

---

# 张学志の博客

发布 1.0

张学志

2021 年 06 月 13 日



---

Contents:

---

<b>1</b>	<b>目录</b>	<b>3</b>
1.1	深度学习 . . . . .	3
1.2	PCL . . . . .	29
1.3	Linux . . . . .	63
1.4	博客运维 . . . . .	120



**简单介绍:**

这是张学志的新博客。欢迎您的访问。博客目前正在建设中。。。

**作者站点:**

- [网站源码](#)
- [CSDN 博客 \(访问量: 100 万\)](#)

**朋友站点:**

- [袁勇](#)



## 1.1 深度学习

### 1.1.1 Caffe

#### caffe 安装系列——综述

##### 说明

- 网上关于 caffe 的安装教程非常多，但是关于每一步是否操作成功，出现了什么样的错误又该如何处理没有给出说明。因为大家的操作系统的环境千差万别，按照博客中的教程一步步的安装，最后可能失败——这是很常见的哦。有的教程甚至省略了一些细节部分，让小白更不知道如何判断每一步是否操作成功，如何处理出现的错误。
- 作者花费了很长时间才成功地将 caffe 装完，期间遇到好多错误，多次重装操作系统。现在将经验写下来，一方面为了和大家分享，讨论；另一方面是为了记录一下下 ~~~

##### 环境

- 操作系统: *Ubuntu 14.04*
- GCC/G++:*4.7.x*
- OpenCV: *2.4.11* 和 *3.0.0*
- Matlab :*R2014b(a)*
- Python: *2.7*

## 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 cuda 和 cudnn
- 安装 Matlab
- 安装 OpenCV
- 安装 Python 依赖包
- 安装 caffe

## 0. 准备工作

- 安装 Ubuntu 14.04 (15.04), 最好安装较新版本的 Ubuntu。为什么选择 Ubuntu 呢? 一方面, 个人使用习惯, 感觉 Ubuntu 安装软件等特别方便, 使用特别顺手; 另一方面, caffe 项目最初貌似在 Ubuntu 上开发的, 原生嘛。
- 安装过程需要下载东东, 因此需要联网。

## 1. 安装 GCC4.7 和 G++4.7 并降级

- 为什么要先安装 GCC 和 G++, 并需要降级呢?
- Ubuntu14.04 版本默认的 GCC 和 G++ 都是 4.8。而 Matlab 默认支持的 mex 编译器是 GCC4.7.x 和 G++4.7.x。因此需要额外安装 GCC4.7 和 G++4.7 并降级。
- 为什么需要先安装编译器 GCC 和 G++, 而不是先安装显卡驱动和 cuda 等等呢?
- 首先, 先安装显卡驱动, 后安装 GCC 并降级的安装顺序过程中, 遇到了很多问题。比如, 在安装 CUDA SAMPLES 的过程中, 遇到了问题 `*/usr/bin/ld: cannot find -lGL*`, 后面通过网上教程, 重新连接该库文件, 仍然不能通过 OpenCV 的编译。但是, 先安装 GCC, 再安装显卡驱动, 就不会遇到这个问题了。
- 其次, 请先看下图。这是在成功地安装完显卡驱动之后, 查看加载的显卡的版本信息时返回的结果。注意, 包含了 GCC4.7.3。说明, 显卡安装过程中, 会和 GCC 的版本产生联系。。。这也就不难理解为什么在编译 Cuda Samples 过程中会遇到上面的问题。

```
root@optimal4:~# cat /proc/driver/nvidia/version
NVRM version: NVIDIA UNIX x86_64 Kernel Module  346.82  Wed Jun 17 10:3
:46 PDT 2015
GCC version:  gcc version 4.7.3 (Ubuntu/Linaro 4.7.3-12ubuntu1)
root@optimal4:~#
```

- 所以, 请先安装 GCC4.7 和 G++4.7, 然后在执行下面的步骤。



## 2. 安装 NVIDIA 显卡驱动

- 为什么需要安装 NVIDIA 显卡驱动, *Ubuntu* 没有自带的显卡驱动吗?
  - *Ubuntu* 自带的显卡驱动是开源的 Nouveau, 据说是一个比较烂的东东。而且, 最关键的是 cuda 不支持 Nouveau。如果想使用 cuda 进行 GPU 计算, 必须安装 NVIDIA 显卡驱动。
- 选择哪个版本的显卡驱动呢
  - 这个问题需要结合操作系统, 显卡和个人需求来讨论。
  - 操作系统影响显卡驱动的版本。比如, 我在 *Ubuntu14.04 Server* 上安装 NVIDIA-352 显卡驱动, 说是由于 dkms, 安装失败。目前, 通过 apt-get 方式可以安装的最新 NVIDIA 显卡驱动是 NVIDIA-346。
  - 显卡嘛, 硬件当然要和驱动适应才行
  - 个人需求, 主要从 cuda 的角度考虑。比如 cuda7.5 需要显卡驱动最低版本是 nvidia-352; cuda7.0 需要显卡驱动最低版本是 nvidia-336; cuda6.5 需要显卡驱动最低版本是 nvidia-33\*; 其他的记不清楚啦。。。
- 显卡驱动的安装方式有哪些
  - 方法一: 去 NVIDIA 官网下载相应的驱动二进制安装包, 然后安装。
  - 方法二: 通过 apt-get 来安装。
  - 区别: apt-get 安装方便, 但是不能安装最新的显卡驱动, 目前 *ubuntu14.04* 通过 apt-get 可以安装 nvidia-346 显卡。
  - 安装过程中注意事项: ① 需要关闭显示管理器, ② 二进制安装需要修改文件, 并重启。

## 3. 安装 cuda 和 cudnn

- 安装 cuda 的方式有哪些?
  - 方法一: 官网下载 cuda 开发包的二进制安装包进行安装。
  - 方法二: 官网下载 cuda 开发包的 deb 文件进行安装。
- cuda 版本的选择问题?
  - 根据个人需求和操作系统来决定, 显卡驱动版本。
  - cuda6.5 是一个分界点, cuda6.5 支持 compute\_11, compute\_12. etc. compute\_1X 系列架构; 从 cuda7.0 开始, 不支持 compute\_1X 系列架构, 最低是 compute\_20 架构。
  - cuda 对显卡驱动有要求。比如 cuda7.5 需要显卡驱动最低版本是 nvidia-352; cuda7.0 需要显卡驱动最低版本是 nvidia-336; cuda6.5 需要显卡驱动最低版本是 nvidia-33\*; 其他的记不清楚啦。因此, 结合自己操作系统可以安装的 NVIDIA 显卡驱动来决定选择哪个版本的 cuda。
- 为什么安装 cudnn?
  - cudnn 可以简单的理解为 CUDa cNN, 即在 GPU 上做卷积运算。最近几年, 深度学习很火, 尤其是 CNN (卷积神经网络)。通过 cudnn, 可以极大的提高 CNN 训练速度。简单的说, 实用 GPU 是为了快, 实用 cudnn 是为了更快。

## 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 cuda 和 cudnn
- 安装 Matlab
- 安装 OpenCV
- 安装 Python 依赖包
- 安装 caffe 的, 深度学习的一大应用对象就是图像和视频。而 OpenCV 是目前最火的开源计算机视觉库, 非常多的项目都用到了 OpenCV, 当然 caffe 也依赖 OpenCV。所以, 需要安装 OpenCV, 否则无法使用 caffe 哦。。。
- **OpenCV 安装简单吗?**
  - 答案是因人而异。有的人觉得简单, 可以自己弄, 有的人觉得难, 没关系, 大神们有写的安装脚本[点此 \\* 下载](#), 运行一下就 OK 了。
  - 但是, 使用别人脚本安装的方法, 也会遇到一些问题。如果遇到问题, 请 Google 解决。
  - 最简单的方式是使用我修改过的脚本, 按照顺序执行 12345 个脚本, 基本不需要修改就能成功安装。
  - 最后提醒, 安装 OpenCV 是挺麻烦的, 请耐心安装, 编译不过的话, 查看错误 Google, 解决了再编译, 一遍遍的尝试, 最后就能解决问题了。
- **应该安装 OpenCV 哪个版本呢?**
  - OpenCV 的版本和 cuda 的版本最好匹配。这样子安排的目的是为了减少错误出现的概率。比如, 我无错误编译成功的组合有【cuda7.0 + opencv3.0】, 【cuda6.5+opencv2.11】。
  - 应该安装最新的, 又不该安装最新的。呵呵, 好别扭哦。针对于低版本的 cuda, 最好安装 opencv2.x。而且是 opencv2.x 中最新的。
  - 低版本 cuda 安装 opencv2.x 的原因是, opencv 的一些文件中涉及一些关于 cuda 架构的设置, opencv2.x 中有支持相应的架构的配置。从这个角度看, cuda6.5 是最保险的, 因为它既支持 compute\_1x, 也支持更高的架构。

## 6. 安装 Python 相关依赖

- **为什么要安装 python 相关依赖 ???**
  - 首先, python 在 linux 中应用非常的广泛, 很多项目都会涉及 python, caffe 也不例外。
  - 其次, caffe 提供了 python 的接口, 为了后面使用, 也需要这些依赖。
- **这些依赖都可以通过 apt-get 安装吗?**
  - 答案是否定的。
  - 首先, google 一下 apt-get vs pip, 查看两者区别。
  - 其次, 安装 theano 的时候, 发现 apt-get 安装的 numPy 和 sciPy 无法通过测试, 并且造成 theano 测试失败。使用 pip 安装成功。参考《[Ubuntu14.04 安装 Theano 详细教程](#)》。

- 最后，在安装 caffe 的过程中，发现有几个 python 依赖包必须通过 pip 安装（即自行编译），否则无法成功地编译 caffe。

## 7. 安装 caffe

- 再重复一遍，请在上面所有步骤成功执行的前提下，安装 caffe，否则编译肯定不会通过的。
- **caffe 源代码能不能直接拿过来编译呢？**
  - 不能。至少需要修改一个文件 Makefile.config。该文件给 caffe 编译提供了必要的信息。
  - 如果 opencv 的版本是 3.0，还需要修改其他项。
  - 其他的请参考安装 caffe 的教程

## 总结

- 至此，ubuntu 下安装 caffe 的工作已经结束了。如果你完全按照本教程操作，相信你一定已经成功安装 caffe 了，并且对 caffe 有了一定的了解。
- **世上无难事只怕有坚持**，安装过程虽然很复杂，但是只要坚持，不断的 Google 解决它，caffe 就一定能安装。
- 错误不可怕，它是成功的障碍，同时也为我们成长提供了阶梯——所谓的能力，很大一部分是通过不断解决问题来获取的。
- 下面开始学习如何使用 caffe 做深度学习的研究喽，祝大家学习愉快。。。

## 参考

- 《Caffe + Ubuntu 15.04 + CUDA 7.0 新手安装配置指南》——欧新宇这个教程很好，请查看本教程的过程中，结合欧新宇的教程一并查看。
- 《caffe - GitHub 主页》

## caffe 安装系列——安装 GCC4.7 和 G++4.7 并降级

### 说明

- 网上关于 caffe 的安装教程非常多，但是关于每一步是否操作成功，出现了什么样的错误又该如何处理没有给出说明。因为大家的操作系统的环境千差万别，按照博客中的教程一步步的安装，最后可能失败——这是很常见的哦。有的教程甚至省略了一些细节部分，让小白更不知道如何判断每一步是否操作成功，如何处理出现的错误。
- 作者花费了很长时间才成功地将 caffe 装完，期间遇到好多错误，多次重装操作系统。现在将经验写下来，一方面为了和大家分享，讨论；另一方面是为了记录一下下 ~~~

## 环境

- 操作系统: *Ubuntu 14.04*
- GCC/G++:*4.7.x*
- OpenCV: *2.4.11* 和 *3.0.0*
- Matlab :*R2014b(a)*
- Python: *2.7*

## 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 cuda 和 cudnn
- 安装 Matlab
- 安装 OpenCV
- 安装 Python 依赖包
- 安装 caffe

## 安装 GCC4.7 和 G++4.7 并降级

- 注意: 需要联网。
- 下载并安装 gcc/g++ 4.7.x

```
sudo apt-get install -y gcc-4.7
sudo apt-get install -y g++-4.7
```

- 链接 gcc/g++ 实现降级

```
cd /usr/bin
sudo rm gcc
sudo ln -s gcc-4.7 gcc
sudo rm g++
sudo ln -s g++-4.7 g++
# 查看是否连接到 4.7.x
ls -al gcc g++
gcc --version
g++ --version
```

## caffe 安装系列——安装 NVIDIA 显卡驱动

### 说明

- 网上关于 caffe 的安装教程非常多，但是关于每一步是否操作成功，出现了什么样的错误又该如何处理没有给出说明。因为大家的操作系统的环境千差万别，按照博客中的教程一步步的安装，最后可能失败——这是很常见的哦。有的教程甚至省略了一些细节部分，让小白更不知道如何判断每一步是否操作成功，如何处理出现的错误。
- 作者花费了很长时间才成功地将 caffe 装完，期间遇到好多错误，多次重装操作系统。现在将经验写下来，一方面为了和大家分享，讨论；另一方面是为了记录一下下 ~~~

### 环境

- 操作系统: *Ubuntu 14.04*
- GCC/G++:*4.7.x*
- OpenCV: *2.4.11* 和 *3.0.0*
- Matlab :*R2014b(a)*
- Python: *2.7*

### 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 cuda 和 cudnn
- 安装 Matlab
- 安装 OpenCV
- 安装 Python 依赖包
- 安装 caffe

### 0. 版本说明

- cuda6.5 *NVIDIA-Linux-x86\_64-346.xx* (Ubuntu 14.04)
- cuda7.0 *NVIDIA-Linux-x86\_64-346.xx* (Ubuntu 14.04)
- cuda7.5 *NVIDIA-Linux-x86\_64-352.xx* (Ubuntu 15.04)

## 1. 使用 apt-get 安装

- 安装 32 位兼容包

```
# 32 位兼容包
# 为什么呢？安装二进制安装包时，提示是否安装 32bit 兼容包，因此，最好安装上 32 兼容包，以后操作系统上
安装 32bit 软件也方便啊。。。
sudo apt-get install lib32z1
```

- 关闭图形显示管理器

```
# 需要关闭图形显示管理器 (Display Manager)。显卡，当然和显示有关喽，因此最好关闭图形显示管理器。
stop lightdm
```

- 安装显卡驱动

```
# 安装显卡驱动。截止到目前为止，346 显卡驱动是 Ubuntu 可以通过 apt-get 安装的最新显卡驱动，也是
cuda7.0 支持的最低显卡驱动版本。
sudo apt-get install nvidia-346
```

- 重启电脑，检查是否成功

- 电脑重启后，命令行中输入 `cat /proc/driver/nvidia/version` 查看显卡信息。如下图所示
- 如果 `/proc/driver/` 目录下没有 `nvidia` 目录，说明 `nvidia` 显卡驱动没有加载，也就意味着安装失败。请参考其它教程，保证安装完成。

```
root@optimal4:~# cat /proc/driver/nvidia/version
NVRM version: NVIDIA UNIX x86_64 Kernel Module  346.82  Wed Jun 17 10:3
:46 PDT 2015
GCC version:  gcc version 4.7.3 (Ubuntu/Linaro 4.7.3-12ubuntu1)
root@optimal4:~#
```

- 注意：务必成功安装显卡驱动之后，再继续其它步骤，否则。。。。
- 注意：先将 GCC 和 G++ 降级，然后再安装驱动。

## 2. 使用二进制安装包安装

- 下载驱动
- 参考《Ubuntu-安装-cuda7.0-单显卡-超详细教程》安装。。。

## caffe 安装系列——安装 cuda 和 cudnn

### 说明

- 网上关于 caffe 的安装教程非常多，但是关于每一步是否操作成功，出现了什么样的错误又该如何处理没有给出说明。因为大家的操作系统的环境千差万别，按照博客中的教程一步步的安装，最后可能失败——这是很常见的哦。有的教程甚至省略了一些细节部分，让小白更不知道如何判断每一步是否操作成功，如何处理出现的错误。
- 作者花费了很长时间才成功地将 caffe 装完，期间遇到好多错误，多次重装操作系统。现在将经验写下来，一方面为了和大家分享，讨论；另一方面是为了记录一下下 ~~~

## 环境

- 操作系统: *Ubuntu 14.04*
- GCC/G++:*4.7.x*
- OpenCV: *2.4.11* 和 *3.0.0*
- Matlab :*R2014b(a)*
- Python: *2.7*

## 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 cuda 和 cudnn
- 安装 Matlab
- 安装 OpenCV
- 安装 Python 依赖包
- 安装 caffe

## 安装 cuda

- 参考 《Ubuntu-安装-cuda7.0-单显卡-超详细教程》 《Caffe + Ubuntu 15.04 + CUDA 7.0 新手安装配置指南》
- 注意 1: 安装 cuda 可以通过二进制安装包安装, 也可以通过 deb 包在线安装。本文使用二进制安装包安装。
- 注意 2: 由于前面已经安装了**显卡驱动**, 因此关于显卡驱动的选项, 选择 **no**, 即不再安装显卡驱动。

```
# 安装依赖库
apt-get install freeglut3-dev build-essential libx11-dev libxmu-dev libgl1-mesa-dev_
↪libglu1-mesa libglu1-mesa-dev libxi-dev
# 添加执行权限, 并安装。注意不要重复安装显卡驱动。
# cuda**run 根据自己下载的版本更改名称。
# 选择安装 Cuda Samples, 后边还会编译它。
# 选择创建软连接。cuda -> cuda7.0
stop lightdm
chmod a+x cuda_7.0.28_linux.run
sudo ./cuda_7.0.28_linux.run
```

- 修改/etc/profile 文件, 将 cuda 添加到环境变量中。【注: 关于这一点, 安装完 cuda 之后, 命令行中有提示!!! 细心看一下】
- 将/usr/local/cuda-7.0/bin 添加到环境变量 **PATH** 路径中, 这样一来, 就可以在任何路径下调用 cuda 相关的可执行文件了。
- 将/usr/local/cuda7.0/lib64 添加环境变量 **\*\*LD\_LIBRARY\_PATH\*\*** 中, 作为共享库使用。这样一来, 后面编译 Cuda Samples 和 OpenCV 时, 就不会提示找不到库的错误了。

- 操作 1: 将以下内容添加到文件 **\*\*/etc/profile\*\*** 的最后面, 保存后, 执行命令 `source /etc/profile`, 使配置生效。

```
PATH=/usr/local/cuda/bin:$PATH
export PATH
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
export LD_ LIBRARY_PATH
```

- 操作 2: 在目录 `/etc/ld.so.conf.d/` 下新建文件 **cuda.conf**, 并添加如下内容。然后执行命令 `sudo ldconfig`, 使配置生效。
- 解释: 下面第一行是上面提到的 `cuda` 库文件路径, 后面 3 行是后来综合调试错误和其它博文总结得到的。所以, 最好加上, 省的出错哦。。。

```
/usr/local/cuda/lib64
/lib
/usr/lib
/usr/lib32
```

- 检查 `cuda` 是否配置好, 在命令行中执行以下命令。

```
# 输入以下命令, 检查是否配置好。如下图所示, 说明安装好。
nvcc --version
```

```
test@optimal4:~$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2015 NVIDIA Corporation
Built on Mon Feb 16 22:59:02 CST 2015
Cuda compilation tools, release 7.0, V7.0.27
test@optimal4:~$
```

## 安装 CUDA SAMPLES

- 为什么安装 *cuda samples*?
- 一方面为了后面学习 `cuda` 使用, 另一方面, 可以检验 `cuda` 是否真的安装成功。如果 `cuda samples` 全部编译通过, 没有一个 `Error` (`Warning` 忽略), 那么就说明成功地安装了 `cuda`。但如果没有通过编译, 或者虽然最后一行显示 **PASS**, 但是编译过程中有 **ERROR**, 请自行 `GOOGLE` 解决之后, 再向下安装, 否则失之毫厘谬以千里!!!
- `make` 时, 请使用 `make -j`, 可以最大限度的使用 `cpu` 编译, 加快编译的速度。

```
# 切换到 cuda-samples 所在目录
# 注意, 换成自己的路径
cd /home/xuezhisd/NVIDIA_CUDA-7.0_Samples
# 编译 make (安装命令 sudo apt-get install cmake)
make -j
# 编译完毕, 切换 release 目录
cd ./bin/x86_64/linux/release
# 检验是否成功
# 运行实例 ./deviceQuery
./deviceQuery
# 可以认真看看自行结果, 它显示了你的 NVIDIA 显卡的相关信息。
```



```

Device 1: "GeForce GTX TITAN X"
  CUDA Driver Version / Runtime Version      7.0 / 7.0
  CUDA Capability Major/Minor version number: 5.2
  Total amount of global memory:             12288 MBytes (12884705280 bytes)
  (24) Multiprocessors, (128) CUDA Cores/MP: 3072 CUDA Cores
  GPU Max Clock rate:                        1076 MHz (1.08 GHz)
  Memory Clock rate:                         3505 Mhz
  Memory Bus Width:                          384-bit
  L2 Cache Size:                             3145728 bytes
  Maximum Texture Dimension Size (x,y,z)     1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory:            65536 bytes
  Total amount of shared memory per block:    49152 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:                      2147483647 bytes
  Texture alignment:                          512 bytes
  Concurrent copy and kernel execution:       Yes with 2 copy engine(s)
  Run time limit on kernels:                   No
  Integrated GPU sharing Host Memory:          No
  Support host page-locked memory mapping:     Yes
  Alignment requirement for Surfaces:          Yes
  Device has ECC support:                      Disabled
  Device supports Unified Addressing (UVA):     Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 3 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
> Peer access from GeForce GTX TITAN X (GPU0) -> GeForce GTX TITAN X (GPU1) : Yes
> Peer access from GeForce GTX TITAN X (GPU1) -> GeForce GTX TITAN X (GPU0) : Yes

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.0, CUDA Runtime Version = 7.0, NumDevs = 2
X TITAN X, Device1 = GeForce GTX TITAN X
Result = PASS

```

- ./deviceQuery 执行结果如下图所示: `[test@home/share/NVIDIA CUDA-7.0 Samples/bin/x86_64/linux/release]$`
- 一个 **Error** 例子
- 安装 CUDA SAMPLES 的过程中, 可能会出现错误 “/usr/bin/ld: cannot find -lGL”
- 通过 Google 搜索, 找到以下解决方法:

```

locate libGL.so
# 返回结果
#/usr/lib/i386-linux-gnu/mesa/libGL.so.1
#/usr/lib/i386-linux-gnu/mesa/libGL.so.1.2.0
sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1.2.0 /usr/lib/libGL.so

```

- 但是, 依然不能编译通过。忽略进行后面的步骤, 依然会提示一些 \*.so 文件找不到。最终在 OpenCV 那里完全无法再安装了, 返回卸掉 **NVIDIA DRIVER** 和 **cuda**, 重新安装驱动和 **cuda**, 成功地编译 cuda samples。
- 安装的驱动的方法是在命令行中执行:

```

sudo stop lightdm
sudo apt-get install nvidia-346

```

- 然后将 libGL.so 链接到 nvidia 提供的 libGL.so.346.82。后面编译 cuda samples 和 opencv 过程中顺利通过。

```

sudo ln -s /usr/lib/nvidia-346-updates/libGL.so.346.82 /usr/lib/libGL.so

```

- 即使用 NVIDIA 提供的 libGL.so 可以完成编译, 不再报错。
- 注意: 上面这个错误, 就是在先装显卡驱动, 后装 gcc4.7 过程遇到的。所以, 请先安装 gcc4.7 和 g++4.7, 然后再安装 NVIDIA 显卡驱动, 再安装 cuda。

## 安装 cudnn

- 安装 cudnn 比较简单, 简单地说, 就是复制几个文件: 库文件和头文件。将 cudnn 的头文件复制到 cuda 安装路径的 include 路径下, 将 cudnn 的库文件复制到 cuda 安装路径的 lib64 路径下。

```
# 解压文件
tar -zxvf cudnn-6.5-linux-x64-v2.tgz
# 切换路径
cd cudnn-6.5-linux-x64-v2
# 复制 lib 文件到 cuda 安装路径下的 lib64/
sudo cp lib* /usr/local/cuda/lib64/
# 复制头文件
sudo cp cudnn.h /usr/local/cuda/include/

# 更新软连接
cd /usr/local/cuda/lib64/
sudo rm -rf libcudnn.so libcudnn.so.6.5
sudo ln -s libcudnn.so.6.5.48 libcudnn.so.6.5
sudo ln -s libcudnn.so.6.5 libcudnn.so
```

- 到目前为止, cudnn 已经安装完了。但是, 是否安装成功了呢, 还得通过下面的 cudnn sample 测试。

```
# 运行 cudnn-sample-v2
tar -zxvf cudnn-sample-v2.tgz
cd cudnn-sample-v2
make
./mnistCUDNN
# 改程序运行成功, 说明 cudnn 安装成功。
```

- 此时可能出现错误: *./mnist CUDNN: error while loading shared libraries: libcudart.so.6.5: cannot ope*
- 解决方法参考 <https://groups.google.com/forum/#!topic/caffe-users/dcZrE3-60mc>
- 方法 1: 在命令行中执行 `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64`
- 方法 2: 在 `/etc/profile` 文件最后添加 `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64`, 并在命令行中执行 `source /etc/profile`。

## caffe 安装系列——安装 Matlab

### 说明

- 网上关于 caffe 的安装教程非常多, 但是关于每一步是否操作成功, 出现了什么样的错误又该如何处理没有给出说明。因为大家的操作系统的环境千差万别, 按照博客中的教程一步步的安装, 最后可能失败——这是很常见的哦。有的教程甚至省略了一些细节部分, 让小白更不知道如何判断每一步是否操作成功, 如何处理出现的错误。
- 作者花费了很长时间才成功地将 caffe 装完, 期间遇到好多错误, 多次重装操作系统。现在将经验写下来, 一方面为了和大家分享, 讨论; 另一方面是为了记录一下下 ~~~

## 环境

- 操作系统: *Ubuntu 14.04*
- GCC/G++:*4.7.x*
- OpenCV: *2.4.11* 和 *3.0.0*
- Matlab :*R2014b(a)*
- Python: *2.7*

## 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 cuda 和 cudnn
- 安装 Matlab
- 安装 OpenCV
- 安装 Python 依赖包
- 安装 caffe

### 1. 下载 Matlab

- 下载地址, 请自行百度解决。太大了, 不方便提供。
- 版本选择问题。最好下载较新的版本。比如 R2014a, R2014b 和 R2015a 等。
- 下载破解文件 Crack 文件。一般情况下, crack 文件是包含在下载的马赛安装镜像里面的——除非你下载的是 MathWork 公司提供的原版。如果里面没有 crack 文件, 自行 *Google* 解决。

### 2. 安装 Matlab

- Matlab 安装过程请参考下面的连接 (欧新宇)。里面给出了详细的教程。
- 说明几点:
  - Windows 下安装 Matlab, 直接双击安装文件即可。但是, 在 Ubuntu 中安装 Matlab 时, 是在**命令行中安装**的。具体而言, 在命令行中, 切换到 Matlab 安装包路径下, 执行 `sudo ./install`, 就会跳出安装对话框。
  - Ubuntu 安装 Matlab 需要在图形界面中操作。如上所言, 执行 `sudo ./install` 之后, 会跳出图形对话框, 如果是在纯粹的命令行上, 应该无法进行。
  - 安装过程中, 依次会提示**输入序列号, 证书路径**。
- 切记: 图形安装对话框结束之后, Matlab 并没有安装完成。
  - 这时尝试打开 Matlab 会报错。通过在命令行中执行 `matlab`, 可以看到返回的错误信息是**没哟激活**。

- 但是, 你可能疑惑, 为什么呢? 最后激活步骤, 明明导入了证书文件 (\*.lic) ?!
- 这是因为, 我们使用的是盗版的 Matlab, 所以还需要额外的一步: 替换 *libmwservices.so* 库文件。
- 替换需要在命令行中操作, 还需要 root 权限 (超级用户)。因为需要将 *libmwservices.so* 复制到 */usr/local/Matlab/R2014b/\*\*\** 目录下, 该目录属于 root 用户, 所以需要在命令行中是, 使用 *cp* (或 *mv*) 命令完成。
- 替换 *libmwservices.so* 之后就可以成功运行 Matlab 了。

```
sudo cp libmwservices.so /usr/local/MATLAB/R2014a/bin/glnxa64/
```

### 3. GCC 和 G++ 版本问题

- 前面《caffe 安装系列——安装 GCC4.7 和 G++4.7 并降级》介绍了如何实现 GCC 和 G++ 降级, 并提到了为什么这样操作——Ubuntu14.04 版本默认的 GCC 和 G++ 都是 4.8。而 Matlab 默认支持的 mex 编译器是 GCC4.7.x 和 G++4.7.x。因此需要额外安装 GCC4.7 和 G++4.7 并降级。\* 请注意一点: 《caffe 安装系列——安装 GCC4.7 和 G++4.7 并降级》实现了 GCC 和 G++ 的降级, 但是并没有更改任何的关于 Matlab 文件。Matlab 要使用 GCC4.7 和 G++4.7, 还需要做一些工作——重新连接 *libstdc++.so.6* 文件。
- 具体操作命令如下所示:

```
# 拷贝文件
sudo cp /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.20 /usr/local/MATLAB/R2014a/sys/os/
↪glnxa64/libstdc++.so.6.0.20
# libstdc++.so.6.0.20 的版本, 可能因为系统不同而不同, 使用最新的就可以了。Ubuntu14.04 的最新版本是
libstdc++.so.6.0.19。
# 切换路径
cd /usr/local/MATLAB/R2014a/sys/os/glnxa64/
# 修改权限
chmod 555 libstdc++.so.6.0.20
# (仅仅是备份, 可以不操作)。
sudo mv libstdc++.so.6 libstdc++.so.6.backup
# 重新链接
ln -s libstdc++.so.6.0.20 libstdc++.so.6
# 使配置生效
sudo ldconfig -v
```

### 4. 建立 Matlab 的快捷方式

- 新建一个文本文件 (使用 *vi*, *gedit* 等), 输入以下内容。退出保存。将该文件放到 */home/yourname/Desktop/* 目录下, 既可以看到 Matlab 的快捷方式。

```
[Desktop Entry]
Name=Matlab 2014b
Exec=/usr/local/MATLAB/R2014b/bin/matlab -desktop
Icon=/usr/local/MATLAB/R2014b/toolbox/nnet/nnresource/icons/matlab.png
Type=Application
```

- **Exec** 是可执行文件, 需要是绝对路径。
- **Icon** 是图标, 如果没有设置的话, 不会显示 Matlab 的标志图标
- 以上路径根据自己安装路径更改。
- 快捷方式带一个小锁的问题。看看该文件的属主是谁, 如果不是自己, 使用 *chown* 命令将属主改为自己; 如果没有执行权限, 使用 *chmod* 命令增加可执行权限。

## 检查

- **检查 Matlab 是否安装成功。** 在命令行中输入 `matlab`，回车。如果打开 `Matalb`，说明安装成功。如果没有打开，将路径切换到 `Matlab` 安装路径下的 `bin` 文件夹下，再次执行 `matlab`，如果还没有打开，就说明没有安装成功。如果任何一次打开了，就说明安装成功了。
- **检查 GCC 版本问题。** 通过命令 `strings /usr/local/MATLAB/R2014a/sys/os/glnxa64/libstdc++.so.6 | grep GLIBCXX`，可以看一下，是否已经成功包含了 `GLIBCXX_3.4.20` (Ubuntu14.04 中是 `GLIBCXX_3.4.19`)，如果已经存在，基本上就成功了。

## 参考链接

- 《Caffe + Ubuntu 15.04 + CUDA 7.0 新手安装配置指南》——欧新宇
- 《caffe 安装系列——安装 GCC4.7 和 G++4.7 并降级》

## caffe 安装系列——安装 OpenCV

### 说明

- 网上关于 `caffe` 的安装教程非常多，但是关于每一步是否操作成功，出现了什么样的错误又该如何处理没有给出说明。因为大家的操作系统的环境千差万别，按照博客中的教程一步步的安装，最后可能失败——这是很常见的哦。有的教程甚至省略了一些细节部分，让小白更不知道如何判断每一步是否操作成功，如何处理出现的错误。
- 作者花费了很长时间才成功地将 `caffe` 装完，期间遇到好多错误，多次重装操作系统。现在将经验写下来，一方面为了和大家分享，讨论；另一方面是为了记录一下下 ~~~

### 环境

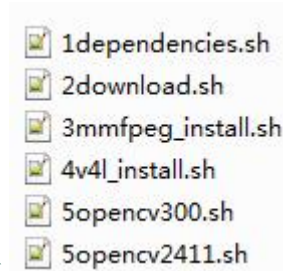
- 操作系统: **Ubuntu 14.04**
- GCC/G++:**4.7.x**
- OpenCV: **2.4.11** 和 **3.0.0**
- Matlab :**R2014b(a)**
- Python: **2.7**

