

关于 Jetson 的系统烧录与环境配置

李祖乐

信息科学与工程学部通信工程专业

目录

一	准备	2
二	进行烧录（参考瑞泰教程）	2
2.1	系统软件包的下载	2
2.2	在PC端Ubuntu系统进行烧录环境准备	4
3.1	系统烧录	5
3.2	配置板子系统	7
三	进行系统迁移（视情况选择）	7
3.1	方法一	7
3.2	方法二	8
四	安装Jepack	10
4.1	基本步骤	10
4.2	问题及解决	13
五	安装系统所需的包	15
5.1	配置cuda环境变量	15
5.2	配置系统级安装包	15
5.3	安装配置Python	15
5.4	安装jtop管理GPU	18
5.5	安装virtualenv来创建虚拟环境	18
六	安装pytorch以及torchvision	18
6.1	基本操作	18
6.2	问题及解决	19
七	安装opencv	19
7.1	情形一	20
7.2	情形二	20

关于Jetson的系统烧录与环境配置

作者: 李祖乐

一、准备

1. **Ubuntu**电脑 **or** 虚拟机 (**Ubuntu**系统)
2. 能够进行数据传输的**Micro-USB**数据线
3. 显示屏、**HDMI**转接线、键鼠

二、进行烧录（参考瑞泰教程）

2.1 系统软件包的下载

2.1.1 烧录所需文件集中在这两个文件夹中，在本部分我选择安装**LT4 R32.7.1**版本



2.1.2 根据**Jetson**类型进行选择



2.1.3 根据载板型号进行选择

当前位置: 资源目录 > 软件包(Software) > 载板支持包(Carrier Bo... > Xavier NX



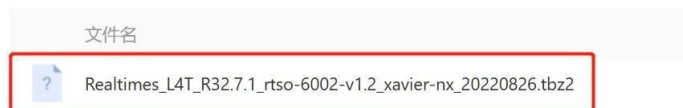
2.1.4 随便选择一个版本

当前位置: 资源目录 > ... > 载板支持包(Carrier Bo... > Xavier NX > RTSO-6002 or 6002E



2.1.5 下载对应文件, 其中rtso-6002对应位置即为载板型号

当前位置: 资源目录 > ... > Xavier NX > RTSO-6002 or 6002E > R32.7.1



2.1.6 选择对应版本的L4T文件



2.1.7 下载相应文件



2.2 在PC端Ubuntu系统进行烧录环境准备

2.2.1 将上述文件拷贝至烧录主机同一目录下

2.2.2 解压 Linux Driver Package

```
$ tar -xvfJetson_Linux_R32.7.1_aarch64.tbz2
```

2.2.3 设置根文件系统

1. 进入Linux Driver Package 的根文件系统目录

```
$ cd <your_L4T_root>/Linux_for_Tegra/roofs
```

2. 解压 the Root File System :

```
$ sudo tar -jxpf ../../Tegra_Linux_Sample-Root-Filesystem_R32.7.1_aarch64.tbz2
```

2.2.4 安装 BSP 支持包

1. 将 Realtime-L4T-.tar 包解压到与 Linux_for_Tegra 文件夹同级目录下面，使用命令：

```
$ tar -xvfRealtime-L4T-<version>.tar
```

2. 进入到 Realtime-L4T 文件夹，运行：

```
$ sudo ./install.sh
```

安装成功会有 **success** 提示！

3. 运行 `apply_binaries.sh` 脚本拷贝 **NVIDIA** 用户空间库进入目标文件系统

```
$ cd ../Linux_for_Tegra/  
$ sudo ./apply_binaries.sh
```

3.1 系统烧录

3.3.1 先c将板子进入**recovery**模式，然后将板子与主机通过**Micro-USB**数据线连接，在**PC**端的**Linux_for_Tegra**目录下进行如下操作：

```
$ sudo ./flash.sh realtimes/rtso-<model> mmcblk0p1
```

注意：**rtso**-指的是板子型号，例如我使用的板子是**rtso-6002E-v1.2**，则命令为：

```
$ sudo ./flash.sh rtso-6002e-1.2vmmcblk0p1
```

3.3.2 可能遇到的问题

1. 板子型号查看，一般来说板子上会带有型号，若找不到可以咨询卖方



2. 关于输入选项，如下，可以发现红框圈中的有很多型号选项，其中**rtso-6002e**表示**rtso-6002-emmc**板，**rtso-6002**则是另一种类型的板子，并且**6002-emmc**下还有**v1.2**、**v1.3**等型号，根据板子进行选择。（一定不要弄错，这些文件是与主板相配的）

```
qw@ubuntu:~/Downloads/327/Linux_for_Tegra$ sudo ./flash.sh rtso-6002  
rtso-6002e-v1.2.conf          rtso-6002-v1.2.conf  
rtso-6002e-v1.2-imx462.conf   rtso-6002-v1.2-imx462.conf  
rtso-6002e-v3.0.conf          rtso-6002-v3.0.conf  
rtso-6002e-v3.0-imx462.conf   rtso-6002-v3.0-imx462.conf  
qw@ubuntu:~/Downloads/327/Linux_for_Tegra$ sudo ./flash.sh rtso-6002e-v1.2 mmcblk0p1
```

3.3.3 报错一

```
qw@ubuntu:~/Downloads/327/Linux_for_Tegra$ sudo ./flash.sh rtso-6002e-v1.2 mmcblk0p1  
[sudo] password for qw:  
#####  
# L4T BSP Information:  
# R32 , REVISION: 7.1  
#####  
Error: probing the target board failed.  
Make sure the target board is connected through  
USB port and is in recovery mode.
```

如上报错需考虑两个原因：

1. 主机与板子进行数据传输的**Micro-USB**数据线是否具有传输数据的作用，可以使用**lsusb**查看是否

含有**Nvidia Crop**;

