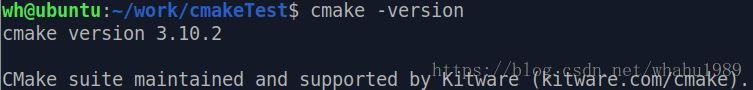
原文：<https://blog.csdn.net/whahu1989/article/details/82078563>

## 一 安装CMake

本文使用ubuntu18.04，安装cmake使用如下命令，  
sudo apt install cmake  
安装完成后，在终端下输入cmake -version查看cmake版本，  
  
这样cmake就安装好了。

## 二 简单样例

首先让我们从最简单的代码入手，先来体验下cmake是如何操作的。编写main.c，如下，

#include <stdio.h>

int main(void)

{

    printf("Hello World\n");

    return 0;

}

然后在main.c相同目录下编写CMakeLists.txt，内容如下，

cmake\_minimum\_required(VERSION 2.8)

project(demo)

add\_executable(main main.c)

第一行意思是表示cmake的最低版本要求是2.8，我们安装的是3.10.2；

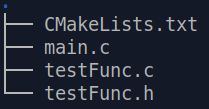
第二行是表示本工程信息，也就是工程名叫demo；

第三行比较关键，表示最终要生成的elf文件的名字叫main，使用的源文件是main.c  
在终端下切到main.c所在的目录下，然后输入以下命令运行cmake，  
 cmake .  
可以看到成功生成了Makefile，还有一些cmake运行时自动生成的文件。  
然后在终端下输入make并回车，

make  
可以看到我们需要的elf文件main也成功生成了，然后运行main，  
 ./main  
运行成功！

PS: 如果想重新生成main，输入make clean就可以删除main这个elf文件。

## 三 同一目录下多个源文件

接下来进入稍微复杂的例子：在同一个目录下有多个源文件。  
在之前的目录下添加2个文件，testFunc.c和testFunc.h。添加完后整体文件结构如下，  


修改CMakeLists.txt，在add\_executable的参数里把testFunc.c加进来

cmake\_minimum\_required(VERSION 2.8)

project(demo)

add\_executable(main main.c testFunc.c)

重新执行cmake生成Makefile并运行make，

如下命令可以把指定目录下所有的源文件存储在一个变量中，这个命令就是 **aux\_source\_directory(dir var)**。  
第一个参数dir是指定目录，第二个参数var是用于存放源文件列表的变量。

我们在main.c所在目录下再添加2个文件，testFunc1.c和testFunc1.h。添加完后整体文件结构如下，  


修改CMakeLists.txt，

cmake\_minimum\_required (VERSION 2.8)

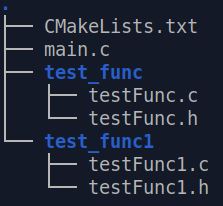
project (demo)

aux\_source\_directory(. SRC\_LIST)

add\_executable(main ${SRC\_LIST})

aux\_source\_directory把当前目录下的源文件存列表存放到变量SRC\_LIST里，然后在add\_executable里调用SRC\_LIST（注意调用变量时的写法）。  
再次执行cmake和make，并运行main，  
可以看到运行成功了。

## 四 不同目录下多个源文件

我们把之前的源文件整理一下（新建2个目录test\_func和test\_func1），整理好后整体文件结构如下，  
  
其中，CMakeLists.txt和main.c在同一目录下，内容修改成如下所示，

cmake\_minimum\_required (VERSION 2.8)

project (demo)

include\_directories (test\_func test\_func1)

aux\_source\_directory (test\_func SRC\_LIST)

aux\_source\_directory (test\_func1 SRC\_LIST1)

add\_executable (main main.c ${SRC\_LIST} ${SRC\_LIST1})

include\_directories：用来向工程添加多个指定头文件的搜索路径，多个路径用空格分隔。

如下：

main.c直接直接引用头文件

#include "testFunc.h"

#include "testFunc.c"

如不使用，则需在main.c里使用include来指定路径，如下

#include "test\_func/testFunc.h"

#include "test\_func1/testFunc1.h"

## 五 正规一点的组织结构

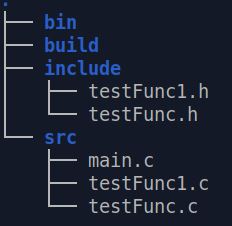
正规一点来说，

源文件放到src目录下

头文件放入到include文件下

生成的对象文件放入到build目录下

最终输出的elf文件会放到bin目录下

这样整个结构更加清晰。让我们把前面的文件再次重新组织下，  


我们在最外层目录下新建一个CMakeLists.txt，内容如下，

cmake\_minimum\_required (VERSION 2.8)

project (demo)

add\_subdirectory (src)

add\_subdirectory()：这个命令可以向当前工程添加存放源文件的子目录，当执行cmake时，就会进入src目录下去找src目录下的CMakeLists.txt

在src目录下也建立一个CMakeLists.txt，内容如下，

aux\_source\_directory (. SRC\_LIST)

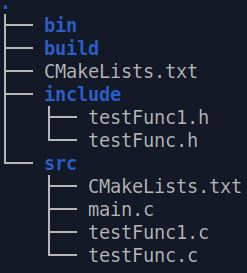
include\_directories (../include)

add\_executable (main ${SRC\_LIST})

set (EXECUTABLE\_OUTPUT\_PATH ${PROJECT\_SOURCE\_DIR}/bin)

set()：是用于定义变量的，EXECUTABLE\_OUT\_PATH和PROJECT\_SOURCE\_DIR是CMake自带的预定义变量，其意义如下，

* EXECUTABLE\_OUTPUT\_PATH ：生成的目标文件的存放位置
* PROJECT\_SOURCE\_DIR：工程的根目录

添加好以上这2个CMakeLists.txt后，整体文件结构如下，  
  
下面来运行cmake，**不过这次先让我们切到build目录下**，然后输入以下命令，  
 cmake ..  
Makefile会在build目录下生成，然后在build目录下运行make，

