爱就是恒久忍耐
专家
码龄9年
暂无认证

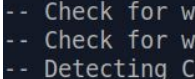
Linux下CMake简明教程

原创 版权

爱就是恒久忍耐 2018-08-26 23:14:45 69016 收藏 947

分类专栏： C/C++ 编译

CMake是开源、跨平台的构建工具，可以让我们通过编写简单的配置文件去生成本地的Makefile，这个配置文件是**独立于运行平台和编译器**的，这样就不用亲自去编写Makefile了，而且配置文件可以直接拿到其它平台上使用，无需修改，非常方便。



本文主要讲述在Linux下如何使用CMake来编译我们的程序。

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，

```
wh@ubuntu:~/work/cmakeTest$ cmake -version
cmake version 3.10.2

CMake suite maintained and supported by Kitware (kitware.com/cmake).
```


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，

```c
#include <stdio.h>
int main(void)
{
 printf("Hello World\n");
 return 0;
}
```

与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)
```

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 .
```

会输出如下信息，

```
wh@ubuntu:~/work/cmakeTest/demo5$ cmake .
-- The C compiler identification is GNU 7.3.0
-- The CXX compiler identification is GNU 7.3.0
-- 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
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wh/work/cmakeTest/demo5
```


再来看看目录下的文件，

```
wh@ubuntu:~/work/cmakeTest/demo5$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  CMakeLists.txt  main.c  Makefile
```


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，

```
wh@ubuntu:~/work/cmakeTest/demo5$ make
Scanning dependencies of target main
[ 50%] Building C object CMakeFiles/main.dir/main.c.o
[100%] Linking C executable main
[100%] Built target main
```


可以看到执行cmake生成的Makefile可以显示进度，并带颜色。再看下目录下的文件，

目录


```
wh@ubuntu:~/work/cmakeTest/demo5$ ls
CMakeCache.txt  cmake_install.cmake  main      Makefile
CMakeFiles      CMakeLists.txt       main.c
```


可以看到我们需要的elf文件main也成功生成了，然后运行main，

```
wh@ubuntu:~/work/cmakeTest/demo5$ ./main
Hello World
```

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，

```
├── CMakeLists.txt
├── main.c
├── testFunc.c
└── testFunc.h
```


testFunc.c内容如下，

```c
/*
** testFunc.c
*/
#include <stdio.h>
#include "testFunc.h"
void func(int data)
{
 printf("data is %d\n", data);
}
```

nc.h内容如下，

```c
/*
** testFunc.h
*/
#ifndef _TEST_FUNC_H_
#define _TEST_FUNC_H_
void func(int data);
#endif
```

ain.c，调用testFunc.h里声明的函数func()，

```c
#include <stdio.h>
#include "testFunc.h"
int main(void)
{
 func(100);
 return 0;
}
```

MakeLists.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)
```

新执行cmake生成Makefile并运行make，

```
wh@ubuntu:~/work/cmakeTest/demo5$ cmake .
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wh/work/cmakeTest/demo5
wh@ubuntu:~/work/cmakeTest/demo5$ make
Scanning dependencies of target main
[ 33%] Building C object CMakeFiles/main.dir/main.c.o
[ 66%] Building C object CMakeFiles/main.dir/testFunc.c.o
[100%] Linking C executable main
[100%] Built target main
```

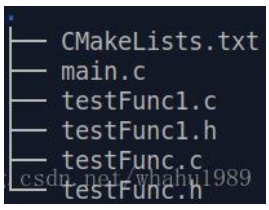

然后运行重新生成的elf文件main，

```
wh@ubuntu:~/work/cmakeTest/demo5$ ./main
data is 100
```


运行成功！

可以类推，如果在同一目录下有多个源文件，那么只要在add_executable里把所有源文件都添加进去就可以了。但是如果有一百个源文件，再这样做就有点坑了，无法体现cmake的优越性，cmake提供了一个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，



testFunc1.c如下，

```
/*
** testFunc1.c
*/
#include <stdio.h>
#include "testFunc1.h"
void func1(int data)
{
 printf("data is %d\n", data);
}
```

nc1.h如下，