2. 板子是否进入了**recovery**模式。

3.3.4 报错二

```
nn@ubuntu: ~/Downloads/Linux_for_Tegra
1.7573] adding BCH for recovery.img
1.8256] adding BCH for tegra194-p3668-all-p3509-0000-6002e.dtb.rec
2.5654]
2.5654] Filling MBI storage info
2.5654] Generating br-bct
2.5681] Performing cfg overlay
2.5682] ['tegra194-mbi-bct-memcfg-p3668-0001-a00.cfg', 'tegra194-memcfg-sw-override.cfg']
2.5686] sw_memcfg_overlay.pl -c tegra194-mbi-bct-memcfg-p3668-0001-a00.cfg -s tegra194-memcfg-sw-override.cfg -o /home/nn/Downloads/Linux_for_Tegra/bootloader/8724/tnp4wj7y531.cfg
2.6814]
2.6816] Updating dev and MSS params in BR BCT
2.6817] tegrabct_v2 --dev_param tegra194-br-bct-qspl.cfg --sdram /home/nn/Downloads/Linux_for_Tegra/bootloader/8724/tnp4wj7y531.cfg --brbct br_bct.cfg --sfuse tegra194-mbi-soft-fuses-l4t.cfg --chip
0x19 0
2.7198]
2.7198] Updating bl info
2.7231] tegrabct_v2 --brbct br_bct_BR.bct --chip 0x19 0 --updateblinfo flash.xml.bin
2.7331]
2.7332] Generating signatures
2.7663] tegrasign_v3.py --key None --list images_list.xml --pubkeyhash pub_key.key
2.7664] Assuming zero filled SBK key
3.3566] Generating br-bct
3.3589] Performing cfg overlay
3.3589] ['/home/nn/Downloads/Linux_for_Tegra/bootloader/8724/tnp4wj7y531.cfg']
3.3589] Updating dev and MSS params in BR BCT
3.3590] tegrabct_v2 --dev_param tegra194-br-bct-qspl.cfg --sdram /home/nn/Downloads/Linux_for_Tegra/bootloader/8724/tnp4wj7y531.cfg --brbct br_bct.cfg --sfuse tegra194-mbi-soft-fuses-l4t.cfg --chip
0x19 0
3.3930]
3.3930] Updating bl info
3.3968] tegrabct_v2 --brbct br_bct_BR.bct --chip 0x19 0 --updateblinfo flash.xml.bin --updatesig images_list_signed.xml
3.4015]
3.4015] Updating snd info
3.4048] tegrabct_v2 --brbct br_bct_BR.bct --chip 0x19 0 --updatesndinfo flash.xml.bin
3.4066]
3.4066] Updating Odmdata
3.4097] tegrabct_v2 --brbct br_bct_BR.bct --chip 0x19 0 --updatefields Odmdata=0x88198000
3.4114]
3.4115] Get Signed section of bct
3.4142] tegrabct_v2 --brbct br_bct_BR.bct --chip 0x19 0 --listbct bct_list.xml
3.4158]
3.4183] tegrasign_v3.py --key None --list bct_list.xml --pubkeyhash pub_key.key --getmontgomeryvalues montgomery.bin
3.4198] /usr/bin/env: 'python': No such file or directory
3.4244]
Error: Return value 127
Command tegrasign_v3.py --key None --list bct_list.xml --pubkeyhash pub_key.key --getmontgomeryvalues montgomery.bin
Failed flashing t186ref.
nn@ubuntu:~/Downloads/Linux_for_Tegra$
```

仔细观察可以发现问题在于一个 `.py` 文件执行错误，可以使用命令 `$ sudo apt install python2.7 python3 python` 安装**python**，然后重新烧录解决。

3.2 配置板子系统

由于瑞泰板子是上电自启动的，将板子和显示屏以及键鼠连接，进入类似**Ubuntu**系统开机配置界面，除了地区选择上海外，其余保持默认即可。能够正常开机，则表明烧录成功。随后可以发现，其图形化界面和**Ubuntu**几乎没有什么区别。

三、进行系统迁移（视情况选择）

3.1 方法一

3.1.1 查看SSD设备名称

系统启动前，将**SSD**插入到板子的内存接口处（由于瑞泰板子自身装有一个**120GB**的内存卡，因此可以直接使用其自身的**SSD**卡即可）。系统启动后，使用 `$ sudo fdisk -l` 命令查看**SSD**设备名称，例如：**nvme0n1**、**mmcblk1**，本文使用**mmcblk1**为例。

3.1.2 对SSD进行格式化

如果**SSD**之前没有进行格式化，需要把**SSD**格式化后再使用。对于已挂载的**SSD**卡，需要使用**umount**卸载**SSD**卡，再格式化。卸载命令：`sudo umount /dev/mmcblk1`；格式化命令：`$ sudo mkfs.ext4 /dev/mmcblk1`

3.1.3 创建一个新的GPT分区

```
$ sudo parted /dev/mmcblk1 mklabel gpt
```

3.1.4 添加分区

```
$ sudo parted /dev/mmcblk1 mkpart primary 0GB <Size>
```

Size是分区的大小，最小**8GB**，建议可以将**SSD**卡内存全部添入

例如：准备分区的大小是**50GB**，则命令是：

```
$ sudo parted /dev/mmcblk1 mkpart primary 0GB 50GB
```

添加完分区后，使用 `$ sudo fdisk -l` 可以看到**mmcblk1**下新增一个分区，名为：**mmcblk1p1**

3.1.5 格式化分区

```
$ sudo mkfs.ext4 /dev/mmcblk1p1
```

把分区格式化为**ext4** 格式

3.1.6 拷贝roofs到SSD

```
$ sudo dd if=/dev/mmcblk0p1 of=/dev/mmcblk1p1 bs=1M
```


其中**mmcblk0p1**是系统原先所在位置，**mmcblk1p1**则是我们的转移目标区域

3.1.7 修复分区

```
$ sudo -s
$ fsck.ext4 /dev/mmcblk1p1
```

若遇到输入**yes or no**，请全部输入**yes**

3.1.8 调整系统分区大小

```
$ sudo resize2fs /dev/mmcblk1p1
```

3.1.9 烧写从SD卡启动系统

```
$ sudo ./flash.sh realtimes/rtso-<model> mmcblk1p1
```

这一步骤和烧录步骤一模一样，需要注意的是目录位置、烧录系统文件与最开始烧录时是一样的，且目标位置名称变化了

重新烧录完后，进入系统后，输入 `$ df -h` 可以看到**mmcblk1p1**成为根目录，系统已从SD卡启动。

3.2 方法二

3.2.1 将SD卡进行格式化操作并挂载到系统下，操作如下：

系统迁移的前提是将SD卡挂载到系统下，先需要使用如下命令将SD卡格式化为**EXT4**格式：

```
$ sudo mke2fs -t ext4 /dev/mmcblk1p1
```

然后将SD卡挂载到**/mnt**下：

```
$ sudo mount /dev/mmcblk1p1 /mnt
```

3.2.2 克隆所需文件：

```
$ git clone https://github.com/jetsonhacks/rootOnNVMe.git
```

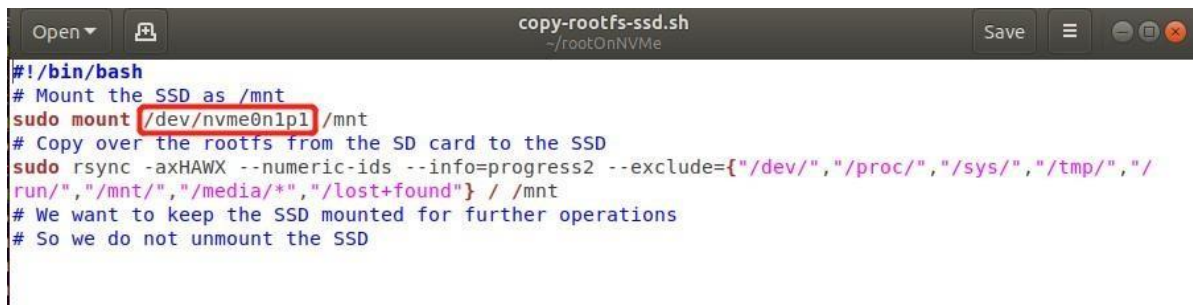
3.2.3 进入下载的文件夹中，编辑脚本文件**copy-rootfs-ssd.sh**，然后运行该脚本，将系统复制到SSD中。

```
cd ./rootOnNVMe
```

```
sudo gedit ./copy-rootfs-ssd.sh
```

