

# Python环境搭建简易指南

---

主要为Windows、Linux和Mac系统下的Conda-Python-CUDA环境搭建方法。

--By ljc

## Python环境搭建简易指南

1. Python版本和解释器
2. Python模块、包、库和环境
3. 包管理器
4. 搭建环境的前置知识
  - 4.1. 命令行
  - 4.2. 虚拟机/双系统
    - 4.2.1. 系统选择
    - 4.2.2. 虚拟机选择
    - 4.2.3. 双系统安装
  - 4.3. Vim编辑器
    - 4.3.1. Vim基本概念
    - 4.3.2. 使用Vim编辑文件的方法
  - 4.4. 环境变量
    - 4.4.1. 什么是环境变量
    - 4.4.2. 为什么要有环境变量
    - 4.4.3. Windows添加环境变量
    - 4.4.4. Mac/Linux添加环境变量
5. conda的安装
  - 5.1. conda基本介绍
  - 5.2. 下载位置
  - 5.3. 安装
    - 5.3.1. exe安装
    - 5.3.2. sh安装
6. conda的基本用法
  - 6.1. conda换源
  - 6.2. conda环境管理
  - 6.3. conda包管理
  - 6.4. 找到你的虚拟环境的解释器
  - 6.5. 效率优化
  - 6.6. conda和pip混用
  - 6.7. 导出导入环境
7. 显卡支持
  - 7.1. Windows
    - 7.1.1. 驱动
    - 7.1.2. CUDA
    - 7.1.3. cuDNN
  - 7.2. Linux
    - 7.2.1. 显卡驱动
    - 7.2.2. CUDA
    - 7.2.3. cuDNN
  - 7.3. pytorch测试显卡

# 1.Python版本和解释器

---

- Python的主要版本为Python2.x和Python3.x。
  - Python2已于2020年末停止更新，最后一个子版本是Python2.7。Python2和Python3间存在较多的语法差异，所以对于新的项目请勿选择使用Python2版本进行开发。而如果想要运行一些使用Python2开发的代码，官方提供了2to3脚本用来快速转换其中的语法内容，位于Python根目录的Tools/scripts目录下，详细用法请查看[链接](#)。
  - Python3于2008年推出3.0版本，从Python3.4开始约每年发布一个新的子版本，每个子版本的维护时间为五年，如2015年首次发行的Python3.5就于2020年10月宣布停止维护。每次子版本更新会增加一些新的语法和标准库，目前最新的是2020年推出的Python3.9。每个子版本间基础语法没有区别，对于新手而言，推荐使用最新版本，体验最新的语法特性。而对于实际项目编写过程中，建议使用较老版本，从而避免版本间不兼容的问题，这将是Python环境搭建主要的难点之一。
- Python是一种解释型语言，运行Python程序就需要安装其相应的解释器。而根据编写Python解释器本身的语言可以将解释器分为CPython、IPython、PyPy等。我们最常用的是CPython，包括从[Python官网](#)下载得到的都是这个使用c语言开发的解释器。

## 2.Python模块、包、库和环境

---

- 初学Python的时候总会接触到这些名词，这几个名词间较容易造成混淆，在这里简单辨析一下：
- 首先将其分为两组，库和环境是逻辑层面的，是一种非常直观的用法。而模块和包则是“存储层面”的，在Python中有具体的定义和规范的。包含关系为：环境>库>包>模块。
  - 模块(module)：每一个.py结尾的文件就叫一个模块。
  - 包(package)：每一个包含\_\_init\_\_.py文件的文件夹叫做包。
  - 库(lib)：实现一类功能的代码集合，其中可能包含任意数量的模块和包。
- Python的库分为标准库和第三方库，其中标准库包含对基本功能的支持，详细列表可以看[这里](#)，而第三方库则为其他组织编写、打包、上传并维护的库。我们常说的Pytorch、numpy等就属于第三方库。
- 而环境则是一个更加宽泛的概念，通常我们会认为环境包含Python解释器、相关库以及其他运行环境，比如对于深度学习非常重要的显卡加速环境。

需要注意的是，人们会在习惯上将包和库两个词混用，如我个人更习惯称conda和pip为“包管理器”而非“库管理器”。所以需要根据语境来判断这两个词含义是否相同，大部分情况下使用两者并没有什么不同，而在某些特殊的情况下最好结合英文package和lib以及相关的定义来区分和理解。

## 3.包管理器

---

- 正如上文所说，我们的Python运行需要标准库和第三方库。其中标准库会随着Python一起被下载安装，但由于第三方库数量巨大，需要由我们自己选择所需要的进行安装。为了便于管理，Python官方使用pypi(Python Package Index)索引来统一记录所有的第三方库(package)，你可以在[这里](#)搜索到所有的package，当然也可以上传自己所编写的package供其他人使用。
- 在Python3.4中官方开始使用pip工具作为python的包管理器。此外，人们还大量使用conda工具管理包和环境。对于python的使用者而言，掌握这两种工具的用法就能够完成基本Python环境的搭建。
- 在安装和学习基本用法之前，我们需要明白pip和conda并不是两个对等的工具：
  - pip：每一个“Python”都会包括一个pip的可执行文件，这个pip用来管理所属“Python”的所有包的安装和卸载。一个pip对应一个“Python”。
  - conda：其中引入了虚拟环境的概念，每一个虚拟环境都对应这一个“Python”、一个pip和一系列包。而一个conda可以创建多个环境。也就是一个conda对应多个“Python”。
- conda引入虚拟环境的优势在于：

1. 我们能够将不同工作目的的环境分割开从而减轻系统开销和复杂的包依赖。比如我们用于网络爬虫项目和神经网络项目的包会有很大不同，就可以建立两个虚拟环境，分别安装两个Python解释器和相关库依赖。在运行不同项目的时候使用对应的Python环境。
2. 我们能保存多个版本的Python和包。这一点尤其在2020年前至关重要，因为当时Python2即将停止维护，人们处在2转向3的中间阶段。所以会使用conda同时建立python2和3两个环境，每个环境间互不干扰。

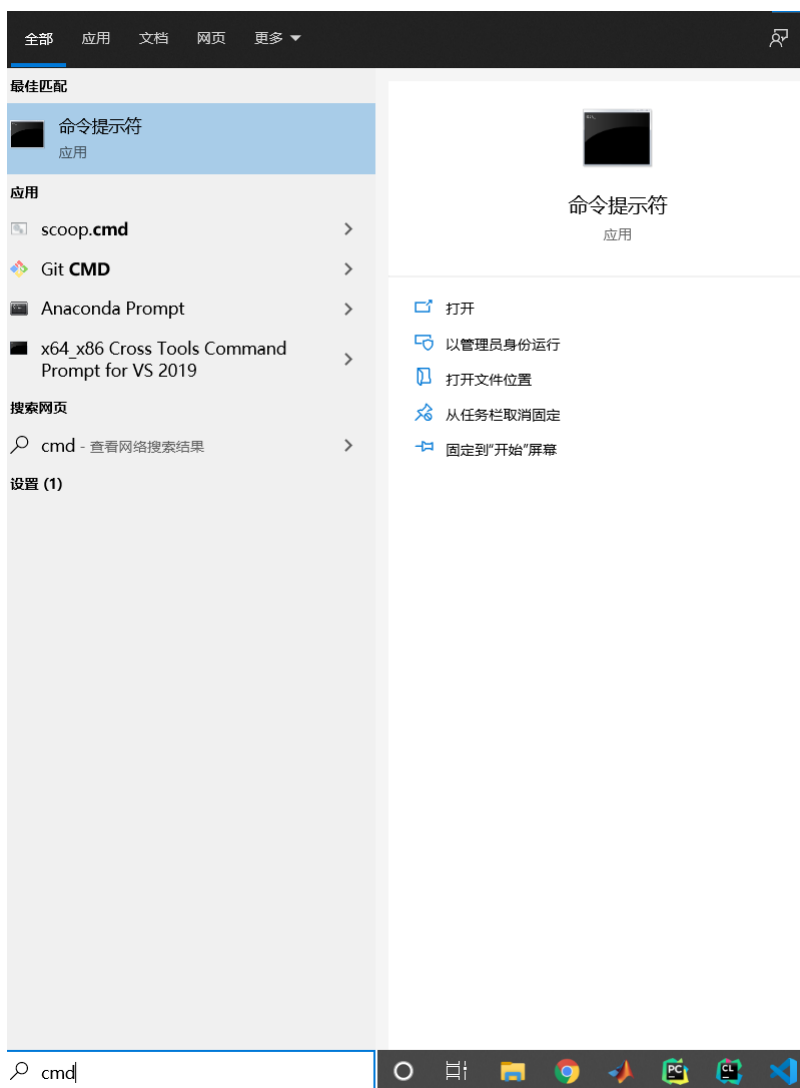
pip和conda两者都有各自的优势，并且我们知道其包含关系为：conda>python>pip。所以我个人非常建议各位在搭建环境的时候首先安装conda，然后使用conda来建立并管理python环境。

