

# 第1章作业

## ✓ 1. 熟悉Linux

1. 请描述apt-get 安装软件的整体步骤，说明Ubuntu 是如何管理软件依赖关系和软件版本的。

注：阅读了鸟哥私房菜第23章的内容，该章主要讲述了RPM机制的软件安装方式。而ubuntu使用dpkg机制的软件安装方式，本人平时只是使用apt-get安装软件，并没有深入了解背后的安装原理，所以本题参考了[ubuntu APT-GET工作原理](#)进行作答。

### 使用apt-get安装软件主要分为四步：

- 扫描本地存放的软件包更新列表（由“apt-get update”命令刷新更新列表，也就是/var/lib/apt/lists/），找到最新版本的软件包；
- 进行软件包依赖关系检查，找到支持该软件正常运行的所有软件包；
- 从软件源所指的镜像站点中，下载相关软件包；
- 解压软件包，并自动完成应用程序的安装和配置。

### Ubuntu 是如何管理软件依赖关系和软件版本：

有时候在Ubuntu下安装软件包时会遇到提示说，下列软件包有未满足的依赖关系这样的问题。实际上linux的包是开源的，而这些包的编辑者大多数是一些大神为了方便使用某个功能编写的。恰好某个复杂功能里用到了这个小功能，那么作为程序员肯定是直接调用现成的。依赖的意思就是你用这个复杂功能之前需要使用某个包或者调用某个包。

这里引入一个APT包管理工具的概念，Ubuntu上用与Debian一样的Deb软件管理工具，apt-get就是Ubuntu的Deb软件管理工具，即APT包管理工具。这个软件会从Ubuntu的软件源库里调用安装所需要安装的包，而且可以自动分析和解决依赖关系，并且将所依赖的软件都搞定。Ubuntu如果想安装指定版本的软件，可以在安装时在安装包名后边指定所要安装的版本即可。

## 2. 什么是软件源？如何更换系统自带的软件源？如何安装来自第三方软件源中的软件？

**软件源**（软件仓库）一般指debian系操作系统的应用程序安装包仓库，其中存放大量的软件包，apt会从软件源中下载软件，在/etc/apt/sources.list中可以为apt配置软件源。

其实在Ubuntu中软件源还细分为两种：Ubuntu官方软件源和PPA软件源

Ubuntu 官方软件源中包含了 Ubuntu 系统中所用到的绝大部分的软件，它对应的源列表是 /etc/apt/sources.list。在这个文件中，记录了 Ubuntu 官方源的地址。像我们平时常用的清华源、阿里源等镜像地址，它其实和 Ubuntu 官方的镜像是相同的，我这里做了替换主要是为了加快 apt 安装和更新软件源的速度。

PPA 源出现的背景是因为系统自带的源是很有限的，我们肯定需要一些其他的软件包然而如果是直接下载deb格式的文件的话，又不能获取到更新和维护，所以这就用到了十分重要的 PPA 源了。

### 如何更新系统自带的软件源

```
1 #这里尽量添加新的软件源，而不要删除掉原来自带的ubuntu软件源
2 sudo gedit /etc/apt/sources.list
3 添加源 #国内开源镜像站点汇总
  https://www.asfor.cn/server/mirror/index.html
4 sudo apt-get update
```

### 安装来自第三方软件源中的软件

我用的是Ubuntu18.04系统，安装搜狗拼音时直接下载deb安装包，用dpkg进行安装，使用 sudo apt install -f 进行修复，对于安装第三方软件没有什么特别的要求。但Ubuntu 16.04 自带的软件中心对第三方软件安装非常不友好，需要先安装gdebi工具，然后使用此工具进行第三方软件的安装。

## 3. 除了apt-get 以外，还有什么方式在系统中安装所需软件？除了Ubuntu 以外，其他发行版

使用什么软件管理工具？请至少各列举两种。

我知道的Ubuntu软件安装主要有以下几种方式：

- **下载deb格式文件进行离线安装**；使用dpkg命令进行安装和卸载deb安装包。
- **apt-get方式在线安装**；主要用于在线从互联网的软件仓库中 搜索、安装、升级、卸载 软件。

- **make方式安装**；对于某些软件仓库没有的库或者安装包，或者下载过慢的这种情况，我们可以选择直接下载其源码，然后进行源码安装。通常分为四步：配置-->编译-->安装-->清除临时文件。
- **pip或者conda安装**；这种方式并不是ubuntu自带的安装工具，但在特定的环境下需要使用这两种安装方式。

```
1 | sudo dpkg -i 安装包 命令
2 | sudo apt-get install 安装包 命令卸载
```

其它发行版本的软件管理工具：

distribution 代表	软件管理机制	使用指令	在线升级机制(指令)
Red Hat/Fedora	RPM	rpm, rpmbuild	YUM (yum)
Debian/Ubuntu	DPKG	dpkg	APT (apt-get)