将文件中的 `/dev/nvme0n1p1` 修改为 `/dev/mmcblk1`

```
sudo ./copy-rootfs-ssd.sh
```



```
Open  copy-rootfs-ssd.sh  Save
~/rootOnNVMe
#!/bin/bash
# Mount the SSD as /mnt
sudo mount /dev/nvme0n1p1 /mnt
# Copy over the rootfs from the SD card to the SSD
sudo rsync -axHAWX --numeric-ids --info=progress2 --exclude={"/dev/", "/proc/", "/sys/", "/tmp/", "/run/", "/mnt/", "/media/*", "/lost+found"} / /mnt
# We want to keep the SSD mounted for further operations
# So we do not unmount the SSD
```

3.2.4 进入到 `rootOnNVMe/data` 目录下，修改其中的脚本文件：`setssdroot.sh` 与 `setssdroot.service`，将其中的 `/dev/nvme0n1p1` 修改为 `/dev/mmcblk1`，方法同上。

```
qw@ubuntu:~/rootOnNVMe/data$ ls
setssdroot.service  setssdroot.sh
```

3.2.5 回到**rootOnNVMe**目录下，执行以下命令：

```
sudo ./setup-service.sh
```

成功后如下图：

```
Created symlink /etc/systemd/system/default.target.wants/setssdroot.service → /etc/systemd/system/setssdroot.service.
Service to set the rootfs to the SSD installed.
Make sure that you have copied the rootfs to SSD.
Reboot for changes to take effect.
```


3.2.6 重新启动jetson板后，使用命令 `$ df -h` 查看，可以发现系统已经完成迁移

四、安装Jepack

4.1、基本步骤

4.1.1 安装前信息确认以及更新软件源（在板子上进行）

给Xavier NX安装软件之前需先确定jetson设备系统L4t版本，因为NVIDIA Jetpack跟该版本号具有一定的对应关系，如果版本号不对应会导致出现一些异常。具体的对应关系可以参考Jetpack的说明：

在Xavier NX设备上使用以下命令可以查看系统的L4T版本号：

```
$ head -n 1 /etc/nv_tegra_release
```

4.1.2 进入载板系统，打开SystemSettings-->Software&Updates>Ubuntu Software

4.1.3 下载安装Jetpack/sdkmanager并运行（在PC端Ubuntu系统上进行）

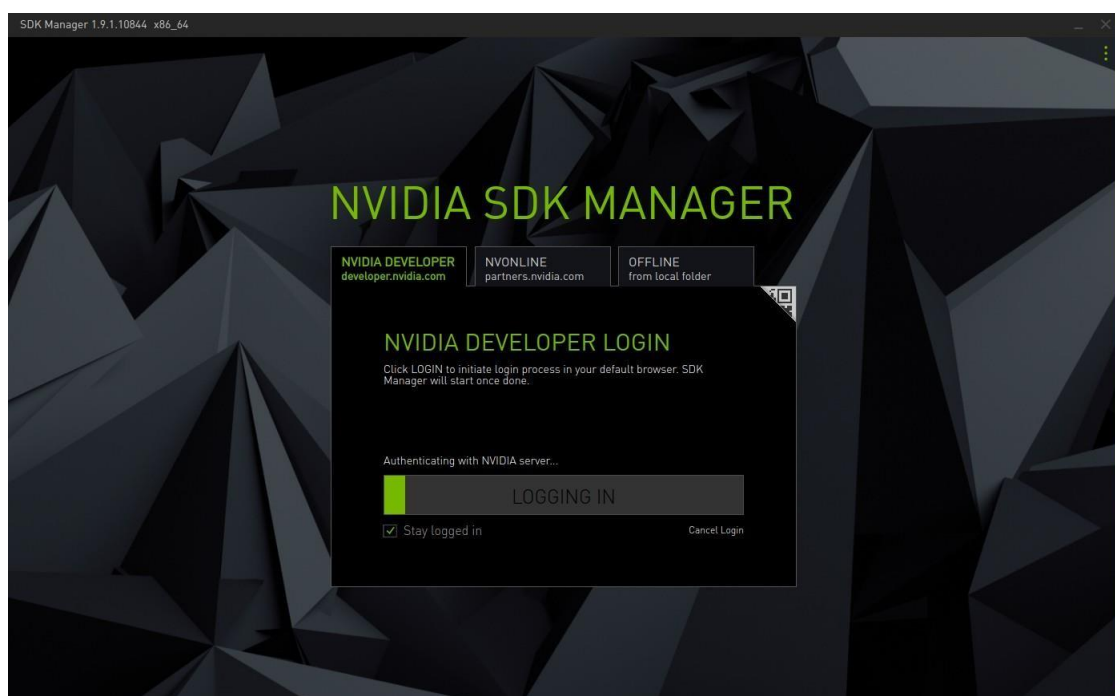
选择安装最新版本的sdkmanager下载安装

安装命令：`$ sudo dpkg -i sdkmanager<.....>.deb` or `$ sudo apt install ./sdkmanager-<.....>.deb`