### 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 `cuda` 和 `cudnn`
- 安装 `Matlab`
- 安装 `OpenCV`

- 安装 Python 依赖包
- 安装 caffe

## 安装 OpenCV



- 安装前, 请下载我的安装脚本, 如下图所示。点此下载
- 这个脚本是我根据 GitHub 上大神写的安装 OpenCV 的脚本改造的, 这个脚本有一些问题, 有些包安装不上, 比如 mmfpeg, 有些包前后安装相互冲突, 后面的会把前面的卸载等等。
- 使用这个脚本的方法就是, 先给 5 个脚本执行权限 `chmod +x *.sh`, 然后依照顺序执行这些脚本。
- 执行 1dependecies.sh 的方法是 `sudo sh 1dependencies.sh`。这个脚本主要是安装 OpenCV 的一些依赖库。

```
#!/bin/bash
#edited by xuezhi zhang.
echo "--- Removing any pre-installed ffmpeg and x264"
sudo apt-get -qq remove ffmpeg x264 libx264-dev
#function install_dependency {
#    echo "--- Installing dependency: $1"
#    sudo apt-get -y install $1
#}
echo "Installing dependency"
sudo apt-get -y install libopencv-dev
sudo apt-get -y install build-essential
# conflict
#sudo apt-get -y install checkinstall
sudo apt-get -y install cmake
sudo apt-get -y install pkg-config
sudo apt-get -y install yasm
sudo apt-get -y install libjasper-dev
sudo apt-get -y install libavcodec-dev
sudo apt-get -y install libavformat-dev
sudo apt-get -y install libswscale-dev
sudo apt-get -y install libdc1394-22-dev
sudo apt-get -y install libxine-dev
#echo "=====
# conflict with libxine-dev
#sudo apt-get -y install libxine2-dev
sudo apt-get -y install libgstreamer0.10-dev
sudo apt-get -y install libgstreamer-plugins-base0.10-dev
sudo apt-get -y install libv4l-dev
sudo apt-get -y install python-dev
sudo apt-get -y install python-numpy
sudo apt-get -y install python-scipy
sudo apt-get -y install python-sphinx
sudo apt-get -y install libtbb-dev
```

(下页继续)

(续上页)

```
sudo apt-get -y install libqt4-dev
sudo apt-get -y install libgtk2.0-dev
sudo apt-get -y install libfaac-dev
sudo apt-get -y install libmp3lame-dev
sudo apt-get -y install libopencore-amrnb-dev
sudo apt-get -y install libopencore-amrwb-dev
sudo apt-get -y install libtheora-dev
sudo apt-get -y install libvorbis-dev
sudo apt-get -y install libxvidcore-dev
sudo apt-get -y install x264
sudo apt-get -y install v4l-utils
sudo apt-get -y install unzip
sudo apt-get -y install libgtk2.0-0
sudo apt-get -y install libjpeg-dev
sudo apt-get -y install libjpeg62
# conflict
#sudo apt-get -y install libjpeg62-dev
sudo apt-get -y install libtiff4-dev
sudo apt-get -y install libtiff5-dev
sudo apt-get -y install qt5-default
echo "=====
echo "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^Over^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
echo "=====

#install_dependency ffmpeg
#install fail, run file of ffmpeg_install to install it
```

- 执行 2download.sh 的方法是 `sudo sh 2download.sh`。这个脚本主要是下载一个安装包，为后面编译安装做准备。
- 执行 2download.sh 前，修改一下脚本，选择 `opencv-2.4.11` 和 `opencv-3.0.0`，注释（#）另外一个就 OK。

```
echo "===== "  
echo "--- Downloading v4l-utils"  
echo "===== "  
wget http://www.linuxtv.org/downloads/v4l-utils/v4l-utils-0.8.5.tar.bz2  
  
echo "===== "  
echo "--- Downloading mmfpeg-2.8"  
echo "===== "  
wget http://ffmpeg.org/releases/ffmpeg-2.8.tar.bz2  
echo "===== "  
echo "Downloading OpenCV"  
echo "===== "  
# opencv-2.4.11 和 opencv-3.0.0 根据自己的需要, 选择下载一个就 OK 了。  
# 下载 opencv-2.4.11.zip  
wget -O opencv-2.4.11.zip http://sourceforge.net/projects/opencvlibrary/files/opencv-  
↳ unix/2.4.11/opencv-2.4.11.zip/download  
# 下载 opencv-3.0.0  
# wget -O opencv-3.0.0.zip http://sourceforge.net/projects/opencvlibrary/files/opencv-  
↳ unix/3.0.0/opencv-3.0.0.zip/download
```

- 执行 3mmfpeg\_install.sh 的方法是 `sudo sh 3mmfpeg_install.sh`。这个脚本主要是安装 mmfpeg。为后面安装 OpenCV 做准备。
- 因为 mmfpeg 不能使用 apt-get 安装，因此需要自己编译安装。



```

echo "=====
echo "Installing dependency"
echo "=====
sudo apt-get install -y libx264-dev libxext-dev libxfixes-dev

echo "=====
echo "--- Configure mmfpeg-2.8"
echo "=====
tar -xf ffmpeg-2.8.tar.bz2
cd ffmpeg-2.8
./configure --prefix=/usr/local/ffmpeg --enable-gpl --enable-version3 --enable-
↪nonfree --enable-postproc --enable-pthreads --enable-libfaac --enable-libmp3lame --
↪enable-libtheora --enable-libx264 --enable-libxvid --enable-x11grab --enable-
↪libvorbis

echo "=====
echo "--- Making mmfpeg-2.8"
echo "=====
make -j

echo "=====
echo "--- Installing mmfpeg-2.8"
echo "=====
make install
cd ..

```

- 执行 4v4l\_install.sh, 安装 v4l。

```

echo "=====
echo "---Installing v4l---"
echo "=====
#wget http://www.linuxtv.org/downloads/v4l-utils/v4l-utils-0.8.5.tar.bz2
tar -xvf v4l-utils-0.8.5.tar.bz2
cd v4l-utils-0.8.5
make -j
sudo make install
cd ..

```

- 安装 OpenCV 有两个选择, OpenCV2.4.11 和 OpenCV3.0.0, 分别对应于 5opencv2411.sh 和 5opencv300.sh, 根据自己的需要选择执行哪个脚本。【注意: 和前面 2download.sh 中下载的版本保持一致】
- 注意根据自己的显卡, 修改 CUDA\_GENERATION。如果是 Fermi 架构, 就需要将 Kepler 修改成 Fermi
- 安装 opencv2.4.11 之前, 根据最下面的错误分析中的错误 1, 修改 opencv-path/cmake/OpenCVDetectCUDA.cmake 文件之后, 再执行脚本, 否则可能出错。

```

# added by xuezhi zhang.
#
# OpenCV 2.4.11
# http://vinayhacks.blogspot.com.es/2011/11/installing-opencv-231-with-ffmpeg-on-64.
↪html
echo "=====
echo "Installing OpenCV"
echo "=====

sudo unzip opencv-2.4.11.zip
cd opencv-2.4.11
sudo mkdir build

```

(下页继续)



(续上页)

```

cd build
# 注意根据自己的显卡, 修改 CUDA_GENERATION。如果是 Fermi 架构, 就需要将 Kepler 修改成 Fermi
sudo cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_
↳TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D_
↳INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON_
↳CUDA_GENERATION=Kepler ..
#make -j 32
sudo make -j
sudo make install
sudo sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.conf'
sudo ldconfig
echo "OpenCV ready to be used"

```

## 判定 OpenCV 是否安装成功

- 安装过程中如果没有返回错误, 说明安装没有问题, 如果有任何错误提示, 请自行 Google 解决。

```

-- Installing: /usr/local/share/OpenCV/samples/gpu/video_reader.cpp
-- Installing: /usr/local/share/OpenCV/samples/gpu/video_writer.cpp
-- Installing: /usr/local/share/OpenCV/samples/gpu/driver_api_multi.cpp
-- Installing: /usr/local/share/OpenCV/samples/gpu/pyr1k_optical_flow.cpp
-- Installing: /usr/local/share/OpenCV/samples/gpu/bgfg_segm.cpp
-- Installing: /usr/local/share/OpenCV/samples/gpu/aloeR.jpg
-- Installing: /usr/local/share/OpenCV/samples/gpu/aloeL.jpg
-- Installing: /usr/local/share/OpenCV/samples/gpu/tsucuba_left.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/tsucuba_right.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/rubberwhale1.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/rubberwhale2.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/basketball1.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/basketball2.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/road.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/CMakeLists.txt
-- Installing: /usr/local/share/OpenCV/samples/ocl/adaptive_bilateral_fil
-- Installing: /usr/local/share/OpenCV/samples/ocl/surf_matcher.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/tvl1_optical_flow.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/stereo_match.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/facedetect.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/hog.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/clahe.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/squares.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/pyr1k_optical_flow.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/bgfg_segm.cpp
-- Installing: /usr/local/share/OpenCV/samples/ocl/CMakeLists.txt
OpenCV ready to be used

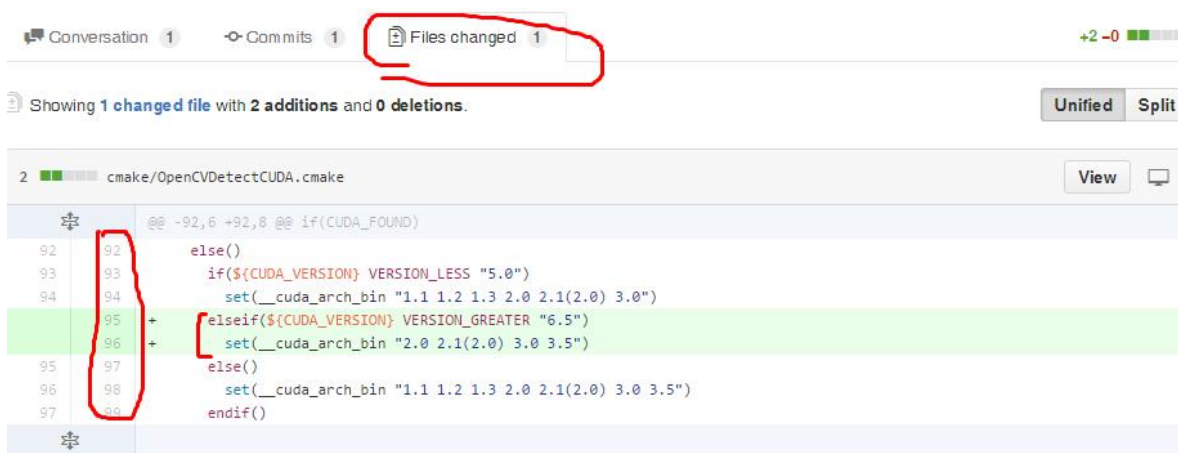
```

- 安装成功之后, 如下所示。

```
root@optimal5:/home/share/opencv#
```

## OpenCV 安装过程中的错误

- 因为 OpenCV 有很多依赖包, 因此安装过程非常的复杂。如果依赖的包没有安装, 或者操作系统的编译环境有问题的话, 很造成容易编译失败。
- 错误 1: *cannot build OpenCV 2.4.11 with CUDA 7.0*
- 这个问题已经解决, 参考网址。
- 这是一个和 cuda 的架构 1.1 (cuda7.0 不再支持架构 1.x) 有关的错误, 通过修改 `opencvpath/cmake/OpenCVDetectCUDA.cmake` 文件解决。如下图所示。



- 错误 2: *error: a storage class is not allowed in an explicit specialization*
- 这是 cuda 和 opencv 版本不适应的错误。
- 这个问题也已经解决, 参考网址。
- 因此, 最好的组合方式【cuda7.0 + opencv3.0.】和【cuda6.5+opencv2.4.11】。这两种组合方法, 我都已经在 Ubuntu14.04 操作系统 (64 位版本) 上成功地安装了。
- 网上安装 OpenCV 的教程很多都是通过 Install-OpenCV 脚本来安装的, 但我在使用过程中发现会出现问题, 而网络教程有没有说明遇到问题如何解决, 因此很坑。最终, 借助 Google, 解决的所有问题, 成功安装了所有依赖和 OpenCV3.0。因此, 如果大家遇到什么编译错误, 不要害怕, google 一下, 一般别人已经遇到这个问题, 并成功地解决了。

## 参考

- 《Caffe + Ubuntu 15.04 + CUDA 7.0 新手安装配置指南》——欧新宇
- opencv 项目
- 我的 opencv 安装脚本
- GitHub 上的 opencv 安装脚本

## caffe 安装系列——安装 python 依赖包

### 说明

- 网上关于 caffe 的安装教程非常多, 但是关于每一步是否操作成功, 出现了什么样的错误又该如何处理没有给出说明。因为大家的操作系统的环境千差万别, 按照博客中的教程一步步的安装, 最后可能失败——这是很常见的哦。有的教程甚至省略了一些细节部分, 让小白更不知道如何判断每一步是否操作成功, 如何处理出现的错误。
- 作者花费了很长时间才成功地将 caffe 装完, 期间遇到好多错误, 多次重装操作系统。现在将经验写下来, 一方面为了和大家分享, 讨论; 另一方面是为了记录一下下 ~~~

## 环境

- 操作系统: *Ubuntu 14.04*
- GCC/G++:*4.7.x*
- OpenCV: *2.4.11* 和 *3.0.0*
- Matlab :*R2014b(a)*
- Python: *2.7*

## 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 cuda 和 cudnn
- 安装 Matlab
- 安装 OpenCV
- 安装 Python 依赖包
- 安装 caffe

## 安装 theano 依赖包

- 安装基本的依赖包

```
# 安装 ipython
# 安装 gfortran, 后面编译过程中会用到
# 安装 blas, Ubuntu 下对应的是 libopenblas, 其它操作系统可能需要安装其它版本的 blas——这是个 OS 相关的。
# 安装 lapack, Ubuntu 下对应的是 liblapack-dev, 和 OS 相关。
# 安装 atlas, Ubuntu 下对应的是 libatlas-base-dev, 和 OS 相关。
# -y, 遇到需要用户选择的项, 都选 y
sudo apt-get install -y ipython ipython-notebook pandoc
sudo apt-get install -y gfortran libopenblas-dev liblapack-dev libatlas-base-dev
# 安装 pip
sudo apt-get install -y python-pip python-dev python-nose g++ git
sudo apt-get install -y python-numpy python-scipy
# 自己编译的原因是, 防止 theano 出错
# 使用豆瓣的 python 源, 下载速度快
sudo pip install numpy -i http://pypi.douban.com/simple
sudo pip install scipy -i http://pypi.douban.com/simple
# 查看目录/tmp/pip_build_root
sudo apt-get install -y python-matplotlib python-sklearn python-sklearn-lib
```

- 安装 boost 和 pyCUDA

```
# 安装 boost 和和 pyCUDA。pyCUDA 需要 boost。
sudo apt-get install -y libboost-all-dev
# 下载 pycuda 源代码
git clone --recursive http://git.tiker.net/trees/pycuda.git
cd pycuda
sudo ./configure.py --cuda-root=/usr/local/cuda --cudadriv-lib-dir=/usr/lib/x86_64-
↳linux-gnu --boost-inc-dir=/usr/include --boost-lib-dir=/usr/lib --boost-python-
↳libname=boost_python --boost-thread-libname=boost_thread --no-use-shipped-boost
# 多核编译
make -j
# 安装
sudo python setup.py install
```

- 安装 Theano

```
sudo pip install theano -i http://pypi.douban.com/simple
```

- 安装 pyCaffe 需要的依赖

```
# 使用 apt-get 安装大多数包
sudo apt-get install -y python-numpy python-scipy python-matplotlib python-sklearn-
↳python-skimage python-h5py python-protobuf python-leveldb python-networkx python-
↳nose python-pandas python-gflags cython ipython python-yaml
sudo apt-get install -y protobuf-c-compiler protobuf-compile
# 因为有些包使用 apt-get 安装失败, 所以使用 pip 重新安装它们, 防止后面编译 caffe 过程中报错。
sudo pip install protobuf --upgrade -i http://pypi.douban.com/simple
sudo pip install pillow --upgrade -i http://pypi.douban.com/simple
sudo pip install six --upgrade -i http://pypi.douban.com/simple
```

至此, 安装 python 依赖包的工作已经完成。也没有什么检查安装成功的评价标准。

## caffe 安装系列——安装 caffe

### 说明

- 网上关于 caffe 的安装教程非常多, 但是关于每一步是否操作成功, 出现了什么样的错误又该如何处理没有给出说明。因为大家的操作系统的环境千差万别, 按照博客中的教程一步步的安装, 最后可能失败——这是很常见的哦。有的教程甚至省略了一些细节部分, 让小白更不知道如何判断每一步是否操作成功, 如何处理出现的错误。
- 作者花费了很长时间才成功地将 caffe 装完, 期间遇到好多错误, 多次重装操作系统。现在将经验写下来, 一方面为了和大家分享, 讨论; 另一方面是为了记录一下下 ~~~

### 环境

- 操作系统: **Ubuntu 14.04**
- GCC/G++:**4.7.x**
- OpenCV: **2.4.11** 和 **3.0.0**
- Matlab :**R2014b(a)**
- Python: **2.7**

## 安装步骤

- 其它链接
- 综述
- 安装 GCC4.7 和 G++4.7 并降级
- 安装显卡驱动
- 安装 cuda 和 cudnn
- 安装 Matlab
- 安装 OpenCV
- 安装 Python 依赖包
- 安装 caffe
- 这已经是 caffe 安装过程的最后一步了。但是行百里者，半于九十，因此还要小心。

## 安装 Google Logging Library (glog)

- 下载 glog。
- glog 下载地址
- 安装命令如下所示

```
# 解压
tar -zxvf glog-0.3.3.tar.gz
# 切换路径
cd glog-0.3.3
sudo ./configure
sudo make -j
sudo make install
```

## 安装其它依赖

- 执行以下命令即可。

```
sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev \
↳ libboost-all-dev libhdf5-serial-dev
sudo apt-get install -y libgflags-dev libgoogle-glog-dev liblmdb-dev protobuf-
↳ compiler protobuf-c-compiler python-pandas
```

## 编辑 Makefile.config 文件

- 操作命令如下所示。

```
unzip caffe-master.zip # 本地解压 caffe-master
# 切换路径
cd /caffe-master
# caffe 源文件中没有 Makefile.config, 需要复制 Makefile.config.example
cp Makefile.config.example Makefile.config
```

(下页继续)

(续上页)

```
# 编辑 Makefile.config  
vi Makefile.config
```

- 修改 Makefile.config
  - 取消第 5 行的注释, 即将 `#USE_CUDNN=1` 改为 `USE_CUDNN=1`;
  - 如果使用本教程系列安装的, 就不需要修改 `BLAS=atlas`, 如果是参考欧新宇的教程, 安装了 MKL, 需要改成 `BLAS=mkl`;
  - 启用 CUDNN, 加注释: `CPU_ONLY:=1` 改成 `# CPU_ONLY:=1`;
  - 配置路径, 实现 caffe 对 Python 和 Matlab 接口的支持:

```
PYTHON_LIB := /usr/local/lib  
MATLAB_DIR := /usr/local/MATLAB/R2014a
```

## 编辑 Makefile 文件

- 如果 openCV 版本是 2.4.x, 此小节可以不再阅读
- 如果 openCV 版本 3.0, 还需要修改 Makefile 文件, 实现对 OpenCV 3.x 的支持。
- 在 Makefile 文件中查找 “Derive include and lib directories” 一节, 修改 “`LIBRARIES +=`” 的最后一行, 增加 `opencv_imgcodecs`, 修改之后为:

```
LIBRARIES += opencv_core opencv_highgui opencv_imgproc opencv_imgcodecs
```

## 编译 caffe-master

- 依次执行下面的命令, 编译 caffe:

```
make all -j  
make test -j  
make runtest -j
```

- runtest 执行结束之后, 如下图所示。这样子就说明安装成功了。



```

root@optimal4: /home/share/caffe/caffe
[ OK ] EltwiseLayerTest/2.TestSumCoeff (0 ms)
[ RUN ] EltwiseLayerTest/2.TestSumCoeffGradient
[ OK ] EltwiseLayerTest/2.TestSumCoeffGradient (75 ms)
[-----] 10 tests from EltwiseLayerTest/2 (367 ms total)

[-----] 4 tests from ContrastiveLossLayerTest/3, where TypeParam = caffe::G
PUDevice<double>
[ RUN ] ContrastiveLossLayerTest/3.TestForward
[ OK ] ContrastiveLossLayerTest/3.TestForward (4 ms)
[ RUN ] ContrastiveLossLayerTest/3.TestForwardLegacy
[ OK ] ContrastiveLossLayerTest/3.TestForwardLegacy (1 ms)
[ RUN ] ContrastiveLossLayerTest/3.TestGradientLegacy
[ OK ] ContrastiveLossLayerTest/3.TestGradientLegacy (436 ms)
[ RUN ] ContrastiveLossLayerTest/3.TestGradient
[ OK ] ContrastiveLossLayerTest/3.TestGradient (438 ms)
[-----] 4 tests from ContrastiveLossLayerTest/3 (879 ms total)

[-----] Global test environment tear-down
[=====] 1677 tests from 242 test cases ran. (314482 ms total)
[ PASSED ] 1677 tests.

YOU HAVE 2 DISABLED TESTS

root@optimal4: /home/share/caffe/caffe#

```

- 编译 Python 和 Matlab 用到的 caffe 文件

```

make pycaffe -j
make matcaffe -j

```

```

root@optimal5: /home/share/caffe/caffe-master# m
touch python/caffe/proto/__init__.py
CXX/LD -o python/caffe/_caffe.so python/caffe/_
PROTOC (python) src/caffe/proto/caffe.proto
root@optimal5: /home/share/caffe/caffe-master# m
MEX matlab/+caffe/private/caffe_.cpp
Building with 'g++'.
MEX completed successfully.
root@optimal5: /home/share/caffe/caffe-master#

```

- pycaffe 和 matcaffe 编译完成后, 如下图所示:
- 编译过程中可能会遇到错误, 比如 `***./include/caffe/util/cudnn.hpp:8:34: fatal error: caffe/proto/caffe.pb.h: No such file or director***`, 这是因为 protobuf 和 pillow 没有安装, 或者是通过 apt-get 安装的。使用 pip 重新安装一遍即可解决问题。删除 caffe-mast (提前保存 Makefile.config 和 Makefile), 解压, 解压重新编译。

```

pip install protobuf --upgrade -i http://pypi.douban.com/simple
pip install pillow --upgrade -i http://pypi.douban.com/simple

```

## 设置 Python 环境变量

- 此时虽然编译完成, 但是 python 还不能使用 caffe, 需要设置 python 的环境变量, 将其 caffe/python 路径添加到 python 环境变量中。
- 操作命令如下所示:

```
sudo vi /etc/profile # 编辑 profile 文件
# 在最后面添加以下语句, 注意将 path 换成你的系统下的路径
export PYTHONPATH=/path/to/caffe/python:$PYTHONPATH
```

- 在 caffe 安装整个过程中, 我么修改/etc/profile 三次, 如下图所示。

```
PATH=/usr/local/cuda/bin:$PATH
export PATH
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
export LD_LIBRARY_PATH
PYTHONPATH=/home/test/caffe/caffe/python:$PYTHONPATH
root@optimal5:~#
```

## 使用 MNIST 数据集进行测试

- 参考欧新宇的教程即可。
- 操作命令如下所示:

```
# 1. 数据预处理
sh data/mnist/get_mnist.sh
# 2. 重建 lmdb 文件。Caffe 支持三种数据格式输入网络, 包括 Image(.jpg, .png 等), leveldb, lmdb,
根据自己需要选择不同输入吧。
# 生成 mnist-train-lmdb 和 mnist-test-lmdb 文件夹, 这里包含了 lmdb 格式的数据集
sh examples/mnist/create_mnist.sh

# 3. 训练 mnist
sh examples/mnist/train_lenet.sh
```

**\*\* 注意: \*\*** 如果在使用过程中出现检测不到 NVIDIA 显卡的情况, 重装显卡驱动和 cuda 即可。

## 总结

- 至此, ubuntu 下安装 caffe 的工作已经结束了。如果你完全按照本教程操作, 相信你一定已经成功安装 caffe 了, 并且对 caffe 有了一定的了解。
- 世上无难事只怕有坚持, 安装过程虽然很复杂, 但是只要坚持, 不断的 Google 解决它, caffe 就一定能安装。
- 错误不可怕, 它是成功的障碍, 同时也为我们成长提供了阶梯——所谓的能力, 很大一部分是通过不断解决问题来获取的。
- 下面开始学习如何使用 caffe 做深度学习的研究喽, 祝大家学习愉快。。。



## 1.2 PCL

### 1.2.1 PCL 系列——三维重构之泊松重构

#### PCL 系列

##### 说明

通过本教程，我们将会学会：

- 如果通过泊松算法进行三维点云重构。
- 程序支持两种文件格式：\*.pcd 和 \*.ply
- 程序先读取点云文件，然后计算法向量，接着使用泊松算法进行重构，最后显示结果。

##### 操作

- 在 VS2010 中新建一个文件 recon\_poisson.cpp，然后将下面的代码复制到文件中。
- 参照之前的文章，配置项目的属性。设置包含目录和库目录和附加依赖项。

```
//点的类型的头文件
#include <pcl/point_types.h>
//点云文件 IO (pcd 文件和 ply 文件)
#include <pcl/io/pcd_io.h>
#include <pcl/io/ply_io.h>
//kd 树
#include <pcl/kdtree/kdtree_flann.h>
//特征提取
#include <pcl/features/normal_3d_omp.h>
#include <pcl/features/normal_3d.h>
//重构
#include <pcl/surface/gp3.h>
#include <pcl/surface/poisson.h>
//可视化
#include <pcl/visualization/pcl_visualizer.h>
//多线程
#include <boost/thread/thread.hpp>
#include <fstream>
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <string>

int main(int argc, char** argv)
{
    // 确定文件格式
    char tmpStr[100];
    strcpy(tmpStr, argv[1]);
    char* pext = strchr(tmpStr, '.');
    std::string extply("ply");
    std::string extpcd("pcd");
    if(pext){
        *pext='\0';
        pext++;
    }
}
```

(下页继续)

(续上页)

```

    }
    std::string ext(pext);
    //如果不支持文件格式, 退出程序
    if (!(ext == extply) || (ext == extpcd)){
        std::cout << "文件格式不支持!" << std::endl;
        std::cout << "支持文件格式: *.pcd 和 *.ply! " << std::endl;
        return(-1);
    }

    //根据文件格式选择输入方式
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>) ; //创建点云对象指针, 用于存储输入
    if (ext == extply){
        if (pcl::io::loadPLYFile(argv[1] , *cloud) == -1){
            PCL_ERROR("Could not read ply file!\n") ;
            return -1;
        }
    }
    else{
        if (pcl::io::loadPCDFile(argv[1] , *cloud) == -1){
            PCL_ERROR("Could not read pcd file!\n") ;
            return -1;
        }
    }

    // 计算法向量
    pcl::PointCloud<pcl::PointNormal>::Ptr cloud_with_normals(new pcl::PointCloud
    ↪<pcl::PointNormal>); //法向量点云对象指针
    pcl::NormalEstimation<pcl::PointXYZ , pcl::Normal> n ; //法线估计对象
    pcl::PointCloud<pcl::Normal>::Ptr normals(new pcl::PointCloud<pcl::Normal>) ; //存储估计的法线的指针
    pcl::search::KdTree<pcl::PointXYZ>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZ>
    ↪) ;
    tree->setInputCloud(cloud) ;
    n.setInputCloud(cloud) ;
    n.setSearchMethod(tree) ;
    n.setKSearch(20);
    n.compute(*normals); //计算法线, 结果存储在 normals 中

    //将点云和法线放到一起
    pcl::concatenateFields(*cloud , *normals , *cloud_with_normals) ;

    //创建搜索树
    pcl::search::KdTree<pcl::PointNormal>::Ptr tree2(new pcl::search::KdTree
    ↪<pcl::PointNormal>) ;
    tree2->setInputCloud(cloud_with_normals) ;
    //创建 Poisson 对象, 并设置参数
    pcl::Poisson<pcl::PointNormal> pn ;
    pn.setConfidence(false); //是否使用法向量的大小作为置信信息。如果 false, 所有法向量均归一化。
    pn.setDegree(2); //设置参数 degree[1,5], 值越大越精细, 耗时越久。
    pn.setDepth(8); //树的最大深度, 求解  $2^d \times 2^d \times 2^d$  立方体元。由于八叉树自适应采样密度, 指定值仅为最大深度。
    pn.setIsoDivide(8); //用于提取 ISO 等值面的算法的深度
    pn.setManifold(false); //是否添加多边形的重心, 当多边形三角化时。设置流行标志, 如果设置为 true, 则对多边形进行细分三角化时添加重心, 设置 false 则不添加

```

(下页继续)

(续上页)

```

pn.setOutputPolygons(false); //是否输出多边形网格（而不是三角化移动立方体的结果）
pn.setSamplesPerNode(3.0); //设置落入一个八叉树结点中的样本点的最小数量。无噪声, [1.0-
↪5.0], 有噪声 [15.-20.] 平滑
pn.setScale(1.25); //设置用于重构的立方体直径和样本边界立方体直径的比率。
pn.setSolverDivide(8); //设置求解线性方程组的 Gauss-Seidel 迭代方法的深度
//pn.setIndices();

//设置搜索方法和输入点云
pn.setSearchMethod(tree2);
pn.setInputCloud(cloud_with_normals);
//创建多变形网格, 用于存储结果
pcl::PolygonMesh mesh;
//执行重构
pn.performReconstruction(mesh);

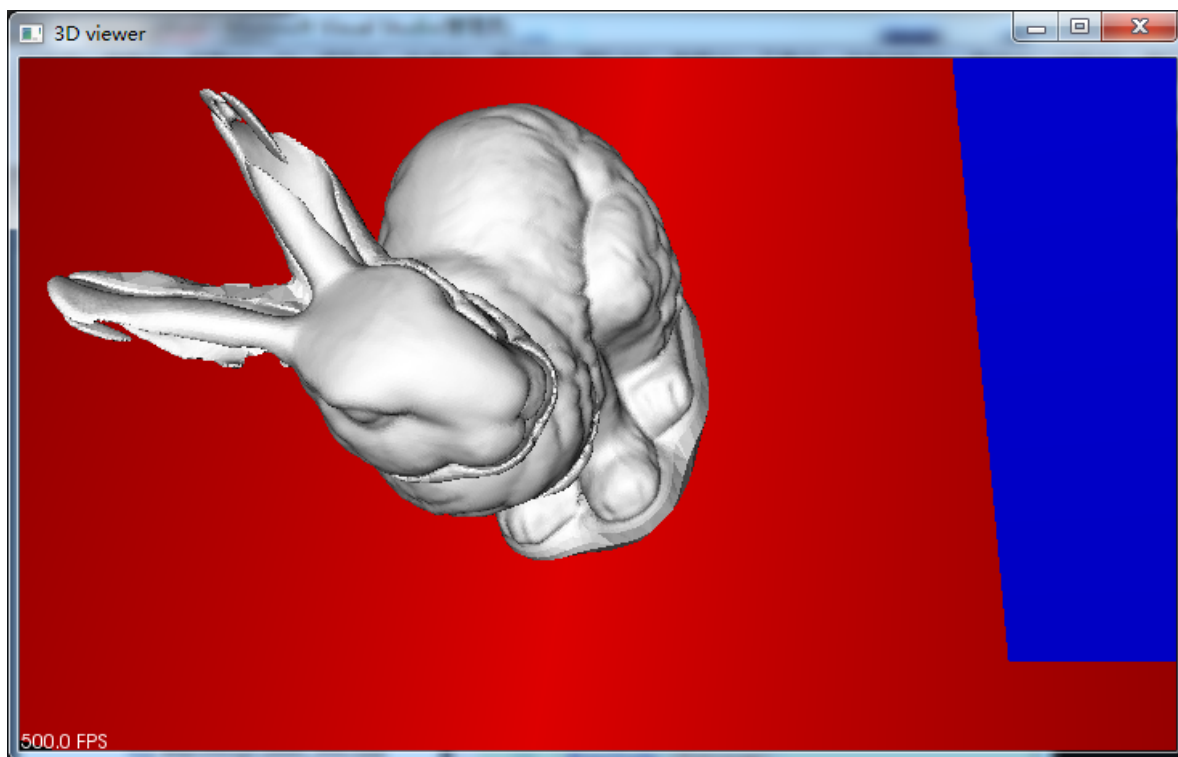
//保存网格图
pcl::io::savePLYFile("result.ply", mesh);

// 显示结果图
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer(new_
↪pcl::visualization::PCLVisualizer("3D viewer"));
viewer->setBackgroundColor(0, 0, 0);
viewer->addPolygonMesh(mesh, "my");
viewer->addCoordinateSystem(50.0);
viewer->initCameraParameters();
while (!viewer->wasStopped()) {
    viewer->spinOnce(100);
    boost::this_thread::sleep(boost::posix_time::microseconds(100000));
}

return 0;
}

```

- 重新生成项目。
- 到改项目的 Debug 目录下, 按住 Shift, 同时点击鼠标右键, 在当前窗口打开 CMD 窗口。
- 在命令行中输入 recon\_poisson.exe bunny.points.ply, 执行程序。得到如下图所示的结果。



## 1.2.2 PCL 系列——三维重构之移动立方体算法

### PCL 系列

#### 说明

通过本教程，我们将会学会：

- 如果通过移动立方体算法进行三维点云重构。
- 程序支持两种文件格式：\*.pcd 和 \*.ply。
- 程序先读取点云文件；然后计算法向量，并将法向量和点云坐标放在一起；接着使用移动立方体算法进行重构，最后显示结果。