```
/*
** testFunc1.h
*/
#ifndef _TEST_FUNC1_H_
#define _TEST_FUNC1_H_
void func1(int data);
#endif
```

main.c，调用testFunc1.h里声明的函数func1()，

```
#include <stdio.h>
#include "testFunc.h"
#include "testFunc1.h"
int main(void)
{
 func(100);
 func1(200);
 return 0;
}
```

MakeLists.txt，

```
cmake_minimum_required (VERSION 2.8)
project (demo)
aux_source_directory(. SRC_LIST)
add_executable(main ${SRC_LIST})
```

x_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
xecutable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，

```
wh@ubuntu:~/work/cmakeTest/demo5$ cmake .
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wh/work/cmakeTest/demo5
wh@ubuntu:~/work/cmakeTest/demo5$ make
Scanning dependencies of target main
[ 25%] Building C object CMakeFiles/main.dir/main.c.o
[ 50%] Building C object CMakeFiles/main.dir/testFunc.c.o
[ 75%] Building C object CMakeFiles/main.dir/testFunc1.c.o
[100%] Linking C executable main
[100%] Built target main
wh@ubuntu:~/work/cmakeTest/demo5$ ./main
data is 100
data is 200
```

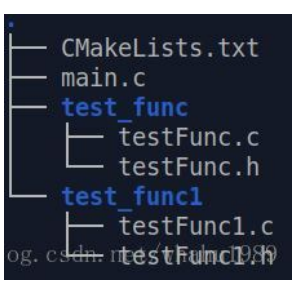

可以看到运行成功了。

aux_source_directory()也存在弊端，它会把指定目录下的所有源文件都加进来，可能会加入一些我们不需要的文件，此时我们可以使用set命令去新建变量来存放需要的源文件，如下，

```
cmake_minimum_required (VERSION 2.8)
project (demo)
set( SRC_LIST
  ./main.c
  ./testFunc1.c
  ./testFunc.c)
add_executable(main ${SRC_LIST})
```

四 不同目录下多个源文件

一般来说，当程序文件比较多时，我们会进行分类管理，把代码根据功能放在不同的目录下，这样方便查找。那么这种情况下如何编写CMakeLists.txt呢？
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，



把之前的testFunc.c和testFunc.h放到test_func目录下，testFunc1.c和testFunc1.h则放到test_func1目

录下。

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})
```

现了一个新的命令：include_directories。该命令是用来向工程添加多个指定头文件的搜索路径之间用空格分隔。
因为main.c里include了testFunc.h和testFunc1.h，如果没有这个命令来指定头文件所在位置，就会无法编译。当然，也可以在main.c里使用include来指定路径，如下

```
#include "test_func/testFunc.h"
#include "test_func1/testFunc1.h"
```

只是这种写法不好看。
另外，我们使用了2次aux_source_directory，因为源文件分布在2个目录下，所以添加2次。

五 正规一点的组织结构

正规一点来说，一般会把源文件放到src目录下，把头文件放入到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)
```

现一个新的命令add_subdirectory()，这个命令可以向当前工程添加存放源文件的子目录，并可以指定中间二进制和目标二进制的存放位置，具体用法可以百度。
这里指定src目录下存放了源文件，当执行cmake时，就会进入src目录下去找src目录下的CMakeLists.txt，所以在src目录下也建立一个CMakeLists.txt，内容如下，

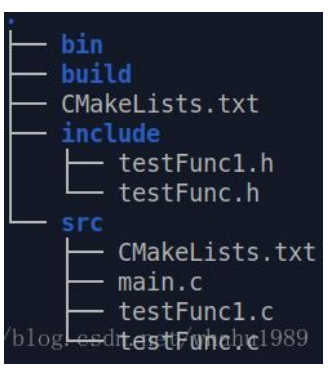
```
aux_source_directory (. SRC_LIST)
include_directories (../include)
add_executable (main ${SRC_LIST})
set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
```

出现一个新的命令set，是用于定义变量的，EXECUTABLE_OUT_PATH和ECT_SOURCE_DIR是CMake自带的预定义变量，其意义如下，

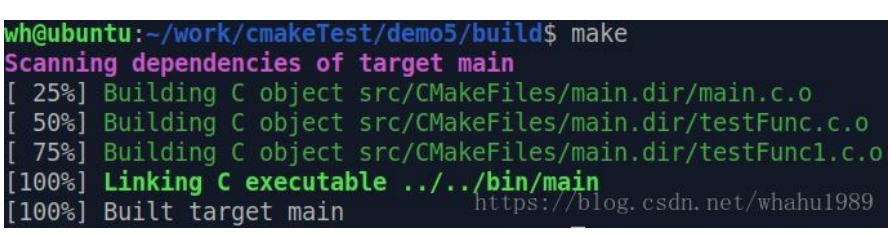
- EXECUTABLE_OUTPUT_PATH ：目标二进制可执行文件的存放位置
- PROJECT_SOURCE_DIR：工程的根目录

所以，这里set的意思是把存放elf文件的位置设置为工程根目录下的bin目录。（cmake有很多预定义变量，详细的可以网上搜索一下）

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，



运行ok，我们再切到bin目录下，发现main已经生成，并运行测试，