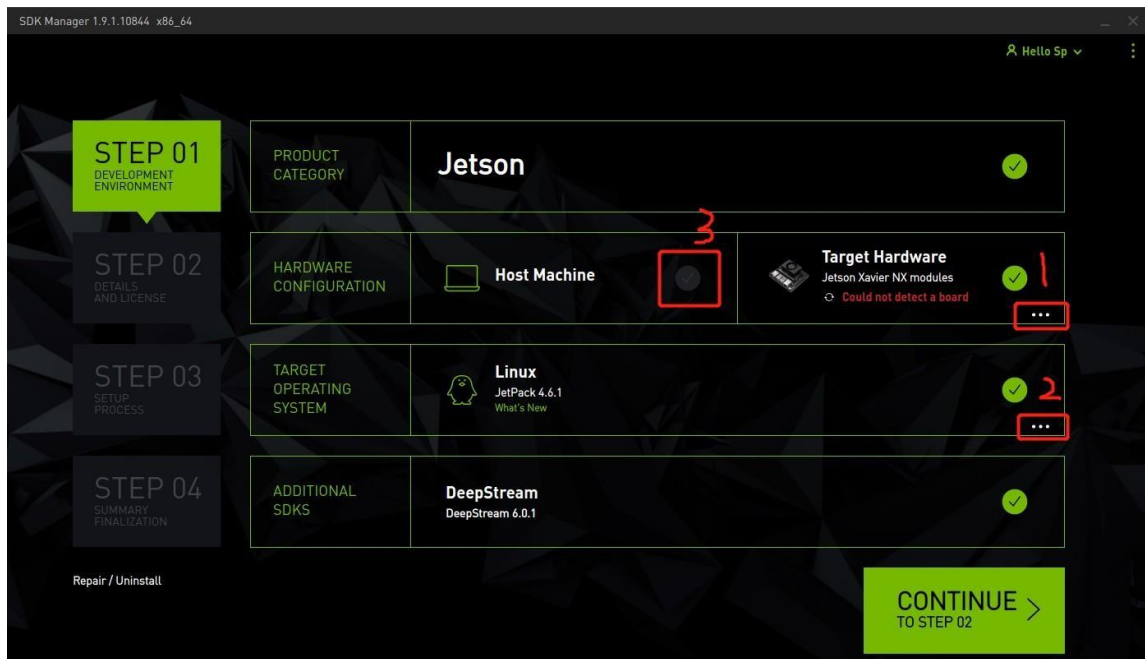
运行命令：`$ sdkmanager`

运行界面和操作如下：

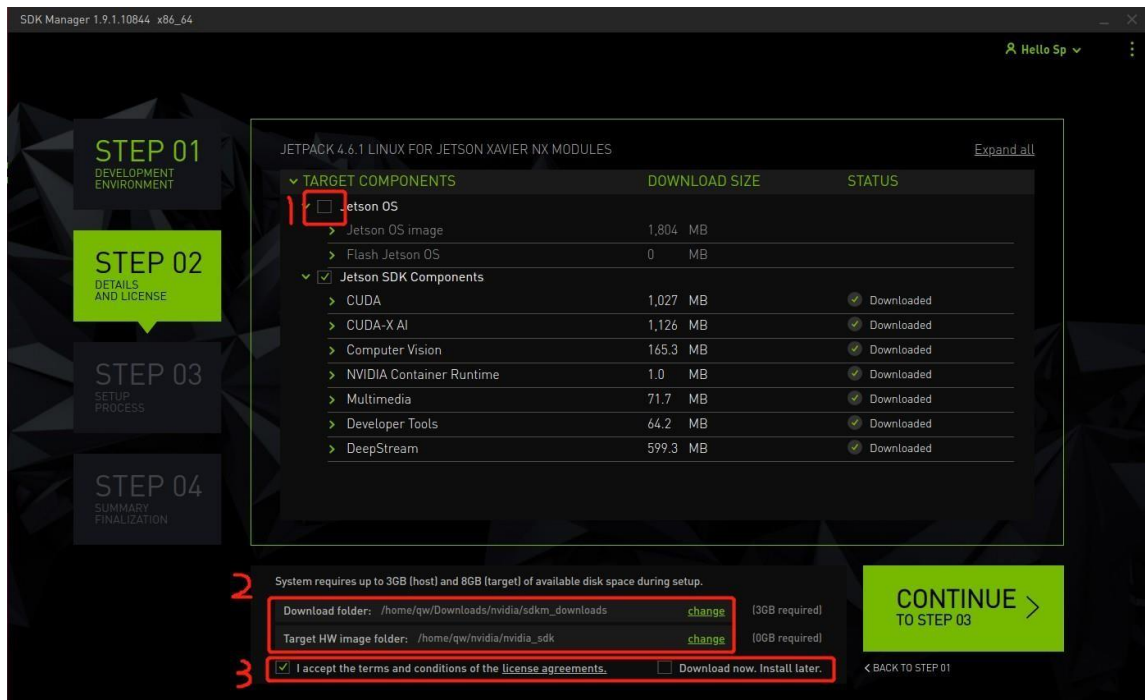
1. 登录界面（低版本可能无法登录）



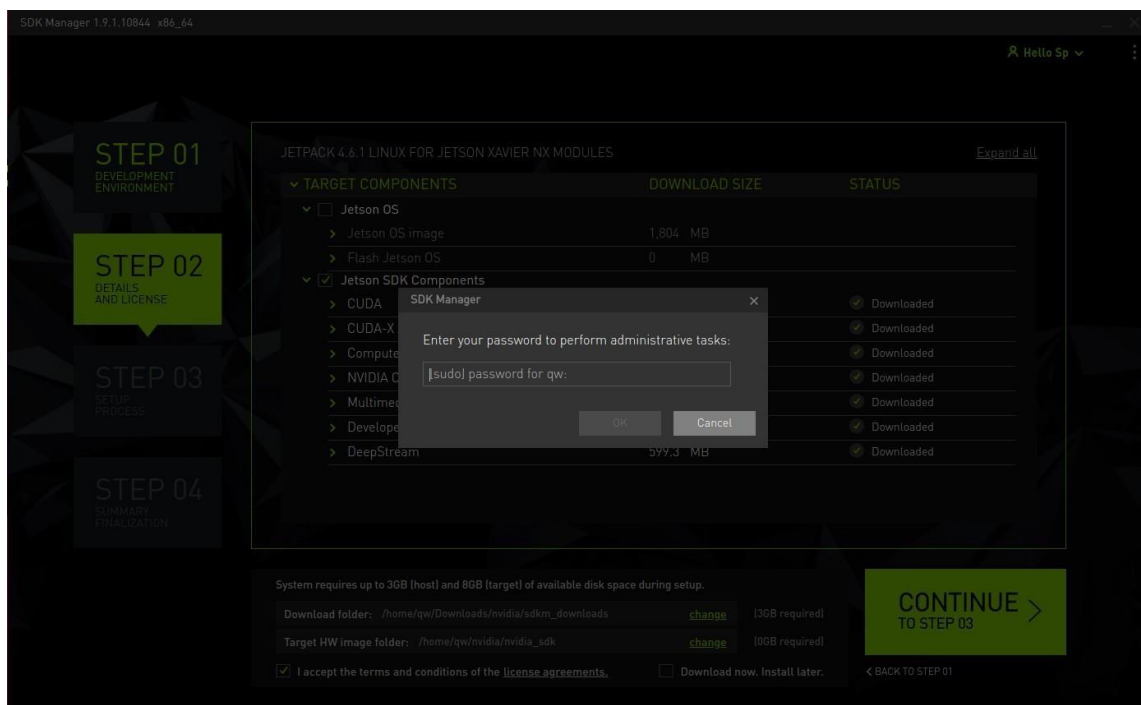
2. 点击圈中的1、2，进行选择Jetson类型以及与LT4相对应的Jetpack版本，3取消勾选。如果2中未发现与其对应的Jetpack版本，则运行sdkmanager的命令改为：`$ sdkmanager --archivedversions`，其它操作不变。