## 4.搭建环境的前置知识

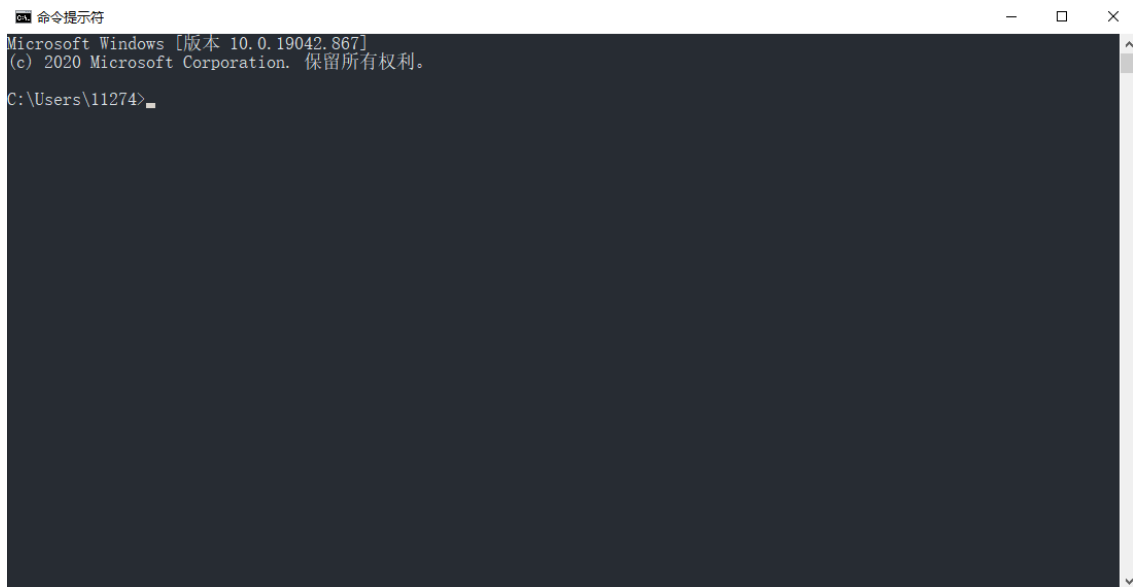
在配环境的过程中会涉及到一些其他工具和知识的使用，有些内容仅涉及部分操作系统，所以大家可以选择性的学习。

### 4.1.命令行

- 命令行作为直接交互的接口，大家之前应该多少都使用过。
  - Win10用户在开始选项旁边的搜索栏搜索cmd后选择打开即可。



- Mac用户在系统工具中打开“终端”工具。
- Ubuntu用户在导航页打开“Terminal”工具。
- 打开之后即可得到如下页面，图中为Win10的cmd：



- 需要注意的是，图中"C:\Users\11274"为当前用户的文件夹，“11274”为当前用户名，如果这个用户名被设置成了中文，将无法正常使用某些python包。我个人建议将其修改为英文或数字，具体的修改方法可以参考[这里](#)，需要使用管理员登陆系统并更改注册表。注意如果操作不当可能会影响个人文件甚至是系统信息，需要谨慎处理。可以先创建一个新的用户进行试验，如果改名失败也不会影响之前的老用户。

## 4.2.虚拟机/双系统

虽然Windows环境下python同样有很好的支持，但作为开发者不可避免地会需要使用Linux系统，所以我个人非常建议大家学习并使用Linux系统，这是一个复杂而且漫长的过程，但能够体会到Linux中的很多强大功能和优势。所以如果之前没有接触过Linux系统，我建议先安装虚拟机，之后逐步尝试安装双系统。而Mac系统和Linux有许多共通之处，不需要专门使用虚拟机。

### 4.2.1.系统选择

- Linux系统具有诸多发行版，这里做简要介绍
  - Ubuntu：使用dpkg包管理器，非常流行的Linux系统，尤其在20.04版本发行后界面有了进一步的提升。我个人就在使用Ubuntu20.04LST作为主力系统，所以非常推荐大家使用。
  - Debian：和Ubuntu同属dpkg派系，界面可能没有Ubuntu那么华丽，但同样稳定，适用个人和服务器。
  - CentOS：使用RPM包管理器，完全开源免费，通常用于替代不免费的RedHat作为服务器操作系统。

### 4.2.2.虚拟机选择

- Windows用户推荐VMware Workstation，该软件分为两个版本：Workstation player和Workstation pro，player可以免费应用与非商业场景，具备基本的虚拟机功能。而pro版本需要购买获得，具备更加完整的功能，在网络上存在一些非正规渠道，此处不做详述。大家可以根据需要自行选择。
- Mac用户推荐Parallels Desktop，该软件具备三个版本，官方渠道可以使用学生验证获得折扣。该软件同样存在非正规渠道。

### 4.2.3.双系统安装

- 在安装双系统之前，请确保你已经备份了当前系统的重要文件，并且已经学习过了磁盘分区、系统引导的基本概念，并且清楚个人电脑进入BIOS的方法，通常不同厂家的主板会有不同的进入方式。
- 然后需要制作启动盘。启动盘应当是一个大于4G或8G的U盘，用于存放系统及安装工具，注意启动盘制作会进行格式化所以注意转移个人文件。我个人不推荐任何的非官方制作工具和下载渠道。

Linux各个免费发行版直接到官网下载LST版本镜像文件，Win10同样使用官网的下载渠道和启动盘制作工具。

- 完成上述过程即可重启、插入启动盘、进入BIOS更改启动选项，然后就跟随该系统的安装程序进行基本信息设置、磁盘分区以及其他设置。并等待完成即可。

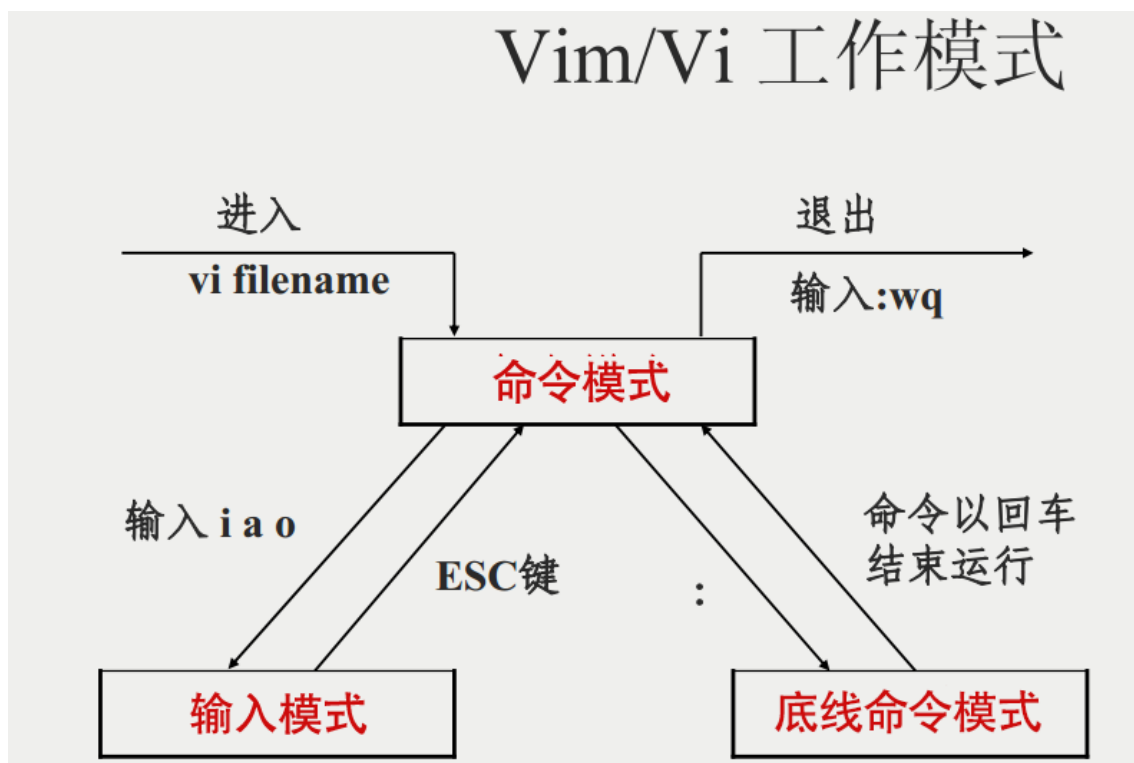
受篇幅的影响，并且网络上已经有大量的资料，所以关于虚拟机和双系统的安装我这里写得非常简略。其实这部分的工具非常成熟，安装系统的流程已经变得极为简化，相信大家结合博客和视频并多加尝试都能够顺利完成。

## 4.3.Vim编辑器

在Windows和Mac系统中都有大家非常熟悉的文本编辑器。而在Linux系统下，我们虽然仍可以使用gedit或vs code这些工具，但我个人在终端更喜欢使用Vim，所以有必要介绍Vim的基本用法。

### 4.3.1.Vim基本概念

- Vim是一个非常强大的文本编辑器，其主要特点之一就是具有完整的命令控制方式，熟练掌握后能够摆脱鼠标，仅通过键盘进行快速编辑和控制。
- 为了区分用户当前的输入是给编辑器的命令还是写入文本的字符，Vim编辑器划分了三个模式，分别是：
  - 命令模式：该模式键盘每个键都对应一个命令，如d是delete删除，h、j、k、l代表光标的左、上、下、右等待，/代表开始搜索，详细用法可以看[这里](#)。
  - 输入模式：正常的编辑模式，键盘每个键都代表本身的字符。
  - 底线命令模式：在命令模式中输入冒号后，光标会移动到底线，此时键盘输入为对编辑器的命令，该命令包括Vim内部的命令和系统的shell命令。如":wq"代表保存并推出，使用!后即可使用shell命令，如"! ls /home"代表查看/home目录
- 三种模式的转换方式：