Archlinux使用pacman进行软件包管理

#### 4. 环境变量PATH 是什么？有什么用途？ LD\_LIBRARY\_PATH 是什么？ 指令ldconfig 有什么

用途？

环境变量PATH：系统通过PATH这个环境变量来获得可执行文件所在的位置，然后运行一下对应的可执行文件。

LD\_LIBRARY\_PATH：程序已经成功编译并且链接成功后，使用LD\_LIBRARY\_PATH来搜索目录，该变量中只有动态库有意义。

**ldconfig的作用：**

ldconfig是一个动态链接库管理命令，为了让动态链接库为系统所共享，还需运行动态链接库的管理命令--ldconfig。ldconfig的用途，主要是在默认搜寻目录 (/lib和/usr/lib) 以及动态库配置文件/etc/ld.so.conf内所列的目录下,搜索出可共享的动态链接库 (lib.so)，进而创建出动态装入程序 (ld.so) 所需的连接和缓存文件.缓存文件默认为 /etc/ld.so.cache，此文件保存已排好序的动态链接库名字列表。

#### 5. Linux 文件权限有哪几种？如何修改一个文件的权限？

文件权限是指文件的访问控制，即那些用户和组群可以**访问文件**以及可以**执行什么样的操作**。默认情况下文件或目录的创建者即为该对象的属主。属主对文件或者目录有特别的操作权限。

访问权限规定三种不同类型的用户：

- 文件属主 (Owner)：文件的所有者，称为属主。
- 同组用户 (Group)：文件属组的同组用户。
- 其它用户 (Others)：可以访问文件的其他用户。

访问权限的表示方法有三种，即三组九位字母表示法、三组九位二进制表示法和三位八进制表示法。**其中用r表示读，用w表示写，用x表示可执行可查找，用-表示无权限。**

### 文件权限的修改方法：

修改文件权限的命令是chmod，执行该命令要求必须为文件属主或者是root用户才能使用。有两种修改方法，字母形式修改权限；数字形式修改权限。

- 字母形式修改权限；即“用户对象 操作符号 操作权限”

```
1 | chmod [选项] 模式 [, 模式] 文件名
```

- 数字形式修改权限

```
1 | chmod 八进制模式 文件名
```

## 6. Linux 用户和用户组是什么概念？用户组的权限是什么意思？有哪些常见的用户组？

Linux操作系统是多用户的分时操作系统，具有功能强大的用户管理机制，它将用户分为组，每个用户都属于某个组，每个用户都需要进行身份验证，同时用户只能在所属组所拥有的权限内工作，这样不仅方便管理，而且增加了系统的安全性。

**用户：**分为普通用户、管理员用户（root用户）和系统用户。普通用户在系统上的任务是进行普通的工作，root用户对系统具有绝对的控制权，但操作不当会对系统造成损毁。所以在进行简单任务是使用普通用户。

**用户组：**用户组是用户的容器，通过组，我们可以更加方便的归类、管理用户。用户能从用户组继承权限，一般分为普通用户组，系统用户组，私有用户组。当创建一个新用户时，若没有指定他所属于的组，系统就建立一个与该用户同名的私有组。当然此时该私有组中只包含这个用户自己。标准组可以容纳多个用户,若使用标准组,在创建一个新的用户时就应该指定他所属于的组。

7. 常见的Linux下C++编译器有哪几种？在你的机器上，默认用的是哪一种？它能够支持C++的哪个标准？

gcc和g++。我的电脑中默认用的是g++，它支持c++11标准。

## ✓ 2. SLAM综述文献阅读

1. SLAM 会在哪些场合中用到？至少列举三个方向。

SLAM的应用场景主要有以下几个方向：

- SLAM技术可用于增强现实（AR, Augmented reality），增强现实是一种在现实场景中无缝融入虚拟物体的一种技术，比如京东或者淘宝一些店铺提供的在线试鞋服务，抖音等提供的一些AR特效服务，利用平板可以观察选中的家具在自己房间摆放和搭配的效果。
- SLAM在自动驾驶领域的应用，如无人驾驶利用激光雷达传感器来获取地图数据，构建地图，规避路程中遇到的障碍实现路径规划。
- SLAM在机器人领域的应用，如扫地机器人等。
- 无人机领域，无人机在飞行过程中需要知道哪里有障碍物，该怎么规避，怎么重新规划路线，显然需要用到slam技术。

2. SLAM 中定位与建图是什么关系？为什么在定位的同时需要建图？

定位是为了精确地确定当前设备在某个环境中的姿态和位置；建图将周围环境的观测部分整合到一个单一的模型中。最初定位和建图是两个相互独立的关系，后来发现这两个步骤是相互依赖的。建图的准确性依赖于定位精度，而定位的实现又离不开精确的建图。

SLAM强调在未知环境下进行整个过程，如果在未知环境下进行定位，首先需要能够识别并理解周围的环境。再利用环境中的外部信息作为定位的基准，所以需要对所处的环境进行建图。