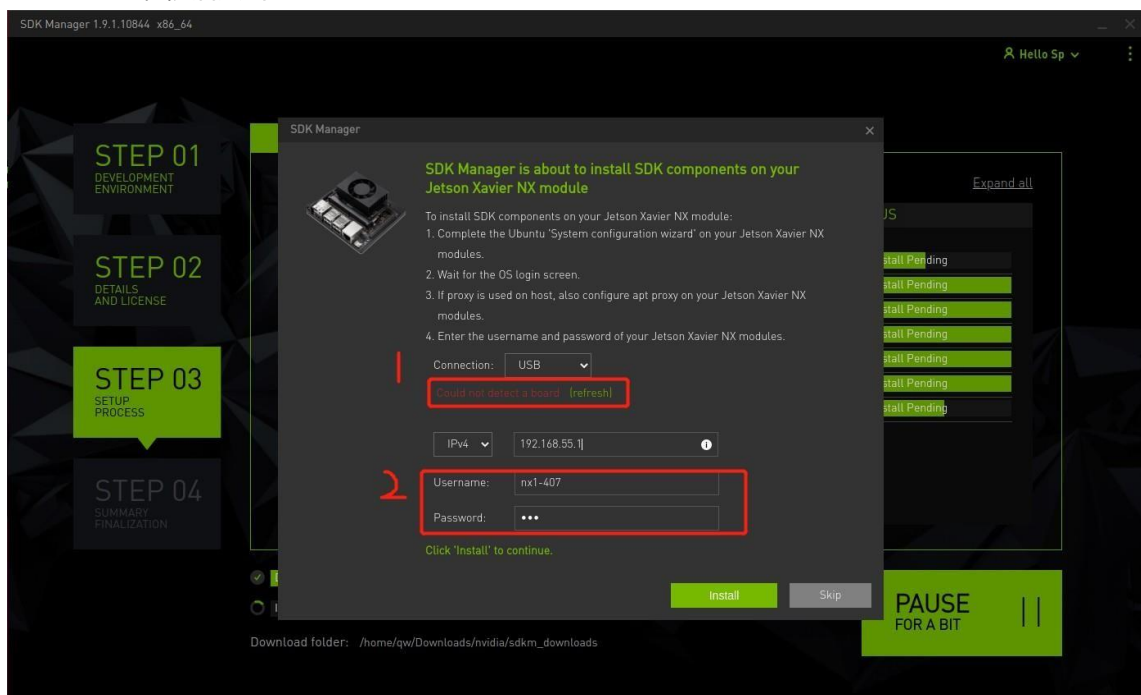
3. 由于已经安装过系统，圈中的1取消勾选，圈中的2可以选择合适的位置（默认不变），圈中的3注意取消勾选Download now.Install later.点击continue。



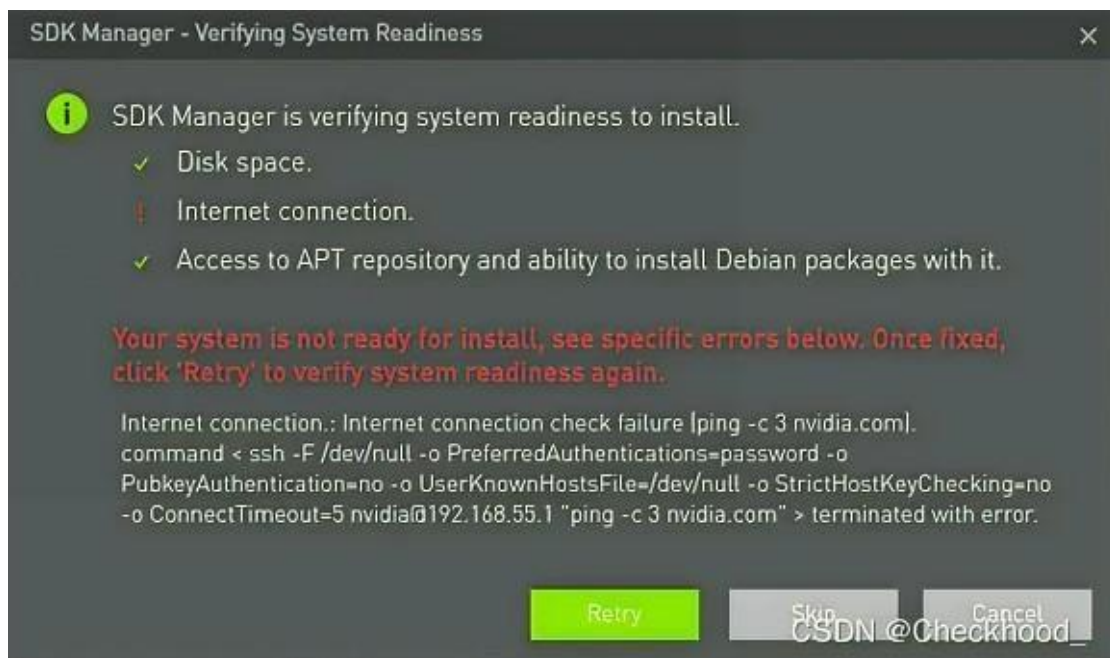
4. 输入PC端Ubuntu主机密码



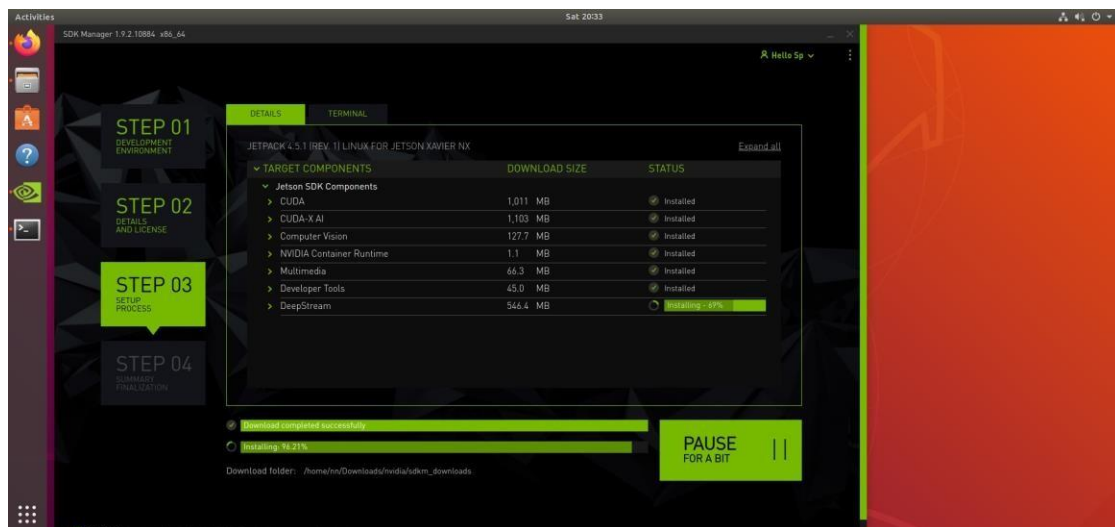
5. 圈中的1，实际上应是显示有设备连接的；圈中的2填写板子Ubuntu系统的Username和密码；随后点击Install。



6. 接着进入以下界面，这些指标都是板子的指标，如内存是否足够、网络是否连接等。



7.最后出现接下来画面即成功



4.2 问题及解决

4.2.1 如上图，如果出现**Internet connection**问题，按照其指示可以采取如下方法解决：

1. 在**jetson**板上终端输入：`$ ping -c 3 www.nvidia.com`

```
nvidia@ubuntu:~$ ping -c 3 www.nvidia.com
PING e33907.a.akamaiedge.net (23.48.214.59) 56(84) bytes of data.
64 bytes from a23-48-214-59.deploy.static.akamaitechnologies.com (23.48.214.59): icmp_seq=1 ttl=128 time=64.4 ms
64 bytes from a23-48-214-59.deploy.static.akamaitechnologies.com (23.48.214.59): icmp_seq=2 ttl=128 time=66.9 ms
64 bytes from a23-48-214-59.deploy.static.akamaitechnologies.com (23.48.214.59): icmp_seq=3 ttl=128 time=64.0 ms

--- e33907.a.akamaiedge.net ping statistics ---
```