```
wh@ubuntu:~/work/cmakeTest/demo5/bin$ ls
main
wh@ubuntu:~/work/cmakeTest/demo5/bin$ ./main
data is 100
data is 200
```


测试OK！

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，而在build目录下运行cmake，生成的附带文件就只会待在build目录下，如果我们不想要这些文件了就可以直接清空build目录，非常方便。

另外一种写法：
前面的工程使用了2个CMakeLists.txt，最外层的CMakeLists.txt用于掌控全局，使用add_subdirectory来控制其它目录下的CMakeLists.txt的运行。

上面的例子也可以只使用一个CMakeLists.txt，把最外层的CMakeLists.txt内容改成如下，

```
cmake_minimum_required (VERSION 2.8)
project (demo)
set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
aux_source_directory (src SRC_LIST)
include_directories (include)
add_executable (main ${SRC_LIST})
```

还要把src目录下的CMakeLists.txt删除。

六 动态库和静态库的编译控制

有时只需要编译出动态库和静态库，然后等着让其它程序去使用。让我们看下这种情况该如何使用cmake。首先按照如下重新组织文件，只留下testFunc.h和TestFunc.c，

```
.
├── build
├── CMakeLists.txt
├── lib
└── testFunc
    ├── testFunc.c
    └── testFunc.h

3 directories, 3 files
```

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。
CMakeLists.txt内容如下，

```
cmake_minimum_required (VERSION 3.5)
project (demo)
set (SRC_LIST ${PROJECT_SOURCE_DIR}/testFunc/testFunc.c)
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)
```

出现了新的命令和预定义变量，

d_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: 设置最终生成的库的名称，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

好了，让我们进入build目录下运行`cmake ..`，成功后再运行make，

```
wh@ubuntu:~/work/cmakeTest/demo6/build$ make
Scanning dependencies of target testFunc_shared
[ 25%] Building C object lib_testFunc/CMakeFiles/testFunc_shared.dir/testFunc.c.o
[ 50%] Linking C shared library ../../lib/libtestFunc.so
[ 50%] Built target testFunc_shared
Scanning dependencies of target testFunc_static
[ 75%] Building C object lib_testFunc/CMakeFiles/testFunc_static.dir/testFunc.c.o
[100%] Linking C static library ../../lib/libtestFunc.a
[100%] Built target testFunc_static
```


cd到lib目录下进行查看，发现已经成功生成了动态库和静态库，

```
wh@ubuntu:~/work/cmakeTest/demo6/lib$ ls
libtestFunc.a  libtestFunc.so
```


PS：前面使用set_target_properties重新定义了库的输出名称，如果不使用set_target_properties也可以，那么库的名称就是add_library里定义的名称，只是连续2次使用add_library指定库名称时（第一个参数），这个名称不能相同，而set_target_properties可以把名称设置为相同，只是最终生成的库文件后缀不同（一个是.so，一个是.a），这样相对来说会好看点。

七 对库进行链接

既然我们已经生成了库，那么就进行链接测试下。重新建一个工程目录，然后把上节生成的库拷贝过来，然后在在工程目录下新建src目录和bin目录，在src目录下添加一个main.c，整体结构如下，

main.c内容如下，