### 3. SLAM 发展历史如何？我们可以将它划分成哪几个阶段？

概率SLAM问题最早起源于1986年在旧金山举办的IEEE机器人自动化会议中。经过几年的深耕，Smith等人发表了具有里程碑意义的论文。在最早提出SLAM的一系列论文中，当时的人们称它为“空间状态不确定性的估计”。而SLAM这一概念最早由Hugh Durrant-Whyte 和 John J.Leonard提出。SLAM主要用于解决移动机器人在未知环境中运行时定位导航与地图构建的问题。从SLAM的提出到现在已有三十多年，大致划分为以下几个阶段：

1. 基于距离传感器的传统SLAM技术：SLAM技术最早是基于概率统计的扩展卡尔曼滤波的方法（EKF-SLAM），后来经过改进学者们提出了Fast SLAM方法，该方法大大减小计算量，提升运算速度。
2. 基于视觉传感器的SLAM技术：2010年后，越来越多的SLAM系统采用视觉传感器作为主要的输入信号，提出了**基于特征法的SALM**、**基于直接法的SLAM**
3. SLAM技术未来研究热点将是**多传感器融合SLAM**、**语义SLAM**、**SLAM于深度学习的结合**

参考文献：

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," in IEEE Robotics & Automation Magazine, vol. 13, no. 2, pp. 99-110, June 2006, doi: 10.1109/MRA.2006.1638022.

[2] 刘明芹,张晓光,徐桂云,李宗周.单机器人SLAM技术的发展及相关主流技术综述[J].计算机工程与应用,2020,56(18):25-35.

### 4. 从什么时候开始SLAM 区分为前端和后端？为什么我们要把SLAM 区分为前端和后端？

SLAM系统结构出现前后端应该是在出现基于视觉的SLAM方法之后，推测应该是在2010年之后，具体何时出现明确的前后端区别，我还不太清楚。

SLAM系统的体系结构包括两个主要部分：前端（或者叫视觉里程计）与后端。前端将传感器数据抽象成适用于估计的模型，估算相邻图像间相机的运动，以及局部地图的样子；而后端在这些经由前端处理的抽象数据上执行推理。后端接受不同时刻视觉里程计测量的相机位姿，以及回环检测的信息，对它们进行优化，得到全局一致的轨迹和地图。前后端所实现的功能不同，目的不同。

## 5. 列举三篇在SLAM 领域的经典文献。

[1] Mur-Artal R, Montiel J M M, Tardos J D. ORB-SLAM: a versatile and accurate monocular SLAM system[J]. IEEE transactions on robotics, 2015, 31(5): 1147-1163.

[2] Davison A J, Reid I D, Molton N D, et al. MonoSLAM: Real-time single camera SLAM[J]. IEEE transactions on pattern analysis and machine intelligence, 2007, 29(6): 1052-1067.

[3] Durrant-Whyte H, Bailey T. Simultaneous localization and mapping: part I[J]. IEEE robotics & automation magazine, 2006, 13(2): 99-110.