### 4.3.2.使用Vim编辑文件的方法

这里介绍在终端使用Vim快速打开、编辑并保存一个文件的方法：

1. cd到目标文件所在目录（不知道cd的请百度），假设我们需要更改的文件为tmp.txt，就使用vim打开该文件：

```
vim tmp.txt
```

2. 之后我们就打开了该文件，当前处于命令模式，可以通过h、j、k、l等命令移动、翻页、搜索找到要更改的位置，按i键进入输入模式，开始正常的编辑工作。
3. 当编辑完毕之后，按键盘的Esc键回到命令模式。输入:进入底线命令模式，如果刚才的编辑成功则输入:wq保存并推出，如果想要放弃刚才的更改则使用:q!不保存退出，即可回到命令行。

Vim的熟练使用需要一定的记忆和大量的练习，上手难度较高，但是熟练后的编辑效率要远高于图形化编辑器。这里加以介绍主要是因为很多Linux相关教程会使用Vim，如果没有兴趣仅掌握上述的基本编辑方法即可。如果有兴趣建议结合上述链接和其他资料进一步学习，同时结合插件和其他工具构建强大的编辑体系。

## 4.4.环境变量

### 4.4.1.什么是环境变量

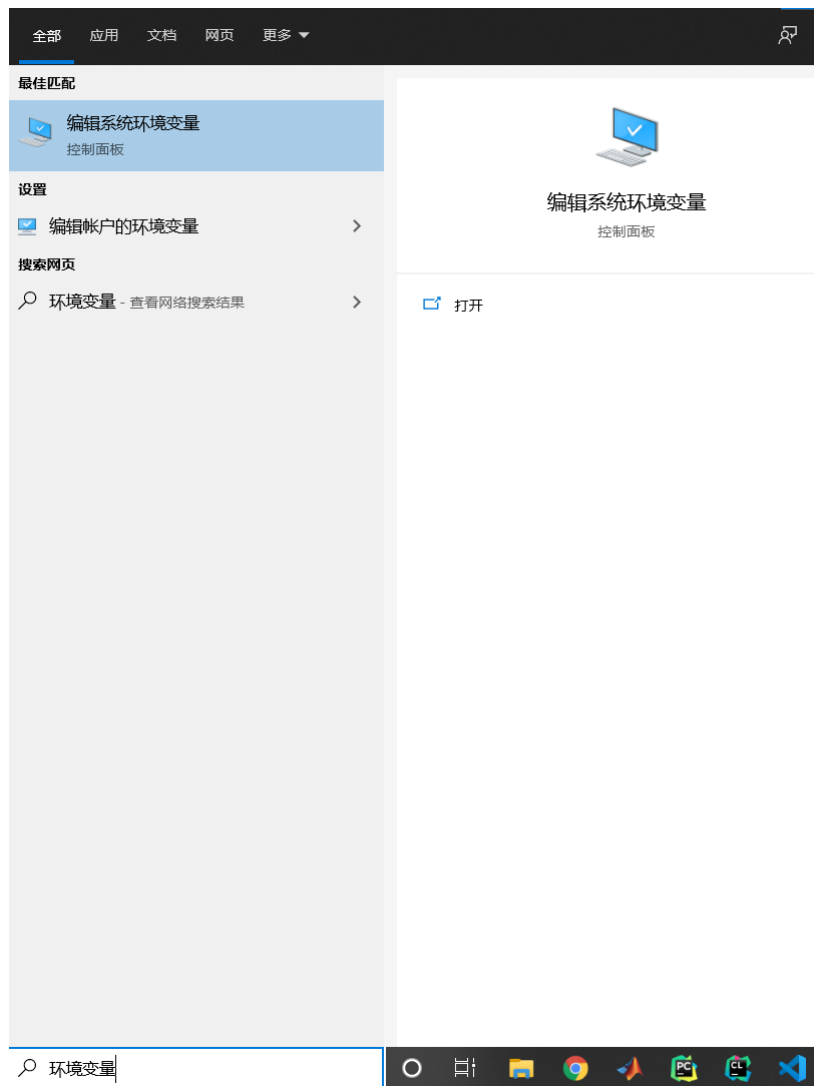
- 我给的不严谨的定义：是操作系统拥有的记录环境的变量。
- 这里的环境代表可执行文件、动态库等内容，也就是我们理解的那个环境的意思。

### 4.4.2.为什么要有环境变量

- 我们在终端输入一个命令仅仅是该命令的名称，终端需要根据这个名称去找到该命令的可执行文件。而计算机路径太过复杂中端当然不可能遍历每个目录，所以需要指定一个查找范围，也就是这个环境变量。
- 如果我们想要让某个命令能够被终端识别，就需要把这个命令所在的位置告诉终端。否则就会出现找不到该命令的问题。

### 4.4.3.Windows添加环境变量

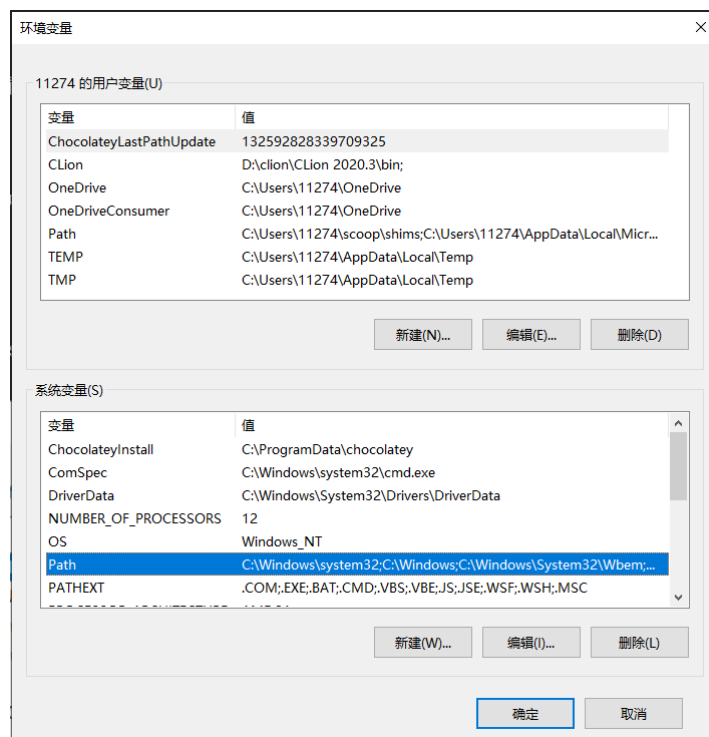
- 搜索环境变量：



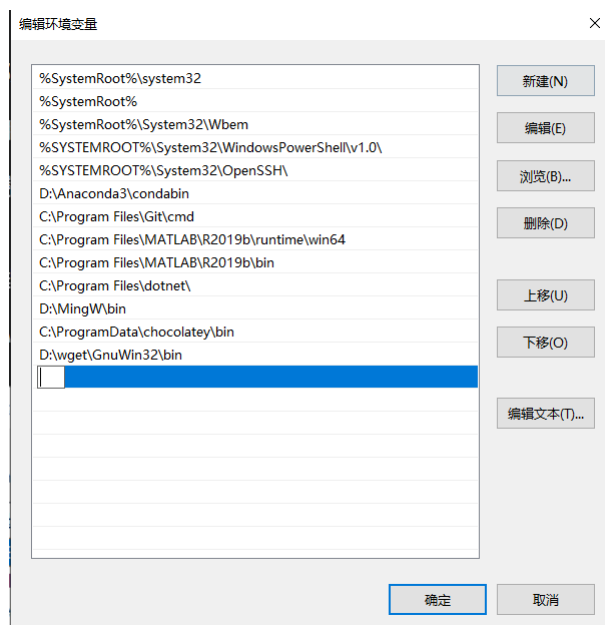
- 进入后点击环境变量



- 我们直接将想要添加的环境变量加入到"系统变量"中的"Path"中：



- 打开Path后，选择添加，输入要添加的环境变量，确定退出即可：



#### 4.4.4. Mac/Linux添加环境变量

- Linux系统中有不止一个配置文件，想要了解可以看[这里](#)。我个人的习惯是添加到/etc/profile或 ~/.bashrc中，如果使用zsh代替bash则在 ~/.zshrc中。
- 最简单的添加方法就是使用vim打开上述的配置文件（选择一个就好），在文件末尾添加：

```
export PATH=要添加的目录:$PATH
```

然后保存推出，之后source重新加载该配置文件，比如假设添加在了 ~/.zshrc中，就：

```
source ~/.zshrc
```

需要声明，上述是我为了理解方便给出的定义和解释，不保证严谨，也不覆盖其所有功能。

## 5.conda的安装



## 5.1.conda基本介绍

- conda：是我们之前提到的包和环境管理器本身。
- anaconda：是一个打包的集合，包含conda本身和常用的众多包的集合，安装包约500MB。
- miniconda：是conda的最小安装环境，仅包含conda、python和一些必备工具，安装包约50MB。

## 5.2.下载位置

- 首先可以在官网下载，[miniconda位置](#)，[anaconda位置](#)。
- 然而国内往往下载速度较慢，如果在官网下载时发现下行速度太慢，可以考虑到[清华镜像站](#)下载。

注意需要选择anaconda3而不是anaconda2或者其他版本，子版本尽量选择最新版本。注意操作系统和架构的版本区别：x86代表32位机，x86\_64或简称的x64代表64位机。