#### 操作

- 在 VS2010 中新建一个文件 recon\_marchingCubes.cpp，然后将下面的代码复制到文件中。
- 参照之前的文章，配置项目的属性。设置包含目录和库目录和附加依赖项。

```
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/io/ply_io.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/features/normal_3d.h>
#include <pcl/surface/marching_cubes_hoppe.h>
#include <pcl/surface/marching_cubes_rbf.h>
#include <pcl/surface/gp3.h>
```

(下页继续)

(续上页)

```

#include <pcl/visualization/pcl_visualizer.h>
#include <boost/thread/thread.hpp>
#include <fstream>
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <string>

int main (int argc, char** argv)
{
    // 确定文件格式
    char tmpStr[100];
    strcpy(tmpStr, argv[1]);
    char* pext = strrchr(tmpStr, '.');
    std::string extply("ply");
    std::string extpcd("pcd");
    if(pext){
        *pext='\0';
        pext++;
    }
    std::string ext(pext);
    //如果不支持文件格式, 退出程序
    if (!((ext == extply)|| (ext == extpcd))){
        std::cout << "文件格式不支持!" << std::endl;
        std::cout << "支持文件格式: *.pcd 和 *.ply! " << std::endl;
        return (-1);
    }

    //根据文件格式选择输入方式
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>); //创建点云对象指针, 用于存储输入
    if (ext == extply){
        if (pcl::io::loadPLYFile(argv[1], *cloud) == -1){
            PCL_ERROR("Could not read ply file!\n");
            return -1;
        }
    }
    else{
        if (pcl::io::loadPCDFile(argv[1], *cloud) == -1){
            PCL_ERROR("Could not read pcd file!\n");
            return -1;
        }
    }

    // 估计法向量
    pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> n;
    pcl::PointCloud<pcl::Normal>::Ptr normals (new pcl::PointCloud<pcl::Normal>);
    pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ>);
    tree->setInputCloud(cloud);
    n.setInputCloud(cloud);
    n.setSearchMethod(tree);
    n.setKSearch(20);
    n.compute (*normals); //计算法线, 结果存储在 normals 中
    /* normals 不能同时包含点的法向量和表面的曲率

```

(下页继续)

(续上页)

```

//将点云和法线放到一起
pcl::PointCloud<pcl::PointNormal>::Ptr cloud_with_normals (new pcl::PointCloud
-><pcl::PointNormal>);
pcl::concatenateFields (*cloud, *normals, *cloud_with_normals);
/* cloud_with_normals = cloud + normals

//创建搜索树
pcl::search::KdTree<pcl::PointNormal>::Ptr tree2 (new pcl::search::KdTree
-><pcl::PointNormal>);
tree2->setInputCloud (cloud_with_normals);

//初始化 MarchingCubes 对象, 并设置参数
pcl::MarchingCubes<pcl::PointNormal> *mc;
mc = new pcl::MarchingCubesHoppe<pcl::PointNormal> ();
/*
if (hoppe_or_rbf == 0)
    mc = new pcl::MarchingCubesHoppe<pcl::PointNormal> ();
else
{
    mc = new pcl::MarchingCubesRBF<pcl::PointNormal> ();
    (reinterpret_cast<pcl::MarchingCubesRBF<pcl::PointNormal>*> (mc))->
->setOffSurfaceDisplacement (off_surface_displacement);
}
*/

//创建多变形网格, 用于存储结果
pcl::PolygonMesh mesh;

//设置 MarchingCubes 对象的参数
mc->setIsoLevel (0.0f);
mc->setGridResolution (50, 50, 50);
mc->setPercentageExtendGrid (0.0f);

//设置搜索方法
mc->setInputCloud (cloud_with_normals);

//执行重构, 结果保存在 mesh 中
mc->reconstruct (mesh);

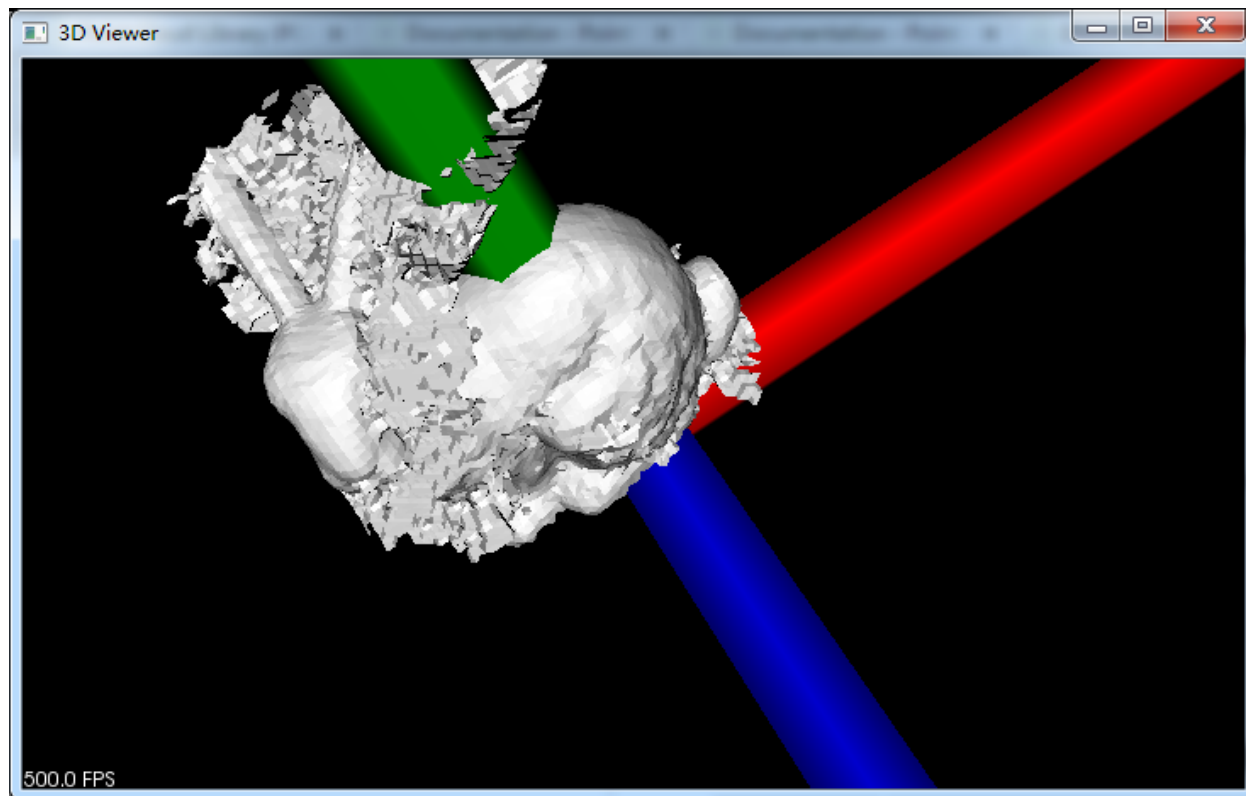
//保存网格图
pcl::io::savePLYFile("result.ply", mesh);

// 显示结果图
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
->pcl::visualization::PCLVisualizer ("3D Viewer"));
viewer->setBackgroundColor (0, 0, 0); //设置背景
viewer->addPolygonMesh(mesh, "my"); //设置显示的网格
viewer->addCoordinateSystem (1.0); //设置坐标系
viewer->initCameraParameters ();
while (!viewer->wasStopped ()) {
    viewer->spinOnce (100);
    boost::this_thread::sleep (boost::posix_time::microseconds (100000));
}

return (0);
}

```

- 重新生成项目。
- 到改项目的 Debug 目录下，按住 Shift，同时点击鼠标右键，在当前窗口打开 CMD 窗口。
- 在命令行中输入 `recon_marchingCubes.exe bunny.points.ply`，执行程序。得到如下图所示的结果。



### 1.2.3 PCL 系列——三维重构之贪婪三角投影算法

#### PCL 系列

##### 说明

通过本教程，我们将会学会：

- 如果通过贪婪三角投影算法进行三维点云重构。
- 程序支持两种文件格式：`*.pcd` 和 `*.ply`。
- 程序先读取点云文件；然后计算法向量，并将法向量和点云坐标放在一起；接着使用贪婪三角投影算法进行重构，最后显示结果。

## 操作

- 在 VS2010 中新建一个文件 recon\_greedyProjection.cpp, 然后将下面的代码复制到文件中。
- 参照之前的文章, 配置项目的属性。设置包含目录和库目录和附加依赖项。

```

/*
 * GreedyProjection 是根据点云进行三角化, 而 poisson 则是对 water-tight 的模型进行重建,
 * 所以形成了封闭 mesh 和很多冗余信息, 需要对 poisson 的重建进行修剪才能得到相对正确的模型
 */

#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/io/ply_io.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/features/normal_3d.h>
#include <pcl/surface/gp3.h>
#include <pcl/visualization/pcl_visualizer.h>
#include <boost/thread/thread.hpp>
#include <fstream>
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <string>

int main (int argc, char** argv)
{
    // 确定文件格式
    char tmpStr[100];
    strcpy(tmpStr, argv[1]);
    char* pext = strchr(tmpStr, '.');
    std::string extply("ply");
    std::string extpcd("pcd");
    if(pext){
        *pext='\0';
        pext++;
    }
    std::string ext(pext);
    //如果不支持文件格式, 退出程序
    if (!(ext == extply) || (ext == extpcd)){
        std::cout << "文件格式不支持!" << std::endl;
        std::cout << "支持文件格式: *.pcd 和 *.ply! " << std::endl;
        return(-1);
    }

    //根据文件格式选择输入方式
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>); //创建点云对象指针, 用于存储输入
    if (ext == extply){
        if (pcl::io::loadPLYFile(argv[1], *cloud) == -1){
            PCL_ERROR("Could not read ply file!\n");
            return -1;
        }
    }
    else{
        if (pcl::io::loadPCDFile(argv[1], *cloud) == -1){
            PCL_ERROR("Could not read pcd file!\n");

```

(下页继续)



(续上页)

```

        return -1;
    }
}

// 估计法向量
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> n;
pcl::PointCloud<pcl::Normal>::Ptr normals (new pcl::PointCloud<pcl::Normal>);
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ>
↪);
tree->setInputCloud (cloud);
n.setInputCloud (cloud);
n.setSearchMethod (tree);
n.setKSearch (20);
n.compute (*normals); //计算法线, 结果存储在 normals 中
/* normals 不能同时包含点的法向量和表面的曲率

//将点云和法线放到一起
pcl::PointCloud<pcl::PointNormal>::Ptr cloud_with_normals (new pcl::PointCloud
↪<pcl::PointNormal>);
pcl::concatenateFields (*cloud, *normals, *cloud_with_normals);
/* cloud_with_normals = cloud + normals

//创建搜索树
pcl::search::KdTree<pcl::PointNormal>::Ptr tree2 (new pcl::search::KdTree
↪<pcl::PointNormal>);
tree2->setInputCloud (cloud_with_normals);

//初始化 GreedyProjectionTriangulation 对象, 并设置参数
pcl::GreedyProjectionTriangulation<pcl::PointNormal> gp3;
    //创建多变形网格, 用于存储结果
pcl::PolygonMesh triangles;

//设置 GreedyProjectionTriangulation 对象的参数
    //第一个参数影响很大
gp3.setSearchRadius (1.5f); //设置连接点之间的最大距离 (最大边长) 用于确定 k 近邻的球半径 【默认
值 0】
gp3.setMu (2.5f); //设置最近邻距离的乘子, 以得到每个点的最终搜索半径 【默认值 0】
gp3.setMaximumNearestNeighbors (100); //设置搜索的最近邻点的最大数量
gp3.setMaximumSurfaceAngle(M_PI/4); // 45 degrees (pi) 最大平面角
gp3.setMinimumAngle(M_PI/18); // 10 degrees 每个三角的最小角度
gp3.setMaximumAngle(2*M_PI/3); // 120 degrees 每个三角的最大角度
gp3.setNormalConsistency(false); //如果法向量一致, 设置为 true

//设置搜索方法和输入点云
gp3.setInputCloud(cloud_with_normals);
gp3.setSearchMethod(tree2);

    //执行重构, 结果保存在 triangles 中
gp3.reconstruct (triangles);

    //保存网格图
pcl::io::savePLYFile("result.ply", triangles);

// Additional vertex information
//std::vector<int> parts = gp3.getPartIDs();
//std::vector<int> states = gp3.getPointStates();

```

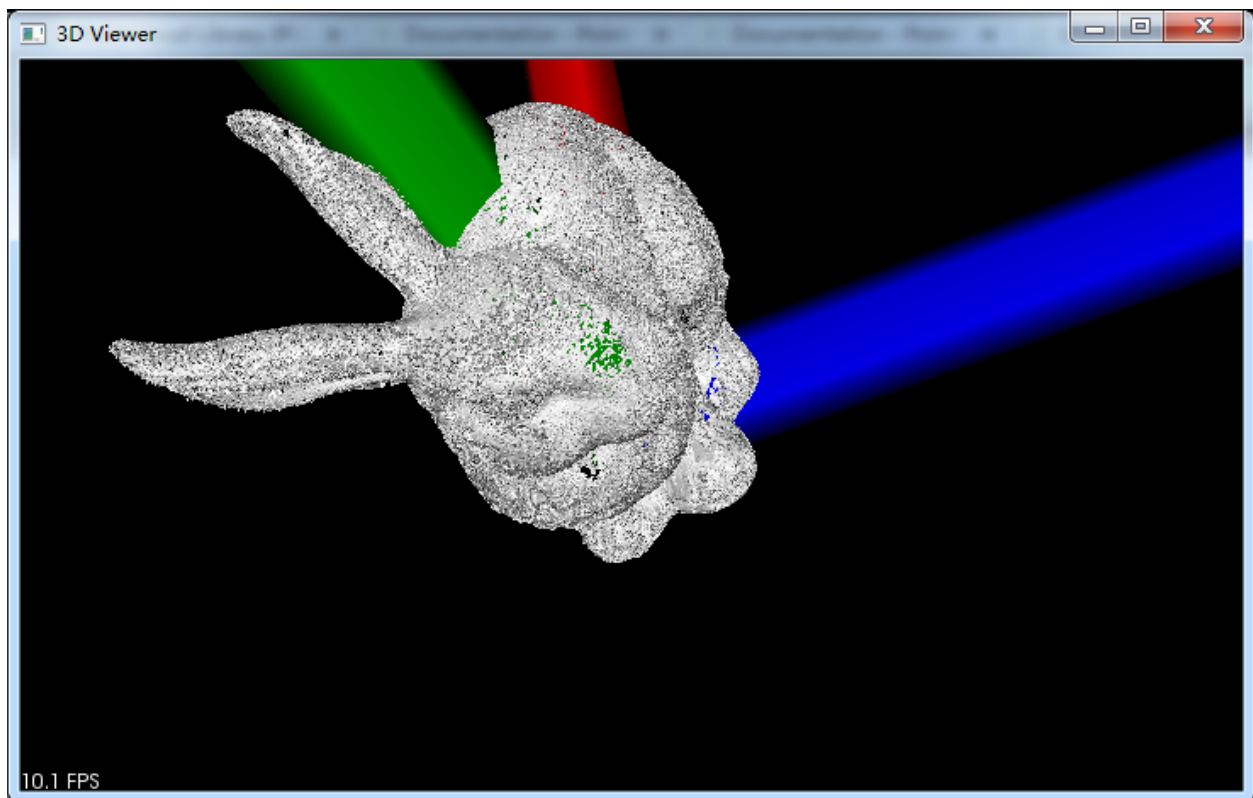
(下页继续)

(续上页)

```
// 显示结果图
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));
viewer->setBackgroundColor (0, 0, 0); //设置背景
viewer->addPolygonMesh(triangles,"my"); //设置显示的网格
viewer->addCoordinateSystem (1.0); //设置坐标系
viewer->initCameraParameters ();
while (!viewer->wasStopped ()) {
    viewer->spinOnce (100);
    boost::this_thread::sleep (boost::posix_time::microseconds (100000));
}

return (0);
}
```

- 重新生成项目。
- 到改项目的 Debug 目录下，按住 Shift，同时点击鼠标右键，在当前窗口打开 CMD 窗口。
- 在命令中输入 recon\_greedyProjection.exe bunny.points.ply，执行程序。得到如下图所示的结果。



## 1.2.4 PCL 系列——从深度图像 (RangeImage) 中提取 NARF 关键点

### PCL 系列

#### 说明

- 通过本教程，我们将会学会：如何从深度图像中提取 NARF 关键点。
- 首先从硬盘中读取点云文件，然后提取它的 NARF 关键点，最后显示结果。
- 下面的代码中，先是命令行解析，然后是读取点云文件，如果点云文件不存在，就创建一个深度图像，并显示它。

#### 操作

- 参照之前的文章，配置项目的属性。设置包含目录和库目录和附加依赖项。
- 在 VS2010 中新建一个文件 narf\_keypoint\_extraction.cpp，然后将下面的代码复制到文件中。

```
#include <iostream> //标准输入输出流
#include <boost/thread/thread.hpp>
#include <pcl/range_image/range_image.h>
#include <pcl/io/pcd_io.h> //PCL 的 PCD 格式文件的输入输出头文件
#include <pcl/visualization/range_image_visualizer.h>
#include <pcl/visualization/pcl_visualizer.h>
#include <pcl/features/range_image_border_extractor.h>
#include <pcl/keypoints/narf_keypoint.h>
#include <pcl/console/parse.h>

typedef pcl::PointXYZ PointType; //定义别名

//参数 全局变量
float angular_resolution = 0.5f; //角坐标分辨率
float support_size = 0.2f; //感兴趣点的尺寸 (球面的直径)
pcl::RangeImage::CoordinateFrame coordinate_frame = pcl::RangeImage::CAMERA_FRAME; //坐标框架: 相机框架 (而不是激光框架)
bool setUnseenToMaxRange = false; //是否将所有不可见的点 看作 最大距离

//帮助
//当用户输入命令行参数-h, 打印帮助信息
void printUsage (const char* progName)
{
    std::cout << "\n\nUsage: "<<progName<<" [options] <scene.pcd>\n\n"
        << "Options:\n"
        << "-----\n"
        << "-r <float>    angular resolution in degrees (default "<<angular_
->resolution<<")\n"
        << "-c <int>      coordinate frame (default "<< (int)coordinate_frame<<")\n"
->"
        << "-m          Treat all unseen points as maximum range readings\n"
        << "-s <float>    support size for the interest points (diameter of the_
->used sphere - "
        << "default "<
-><support_size<<")\n"
        << "-h          this help\n"
        << "\n\n";
```

(下页继续)

(续上页)

```

}

int main (int argc, char** argv)
{
    //解析 命令行 参数
    if (pcl::console::find_argument (argc, argv, "-h") >= 0)
    {
        printUsage (argv[0]);
        return 0;
    }
    if (pcl::console::find_argument (argc, argv, "-m") >= 0)
    {
        setUnseenToMaxRange = true;
        cout << "Setting unseen values in range image to maximum range readings.\n";
    }
    int tmp_coordinate_frame;
    if (pcl::console::parse (argc, argv, "-c", tmp_coordinate_frame) >= 0)
    {
        coordinate_frame = pcl::RangeImage::CoordinateFrame (tmp_coordinate_frame); //以函
        数的方式初始化 (0: 相机框架; 1: 激光框架)
        cout << "Using coordinate frame "<< (int)coordinate_frame<<".\n";
    }
    if (pcl::console::parse (argc, argv, "-s", support_size) >= 0)
        cout << "Setting support size to "<<support_size<<".\n";
    if (pcl::console::parse (argc, argv, "-r", angular_resolution) >= 0)
        cout << "Setting angular resolution to "<<angular_resolution<<"deg.\n";
    angular_resolution = pcl::deg2rad (angular_resolution);

    //读取 pcd 文件; 如果没有指定文件, 就创建样本点
    pcl::PointCloud<PointType>::Ptr point_cloud_ptr (new pcl::PointCloud<PointType>); //
    ↪点云指针
    pcl::PointCloud<PointType>& point_cloud = *point_cloud_ptr; //上面点云的别名
    pcl::PointCloud<pcl::PointWithViewpoint> far_ranges; //带视角的点构成的点云
    Eigen::Affine3f scene_sensor_pose (Eigen::Affine3f::Identity ()); //仿射变换
    std::vector<int> pcd_filename_indices = pcl::console::parse_file_extension_argument_
    ↪(argc, argv, "pcd"); //检查参数中是否有 pcd 格式文件名, 返回参数向量中的索引号
    if (!pcd_filename_indices.empty ()) //如果指定了 pcd 文件, 读取 pcd 文件和对应的远距离 pcd
    文件
    {
        std::string filename = argv[pcd_filename_indices[0]]; //文件名
        if (pcl::io::loadPCDFile (filename, point_cloud) == -1) //读取 pcd 文件
        {
            cerr << "Was not able to open file \""<<filename<<"\".\n"; //是否应该是 std::cerr
            printUsage (argv[0]);
            return 0;
        }
        scene_sensor_pose = Eigen::Affine3f (Eigen::Translation3f (point_cloud.sensor_
        ↪origin_[0],
                                                                    point_cloud.sensor_
        ↪origin_[1],
                                                                    point_cloud.sensor_
        ↪origin_[2])) *
        Eigen::Affine3f (point_cloud.sensor_orientation_); //设置传感器
        的姿势
    }
}

```

(下页继续)

(续上页)

```

std::string far_ranges_filename = pcl::getFilenameWithoutExtension (filename)+"_
→far_ranges.pcd"; //远距离文件名
if (pcl::io::loadPCDFile (far_ranges_filename.c_str (), far_ranges) == -1) //读取远
距离 pcd 文件
    std::cout << "Far ranges file \""<<far_ranges_filename<<" does not exists.\n";
}
else //没有指定 pcd 文件, 生成点云, 并填充它
{
    setUnseenToMaxRange = true;
    cout << "\nNo *.pcd file given => Genarating example point cloud.\n\n";
    for (float x=-0.5f; x<=0.5f; x+=0.01f)
    {
        for (float y=-0.5f; y<=0.5f; y+=0.01f)
        {
            PointType point;
            point.x = x; point.y = y; point.z = 2.0f - y;
            point_cloud.points.push_back (point); //设置点云中点的坐标
        }
    }
    point_cloud.width = (int) point_cloud.points.size ();
    point_cloud.height = 1;
}

//从点云数据, 创建深度图像
float noise_level = 0.0;
float min_range = 0.0f;
int border_size = 1;
boost::shared_ptr<pcl::RangeImage> range_image_ptr (new pcl::RangeImage); //创建
RangeImage 对象 (指针)
pcl::RangeImage& range_image = *range_image_ptr; //引用
range_image.createFromPointCloud (point_cloud, angular_resolution, pcl::deg2rad_
→(360.0f), pcl::deg2rad (180.0f),
                                scene_sensor_pose, coordinate_frame, noise_level,
→min_range, border_size); //从点云创建深度图像
range_image.integrateFarRanges (far_ranges); //整合远距离点云
if (setUnseenToMaxRange)
    range_image.setUnseenToMaxRange ();

//打开 3D 观察图形窗口, 并添加点云
pcl::visualization::PCLVisualizer viewer ("3D Viewer"); //创建 3D Viewer 对象
viewer.setBackgroundColor (1, 1, 1); //设置背景色
pcl::visualization::PointCloudColorHandlerCustom<pcl::PointWithRange> range_image_
→color_handler (range_image_ptr, 0, 0, 0);
viewer.addPointCloud (range_image_ptr, range_image_color_handler, "range image"); //
→添加点云
viewer.setPointCloudRenderingProperties (pcl::visualization::PCL_VISUALIZER_POINT_
→SIZE, 1, "range image");
//viewer.addCoordinateSystem (1.0f, "global");
//PointCloudColorHandlerCustom<PointType> point_cloud_color_handler (point_cloud_
→ptr, 150, 150, 150);
//viewer.addPointCloud (point_cloud_ptr, point_cloud_color_handler, "original point_
→cloud");
viewer.initCameraParameters ();
//setViewerPose (viewer, range_image.getTransformationToWorldSystem ());

```

(下页继续)

(续上页)

```
//显示深度图像 (平面图, 上面的 3D 显示)
pcl::visualization::RangeImageVisualizer range_image_widget ("Range image");
range_image_widget.showRangeImage (range_image);

//提取 NARF 关键点
pcl::RangeImageBorderExtractor range_image_border_extractor; //创建深度图像的边界提取器,
用于提取 NARF 关键点
pcl::NarfKeypoint narf_keypoint_detector (&range_image_border_extractor); //创建 NARF
对象
narf_keypoint_detector.setRangeImage (&range_image);
narf_keypoint_detector.getParameters ().support_size = support_size;
//narf_keypoint_detector.getParameters ().add_points_on_straight_edges = true;
//narf_keypoint_detector.getParameters ().distance_for_additional_points = 0.5;

pcl::PointCloud<int> keypoint_indices; //用于存储关键点的索引
narf_keypoint_detector.compute (keypoint_indices); //计算 NARF 关键点
std::cout << "Found "<<keypoint_indices.points.size ()<<" key points.\n";

//在 range_image_widget 中显示关键点
//for (size_t i=0; i<keypoint_indices.points.size (); ++i)
//    range_image_widget.markPoint (keypoint_indices.points[i]%range_image.width,
//                                  //keypoint_indices.points[i]/range_image.width);

//在 3D 图形窗口中显示关键点
pcl::PointCloud<pcl::PointXYZ>::Ptr keypoints_ptr (new pcl::PointCloud
-><pcl::PointXYZ>); //创建关键点指针
pcl::PointCloud<pcl::PointXYZ>& keypoints = *keypoints_ptr; //引用
keypoints.points.resize (keypoint_indices.points.size ()); //点云变形, 无序
for (size_t i=0; i<keypoint_indices.points.size (); ++i)
    keypoints.points[i].getVector3fMap () = range_image.points[keypoint_indices.
->points[i]].getVector3fMap ();

pcl::visualization::PointCloudColorHandlerCustom<pcl::PointXYZ> keypoints_color_
->handler (keypoints_ptr, 0, 255, 0);
viewer.addPointCloud<pcl::PointXYZ> (keypoints_ptr, keypoints_color_handler,
->"keypoints");
viewer.setPointCloudRenderingProperties (pcl::visualization::PCL_VISUALIZER_POINT_
->SIZE, 7, "keypoints");

// -----Main loop-----
while (!viewer.wasStopped ())
{
    range_image_widget.spinOnce (); // 处理 GUI 事件
    viewer.spinOnce ();
    pcl_sleep(0.01);
}
}
```

- 重新生成项目。
- 到改项目的 Debug 目录下, 按住 Shift, 同时点击鼠标右键, 在当前窗口打开 CMD 窗口。
- 在命令行中输入 narf\_keypoint\_extraction.exe 执行程序, narf\_keypoint\_extraction.exe -h 显示帮助信息。

## 参考

### 1.2.5 PCL 系列——如何使用迭代最近点法 (ICP) 配准

#### PCL 系列

#### 说明

通过本教程，我们将会学会：

- 如何使用迭代最近点法 (Iterative Closest Point) 判断一个点云是否是另一个点云的刚体变换
- 使用的方法是：最小化两个点云中对应点之间的距离，并刚性变换他们。
- 代码说明：先生成输入点云（待变换的点云）和目标点云（变换参照），然后创建 ICP 对象，设置该对象的输入点云和目标点云，然后进行配准，并显示 ICP 配准信息和变换矩阵。

#### 操作

- 在 VS2010 中新建一个文件 iterative\_closest\_point.cpp，然后将下面的代码复制到文件中。
- 参照之前的文章，配置项目的属性。设置包含目录和库目录和附加依赖项。

```
#include <iostream> //标准输入/输出
#include <pcl/io/pcd_io.h> //pcd 文件输入/输出
#include <pcl/point_types.h> //各种点类型
#include <pcl/registration/icp.h> //ICP (iterative closest point) 配准

int main (int argc, char** argv)
{
    //创建点云指针
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_in (new pcl::PointCloud<pcl::PointXYZ>); //
    ↪//创建输入点云 (指针)
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_out (new pcl::PointCloud<pcl::PointXYZ>); ↪
    ↪//创建输出/目标点云 (指针)

    //生成并填充点云 cloud_in
    cloud_in->width = 5;
    cloud_in->height = 1;
    cloud_in->is_dense = false;
    cloud_in->points.resize (cloud_in->width * cloud_in->height); //变形，无序
    for (size_t i = 0; i < cloud_in->points.size (); ++i) //随机数初始化点的坐标
    {
        cloud_in->points[i].x = 1024 * rand () / (RAND_MAX + 1.0f);
        cloud_in->points[i].y = 1024 * rand () / (RAND_MAX + 1.0f);
        cloud_in->points[i].z = 1024 * rand () / (RAND_MAX + 1.0f);
    }
    std::cout << "Saved " << cloud_in->points.size () << " data points to input:"
        << std::endl;
    //打印点云 cloud_in 中所有点的坐标信息
    for (size_t i = 0; i < cloud_in->points.size (); ++i) std::cout << "      " <<
        cloud_in->points[i].x << " " << cloud_in->points[i].y << " " <<
        cloud_in->points[i].z << std::endl;

    // 填充点云 cloud_out
    *cloud_out = *cloud_in; //初始化 cloud_out
    std::cout << "size:" << cloud_out->points.size() << std::endl;
```

(下页继续)

(续上页)

```

for (size_t i = 0; i < cloud_in->points.size (); ++i)
    cloud_out->points[i].x = cloud_in->points[i].x + 0.7f; //平移 cloud_in 得到 cloud_out
std::cout << "Transformed " << cloud_in->points.size () << " data points:"
    << std::endl;
//打印点云 cloud_out 中所有点的坐标信息 (每一行对应一个点的 xyz 坐标)
for (size_t i = 0; i < cloud_out->points.size (); ++i)
    std::cout << "      " << cloud_out->points[i].x << " " <<
        cloud_out->points[i].y << " " << cloud_out->points[i].z << std::endl;
//*****
// ICP 配准
//*****
pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp; //创建 ICP 对象, 用于 ICP
配准
icp.setInputCloud(cloud_in); //设置输入点云
icp.setInputTarget(cloud_out); //设置目标点云 (输入点云进行仿射变换, 得到目标点云)
pcl::PointCloud<pcl::PointXYZ> Final; //存储结果
//进行配准, 结果存储在 Final 中
icp.align(Final);
//输出 ICP 配准的信息 (是否收敛, 拟合度)
std::cout << "has converged:" << icp.hasConverged() << " score: " <<
icp.getFitnessScore() << std::endl;
//输出最终的变换矩阵 (4x4)
std::cout << icp.getFinalTransformation() << std::endl;

return (0);
}

```

- 重新生成项目。
- 到改项目的 Debug 目录下, 按住 Shift, 同时点击鼠标右键, 在当前窗口打开 CMD 窗口。
- 在命令行中输入 iterative\_closest\_point.exe, 可以得到下图所示结果。

```

C:\Windows\system32\cmd.exe
Saved 5 data points to input:
1.28125 577.094 197.938
828.125 599.031 491.375
358.688 917.438 842.563
764.5 178.281 879.531
727.531 525.844 311.281
size:5
Transformed 5 data points:
1.98125 577.094 197.938
828.825 599.031 491.375
359.388 917.438 842.563
765.2 178.281 879.531
728.231 525.844 311.281
has converged:1 score: 1.90389e-007
1 3.87775e-007 -1.50904e-007 0.700134
7.59784e-007 1 -3.57652e-007 -0.00079298
-9.58419e-007 -4.69054e-007 1 0.000304581
0 0 0 1
请按任意键继续. . .