```c
#include <stdio.h>
#include "testFunc.h"
int main(void)
{
    func(100);

    return 0;
}
```

录下的CMakeLists.txt内容如下，

```
cmake_minimum_required (VERSION 3.5)
project (demo)
set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
set (SRC_LIST ${PROJECT_SOURCE_DIR}/src/main.c)
# find testFunc.h
include_directories (${PROJECT_SOURCE_DIR}/testFunc/inc)
find_library(TESTFUNC_LIB testFunc HINTS ${PROJECT_SOURCE_DIR}/testFunc/lib)
add_executable (main ${SRC_LIST})
target_link_libraries (main ${TESTFUNC_LIB})
```

现2个新的命令，

d_library: 在指定目录下查找指定库，并把**库的绝对路径**存放到变量里，其第一个参数是变量称，第二个参数是库名称，第三个参数是HINTS，第4个参数是路径，其它用法可以参考ake文档

get_link_libraries: 把目标文件与库文件进行链接

使用find_library的好处是在执行`cmake ..`时就会去查找库是否存在，这样可以提前发现错误，不用等到链接时。

cd到build目录下，然后运行`cmake .. && make`，最后进入到bin目录下查看，发现main已经生成，运行之，

运行成功！

ps：在lib目录下有testFunc的静态库和动态库，`find_library(TESTFUNC_LIB testFunc ...`默认是查找动态库，如果想直接指定使用动态库还是静态库，可以写成`find_library(TESTFUNC_LIB libtestFunc.so ...`或者`find_library(TESTFUNC_LIB libtestFunc.a ...`

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

之前本节教程使用的是库查找方法是link_directories，但是很多读者反映运行时有问题，本人去官方文档上查了下，发现不建议使用了，推荐使用find_library或者find_package

link_directories ¶

Specify directories in which the linker will look for libraries.

```
link_directories(directory1 directory2 ...)
```

Specify the paths in which the linker should search for libraries. The command will apply only to targets created after it is called. Relative paths given to this command are interpreted as relative to the current source directory, see CMP0015.

Note that this command is rarely necessary. Library locations returned by find_package() and find_library() are absolute paths. Pass these absolute library file paths directly to the target_link_libraries() command. CMake will ensure the linker finds them.

八 添加编译选项

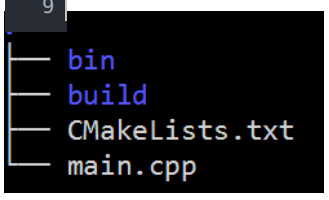
有时编译程序时想添加一些编译选项，如-Wall，-std=c++11等，就可以使用add_compile_options来进行操作。
这里以一个简单程序来做演示，main.cpp如下

```c
#include <iostream>
int main(void)
{
    auto data = 100;
    std::cout << "data: " << data << "\n";
    return 0;
}
```

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)
```

录结构如下，



然后cd到build目录下，执行`cmake .. && make`命令，就可以在bin目录下得到main的elf文件

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，可以使用cmake的option命令，主要遇到的情况分为2种：

1. 本来要生成多个bin或库文件，现在只想生成部分指定的bin或库文件
2. 对于同一个bin文件，只想编译其中部分代码（使用宏来控制）

第1种情况

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 3.5)
project(demo)
option(MYDEBUG "enable debug compilation" OFF)
set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
add_subdirectory(src)
```

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 3.5)
add_executable(main1 main1.c)
if (MYDEBUG)
    add_executable(main2 main2.c)
else()
    message(STATUS "Currently is not in debug mode")
endif()
```

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）

```
work@ubuntu:~/work/cmakeLearn/test002/build$ make
Scanning dependencies of target main2
[ 25%] Building C object src/CMakeFiles/main2.dir/main2.c.o
[ 50%] Linking C executable ../../bin/main2
[ 50%] Built target main2
Scanning dependencies of target main1
[ 75%] Building C object src/CMakeFiles/main1.dir/main1.c.o
[100%] Linking C executable ../../bin/main1
[100%] Built target main1
```


