[CMake入门教程 （一）](http://www.cnblogs.com/adzuki/archive/2013/05/26/CMakeTutor_1.html)

本文为CMake官方教程翻译文档。

英文原文地址：<http://www.cmake.org/cmake/help/cmake_tutorial.html>

对应源码下载地址：[http://public.kitware.com/cgi-bin/viewcvs.cgi/Tests/Tutorial.tar.gz?root=CMake&view=tar](http://public.kitware.com/cgi-bin/viewcvs.cgi/Tests/Tutorial.tar.gz?root=CMake&view=tar" \o "http://public.kitware.com/cgi-bin/viewcvs.cgi/Tests/Tutorial.tar.gz?root=CMake&view=tar" \t "_blank)

第一步：一个基础入门例子

大多数的源码都会编译为可执行文件，对于简单的工程只需要两行的CMakeLists.txt文件就行了，我们从这里开始我们的CMake之旅。

CMakeLists.txt文件如下：

cmake\_minimum\_required (VERSION 2.6)

project (Tutorial)

add\_executable(Tutorial tutorial.cxx)

注意到这个例子里面全部命令使用小写，CMake支持大写、小写或者混合大小写命令。tutorial.cxx文件中的源码会计算一个数的平方根，第一个版本的会非常简单，代码如下：

[复制代码](javascript:void(0);)

// A simple program that computes the square root of a number

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int main (int argc, char \*argv[])

{

if (argc < 2)

{

fprintf(stdout,"Usage: %s number\n",argv[0]);

return 1;

}

double inputValue = atof(argv[1]);

double outputValue = sqrt(inputValue);

fprintf(stdout,"The square root of %g is %g\n",

inputValue, outputValue);

return 0;

}

[复制代码](javascript:void(0);)

添加一个版本号和配置头文件

我们要添加的第一个功能就是为我们的可运行程序和项目提供一个版本号。你可以在源码中专门做这件事，CMakeLists文件提供更灵活的方式来实现。添加了版本号的CMakeLists文件如下：

[复制代码](javascript:void(0);)

cmake\_minimum\_required (VERSION 2.6)

project (Tutorial)

# The version number.

set (Tutorial\_VERSION\_MAJOR 1)

set (Tutorial\_VERSION\_MINOR 0)

# configure a header file to pass some of the CMake settings

# to the source code

configure\_file (

"${PROJECT\_SOURCE\_DIR}/TutorialConfig.h.in"

"${PROJECT\_BINARY\_DIR}/TutorialConfig.h"

)

# add the binary tree to the search path for include files

# so that we will find TutorialConfig.h

include\_directories("${PROJECT\_BINARY\_DIR}")

# add the executable

add\_executable(Tutorial tutorial.cxx)

[复制代码](javascript:void(0);)

在我们的源码树种编写配置文件之后，我们还需要把路径添加到包含文件的搜索路径中。在我们的源码树中创建一个TutorialConfig.h.in文件，内容如下：

// the configured options and settings for Tutorial

#define Tutorial\_VERSION\_MAJOR @Tutorial\_VERSION\_MAJOR@

#define Tutorial\_VERSION\_MINOR @Tutorial\_VERSION\_MINOR@

当CMake配置这个头文件的时候，使用CMakeLists文件中的变量替换@Tutorial\_VERSION\_MAJOR@ 和 @Tutorial\_VERSION\_MINOR@ 。接下来修改tutorial.cxx文件包含配置好的头文件来使用版本号。修改后的源码如下：

[复制代码](javascript:void(0);)

// A simple program that computes the square root of a number

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "TutorialConfig.h"

int main (int argc, char \*argv[])

{

if (argc < 2)

{

fprintf(stdout,"%s Version %d.%d\n",

argv[0],

Tutorial\_VERSION\_MAJOR,

Tutorial\_VERSION\_MINOR);

fprintf(stdout,"Usage: %s number\n",argv[0]);

return 1;

}

double inputValue = atof(argv[1]);

double outputValue = sqrt(inputValue);

fprintf(stdout,"The square root of %g is %g\n",

inputValue, outputValue);

return 0;

}

[复制代码](javascript:void(0);)

主要的变化是包含了TutorialConfig.h头文件，在Usage信息中输出版本号。

至此一个简单的项目就创建完成了，我们再源码中创建build目录，使用命令行进入build目录。

运行：

cmake ..

make

即可生成项目，通过为CMake添加-G参数来生成不同平台的makefile文件

运行Tutorial会提示带版本号的Usage信息，输入Tutorial 25 可以计算25的平方根为5.

# [cmake使用一 —— cmake语法](http://blog.csdn.net/kaizi318/article/details/7721660)

分类： [CMake](http://blog.csdn.net/kaizi318/article/category/1177838)2012-07-06 11:49 845人阅读 [评论](http://blog.csdn.net/kaizi318/article/details/7721660#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/kaizi318/article/details/7721660#report)

[正则表达式](http://www.csdn.net/tag/%e6%ad%a3%e5%88%99%e8%a1%a8%e8%be%be%e5%bc%8f)[command](http://www.csdn.net/tag/command)[string](http://www.csdn.net/tag/string)[function](http://www.csdn.net/tag/function)[makefile](http://www.csdn.net/tag/makefile)[path](http://www.csdn.net/tag/path)

Cmake是一个用于生成交叉平台编译工程的工具。它类似autoconf。它产生你所选择的编译环境的本地makefile和workspaces。

英文原文地址：<http://www.cmake.org/cmake/help/syntax.html>

Cmake的输入是在源码目录下的CMakeLists.txt文件。这个文件可以用include 或者 add\_subdirectory 命令增加入其它的输入文件。

**语法**

CMakeList.txt文件是由注释、命令和空白字符组成。

注释是由 # 开始，到行结尾。

命令是由：命令名、（、空格分隔的参数、）组成。

例如：command (args….)