```



## 参考

### 1.2.6 PCL 系列——如何可视化深度图像

#### PCL 系列

#### 说明

通过本教程，我们将会学会：

- 如何通过两种方式可视化深度图像。
- 一种方式是在 3D viewer 中以点云的方式显示。（深度图来源于点云图）
- 一种方式是作为一幅图像显示（以不同的颜色表示不同的深度值）

#### 操作

- 在 VS2010 中新建一个文件 range\_image\_visualization.cpp，然后将下面的代码复制到文件中。
- 参照之前的文章，配置项目的属性。设置包含目录和库目录和附加依赖项。

```
#include <iostream> //标准输入/输出
#include <boost/thread/thread.hpp> //多线程
#include <pcl/common/common_headers.h>
#include <pcl/range_image/range_image.h> //深度图有关头文件
#include <pcl/io/pcd_io.h> //pcd 文件输入/输出
#include <pcl/visualization/range_image_visualizer.h> //深度图可视化
#include <pcl/visualization/pcl_visualizer.h>
#include <pcl/console/parse.h> //命令行参数解析

typedef pcl::PointXYZ PointType;

//参数 全局
float angular_resolution_x = 0.5f, //角分辨率（单位弧度）
      angular_resolution_y = angular_resolution_x;
pcl::RangeImage::CoordinateFrame coordinate_frame = pcl::RangeImage::CAMERA_FRAME; //坐标帧（相机帧）
bool live_update = true; //是否根据选择的视角更新深度图像

// 打印帮助信息
void printUsage (const char* progName)
{
    std::cout << "\n\nUsage: "<<progName<<" [options] <scene.pcd>\n\n"
               << "Options:\n"
               << "-----\n"
               << "-rx <float>   angular resolution in degrees (default "<<angular_
->resolution_x<<")\n"
               << "-ry <float>   angular resolution in degrees (default "<<angular_
->resolution_y<<")\n"
               << "-c <int>     coordinate frame (default "<< (int)coordinate_frame<<")\n
->"
               << "-l           live update - update the range image according to the
->selected view in the 3D viewer.\n"
               << "-h           this help\n"
               << "\n\n";
```

(下页继续)

(续上页)

```

}

/*
void setViewerPose (pcl::visualization::PCLVisualizer& viewer, const Eigen::Affine3f&
↪viewer_pose)
{
    Eigen::Vector3f pos_vector = viewer_pose * Eigen::Vector3f(0, 0, 0);
    Eigen::Vector3f look_at_vector = viewer_pose.rotation () * Eigen::Vector3f(0, 0, 1)
↪+ pos_vector;
    Eigen::Vector3f up_vector = viewer_pose.rotation () * Eigen::Vector3f(0, -1, 0);
    viewer.setCameraPosition (pos_vector[0], pos_vector[1], pos_vector[2],
                            <!-- more -->

}
if (pcl::console::find_argument (argc, argv, "-l") >= 0)
{
    live_update = true;
    std::cout << "Live update is on.\n";
}
if (pcl::console::parse (argc, argv, "-rx", angular_resolution_x) >= 0)
    std::cout << "Setting angular resolution in x-direction to "<<angular_resolution_x
↪<<"deg.\n";
if (pcl::console::parse (argc, argv, "-ry", angular_resolution_y) >= 0)
    std::cout << "Setting angular resolution in y-direction to "<<angular_resolution_y
↪<<"deg.\n";
int tmp_coordinate_frame;
if (pcl::console::parse (argc, argv, "-c", tmp_coordinate_frame) >= 0)
{
    coordinate_frame = pcl::RangeImage::CoordinateFrame (tmp_coordinate_frame);
    std::cout << "Using coordinate frame "<< (int)coordinate_frame<<".\n";
}
angular_resolution_x = pcl::deg2rad (angular_resolution_x);
angular_resolution_y = pcl::deg2rad (angular_resolution_y);

//读取 pcd 文件。如果没有指定文件, 则创建样本云点
pcl::PointCloud<PointType>::Ptr point_cloud_ptr (new pcl::PointCloud<PointType>);
pcl::PointCloud<PointType>& point_cloud = *point_cloud_ptr;
Eigen::Affine3f scene_sensor_pose (Eigen::Affine3f::Identity ());
std::vector<int> pcd_filename_indices = pcl::console::parse_file_extension_argument
↪(argc, argv, "pcd");
if (!pcd_filename_indices.empty ())
{
    std::string filename = argv[pcd_filename_indices[0]];
    if (pcl::io::loadPCDFile (filename, point_cloud) == -1)
    {
        std::cout << "Was not able to open file \""<<filename<<"\".\n";
        printUsage (argv[0]);
        return 0;
    }
    scene_sensor_pose = Eigen::Affine3f (Eigen::Translation3f (point_cloud.sensor_
↪origin_[0],
                                                                    point_cloud.sensor_
↪origin_[1],
                                                                    point_cloud.sensor_
↪origin_[2])) *
                                Eigen::Affine3f (point_cloud.sensor_orientation_);
}

```

(下页继续)

(续上页)

```

else
{
    std::cout << "\nNo *.pcd file given => Generating example point cloud.\n\n";
    for (float x=-0.5f; x<=0.5f; x+=0.01f)
    {
        for (float y=-0.5f; y<=0.5f; y+=0.01f)
        {
            PointType point; point.x = x; point.y = y; point.z = 2.0f - y;
            point_cloud.points.push_back (point);
        }
    }
    point_cloud.width = (int) point_cloud.points.size (); point_cloud.height = 1;
}

//从点云创建出深度图
float noise_level = 0.0;
float min_range = 0.0f;
int border_size = 1;
boost::shared_ptr<pcl::RangeImage> range_image_ptr(new pcl::RangeImage); //深度图指针
pcl::RangeImage& range_image = *range_image_ptr; //引用
range_image.createFromPointCloud (point_cloud, angular_resolution_x, angular_
↪resolution_y,
                                pcl::deg2rad (360.0f), pcl::deg2rad (180.0f),
                                scene_sensor_pose, coordinate_frame, noise_level,
↪min_range, border_size); //从点云创建出深度图

//打开一个 3D 图形窗口, 并添加点云数据
pcl::visualization::PCLVisualizer viewer ("3D Viewer");
viewer.setBackgroundColor (1, 1, 1); //背景
pcl::visualization::PointCloudColorHandlerCustom<pcl::PointWithRange> range_image_
↪color_handler (range_image_ptr, 0, 0, 0);
viewer.addPointCloud (range_image_ptr, range_image_color_handler, "range image");
viewer.setPointCloudRenderingProperties (pcl::visualization::PCL_VISUALIZER_POINT_
↪SIZE, 1, "range image");
//viewer.addCoordinateSystem (1.0f, "global");
//PointCloudColorHandlerCustom<PointType> point_cloud_color_handler (point_cloud_
↪ptr, 150, 150, 150);
//viewer.addPointCloud (point_cloud_ptr, point_cloud_color_handler, "original point_
↪cloud");
viewer.initCameraParameters ();
//setViewerPose(viewer, range_image.getTransformationToWorldSystem ()); //PCL 1.6 出
错

//以图像的形式显示深度图像, 深度值作为颜色显示
pcl::visualization::RangeImageVisualizer range_image_widget ("Range image");
range_image_widget.showRangeImage (range_image);

//主循环
while (!viewer.wasStopped ())
{
    range_image_widget.spinOnce ();
    viewer.spinOnce ();
    pcl_sleep (0.01);

    if (live_update) //根据 3D 显示, 实时更新 2D 图像

```

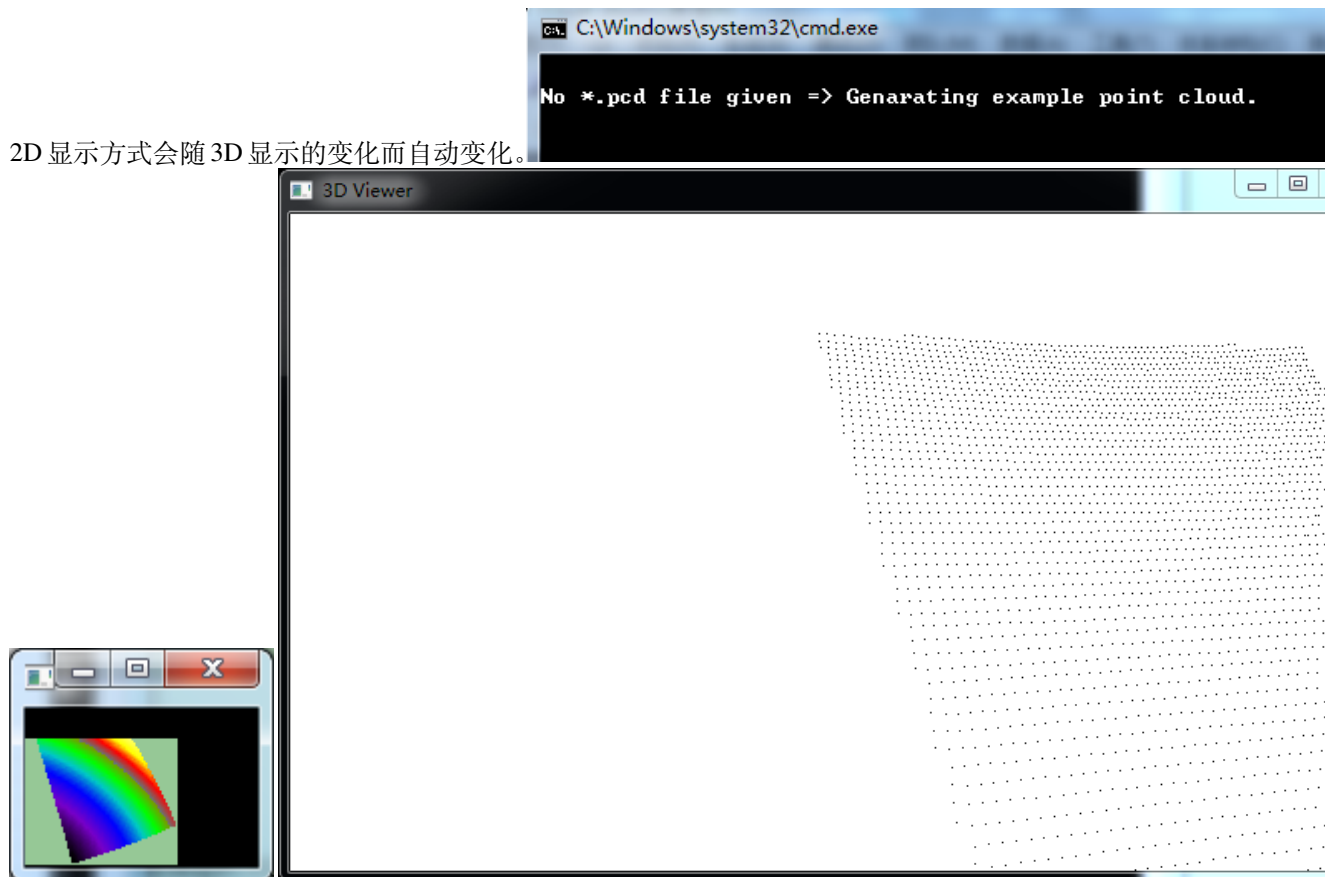
(下页继续)

(续上页)

```
{
    scene_sensor_pose = viewer.getViewerPose(); //获取观测姿势
    range_image.createFromPointCloud (point_cloud, angular_resolution_x, angular_
    ↪resolution_y,
                                pcl::deg2rad (360.0f), pcl::deg2rad (180.0f),
                                scene_sensor_pose, pcl::RangeImage::LASER_
    ↪FRAME, noise_level, min_range, border_size); //重新生成新的深度图
    range_image_widget.showRangeImage (range_image); //重新显示
}
}
```

- 重新生成项目。
- 到改项目的 Debug 目录下，按住 Shift，同时点击鼠标右键，在当前窗口打开 CMD 窗口。
- 在命令行中输入 range\_image\_visualization.exe，执行程序。结果如下图所示。
  - 图 1 是命令行的显示，因为没有指定 pcd 文件，程序生成了点云数据。
  - 图 2 是 2D 显示方式。
  - 图 3 是 3D 显示方式。

- 2D 显示方式会随 3D 显示的变化而自动变化。



## 参考

### 1.2.7 PCL 系列——如何逐渐地配准一对点云

#### PCL 系列

#### 说明

通过本教程，我们将会学会：

- 如何配准多个点云图。
- 配准的方法是：点云图两两配准，计算它们的变换矩阵，然后计算总的变换矩阵。
- 两个点云配准使用的是非线性 ICP 算法，它是 ICP 的算法的变体，使用 Levenberg-Marquardt 最优化。

#### 操作

- 在 VS2010 中新建一个文件 pairwise\_incremental\_registration.cpp，然后将下面的代码复制到文件中。
- 参照之前的文章，配置项目的属性。设置包含目录和库目录和附加依赖项。

```
#include <boost/make_shared.hpp> //共享指针
//点/点云
#include <pcl/point_types.h>
#include <pcl/point_cloud.h>
#include <pcl/point_representation.h>
//pcd 文件输入/输出
#include <pcl/io/pcd_io.h>
//滤波
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/filter.h>
//特征
#include <pcl/features/normal_3d.h>
//配准
#include <pcl/registration/icp.h> //ICP 方法
#include <pcl/registration/icp_nl.h>
#include <pcl/registration/transforms.h>
//可视化
#include <pcl/visualization/pcl_visualizer.h>

//命名空间
using pcl::visualization::PointCloudColorHandlerGenericField;
using pcl::visualization::PointCloudColorHandlerCustom;

//定义类型的别名
typedef pcl::PointXYZ PointT;
typedef pcl::PointCloud<PointT> PointCloud;
typedef pcl::PointNormal PointNormalT;
typedef pcl::PointCloud<PointNormalT> PointCloudWithNormals;

//全局变量
//可视化对象
pcl::visualization::PCLVisualizer *p;
//左视区和右视区，可视化窗口分成左右两部分
int vp_1, vp_2;
```

(下页继续)

(续上页)

```

//定义结构体, 用于处理点云
struct PCD
{
    PointCloud::Ptr cloud; //点云指针
    std::string f_name; //文件名
    //构造函数
    PCD() : cloud (new PointCloud) {}; //初始化
};

// 定义新的点表达方式< x, y, z, curvature > 坐标 + 曲率
class MyPointRepresentation : public pcl::PointRepresentation <PointNormalT> //继承关系
{
    using pcl::PointRepresentation<PointNormalT>::nr_dimensions_;
public:
    MyPointRepresentation ()
    {
        //指定维数
        nr_dimensions_ = 4;
    }

    //重载函数 copyToFloatArray, 以定义自己的特征向量
    virtual void copyToFloatArray (const PointNormalT &p, float * out) const
    {
        //< x, y, z, curvature > 坐标 xyz 和曲率
        out[0] = p.x;
        out[1] = p.y;
        out[2] = p.z;
        out[3] = p.curvature;
    }
};

//在窗口的左视区, 简单的显示源点云和目标点云
void showCloudsLeft(const PointCloud::Ptr cloud_target, const PointCloud::Ptr cloud_
↪source)
{
    p->removePointCloud ("vp1_target"); //根据给定的 ID, 从屏幕中去除一个点云。参数是 ID
    p->removePointCloud ("vp1_source"); //
    PointCloudColorHandlerCustom<PointT> tgt_h (cloud_target, 0, 255, 0); //目标点云绿色
    PointCloudColorHandlerCustom<PointT> src_h (cloud_source, 255, 0, 0); //源点云红色
    p->addPointCloud (cloud_target, tgt_h, "vp1_target", vp_1); //加载点云
    p->addPointCloud (cloud_source, src_h, "vp1_source", vp_1);
    PCL_INFO ("Press q to begin the registration.\n"); //在命令行中显示提示信息
    p-> spin();
}

//在窗口的右视区, 简单的显示源点云和目标点云
void showCloudsRight(const PointCloudWithNormals::Ptr cloud_target, const
↪PointCloudWithNormals::Ptr cloud_source)
{
    p->removePointCloud ("source"); //根据给定的 ID, 从屏幕中去除一个点云。参数是 ID
    p->removePointCloud ("target");
}

```

(下页继续)

(续上页)

```

PointCloudColorHandlerGenericField<PointNormalT> tgt_color_handler (cloud_target,
↪ "curvature"); //目标点云彩色句柄
if (!tgt_color_handler.isCapable ())
    PCL_WARN ("Cannot create curvature color handler!");
PointCloudColorHandlerGenericField<PointNormalT> src_color_handler (cloud_source,
↪ "curvature"); //源点云彩色句柄
if (!src_color_handler.isCapable ())
    PCL_WARN ("Cannot create curvature color handler!");
p->addPointCloud (cloud_target, tgt_color_handler, "target", vp_2); //加载点云
p->addPointCloud (cloud_source, src_color_handler, "source", vp_2);
p->spinOnce();
}

// 读取一系列的 PCD 文件 (希望配准的点云文件)
// 参数 argc 参数的数量 (来自 main())
// 参数 argv 参数的列表 (来自 main())
// 参数 models 点云数据集的结果向量
void loadData (int argc, char **argv, std::vector<PCD, Eigen::aligned_allocator<PCD> >
↪ &models)
{
    std::string extension (".pcd"); //声明并初始化 string 类型变量 extension, 表示文件后缀名
    // 通过遍历文件名, 读取 pcd 文件
    for (int i = 1; i < argc; i++) //遍历所有的文件名 (略过程序名)
    {
        std::string fname = std::string (argv[i]);
        if (fname.size () <= extension.size ()) //文件名的长度是否符合要求
            continue;

        std::transform (fname.begin (), fname.end (), fname.begin (), ↪
↪ (int(*) (int))tolower); //将某操作 (小写字母化) 应用于指定范围的每个元素
        //检查文件是否是 pcd 文件
        if (fname.compare (fname.size () - extension.size (), extension.size (), ↪
↪ extension) == 0)
        {
            // 读取点云, 并保存到 models
            PCD m;
            m.f_name = argv[i];
            pcl::io::loadPCDFile (argv[i], *m.cloud); //读取点云数据
            //去除点云中的 NaN 点 (xyz 都是 NaN)
            std::vector<int> indices; //保存去除的点的索引
            pcl::removeNaNFromPointCloud(*m.cloud, *m.cloud, indices); //去除点云中的 NaN 点

            models.push_back (m);
        }
    }
}

//简单地配准一对点云数据, 并返回结果
//参数 cloud_src 源点云
//参数 cloud_tgt 目标点云
//参数 output 输出点云
//参数 final_transform 成对变换矩阵
//参数 downsample 是否下采样
void pairAlign (const PointCloud::Ptr cloud_src, const PointCloud::Ptr cloud_tgt, ↪
↪ PointCloud::Ptr output, Eigen::Matrix4f &final_transform, bool downsample = false)

```

(下页继续)

(续上页)

```

{
    //
    //为了一致性和速度, 下采样
    // \note enable this for large datasets
    PointCloud::Ptr src (new PointCloud); //创建点云指针
    PointCloud::Ptr tgt (new PointCloud);
    pcl::VoxelGrid<PointT> grid; //VoxelGrid 把一个给定的点云, 聚集在一个局部的 3D 网格上, 并下
    采样和滤波点云数据
    if (downsample) //下采样
    {
        grid.setLeafSize (0.05, 0.05, 0.05); //设置体元网格的叶子大小
        //下采样 源点云
        grid.setInputCloud (cloud_src); //设置输入点云
        grid.filter (*src); //下采样和滤波, 并存储在 src 中
        //下采样 目标点云
        grid.setInputCloud (cloud_tgt);
        grid.filter (*tgt);
    }
    else //不下采样
    {
        src = cloud_src; //直接复制
        tgt = cloud_tgt;
    }

    //计算曲面的法向量和曲率
    PointCloudWithNormals::Ptr points_with_normals_src (new PointCloudWithNormals); //创
    建源点云指针 (注意点的类型包含坐标和法向量)
    PointCloudWithNormals::Ptr points_with_normals_tgt (new PointCloudWithNormals); //创
    建目标点云指针 (注意点的类型包含坐标和法向量)
    pcl::NormalEstimation<PointT, PointNormalT> norm_est; //该对象用于计算法向量
    pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ>
    → ()); //创建 kd 树, 用于计算法向量的搜索方法
    norm_est.setSearchMethod (tree); //设置搜索方法
    norm_est.setKSearch (30); //设置最近邻的数量
    norm_est.setInputCloud (src); //设置输入云
    norm_est.compute (*points_with_normals_src); //计算法向量, 并存储在 points_with_normals_
    → src
    pcl::copyPointCloud (*src, *points_with_normals_src); //复制点云 (坐标) 到 points_with_
    → normals_src (包含坐标和法向量)
    norm_est.setInputCloud (tgt); //这 3 行计算目标点云的法向量, 同上
    norm_est.compute (*points_with_normals_tgt);
    pcl::copyPointCloud (*tgt, *points_with_normals_tgt);

    //创建一个 自定义点表达方式的 实例
    MyPointRepresentation point_representation;
    //加权曲率维度, 以和坐标 xyz 保持平衡
    float alpha[4] = {1.0, 1.0, 1.0, 1.0};
    point_representation.setRescaleValues (alpha); //设置缩放值 (向量化点时使用)

    //创建非线性 ICP 对象 并设置参数
    pcl::IterativeClosestPointNonLinear<PointNormalT, PointNormalT> reg; //创建非线性 ICP
    对象 (ICP 变体, 使用 Levenberg-Marquardt 最优化)
    reg.setTransformationEpsilon (1e-6); //设置容许的最大误差 (迭代最优化)
    //***** 注意: 根据自己数据库的大小调节该参数
    reg.setMaxCorrespondenceDistance (0.1); //设置对应点之间的最大距离 (0.1m), 在配准过程中,
    忽略大于该阈值的点

```

(下页继续)



(续上页)

```

    reg.setPointRepresentation (boost::make_shared<const MyPointRepresentation> (point_
    ↪representation)); //设置点表达
        //设置源点云和目标点云
    //reg.setInputSource (points_with_normals_src); //版本不符合, 使用下面的语句
        reg.setInputCloud (points_with_normals_src); //设置输入点云 (待变换的点云)
    reg.setInputTarget (points_with_normals_tgt); //设置目标点云
        reg.setMaximumIterations (2); //设置内部优化的迭代次数

    // Run the same optimization in a loop and visualize the results
    Eigen::Matrix4f Ti = Eigen::Matrix4f::Identity (), prev, targetToSource;
    PointCloudWithNormals::Ptr reg_result = points_with_normals_src; //用于存储结果 (坐标 +
    法向量)

    for (int i = 0; i < 30; ++i) //迭代
    {
        PCL_INFO ("Iteration Nr. %d.\n", i); //命令行显示迭代的次数
        //保存点云, 用于可视化
        points_with_normals_src = reg_result; //
        //估计
        //reg.setInputSource (points_with_normals_src);
            reg.setInputCloud (points_with_normals_src); //重新设置输入点云 (待变换的点
    云), 因为经过上一次迭代, 已经发生变换了
        reg.align (*reg_result); //对齐 (配准) 两个点云

        Ti = reg.getFinalTransformation () * Ti; //累积 (每次迭代的) 变换矩阵
            //如果这次变换和上次变换的误差比阈值小, 通过减小最大的对应点距离的方法来进行细化
        if (fabs ((reg.getLastIncrementalTransformation () - prev).sum ()) < reg.
    ↪getTransformationEpsilon ())
            reg.setMaxCorrespondenceDistance (reg.getMaxCorrespondenceDistance () - 0.001); ↪
    ↪//减小对应点之间的最大距离 (上面设置过)
        prev = reg.getLastIncrementalTransformation (); //上一次变换的误差
        //显示当前配准状态, 在窗口的右视区, 简单的显示源点云和目标点云
        showCloudsRight(points_with_normals_tgt, points_with_normals_src);
    }

    targetToSource = Ti.inverse(); //计算从目标点云到源点云的变换矩阵
    pcl::transformPointCloud (*cloud_tgt, *output, targetToSource); //将目标点云 变换回到 源
    点云帧

    p->removePointCloud ("source"); //根据给定的 ID, 从屏幕中去除一个点云。参数是 ID
    p->removePointCloud ("target");
    PointCloudColorHandlerCustom<PointT> cloud_tgt_h (output, 0, 255, 0); //设置点云显示颜
    色, 下同
    PointCloudColorHandlerCustom<PointT> cloud_src_h (cloud_src, 255, 0, 0);
    p->addPointCloud (output, cloud_tgt_h, "target", vp_2); //添加点云数据, 下同
    p->addPointCloud (cloud_src, cloud_src_h, "source", vp_2);

    PCL_INFO ("Press q to continue the registration.\n");
    p->spin ();

    p->removePointCloud ("source");
    p->removePointCloud ("target");

    //add the source to the transformed target
    *output += *cloud_src; // 拼接点云图 (的点) 点数数目是两个点云的点数和

```

(下页继续)

(续上页)

```

    final_transform = targetToSource; //最终的变换。目标点云到源点云的变换矩阵
}

//***** 入口函数 *****

//主函数
int main (int argc, char** argv)
{
    //读取数据
    std::vector<PCD, Eigen::aligned_allocator<PCD> > data; //模型
    loadData (argc, argv, data); //读取 pcd 文件数据, 定义见上面

    //检查用户数据
    if (data.empty ())
    {
        PCL_ERROR ("Syntax is: %s <source.pcd> <target.pcd> [*]", argv[0]); //语法
        PCL_ERROR ("[*] - multiple files can be added. The registration results of (i,
→i+1) will be registered against (i+2), etc"); //可以使用多个文件
        return (-1);
    }
    PCL_INFO ("Loaded %d datasets.", (int)data.size ()); //显示读取了多少个点云文件

    //创建一个 PCLVisualizer 对象
    p = new pcl::visualization::PCLVisualizer (argc, argv, "Pairwise Incremental
→Registration example"); //p 是全局变量
    p->createViewPort (0.0, 0, 0.5, 1.0, vp_1); //创建左视区
    p->createViewPort (0.5, 0, 1.0, 1.0, vp_2); //创建右视区

    //创建点云指针和变换矩阵
    PointCloud::Ptr result (new PointCloud), source, target; //创建 3 个点云指针, 分别
用于结果, 源点云和目标点云
    //全局变换矩阵, 单位矩阵, 成对变换
    //逗号表达式, 先创建一个单位矩阵, 然后将成对变换 赋给 全局变换
    Eigen::Matrix4f GlobalTransform = Eigen::Matrix4f::Identity (), pairTransform;

    //遍历所有的点云文件
    for (size_t i = 1; i < data.size (); ++i)
    {
        source = data[i-1].cloud; //源点云
        target = data[i].cloud; //目标点云
        showCloudsLeft(source, target); //在左视区, 简单的显示源点云和目标点云
        PointCloud::Ptr temp (new PointCloud); //创建临时点云指针
        //显示正在配准的点云文件名和各自的点数
        PCL_INFO ("Aligning %s (%d points) with %s (%d points).\n", data[i-1].f_name.c_
→str (), source->points.size (), data[i].f_name.c_str (), target->points.size ());

        //*****
        //配准 2 个点云, 函数定义见上面
        pairAlign (source, target, temp, pairTransform, true);
        //将当前的一对点云数据, 变换到全局变换中。
        pcl::transformPointCloud (*temp, *result, GlobalTransform);
        //更新全局变换
        GlobalTransform = GlobalTransform * pairTransform;
        //*****
    }
}

```

(下页继续)

(续上页)

```

        // 保存成对的配准结果, 变换到第一个点云帧
        std::stringstream ss; //这两句是生成文件名
        ss << i << ".pcd";
        pcl::io::savePCDFile (ss.str (), *result, true); //保存成对的配准结果
    }
}

```

- 重新生成项目。
- 到改项目的 Debug 目录下, 按住 Shift, 同时点击鼠标右键, 在当前窗口打开 CMD 窗口。
- 在命令行中输入 pairwise\_incremental\_registration.exe capture0001.pcd capture0002.pcd capture0003.pcd capture0004.pcd capture0005.pcd, 执行程序。得到如下图所示的结果。

## 参考

## 1.2.8 PCL 系列——将点云数据写入 PCD 格式文件

### PCL 系列

#### 操作

- 在 VS2010 中新建一个文件 pcd\_write.cpp, 然后将下面的代码复制到文件中。
- 参照之前的文章, 配置项目的属性。设置包含目录和库目录和附加依赖项。

```

#include <iostream> //标准输入输出流
#include <pcl/io/pcd_io.h> //PCL 的 PCD 格式文件的输入输出头文件
#include <pcl/point_types.h> //PCL 对各种格式的点的支持头文件

int main (int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZ> cloud; // 创建点云 (不是指针)

    //填充点云数据
    cloud.width = 5; //设置点云宽度
    cloud.height = 1; //设置点云高度
    cloud.is_dense = false; //非密集型
    cloud.points.resize (cloud.width * cloud.height); //变形, 无序
    //设置这些点的坐标
    for (size_t i = 0; i < cloud.points.size (); ++i)
    {
        cloud.points[i].x = 1024 * rand () / (RAND_MAX + 1.0f);
        cloud.points[i].y = 1024 * rand () / (RAND_MAX + 1.0f);
        cloud.points[i].z = 1024 * rand () / (RAND_MAX + 1.0f);
    }

    //保存到 PCD 文件
    pcl::io::savePCDFileASCII ("test_pcd.pcd", cloud); //将点云保存到 PCD 文件中
    std::cerr << "Saved " << cloud.points.size () << " data points to test_pcd.pcd." << "\n";
    //显示点云数据
    for (size_t i = 0; i < cloud.points.size (); ++i)
        std::cerr << " " << cloud.points[i].x << " " << cloud.points[i].y << " " << cloud.points[i].z << "\n";
}

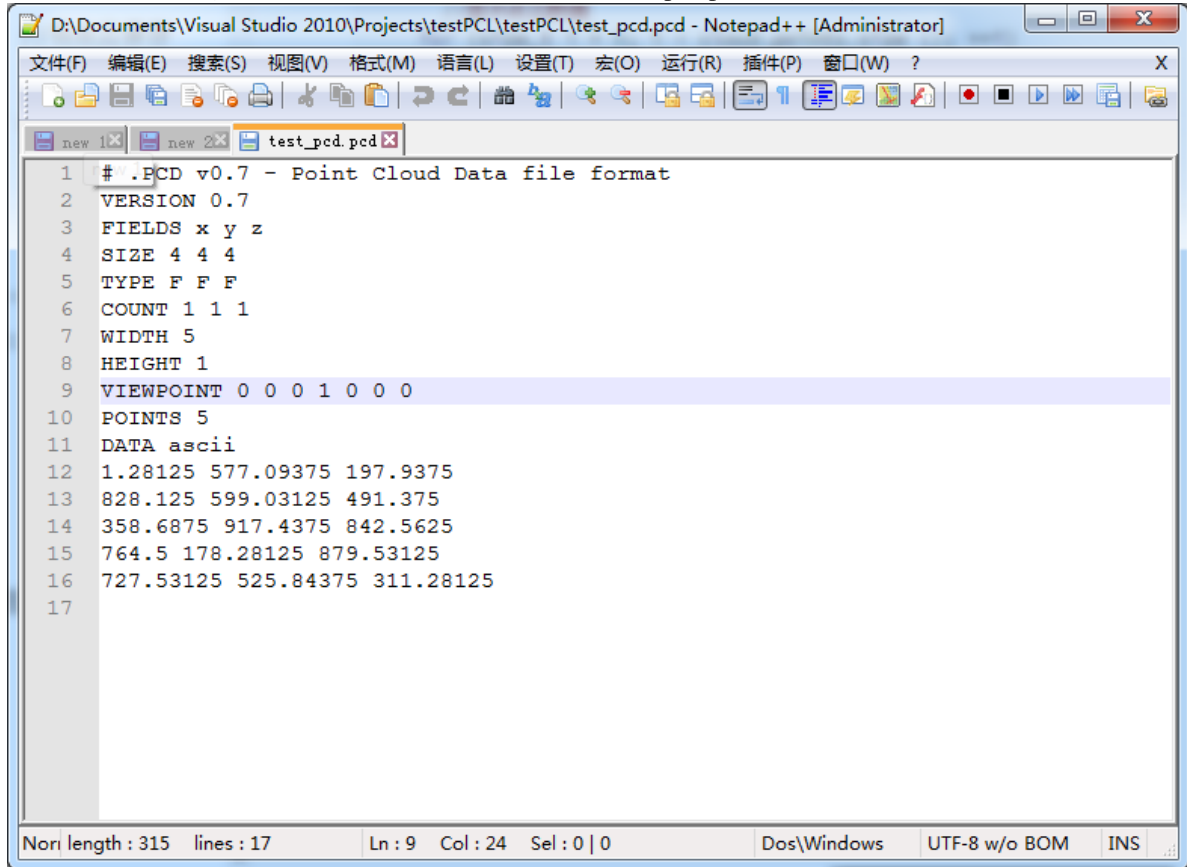
```

(下页继续)

(续上页)

```
return (0);
}
```

- 运行结束后，在项目目录下生成 test\_pcd.pcd 文件，内容如下图所示。



```

1 #.PCD v0.7 - Point Cloud Data file format
2 VERSION 0.7
3 FIELDS x y z
4 SIZE 4 4 4
5 TYPE F F F
6 COUNT 1 1 1
7 WIDTH 5
8 HEIGHT 1
9 VIEWPOINT 0 0 0 1 0 0 0
10 POINTS 5
11 DATA ascii
12 1.28125 577.09375 197.9375
13 828.125 599.03125 491.375
14 358.6875 917.4375 842.5625
15 764.5 178.28125 879.53125
16 727.53125 525.84375 311.28125
17

```

## 参考

### 1.2.9 PCL 系列——平面模型分割

#### PCL 系列

#### 说明

#### 代码下载

## 代码

```

#include <iostream>
#include <pcl/ModelCoefficients.h>           //模型系数
#include <pcl/io/pcd_io.h>                   //输入输出
#include <pcl/point_types.h>                 //点云 (类型)
#include <pcl/sample_consensus/method_types.h> //随机样本一致性算法 方法类型
#include <pcl/sample_consensus/model_types.h> //随机样本一致性算法 模型类型
#include <pcl/segmentation/sac_segmentation.h> //随机样本一致性算法 分割方法

int
main (int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZ> cloud; //创建点云对象, 用于存储点云数据
    //填充点云数据
    cloud.width = 15;
    cloud.height = 1;
    cloud.points.resize (cloud.width * cloud.height);

    //生成随机数据
    for (size_t i = 0; i < cloud.points.size (); ++i)
    {
        cloud.points[i].x = 1024 * rand () / (RAND_MAX + 1.0f);
        cloud.points[i].y = 1024 * rand () / (RAND_MAX + 1.0f);
        cloud.points[i].z = 1.0;
    }

    //设置几个局外点
    cloud.points[0].z = 2.0;
    cloud.points[3].z = -2.0;
    cloud.points[6].z = 4.0;
    //显示点云数量和坐标信息
    std::cerr << "Point cloud data: " << cloud.points.size () << " points" << std::endl;
    for (size_t i = 0; i < cloud.points.size (); ++i)
        std::cerr << " " << cloud.points[i].x << " "
                    << cloud.points[i].y << " "
                    << cloud.points[i].z << "\n";
    ↪std::endl;

    pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients); //存储输出的模型的系数
    pcl::PointIndices::Ptr inliers (new pcl::PointIndices); //存储内点, 使用的点

    //创建分割对象
    pcl::SACSegmentation<pcl::PointXYZ> seg;
    //可选设置
    seg.setOptimizeCoefficients (true);
    //必须设置
    seg.setModelType (pcl::SACMODEL_PLANE); //设置模型类型, 检测平面
    seg.setMethodType (pcl::SAC_RANSAC); //设置方法【聚类或随机样本一致性】
    seg.setDistanceThreshold (0.01);
    seg.setInputCloud (cloud.makeShared ());
    seg.segment (*inliers, *coefficients); //分割操作

    if (inliers->indices.size () == 0) //根据内点数量, 判断是否成功

```

(下页继续)

(续上页)

```
{  
  
    PCL_ERROR ("Could not estimate a planar model for the given dataset.");  
    return (-1);  
}  
  
    //显示模型的系数  
std::cerr << "Model coefficients: " << coefficients->values[0] << " "  
                                                    <<coefficients->values[1] << " "  
                                                    <<coefficients->values[2] << " "  
                                                    <<coefficients->values[3] <<std::endl;  
  
    //显示估计平面模型过程中使用的内点  
std::cerr << "Model inliers: " << inliers->indices.size () << std::endl;  
    for (size_t i = 0; i < inliers->indices.size (); ++i)  
        std::cerr << inliers->indices[i] << "      " <<cloud.  
->points[inliers->indices[i]].x << " " <<cloud.  
->points[inliers->indices[i]].y << " " <<cloud.  
->points[inliers->indices[i]].z << std::endl;  
    return (0);  
}
```

## 运行结果

```

C:\Windows\system32\cmd.exe
Point cloud data: 15 points
  1.28125 577.094 2
 197.938 828.125 1
 599.031 491.375 1
 358.688 917.438 -2
 842.563 764.5 1
 178.281 879.531 1
 727.531 525.844 4
 311.281 15.3438 1
 93.5938 373.188 1
 150.844 169.875 1
 1012.22 456.375 1
 121.938 4.78125 1
 9.125 386.938 1
 544.406 584.875 1
 616.188 621.719 1
Model coefficients: 0 0 1 -1
Model inliers: 12
1   197.938 828.125 1
2   599.031 491.375 1
4   842.563 764.5 1
5   178.281 879.531 1
7   311.281 15.3438 1
8   93.5938 373.188 1
9   150.844 169.875 1
10  1012.22 456.375 1
11  121.938 4.78125 1
12  9.125 386.938 1
13  544.406 584.875 1
14  616.188 621.719 1
请按任意键继续. . .