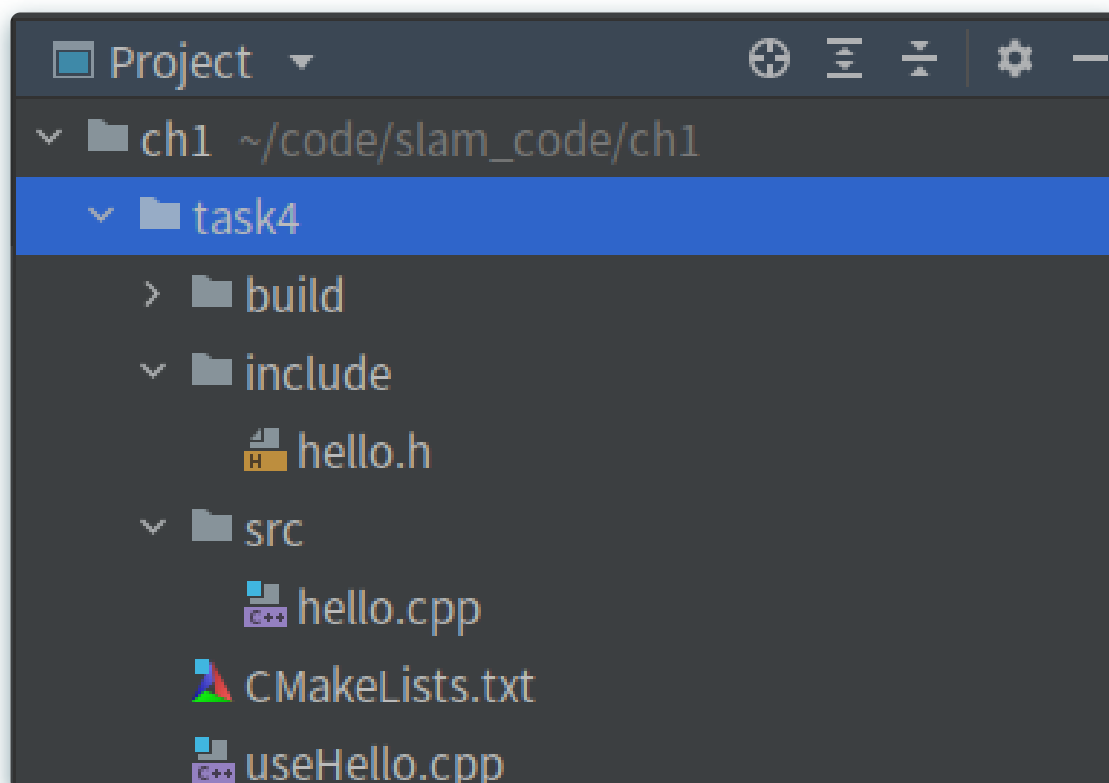
## ✓ 3. CMake练习

参考资料：CMake Practice.pdf文档

### 1-4题

阅读CMake Practice.pdf后，开始组织代码文件和编写CMakeLists.txt。之前也并没有怎么接触过CMakeLists.txt编写，所以在初步编写是也出现了各种各样的问题。经过，出错 -> 再看书 -> 出错 -> 网上搜 -> 编写，最后终于可以运行 1 - 4 要求的格式。

代码的组织架构如下图所示：

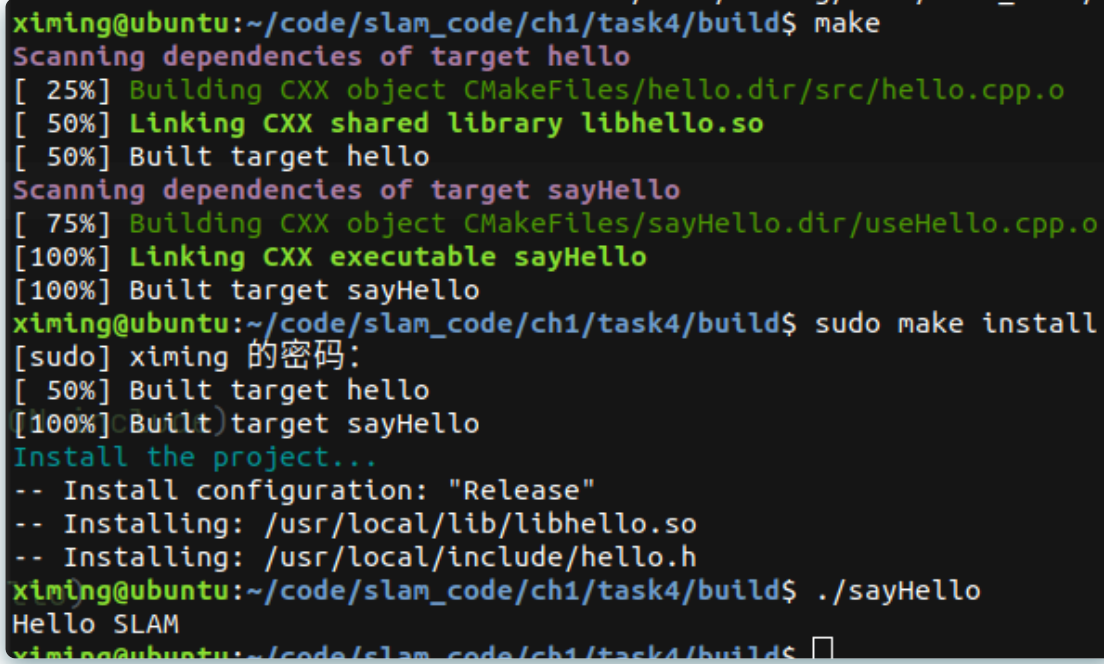




cmd运行指令:

```
1 mkdir build
2 cd build
3 cmake -DCMAKE_INSTALL_PREFIX=/usr/local ..
4 make
5 sudo make install
6 ./sayHello
```

运行结果截图:



```
ximing@ubuntu:~/code/slam_code/ch1/task4/build$ make
Scanning dependencies of target hello
[ 25%] Building CXX object CMakeFiles/hello.dir/src/hello.cpp.o
[ 50%] Linking CXX shared library libhello.so
[ 50%] Built target hello
Scanning dependencies of target sayHello
[ 75%] Building CXX object CMakeFiles/sayHello.dir/useHello.cpp.o
[100%] Linking CXX executable sayHello
[100%] Built target sayHello
ximing@ubuntu:~/code/slam_code/ch1/task4/build$ sudo make install
[sudo] ximing 的密码:
[ 50%] Built target hello
[100%] Built target sayHello
Install the project...
-- Install configuration: "Release"
-- Installing: /usr/local/lib/libhello.so
-- Installing: /usr/local/include/hello.h
ximing@ubuntu:~/code/slam_code/ch1/task4/build$ ./sayHello
Hello SLAM
ximing@ubuntu:~/code/slam_code/ch1/task4/build$
```