第2种情况

假设我们有个main.c，其内容如下，

```c
#include <stdio.h>
int main(void)
{
#ifdef WWW1
    printf("hello world1\n");
#endif
#ifdef WWW2
    printf("hello world2\n");
#endif
    return 0;
}
```

过定义宏来控制打印的信息，我们CMakeLists.txt内容如下，

```
cmake_minimum_required(VERSION 3.5)
project(demo)
set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
option(WWW1 "print one message" OFF)
option(WWW2 "print another message" OFF)
if (WWW1)
    add_compile_options(-DWWW1)
endif()
if (WWW2)
    add_compile_options(-DWWW2)
endif()
add_executable(main main.c)
```

option的名字保持和main.c里的宏名称一致，这样更加直观，也可以选择不同的名字。通过与
ompile_options()的配合，就可以控制单个bin文件的打印输出了。

程结构如下，

```
.
├── bin
├── build
├── CMakeLists.txt
└── main.c

2 directories, 2 files
```

cd到build目录下执行`cmake .. && make`，然后到bin目录下执行`./main`，可以看到打印为空，
接着分别按照下面指令去执行，然后查看打印效果，

- `cmake .. -DWWW1=ON -DWWW2=OFF && make`
- `cmake .. -DWWW1=OFF -DWWW2=ON && make`
- `cmake .. -DWWW1=ON -DWWW2=ON && make`

这里有个小坑要注意下：假设使用cmake设置了一个option叫A，下次再设置别的option例如B，如果没有删除上次执行cmake时产生的缓存文件，那么这次虽然没设置A，也会默认使用A上次的option值。

所以如果option有变化，要么删除上次执行cmake时产生的缓存文件，要么把所有的option都显式的指定其值。

十 总结

以上是自己学习CMake的一点学习记录，通过简单的例子让大家入门CMake，学习的同时也阅读了很多网友的博客。CMake的知识点还有很多，具体详情可以在网上搜索。总之，CMake可以让我们不用去编写复杂的Makefile，并且跨平台，是个非常强大并值得一学的工具。

如果有写的不对的地方，希望能留言指正，谢谢阅读。

TChuancey： 奈何老夫没文化，一句卧槽行天下。卧槽，牛逼 1 年前 回复 •••

8

码皇 爱就是恒久忍耐 博主 回复： 马哥本尊？ 1 年前 回复 •••

大大的开发： 从头到尾每个字看了一遍，写的非常好和精炼。从完全不懂到可以基本工作了，谢谢作者！ 1 年前 回复 •••

4

码皇 爱就是恒久忍耐 博主 回复： 谢谢阅读 1 年前 回复 •••

CppIsDragonSword： 全网写的最好的，没有之一。谢谢博主 3 月前 回复 •••

1

佛大型男： 膜拜 2 天前 回复 •••

ASURAz： 非常感谢博主 13 天前 回复 •••

langyuebzh： find_library() 默认查找的是静态库， 不是动态库 17 天前 回复 •••

码皇 爱就是恒久忍耐 博主 回复： 哦，那可能是老版本的问题，我文中用的是3.3以上的版本 16 天前 回复 •••

langyuebzh 回复 爱就是恒久忍耐： UOS CMake 3.3.2,光指定名称不加后缀.so，编译就报找不到.a文件，加上.so就没问题。 16 天前 回复 •••

码皇 爱就是恒久忍耐 博主 回复： 经过重新测试，默认查找的是动态库。Debian10, CMake 3.13.4 17 天前 回复 •••

a81842148840： 大家都太懒，第九条添加控制选项的小坑如果说的再清楚一点就完美了，瑕不掩瑜，我是跪着看完的 22 天前 回复 •••

在下三岁陈： 看着您的教程我能打出来 让我自己弄还是会出现问题 能不能向您请教一下 2 月前 回复 •••

在下三岁陈： 还是没太看懂怎么对库进行链接 2 月前 回复 •••

hawanglc： 太优秀的手册 2 月前 回复 •••

< 1 2 3 4 5 6 >

相关推荐

面向Linux C++的CMake简明教程(Jetson Nano)_冰海的博...