2. 将出现的**Ip**地址放到**jetson**板子的**/etc/hosts**文件中

3. 使用命令：`$ sudo gedit /etc/hosts`

在文件中添加一行：`23.48.214.59 nvidia.com`

4. 保存后点击**Retry**即可。

如果上述方法行不通，则可能是**sdkmanager**版本不是最新造成的。

4.2.2 若出现第三个**Accept to APT.....**错误，则可能是**jetson**板上的源未更新造成的。

1. 可以在**jetson**板上依次输入命令：

```
$ sudo apt-get update & $ sudo apt-get upgrade
```

2. 更新完后**Retry**即可。

3. 有时光多**Retry**几次可能就会成功。

若上述操作失效，则可以替换**Jetson**板上的**Ubuntu**源（注意是**Jetson**源），重新更新尝试一下；或者退出**sdkmanager**重新弄一遍。

五、安装系统所需的包

5.1 配置cuda环境变量

```
vi ~/.bashrc # 打开/.bashrc文件
```

将以下内容输入该文件中（末尾即可）

```
export PATH=/usr/local/cuda10.2/bin:$PATH # cuda后跟的是cuda版本
export LD_LIBRARY_PATH=/usr/local/cuda10.2/lib64:$LD_LIBRARY_PATH
export CUDA_ROOT=/usr/local/cuda10.2 # 这一项不是很确定是否有用
```

5.2 配置系统级安装包

输入以下命令即可

```
sudo apt-get install build-essential make cmake cmake-curses-gui
sudo apt-get install git g++ pkg-config curl zip zlib1g-dev libopenblas-base
libopenmpi-dev
sudo apt-get install libatlas-base-dev gfortran libcanberra-gtk-module
libcanberra-gtk3-module
sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev
sudo apt-get install nano locate screen
#scipy 依赖和 cython
sudo apt-get install libfreetype6-dev
sudo apt-get install protobuf-compiler libprotobuf-dev openssl
sudo apt-get install libssl-dev libcurl4-openssl-dev
sudo apt-get install cython3
sudo apt-get install libxml2-dev libxslt1-dev
#opencv 依赖
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install libxvidcore-dev libavresample-dev
sudo apt-get install libtiff-dev libjpeg-dev libpng-dev
```

5.3 安装配置Python

5.3.1 默认安装python3.6版本


```

sudo apt-get install -y python3-dev python3-testresources python3-setuptools
wget https://bootstrap.pypa.io/pip/3.6/get-pip.py
sudo python3 get-pip.py # 该命令可能会失败，显示python3使用的python3.6版本较低，导致不成功，只要下载对应版本的pip就行，使用以下命令进行替换：
https://bootstrap.pypa.io/pip/3.6/get-pip.py
rm get-pip.py # 然后换pip源
mkdir ~/.pip
vim ~/.pip/pip.conf # 换清华源

```

粘贴以下内容到该文件

```

[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple

```

5.3.2 解决sudo python3 get-pip.py失败，这里我选择使用python3.7版本

1. 安装python3.7

```

sudo apt-get update # 更新(可跳过)
sudo apt-get install python3.7
python3.7 -V # 查看是否安装成功

```

2. 创建软连接，使python3默认指向python3.7

- ◆ 首先把之前的软连接删除：

```

$ sudo rm -rf /usr/bin/python3
which python3.7 # 查看python3.7 安装路径.这里输出/usr/bin/python3.7

```

创建新的软连接：

```

sudo ln -s /usr/bin/python3.7 /usr/bin/python3 # 添加python3的软链接。
/usr/bin/python3.7 即 which python3.7输出的安装路径
python3 -V # 测试是否安装成功

```

3. 切换python版本，

- ◆ 首先查看python文件

```
ls /usr/bin/python # 查看python文件，会出现python2.7、python3.6、python3.7三个版本
```

- ◆ 首先看看系统是否配置过python相关的管理信息

```
update-alternatives --list python # 如果显示：no alternatives for python，表示没有配置。
```

- ◆ 使用以下命令进行配置：

```

sudo update-alternatives --install /usr/bin/python python
/usr/bin/python2.7 1
sudo update-alternatives --install /usr/bin/python python
/usr/bin/python3.6 2
sudo update-alternatives --install /usr/bin/python python
/usr/bin/python3.7 3

```

- ◆ 配置成功，会提示你 in auto mode .这个时候我们再次查看配置

```
update-alternatives --list python
```

- ◆ 会看到我们的三个版本已经成功进行了配置。如果需要切换版本，输入命令


```
sudo update-alternatives --config python # 然后输入相应的版本序号即可
```

- ◆ 最后使用: `python --version` 查看版本号, 可以发现配置成功

5.4 安装jtop管理GPU

输入命令安装**jtop**:

```
sudo -H pip install jetson-stats
```

安装成功后, 输入 `sudo jtop` 即可查看硬件相关信息

5.5 安装virtualenv来创建虚拟环境

1. 安装与配置

- ◆ 使用**pip**下载**virtualenv**

```
sudo pip install virtualenvvirtualenvwrapper
vim ~/.bashrc
```

- ◆ 将以下内容输入其中:

```
#virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs #指定所有的虚拟环境的安装位置
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3.6 #指定解释器, 改为cuda对应的python版本
```

- ◆ 终端输入指令激活**virtualenv**

```
sudo mkdir $HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh #进行激活生效
source ~/.bashrc #重新载入
```

2. virtualenv的使用相关操作

```
mkvirtualenv name # 创建一个环境
mkvirtualenv -p /python目录/python.exe name # 不使用默认python版本、使用指定python版本创建环境
workon name # 激活环境
deactivate # 退出
rmvirtualenv name # 删除
lsvirtualenv # 所有环境
cpvirtualenv source_name dest_name # 复制虚拟环境
```

六、安装pytorch以及torchvision

6.1 基本操作

1. 创建虚拟环境

- ◆ 使用以下命令创建虚拟环境

```
Mkvirtualenv -p /usr/bin/python3.6 torchli # -p 后面跟着的是制定python的版本, 因为有的pytorch对python版本有要求, 所以要指定版本
workon torchli # 进入虚拟环境
```

- 如果遇见 `workon command not find` , 则使用以下命令

```
source virtualenvwrapper.sh
```

- 然后再 `workon torchli`

2. 安装pytorch

- 首先下载**pytorchpipwheels** (对应自己装的**Jatpack**的版本)
- 下载完后, 输入命令:

```
pip install torch-1.10.0-cp36-cp36m-linux_aarch64.whl # 安装pytorch
git clone --branch v0.11.1 https://github.com/pytorch/vision torchvision #
下载0.11.1版本
cd torchvision
export BUILD_VERSION=0.11.1
python setup.py install --user # --user可加可不加
```

3. 验证是否安装成功

- 上述操作完成后, 输入命令:

```
python # 进入python:
import torch # 导入torch库
torch.cuda.is_available() # 验证cuda是否能使用, 输出应为True, 则pytorch安装成功
import torchvision # 导入torchvision库
print(Torchvision.__version__) # 打印版本号, 未报错即成功
```