5.为我的库提供 FindHello.cmake 文件

遇到的问题 1: cmake无法找到FindHELLO.cmake文件



```
ximing@ubuntu: ~/code/slam_code/ch1/task4/build
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
CMake Warning at CMakeLists.txt:23 (FIND_PACKAGE):
  By not providing "FindHELLO.cmake" in CMAKE_MODULE_PATH this project has
  asked CMake to find a package configuration file provided by "HELLO", but
  CMake did not find one.

  Could not find a package configuration file provided by "HELLO" with any of
  the following names:

    HELLOConfig.cmake
    hello-config.cmake

  Add the installation prefix of "HELLO" to CMAKE_PREFIX_PATH or set
  "HELLO_DIR" to a directory containing one of the above files.  If "HELLO"
  provides a separate development package or SDK, be sure it has been
  installed.

-- Configuring done
-- Generating done
-- Build files have been written to: /home/ximing/code/slam_code/ch1/task4/build
```

网上查找资料，最后发现CMakeLists.txt文件中，下面的FIND\_PACKAGE(Hello REQUIRED)这句代码中，Hello需要与FindHello.cmake的大小写匹配才可。修改后可正常运行，截图如下，最后运行./sayHello可正常运行。其中FindHello.cmake和CMakeLists.txt的编写代码文件可见。

```
ximing@ubuntu: ~/code/slam_code/ch1/task4/build
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
MakeFiles install_manifest.txt Makefile
ximing@ubuntu:~/code/slam_code/ch1/task4/build$ cmake -DCMAKE_INSTALL_PREFIX=/u
/local ..
- The C compiler identification is GNU 7.5.0
- The CXX compiler identification is GNU 7.5.0
- Check for working C compiler: /usr/bin/cc
- Check for working C compiler: /usr/bin/cc -- works
- Detecting C compiler ABI info
- Detecting C compiler ABI info - done
- Detecting C compile features
- Detecting C compile features - done
- Check for working CXX compiler: /usr/bin/c++
- Check for working CXX compiler: /usr/bin/c++ -- works
- Detecting CXX compiler ABI info
- Detecting CXX compiler ABI info - done
- Detecting CXX compile features
- Detecting CXX compile features - done
home/ximing/code/slam_code/ch1/task4/cmake
- Found Hello: /usr/local/lib/libhello.so
home/ximing/code/slam_code/ch1/task4/cmake
- Configuring done
- Generating done
- Build files have been written to: /home/ximing/code/slam_code/ch1/task4/buil
ximing@ubuntu:~/code/slam_code/ch1/task4/build$ make
```

## ✓ 4. gflags, glog, gtest的使用

参考资料: <https://zhuanlan.zhihu.com/p/108477489>

<https://blog.csdn.net/alwaysrun/article/details/108418769>

### 1. gflags, glog, gtest的安装

#### gflags的安装

```
1 | git clone https://ghproxy.com/https://github.com/gflags/gflags.git
   | gflags_stable
2 | cd gflags_stable
3 | mkdir build
4 | cd build
5 | cmake ..
6 | make
7 | sudo make install
```

#### glog的安装

```
1 | git clone https://ghproxy.com/https://github.com/google/glog
2 | sudo apt-get install autoconf automake libtool
3 | mkdir build
4 | cd build
5 | cmake ..
6 | make
7 | sudo make install
```

初次编译glog时出错, 如下图所示, 网上搜索解决方案, 需要把gflag编译成动态库或者使用apt-get方法进行安装。

```
1 | 重新执行gflags的安装过程, 将gflag编译成动态链接库
2 | cmake -DBUILD_SHARED_LIBS=ON -DBUILD_STATIC_LIBS=ON -
   | DINSTALL_HEADERS=ON -DINSTALL_SHARED_LIBS=ON -
   | DINSTALL_STATIC_LIBS=ON -DCMAKE_INSTALL_PREFIX=/usr/ ..
```

