原文地址：<https://blog.csdn.net/qingzhuyuxian/article/details/86607337>

**基本语法**

**注释的语法**

# 字符开始到此行结束。

**CMake命令**

命令可以是一个内置命令（例如：project，add\_executable等），也可以是一个用户定义的宏（macro）或者函数（function）

**数据类型**

CMake的基本数据类型是字符串，一组字符串在一起称为列表（list），例如：

# 通过 set 命令构建一个 listVAR

set(VAR a b c)

**变量**

使用 set() 定义变量

使用语法 ${VariableName} 来访问变量

set(VAR a b c)

# 输出 VAR = a;b;c

message("VAR = ${VAR}")

使用语法 $ENV{VariableName} 来访问环境变量的值

**If语句**

if(expression)

#...

elseif(expression2)

#...

else()

#...

endif()

对于 if(string) 来说：

如果 string 为（不区分大小写）1、ON、YES、TRUE、Y、非 0 的数则表示真

如果 string 为（不区分大小写）0、OFF、NO、FALSE、N、IGNORE、空字符串、以 -NOTFOUND 结尾的字符串则表示假

示例：

cmake\_minimum\_required(VERSION 2.8)

set(YES 0)

# 输出 True

if(YES)

message(True)

else()

message(False)

endif()

# 输出 False

if(${YES})

message(True)

else()

message(False)

endif()

表达式中可以包含操作符，操作符包括：

一元操作符，例如：EXISTS、COMMAND、DEFINED 等

二元操作符，例如：EQUAL、LESS、GREATER、STRLESS、STRGREATER 等

NOT（非操作符）

AND（与操作符）、OR（或操作符）

操作符优先级：一元操作符 > 二元操作符 > NOT > AND、OR

常用操作符示例：

if(NOT expression)

if(expr1 AND expr2)

if(expr1 OR expr2)

if(COMMAND command-name)

为真的前提是存在 command-name 命令、宏或函数且能够被调用

if(EXISTS name)

为真的前提是存在 name 的文件或者目录（应该使用绝对路径）

if(file1 IS\_NEWER\_THAN file2)

为真的前提是 file1 比 file2 新或者 file1、file2 中有一个文件不存在

if(IS\_DIRECTORY directory-name)

为真的前提是 directory-name 表示的是一个目录（应该使用绝对路径）

if(variable|string MATCHES regex)

为真的前提是变量值或者字符串匹配 regex 正则表达式

if(variable|string LESS variable|string)

if(variable|string GREATER variable|string)

if(variable|string EQUAL variable|string)

为真的前提是变量值或者字符串为有效的数字且满足小于（大于、等于）的条件

if(variable|string STRLESS variable|string)

if(variable|string STRGREATERvariable|string)

if(variable|string STREQUALvariable|string)

为真的前提是变量值或者字符串以字典序满足小于（大于、等于）的条件

if(DEFINED variable)

为真的前提是 variable 表示的变量被定义了。

**foreach 循环**

set(VAR a b c)

foreach(f ${VAR})

message(${f})

endforeach()

**while 循环**

set(VAR 5)

while(${VAR} GREATER 0)

message(${VAR})

math(EXPR VAR "${VAR} - 1")

endwhile()

**函数和宏定义**

函数使用set变量为局部作用域，函数外无法使用

宏则使用set变量为全局作用域，宏外可以使用

范例：

# 定义一个宏 hello

set(MESSAGE )

macro(hello MESSAGE)

set(MESSAGE aaa)

endmacro()

message(${MESSAGE })

# 定义一个函数 hello

function(hello MESSAGE)

set(MESSAGE aaa) # 错误，函数不能使用全局作用域

endfunction()

函数和宏可以通过命令 return() 返回，但是函数和宏的返回值必须通过参数传递出去。例如：

cmake\_minimum\_required(VERSION 2.8)

function(get\_func RESULT)