```

## 参考

《点云库 PCL 学习教程》14 章

## 1.2.10 PCL 系列——拼接两个点云

## PCL 系列

## 说明

通过本教程，我们将会学会：

- 如何拼接两个不同的点云的点，约束条件是两个数据集中的域的数量和类型必须相等。\* 如何拼接两个不同点云的域，约束条件是连个数据集中的点的数量必须相等。

## 操作

- 在 VS2010 中新建一个文件 concatenate\_clouds.cpp, 然后将下面的代码复制到文件中。
- 参照之前的文章, 配置项目的属性。设置包含目录和库目录和附加依赖项。

```
#include <iostream> //标准输入输出流
#include <pcl/io/pcd_io.h> //PCL 的 PCD 格式文件的输入输出头文件
#include <pcl/point_types.h> //PCL 对各种格式的点的支持头文件
//比如, 你的程序遇到调用栈用完了的威胁。你说, 你到什么地方借内存,
//存放你的错误信息? cerr 的目的, 就是在你最需要它的紧急情况下,
//还能得到输出功能的支持。 缓冲区的目的, 就是减少刷屏的次数

// 程序拼接 A 和 B 到 C
int main (int argc, char** argv)
{
    if (argc != 2) // 需要一个参数 -f 或 -p
    {
        std::cerr << "please specify command line arg '-f' or '-p'" << std::endl;
        exit(0);
    }

    // 用于拼接不同点云的点的变量
    pcl::PointCloud<pcl::PointXYZ> cloud_a, cloud_b, cloud_c; //创建点云 (不是指针), 存储点坐标 xyz

    // 用于拼接不同点云的域 (点和法向量) 的变量
    pcl::PointCloud<pcl::Normal> n_cloud_b; //创建点云, 储存法向量
    pcl::PointCloud<pcl::PointNormal> p_n_cloud_c; //创建点云, 储存点坐标和法向量

    //填充点云数据
    cloud_a.width = 5; //设置宽度
    cloud_a.height = cloud_b.height = n_cloud_b.height = 1; //设置高度
    cloud_a.points.resize (cloud_a.width * cloud_a.height); //变形, 无序
    if (strcmp(argv[1], "-p") == 0) //根据输入参数, 设置点云
    {
        cloud_b.width = 3; //cloud_b 用于拼接不同点云的点
        cloud_b.points.resize (cloud_b.width * cloud_b.height);
    }
    else{
        n_cloud_b.width = 5; //n_cloud_b 用于拼接不同点云的域
        n_cloud_b.points.resize (n_cloud_b.width * n_cloud_b.height);
    }
    for (size_t i = 0; i < cloud_a.points.size (); ++i) //设置 cloud_a 中点的坐标 (随机数)
    {
        cloud_a.points[i].x = 1024 * rand () / (RAND_MAX + 1.0f);
        cloud_a.points[i].y = 1024 * rand () / (RAND_MAX + 1.0f);
        cloud_a.points[i].z = 1024 * rand () / (RAND_MAX + 1.0f);
    }
    if (strcmp(argv[1], "-p") == 0)
        for (size_t i = 0; i < cloud_b.points.size (); ++i) //设置 cloud_b 中点的坐标 (随机数)
        {
            cloud_b.points[i].x = 1024 * rand () / (RAND_MAX + 1.0f);
            cloud_b.points[i].y = 1024 * rand () / (RAND_MAX + 1.0f);
            cloud_b.points[i].z = 1024 * rand () / (RAND_MAX + 1.0f);
        }
    else // -f
        for (size_t i = 0; i < n_cloud_b.points.size (); ++i) //设置 n_cloud_b 中点的坐标 (随机数)
        {
```

(下页继续)



(续上页)

```

        n_cloud_b.points[i].normal[0] = 1024 * rand () / (RAND_MAX + 1.0f);
        n_cloud_b.points[i].normal[1] = 1024 * rand () / (RAND_MAX + 1.0f);
        n_cloud_b.points[i].normal[2] = 1024 * rand () / (RAND_MAX + 1.0f);
    }

    // 打印拼接用的数据 A 和 B
    std::cerr << "Cloud A: " << std::endl;
    for (size_t i = 0; i < cloud_a.points.size (); ++i) //打印 cloud_a 的点坐标信息
        std::cerr << " " << cloud_a.points[i].x << " " << cloud_a.points[i].y << " " <<
        cloud_a.points[i].z << std::endl;

    std::cerr << "Cloud B: " << std::endl; //打印 Cloud B
    if (strcmp(argv[1], "-p") == 0) //若输入参数是-p, 打印 cloud_b;
        for (size_t i = 0; i < cloud_b.points.size (); ++i)
            std::cerr << " " << cloud_b.points[i].x << " " << cloud_b.points[i].y << " " <<
            cloud_b.points[i].z << std::endl;
        else //若-f, 打印 n_cloud_b
            for (size_t i = 0; i < n_cloud_b.points.size (); ++i)
                std::cerr << " " << n_cloud_b.points[i].normal[0] << " " << n_cloud_b.
                points[i].normal[1] << " " << n_cloud_b.points[i].normal[2] << std::endl;

    //复制点云中的点
    if (strcmp(argv[1], "-p") == 0)
    {
        cloud_c = cloud_a;
        cloud_c += cloud_b; // cloud_a + cloud_b 意思是 cloud_c 包含了 a 和 b 中的点, c 的点数_
        = a 的点数 +b 的点数
        std::cerr << "Cloud C: " << std::endl; //打印 Cloud C
        for (size_t i = 0; i < cloud_c.points.size (); ++i) //打印 Cloud C
            std::cerr << " " << cloud_c.points[i].x << " " << cloud_c.points[i].y << " " <<
            cloud_c.points[i].z << " " << std::endl;
    }
    else //若输入参数是-f
    {
        pcl::concatenateFields (cloud_a, n_cloud_b, p_n_cloud_c); //拼接 (点) cloud_a 和 (法向
        量) n_cloud_b 到 p_n_cloud_c
        std::cerr << "Cloud C: " << std::endl;
        for (size_t i = 0; i < p_n_cloud_c.points.size (); ++i) //打印 Cloud C
            std::cerr << " " <<
            p_n_cloud_c.points[i].x << " " << p_n_cloud_c.points[i].y << " " << p_n_cloud_
            c.points[i].z << " " <<
            p_n_cloud_c.points[i].normal[0] << " " << p_n_cloud_c.points[i].normal[1] <<
            " " << p_n_cloud_c.points[i].normal[2] << std::endl;
    }

    return (0);
}

```

- 重新生成项目。
- 到改项目的 Debug 目录下, 按住 Shift, 同时点击鼠标右键, 在当前窗口打开 CMD 窗口。
- 在 CMD 窗口中, 输入命令 `concatenate_clouds.exe -p`, 执行拼接不同点云的点。结果如下图所示

```

C:\Windows\system32\cmd.exe
Cloud A:
  1.28125 577.094 197.938
  828.125 599.031 491.375
  358.688 917.438 842.563
  764.5 178.281 879.531
  727.531 525.844 311.281
Cloud B:
  15.3438 93.5938 373.188
  150.844 169.875 1012.22
  456.375 121.938 4.78125
Cloud C:
  1.28125 577.094 197.938
  828.125 599.031 491.375
  358.688 917.438 842.563
  764.5 178.281 879.531
  727.531 525.844 311.281
  15.3438 93.5938 373.188
  150.844 169.875 1012.22
  456.375 121.938 4.78125
请按任意键继续. . .
  
```

示。

- 在 CMD 窗口中，输入命令 `concatenate_clouds.exe -f`，执行拼接不同点云的域（比如点和法向量）。结果如下图所示。

```

C:\Windows\system32\cmd.exe
Cloud A:
  1.28125 577.094 197.938
  828.125 599.031 491.375
  358.688 917.438 842.563
  764.5 178.281 879.531
  727.531 525.844 311.281
Cloud B:
  15.3438 93.5938 373.188
  150.844 169.875 1012.22
  456.375 121.938 4.78125
  9.125 386.938 544.406
  584.875 616.188 621.719
Cloud C:
  1.28125 577.094 197.938 15.3438 93.5938 373.188
  828.125 599.031 491.375 150.844 169.875 1012.22
  358.688 917.438 842.563 456.375 121.938 4.78125
  764.5 178.281 879.531 9.125 386.938 544.406
  727.531 525.844 311.281 584.875 616.188 621.719
请按任意键继续. . .
  
```

## 参考

### 1.2.11 PCL 系列——读入 PCD 格式文件

#### PCL 系列

#### 操作

- 在 VS2010 中新建一个文件 `read_pcd.cpp`，然后将下面的代码复制到文件中。

- 参照之前的文章，配置项目的属性。设置包含目录和库目录和附加依赖项。

```
#include <iostream> //标准输入输出流
#include <pcl/io/pcd_io.h> //PCL 的 PCD 格式文件的输入输出头文件
#include <pcl/point_types.h> //PCL 对各种格式的点的支持头文件

int main (int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>); // 创建点云 (指针)

    if (pcl::io::loadPCDFile<pcl::PointXYZ> ("test_pcd.pcd", *cloud) == -1) /* 读入 PCD 格式的文件，如果文件不存在，返回-1
    {
        PCL_ERROR ("Couldn't read file test_pcd.pcd \n"); //文件不存在时，返回错误，终止程序。
        return (-1);
    }
    std::cout << "Loaded "
        << cloud->width * cloud->height
        << " data points from test_file.pcd with the following fields: "
        << std::endl;
    //for (size_t i = 0; i < cloud->points.size (); ++i) //显示所有的点
    for (size_t i = 0; i < 5; ++i) // 为了方便观察，只显示前 5 个点
        std::cout << " " << cloud->points[i].x
            << " " << cloud->points[i].y
            << " " << cloud->points[i].z << std::endl;

    return (0);
}
```

```
C:\Windows\system32\cmd.exe
Loaded 10031 data points from test_pcd.pcd with the following fields:
-36.64 -61.93 -6.34
-36.559 -61.959 -6.84
-37.02 -62.02 -6.42
-36.71 -62.19 -6.38
-36.419 -62.35 -6.38
-36.19 -62.48 -6.5
-37.14 -62.29 -6.26
-36.85 -62.459 -6.23
-37.539 -62.37 -6.3
-37.22 -62.54 -6.23
请按任意键继续...
```

- 编译运行,如下图所示。

## 参考

## 1.3 Linux

### 1.3.1 内网穿透之 ngrok

- 内网穿透：从外网（有 IP）访问局域网中的设备。

## 问题

- 在家里有一台计算机，希望在外面时，能够查看家里计算机运行情况。但我家的宽带是移动宽带，没有给分配公网 IP。

## 解决方法

- 这是一个典型的外网访问内网问题，有一些公司在提供相关技术，比如花生壳。
- 本文使用 ngrok 来解决这个问题。

## 第一步，注册账号，开通隧道

- 访问网站：[sunny-ngrok](#)，注册账号。
- 开通 tcp 协议的隧道，如下图所示。（**注意：**试验的话，购买网页最下面的免费的套餐即可）
- 点击隧道管理，查看刚刚开通的隧道的相关信息，如下所示：
- 记住以下三条信息。

```
隧道 id:5f191234d990a838
隧道端口:14568
服务器地址:free.idcfengye.com (请不要暴露此地址，避免服务器遭受攻击，谢谢)
```

## 第二步，下载客户端

- 点击[下载链接](#)，下载客户端。比如，我是 Ubuntu 系统，下载 Linux 64bit 版本。

## 第三步，安装客户端

- 安装很简单：解压后，直接执行可执行命令即可。
- 以下是我的操作步骤，我将它放到/usr/local/bin/目录下，方便任何地方执行。

```
unzip linux_amd64.zip
cd linux_amd64/
ls
sudo cp sunny /usr/local/bin/
cd
sunny -h
```

## 第四步，启动客户端

- 启动客户端很简单，一句命令即可。
- 需要使用隧道 id。比如，上面我的隧道 id 是：隧道 id:5f191234d990a838
- 执行命令是：sunny clientid 隧道 id
- 注意替换成自己的隧道 id：

```
sunny clientid 5f191234d990a838
```

## 第五步，测试

- 换一台电脑，执行以下命令测试：ssh -p 端口号 (ngrok 注册时的端口号) 用户名 @ 服务器地址
- 比如，我的用户名是：xuezhisd，端口号是：14568，服务器地址是：free.idcfengye.com。
- 我的测试命令是：ssh -p 14568 xuezhisd@free.idcfengye.com。
- 成功地访问家里的计算机，顺利解决问题！ :-D

## 参考

- [sunny-ngrok](#) 提供免费 ngrok。
- 使用 ssh，ngrok 外网远程连接 linux 提供了参考教程。

## 1.3.2 双系统 (Linux) 时间不一致问题

### 解决双系统时间不一致问题

#### 问题描述

- 最近需要使用 Linux (Ubuntu)，所以在自己的电脑上安装了双系统 (Windows+Ubuntu)。但是发现 Windows 的时间不正确，每次开机时，时间都会慢 8 个小时。
- 每次通过同步互联网时间可以更正，但是重启后又是如此。
- 后来，发现 windows time 服务没有开启。将其设置为自启动之后，每次开机后依然如此。而且 windows time 服务没有开启。
- 最后发现两个系统的时间相差 8 小时，意识到是两个系统设置不同引起的。

#### 问题分析

- UTC 即 Universal Time Coordinated，协调世界时
- GMT 即 Greenwich Mean Time，格林尼治平时
- Windows 把系统硬件时间当作本地时间 (local time)，即操作系统中显示的时间跟 BIOS 中显示的时间是一样的。
- Linux/Unix/Mac 把硬件时间当作 UTC，操作系统中显示的时间是硬件时间经过换算得来的。
- 当 PC 中同时有多系统共存时，就出现了问题。

## 问题解决

- 方法一 **Ubuntu 不使用 UTC**
- 如果 Ubuntu 不使用 UTC 时间，则可以与 Windows 时间保持一致。
- ubuntu 默认开启 UTC, 即协调世界时，关闭它即可

```
sudo vi /etc/default/rcS
```

- 将 UTC=yes 改为 UTC=no
- 方法二修改 **Windows 对硬件时间的对待方式**
- 让 Windows 把硬件时间当作 UTC
- 方法:
  - 开始-> 运行->CMD, 在命令行中输入下面命令并回车

```
Reg add KLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation /v RealTimeIsUniversal /t REG_DWORD /d 1
```

## CentOS 时间同步问题

### 编辑时间配置文件

- 使用 vi 编辑/etc/sysconfig/clock 文件，添加如下内容：

```
ZONE="Asia/Shanghai"  
UTC=false          # 设置为 false, 硬件时钟不于 utc 时间一致  
ARC=false
```

- 使配置立即生效 source /etc/sysconfig/clock

### 设置为上海时区

- 执行下面的命令，将 Linux 的时区设置为上海时区

```
ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

### 对准时间

- 注意：需要联网
- 安装 ntpdate 的命令:yum install ntp
- 执行以下命令，对准时间

```
ntpdate us.pool.ntp.org
```

### 设置硬件时间和系统时间一致并校准

- 命令:

```
/sbin/hwclock --systohc # 设置硬件时间和系统时间一致并校准
```

至此，CentOS 的时间已经同步了。

### 参考

- <http://www.ithov.com/linux/122268.shtml>
- <http://os.51cto.com/art/201004/192549.htm>

## 1.3.3 Linux 学习笔记

### 终端

#### 终端和控制台

- 终端：一台只有显示器和键盘，能够通过串口连接到计算机的设备就叫终端。一台计算机可以有 N 个终端。
- 控制台：直接连接在电脑上的那套键盘和显示器就叫做控制台。一台计算机只有一个控制台。
- 区别：终端是通过串口连接上的，不是计算机本身就有的设备；控制台是计算机本身就有的设备。

#### 终端的本质

- 终端本质上对应着 Linux 上的/dev/tty 设备。

#### 终端模拟器

gnome-terminal、kconsole、xterm、xfce 等等

#### 终端的 6 个 Virtual consoles

- Linux 默认提供了 6 个 virtual consoles，可以通过 [Ctrl]+[Alt]+[F1]~[F6] 进行切换。
- 切换回图形界面：[Ctrl]+[Alt]+[F7]

## Shell

### 历史命令

向上键

### 获取帮助

```
man command_name
```

### 一些命令

- banner 绘制图形
  - sudo apt-get update; sudo apt-get install sysvbanner
  - banner shiyanlou

## Linux 用户管理

### 查看用户

- who

```
who am i
# 或
who mom likes
whoami # 只返回用户名
```

- pts/0 中的 pts 表示伪终端，“伪”是相对于/dev/tty 设备而言。
- [Ctrl]+[Alt]+[F1]~[F7] 进行切换的/dev/tty 是真终端
- 0 是伪终端的序号。

### 文件类型

- Linux 里面一切皆文件
- 标识符 l 类型
- dl 目录
- ll 软链接
- bl 块设备
- cl 字符设备
- ssocket
- pl 管道
- -l 普通文件



## 文件权限

- rwx(421)
- 权限 | 二进制数 | 十进制数
- rl100l4
- wl010l2
- xl001l1

## 1.3.4 Linux 下安装 Eclipse

### 前提

- 安装 Eclipse 之前，需要先安装 Java。
- 安装 Java 参考 [ubuntu 下搭建 JAVA 开发环境](#)

### 下载安装包

- 下载地址：[点此下载](#)
- 提示：校园网可以使用 Ipv6 站点下载，速度快。

### 复制文件

- 将文件复制到安装路径下 /usr/local/，并解压，最后执行。

```
cp eclipse-java-luna-SR2-linux-gtk-x86_64.tar.gz /usr/local/
# 解压
tar -zxvf eclipse-java-luna-SR2-linux-gtk-x86_64.tar.gz
cd eclipse
# 命令行中启动 Eclipse
./eclipse
```

### 制作快捷方式

```
vi Eclipse.desktop

# 在文件中输入下面的内容
[Desktop Entry]
Name=Eclipse
Exec=/usr/local/eclipse/eclipse -desktop
Icon=/usr/local/eclipse/plugins/org.eclipse.platform_4.4.2.v20150204-1700/eclipse64.
  ➡png
Type=Application
Comment=
Path=
Terminal=false
StartupNotify=false
```

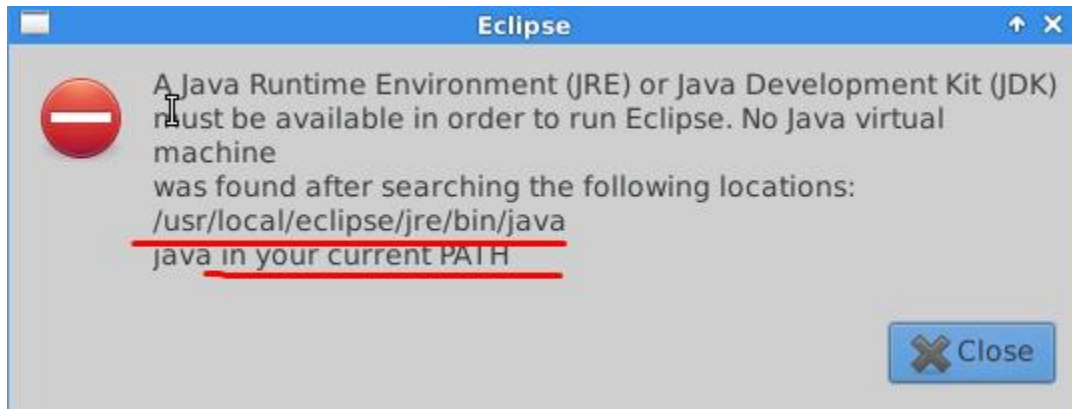
(下页继续)

(续上页)

```
# 复制到桌面中
mv Eclipse.desktop ~/Desktop/
```

## 错误分析

- 双击快捷方式时，如果出现以下错误，



- 如果安装了 JDK，通过在 eclipse/路径下新建 jre/bin 来解决。

```
mkdir jre
cd jre
ln -s $JAVA_HOME/bin bin
```

- 重新双击快捷方式，解决错误。

## 参考

### 1.3.5 Linux 终端提示符显示颜色

#### 用法

- 将以下内容复制到 ~/.bashrc 中，

```
PS1='[\[\e[32;40m\]\u@\w]\$[\e[m\]'
```

- 效果如下图所示。

```
[xuezhisd@~]$cat test.py
import sys

print sys.argv
[xuezhisd@~]$
```

## 参考

### 1.3.6 Linux 判断文件和文件夹是否存在

- shell 判断文件, 目录是否存在或者具有权限

```
#!/bin/sh

myPath="/var/log/httpd/"
myFile="/var /log/httpd/access.log"

# 这里的-x 参数判断$myPath 是否存在并且是否具有可执行权限
if [ ! -x "$myPath" ]; then
    mkdir "$myPath"
fi

# 这里的-d 参数判断$myPath 是否存在
if [ ! -d "$myPath" ]; then
    mkdir "$myPath"
fi

# 这里的-f 参数判断$myFile 是否存在
if [ ! -f "$myFile" ]; then
    touch "$myFile"
fi

# 其他参数还有-n, -n 是判断一个变量是否是否有值
if [ ! -n "$myVar" ]; then
    echo "$myVar is empty"
    exit 0
fi

# 两个变量判断是否相等
if [ "$var1" = "$var2" ]; then
    echo '$var1 eq $var2'
else
    echo '$var1 not eq $var2'
fi
```

- 转载地址: <http://www.cnblogs.com/platero/p/4021561.html>

### 1.3.7 Linux 命令——adduser

#### adduser

- **功能:** 建立用户账号
- 在 Slackware 中, adduser 指令是个 script 程序, 利用交谈的方式取得输入的用户帐号资料, 然后再交由真正建立帐号的 useradd 指令建立新用户, 如此可方便管理员建立用户帐号。
- 在 RedHatLinux 中, adduser 指令则是 useradd 指令的符号连接, 两者实际上是同一个指令。
- **参考:** useradd

## useradd

- 功能：建立用户账号
- 提示：
  - 使用 useradd 指令所建立的帐号，实际上是保存在/etc/passwd 文本文件中；
  - 帐号建好之后，再用 passwd 设定帐号的密码；
  - 可用 userdel 删除帐号；
  - 设定 ID 值时尽量要大于 500，以免冲突。因为 Linux 安装后会建立一些特别用户，一般 0 到 499 之间的值留给 bin、mail 这样的系统账号。
  - 系统用户的权限稍微大一些，可以修改自己的某些信息；系统用户的 UID<500；
- 选项：
  - c：加上备注文字。备注文字会保存在 passwd 的备注栏位中；
  - d：指定用户登入时的启始目录。
  - D：变更预设值；
  - e：指定帐号的有效期限，缺省表示永久有效；
  - f：指定在密码过期后多少天即关闭该帐号；
  - g：指定用户所属的群组。
  - G：指定用户所属的附加群组；
  - m：自动建立用户的登入目录；
  - M：不要自动建立用户的登入目录；
  - r：建立系统帐号；
  - s：指定用户登入后所使用的 shell；
  - u：指定用户 id；
- 示例：

```
# 新建用户 test
useradd test
# 新建用户 test, 并添加备注
useradd test -c "temp user"
# 新建用户 test, 并指定目录
useradd test -d /home/test

# 添加新用户 jack, 主要组:sales, 次要组: company
useradd jack -g sales -G company,employees
# 新建用户 test, 并自动建立用户登入目录
useradd test -m
# 新建系统用户 test
useradd test -r
# 新建一个用户, 并指定用户登录后使用的 shell
useradd test -s /bin/bash
# 建立一个新用户账户, 并设置 ID:
useradd test -u 544
```

## 参考网址

### 1.3.8 Linux 命令——alias

#### alias

- 功能：设置指令的别名
- 提示：
  - 该命令可将一些较长的命令进行简化；
  - 使用 alias 时，用户必须使用单引号'' 将原来的命令引起来，防止特殊字符导致错误；
  - alias 命令的作用只局限于该次登入的操作，若要每次登入都能够使用这些命令别名，则可将相应的 alias 命令存放到 bash 的初始化文件/etc/.bashrc 中；
- 选项：
  - -p：打印已经设置的命令别名
- 示例：

```
# 显示已定义的命令别名
alias
alias -p
# 显示某命令别名的信息
alias ll
# 定义命令 xuezhisd, 显示登录用户信息
alias xuezhisd='who am i'
```

## 参考网址

### 1.3.9 Linux 命令——awk

#### awk

- 功能：模式匹配语言
  - 参考：《awk 工作原理》
  - awk 是一种编程语言，用于在 \*nix 下对文本和数据进行处理；
  - 数据可以来自标准输入 (stdin)、一个或多个文件，或其它命令的输出；
  - 它支持用户自定义函数和动态正则表达式等先进功能，是 \*nix 下的一个强大编程工具；
  - 它可以在命令行中使用，但更多是作为脚本来使用；
  - awk 有很多内建的功能，比如数组、函数等，这是它和 C 语言的相同之处；
  - 灵活性是 awk 一大的优势。
  - awk 脚本是由模式和操作组成的；
    - \* 模式可以是以下任意一个：
      - 正则表达式：使用通配符的扩展集；
      - 关系表达式：使用运算符进行操作，可以是字符串或数字的比较测试。
      - 模式匹配表达式：用运算符 ~（匹配）和 ~！（不匹配）；

- BEGIN 语句块、pattern 语句块、END 语句块；
- \* 操作由一个或多个命令、函数、表达式组成，之间由换行符或分号隔开，并位于大括号内，主要部分是：
  - 变量或数组赋值
  - 输出命令
  - 内置函数
  - 控制流语句
- 选项：
  - -F fs fs 指定输入分隔符，fs 可以是字符串或正则表达式，如-F: -v var=value 赋值一个用户定义变量，将外部变量传递给 awk -f scripfile 从脚本文件中读取 awk 命令 -m[fr] val 对 val 值设置内在限制，-mf 选项限制分配给 val 的最大块数目；-mr 选项限制记录的最大数目。这两个功能是 Bell 实验室版 awk 的扩展功能，在标准 awk 中不适用。
  - -F: 指定将输入分离器
  - -f: 指定程序的源文件
  - -help: 显示帮助信息
  - -version: 显示版本信息
- 示例：

```
#
```

## 参考网址

### 1.3.10 Linux 命令——diff

#### diff

- 功能：逐行比较两个文件
- 提示：
  - 默认情况下，跟两个参数，指定对比的文件名称；
  - 如果一个参数指定文件名称，另一个参数指定路径，则将与该路径下同名的文件进行比较；
  - 如果使用“-”代替“文件”参数，则要比对的内容将来自标准输入；
  - diff 命令是以逐行的方式，比较文本文件的异同处；
  - 如果该命令指定进行目录的比较，则将会比较该目录中具有相同文件名的文件，而不会对其子目录文件进行任何比较操作；
  - 返回结果中，字母”a”、”d”、”c” 分别表示添加、删除及修改操作；
- 选项：
  - -c: 显示全部内容，并标出不同之处；
  - -i: 不检查大小写；
- 示例：

```
# 比较 test.txt 和 test.c
diff test.txt test.c
# 比较 test.txt 和 work/test.txt
diff test.txt work
# 比较 test.txt 和 test.c, 将显示全部内容, 并标出不同之处
diff test.txt test.c -c
# 比较 test.txt 和 test.c, 但忽略大小写
diff test.txt test.c -i
```

## 参考网址

### 1.3.11 Linux 命令——find

#### find

- **功能：**在指定目录下查找文件
- **提示：**
  - 任何位于参数之前的字符串都将被视为欲查找的目录名；
  - 如果使用该命令时，不设置任何参数，则 find 命令将在当前目录下查找子目录与文件，并且将查找到的子目录和文件全部进行显示；
- **选项：**
  - -name: 指定模式，大小写敏感；
  - -iname: 指定模式，大小写不敏感；
  - -type: 指定文件类型
  - -help: 显示帮助信息
  - -version: 显示版本信息
- **示例：**

```
# 列出当前目录及子目录下的所有文件和文件夹
find .