5-24

本教程面向Linux系统,重点讲解CMake的基本使用方法,用于构建C++项目。采用的平台为Jetson Nano嵌入式开发板,arm64系统。当然,本教程同样适合绝大部分Linux平台。在讲解CMake使用方法之前,先要了解gcc、make和MakeFile的概念。

cmake简明教程-半小时从入门到精通_MyBlog

5-10

cmake简明教程-半小时从入门到精通 - qccz123456的博客 - CSDN博客 https://blog.csdn.net/qccz123456/article/...etails/80639817 === cmake 从放弃到入门_Linux教程_Linux公社-Linux系统门户网站 https://www...

Linux下用cmake编译大型C/C++项目

SunEve

1万+

Linux下用cmake编译大型C/C++项目

Linux下cmake简易教程

冰糖葫芦的夏天的博客

1528

0. CMake介绍： CMake是一个跨平台的构建系统生成工具。它使用平台无关的CMake清单文件CMakeLists.txt，指定工程的构建过程；源码树的每个路径下都有这个文件。CMake产生一个适用于具体平台的构建系统，用户使用这个系统构建自己的工程。 一个工程或项目作管理时，咱们在Linux/unix或ELinux下采用Makefile,CodeBlocks,KDevelop,

cmake的一些基础知识

Deep Learning and NLP Farm

2万+

本博文的大概框架：内里有重复，待有时间重新整理 1， cmake 的介绍，下载，安装和使用 2， cmake 的手册详...，我关注了 -C和-G 的使用 3， 在Linux中构建cmake 的工程 第一个问题： cmake 介绍，下载和安装以及使用：https://fukun.org/archives/0421949.html cmake是kitware公司以及一些开源开发者在开发

cMAKE

TQW4321的专栏

883

cmake 简介tt posted @ 2007年10月12日 19:38 in cmake , 3393 阅读 CMake 编译gtk文件 CMake是一个跨平台.的安装(编译)工具,可以用简单的语句来描述所有平台的安装(编译过程)。他能够输出各种各样的makefile或者project文件,能测试编译器所支持的C++特性,类似UNIX下的automake。CMake 使

CMake 入门实战(精)

起风了

9472

http://www.hahack.com/codes/cmake/ 从实例入手，讲解 CMake 的常见用法。 Contents 什么是 CMake入门案例... ：单个源文件多个源文件自定义编译选项安装和测试支持 gdb添加环境检查添加版本号生成安装包将其他平台的项目迁移到 CMake相关链接类似工具 什么是 CMake All probl

CMake入门指南

weixin_34302561的博客

3053

CMake是一个比make更高级的编译配置工具，它可以根据不同平台、不同的编译器，生成相应的Makefile或者vcp... oj项目。通过编写CMakeLists.txt，可以控制生成的Makefile，从而控制编译过程。CMake自动生成的Makefile不仅可以通过make命令构建项目生成目标文件，还支持安装（make install）、测试安装的程序是否能正确执行（make test，或者cte...

linux cmake 教程,CMake简明教程

weixin_42360733的博客

7

阅读量：121为什么使用CMake实现C/C++编译的跨平台一致性无CMake时添加一个新源文件，需要在各个平台的... 工程文件里分别修改有CMake时添加一个新源文件，只需修改CMakeLists即可基本语法特性基本语法格式：指令(参数1 参数2)参数使用括弧括起参数之间使用空格或分号分开指令是大小写无关的add_executable(hello main.cpp hello.cpp)ADD_EXEC...

Windows下CMake安装教程

u011231598的博客

11万+

环境：Windows 64位首先下载CMake官网下载地址：https://cmake.org/download/里面好多版本，根据自己需要版... 本进行下载。【注意】选择好自己电脑是什么系统，以及是32位还是64位。（P.S.x86指的是32位系统；x64指的是64位系统）下载完成后，双击进行安装。1.欢迎界面。点【next】2.同意协议。勾选，然后点【next】3.按图中红框勾选，然后点【next】...

cmake使用教程

gs

5689

一、基础知识 1、编译器 简单讲，编译器就是将“一种语言（通常为高级语言）”翻译为“另一种语言（通常为低级语... 言）”的程序。一个现代编译器的主要工作流程：源代码 (source code) → 预处理器 (preprocessor) → 编译器 (compiler) → 目标代码 (object code) → 链接器 (Linker) → 可执行程序 (executables) 高级计算机语言便于...