上面的command可以是一个命令名；或者是一个宏；也可以是一个函数名。

args是以空格分隔的参数例表（如果参数中包含空格，则要加双引号）

除了用于分隔参数的空白字符（空格、换行号、tabs）都是被忽略不计的。任何包含在双引号中的字符都做为一个参数。一个反斜杠用于转换码。

命令名是大小写不敏感的。

**字符串(string)和字符串列表（lists）**

CMake的基本数据类型是字符串（string）。CMake也支持由字符串组成的字符串列表。字符串列表可以由；或空格分隔的组成。例如：下面设置变量var是等效的。

set(var a;b;c)

set(var a b c)

字符串列表可以用 foreach 命令叠代（iterated）或list命令操作。

**变量**

CMake支持简单的变量：字符串或字符串列表。用 ${VAR} 语法得到变量的引用。

可以用一个set命令把一个字符串列表设置为一个变量，然后把这个变量传递给需要传递多参数的函数。例如：

set(Foo a b c)

command(${Foo})

上面两行等效 command(a b c)

如果你想传把一个字符串列表做为一个单独的参数传递给函数，用双引号包含它。例如：

Command(“${Foo}”)

等效于：command(“a b c”)

环境变量：

用$ENV{VAR}得到环境变量的引用

设置环境变量：

Set(ENV{VAR} /home)

**程序流控制**

CMake提供三种程序流控制结构：

1.         条件声明：if

# some\_command will be called if the variable's value is not:   
# empty, 0, N, NO, OFF, FALSE, NOTFOUND, or -NOTFOUND.   
if(var)   
   some\_command(...)   
endif(var)

2.         循环控制：foreach 和 while

set(VAR a b c)   
  # loop over a, b,c with the variable f   
foreach(f ${VAR})   
    message(${f})   
endforeach(f)

3.         程序定义：宏或函数（函数在CMake2.6以后的版本才支持）。函数建立本地范围内的变量，宏用于全局范围内。

# define a macro hello   
macro(hello MESSAGE)  
    message(${MESSAGE})  
endmacro(hello)   
# call the macro with the string "hello world"   
hello("hello world")   
# define a function hello   
function(hello MESSAGE)  
    message(${MESSAGE})   
endfunction(hello)

译者注：

函数可以返回，可以用 return() 命令返回。如果要从函数中返回值，只能通过参数返回：

#定义函数 get\_lib 从给定的目录查找指定的库，并把它传回到参数 lib\_FILE 中

function(get\_lib lib\_FILE lib\_NAME lib\_PATH)

       #message("lib\_name:""${lib\_NAME}")

       set(\_\_LIB "\_\_LIB-NOTFOUND")

       #message("\_\_LIB:""${\_\_LIB}")

       find\_library(\_\_LIB ${lib\_NAME} ${lib\_PATH})

       if(\_\_LIB STREQUAL "\_\_LIB-NOTFOUND")

              message("don't find ${lib\_NAME} librarys in ${lib\_PATH}")

              return()

       endif()

       #message("\_\_LIB:""${\_\_LIB}")

       set(${lib\_FILE} ${\_\_LIB} PARENT\_SCOPE)

endfunction(get\_lib)

set命令中 PARENT\_SCOPE表示传递给函数调用者所拥有的变量

**引号、字符串和escapes**

一个用双引号包含的字符串，是这个字符串的字面含义。一个字符串能包含多行。例如：

set (MY\_STRING "this is a string with a   
  newline in  
  it")

也能在字符串中用转义字符

set (VAR "  
   hello   
  world   
  ")  
message ( "\${VAR} = ${VAR}")  
  # prints out  
  ${VAR} =   
    hello   
    world

标准C的转义字符被支持

message("\n\thello world")  
# prints out   
hello world

message(hell"o")   -> prints hell"o"   
message(hell"o")   -> prints hell"o"   
message(hell\"o\")  -> prints hell"o"

引号必须是匹配的

message(hell"o)   -> produces this error:   
Parse error. Function missing ending ")".   
Instead found unterminated string with text "o)   
".   
  
    message(hell\"o) -> prints hell"o

**正则表达式**

一些CMake命令（如if 和 string），能使用正则表达式或使用正则表达式作为参数。一个简单的形式，一个正则表达式用于在一个序列的字符串中精确查找一个字符。然而在大多时候，一个精确查找是不知道的或者只是匹配最前或者最后字符。所以这里用正则表达式进行不同的转换。Cmake标准是可被描述的。这个描述是基于开源的正则表达式类（Texas Instruments）。

* + ^ 匹配一行或一字符串开头
  + $匹配一行或一字符串结尾
  + .匹配单一字符或一个新行
  + [ ]匹配括号中的任一字符
  + [^ ] 匹配不在括号内的任一字符
  + [-] 匹配指定范围内的字符
  + \* 匹配0次或多次
  + + 匹配一次或多次
  + ? 匹配0次或一次
  + ()保存匹配的表达式并用随后的替换它