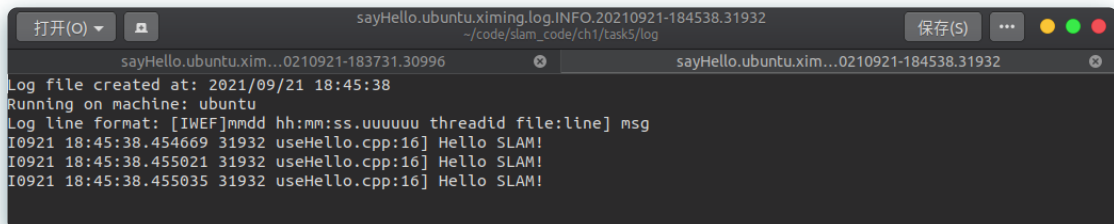
```
ximing@ubuntu:/usr/glog/build$ make
[ 4%] Linking CXX shared library libglog.so
/usr/bin/ld: /usr/local/lib/libgflags.a(gflags.cc.o): relocation R_X86_64_PC32 against symbol `stderr@GLIBC_2.2.5' can not be used when making a shared object;
recompile with -fPIC
/usr/bin/ld: 最后的链接失败: 错误的值
collect2: error: ld returned 1 exit status
CMakeFiles/glog.dir/build.make:251: recipe for target 'libglog.so.0.6.0' failed
make[2]: *** [libglog.so.0.6.0] Error 1
CMakeFiles/Makefile2:1079: recipe for target 'CMakeFiles/glog.dir/all' failed
make[1]: *** [CMakeFiles/glog.dir/all] Error 2
Makefile:162: recipe for target 'all' failed
make: *** [all] Error 2
ximing@ubuntu:/usr/glog/build$ sudo apt-get install glog
```

## gtest安装

```
1 git clone
  https://ghproxy.com/https://github.com/google/googletest.git
2 cd googletest
3 mkdir build
4 cd build
5 cmake ..
6 make
7 sudo make install
```

## 代码编写过程

2-3题：我直接将sayHello函数放入了useHello.cpp代码中，具体代码编写和CMakeLists.txt编写见代码文件。最后的日志文件如下图所示。



```
sayHello.ubuntu.ximing.log.INFO.20210921-184538.31932
~/code/slam_code/ch1/task5/log
sayHello.ubuntu.xim...0210921-183731.30996
sayHello.ubuntu.xim...0210921-184538.31932
Log file created at: 2021/09/21 18:45:38
Running on machine: ubuntu
Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
I0921 18:45:38.454669 31932 useHello.cpp:16] Hello SLAM!
I0921 18:45:38.455021 31932 useHello.cpp:16] Hello SLAM!
I0921 18:45:38.455035 31932 useHello.cpp:16] Hello SLAM!
```

由于本人能力有限，gtest的编写过程理解不深，没能写出来。

## ✓ 5. 理解ORB-SLAM2框架

### 1. ORB-SLAM2下载完成终端截图

```
/home/ximing
ximing@ubuntu:~$ git clone https://ghproxy.com/https://github.com/raulmur/ORB_SLAM2
正克隆到 'ORB_SLAM2'...
remote: Enumerating objects: 566, done.
remote: Total 566 (delta 0), reused 0 (delta 0), pack-reused 566
接收对象中: 100% (566/566), 41.41 MiB | 250.00 KiB/s, 完成.
处理 delta 中: 100% (182/182), 完成.
ximing@ubuntu:~$
```

## 2. 阅读 ORB-SLAM2 代码目录下的 CMakeLists.txt,回答问题:

- ORB-SLAM2将编译出什么结果?有几个库文件和可执行文件?

CMakeLists.txt的51-71行: 代码将由19个.cc文件生成一个动态链接库, 在/lib文件夹下的libORB\_SLAM2.so文件;

85-86行: 将会产生一个可执行文件 `rgbd_tum`;

91-92行: 将会产生一个可执行文件 `stereo_kitti`;

95-96行: 将会产生一个可执行文件 `stereo_euroc`;

102-103行: 将会产生一个可执行文件 `mono_tum`;

106-107行: 将会产生一个可执行文件 `mono_kitti`;

110-111行: 将会产生一个可执行文件 `mono_euroc`。

- ORB-SLAM2 中的 include, src, Examples 三个文件夹中都含有什么内容?

include和src文件夹下是一些需要编译成动态链接库文件的头文件和源代码。其中 Tracking.cc主要是从图像中提取ORB特征; LocalMapping.cc这部分用来完成局部地图构建。LoopClosing.cc分为两个过程, 分别是闭环探测和闭环校正。

Examples: 此目录下是Monocular (单目)、RGB-D、ROS和Stereo文件

- ORB-SLAM2 中的可执行文件链接到了哪些库?它们的名字是什么?

以stereo\_kitti为例, 可执行文件链接到了变量`${PROJECT_NAME}`\$, 而此变量是src文件夹下的代码文件构建的动态链接库 `libORB_SLAM2.so`。