CMake入门教程

迪迦・奥特曼

2万+

简介 cmake的亮点在于编译复杂项目上的应用 —— cmake是一个跨平台的Makefile 生成工具! 一言以蔽之——cm... ke 为项目自动生成Makefile, 虽然cmake功能远不止步于此，但是本文聚焦于此。 例1:Hello World 源代码只有一个文件HelloWorld.cpp #include<iostream> int main(int argc, char *...

CMake和Make之间的区别

Forgive Me

4万+

本文翻译的是一篇英文文档，主要讲述的是CMake和Make之间的区别。下文中首先列出文章的中文翻译，然后紧接着的是英文原文。 下面是中文翻译部分： 编程人员已经使用CMake和Make很长一段时间了。当你加入一家大公司或者开始在一个具有大量代码的工程上开展工作时，你需要注意所有的构建。你需要看到处跳转的“CMakeLists.txt”文件。你应该会在终端使用”cmake”和”make”。很多人都

CMake介绍与使用

gua_MASS的博客

2万+

一、写本文的目的 CMake有很多复杂且与程序结构密切相关的功能，诸如跨平台编译，生成安装包，输出标准构建文档等等，如果同学们有兴趣的话可以自行学习和了解，能够很好的加深对程序架构的理解与认识。本文面向的对象是只有《c语言程序设计》这一门先修课作为基础的，参与年度计划的同学们，考虑到同学们的接受能力以及笔者自身水平有限，故只介绍cmake-gui的部分用法。以帮助大家完成年度计划任务。 二、C...

CMakeLists写法总结

同学醒醒放学了

8402

个人最近学习了一些关于常见的CMakeLists的一些写法格式，分享给大家。 CMAKE_MINIMUM_REQUIRED(VER... SION xxx) 该项表示要求CMAKE的最低版本号. PROJECT(aim1) 此项表示所建立的工程名称 aim1； FIND_PACKAGE（xxx）该命令首先会在模块路径中寻找Find<name>.cmake，这是查...

Linux下CMake使用介绍

网络资源是无限的

2万+

Linux下CMake使用介绍！

CMAKE学习经典书籍《Mastering CMake》

01-15

CMAKE学习经典书籍《Mastering CMake》，网站上另外的一个资源的发布者太不厚道了，要10分。从别的网站上... 下载了，分享给大家。

Linux下cmake的编译方式

嵌入式

7829

cmake是一个比make更高级的编译配置工具，它可以根据不同平台、不同的编译器，生成相应的Makefile或者vcp... j项目。 通过编写CMakeLists.txt，可以控制生成的Makefile，从而控制编译过程。cmake自动生成的Makefile不仅可以通过make命令构建项目生成目标文件，还支持安装（make install）、测试安装的程序是否能正确执行（make tes...

python截取字符串后三位_python如何截取字符串后几位

weixin_39747615的博客

2500

字符串切片也就是截取字符串，取子串。Python中字符串切片方法字符串[开始索引：结束索引：步长]切取字符串为... 开始索引到结束索引-1内的字符串步长不指定时步长为1，字符串[开始索引：结束索引]下面是基于python2+版本；如果是python3+版本，print输出的内容要加括号。str = '0123456789'print str[0:3] #截取第一位到第三位的字符print str[:] ...

cmake 学习笔记(一)

1+1=10

15万+

最大的Qt4程序群(KDE4)采用cmake作为构建系统Qt4的python绑定(pyside)采用了cmake作为构建系统开源的图像... 处理库 opencv 采用cmake 作为构建系统... 看来不学习一下cmake是不行了，一点一点来吧，找个最简单的C程序，慢慢复杂化，试试看