## 5.3.安装

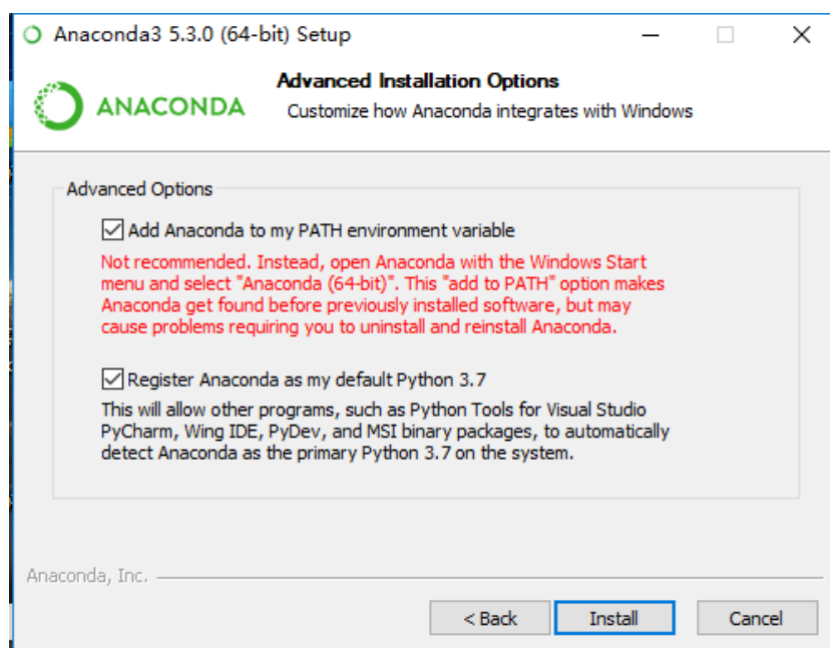
- 我们可以注意到清华源中每个子版本通常都会发布六个安装包，分别为：
  - Windows-x86.exe
  - Windows-x86\_64.exe
  - MacOSX-x86\_64.sh
  - MacOSX-x86\_64.pkg
  - Linux-x86.sh
  - Linux-x86\_64.sh

部分子版本会有Linux-ppc64le.sh的版本。

- 其中，Windows全部为可执行文件安装，Linux全部为脚本安装，Mac不包含32位机，支持脚本安装和pkg'安装包安装。下面主要介绍.exe和.sh的安装方式。

### 5.3.1.exe安装

- .exe安装较为简单，过程中遇到如下界面记得勾选第一项，就可以自动添加环境变量。第二项推荐勾选。



- 安装完成后，打开cmd，输入

```
conda --version
```

如果出现版本信息则证明安装成功。

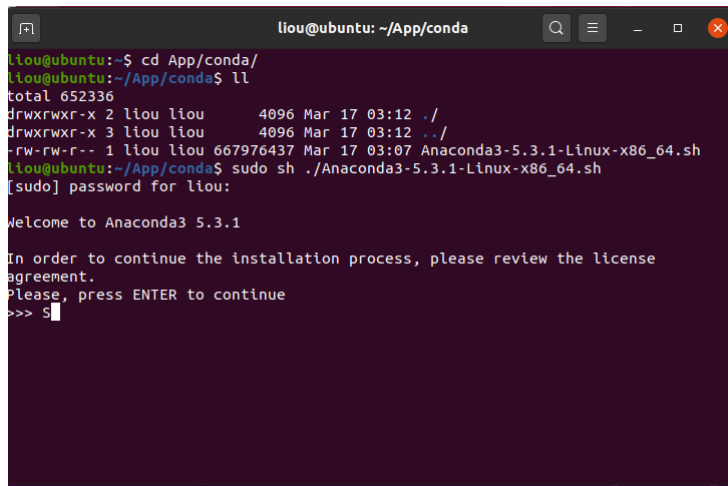
- 如果需要手动添加环境变量，请将安装路径下的condabin目录作为环境变量。如我这里安装在了D:\Anaconda3，则将D:\Anaconda3\condabin添加到环境变量中即可。

### 5.3.2.sh安装

- cd到.sh所在目录，输入命令：

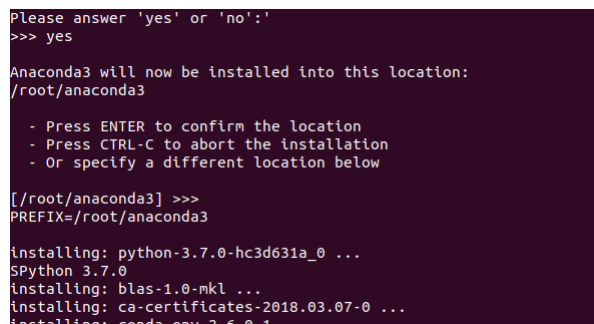
```
sudo sh ./文件名.sh
```

- 回车后输入密码：



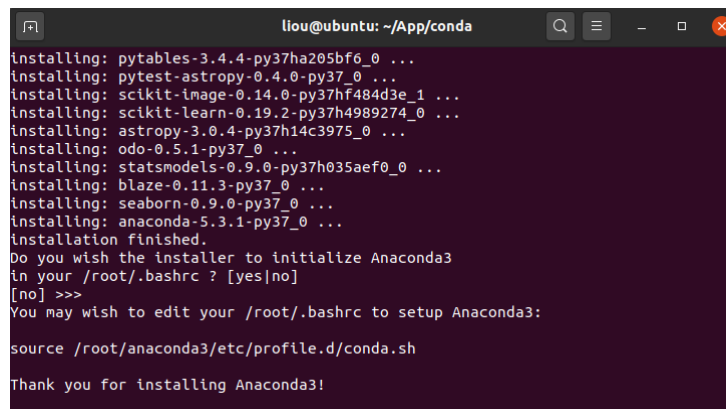
```
liou@ubuntu: ~/App/conda
liou@ubuntu:~$ cd App/conda/
liou@ubuntu:~/App/conda$ ll
total 652336
drwxrwxr-x 2 liou liou 4096 Mar 17 03:12 ./
drwxrwxr-x 3 liou liou 4096 Mar 17 03:12 ../
-rw-rw-r-- 1 liou liou 667976437 Mar 17 03:07 Anaconda3-5.3.1-Linux-x86_64.sh
liou@ubuntu:~/App/conda$ sudo sh ./Anaconda3-5.3.1-Linux-x86_64.sh
[sudo] password for liou:
Welcome to Anaconda3 5.3.1
In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>> S
```

- 回车后出现用户协议，浏览结束后输入yes同意。然后会提示选择安装目录，注意这里默认安装位置在root目录下，该目录进行操作时需要开权限，所以建议更改到个人目录或者其他目录下。之后等待安装：



```
Please answer 'yes' or 'no':
>>> yes
Anaconda3 will now be installed into this location:
/root/anaconda3
- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below
[/root/anaconda3] >>>
PREFIX=/root/anaconda3
installing: python-3.7.0-hc3d631a_0 ...
SPython 3.7.0
installing: blas-1.0-mkl ...
installing: ca-certificates-2018.03.07-0 ...
installing: conda-env-2.6.0-1
```

- 安装结束后，会提示时候需要初始化conda，此处建议选择yes。如果选no则需要根据提示和前文方法手动配置环境变量。



```
installing: pytables-3.4.4-py37ha205bf6_0 ...
installing: pytest-astropy-0.4.0-py37_0 ...
installing: scikit-image-0.14.0-py37hf484d3e_1 ...
installing: scikit-learn-0.19.2-py37h4989274_0 ...
installing: astropy-3.0.4-py37h14c3975_0 ...
installing: odo-0.5.1-py37_0 ...
installing: statsmodels-0.9.0-py37h035aef0_0 ...
installing: blaze-0.11.3-py37_0 ...
installing: seaborn-0.9.0-py37_0 ...
installing: anaconda-5.3.1-py37_0 ...
Installation finished.
Do you wish the installer to initialize Anaconda3
in your /root/.bashrc ? [yes/no]
[no] >>>
You may wish to edit your /root/.bashrc to setup Anaconda3:
source /root/anaconda3/etc/profile.d/conda.sh
Thank you for installing Anaconda3!
```

- 出现安装成功提示后，会推荐安装vs coda，此处根据个人需求选择。

```
liou@ubuntu: ~/App/conda
Installation finished.
Do you wish the installer to initialize Anaconda3
in your /root/.bashrc ? [yes|no]
[no] >>>
You may wish to edit your /root/.bashrc to setup Anaconda3:

source /root/anaconda3/etc/profile.d/conda.sh

Thank you for installing Anaconda3!

=====

Anaconda is partnered with Microsoft! Microsoft VSCode is a streamlined
code editor with support for development operations like debugging, task
running and version control.

To install Visual Studio Code, you will need:
- Administrator Privileges
- Internet connectivity

Visual Studio Code License: https://code.visualstudio.com/license

Do you wish to proceed with the installation of Microsoft VSCode? [yes|no]
>>>
```

- 安装结束后，输入conda --version得到版本即代表安装成功。

## 6.conda的基本用法

### 6.1.conda换源

- 正如之前提到的，国内有些安装线路速度很慢会影响正常使用，所以当安装包的过程中发现下行速度太慢的话就需要更换国内镜像站。
- 国内有多个镜像站提供服务，常用的比如清华、阿里、中科院等。换源代码往往仅需要一行或几行包含地址的命令，可以通过百度获取。例如更换清华源的[官方推荐配置地址](#)，仅需要在终端依次输入如下命令即可添加清华源：

```
conda config --add channels
https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
conda config --add channels
https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
conda config --add channels
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/pytorch/
conda config --set show_channel_urls yes
```