这里解释一下为什么在build目录下运行cmake？从前面几个case中可以看到，如果不这样做，cmake运行时生成的附带文件就会跟源码文件混在一起，这样会对程序的目录结构造成污染

## 六 动态库和静态库的编译控制

有时我们只需要编译出动态库，静态库，然后等着让其它程序去使用。  
  
我们会在build目录下运行cmake，并把生成的库文件存放到lib目录下。

最外层的CMakeLists.txt内容如下，

cmake\_minimum\_required (VERSION 2.8)

project (demo)

add\_subdirectory (lib\_testFunc)

lib\_testFunc目录下的CMakeLists.txt如下，

aux\_source\_directory (. SRC\_LIST)

add\_library (testFunc\_shared SHARED ${SRC\_LIST})

add\_library (testFunc\_static STATIC ${SRC\_LIST})

set\_target\_properties (testFunc\_shared PROPERTIES OUTPUT\_NAME "testFunc")

set\_target\_properties (testFunc\_static PROPERTIES OUTPUT\_NAME "testFunc")

set (LIBRARY\_OUTPUT\_PATH ${PROJECT\_SOURCE\_DIR}/lib)

这里又出现了新的命令和预定义变量，

* add\_library: 生成动态库或静态库 (第1个参数指定库的名字；第2个参数决定是动态还是静态，如果没有就默认静态；第3个参数指定生成库的源文件)
* set\_target\_properties: 设置输出的名称，还有其它功能，如设置库的版本号等等
* LIBRARY\_OUTPUT\_PATH: 库文件的默认输出路径，这里设置为工程目录下的lib目录

## 七 对库进行链接

前面讲解生成库，现在进行库连接，整体结构如下  
  
修改工程目录下的CMakeLists.txt，如下，

cmake\_minimum\_required (VERSION 2.8)

project (demo)

add\_subdirectory (lib\_testFunc)

add\_subdirectory (src)

src目录下的CMakeLists.txt如下，

aux\_source\_directory (. SRC\_LIST)

# find testFunc.h

include\_directories (../lib\_testFunc)

link\_directories (${PROJECT\_SOURCE\_DIR}/lib)

add\_executable (main ${SRC\_LIST})

target\_link\_libraries (main testFunc)

set (EXECUTABLE\_OUTPUT\_PATH ${PROJECT\_SOURCE\_DIR}/bin)

这里出现2个新的命令，

* link\_directories: 添加非标准的共享库搜索路径
* target\_link\_libraries: 把目标文件与库文件进行链接

ps：target\_link\_libraries (main testFunc) 默认是使用动态库，如果lib目录下只有静态库，那么这种写法就会去链接静态库。也可以直接指定使用动态库还是静态库，写法是：target\_link\_libraries (main libtestFunc.so)或target\_link\_libraries (main libtestFunc.a)

ps： 查看elf文件使用了哪些库，可以使用readelf -d ./xx来查看

## 八 添加编译选项

有时编译程序时想添加一些编译选项，如-Wall，-std=c++11等，就可以使用add\_compile\_options来进行操作。

CMakeLists.txt内容如下，

cmake\_minimum\_required (VERSION 2.8)

project (demo)

set (EXECUTABLE\_OUTPUT\_PATH ${PROJECT\_SOURCE\_DIR}/bin)

add\_compile\_options(-std=c++11 -Wall)

add\_executable(main main.cpp)

## 九 添加控制选项

有时希望在编译代码时只编译一些指定的源码，例如本来要编译生成多个bin或库文件，现在只想生成某些指定的bin或库文件，这时可以使用cmake的option命令。

这里仍然使用例子来解释，假设我们现在的工程会生成2个bin文件，main1和main2，现在整体结构体如下，  
  
外层的CMakeLists.txt内容如下，

cmake\_minimum\_required(VERSION 2.8)

project(demo)

option(MYDEBUG "enable debug compilation" OFF)

set (EXECUTABLE\_OUTPUT\_PATH ${PROJECT\_SOURCE\_DIR}/bin)

add\_subdirectory(src)

这里使用了option命令，其第一个参数是这个option的名字，第二个参数是字符串，用来描述这个option是来干嘛的，第三个是option的值，ON或OFF，也可以不写，不写就是默认OFF。

然后编写src目录下的CMakeLists.txt，如下

cmake\_minimum\_required (VERSION 2.8)

add\_executable(main1 main1.c)

if (MYDEBUG)

add\_executable(main2 main2.c)

else()

message(STATUS "Currently is not in debug mode")

endif()

注意，这里使用了if-else来根据option来决定是否编译main2.c  
其中main1.c和main2.c的内容如下，

// main1.c#include <stdio.h>

int main(void){

printf("hello, this main1\n");

return 0;

}

// main2.c#include <stdio.h>

int main(void){

printf("hello, this main2\n");

return 0;

}

然后cd到build目录下输入cmake .. && make就可以只编译出main1，如果想编译出main2，就把MYDEBUG设置为ON，再次输入cmake .. && make重新编译。

每次想改变MYDEBUG时都需要去修改CMakeLists.txt，有点麻烦，其实可以通过cmake的命令行去操作，例如我们想把MYDEBUG设置为OFF，先cd到build目录，然后输入cmake .. -DMYDEBUG=ON，这样就可以编译出main1和main2 （在bin目录下）  