#RESULT 的值为实参的值，因此需要使用 ${RESULT}

#这里使用 PARENT\_SCOPE 是因为函数会构建一个局部作用域

set(${RESULT} "Hello Function" PARENT\_SCOPE)

endfunction()

**字符串的一些问题**

字符串可跨行且支持转移字符，例如：

set(VAR "hello

world")

# 输出结果为：

# ${VAR} = hello

# world

message("\${VAR} = ${VAR}")

**CMake命令**

**project 命令**

命令语法：project(<projectname> [languageName1 languageName2 … ] )

命令简述：用于指定项目的名称

使用范例：project(Main)

**cmake\_minimum\_required命令**

命令语法：cmake\_minimum\_required(VERSION major[.minor[.patch[.tweak]]][FATAL\_ERROR])

命令简述：用于指定需要的 CMake 的最低版本

使用范例：cmake\_minimum\_required(VERSION 2.8)

**aux\_source\_directory命令**

命令语法：aux\_source\_directory(<dir> <variable>)

命令简述：用于将 dir 目录下的所有源文件的名字保存在变量 variable 中

使用范例：aux\_source\_directory(. DIR\_SRCS)

**add\_executable 命令**

命令语法：add\_executable(<name> [WIN32] [MACOSX\_BUNDLE][EXCLUDE\_FROM\_ALL] source1 source2 … sourceN)

命令简述：用于指定从一组源文件 source1 source2 … sourceN 编译出一个可执行文件且命名为 name

使用范例：add\_executable(Main ${DIR\_SRCS})

**add\_library 命令**

命令语法：add\_library([STATIC | SHARED | MODULE] [EXCLUDE\_FROM\_ALL] source1source2 … sourceN)

命令简述：用于指定从一组源文件 source1 source2 … sourceN 编译出一个库文件且命名为 name

使用范例：add\_library(Lib ${DIR\_SRCS})

**add\_dependencies 命令**

命令语法：add\_dependencies(target-name depend-target1 depend-target2 …)

命令简述：用于指定某个目标（可执行文件或者库文件）依赖于其他的目标。这里的目标必须是 add\_executable、add\_library、add\_custom\_target 命令创建的目标

**add\_subdirectory 命令**

命令语法：add\_subdirectory(source\_dir [binary\_dir] [EXCLUDE\_FROM\_ALL])

命令简述：用于添加一个需要进行构建的子目录（其目录下有CMakeLists.txt）

使用范例：add\_subdirectory(Lib)

**target\_link\_libraries命令**

命令语法：target\_link\_libraries(<target> [item1 [item2 […]]][[debug|optimized|general] ] …)

命令简述：用于指定 target 需要链接 item1 item2 …。这里 target 必须已经被创建，链接的 item 可以是已经存在的 target（依赖关系会自动添加）

使用范例：target\_link\_libraries(Main Lib)

**set 命令**

命令语法：set(<variable> <value> [[CACHE <type><docstring> [FORCE]] | PARENT\_SCOPE])

命令简述：用于设定变量 variable 的值为 value。如果指定了 CACHE 变量将被放入 Cache（缓存）中。

使用范例：set(ProjectName Main)

**unset 命令**

命令语法：unset(<variable> [CACHE])

命令简述：用于移除变量 variable。如果指定了 CACHE 变量将被从 Cache 中移除。

使用范例：unset(VAR CACHE)

**message 命令**

命令语法：message([STATUS|WARNING|AUTHOR\_WARNING|FATAL\_ERROR|SEND\_ERROR] “message todisplay”…)

命令简述：用于输出信息

使用范例：message(“Hello World”)

**include\_directories 命令**

命令语法：include\_directories([AFTER|BEFORE] [SYSTEM] dir1 dir2 …)

命令简述：用于设定目录，这些设定的目录将被编译器用来查找 include 文件

使用范例：include\_directories(${PROJECT\_SOURCE\_DIR}/lib)

**find\_path 命令**

命令语法：find\_path(<VAR> name1 [path1 path2 …])