添加后我们所以安装一个包进行测试：

```
C:\Users\11274>conda install pandas
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: D:\Anaconda3

added / updated specs:
- pandas

The following packages will be downloaded:

package                                     build                                size  source
-----
ca-certificates-2020.10.14                 0                                  122 KB https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
certifi-2020.6.20                           pyhd3eb1b0_3                      155 KB https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
conda-4.9.2                                 py38haa95532_0                    2.9 MB https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
openssl-1.1.1h                             he774522_0                        4.8 MB https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
pandas-1.1.3                               py38ha925a31_0                    7.5 MB https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main

Total: 15.5 MB

The following packages will be SUPERSEDED by a higher-priority channel:

ca-certificates pkgs/main --> anaconda/pkgs/main
certifi         pkgs/main --> anaconda/pkgs/main
conda           pkgs/main --> anaconda/pkgs/main
openssl         pkgs/main --> anaconda/pkgs/main
pandas          pkgs/main --> anaconda/pkgs/main

Proceed ([y]/n)?
```

可以看到安装包的源头都变成了清华源。

## 6.2.conda环境管理

- 查看虚拟环境：使用命令：

```
conda env list
```

即可看到当前创建的所有虚拟环境，其中base为默认环境，在未创建或未切换时会使用这个环境。星号标注了当前启用的环境：

```
C:\Users\11274>conda env list
# conda environments:
#
base            * D:\Anaconda3
NLP             D:\Anaconda3\envs\NLP
Torch           D:\Anaconda3\envs\Torch
```

- 创建虚拟环境：使用命令：

```
conda create --name 环境名称 (python=3 numpy...)
```

其中，--name指定了新环境的名称，后面括号内的内容为可选项，即可以在创建的同时指定python环境、预装包等操作。如果不指定则会创建一个空的环境。如我们建立一个名为Test的新环境：

```
(Torch) C:\Users\11274>conda create --name Test
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: D:\Anaconda3\envs\Test

Proceed ([y]/n)? y
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#   $ conda activate Test
#
# To deactivate an active environment, use
#
#   $ conda deactivate

(Torch) C:\Users\11274>conda list
# packages in environment at D:\Anaconda3\envs\Torch:
#
# Name                      Version           Build  Channel
(Torch) C:\Users\11274>conda env list
# conda environments:
#
base            D:\Anaconda3
NLP             D:\Anaconda3\envs\NLP
Test           D:\Anaconda3\envs\Test
Torch          * D:\Anaconda3\envs\Torch
```

- 切换虚拟环境：成功创建环境后会出现切换命令的提示，Windows和Linux会有不同。

- Windows:

```
conda activate Test    --激活环境
conda deactivate       --注销环境
```

- Linux:

```
source activate Test   --激活环境
source deactivate      --注销环境
```

切换后命令提示前会出现(Test)标识，再次查看conda env list可以发现星号的变化：

```
(Torch) C:\Users\11274>conda activate Test
(Test) C:\Users\11274>conda env list
# conda environments:
#
base                D:\Anaconda3
NLP                 D:\Anaconda3\envs\NLP
Test                * D:\Anaconda3\envs\Test
Torch              D:\Anaconda3\envs\Torch
```

- 删除虚拟环境：注销环境后，使用remove命令：

```
C:\Users\11274>conda env remove --name Test
Remove all packages in environment D:\Anaconda3\envs\Test:

C:\Users\11274>conda env list
# conda environments:
#
base                * D:\Anaconda3
NLP                 D:\Anaconda3\envs\NLP
Torch              D:\Anaconda3\envs\Torch
```

## 6.3.conda包管理

- 搜索包：使用命令conda search搜索指定包，查看该包的基本信息：

```
C:\Users\11274>conda search numpy
Loading channels: done
```

#	Name	Version	Build	Channel
	numpy	1.6.2	py26_0	anaconda/pkgs/free
	numpy	1.6.2	py26_4	anaconda/pkgs/free
	numpy	1.6.2	py27_0	anaconda/pkgs/free
	numpy	1.6.2	py27_4	anaconda/pkgs/free
	numpy	1.7.0	py26_0	anaconda/pkgs/free
	numpy	1.7.0	py27_0	anaconda/pkgs/free
	numpy	1.7.0	py33_0	anaconda/pkgs/free
	numpy	1.7.1	py26_0	anaconda/pkgs/free
	numpy	1.7.1	py26_1	anaconda/pkgs/free
	numpy	1.7.1	py26_2	anaconda/pkgs/free
	numpy	1.7.1	py26_3	anaconda/pkgs/free
	numpy	1.7.1	py27_0	anaconda/pkgs/free
	numpy	1.7.1	py27_1	anaconda/pkgs/free
	numpy	1.7.1	py27_2	anaconda/pkgs/free
	numpy	1.7.1	py27_3	anaconda/pkgs/free
	numpy	1.7.1	py33_0	anaconda/pkgs/free
	numpy	1.7.1	py33_1	anaconda/pkgs/free
	numpy	1.7.1	py33_2	anaconda/pkgs/free
	numpy	1.7.1	py33_3	anaconda/pkgs/free
	numpy	1.8.0	py26_0	anaconda/pkgs/free
	numpy	1.8.0	py27_0	anaconda/pkgs/free
	numpy	1.8.0	py33_0	anaconda/pkgs/free
	numpy	1.8.1	py26_0	anaconda/pkgs/free
	numpy	1.8.1	py27_0	anaconda/pkgs/free
	numpy	1.8.1	py33_0	anaconda/pkgs/free
	numpy	1.8.1	py34_0	anaconda/pkgs/free
	numpy	1.8.2	py26_0	anaconda/pkgs/free
	numpy	1.8.2	py27_0	anaconda/pkgs/free
	numpy	1.8.2	py33_0	anaconda/pkgs/free
	numpy	1.8.2	py34_0	anaconda/pkgs/free
	numpy	1.9.0	py26_0	anaconda/pkgs/free
	numpy	1.9.0	py27_0	anaconda/pkgs/free
	numpy	1.9.0	py33_0	anaconda/pkgs/free
	numpy	1.9.0	py34_0	anaconda/pkgs/free
	numpy	1.9.1	py26_0	anaconda/pkgs/free
	numpy	1.9.1	py27_0	anaconda/pkgs/free
	numpy	1.9.1	py33_0	anaconda/pkgs/free
	numpy	1.9.1	py34_0	anaconda/pkgs/free
	numpy	1.9.2	py26_0	anaconda/pkgs/free
	numpy	1.9.2	py27_0	anaconda/pkgs/free
	numpy	1.9.2	py27_2	anaconda/pkgs/free
	numpy	1.9.2	py33_0	anaconda/pkgs/free
	numpy	1.9.2	py34_0	anaconda/pkgs/free
	numpy	1.9.2	py34_2	anaconda/pkgs/free
	numpy	1.9.2	py35_0	anaconda/pkgs/free
	numpy	1.9.3	py27_0	anaconda/pkgs/free
	numpy	1.9.3	py27_1	anaconda/pkgs/free
	numpy	1.9.3	py27_3	anaconda/pkgs/free
	numpy	1.9.3	py27he0c0ee4_6	anaconda/pkgs/main
	numpy	1.9.3	py27he0c0ee4_6	pkgs/main

- 安装包：使用命令：

```
conda install 包名(=版本号)
```

其中的版本号为可选内容。注意conda安装包前会使用算法搜索版本依赖，如果不指定版本就不能控制conda安装的版本，同时会加大搜索空间的范围，这一点后面会提到。此外，conda会根据包的版本依赖增加甚至更改相关包，所以每次的解析结果中会出现多个包的安装以及版本变更。

```
C:\Users\11274>conda install mumpy=1.19.2
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: D:\Anaconda3

added / updated specs:
- mumpy=1.19.2

The following packages will be downloaded:



| package                    | build          | size   | url                                                     |
|----------------------------|----------------|--------|---------------------------------------------------------|
| ca-certificates-2020.10.14 | 0              | 122 KB | https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main |
| certifi-2020.6.20          | pyhd3eb1b0_3   | 155 KB | https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main |
| conda-4.9.2                | py38haa95532_0 | 2.9 MB | https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main |
| mumpy-1.19.2               | py38haa95532_0 | 22 KB  | https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main |
| openssl-1.1.1h             | he774522_0     | 4.8 MB | https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main |



Total: 8.0 MB

The following packages will be SUPERSEDED by a higher-priority channel:



| package         | pkgs/main --> | anaconda/pkgs/main |
|-----------------|---------------|--------------------|
| ca-certificates | pkgs/main --> | anaconda/pkgs/main |
| certifi         | pkgs/main --> | anaconda/pkgs/main |
| conda           | pkgs/main --> | anaconda/pkgs/main |
| mumpy           | pkgs/main --> | anaconda/pkgs/main |
| openssl         | pkgs/main --> | anaconda/pkgs/main |



Proceed ([y]/n)? y

Downloading and Extracting Packages
openssl-1.1.1h | 4.8 MB | 100%
mumpy-1.19.2 | 22 KB | 0%
```