6.2 问题及解决

6.2.1. 激活环境失败, 报错输出: `workon command not find`

输入命令 `source virtualenvwrapper.sh` # 输入之后再尝试激活即可

6.2.2. `import torch`时报错,**Illegal instruction (core dumped)**

```
# 修改环境变量
sudo gedit /etc/profile
export OPENBLAS_CORETYPE=ARMV8 # 将其加入最后面一行, 然后保存
# 更新环境变量
source /etc/profile
```

6.2.3. `import torchvision`报错

1. 在**`import torchvision`**过程中遇到有关**PIL**库的提示报错

解决方法: 尝试卸载当前的**pillow**, 安装较低版本的**pillow**

输入命令:

```
pip uninstall pillow
pip install pillow==6.1 # 一般来说安装此版本即可解决问题
```

2. 由于当前在**torchvision**文件夹下安装的**torchvision**, 若不退出此路径也会导致**`import torchvision`**报错, 报错提示路径问题

解决方法: 退出当前文件夹路径, 输入以下命令:

```
cd .. # 返回上级路径, 然后再次尝试import torchvision
```

七、安装opencv

7.1 情形一

1. 首先在终端执行以下指令查找编译好的**cv2**库文件的路径

```
sudo find / -iname "*cv2*" # 得到类似路径 /usr/lib/python3.6/dist-packages/cv2/python-3.6/cv2.cython-36m-aarch64-linux-gnu.so, 重点是cv2.cython-36m-aarch64-linux-gnu.so文件要求一致
```

2. 随后进入虚拟环境的**site-packages**文件夹下,并链接到查找到的**cv2**库文件路径即可

```
cd /home/nx407/.virtualenv/torchli/lib/python3.6/site-packages # 其中nx407是用户名、torchli是我建立的虚拟环境名
ln -s /usr/lib/python3.6/dist-packages/cv2/python-3.6/cv2.cython-36m-aarch64-linux-gnu.so cv2.so # 注意ln, link的意思, 不是大写i
```

3. 安装完成后, 在虚拟环境中执行下列指令以确保**python**能正确调用**cv2**。

```
python //进入python
import cv2
cv2.__version__ //若安装成功且能正常调用, 此处能输出安装的从v版本
quit() //退出python
```

7.2 情形二

1. 查看已预安装**Opencv**版本

```
pkg-config --modversion opencv
```

2. 卸载原**Opencv**

- ◆ 如果是自己之前安装的话, 就找到当初安装**Opencv**的**build**文件夹路径, 然后进入该**build**目录执行卸载操作:

```
cd ../opencv-4.x.x/build
sudo make uninstall
cd ../opencv-4.x.x
sudo rm -r build
```

- ◆ 若不是则执行以下操作:

```
sudo apt-get purge libopencv*
sudo apt-get purge python-numpy
sudo apt autoremove // 删除其他未使用的apt包, 可有可无
sudo apt-get update
sudo apt-get dist-upgrade
```

3. 下载**Opencv3.4.5.zip**文件

4. 安装一些相关库和包

```

sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev
libavformat-dev libswscale-dev
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev
libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
sudo apt-get install --only-upgrade g++-5 cpp-5 gcc-5

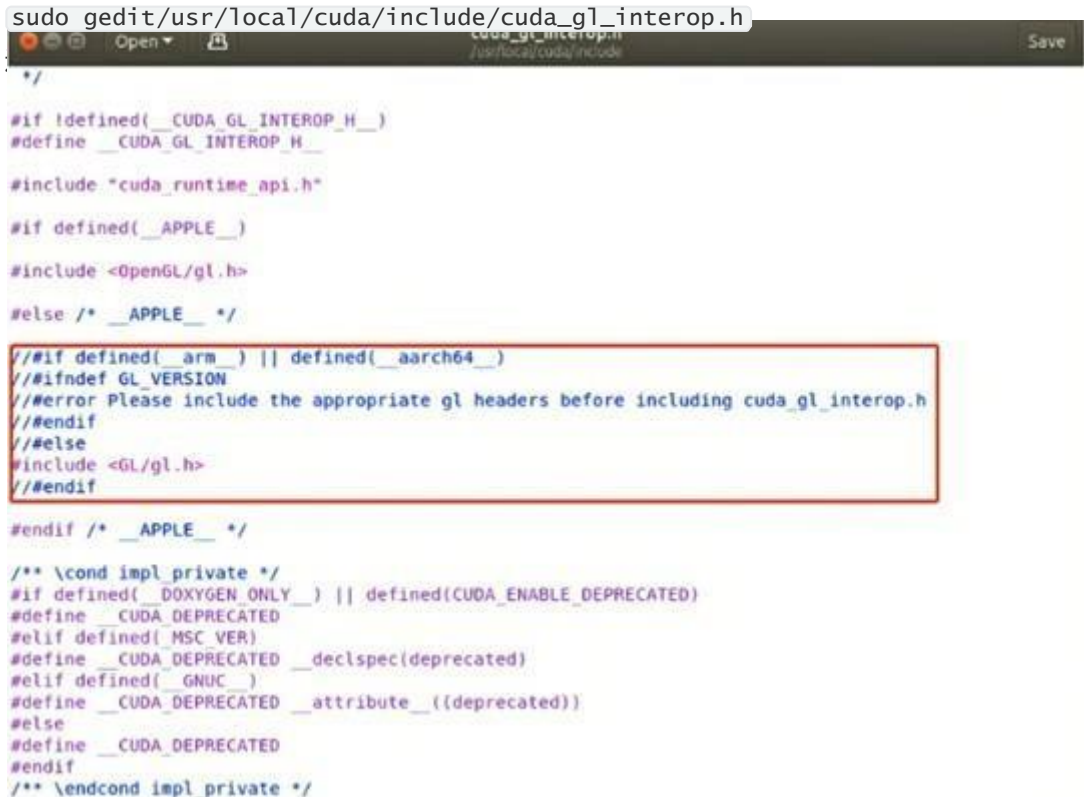
```

如果上述指令失败的话，可以尝试网上换源，如下
更新后再安装Qt5

`sudo apt-get install qt5-default` # qt5必须要安装成功，不过丢包概率不大，都能装上的

5. CUDA部分源码的修改

先找到 `cuda_gl_interop.h` 文件，一般都是在 `/usr/local/cuda/include` 里
然后在命令框输入



```

sudo gedit /usr/local/cuda/include/cuda_gl_interop.h

```

```

/*
 *
 *
 */

#ifndef __CUDA_GL_INTEROP_H__
#define __CUDA_GL_INTEROP_H__

#include "cuda_runtime_api.h"

#if defined( __APPLE__ )

#include <OpenGL/gl.h>

#else /* __APPLE__ */

// #if defined( __arm__ ) || defined( __aarch64__ )
// #ifndef GL_VERSION
// #error Please include the appropriate gl headers before including cuda_gl_interop.h
// #endif
// #else
// #include <GL/gl.h>
// #endif

#endif /* __APPLE__ */

/** \cond impl private */
#if defined( __DOXYGEN_ONLY__ ) || defined( CUDA_ENABLE_DEPRECATED )
#define CUDA_DEPRECATED
#elif defined( _MSC_VER )
#define CUDA_DEPRECATED __declspec(deprecated)
#elif defined( _GNUC__ )
#define CUDA_DEPRECATED __attribute__((deprecated))
#else
#define CUDA_DEPRECATED
#endif
/** \endcond impl private */