# 在主目录下，查找 test.txt 文件
find ~ -name test.txt
# 在主目录及其子目录下，查找 test.txt 文件
find ~ -name "test.txt"

# 在主目录下，查找以 txt 结尾的文件
find ~ -name "*.txt"
# 在主目录下，查找以 txt 结尾的文件【但忽略大小写】
find ~ -iname "*.txt"

# 根据【文件类型】查找
# 在主目录及其子目录下查找“普通文件”
find ~ -type f
# 在主目录及其子目录下查找“符号连接”
find ~ -type l
# 在主目录及其子目录下查找“目录”
find ~ -type d
```

(下页继续)

(续上页)

```
# 在主目录及其子目录下查找“字符设备”
find ~ -type c
# 在主目录及其子目录下查找“块设备”
find ~ -type b
# 在主目录及其子目录下查找“套接字”
find ~ -type s

# 显示帮助信息
find --help
# 显示版本信息
find --version
```

## 参考网址

### 1.3.12 Linux 命令——unalias

#### unalias

- 功能：取消命令别名 (shell 内建命令)
- 提示：
  - 如果需要取消任意一个命令别名，则使用该命令别名作为指令的参数选项即可；
  - 如果使用 -a 选项，则表示取消所有已经存在的命令别名；
- 选项：
  - -a：取消所有命令别名；
- 示例：

```
# 定义命令 xuezhisd, 显示登录用户信息
alias xuezhisd='who am i'
# 测试命令 xuezhisd 是否可用 (可用)
xuezhisd
# 删除命令别名 xuezhisd
unalias xuezhisd
# 测试命令 xuezhisd 是否可用 (不可用)
xuezhisd
```

## 参考网址

### 1.3.13 Linux 命令——uname

#### uname

- 功能：显示系统信息
- 提示：
  - 系统信息包括：内核版本号、硬件架构、主机名称和操作系统类型等；
- 选项：
  - -a 或 -all：显示全部的信息；



- m 或-machine: 显示机器硬件架构 (如 x86\_64);
- n 或-nodename: 显示主机名;
- o: 显示操作系统名称;
- r: 显示内核的发行编号;
- s: 显示内核名称;
- v: 显示内核的版本;
- help: 显示帮助;
- version: 显示版本信息。

- 示例:

```
# 显示内核名称 (如 Linux)
uname
uname -s
# 显示操作系统名称
uname -o
# 显示所有信息
uname -a
# 显示系统架构 (如 x86_64)
uname -m
# 显示主机名
uname -n
# 显示内核的发行编号
uname -r
# 显示内核的版本
uname -v
# 显示帮助
uname --help
# 显示版本
uname --version
```

## 参考网址

### 1.3.14 Linux 命令——useradd

#### useradd

- 功能: 建立用户账号
- 提示:
  - 使用 useradd 指令所建立的帐号, 实际上是保存在/etc/passwd 文本文件中;
  - 帐号建好之后, 再用 passwd 设定帐号的密码;
  - 可用 userdel 删除帐号;
  - 设定 ID 值时尽量要大于 500, 以免冲突。因为 Linux 安装后会建立一些特别用户, 一般 0 到 499 之间的值留给 bin、mail 这样的系统账号。
  - 系统用户的权限稍微大一些, 可以修改自己的某些信息; 系统用户的 UID<500;
- 选项:
  - c: 加上备注文字。备注文字会保存在 passwd 的备注栏位中;

- d: 指定用户登入时的启始目录。
- D: 变更预设值;
- e: 指定帐号的有效期限, 缺省表示永久有效;
- f: 指定在密码过期后多少天即关闭该帐号;
- g: 指定用户所属的群组。
- G: 指定用户所属的附加群组;
- m: 自动建立用户的登入目录;
- M: 不要自动建立用户的登入目录;
- r: 建立系统帐号;
- s: 指定用户登入后所使用的 shell;
- u: 指定用户 id;

• 示例:

```
# 新建用户 test
useradd test
# 新建用户 test, 并添加备注
useradd test -c "temp user"
# 新建用户 test, 并指定目录
useradd test -d /home/test
# 添加新用户 jack, 主要组:sales, 次要组: company
useradd jack -g sales -G company,employees
# 新建用户 test, 并自动建立用户登入目录
useradd test -m
# 新建系统用户 test
useradd test -r
# 新建一个用户, 并指定用户登录后使用的 shell
useradd test -s /bin/bash
# 建立一个新用户账户, 并设置 ID:
useradd test -u 544
```

## 参考网址

### 1.3.15 Linux 命令——userdel

#### userdel

- 功能: 删除用户账号
- 提示:
  - 一次删除一个用户;
  - 删除的用户必须存在;
  - userdel 不允许你移除正在线上的使用者帐号。你必须砍掉此帐号现在在系统上执行的程序才能进行帐号删除;
- 选项:
  - f: 强制删除用户, 即使用户当前已登录;
  - r: 删除用户的同时, 删除与用户相关的所有文件。

- 示例:

```
# 删除用户 test
userdel test
# 删除用户 test 的同时, 删除用户登录目录以及目录下的文件
userdel test -r
# 强制删除用户
userdel test -f
# 强制删除用户, 并删除用户登录目录以及目录下的文件
userdel test -fr

# 删除已经登录的用户
# 先查看当前登录的用户, 并记录 TTY
w
# 关闭希望删除的用户
pkill -kill -t [TTY]
userdel test -rf
```

## 参考网址

### 1.3.16 Linux 命令——visudo

#### visudo

- 功能: 编辑 sudoers 文件
- 提示:
  - 需要超级用户权限;
  - 默认编辑/etc/sudoers 文件;
  - sudoers 文件的默认权限是 440, 即默认无法修改;
  - visudo 可以在不更改 sudoers 文件权限的情况下, 直接修改 sudoers 文件;
- 选项:
  - -c: 使用 **check-only** 模式, 打印语法等错误信息;
  - -q: 使用 **quiet** 模式, 不打印语法等错误信息;
  - -f: 指定 sudoers 文件;
- 示例:

```
# 编辑/etc/sudoers 文件
visudo
# 编辑/etc/sudoers 文件, 并检查语法等错误
visudo -c
# 编辑/etc/sudoers 文件, 但不打印语法等错误
visudo -q
# 显示版本信息
visudo -V
```

## 参考网址

### 1.3.17 Linux 命令——whoami

#### whoami

- **功能：**打印当前有效的用户名
- **提示：**
  - 等价于执行 `id -un` 命令；
  - 如果使用 `su` 切换用户，打印切换后的用户名；
- **选项：**
  - `--help`：显示在线帮助；
  - `--version`：显示版本信息。
- **示例：**

```
# 显示帮助
whoami --help
# 显示版本信息
whoami --version
```

## 参考网址

### 1.3.18 Linux 命令——who

#### who

- **功能：**显示登录用户的信息
- **提示：**
  - 执行 `who` 命令可得知目前有那些用户登入系统；
  - 单独执行 `who` 命令会列出登入帐号，使用的终端机，登入时间和从何处登入；
- **选项：**
  - `-b`：显示系统上次启动时间
  - `-H` 或 `-heading`：显示头信息【标题栏】；
  - `-u`：显示闲置时间；
  - `-m`：显示当前用户的信息，等价于 `who am i`；
  - `-q` 或 `-count`：只显示登入系统的用户名和总人数；
  - `-r`：显示当前运行级别（runlevel）；
  - `-s`：此参数将忽略不予处理，仅负责解决 `who` 指令其他版本的兼容性问题；
  - `-w` 或 `-T`：显示用户的信息状态栏；
  - `--help`：在线帮助；
  - `--version`：显示版本信息；

- 示例:

```
# 显示系统上次启动时间
who -b
# 显示头信息【标题栏】
who -H
who --heading
# 突出显示“闲置时间”
# 若该用户在前一分钟之内有进行任何动作，将标示成"."号，如果该用户已超过 24 小时没有任何动作，则标示出
→ "old" 字符串
who -u
# 显示当前用户的信息
who -m
who am i
# 只显示登入系统的用户名和总人数
who -q
who --count
# 显示当前运行级别
who -r
# 显示用户的信息状态栏
who -w
who -T
# 显示在线帮助信息
who --help
# 显示版本信息
who --version
```

## 参考网址

### 1.3.19 Linux 命令——w

#### w

- 功能：显示已经登陆的用户列表，和他们正在执行的指令。
- 提示：
  - 单独执行 w 命令会显示所有的用户和他们执行的命令；
  - 您也可指定用户名称，仅显示某位用户和他正在执行的命令；
- 选项：
  - -h：不打印头信息；
  - -u：显示进程和 cpu 时间时，忽略用户名【su 切换到其它用户时，有区别】；
  - -s：使用短输出格式；
  - -f：显示用户从哪登录；
  - -V：显示版本信息。
- 示例：

```
# 显示所有用户的信息
w
# 显示用户 test 的信息
w test
```

(下页继续)

(续上页)

```
# 显示所有用户的信息, 但是不显示头信息 【标题栏】
w -h
# 显示短输出格式 【精简格式】
w -s
# 显示/不显示用户从哪里登录
w -f
# 显示版本
w -V
```

```
[xuezhisd@moban home]$ su -
Password:
[root@moban ~]# w
 20:13:37 up 1:02, 2 users, load average: 0.29, 0.11, 0.04
USER      TTY      FROM          LOGIN@      IDLE        JCPU       PCPU WHAT
xuezhisd pts/0     192.168.154.1 19:12       0.00s      0.40s      0.26s sshd: xuezhisd [priv]
test      pts/1     192.168.154.1 19:49       24:32      0.02s      0.02s -bash
[root@moban ~]# w -u
 20:13:47 up 1:03, 2 users, load average: 0.24, 0.11, 0.04
USER      TTY      FROM          LOGIN@      IDLE        JCPU       PCPU WHAT
xuezhisd pts/0     192.168.154.1 19:12       0.00s      0.41s      0.01s w -u
test      pts/1     192.168.154.1 19:49       24:42      0.02s      0.02s -bash
```

- 参数 u 举例说明:

```
[xuezhisd@moban home]$ w
 20:17:16 up 1:06, 2 users, load average: 0.07, 0.06, 0.02
USER      TTY      FROM          LOGIN@      IDLE        JCPU       PCPU WHAT
xuezhisd pts/0     192.168.154.1 19:12       0.00s      0.42s      0.02s w
test      pts/1     192.168.154.1 19:49       28:11      0.02s      0.02s -bash
[xuezhisd@moban home]$ w -s
 20:17:19 up 1:06, 2 users, load average: 0.07, 0.06, 0.02
USER      TTY      FROM          IDLE WHAT
xuezhisd pts/0     192.168.154.1 0.00s w -s
test      pts/1     192.168.154.1 28:14 -bash
```

- 参数 s 举例说明:

## 参考网址

### 1.3.20 Ubuntu 14.04 64bit 安装 32 位兼容包

#### 问题描述

- 我的操作系统是 64bit 版的 Ubuntu 14.04, 很多 32bit 的软件无法安装使用, 因此希望通过安装 32 位兼容包解决。

## 安装过程

- 在 Ubuntu 13.10 之前, 可以通过安装 **ia32-libs** 来支持 32 位软件。但是, 该版本之后, Ubuntu 就去掉了。
- 不过可以通过安装替的 32 位兼容包来实现该功能。
- 命令行中输入以下命令

```
sudo apt-get install ia32-libs
```

返回的信息如下所示,

```
Package ia32-libs is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source
However the following packages replace it:
  lib32z1 lib32ncurses5 lib32bz2-1.0
```

- 也就是说可以通过 lib32z1 lib32ncurses5 lib32bz2-1.0 是 ia32-libs 的替代安装包。可以安装任何一个来支持 32 位软件。
- 安装命令如下所示。

```
sudo apt-get install lib32z1
```

## 参考

### 1.3.21 Ubuntu14.04 安装 Theano 详细教程

因为最近需要学习深度学习, 因此想要配置 Theano, 来开发深度学习算法。但是发现 Theano 安装总是出现问题。于是在这里中总结一下。

## 环境

- 操作系统: ubuntu14.04
- Python: 2.7.6
- 需要联网

## 相关库简介

- **BLAS** (Basic Linear Algebra Subprograms) 是基础线性代数子程序库, 里面拥有大量已经编写好的关于线性代数运算的程序;
- **LAPACK** (Linear Algebra PACKage) 包含了求解科学与工程计算中最常见的数值线性代数问题, 如求解线性方程组、线性最小二乘问题、特征值问题和奇异值问题等;
- **ATLAS** 是 python 下的一个线性代数库, 是基于另外两个线性代数库 BLAS 和 lapack 的;
- **NumPy** 提供了一个在 python 中做科学计算的基础库, 它重在数值计算, 甚至可以说是用于多维数组处理的库;
- **SciPy** 是基于 numpy, 提供了一个在 python 中做科学计算的工具集, 也就是说它是更上一个层次的库;
- **Theano** 则是基于 NumPy 以及 SciPy 的一个更高级的用于科学计算的库。

## 相关库的关系

- 要安装 Theano，就需要先安装好 numpy 和 scipy；
- 要安装 numpy 和 scipy，就需要 ATLAS；
- 要安装 ATLAS，就需要安装 BLAS 和 LAPACK；

## 相关库的安装顺序

- 安装顺序: *BLAS*  $\rightarrow$  *LAPACK*  $\rightarrow$  *ATLAS*  $\rightarrow$  *numpy*  $\rightarrow$  *scipy*  $\rightarrow$  *Theano*

## 检查 numpy 和 scipy 是否通过测试

- 说明：如果你的 `numpy` 和 `scipy` 是通过 `apt-get` 安装的，那么它们的单元测试可能会通不过!!!（我在安装过程中没有通过测试，`scipy` 出现了 **Error**）
- 如果 `numpy` 或 `scipy` 不能通过测试，就需要卸载，然后重新按照本文介绍的顺序安装。

```
# 检查 numpy 是否通过测试
python -c "import numpy;numpy.test()"
```

- 如果 `numpy` 通过测试，会出现如下图所示的结果。注意最后一行 `errors=0 failures=0`。如果没有通过测试，需要卸载并重新安装。

```

.....K.....
-----
Ran 5593 tests in 26.163s

OK (KNOWNFAIL=5, SKIP=14)
<nose.result.TextTestResult run=5593 errors=0 failures=0>
>>>

```

- 如果 `scipy` 通过测试, 会出现如下图所示的结果。注意最后一行 **`errors=0 failures=0`**。如果没有通过测试, 需要卸载并重新安装。



```
.....S.....  
.....S...S.....  
.....S.S.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....K.....  
.....  
.....  
-----  
-----  
Ran 18456 tests in 213.386s  
  
OK (KNOWNFAIL=98, SKIP=1182)  
<nose.result.TextTestResult run=18456 errors=0 failures=0>  
>>>
```

- 注意: `scipy` 很可能会出现测试错误, 比如我安装过程中出现下图所示的错误。

```

=====
ERROR: test_fitpack.TestSplder.test_kink
-----
-----
Traceback (most recent call last):
  File "/usr/lib/python2.7/dist-packages/nose/case.py", line
197, in runTest
    self.test(*self.arg)
  File "/usr/lib/python2.7/dist-packages/scipy/interpolate/te
sts/test_fitpack.py", line 329, in test_kink
    splder(spl2, 2) # Should work
  File "/usr/lib/python2.7/dist-packages/scipy/interpolate/fit
pack.py", line 1186, in splder
    "and is not differentiable %d times") % n)
ValueError: The spline has internal repeated knots and is not
differentiable 2 times
-----
-----
Ran 8936 tests in 114.328s

```

## 卸载 numpy 和 scipy

```
# 卸载 numpy
sudo apt-get remove python-numpy

# 卸载 scipy
sudo apt-get remove python-scipy
```

## 安装各种包

```
# 安装 gfortran, 后面编译过程中会用到
sudo apt-get install gfortran
# 安装 blas, Ubuntu 下对应的是 libopenblas, 其它操作系统可能需要安装其它版本的 blas——这是个 OS 相关的。
sudo apt-get install libopenblas-dev
# 安装 lapack, Ubuntu 下对应的是 liblapack-dev, 和 OS 相关。
sudo apt-get install liblapack-dev
# 安装 atlas, Ubuntu 下对应的是 libatlas-base-dev, 和 OS 相关。
sudo apt-get install libatlas-base-dev
```

## 安装 numpy 和 scipy

- 使用 pip 安装 numpy 和 scipy
- 安装 pip 的命令: **sudo apt-get install python-pip**
- 注意: 一定要在安装完 lapack/blas 之后, 再安装 numpy 和 scipy。否则, 会出现错误 **\*\*\*no lapack/blas resources found\*\*\***
- 安装 numpy 和 scipy 的命令如下所示。

```
# 安装 numpy
sudo pip install numpy
# 测试 numpy
# 测试通过才能进行下一步 ~ ~
python -c "import numpy; numpy.test()"

# 安装 scipy
sudo pip install scipy
# 测试 scipy
# 测试通过才能进行下一步 ~ ~
python -c "import scipy; scipy.test()"
```

## 安装线性计算库、numPy 和 sciPy 的编译方法（推荐）

### 编译安装 OpenBlas

- 为什么安装 OpenBLAS? 因为 OpenBLAS 的速度比 atlas 快。速度对比请参考: 《[Benchmark OpenBLAS, Intel MKL vs ATLAS](#)》
- 下载 OpenBLAS。github 下载地址
- 安装 OpenBLAS

```
tar -zxvf OpenBLAS**.tar.gz
cd OpenBLAS**
make -FC gfortran # 需要已经安装 gfortran
make install # 安装在/opt/OpenBLAS/目录下
```

## 安装 numPy

- 下载 numPy 。 [github](#) 豆瓣镜像
- 解压下载包，并配置 OpenBLAS。

```
tar -zxvf numpy**.tar.gz
cd numpy**
cp site.cfg.example site.cfg
vi site.cfg
# 将以下 4 行去掉注释。【101 行--104 行】
#[openblas]
#libraries = openblas
#library_dirs = /opt/OpenBLAS/lib
#include_dirs = /opt/OpenBLAS/include
```

- 安装 numPy。

```
sudo python setup.py config
sudo python setup.py build
sudo python setup.py install
```

## 安装 sciPy

- 下载 sciPy 。 [github](#) 豆瓣镜像
- 解压下载包，并配置 OpenBLAS。

```
tar -zxvf scipy**.tar.gz
cd scipy**
# 将 numPy** 中的配置文件复制到此处
cp ../numpy**/site.cfg.example ./site.cfg
```

- 安装 sciPy。

```
sudo python setup.py config
sudo python setup.py build
sudo python setup.py install
```

## 安装其它库

- 为了安装 Theano, 最后还需要安装一些库, 可以参考官方教程

```
sudo apt-get install python-dev
sudo apt-get install python-pip
sudo apt-get install python-nose
sudo apt-get install g++
sudo apt-get install git
```

## 安装 Theano

- 前面的操作如果没有出现错误, 就可以开始安装 Theano 了。命令如下所示。

```
# 安装 Theano
sudo pip install Theano

# 测试 Theano
python -c "import theano;theano.test()"
```

## 引用

- ATLAS + NumPy + SciPy + Theano 的 python 科学计算环境搭建
- Compiling numpy with OpenBLAS integration
- 依赖关系
- 官方安装教程
- <http://stackoverflow.com/questions/7496547/does-python-sciPy-need-blas>
- <http://www.linuxidc.com/Linux/2014-10/107503.htm>
- <http://www.scipy.org/scipylib/building/linux.html#installation-from-source>

## 1.3.22 Ubuntu Server 14.04 安装 Gnome 桌面环境

### 操作

执行以下命令:

```
sudo apt-get install xinit
sudo apt-get install gdm
sudo apt-get update # 为下一条命令准备, 否则下一条命令执行错误。
sudo apt-get install ubuntu-desktop
```

**参考:**

- <http://mikewolffi.blog.163.com/blog/static/860452882014118103755772/>
- <http://blog.csdn.net/sunbaigui/article/details/6624110>
- <http://jingyan.baidu.com/album/64d05a0262b613de55f73b0e.html?picindex=5>

### 1.3.23 Ubuntu Server 添加、删除和修改用户

#### 添加用户

- 添加用户的命令如下所示。添加一个用户名为 sam 的用户，主目录是/home/sam。

```
# -d 指定用户目录
# -m 创建用户目录 (如果该目录不存在)
# useradd -d /home/sam -m sam
```

#### 修改密码

- root 用在终端中运行以下命令，为 sam 用户重新设置密码。

```
sudo passwd sam
```

- 设置 root 密码。root 用户初始密码是随机的。按照下面的命令可以重新设置固定的 root 密码。

```
sudo passwd root
```

#### 删除用户

- root 用在终端中运行以下命令，将删除 sam，但不删除其主目录。

```
sudo userdel sam
```

#### 添加超级用户权限

- 修改/etc/sudoers 文件，将 sam 添加到超级用户列表中，可以将 sam 设置成超级用户。
  - 执行下面的命令，使用 vi 打开 sudoers 文件；

```
vi /etc/sudoers
```

- 在 root ALL=(ALL) ALL 后面添加新的一行 sam ALL=(ALL) ALL，保存并退出，添加超级用户权限完成。
- sam 具有了超级用户权限，就可以具有 root 所有权限。执行命令式，使用 sudo 开头，并输入 sam 的密码，即能使用 root 权限执行命令。

## 1.3.24 ubuntu 中 apt-get 安装与默认路径

### 一、apt-get 安装

- deb 是 debian linux 的安装格式，跟 red hat 的 rpm 非常相似，最基本的安装命令是：dpkg -i file.deb 或者直接双击此文件
- dpkg 是 Debian Package 的简写，是为 Debian 专门开发的套件管理系统，方便软件的安装、更新及移除。所有源自 Debian 的 Linux 发行版都使用 dpkg，例如 Ubuntu、Knoppix 等。

以下是一些 dpkg 的普通用法：

#### dpkg -i

- 安装一个 Debian 软件包，如你手动下载的文件。

#### dpkg -c

- 列出的内容。

#### dpkg -l

- 从中提取包裹信息。

#### dpkg -r

- 移除一个已安装的包裹。

#### dpkg -P

- 完全清除一个已安装的包裹。和 remove 不同的是，remove 只是删掉数据和可执行文件，purge 另外还删除所有的配制文件。

#### dpkg -L

- 列出安装的所有文件清单。同时请看 dpkg -c 来检查一个.deb 文件的内容。

#### dpkg -s

- 显示已安装包裹的信息。同时请看 apt-cache 显示 Debian 存档中的包裹信息，以及 dpkg -l 来显示从一个.deb 文件中提取的包裹信息。

## dpkg-reconfigure

- 重新配制一个已经安装的包裹，如果它使用的是 debconf (debconf 为包裹安装提供了一个统一的配制界面)。

## 二、软件安装后相关文件位置

### 下载的软件存放位置

- /var/cache/apt/archives

### 安装后软件默认位置

- /usr/share

### 可执行文件位置

- /usr/bin

### 配置文件位置

- /etc

### lib 文件位置

- /usr/lib

## 1.3.25 Ubuntu 使用过程中遇到的问题

### 打开终端的方法

#### 1. 快捷键

- 打开终端快捷键: [Ctrl] + [Alt] + [T]

#### 2. Dash 主页图标

- 在 Dash 主页中搜索 terminal, 即可找到终端。

## 打开 Dash 的快捷键

- windows 键

## 将./添加到 PATH

### 目的

编写的脚本文件 test.py，赋予执行权限之后，每一次都需下面的命令来执行。想要省去 ./

```
./test.py
```

变成

```
test.py
```

### 1. 临时方法

在终端中即可修改，但是重新登陆后失效。

执行如下命令即可。

```
export PATH="$PATH:."
```

### 2. 修改 environment 文件

- 修改/etc/environment 文件，可以永久有效。
- 需要 root 权限
- 需要重启电脑生效
- 添加在其他路径的前面

```
sudo vi /etc/environment
```

输入密码，修改 PATH。修改前：

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/  
↪local/games"
```

修改后：

```
PATH=".:usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/  
↪local/games"
```

注意：. 添加在**最前面**。

- 输入以下命令，使配置立刻生效，避免重启电脑。

```
source /etc/environment
```



### 3. 修改 ~/.bashrc 文件

- 该文件编辑保存后，可立即在新打开的终端窗口内生效。
- 该方式只对当前用户有效。
- 在该文件的最后添加如下语句

```
export PATH=.:$PATH
```

### 4. 修改 profile 文件

- 文件位置：**/etc/profile**
- 该文件编辑保存后，重启系统，变量生效。
- 该方式添加的变量对所有的用户都有效。
- 在文件的最后面追加：

```
export PATH=.:$PATH
```

### 安装软件源码的步骤

分为 3 个步骤

```
./configure
make
make install
make clean # 清理临时文件
```

参考

### vim 不显色的问题

- 打开 vim 查看加载那些脚本。底行模式下输入:scriptnames。
- 设置.vimrc 文件，将该文件设置成以下内容。

```
set nocompatible          " Vim settings, not Vi settings.  must be first
set autoindent             " Auto align when insert new line, for instance, when using o↵
↵or O to insert new line.
set ruler                 " Show ruler at the bottom-right of vim window
set showcmd
set backspace=indent,eol,start      " Enable delete for backspace under insert↵
↵mode"
colorscheme darkblue
set number                " Show line number
syntax on
if &term =~ "xterm"
  if has("terminfo")
    set t_Co=8
    set t_Sf=^[[3%p1%dm
    set t_Sb=^[[4%p1%dm
  else
```

(下页继续)

(续上页)

```
set t_Co=8
set t_Sf=^[[3%dm
set t_Sb=^[[4%dm
endif
endif
```

参考文章

### vim 不能正常使用方向键与退格键的问题

- vim 出现问题的原因可能是由于版本低
- 在 7.3 中出现这些问题，但是在 vim7.4 中没有这些问题。## 方法 1
- 安装 vim full 版本。full 版本下键盘正常。## 方法 2
- 在 vi 底行模式下，使用命令 \*\*: set nocompatible\*\*，就是设置 vi 不使用兼容模式。

参考文章

### 如何是配置立即生效

- 使配置立即生效的命令 \*\*source\*\*
- 好处：不用重启电脑
- 示例 1：

```
# 使 environment 修改后立即生效
source /etc/environment
```

- 示例 2：

```
# 使 profile 修改后立即生效
source /etc/profile
```

## 1.3.26 Ubuntu-安装-Chrome

### 一、下载 Chrome 的安装包

- 打开终端，输入一下命令（区分 64 位和 32 位），下载 Chrome 的 deb 安装包。

```
#32 位
wget https://dl.google.com/linux/direct/google-chrome-stable_current_i386.deb
#64 位
wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
```

## 二、安装 Chrome

- 注意：需要 root 权限
- 在 Terminal 中输入以下命令

```
#32 位安装命令
sudo dpkg -i google-chrome-stable_current_i386.deb
#64 位安装命令
sudo dpkg -i google-chrome-stable_current_amd64.deb
```

## 三、修复安装错误

- 安装过程中会提示错误信息“在处理时有错误发生：google-chrome-stable”
- 英文错误：

```
dpkg: error processing package google-chrome-stable (--install):
dependency problems - leaving unconfigured
```

- 出现这个错误之后，执行下面的命令

```
sudo apt-get -f install
```

- 执行完成后，Chrome 就安装成功了。
- 在 Dash 中搜索 Chrome，打开 Chrome，并绑定到启动栏（launcher）

## 四、参考文章：

- <http://www.linuxidc.com/Linux/2014-04/100645.htm>
- <http://jingyan.baidu.com/article/a681b0de18071e3b1843463b.html>

## 1.3.27 Ubuntu-安装-cuda7.0-单显卡-超详细教程

### 一、说明

- 本教程是在台式机上安装的，只有一个 NVIDIA 显卡。
- 操作系统是 Ubuntu 14.04 (64bit)。
- 双显卡的笔记本请移步[Ubuntu-安装-cuda7.0-双显卡-超详细教程](#)

### 二、准备

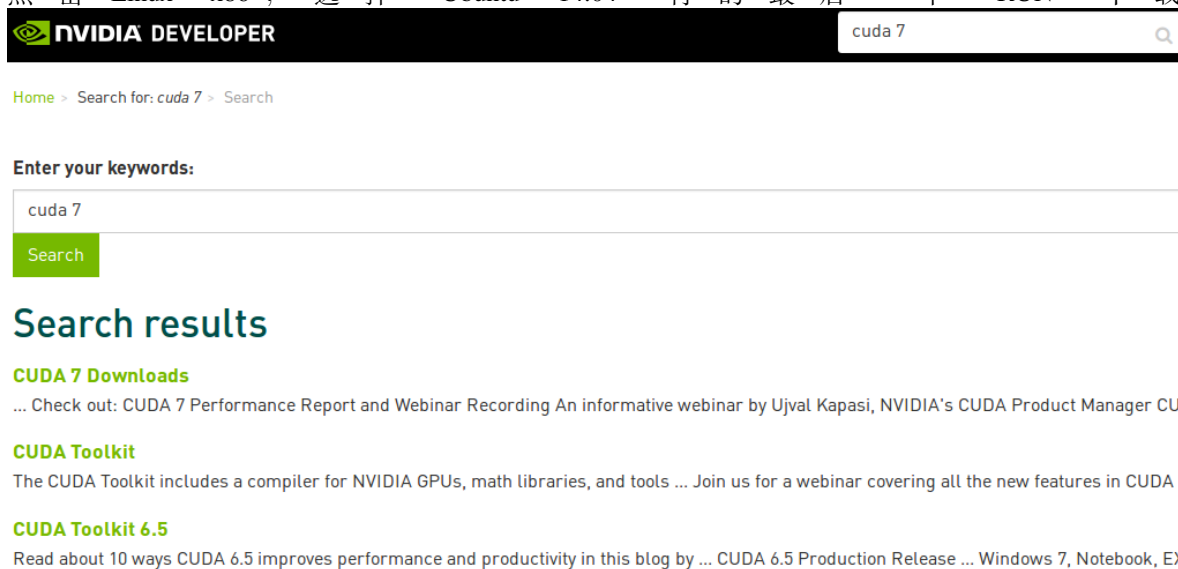
说明：本文假设下载的文件都在 ~/Downloads/下面

## 1. 更新操作系统

```
sudo apt-get update
```

## 2. 下载 cuda7.0

- 下载 cuda。点此[下载](#)
- 如果不是该版本，可以搜索，如下图所示；
- 点击”Linux x86”，选择 “Ubuntu 14.04” 行的最后一个 “RUN” 下载；



Windows	Linux x86	Linux POWER8	Mac OSX
Version	Network Installer	Local Package Installer	Runfile Installer
Fedora 21	<a href="#">RPM</a> (3KB)	<a href="#">RPM</a> (1GB)	<a href="#">RUN</a> (1.1GB)
OpenSUSE 13.2	<a href="#">RPM</a> (3KB)	<a href="#">RPM</a> (1GB)	<a href="#">RUN</a> (1.1GB)
OpenSUSE 13.1	<a href="#">RPM</a> (3KB)	<a href="#">RPM</a> (1GB)	<a href="#">RUN</a> (1.1GB)
RHEL 7 CentOS 7	<a href="#">RPM</a> (10KB)	<a href="#">RPM</a> (1GB)	<a href="#">RUN</a> (1.1GB)
RHEL 6 CentOS 6	<a href="#">RPM</a> (18KB)	<a href="#">RPM</a> (1GB)	<a href="#">RUN</a> (1.1GB)
SLES 12	<a href="#">RPM</a> (3KB)	<a href="#">RPM</a> (1.1GB)	<a href="#">RUN</a> (1.1GB)
SLES 11 (SP3)	<a href="#">RPM</a> (3KB)	<a href="#">RPM</a> (1.1GB)	<a href="#">RUN</a> (1.1GB)
<a href="#">SteamOS 1.0-beta</a>			<a href="#">RUN</a> (1.1GB)
Ubuntu 14.10	<a href="#">DEB</a> (3KB)	<a href="#">DEB</a> (1.5GB)	<a href="#">RUN</a> (1.1GB)
Ubuntu 14.04*	<a href="#">DEB</a> (10KB)	<a href="#">DEB</a> (902MB)	<a href="#">RUN</a> (1.1GB)

### 3. 下载 NVIDIA 显卡驱动

- 下载驱动。点此下载
- 选择自己电脑对应的配置，然后点击“搜索”；

#### NVIDIA 驱动程序下载

选项 1: 手动查找适用于我的 NVIDIA 产品的驱动程序。

产品类型:

产品系列:

产品家族:

操作系统:

语言:

- 点击“DOWNLOAD”，进行下载；

## LINUX X64 (AMD64/EM64T) DISPLAY DRIVER

**Version:** 352.30  
**Release Date:** 2015.7.28  
**Operating System:** Linux 64-bit  
**Language:** English (US)  
**File Size:** 74.07 MB

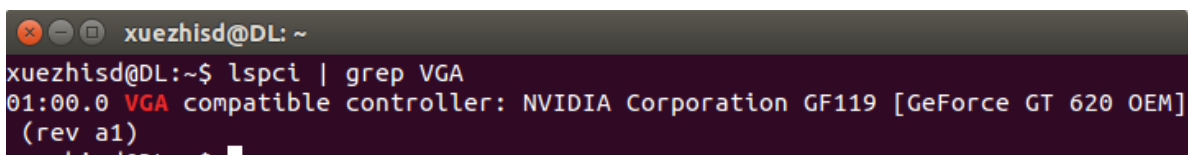
DOWNLOAD

### 三、验证系统是否符合要求

#### 1. 验证显卡是否支持 cuda

- 终端输入命令，如图所示。如果只返回两条信息，即有两个显卡，可以继续下面教程，否则请移步《Ubuntu-安装-cuda7.0-双显卡-超详细教程》

```
lspci | grep VGA
```



```
xuezhisd@DL: ~  
xuezhisd@DL:~$ lspci | grep VGA  
01:00.0 VGA compatible controller: NVIDIA Corporation GF119 [GeForce GT 620 OEM]  
(rev a1)
```

- 如果操作系统是 Ubuntu Server，即使有 NVIDIA 显卡，执行上面的命令可能也不会返回相关信息。这时需要执行的命令是 `lspci |grep nvidia -i`，就能返回正常的结果。
- 终端会显示显卡型号，在 <https://developer.nvidia.com/cuda-gpus> 中查找，是否有你的显卡型号，只要存在，就表明显卡支持 cuda，同时还可以查看显卡的计算能力，数值越大越好。

#### 2. 查看系统类型

- 命令行输入下面命令；

```
uname -m && cat /etc/*release
```

- 第一行显示有 x86\_64，说明系统是 x86 构架 64 位系统；

```
xuezhisd@DL:~$ uname -m && cat /etc/*release
x86_64
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.2 LTS"
NAME="Ubuntu"
VERSION="14.04.2 LTS, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 14.04.2 LTS"
VERSION_ID="14.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
xuezhisd@DL:~$
```

#### 四、安装 NVIDIA 显卡驱动

说明：前面下载的 cuda7.0\*.run 中含有 NVIDIA 显卡驱动，而且安装时会提示安装驱动。但是，由于 Ubuntu 中已经为 NVIDIA 显卡安装了驱动 nouveau，而且该驱动正在运行中，所以安装驱动的过程稍微麻烦，建议先单独安装显卡驱动，再安装 cuda

##### 1. 切换到 tty1

- 说明，视窗会使用显卡，如果在视窗下，原来的驱动 nouveau 不会停止，也就不能安装成功
- 按组合键” Ctrl+Alt+F1 “，切换到 tty1；

## 2. 关闭显示器管理器

- 输入命令 `sudo stop lightdm`, 关闭显示器管理器。否则后面旧的显卡驱动无法禁用, 新的显卡驱动无法安装。

## 3. 禁用旧的显卡驱动

- 注意: 禁用旧的显卡驱动这一步可以不操作, 后面安装新的 NVIDIA 显卡驱动时, 会提示该问题; 那时选择相应的选项即可创建该文件。
- 禁用旧的显卡驱动; 切换到 `/etc/modprobe.d/`, 新建文件 `nvidia-installer-disable-nouveau.conf`, 输入以下内容, 保存退出;

```
blacklist nouveau
options nouveau modeset=0
```

- 在文件 `/etc/default/grub` 的最后, 添加一行, 如下所示;

```
rdblacklist nouveau
nouveau.modeset=0
```