- 查看包：使用命令conda list可以查看当前环境的所有包：

```
C:\Users\11274>conda list
# packages in environment at D:\Anaconda3:
#
# Name                                Version                                Build                                Channel
ipyw_jlab_nb_ext_conf                0.1.0                                py38_0                             defaults
alabaster                             0.7.12                               py_0                               defaults
anaconda                              2020.11                              py38_0                             defaults
anaconda-client                       1.7.2                                py38_0                             defaults
anaconda-navigator                    1.10.0                               py38_0                             defaults
anaconda-project                      0.8.4                                py_0                               defaults
argh                                   0.26.2                               py38_0                             defaults
argon2-cffi                           20.1.0                               py38he774522_1                     defaults
asn1crypto                            1.4.0                                py_0                               defaults
astroid                                2.4.2                                py38_0                             defaults
astropy                               4.0.2                                py38he774522_0                     defaults
async_generator                       1.10                                 py_0                               defaults
atomicwrites                           1.4.0                                py_0                               defaults
attrs                                  20.3.0                               pyhd3eb1b0_0                       defaults
autopep8                              1.5.4                                py_0                               defaults
babel                                  2.8.1                                pyhd3eb1b0_0                       defaults
backcall                               0.2.0                                py_0                               defaults
backports                              1.0                                   py_2                               defaults
backports.shutil_get_terminal_size    1.0.0                                py38_2                             defaults
bcrypt                                 3.2.0                                py38he774522_0                     defaults
beautifulsoup4                        4.9.3                                pyhb0f4dca_0                       defaults
bitarray                              1.6.1                                py38h2bbff1b_0                     defaults
bkcharts                              0.2                                   py38_0                             defaults
blas                                   1.0                                   mkl                                 defaults
bleach                                 3.2.1                                py_0                               defaults
blosc                                  1.20.1                               h7bd577a_0                         defaults
bokeh                                  2.2.3                                py38_0                             defaults
boto                                    2.49.0                               py38_0                             defaults
bottleneck                            1.3.2                                py38h2a96729_1                     defaults
brotlipy                              0.7.0                                py38he774522_1000                  defaults
bzip2                                  1.0.8                                he774522_0                         defaults
ca-certificates                       2020.10.14                           0                                   https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
certifi                               2020.6.20                             pyhd3eb1b0_3                       https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
cffi                                   1.14.3                               py38h7a1dbc1_0                     defaults
chardet                               3.0.4                                py38_1003                          defaults
click                                  7.1.2                                py_0                               defaults
cloudpickle                            1.6.0                                py_0                               defaults
clyent                                 1.2.2                                py38_1                             defaults
colorama                              0.4.4                                py_0                               defaults
comtypes                              1.1.7                                py38_1001                          defaults
conda                                  4.9.2                                py38haa95532_0                     https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
conda-build                            3.21.3                               py38haa95532_0                     defaults
conda-package-handling                 1.7.2                                py38h76e460a_0                     defaults
console_shortcut                       0.1.1                                4                                   defaults
contextlib2                            0.6.0.post1                           py_0                               defaults
cryptography                           3.1.1                                py38h7a1dbc1_0                     defaults
cudatoolkit                           8.0                                   4                                   https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
curl                                    7.71.1                               h2a8f88b_1                         defaults
cyclor                                 0.10.0                               py38_0                             defaults
cython                                 0.29.21                              py38ha925a31_0                     defaults
cytoolz                                0.11.0                               py38he774522_0                     defaults
dask                                    2.30.0                               py_0                               defaults
```

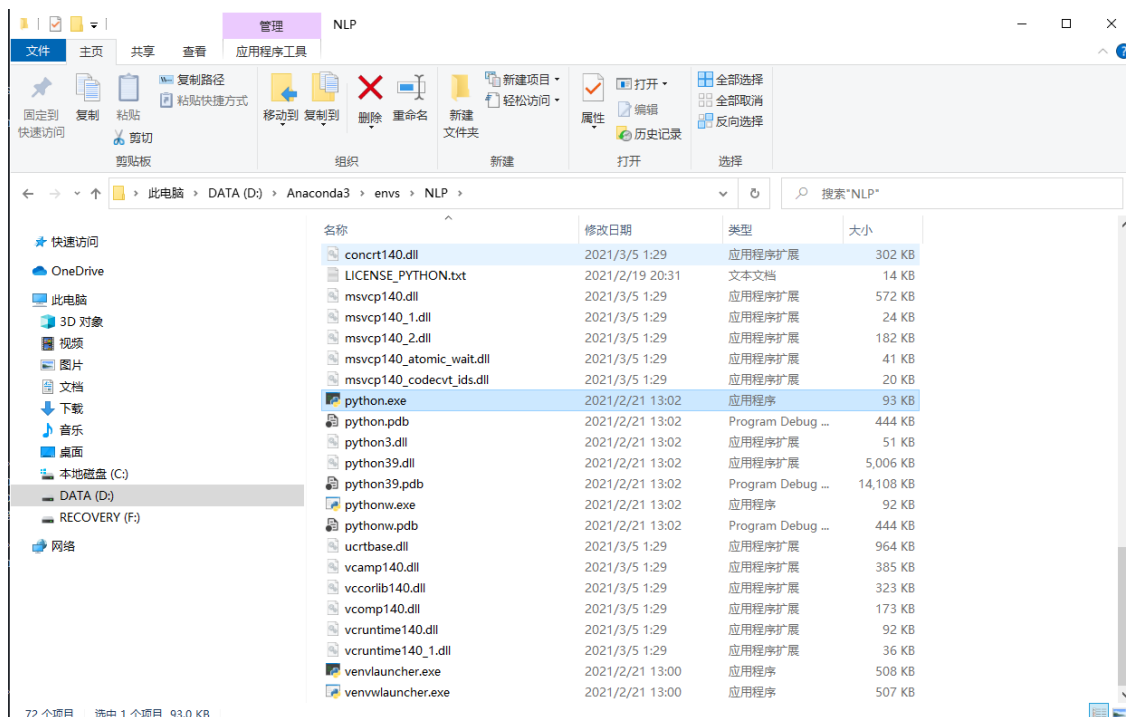
## 6.4.找到你的虚拟环境的解释器

- 我们在使用一些开发环境软件时，运行前需要指定解释器的位置，也就是我们每个虚拟环境中的解释器。
- 我们需要先找到环境位置，Windows一般位于anaconda安装目录下的envs目录中，而Linux通常在.conda目录的envs目录中。我们可以通过conda env list查看该虚拟环境的位置，如图：



```
C:\Users\11274>conda env list
# conda environments:
#
base                                 * D:\Anaconda3
NLP                                D:\Anaconda3\envs\NLP
Test                               D:\Anaconda3\envs\Test
Torch                              D:\Anaconda3\envs\Torch
```

在Windows下，解释器就在该目录下，如图找到这个python.exe即可：



而在Linux中，解释器在该目录下的bin目录中，也是一个python加版本号的可执行文件。

## 6.5.效率优化

- 正如上文提及的conda安装前会先搜索版本依赖关系，这是一个非常耗时的步骤，随着包数量增加和目标包版本数的增加，solving environment的时间会越来越长。此时有几个解决方法：
  - 使用更新版本的conda。
  - 时常使用conda clean清理无用包。
  - 时常使用conda update和upgrade更新。
  - 下载时指定包的版本。
  - 按照工作类型分割虚拟环境。

## 6.6.conda和pip混用

- pip基本命令和conda非常相似，不做叙述。
- 有些情况下，会出现包的名字不同意或者有的包只能用其中一个工具安装。此时需要灵活选用不同工具。
- pip更少关心依赖问题，所以安装速度要更快于conda。
- pip和conda安装的路径略有不同，所以会出现conda装的包pip删不掉反之同理的情况。这时候就需要用安装的工具进行删除，或者直接用conda删除整个环境。
- pip同样需要换源使用，换源方法请百度。



## 6.7.导出导入环境

- conda和pip都支持环境导入导出功能。

- conda:

```
conda env export > 环境名.yaml    --导出
conda env create -f 环境名.yaml    --导入
```

- pip:

```
pip freeze > 环境名.txt           --导出
pip install -r 环境名.txt          --导入
```

## 7.显卡支持

虽然在今年pytorch1.8宣布正式支持AMD ROCm，但是现在深度学习项目仍主要使用Nvidia显卡进行计算，所以这里主要介绍Nvidia的Driver-CUDA-Cudnn环境配置方法。该环境主要包含三个部分：显卡驱动、CUDA和cuDNN。

### 7.1.Windows

#### 7.1.1.驱动

- Windows环境下配置显卡支持相对容易，首先来说，Windows系统通常不需要手动安装Nvidia显卡驱动，在桌面右键中点击Nvidia控制面板即可查看到详细的版本信息。

#### 7.1.2.CUDA

- 如果显卡驱动正常工作，则开始安装CUDA，来到[官网](#)，选择对应的与驱动想配套的CUDA版本进行下载，注意选择离线版本：

### 选择目标平台

单击指示您目标平台的绿色按钮。系统仅会显示出受支持的平台。下载并使用软件即表示您同意完全遵守 [CUDA EULA](#) 的条款和条件。

**操作系统**

Linux Windows

**架构**

x86\_64

**版本**

10 Server 2019 Server 2016

**安装程序类型**

exe (local) exe (network)

### Download Installer for Windows 10 x86\_64

The base installer is available for download below.