命令简述：用于查找包含文件 name1 的路径，如果找到则将路径保存在 VAR 中（此路径为一个绝对路径），如果没有找到则结果为 <VAR>-NOTFOUND。默认的情况下，VAR 会被保存在 Cache 中，这时候我们需要清除 VAR 才可以进行下一次查询（使用 unset 命令）。

使用范例：

find\_path(LUA\_INCLUDE\_PATH lua.h${LUA\_INCLUDE\_FIND\_PATH})

if(NOT LUA\_INCLUDE\_PATH)

message(SEND\_ERROR "Header file lua.h not found")

endif()

**find\_library 命令**

命令语法：find\_library(<VAR> name1 [path1 path2 …])

命令简述：用于查找库文件 name1 的路径，如果找到则将路径保存在 VAR 中（此路径为一个绝对路径），如果没有找到则结果为 <VAR>-NOTFOUND。一个类似的命令 link\_directories 已经不太建议使用了

**add\_definitions 命令**

命令语法：add\_definitions(-DFOO -DBAR …)

命令简述：用于添加编译器命令行标志（选项），通常的情况下我们使用其来添加预处理器定义

使用范例：add\_definitions(-D\_UNICODE -DUNICODE)

**execute\_process 命令**

命令语法：

execute\_process(COMMAND <cmd1>[args1...]]

[COMMAND <cmd2>[args2...] [...]]

[WORKING\_DIRECTORY<directory>]

[TIMEOUT <seconds>]

[RESULT\_VARIABLE<variable>]

[OUTPUT\_VARIABLE<variable>]

[ERROR\_VARIABLE<variable>]

[INPUT\_FILE <file>]

[OUTPUT\_FILE <file>]

[ERROR\_FILE <file>]

[OUTPUT\_QUIET]

[ERROR\_QUIET]

[OUTPUT\_STRIP\_TRAILING\_WHITESPACE]

[ERROR\_STRIP\_TRAILING\_WHITESPACE])

命令简述：用于执行一个或者多个外部命令。

使用范例：execute\_process(COMMAND ls)

**file 命令**

命令简述：此命令提供了丰富的文件和目录的相关操作（这里仅说一下比较常用的）

使用范例：

# 目录的遍历

# GLOB 用于产生一个文件（目录）路径列表并保存在variable 中

# 文件路径列表中的每个文件的文件名都能匹配globbing expressions（非正则表达式，但是类似）

# 如果指定了 RELATIVE 路径，那么返回的文件路径列表中的路径为相对于 RELATIVE 的路径

# file(GLOB variable [RELATIVE path][globbing expressions]...)

# 获取当前目录下的所有的文件（目录）的路径并保存到 ALL\_FILE\_PATH 变量中

file(GLOB ALL\_FILE\_PATH ./\*)

# 获取当前目录下的 .h 文件的文件名并保存到ALL\_H\_FILE 变量中

# 这里的变量CMAKE\_CURRENT\_LIST\_DIR 表示正在处理的 CMakeLists.txt 文件的所在的目录的绝对路径（2.8.3 以及以后版本才支持）

file(GLOB ALL\_H\_FILE RELATIVE${CMAKE\_CURRENT\_LIST\_DIR} ${CMAKE\_CURRENT\_LIST\_DIR}/\*.h)

**string命令**

替换：

string(REPLACE <match\_string> <replace\_string> <output variable> <input> [<input>...])

正则匹配，结果放入<output variable>中：

string(REGEX MATCH <regular\_expression> <output variable> <input> [<input>...])

取子串：

string(SUBSTRING <string> <begin> <length> <output variable>)

示例：

string(REPLACE ${PROJECT\_SOURCE\_DIR}/ "" newf ${f})

**add\_definitions**

定义一个宏，示例：

add\_definitions(-D \_SysEncodeUTF8)

这里定义了宏\_SysEncodeUTF8，我们可以在源码文件中使用这个宏（如：#ifdef \_SysEncodeUTF8）