## 4. 安装新的驱动

- 小技巧: 如果按照第 3 步做完了, 重启后依然无法安装新的 NVIDIA 显卡驱动, 那么可以通过 `software manger` 安装 NVIDIA 低版本的显卡驱动 (它就会自动关闭 `nouveau`), 重启后卸载低版本显卡驱动, 接着安装当前版本的显卡驱动。这样一般情况下可以成功。(曲线救国)
- 执行以下命令安装

```
# 切换到驱动所在的路径
cd ~/Downloads
# 查看驱动的名称 (ls)
ls
# 开始安装, 注意将驱动名换成你下载的驱动的名称
sudo sh ./NVIDIA-Linux-x86_64-352.30.run
```

- 安装过程中, 根据提示, 选择 `accept`, ‘yes’ 或默认选项
  - 选项: `Accept` → `Continue` → `installation` → `OK` → `OK` → `OK`
- 当提示重启时, 重启继续。(如果前面没有禁用 `nouveau`, 会提示向 `/etc/modprobe.d/` 路径下添加文件, 其实是在尝试禁用 `nouveau`。如果出现该提示, 需要重启电脑之后, 再次运行显卡安装程序, 重新开始安装一遍。)
- 重启后, 切换到 `tty` (`Ctrl+Alt+F1`), 关闭显示器管理 (`sudo stop lightdm`), 重新运行安装驱动命令 (`sudo sh ./NVIDIA-Linux-x86_64-352.30.run`)
- 当提示 `***would you run nvidia-Xconfig utility ...**`, 选择 `Yes`。



## 5. 检验驱动是否安装成功

- 打开终端，输入以下命令：

```
cat /proc/driver/nvidia/version
```

- 如果显示驱动版本，说明安装成功。如下图所示。

```
xuezhisd@DL:~/Downloads$ cat /proc/driver/nvidia/version
NVRM version: NVIDIA UNIX x86_64 Kernel Module  352.30  Tue Jul 21 18:53:45 PDT
2015
GCC version:  gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)
```

## 五、安装 cuda

注意，请在成功安装显卡驱动的前提下继续安装 *cuda*，否则也会安装失败。

### 1. 安装依赖库

- 请先在 Terminal 中安装以下依赖库：

```
sudo apt-get install freeglut3-dev
sudo apt-get install build-essential
sudo apt-get install libx11-dev
sudo apt-get install libxmu-dev
sudo apt-get install libxi-dev
sudo apt-get install libglu1-mesa
sudo apt-get install libglu1-mesa-dev
```

### 2. 运行 cuda 安装命令

- 在 Terminal 中运行以下命令，开始安装 *cuda*：

```
# 注意，将 cuda 名称换成自己下载的名称
# 添加执行权限
chmod a+x cuda_7.0.28_linux.run
sudo ./cuda_7.0.28_linux.run
```

- 安装过程中会提示安装 **NVIDIA 驱动**，OpenGL，CUDA 安装包和 SAMPLE 包。此处只安装后面三个。由于前面已经安装了最新版的 NVIDIA 驱动，此处不用再安装。
- 安装过程中，根据提示，选择 *accept*，‘yes’ 或默认选项；
- *\*\*...symbolic link ...*选项选择 *Yes*，否则没有 */usr/local/cuda/*，只有 */usr/local/cuda7.0/*
- 安装完成之后，启动显示器管理 (`sudo start lightdm`)，切换回视窗界面。

### 3. 配置环境变量

- 打开终端，在文件/etc/profile 的最后添加以下内容：

```
PATH=/usr/local/cuda/bin:$PATH
export PATH
```

- 保存后, 执行下列命令, 使环境变量立即生效

```
source /etc/profile
```

- 在 /etc/ld.so.conf.d/新建文件 cuda.conf，并添加如下内容：

```
/usr/local/cuda/lib64
```

- 执行下列命令使之立刻生效:

```
sudo ldconfig
```

### 4. 检验环境是否设置好

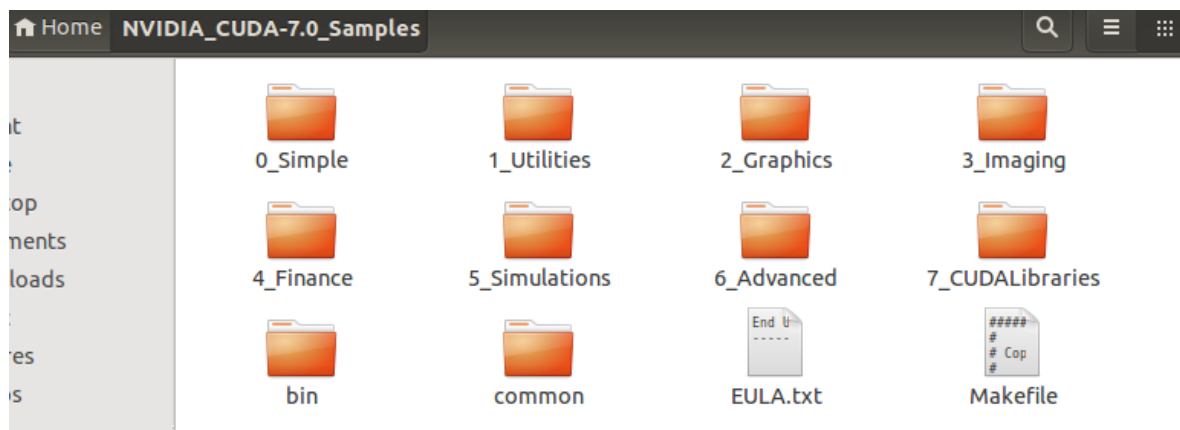
- 打开终端，输入 cuda，按 2 次” Tab 键 “，如果有弹出的命令提示，就说明环境配置成功。

### 5. 安装其他一些辅助库

```
sudo apt-get install freeglut3-dev build-essential libx11-dev libxmu-dev libxi-dev_
↪ libgl1-mesa-glx libglu1-mesa libglu1-mesa-dev
```

## 六、安装 CUDA SAMPLES

- 安装 cuda 过程中, cuda samples 默认安装了, 并且存储在当前用户目录下 (/home/xuezhisd/NVIDIA\_CUDA-7.0\_Samples), 如下图所示:



## 1. 验证 nvcc

- 输入命令 `*nvcc --version*`, 如果已经安装了, 会显示版本号; 如果没有安装, 按照提示完成安装。

## 2. make cuda samples

- 切换到 `cuda-samples` 路径, 使用 `***make***` 命令:

```
# 切换到 cuda-samples 所在目录
# 注意, 换成自己的路径
cd /home/xuezhisd/NVIDIA_CUDA-7.0_Samples
# 编译 make (安装命令 sudo apt-get install cmake)
make
# 编译完毕, 切换 release 目录
cd ./bin/x86_64/linux/release
# 检验是否成功
# 运行实例 ./deviceQuery
./deviceQuery
```

- 显示如下结果, 说明成功。

```
Detected 1 CUDA Capable device(s)

Device 0: "GeForce GT 620"
  CUDA Driver Version / Runtime Version      7.5 / 7.0
  CUDA Capability Major/Minor version number: 2.1
  Total amount of global memory:              1021 MBytes (10702684
  ( 1) Multiprocessors, ( 48) CUDA Cores/MP: 48 CUDA Cores
  GPU Max Clock rate:                        1620 MHz (1.62 GHz)
  Memory Clock rate:                         897 Mhz
  Memory Bus Width:                          64-bit
  L2 Cache Size:                             65536 bytes
  Maximum Texture Dimension Size (x,y,z)      1D=(65536), 2D=(65536
  3D=(2048, 2048, 2048)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 laye
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 20
  Total amount of constant memory:            65536 bytes
  Total amount of shared memory per block:     49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                  32
  Maximum number of threads per multiprocessor: 1536
```

```
Maximum number of threads per multiprocessor: 1536
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): (65535, 65535, 65535)
Maximum memory pitch: 2147483647 bytes
Texture alignment: 512 bytes
Concurrent copy and kernel execution: Yes with 1 copy engine
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
Device supports Unified Addressing (UVA): Yes
Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with d
ltaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.5, CUDA Run
on = 7.0, NumDevs = 1, Device0 = GeForce GT 620
Result = PASS
```

## 七、安装 cuDNN

### 1. 下载 cuDNN

- 官方下载 需要注册和验证，麻烦。
- 我将我下载文件上传到百度网盘中了，有需要的可以点击此处直接下载 [cudnn & cudnn-samples](#)

### 2. 安装 cuDNN

- 解压，并复制文件

```
# 解压文件
tar -zxvf cudnn-6.5-linux-x64-v2.tgz
# 切换路径
cd cudnn-6.5-linux-x64-v2
# 复制 lib 文件到 cuda 安装路径下的 lib64/
sudo cp lib* /usr/local/cuda/lib64/
# 复制头文件
sudo cp cudnn.h /usr/local/cuda/include/
```

- 更新软连接

```
cd /usr/local/cuda/lib64/
sudo rm -rf libcudnn.so libcudnn.so.6.5
sudo ln -s libcudnn.so.6.5.48 libcudnn.so.6.5
sudo ln -s libcudnn.so.6.5 libcudnn.so
```

至此，cuDNN 完成安装。

## 八、参考文章

- Ubuntu 13.04 双显卡安装 NVIDIA GT 630M 驱动
- Caffe + Ubuntu 14.04 64bit + CUDA 6.5 配置说明
- Ubuntu14.04+cuda6.5+opencv2.4.9+MATLAB2013a+caffe 配置记录 (二)
- ERROR: Module nouveau is in use
- HOW-TO: Unload nouveau and Install Nvidia Driver
- Caffe 在 Ubuntu 14.04 64bit 下的安装
- ubuntu14.04LTS+opencv2.4.9+matlab 2014a+caffe+cuDNN
- Optimus 双显卡用 Bumblebee3.0 在 ubuntu12.04 下配置 CUDA4.2
- Ubuntu 14.04 Nvidia 显卡驱动安装及设置
- ubuntu 11.10 驱动 gtx 460 显卡禁用 nouveau
- ERROR: 启动过程中, 会出现两个错误, 执行以下命令可以解决。

```
sudo apt-get purge nvidia-304
sudo apt-get install nvidia-331
```

### 1.3.28 Ubuntu-安装-Python 包

- Ubuntu 下安装 Python 包可以使用两种方法, 一种是用 `apt-get`, 另一种是使用 `pip`。

#### apt-get 安装示例

```
sudo apt-get install python-numpy
sudo apt-get install python-matplotlib
sudo apt-get install python-scipy
sudo apt-get install python-sklearn
sudo apt-get install python-setuptools
sudo apt-get install python-pip
sudo apt-get install python-jinja2
sudo apt-get install python-PIL
sudo apt-get install python-dev
sudo apt-get install ipython
```

#### pip 安装示例

```
sudo pip install nolearn
sudo pip install Theano
```

## 1.3.29 Ubuntu-安装-Shadowsocks

### 系统环境

- Ubuntu 14.04
- Python2.7

### 安装依赖关系

```
sudo apt-get update
sudo python --version
sudo apt-get install python-gevent python-pip
sudo pip install shadowsocks
```

### 找到 shadowsocks 的安装位置

```
sudo find / -name shadowsocks*
```

其中，“/”是根目录下，\*是通配符。我的安装路径是/usr/local/lib/python2.7/dist-packages/shadowsocks

### 新建并修改配置文件

在安装路径下新建文件 config.json，路径/usr/local/lib/python2.7/dist-packages/shadowsocks，创建命令 `sudo touch config.json`

文件内容：

```
{
  "server": "79.45.115.110",
  "server_port": 12801,
  "local_port": 10808,
  "password": "123456",
  "timeout": 600,
  "method": "aes-256-cfb"
}
```

### 运行 shadowsocks

在安装路径下，输入 `sslocal`，即可启动 shadowsocks。

## 开机自动启动

- 修改 rc.local 文件

```
cd /etc/  
sudo vim rc.local
```

- 加上一行:

```
/usr/local/bin/sslocal -c /usr/local/lib/python2.7/dist-packages/shadowsocks/config.  
→json
```

## 配置 SwitchySharp

- 下载 Switchysharp
- 安装 Switchysharp
- 配置 如下图所示, 注意: 端口根据根据配置中填写, 上面的配置是 10808, 所以此处端口应该由 1080 改为: 10808

The screenshot shows the SwitchySharp configuration interface. On the left, under '所有情景模式' (All Scenarios), 'SSH' is selected. Below it is a '新建情景模式' (New Scenario Mode) button. The main area is '详细配置' (Detailed Configuration). Under '手动配置' (Manual Configuration), the 'SOCKS 代理' (SOCKS Proxy) is set to '127.0.0.1' and the '端口' (Port) is '1080'. The 'SOCKS v5' option is selected. At the bottom, the '不代理的地址' (Bypass addresses) field contains 'localhost; 127.0.0.1; <local>'. The '保存' (Save) button is highlighted with a red circle.

### 1.3.30 Ubuntu-安装-theano+caffe-超详细教程

#### 一、说明

- 本文是继《Ubuntu-安装-cuda7.0-单显卡-超详细教程》之后的续篇。theano 和 caffe 是深度学习库，对运算能力需求很大，最好使用 cuda 进行加速。所以，请先阅读《Ubuntu-安装-cuda7.0-单显卡-超详细教程》，成功安装 cuda 之后，再来安装 theano 和 caffe。

#### 二、安装 Theano

##### 1. 安装各种包

- 安装 gfortran, numpy, scipy, sklearn, blas, atlas 等包

```
# 安装 gfortran, 后面编译过程中会用到
sudo apt-get install gfortran

# 安装 blas, Ubuntu 下对应的是 libopenblas, 其它操作系统可能需要安装其它版本的 blas——这是个 OS 相关的。
sudo apt-get install libopenblas-dev

# 安装 lapack, Ubuntu 下对应的是 liblapack-dev, 和 OS 相关。
sudo apt-get install liblapack-dev

# 安装 atlas, Ubuntu 下对应的是 libatlas-base-dev, 和 OS 相关。
sudo apt-get install libatlas-base-dev

# 安装 pip
sudo apt-get install python-pip
sudo apt-get install python-dev
sudo apt-get install python-nose
sudo apt-get install g++
sudo apt-get install git
```

##### 2. 安装 numpy 和 scipy

- 安装这两个 python 库有点问题，如果使用 apt-get 安装，后面的 test 不能通过。如果使用 pip 安装，有得考虑各种依赖关系。
- 所以，先使用 apt-get 安装，然后再卸载，最后再使用 pip 安装。这样，既能不考虑依赖关系，又能通过后面的 test() 测试。

```
# 安装 numpy 和 scipy
sudo apt-get install python-numpy
sudo apt-get install python-scipy
sudo apt-get install python-sklearn
# 卸载 numpy 和 scipy
sudo apt-get remove python-numpy
sudo apt-get remove python-scipy
# 安装 numpy
sudo pip install numpy
# 测试 numpy
# 如果没有安装 python-nose, 测试会出错!
python -c "import numpy;numpy.test()"
```

(下页继续)



(续上页)

```
# 安装 scipy
sudo pip install scipy
# 测试 scipy
python -c "import scipy;scipy.test()"
```

### 3. 安装 Theano

- 前面的操作如果没有出现错误，就可以开始安装 Theano 了。命令如下所示。

```
# 安装 Theano
sudo pip install Theano
# 测试 Theano
python -c "import theano;theano.test()"
```

### 4. 安装 pyCUDA

- 测试 Theano 时，提示 PyCUDA import 错误，因此需要安装 pyCUDA。而 PyCUDA 需要以 Boost 为基础，所以应该先安装 Boost。
- 使用 pip 安装 pyCUDA。

```
# 安装 boost
sudo apt-get install libboost-all-dev
```

- 如果使用 pip 安装 pyCUDA 出错，使用下面安装方式。参考文章：《Ubuntu Theano CUDA》

```
git clone --recursive http://git.tiker.net/trees/pycuda.git
cd pycuda
sudo ./configure.py --cuda-root=/usr/local/cuda --cudadr-lib-dir=/usr/lib/x86_64-
↳linux-gnu --boost-inc-dir=/usr/include --boost-lib-dir=/usr/lib --boost-python-
↳libname=boost_python --boost-thread-libname=boost_thread --no-use-shipped-boost
make -j 4 # 电脑核数
sudo python setup.py install
```

### 5. 解决 cuda\_ndarray.cu 错误

- 如果出现错误： **ERROR (theano.sandbox.cuda): Failed to compile cuda\_ndarray.cu: libcublas.so.6.5 cannot open shared object file: No such file or directory**，需要运行以下命令：

```
sudo ldconfig /usr/local/cuda-7.0/lib64
```

## 6. 配置 Theano

- 在主目录下新建.theanorc 文件

```
cd ~
vi .theanorc
```

- 在.theanorc 中输入下面的内容

```
[cuda]
root=/usr/local/cuda/bin/
[global]
floatX = float32
device = gpu0
[nvcc]
fastmath = True
```

## 7. 测试 Theano 是否在使用 GPU

- 将下列 python 代码复制到 useGPU.py, 并运行。

```
from theano import function, config, shared, sandbox
import theano.tensor as T
import numpy
import time

vlen = 10 * 30 * 768 # 10 x #cores x # threads per core
iters = 1000

rng = numpy.random.RandomState(22)
x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
f = function([], T.exp(x))
print f.maker.fgraph.toposort()
t0 = time.time()
for i in xrange(iters):
    r = f()
t1 = time.time()
print 'Looping %d times took' % iters, t1 - t0, 'seconds'
print 'Result is', r
if numpy.any([isinstance(x.op, T.Elemwise) for x in f.maker.fgraph.toposort()]):
    print 'Used the cpu'
else:
    print 'Used the gpu'
```

- 假定上面已经设置文件.theanorc, 运行命令如下所示:

```
python useGPU.py
```

- 如果出现下面的错误信息, 请运行命令 `sudo ldconfig /usr/local/cuda-7.0/lib64`[参考](#)

```
# 错误信息
ERROR (theano.sandbox.cuda): Failed to compile cuda_ndarray.cu: libcublas.so.7.0:
↪cannot open shared object file: No such file or directory
```

## Theano 相关资料

- 参考: [Using the GPU & THEANO\\_FLAGS & THEANORC](#)
- CUDA Toolkit 默认安装在/usr/local/cuda/, 含有 bin, lib, include 等子文件夹。/usr/local/cuda/bin/文件夹称为 cuda root 文件夹。
- 使用 Theano 时, 必须告诉它 CUDA root 文件夹, 有 3 种方法:
  - 定义 \$CUDA\_ROOT 环境变量。例如, CUDA\_ROOT=/usr/local/cuda/bin/
  - 在 THEANO\_FLAGS 中添加 cuda.root 标识。例如, THEANO\_FLAGS='cuda.root=/usr/local/cuda/bin/'
  - 在.theanorc 文件夹中添加 [cuda]

```
[cuda]
root=/usr/local/cuda/bin/
```

- 还需要更改设备选项 (gpu or gpu0 or gpu1), 设置默认的浮点计算类型 (float32)
  - 方法一: THEANO\_FLAGS=' cuda.root=/usr/local/cuda/bin/,device=gpu,floatX=float32'
  - 方法二: 设置.theanorc 文件的 [global] 选项:

```
[cuda]
root=/usr/local/cuda/bin/
[global]
device = gpu
floatX = float32
```

- 注意:
  - 如果电脑有多个 GPU, 而配置是 'device=gpu', 驱动会选择其中一个使用 (一般是 gpu0)。可以使用 nvidia-smi 改变这一规则。
  - 可以通过指定 'device=gpuX' 来选择一个特定的 GPU。
  - 默认使用 GPU 计算。如果 GPU 有问题, Theano 会退回使用 CPU。可以通过设置标识 'force\_device=True', 当 GPU 不能使用时, 弹出错误信息。

## 安装 OpenCV

### 下载 OpenCV

- 下载地址: <https://github.com/jayrambhia/Install-OpenCV>, 这是根据大神编译过的版本进行安装的。
- 切换到文件保存的文件夹, 然后安装依赖项:

```
# 切换路径
cd ~/Downloads/Install-OpenCV-master/Ubuntu
# 安装 OpenCV 的依赖项
sudo ./dependencies.sh
```

- 修改 opencv2\_4\_9.sh
- 参考 [Ubuntu14.04 安装 ffmpeg 下载地址](#)
- 如果不添加 CUDA\_GENERATION, 编译过程会失败 [参考文章](#)

```
# 切换路径
cd 2.4
# 修改 opencv2_4_9.sh, 添加 CUDA_GENERATION
# 根据显卡支持 Fermi 或 Kepler 做相应的修改。此处以 Fermi 为例。
sudo gedit opencv2_4_9.sh
```

- 将以下内容

```
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -
↪D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_
↪PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON -D ..
```

- 修改为:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -
↪D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_
↪PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON -D CUDA_
↪GENERATION=Fermi ..
```

- 即在最后的.. 前面添加 \*\*\*CUDA\_GENERATION=Fermi\*\*\*
- 安装最新版 opencv2\_4\_9

```
# 添加执行权限
sudo chmod +x opencv2_4_9.sh
# 安装 OpenCV
sudo ./opencv2_4_9.sh
```

```
--
-- Documentation:
--   Build Documentation:      NO
--   Sphinx:                  NO
--   PdfLaTeX compiler:       NO
--
-- Tests and samples:
--   Tests:                   YES
--   Performance tests:       YES
--   C/C++ Examples:          YES
--
-- Install path:              /usr/local
--
-- cvconfig.h is in:          /home/xuezhisd/Downloads/Inst
er/Ubuntu/2.4/OpenCV/opencv-2.4.9/build
-- .....
--
-- Configuring done
-- Generating done
-- Build files have been written to: /home/xuezhisd/Downloads/Inst
ter/Ubuntu/2.4/OpenCV/opencv-2.4.9/build
[ 0%] Built target opencv_core_pch_dephelp
[ 0%] Built target pch_Generate_opencv_core
make[2]: *** No rule to make target `/usr/lib/x86_64-linux-gnu/li
```

- 等待...,直到安装完成。如下图所示。

## 安装其它依赖项

- 执行以下命令，安装其它依赖项

```
sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev_
↪libboost-all-dev libhdf5-serial-dev libgflags-dev libgoogle-glog-dev liblmdb-dev_
↪protobuf-compiler protobuf-c-compiler python-pandas
```

## 安装 glog

1. Google Logging Library (glog)，下载地址：<https://code.google.com/p/google-glog/>，然后解压安装：

```
$ tar zxvf glog-0.3.3.tar.gz
$ ./configure
$ make
$ sudo make install
```

## 安装 caffe

### 下载 caffe

- 下载地址：<https://github.com/BVLC/caffe>

### 编辑 Makefile.config

- 参考：<http://caffe.berkeleyvision.org/installation.html> <http://ouxinyu.github.io/Blogs/20140723001.html>
- 执行以下命令，通过设置 Makefile.config 文件，来使用 CuDNN 来加速。

```
unzip caffe-master.zip # 本地解压 caffe-master
cd /caffe-master # 切换路径
vi Makefile.config # 编辑 Makefile.config
```

- 开始编辑 Makefile.config
  - 取消第 5 行的注释，即将 `#USE_CUDNN=1` 改为 `USE_CUDNN=1`
  - 将 `BLAS=atlas` 改为 `BLAS=open`

执行以下命令，

### 配置 Python 相关选项

- 安装 python 依赖库

```
cd python # 切换到./caffe-master/python/路径下
for req in $(cat requirements.txt); do sudo pip install $req; done
```

- 设置 Python 环境变量

```
sudo vi /etc/profile # 编辑 profile 文件
# 在最后面添加以下语句，注意将 path 换成你的系统下的路径
export PYTHONPATH=/path/to/caffe/python:$PYTHONPATH
```

## 安装 Matlab

- 选择 Mathworks.Matlab.R2014a.Unix.iso - 右键 - 使用磁盘映像挂载器打开”
  - 进入装载的虚拟光盘，拷贝全部文件至 home/Matlab 文件夹
1. 授权安装文件夹 `$ chmod a+x Matlab -R`
  2. 安装 `$ sudo ./install` 拷贝 `libmwservices.so` 至 `/usr/local/MATLAB/R2014a/bin/glnxa64`

```
$ sudo cp libmwservices.so /usr/local/MATLAB/R2014a/bin/glnxa64/  
安装完毕，程序默认启动路径：  
$ sh /usr/local/MATLAB/R2014a/bin/matlab
```

## 参考文章

### 1.3.31 Ubuntu-安装-搜狗输入法-详细教程

#### 下载 deb 安装包

- 安装包下载地址
- 系统 32 位的点击 “立即下载 32bit”，如下图所示；



- 系统 64 位的点击“立即下载 64bit”，如下图所示。

## 安装 fcitx

- 打开 Terminal，输入以下命令，安装 fcitx

```
sudo apt-get install fcitx
```

### 安装搜狗输入法

- 输入以下命令，安装第一步下载的 deb 安装包
- 注意：deb 包名根据你下载的包名而定。
- 下面以 64 位版本为例。

```
sudo dpkg -i sogoupinyin_1.2.0.0048_amd64.deb
```

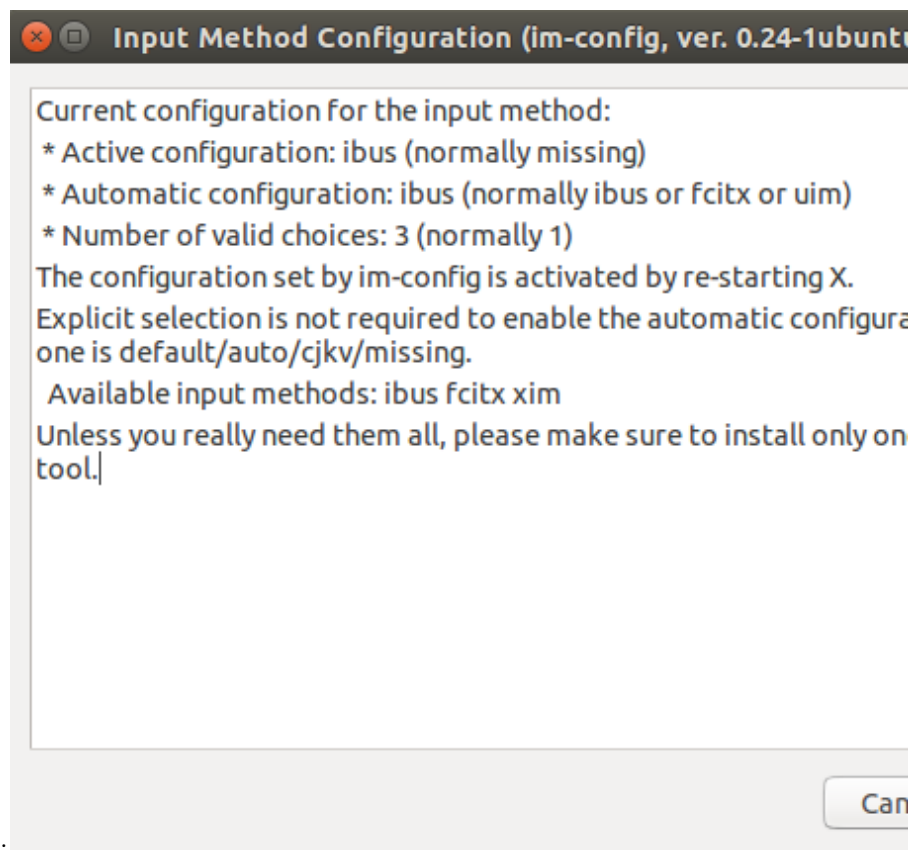
- 如果出现错误，则输入以下命令，修复。

