境中也使用这些库，那么可使用如下命令：

  ```
  $ virtualenv --system-site-packages env27
  ```

  如果不想使用可使用如下命令：

```
$ virtualenv --no-site-packages env27
```

3. 启动和退出虚拟环境

- 启动虚拟环境 要想启动env27虚拟环境，只要运行如下命令：

  ```
  $ source env27/bin/activate
  ```

  这个命令会修改系统路径$PATH，把env27/bin的路径至于系统路径之前。

  source 命令表示更改当前的 shell 环境。

  启动了虚拟环境之后，所有pip命令新安装的第三方包都将安装在当前环境下，而不会影响系统环境或者其它虚拟环境。

- 退出虚拟环境 退出当前虚拟环境只需执行如下命令：

  ```
  $ deactivate
  ```

virtualenvwrapper

virtualenvwrapper 时一个基于 virtualenv 之上的工具，它将所有的虚拟环境统一管理。

- 安装

  如下命令：

  ```
  $ sudo pip install virtualenvwrapper
  ```

virtualenvwrapper 默认将所有的虚拟环境放在 ～/.virtualenvs 目录下管理，可以修改环境变量 WORKON_HOME 来指定虚拟环境的保存目录。

使用如下命令来启动 virtualenvwrapper：

```
$ source /usr/local/bin/virtualenvwrapper.sh
```

还可以将该命令添加到 ~/.bashrc 或 ~/.profie 等 shell 启动文件中，以便登陆 shell 后可直接使用 virtualenvwrapper 提供的命令。
对于 virtualenvwrapper 的配置：
代码如下：

```

```

```
if [ `id -u` != '0' ]; then

  export VIRTUALENV_USE_DISTRIBUTE=1        # <-- Always use pip/distribute
  export WORKON_HOME=$HOME/.virtualenvs       # <-- Where all virtualenvs will be
stored
  source /usr/local/bin/virtualenvwrapper.sh
  export PIP_VIRTUALENV_BASE=$WORKON_HOME
  export PIP_RESPECT_VIRTUALENV=true

fi
```

将上面的配置添加到 ~/.bashrc 的末尾，然后将下面的命令运行一次：

```
$ source ~/.bashrc
```

- 用法 创建虚拟环境

  ```
  $ mkvirtualenv env27
  ```

  创建指定解释器的虚拟环境

  ```
  $ mkvirtualenv -p python3.4 env34
  ```

  启动虚拟环境

  ```
  $ workon env27
  ```

  退出虚拟环境

  ```
  $ deactivate
  ```

  删除虚拟环境

  ```
  $ rmvirtualenv env27
  ```

然后最后，以上都没成功，以后再研究；最后特别简单一条命令成功了
就，`$ python3 -m venv py3`

```
$ cd /opt
$ python3 -m venv py3
$ source /opt/py3/bin/activate

# 看到下面的提示符代表成功，以下所有命令均在该虚拟环境中运行
(py3) [root@localhost py3]$
```

然后在 Jenkins 配置 shell 命令，在调用 robot 前，先进入 Python3 环境：

```
$ source /opt/py3/bin/activate
```

shell 命令最后再退出 Python3 环境：

```
$ deactivate
```

By the way, there's another question

- 在 Linux 下，升级到 Python3.7 以上会有不能使用 pip 的问题，提示 SSL 错误。 网上有很多说法，比如重新编译，加参数，改配置文件等等。
  尝试后都没有成功，最后找到原因, 感觉这个还挺靠谱。
  然后使用了最简单的办法...安装 Python3.6, 避免不能使用 pip 的问题，有空再研究 3.7 的问题。

- 在创建 Python 虚拟环境的方法中，pip 使用需要注意
  首先是需要注意创建虚拟环境时，配置的参数，是独立的环境内单独使用 pip 包，还是共用；
  其次是有可能会出现提示，不可识别 pip 命令。此时需要进入到虚拟环境中的 python 目录并替换使用：

  ```
  ./pip   # OR
  sudo ./pip  # ./pip 是运行当前目录下的pip进行安装
  ```

  即可解决

在 Windows 中，可以这样
暂时还没有在 Windows 下并存的需要，方法有待验证

Reference

在Linux上安装Python3
linux-Centos7安装python3并与python2共存
linux下切换python2和python3（转）
python------virtualenv&virtualenvwrapper的使用
Linux 中python2 、python3 共存
关于Ubuntu中python虚拟环境无法将安装包pip安装到虚拟环境中

2018.11.24