```

找到红框内代码，并按照图中代码更改为以上内容，然后点**save**保存，退出。

6. 开始编译安装opencv-3.4.5

找到**opencv-3.4.5**所在文件夹（即前文解压后的文件夹），然后**cd**到**build**文件夹里面。
控制台输入：

```

cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local \
-D WITH_CUDA=ON -D CUDA_ARCH_BIN="6.2" -D CUDA_ARCH_PTX="" \
-D WITH_CUBLAS=ON -D ENABLE_FAST_MATH=ON -D CUDA_FAST_MATH=ON \
-D ENABLE_NEON=ON -D WITH_LIBV4L=ON -D BUILD_TESTS=OFF \
-D BUILD_PERF_TESTS=OFF -D BUILD_EXAMPLES=OFF \
-D WITH_QT=ON -D WITH_OPENGL=ON

```

接着输入：

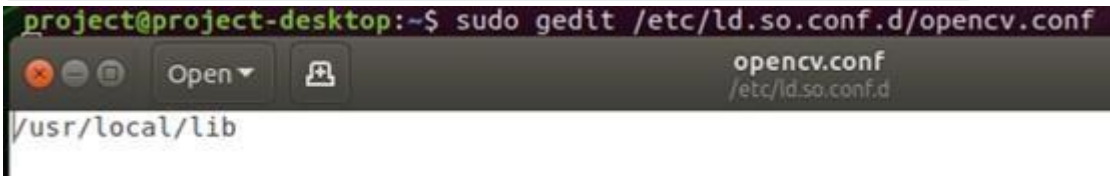
```

sudo make
sudo make install

```

7. 配置opencv环境:

`sudo gedit /etc/ld.so.conf.d/opencv.conf` # 创建并打开该文件, 输入以下内容



执行下面的命令, 使得刚才配置的路径生效
`sudo ldconfig`

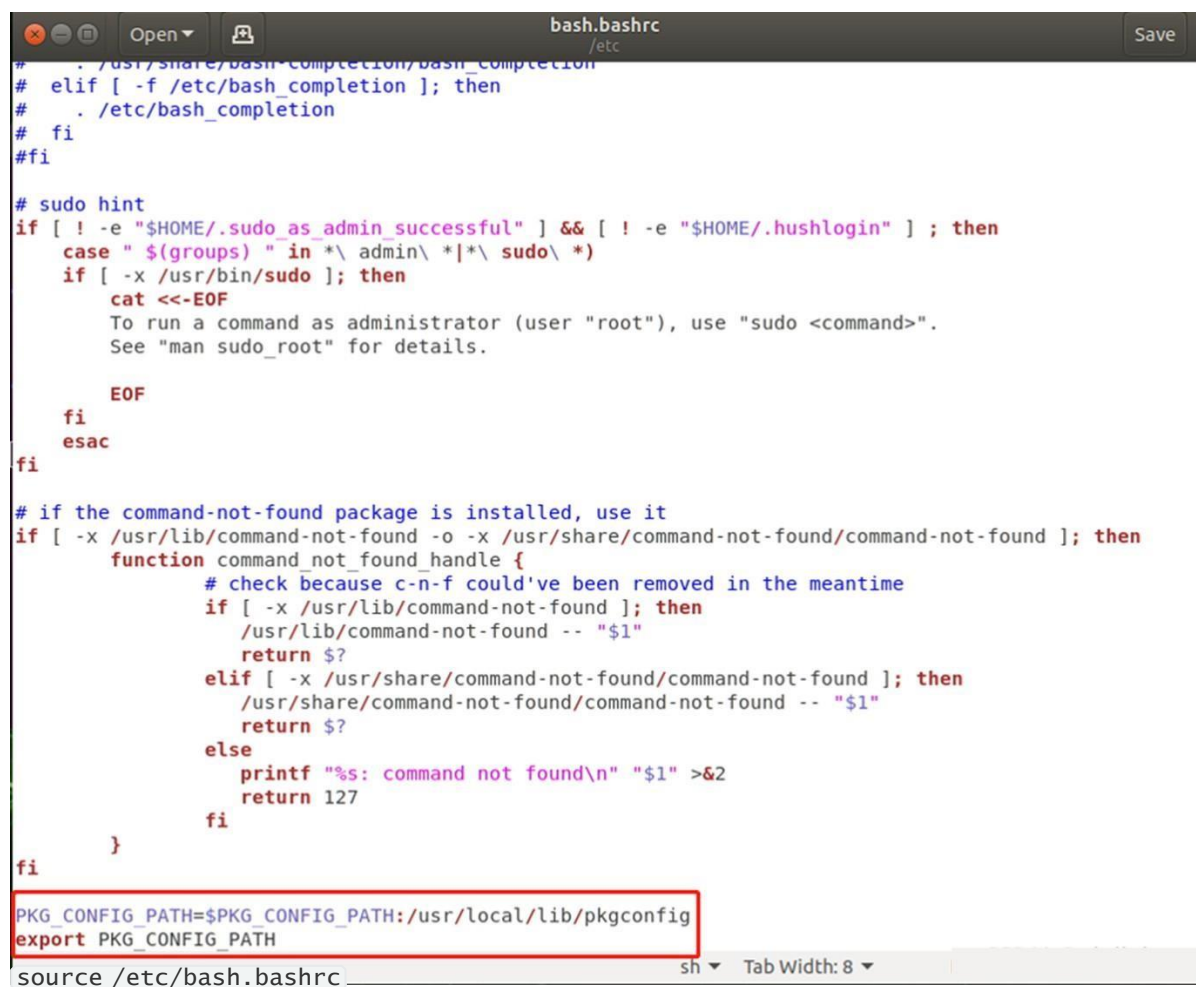
然后打开**bash.bashrc**文件

`sudo gedit /etc/bash.bashrc`

在打开文件的最后加入如下命令

```
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

如图:



可以先在**jtop**查看**opencv**的安装版本, 正确安装后会显示:

```

# if the command-not-found package is installed, use it
if [ -x /usr/lib/command-not-found -o -x /usr/share/command-not-found/command-not-found ]; then
    function command_not_found_handle {
        # check because c-n-f could've been removed in the meantime
        if [ -x /usr/lib/command-not-found ]; then
            /usr/lib/command-not-found -- "$1"
            return $?
        elif [ -x /usr/share/command-not-found/command-not-found ]; then
            /usr/share/command-not-found/command-not-found -- "$1"
            return $?
        else
            printf "%s: command not found\n" "$1" >&2
            return 127
        fi
    fi
fi

PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH

```

9. 之后连接一个摄像头，在终端输入，测试**opencv**是否能正常使用：

```

cd /opencv-3.4.5/samples/cpp/example_cmake
cmake .
make
# 成功make后，执行
./opencv_example

```

10. 上述 `./opencv_example` 命令如果失败，原因可能是摄像头的索引错误，
 输入 `lsusb`
 查看摄像头，发现**video1**
 换一个**usb**接口插入即可，再执行上述命令，可以发现摄像头成功打开。