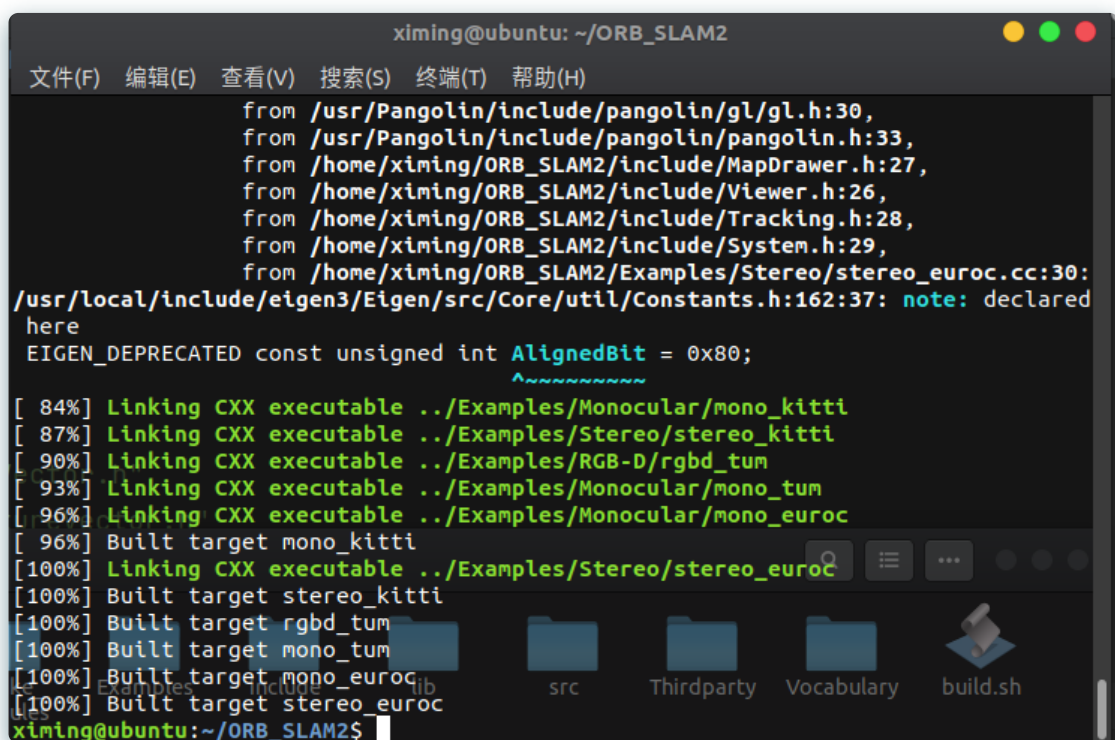
```
add_executable(stereo_kitti
Examples/Stereo/stereo_kitti.cc)
target_link_libraries(stereo_kitti ${PROJECT_NAME})
```

另外还有OpenCV\_LIBS、EIGEN3\_LIBS、Pangolin\_LIBRARIES、libDBow2.so、libg2o.so库文件。

## ✓ 6. \*使用摄像头或视频运行ORB-SLAM2

1. 编译 ORB-SLAM2截图，因为之前安装过pangolin、opencv和eigen3等库，所以在编译这一步没有出什么大的问题，且参照群里对于usleep的错误解决方案，解决了这个问题。

```
1 # 执行编译的命令
2 cd ORB_SLAM2
3 chmod +x build.sh
4 ./build.sh
```