```
sudo apt-get update  
sudo apt-get -f install
```

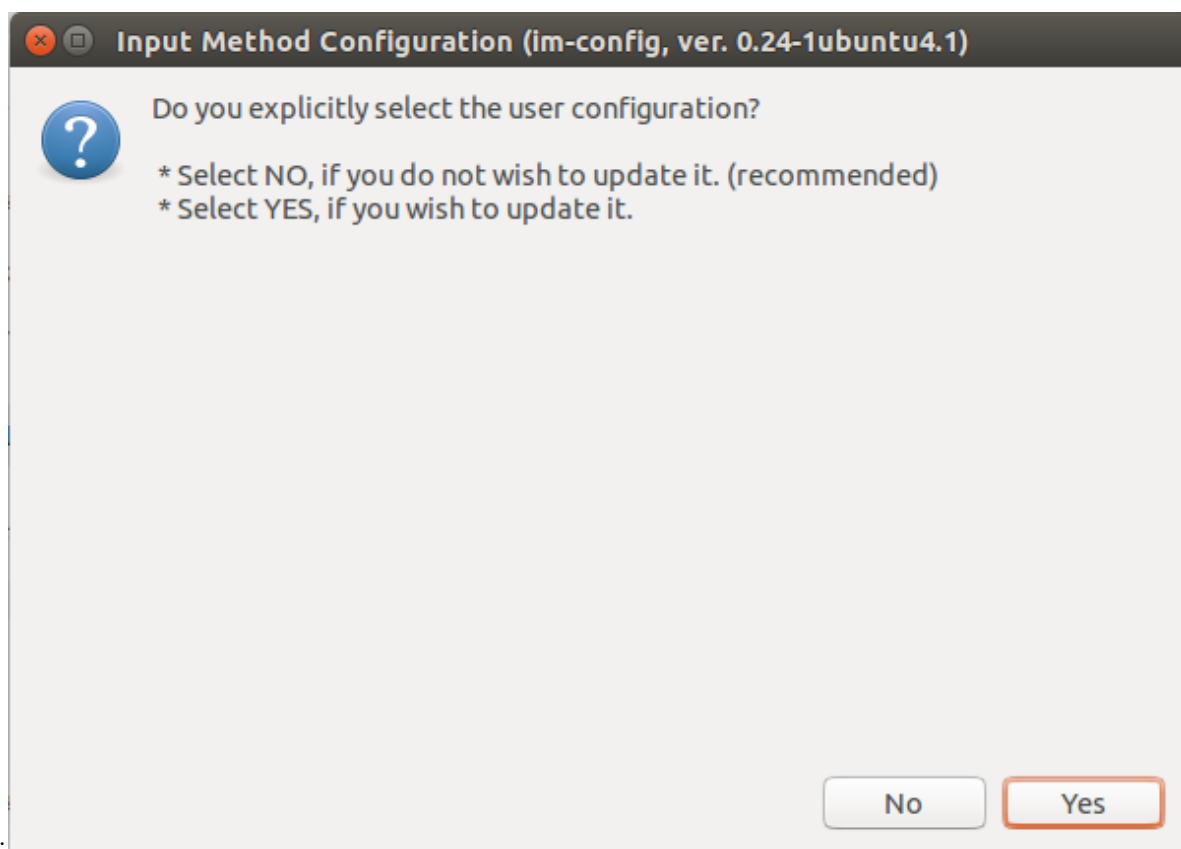
### 配置输入方法

- 输入以下命令

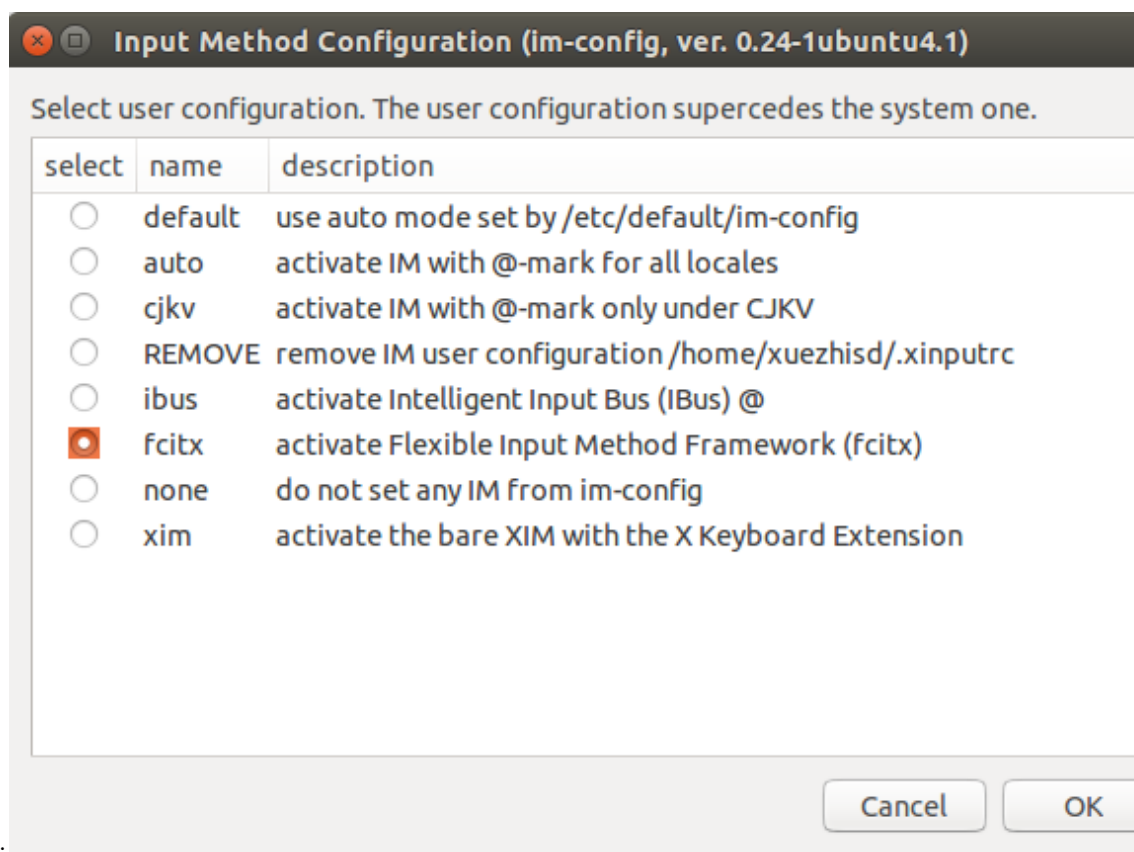
```
im-config
```



- 跳出一个对话框，如下图所示，点击“OK”；

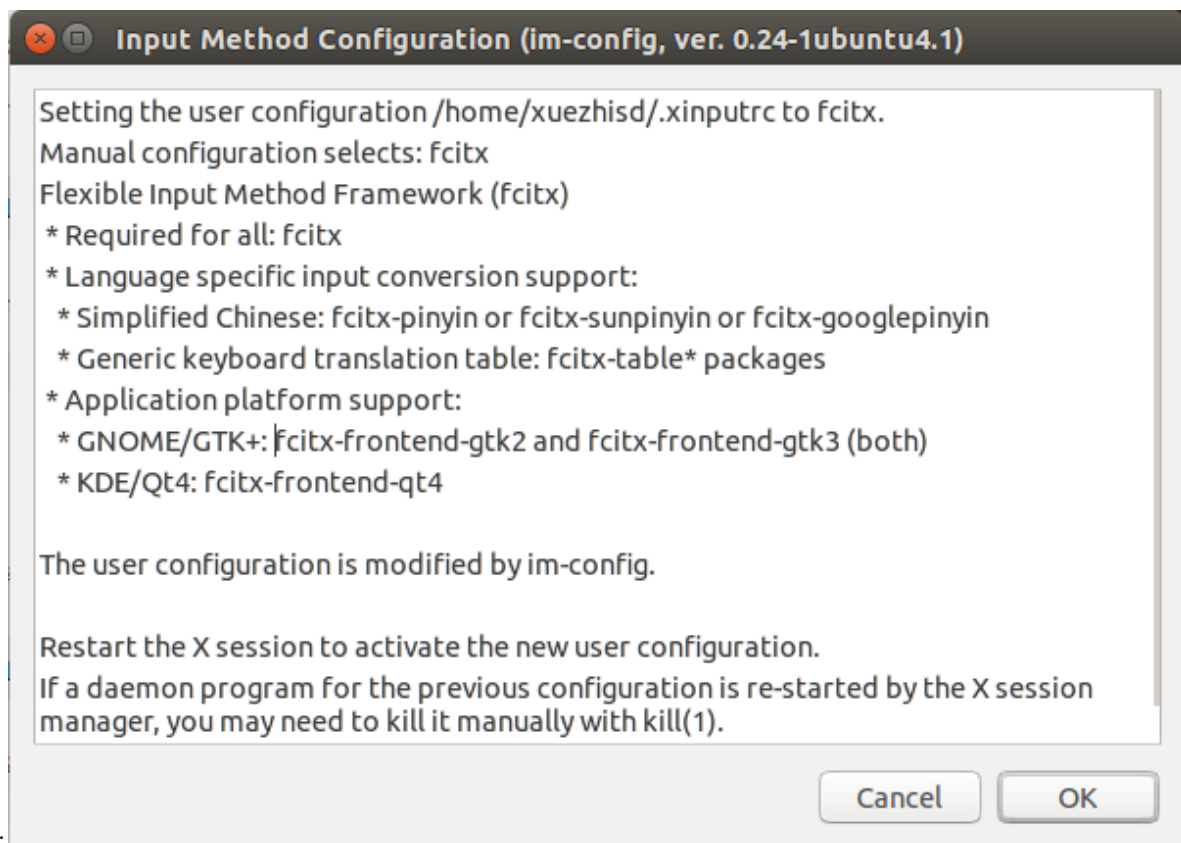


- 点击“Yes”;



- 选择 fcitx, 点击“OK”;





- 点击“OK”;

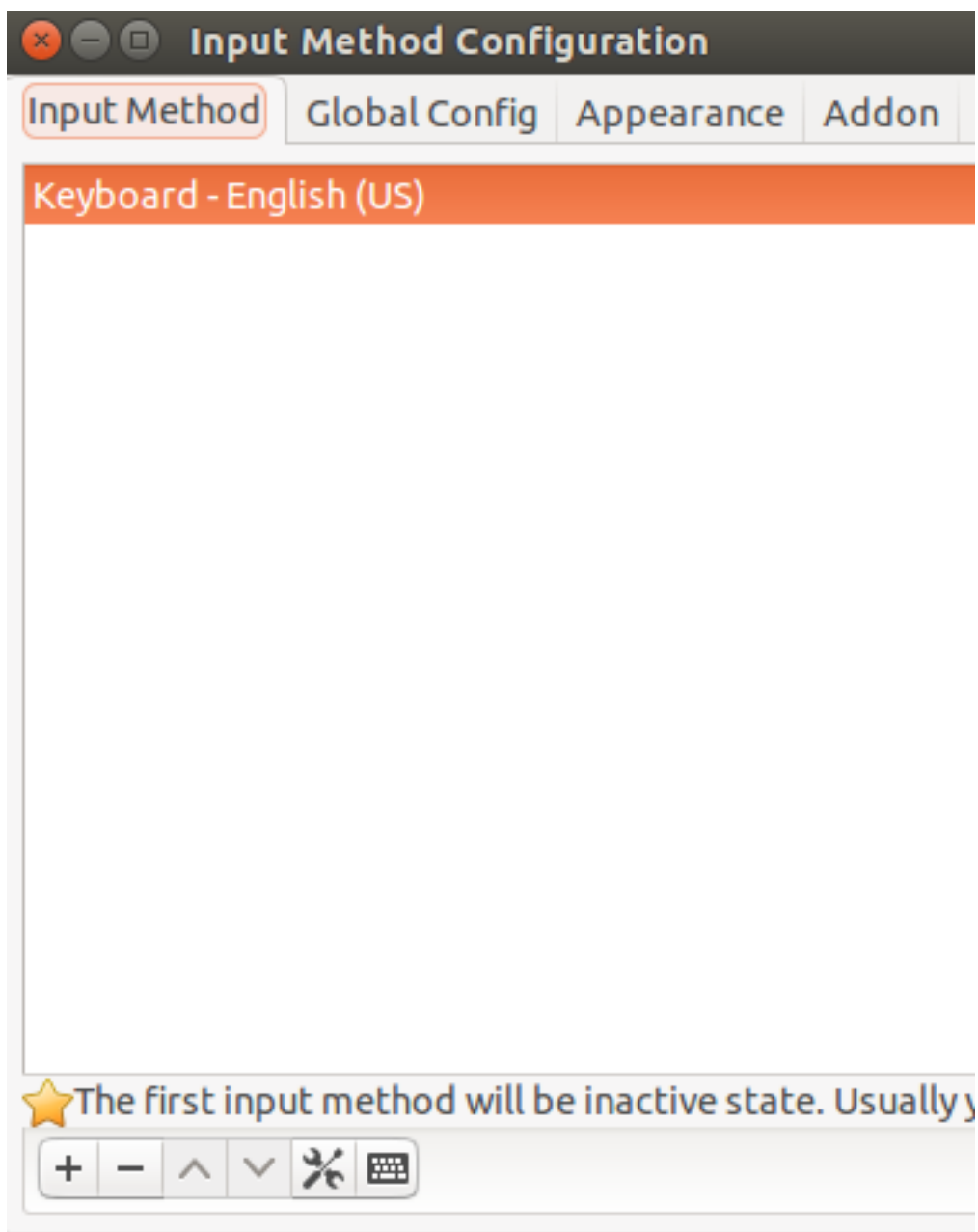
## 重启

- 重启电脑，使配置生效。

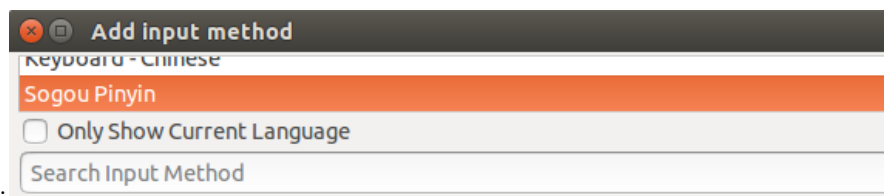
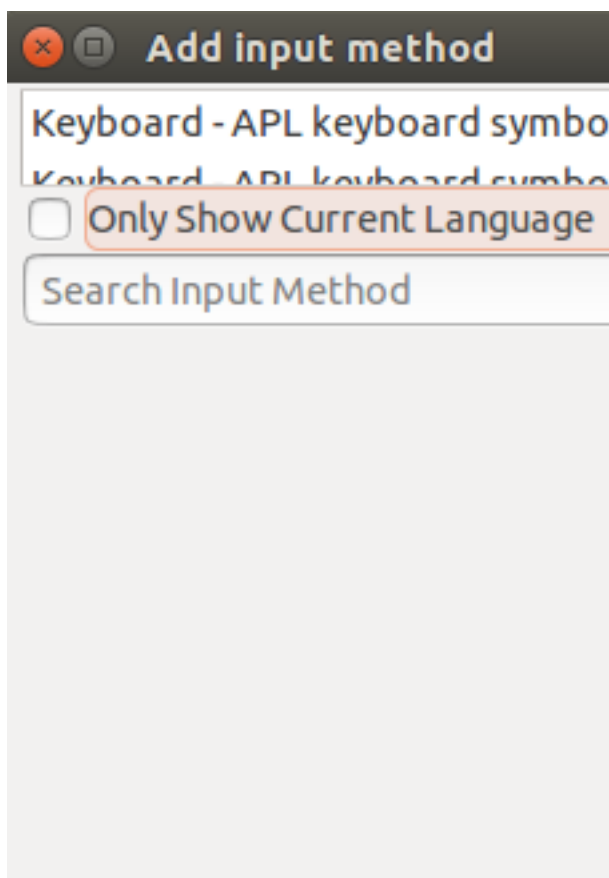
## 添加搜狗输入法

- 点击右上角的 \*\* 小键盘图标 \*\*，
- 选择 \*\* 配置选项，打开输入法配置对话框，点击左下角的 +\*\* 按钮；

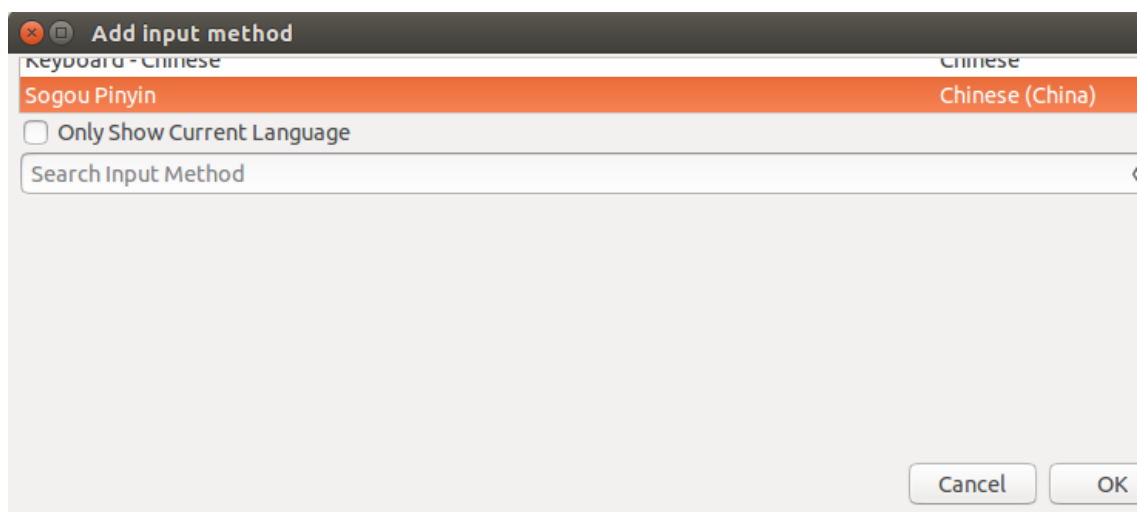




- 在弹出的对话框中，取消 Only show current language 选项；



- 拖动滚动条到最后, 选择 `sougou china`;



- 点击“Ok”, 完成安装。
- 注: 切换输入法快捷键 “Ctrl + 空格”, 切换中英文快捷键 “Shift”

至此, 搜狗输入法完成安装。。。

## 参考

- <http://pinyin.sogou.com/linux/help.php>
- <http://jingyan.baidu.com/article/0202781189b6ef1bcc9ce513.html>

## 1.3.32 Ubuntu-安装-有道词典

- Windows 下的有道词典非常地好用，在 Ubuntu 下也可以使用有道词典啦!!!

### 添加源，并更新

```
# 添加软件源
sudo add-apt-repository ppa:justzx2011/openyoudao-v0.4
# 更新
sudo apt-get update
```

### 安装软件包

```
# 安装软件包
sudo apt-get install openyoudao
```

## 1.4 博客运维

### 1.4.1 域名重定向的变通实现方法

#### 问题

- 在 Github 上搭建了一个博客：<https://www.xuezhisd.top/>，希望多个域名访问这个博客（历史遗留问题）！
  - 比如，[xuezhisd.top](https://www.xuezhisd.top/), [blog.xuezhisd.top](https://www.xuezhisd.top/), [note.xuezhisd.top](https://www.xuezhisd.top/)。即
  - <https://note.xuezhisd.top/> 自动跳转到：<https://www.xuezhisd.top/>
  - <https://blog.xuezhisd.top/> 自动跳转到：<https://www.xuezhisd.top/>

#### 失败方法

- 方案一：CNAME。GitHub 的 Page 服务不能够通过设置 CNAME 设置多个域名。
- 方案二：重定向。阿里云的域名管理，可以设置重定向，但需要备案！域名 [xuezhisd.top](https://www.xuezhisd.top/) 没有备案，不能使用这个功能。
- 方案一和方案二都失败了。

## 成功方法

- 建立另外一个 repo: [https://github.com/algoboy101/note\\_redirect\\_note](https://github.com/algoboy101/note_redirect_note), 并创建 gh-pages 分支;
- 设置该 repo 的 CNAME 为: [note.xuezhisd.top](https://note.xuezhisd.top)
- 创建一个 index.html 文件, 写入以下内容, 使用 JavaScript 实现跳转到 <https://www.xuezhisd.top> 的功能:

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      window.location.href="https://www.xuezhisd.top/"; //1
      window.history.back(-1); //2 页面返回
      self.location='https://www.xuezhisd.top/'; //3
      top.location='https://www.xuezhisd.top/'; //4
    </script>
  </head>
</html>
```

- 提交到 gh-pages 分支。
- 尝试访问: <https://note.xuezhisd.top/>。链接自动跳转到: <https://www.xuezhisd.top/>。

## 原理解释

- 新创建的 repo 提供了额外的 CNAME, 可以用来新绑定一个域名。
- 该 repo 中的 index.html 实现你跳转功能: 访问到该文件, 立马跳转到指定链接。

## 1.4.2 优化 travis-ci 配置文件

### 问题

- 旧版本的 .travis.yml 内容如下:

```
language: python - "3.7"

before_install:
  - wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O_
↪miniconda.sh
  - chmod +x miniconda.sh
  - "./miniconda.sh -b"
  - export PATH=/home/travis/miniconda3/bin:$PATH
  - conda update --yes conda
  - sudo ln -s /run/shm /dev/shm

install:
  - conda install --yes python="3.7" psutil sphinx numpy scipy gdal cython h5py_
↪pycurl shapely mock matplotlib
  - pip install --user travis-sphinx recommonmark sphinx_rtd_theme

script:
  # travis-sphinx build --source=./
  travis-sphinx build --source=./source

after_success:
  # travis-sphinx deploy
  travis-sphinx deploy --cname www.xuezhisd.top
```

- 构建时间是 2 分钟多，作者认为时间太长，有优化空间。

### 优化方法

- **分析：** 时间都浪费在 miniconda 相关的操作上；
- **思路：** 将 miniconda 干掉，直接使用系统自带的 python3；
- 使用 apt 安装 pip3，并升级；
- 使用 pip 直接安装 sphinx 等相关的 package。
- 新版本的 .travis.yml 内容如下所示：

```
language: python - "3.7"

before_install:
  #- sudo apt-get update
  - sudo apt-get install python3-pip python3-setuptools python3-wheel
  - sudo pip3 install pip --upgrade
  - sudo apt-get remove python3-pip
install:
  - pip3 install --user sphinx travis-sphinx recommonmark sphinx_rtd_theme
script:
  # travis-sphinx build --source=./
  travis-sphinx build --source=./source
after_success:
  # travis-sphinx deploy
  travis-sphinx deploy --cname www.xuezhisd.top
```

### 结果

- 时间从 2 分钟多，减少到一分钟左右。

## 1.4.3 hexo 教程系列——hexo 安装教程

### 安装 node.js

- node.js 官方下载地址
- 从上面的链接下载 node.js，并安装。
  - 注意：官方链接可能需要翻墙
  - 注意：我的操作系统是 **Windows 7 (64bit)**

## 设置 npm 淘宝镜像站

- npm 默认的源的下载速度可能很慢，建议使用淘宝镜像替换。
- 执行下面的命令，将 npm 的源设置成淘宝镜像站。

```
npm config set registry "https://registry.npm.taobao.org"
```

## 申请 Github 账号

- [Github 注册页面](#)
- 输入用户名，Email，密码，注册账号。比如我的用户名是：xuezhisd

## 创建博客仓库

- 注意，仓库名应该为：用户名.github.io。比如，我的仓库名是：xuezhisd.github.io。

## 安装 git

- [git 下载地址](#)
- 下载 git 安装文件，双击执行安装。

## 配置 ssh

- 打开 git bash 终端。
- 设置 user.name 和 user.email。

```
git config --global user.name " 你的 GitHub 用户名"  
git config --global user.email " 你的 GitHub 注册邮箱"
```

- 生成 ssh 密匙

```
ssh-keygen -t rsa -C "你的 GitHub 注册邮箱"
```

- 此时，在用户文件夹下就会有一个新的文件夹.ssh，里面有刚刚创建的 ssh 密钥文件 id\_rsa 和 id\_rsa.pub。

## 将公匙添加到 github 上

- 详细教程自行 baidu。
- 用户头像 → Settings → SSH and GPG keys → New SSH key → 将 id\_rsa.pub 中的内容复制到 Key 文本框中，然后点击 Add SSH key(添加 SSH) 按钮。

## 安装 hexo

- 执行以下命令安装 hexo。

```
# 安装 hexo
npm install hexo-cli g
# 初始化博客文件夹
hexo init blog
# 切换到该路径
cd blog
# 安装 hexo 的扩展插件
npm install
# 安装其它插件
npm install hexo-server --save
npm install hexo-admin --save
npm install hexo-generator-archive --save
npm install hexo-generator-feed --save
npm install hexo-generator-search --save
npm install hexo-generator-tag --save
npm install hexo-deployer-git --save
npm install hexo-generator-sitemap --save
```

## 初探 hexo

- 第一次使用 hexo，在本地创建服务器使用。

```
# 生成静态页面
hexo generate
# 开启本地服务器
hexo s
```

- 打开浏览器，地址栏中输入：**http://localhost:4000/**，应该可以看见刚刚创建的博客了。
- 问题：为什么访问 **http://localhost:4000/**，无反应？
  - 解决方法：可能是由于端口问题引起的。使用 Ctrl+C 中断本地服务，使用命令 `hexo s -p 5000` 重新开启本地服务，访问 **\*\*http://localhost:5000/\*\*** 可以看到博客页面了。

## 将 hexo 博客部署到 github 上

- 修改配置文件 **blog/\_config.yml**，修改 deploy 项的内容，如下所示：

```
# Deployment 注释
## Docs: https://hexo.io/docs/deployment.html
deploy:
  # 类型
  type: git
  # 仓库
  repo: git@github.com:xuezhisd/xuezhisd.github.io.git
  # 分支
  branch: master
```

- 注意： **type: git** 中的冒号后面由空格。
- 注意： 将 `git@github.com:xuezhisd/xuezhisd.github.io.git` 中的用户名换成自己的用户名 `git@github.com:github_username/github_username.github.io.git`。



## 部署 hexo

- 输入下面的命令将 hexo 博客部署到 github 中：

```
# 清空静态页面
hexo clean
# 生成静态页面
hexo generate
# 部署
hexo deploy
```

- 打开网页，输入 `http://github_username.github.io`，打开 github 上托管的博客。如我的博客地址是：`http://xuezhisd.github.io`。

## hexo 命令缩写

- hexo 支持命令缩写，如下所示。hexo g 等价于 hexo generate

```
hexo g: hexo generate
hexo c: hexo clean
hexo s: hexo server
hexo d: hexo deploy
```

## hexo 组合命令

- 将多个命令组合在一起使用，省事！

```
# 清除、生成、启动
hexo clean && hexo g -s
# 清除、生成、部署
hexo clean && hexo g -d
```

## 常见问题

### hexo deploy 没有反应？

- 修改配置文件：`_config.yml` 时，冒号后面没加空格。

### hexo s 网站打不开？

- 端口占用，换个端口就好了。执行命令 `hexo s -p 5000`，并在浏览器地址栏输入 `http://localhost:5000`，回车访问。

## 如何换主题？

- 将主题下载后，放到 `themes` 文件夹中即可。例如，下面命令安装 `next` 主题：`git clone https://github.com/iissnan/hexo-theme-next themes/next`。

## 参考博客

### 1.4.4 hexo 教程系列——hexo 配置教程

#### 网站的配置文件

- 网站配置文件的存储位置：`hexo_blog/_config.yml`。
- 配置文件中，冒号后面一定要加空格。
- `hexo` 官方配置说明
- 我的配置文件内容如下所示。

```
# Hexo Configuration Hexo 配置文件
## Docs: https://hexo.io/docs/configuration.html
## Source: https://github.com/hexojs/hexo/

# 网站信息
# 标题
title: 学志の博客
# 副标题
subtitle: 记录学习的技能和遇到的问题
# 网站描述
description: 做自己爱做的事，爱自己在做的事！
# 作者昵称
author: 张学志
# 网站语言，默认英语，设置简体中文
language: zh-Hans

# 时区，默认电脑时区
#timezone:
timezone: Asia/Shanghai

# 网址设置
# 如果网站是放在子目录中，将 url 设置成'http://yoursite.com/child'，将 root 设置成'/child/'
## If your site is put in a subdirectory, set url as 'http://yoursite.com/child' and
↳root as '/child/'
# 网址
url: http://zhangxuezhi.com
# 网站根目录。如果网站是放在子目录中，将 root 设置成'子目录名'
root: /
# 文章链接地址格式。即文章存放的目录。
permalink: :year/:month/:day/:title/
permalink_defaults:

# 目录设置
# 资源文件夹，放在里面的文件会上传到 github 中
```

(下页继续)

(续上页)

```

source_dir: source
# 公共文件夹, 存放生成的静态文件
public_dir: public
# 标签文件夹, 默认是 tags。实际存放在 source/tags 中。
tag_dir: tags
rss_dir: rss
# 档案文件夹, 默认是 archives。
archive_dir: archives
# 分类文件夹, 默认是 categories。实际存放在 source/categories 中。
category_dir: categories
# 代码文件夹, 默认是 downloads/code
code_dir: downloads/code
# 国际化文件夹, 默认跟 language 相同
i18n_dir: :lang
# 不需要渲染的文件夹或文件夹, 放在 [] 中
# 这两个文件是百度和 google 的站长验证文件, 不能渲染, 否则会改变内容, 不能验证过
skip_render: [baidu_verify_R9MZjdMkXT.html, google0f8fac7da2b48ef8.html, README.md, 模
板.md]

# 写作选项
# 新建博文 (帖子) 的默认名称
# File name of new posts
new_post_name: :title.md
# 默认布局模板是 post, 而不是 draft 和 page
default_layout: post
# 是否将标题转换成标题形式 (首字母大写)
titlecase: false # Transform title into titlecase
# 在新标签页面中打开网页
external_link: true # Open external links in new tab
filename_case: 0
# 是否渲染草稿
render_drafts: false
# 启动 Asset 文件夹
post_asset_folder: false
# 把链接改为与根目录的相对位址
relative_link: false
# 显示未来的文章
future: true
# 代码块的设置
highlight:
  enable: true # 使用代码高亮
  line_number: true # 显示行号
  auto_detect: true # 自动检测语言
  tab_replace:

# 分类和标签
# 默认分类
default_category: uncategorized
# 分类别名
category_map:
# 标签别名
tag_map:

```

(下页继续)

(续上页)

```
# 日期和时间格式
#Hexo 使用 Moment.js 来解析和显示时间。
## You can customize the date format as defined in
## http://momentjs.com/docs/#/displaying/format/
date_format: YYYY-MM-DD
time_format: HH:mm:ss

# 分页配置
# -----下面选项需要对应插件的支持-----
# npm install hexo-generator-index --save
# npm install hexo-generator-archive --save
# npm install hexo-generator-category --save
# npm install hexo-generator-tag --save
## Set per_page to 0 to disable pagination
# 每页显示的文章量
#per_page: 20
# 首页的分页设置
index_generator:
  per_page: 5
# 归档页的分页设置
archive_generator:
  per_page: 30
  yearly: true
  monthly: true
# 标签页的分页设置
tag_generator:
  per_page: 20

# 分页路径, 在 public 中可以看到
#pagination_dir: page

# Extensions 拓展插件配置
## Plugins: https://hexo.io/plugins/
plugins:
baidusitemap:
  path: baidusitemap.xml

# 配置 RSS
feed:
  #feed 类型 (atom/rss2)
  type: atom
  #rss 路径
  path: atom.xml
  # 在 rss 中最多生成的文章数 (0 显示所有)
  limit: 0

# 自定义站点内容搜索
# 需要先安装插件:
# npm install hexo-generator-search --save
search:
  path: search.xml
```

(下页继续)

(续上页)

```
# 如只想索引文章, 可设置为 post
field: all

# 主题配置
## Themes: https://hexo.io/themes/
#theme: false # 禁用主题
#theme: landscape
theme: next

# 部署配置
## Docs: https://hexo.io/docs/deployment.html
deploy:
  type: git
  #repo: https://github.com/xuezhisd/xuezhisd.github.io.git
  repo:
    # 部署到 github
    github: git@github.com:xuezhisd/xuezhisd.github.io.git, master
    # 部署到 coding.net. 取消注释, 可同时部署
    #coding: git@git.coding.net:xuezhisd/blog.git,coding-pages
  #type: baidu_url_submitter
```

## 主题的配置文件

- 默认主题是 landscape
- hexo 官方主题页面。从中选择喜欢的主题, 下载并放到 themes 文件中。
- 我使用的主题是 NexT。

## 参考博客

### 1.4.5 hexo 教程系列——使用 Travis 自动部署 hexo

本文介绍了如何使用 Travis CI 自动部署 hexo, 解决了家里和公司两地写博客的麻烦。本博客实现了博客仓库中的 dev 分支改动时, 自动部署到 master 分支中。

## 说明

- 为了在不同电脑上写完并提交博文后, 博客能自动更新内容。本文介绍如何通过 Travis CI 来实现 github 上的 hexo 博客自动部署。
- 本教程是在 Windows 7 上操作的。
- 本教程使用了 GitHub API 实现, 用到了 Linux 中的 sed 命令。

## 新建 Personal Access Token

- 点击右上角头像旁边的三角，在菜单中点击 “Setting”，进入设置页面。
- 点击左侧栏的最下面的 “Personal access tokens”，创建 Personal access tokens。
- 点击右上角的 “Generate new token”，输入用户密码，进入 “New personal access token” 页面。
- 设置 Token description（其实就是名称），选择相应的权限，如下图所示。
- 点击 “Generate token” 按钮，生成 Personal access tokens。如下图中红线标注的部分。**注意：这行 token 只会在刚刚创建完成后显示一次，以后不再显示。因此，复制并保存到本地。**

## 配置 Travis CI

### 登录并配置 Travis CI

- Travis CI 是目前新兴的开源持续集成构建项目。可以直接使用 GitHub 账号登录。
- 将鼠标放在用户名上，在弹出的菜单中点击 “Accounts”，将会显示你在 GitHub 上的仓库。如下图所示。
- 找到自己的博客项目，点击 X 号，将其变成√号。再点击右侧的齿轮，进入该仓库的配置页面。
- 在项目的设置中开启 Build only if .travis.yml is present 这一项。如下图所示。

### 本地安装 Travis

- 首先安装 Ruby，直接官网下载，双击安装就 OK 了。
- 在 Windows 下，安装 travis 之前，需要解决一个问题：**SSL 证书问题**，否则不能成功安装。详情请点击该链接：[参考教程](#)。
- 修复好上述问题后，执行下面的命令安装 travis。

```
# 安装 travis
gem install travis
```

### 创建并修改配置文件

- 打开博客项目文件夹，在项目根目录新建.travis.yml 配置文件。

```
cd 博客项目文件夹根目录
touch .travis.yml
```

- 执行下面的命令，加密上面生成的 Personal access tokens，并添加到.travis.yml 配置文件。

```
# 这里的 REPO_TOKEN 是变量名，在后面的配置文件中会用到
# TOKEN 是上面 github 生成的 Token
travis encrypt 'REPO_TOKEN=<TOKEN>' --add
```

- 上述命令指向完后，.travis.yml 配置文件的内容如下所示。

```
env:
  global:
    secure: F/gYu+bVe3aWs6YnuB5PNZLEmHH4CGf0najk8JI3/N+SFL0TH8tyYg+O1sXR3EIg.....省略..
    →...iwr5HQG/G3fdL4JcXiAJIm/iH9ndDyGV5EGR1CmseNGNWrwnRZ7t1KE=
```

- 使用本地编辑器打开.travis.yml 配置文件，将其修改成如下所示的内容。注意：有些内容是你自己的!!!

```
language: node_js
node_js:
- "4" # nodejs 的版本
branches:
  only:
    - dev # 设置自动化部署的源码分支

# -----
# 下面是你的 Token 加密信息，不要替换
# -----
env:
  global:
    secure: F/gYu+bVe3aWs6Yn.....省略..../iH9ndDyGV5EGR1CmseNGNWrwRZ7tlKE=

before_install:
- export TZ='Asia/Shanghai'
- npm install -g hexo
- npm install -g hexo-cli
before_script:
# -----
# 设置 github 账户信息 注意修改成自己的信息
# -----
- git config --global user.name "xuezhisd"
- git config --global user.email xuezhi@126.com
# -----
# github 仓库操作 注意将仓库修改成自己的
# -----
- sed -i'' "s~git@github.com:xuezhisd/xuezhisd.github.io.git~https://${REPO_TOKEN}:x-
  →oauth-basic@github.com:xuezhisd/xuezhisd.github.io.git~" _config.yml
# 安装依赖组件
install:
- npm install
# 执行的命令
script:
- hexo clean
- hexo generate
# 执行成功后执行
after_success:
- hexo deploy
```

- 至此，Travis CI 已经配置完成了。

## 创建仓库，推送到 GitHub

- 新建仓库 blog。

```
git init blog
cd blog
```

- 将原来博客目录下的以下内容复制到 blog 文件夹中。
- 修改.gitignore 文件，取消对 node\_modules 文件夹的忽略。（即删除对应的行）
- 关联 GitHub 上的远程仓库。

```
# 将 github 仓库改为自己的博客仓库
git remote add origin git@github.com:xuezhisd/xuezhisd.github.io.git
```

- 提交本地修改，推送到 GitHub 上。

```
# 添加文件到暂存区
git add .
# 提交修改
git commit -m "test travis"
# 推送至远程仓库的 dev 分支
git push -u origin dev
```

- push 本地的代码至远程仓库之后，在 <https://travis-ci.org> 后台查看相关情况。如果如下图所示，就代表成功了。

## 参考博客

### 1.4.6 测试公式

$E = mc^2$

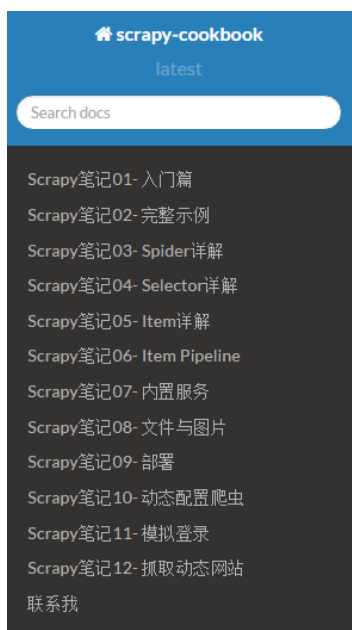
$$E = mc^2$$

### 1.4.7 测试代码块

```
import numpy as np
```

### 1.4.8 测试链接

### 1.4.9 测试图片



Docs » Welcome to scrapy-cookbook's documentation!

[Edit on C](#)

## Welcome to scrapy-cookbook's documentation!

Contents:

- Scrapy笔记01-入门篇
  - 安装scrapy
  - 简单示例
  - Scrapy特性一览
- Scrapy笔记02-完整示例
  - 创建Scrapy工程
  - 定义我们的Item
  - 第一个Spider
  - 运行爬虫
  - 处理链接
  - 导出抓取数据
  - 保存数据到数据库
  - 下一步