基本安装程序

Download [3.1 GB] 

安装说明：

1. Double click cuda\_11.1.0\_456.43\_win10.exe
2. Follow on-screen prompts

- 下载成功后进入下载器，注意选择自定义安装：



其中的Nsight是CUDA的开发工具，我们通常不会使用。有些情况下安装时Visual Studio Intergration会报错，可以取消这个选项。



除此之外的其他组件大家按需安装：



之后确定路径之后开始安装即可。安装完成后，进入cmd输入nvcc --version，返回版本即可：

```
C:\Users\11274>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Tue_Sep_15_19:12:04_Pacific_Daylight_Time_2020
Cuda compilation tools, release 11.1, V11.1.74
Build cuda_11.1.relgpu_drvr455TC455_06.29069683_0
C:\Users\11274>
```

### 7.1.3.cuDNN

- 首先来到[官网](#)，注意cnDNN的下载需要登陆Nvidia账号，如果还没有账号需要先注册。认证成功后即可选择下载版本：

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

☒ I Agree To the Terms of the [cuDNN Software License Agreement](#)

Note: Please refer to the [Installation Guide](#) for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.

For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the [Deep Learning SDK Documentation](#) web page.

[Download cuDNN v8.1.1 \(Feburary 26th, 2021\), for CUDA 11.0, 11.1 and 11.2](#)

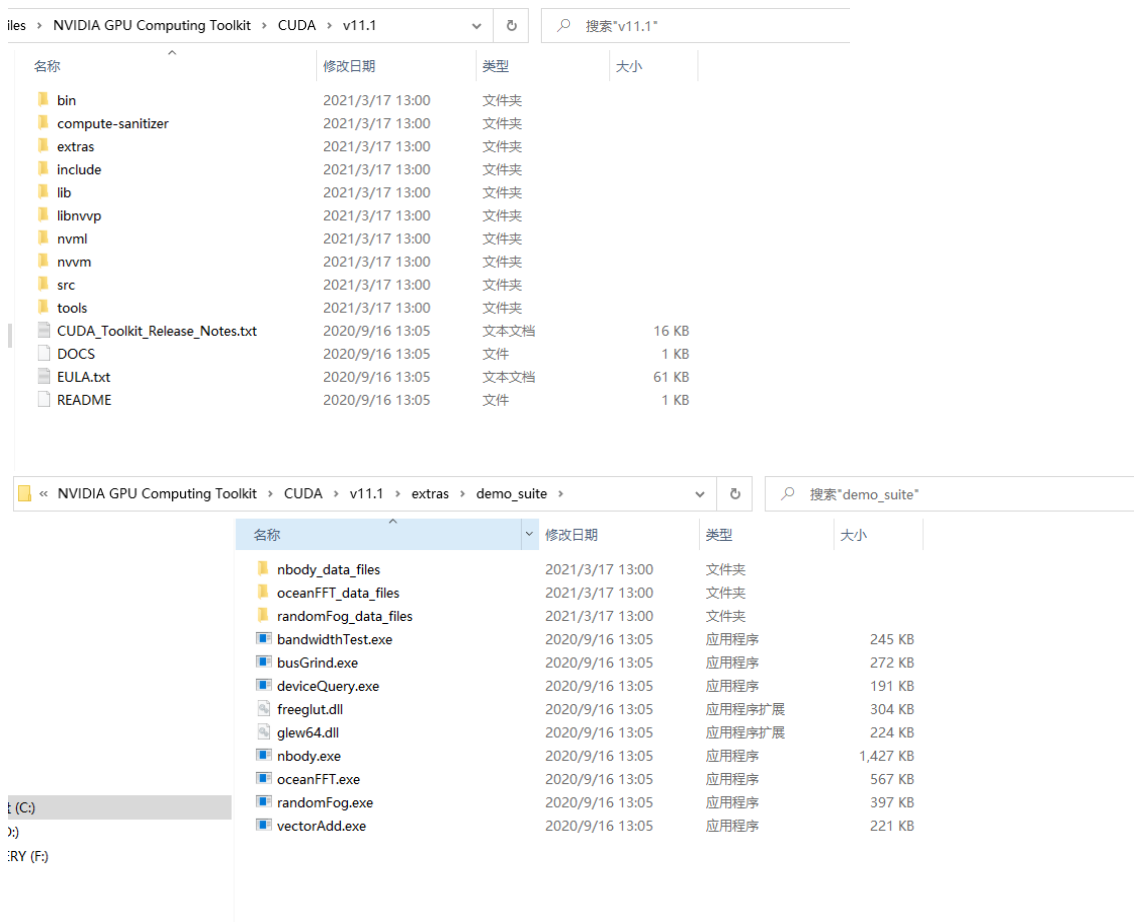
#### Library for Windows and Linux, Ubuntu(x86\_64, armsbsa, PPC architecture)

[cuDNN Library for Linux \(aarch64sbsa\)](#)  
[cuDNN Library for Linux \(x86\\_64\)](#)  
[cuDNN Library for Linux \(PPC\)](#)  
[cuDNN Library for Windows \(x86\)](#)  
[cuDNN Runtime Library for Ubuntu20.04 x86\\_64 \(Deb\)](#)  
[cuDNN Developer Library for Ubuntu20.04 x86\\_64 \(Deb\)](#)  
[cuDNN Code Samples and User Guide for Ubuntu20.04 x86\\_64 \(Deb\)](#)  
[cuDNN Runtime Library for Ubuntu20.04 aarch64sbsa \(Deb\)](#)  
[cuDNN Developer Library for Ubuntu20.04 aarch64sbsa \(Deb\)](#)  
[cuDNN Code Samples and User Guide for Ubuntu20.04 aarch64sbsa \(Deb\)](#)  
[cuDNN Cross-compile Library for Ubuntu20.04 aarch64sbsa \(Deb\)](#)  
[cuDNN Developer Cross-compile Library for Ubuntu20.04 aarch64sbsa \(Deb\)](#)  
[cuDNN Runtime Library for Ubuntu18.04 x86\\_64 \(Deb\)](#)  
[cuDNN Developer Library for Ubuntu18.04 x86\\_64 \(Deb\)](#)

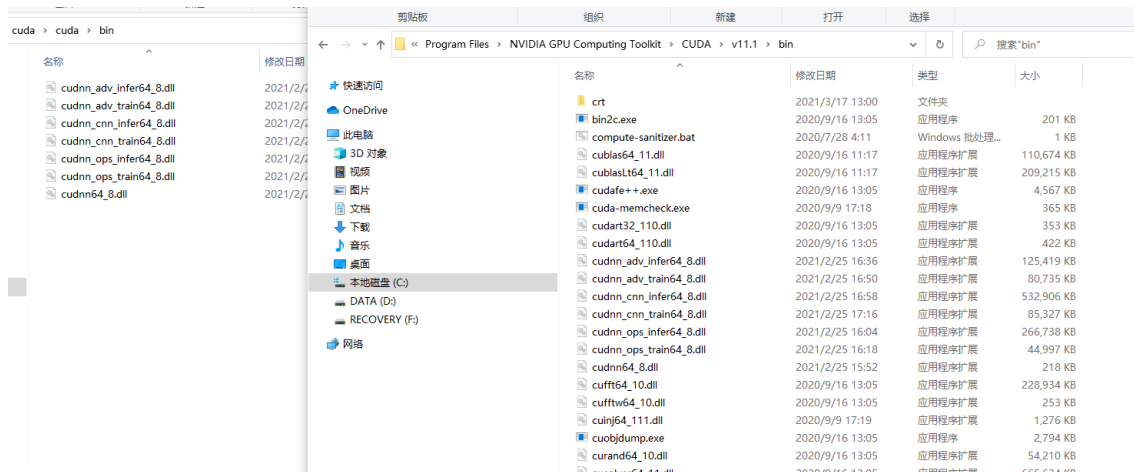
- 下载成功后解压压缩包，得到如下内容：

.cuda > cuda >				
搜索"cuda"				
名称	修改日期	类型	大小	
bin	2021/3/17 13:21	文件夹		
include	2021/3/17 13:21	文件夹		
lib	2021/3/17 13:21	文件夹		
NVIDIA_SL_A_cuDNN_Support.txt	2021/2/22 17:15	文本文档	21 KB	

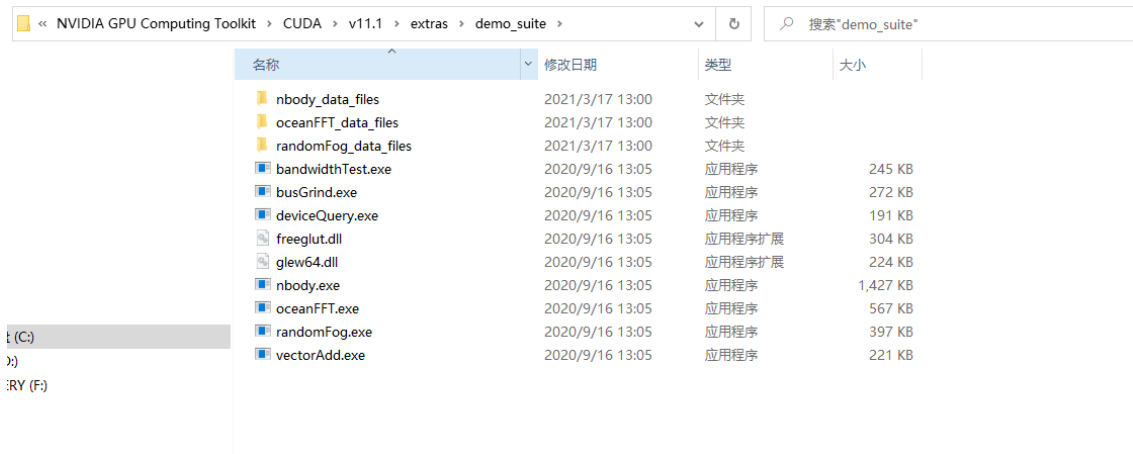
- cuDNN相当于对CUDA的一个神经网络补丁，所以我们只需要将补丁内的东西移动到CUDA目录下。我们找到CUDA的安装路径，默认应该为C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1：