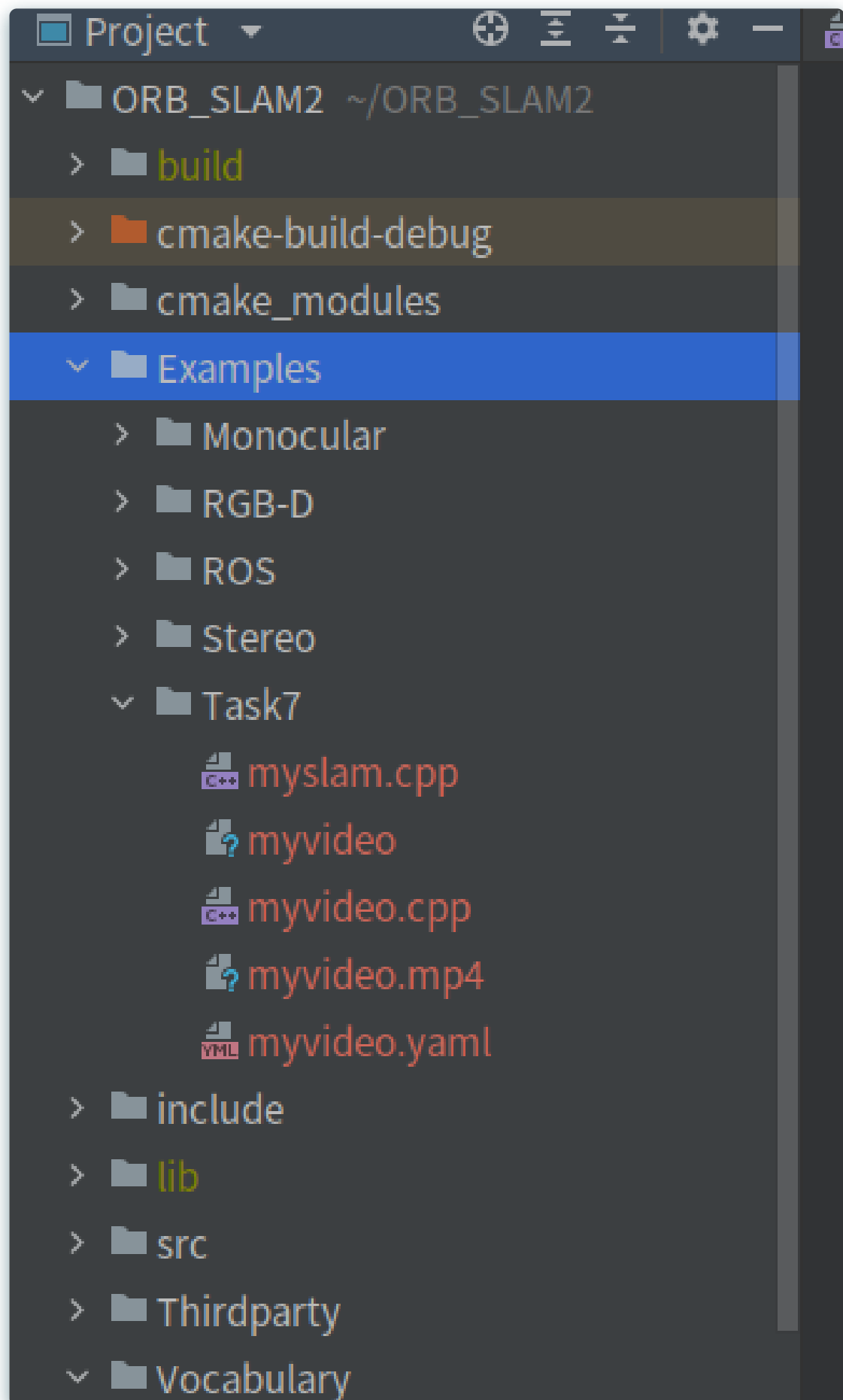


```
ximing@ubuntu: ~/ORB_SLAM2
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

from /usr/Pangolin/include/pangolin/gl/gl.h:30,
from /usr/Pangolin/include/pangolin/pangolin.h:33,
from /home/ximing/ORB_SLAM2/include/MapDrawer.h:27,
from /home/ximing/ORB_SLAM2/include/Viewer.h:26,
from /home/ximing/ORB_SLAM2/include/Tracking.h:28,
from /home/ximing/ORB_SLAM2/include/System.h:29,
from /home/ximing/ORB_SLAM2/Examples/Stereo/stereo_euroc.cc:30:
/usr/local/include/eigen3/Eigen/src/Core/util/Constants.h:162:37: note: declared
here
EIGEN_DEPRECATED const unsigned int AlignedBit = 0x80;
[ 84%] Linking CXX executable ../Examples/Monocular/mono_kitti
[ 87%] Linking CXX executable ../Examples/Stereo/stereo_kitti
[ 90%] Linking CXX executable ../Examples/RGB-D/rgbd_tum
[ 93%] Linking CXX executable ../Examples/Monocular/mono_tum
[ 96%] Linking CXX executable ../Examples/Monocular/mono_euroc
[ 96%] Built target mono_kitti
[100%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[100%] Built target stereo_kitti
[100%] Built target rgbd_tum
[100%] Built target mono_tum
[100%] Built target mono_euroc
[100%] Built target stereo_euroc
ximing@ubuntu:~/ORB_SLAM2$
```

2. 如何将 myslam.cpp或 myvideo.cpp 加入到 ORB-SLAM2 工程中?请给出你的 CMakeLists.txt 修改方案。

如下图所示，在Examples文件夹中创建Task7文件夹，将提供的代码myvideo.cpp等放置到Task7文件夹下。



CMakeLists.txt文件编写：

在ORB-SLAM2原代码基础上，添加如下命令

```

1 | set(CMAKE_RUNTIME_OUTPUT_DIRECTORY
   | ${PROJECT_SOURCE_DIR}/Examples/Task7)
2 | add_executable(myvideo
   |     Examples/Task7/myvideo.cpp)
3 | target_link_libraries(myvideo ${PROJECT_NAME})

```

### 3. 运行 ORB-SLAM2

初次运行时报出如下错误，怀着忐忑的心情在群里@了助教韩爽，后来经过提示发现原来是myvideo.cpp中一些路径问题没有修改对。这里要特别感谢助教们的热心解答，即便是在中秋节也很热心的在群里帮助大家解决问题。感谢！！

```

mySLAM.cpp myvideo myvideo.cpp myvideo.mp4 myvideo.yaml
xing@ubuntu:~/ORB_SLAM2/Examples/Task7$ ./myvideo

ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.
This program comes with ABSOLUTELY NO WARRANTY;
This is free software and you are welcome to redistribute it
under certain conditions. See LICENSE.txt.

Input sensor was set to: Monocular

Loading ORB Vocabulary. This could take a while...

```

成功运行orb-slam2的测试文件，如下图所示：