- 所以我们就把cuDNN中三个文件夹的内容对应的移动到CUDA路径下，注意一定要对应，如bin移动到bin，lib移动到lib。



- 之后我们验证是否安装成功，来到CUDA安装目录的extras\demo\_suite文件夹中，



- 我们拷贝这个路径，进入cmd，cd到此处，首先运行deviceQuery.exe：

```
C:\Users\11274>cd C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1\extras\demo_suite
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1\extras\demo_suite>deviceQuery.exe
deviceQuery.exe Starting...

  CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 1060"
  CUDA Driver Version / Runtime Version      11.1 / 11.1
  CUDA Capability Major/Minor version number: 6.1
  Total amount of global memory:              6144 MBytes (6442450944 bytes)
  (10) Multiprocessors, (128) CUDA Cores/MP: 1280 CUDA Cores
  GPU Max Clock rate:                        1733 MHz (1.73 GHz)
  Memory Clock rate:                         4004 Mhz
  Memory Bus Width:                          192-bit
  L2 Cache Size:                             1572864 bytes
  Maximum Texture Dimension Size (x,y,z)      1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers
  Total amount of constant memory:             zu bytes
  Total amount of shared memory per block:     zu bytes
  Total number of registers available per block: 65536
  Warp size:                                  32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:         1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                       zu bytes
  Texture alignment:                          zu bytes
  Concurrent copy and kernel execution:       Yes with 5 copy engine(s)
  Run time limit on kernels:                   Yes
  Integrated GPU sharing Host Memory:          No
  Support host page-locked memory mapping:    Yes
  Alignment requirement for Surfaces:         Yes
  Device has ECC support:                      Disabled
  CUDA Device Driver Mode (TCC or WDDM):       WDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA):    Yes
  Device supports Compute Preemption:         Yes
  Supports Cooperative Kernel Launch:         Yes
  Supports MultiDevice Co-op Kernel Launch:   No
  Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.1, CUDA Runtime Version = 11.1, NumDevs = 1, Device0 = GeForce GTX 1060
Result = PASS
```

注意最后的结果=Pass代表成功。然后运行bandwidthTest.exe，同样注意Result=Pass。

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1\extras\demo_suite>bandwidthTest.exe
[CUDA Bandwidth Test] - Starting...
Running on...

Device 0: GeForce GTX 1060
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                   5814.4

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                   5872.0

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                   114923.2

Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
```

如果验证成功，则证明显卡环境配置成功。

## 7.2.Linux

在Linux下的安装思路与Windows类似，但是由于nvidia没有给Linux很好的支持，所以安装复杂度有所增加，主要原因在于：

1. 由于nvidia没有和Ubuntu很好的合作，所以Ubuntu默认采用另一个开源显卡驱动，该驱动和nvidia不兼容，所以需要禁用该驱动后安装nvidia驱动。
2. 安装nvidia驱动时需要关闭图形界面，进入命令行模式，该模式默认编码非utf-8，中文会乱码。
3. 安装过程在命令行中进行，需要配置的内容较多，过程需要对Linux较为熟悉。

所以Linux下配置显卡环境难度较大，需要多加尝试。这里简单罗列必要的步骤以及可能出现的问题。

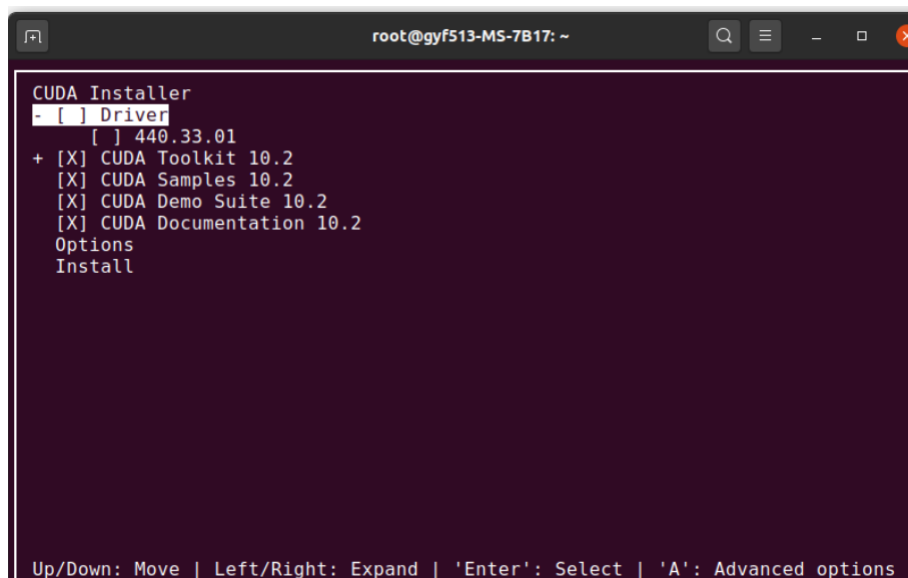
### 7.2.1.显卡驱动

首先来到[官网](#)查询个人设备的推荐驱动版本。

- Ubuntu的显卡驱动有三种安装方式：
  - 软件和更新安装
  - 命令行apt安装
  - .sh文件安装
- 我个人推荐按照顺序依次尝试
  - 最方便的是使用软件和更新安装，该方法能够完全依赖系统自动完成，但不是总能成功。
  - 如果第一种方法失败，则尝试采用第二种，采用类似`sudo apt install nvidia-driver-版本号`的命令尝试安装，该方法也较为方便。
  - 如果前两种方法都失败，则需要去官网下载.sh文件手动进行安装。
    1. 去到上文链接直接下载驱动文件。
    2. 在`/etc/modprobe.d/blacklist.conf`中添加黑名单`blacklist nouveau`禁用默认驱动。
    3. 确认禁用成功。
    4. 重启电脑进入BIOS关闭安全启动选项。
    5. `ctrl+Alt+F1`进入控制台，关闭显示管理器。
    6. `cd`到驱动位置
    7. `sudo sh`执行安装。
    8. 安装结束后重启显示管理器回到图形界面，终端输入`nvidia-smi`测试。
  - 实际操作过程中，12345步根据教程能够轻松完成。4是为7的铺垫。为了确保5中不受乱码影响，最好的办法是将驱动文件存放在纯英文路径下，或者安装中文文字包。
  - 而第7步则是最为关键的一步也是会出现诸多问题的一步。在这里遇到的问题种类很多，但大多能够搜索到相应解法，所以大家多加尝试总能找到解决的方法。

### 7.2.2.CUDA

- 到上文的官网链接中下载CUDA的.sh文件。
- Ubuntu20自带的gcc版本为9.7，通常建议先降级到gcc7，虽然我也尝试使用gcc9成功安装。但是在安装过程中会出现更多的配置选项。
- 使用`sudo sh`运行脚本，CUDA会自带显卡驱动，所以在安装是取消Driver的选项。



- 安装过程中同样会出现一些配置选项，大家根据内容百度选择即可。
- 结束后使用`nvcc -V`命令测试。

### 7.2.3.cuDNN

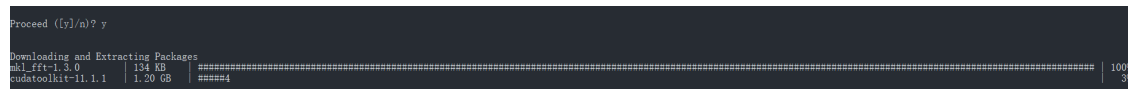
- 如果成功安装驱动和CUDA，就成功越过了所有的难关。Linux下的cuDNN安装方法和Windows下一致。安装成功后运行测试即可。

## 7.3.pytorch测试显卡

- 我们首先使用conda建立一个空的虚拟环境Torch，然后激活该环境，输入命令：

```
conda install pytorch torchvision cudatoolkit=版本号
```

解析完成后输入y开始安装：



- 安装成功后，我们直接在命令行输入python进入交互式解释器，然后逐行输入pytorch设备测试代码：

```
import torch
flag = torch.cuda.is_available ()
print (flag)
```

如果返回True，则证明该设备上已经成功配置好了pytorch的GPU环境。

值得注意的是，我个人在前几年安装CUDA和cuDNN时需要格外注意版本的匹配问题，包括显卡型号、显卡驱动版本、CUDA版本、cuDNN版本都有较为严格的对应方式。现在搜索到的很多教程中也都会做很多关于版本问题的核对。然而在写这一次教程时我发现，Nvidia最新的这些工具中少了很多对版本的严格要求，比如cuDNN仅分为适配CUDA10.2和CUDA11.0、11.1、11.2两个版本。也就是说Nvidia在不断更新的过程中有了更好的兼容性，大家在安装时的实际情况可能与搜索到的教程不符，这时应先按照官网的说明进行具体的选择。

