

CMake 入门笔记

王逸凡

计 62

2017.3.21

什么是 Make

什么是 Make

- Make 是一种工具，它能帮助你管理你的源文件以及一些和其它的非源文件。

什么是 Make

- Make 是一种工具，它能帮助你管理你的源文件以及一些和其它的非源文件。
- Make 的核心在于 Makefile 文件。通过 Makefile 文件，你可以简便而有效地按照你的想法来决定源文件之间的关系，并且驱动编译器为你进行编译。

什么是 Make

- Make 是一种工具，它能帮助你管理你的源文件以及一些和其它的非源文件。
- Make 的核心在于 Makefile 文件。通过 Makefile 文件，你可以简便而有效地按照你的想法来决定源文件之间的关系，并且驱动编译器为你进行编译。
- 因此，我们可以认为 Make 是在源文件上一层的管理工具。

什么是 CMake

什么是 CMake

- CMake 的全称是 Cross-Platform Make。它可以被认为是在 Make 的上一级的管理工具，因为它的一个直接的作用就是帮我们生成 Makefile 文件。

什么是 CMake

- CMake 的全称是 Cross-Platform Make。它可以被认为是在 Make 的上级别的管理工具，因为它的一个直接的作用就是帮我们生成 Makefile 文件。
- 和用 Makefile 文件来使用 Make 类似，我们需要通过编写一个名为 CMakeLists.txt 的文件来使用 CMake。

为什么要使用 CMake

为什么要使用 CMake

- CMake 最核心的优势就体现在它的名字中——Cross Platform。

为什么要使用 CMake

- CMake 最核心的优势就体现在它的名字中——Cross Platform。
- 如果我们需要在不同的系统下使用不同的编译器平台来运行我们的代码，就会产生很多麻烦的地方。

为什么要使用 CMake

- CMake 最核心的优势就体现在它的名字中——Cross Platform。
- 如果我们需要在不同的系统下使用不同的编译器平台来运行我们的代码，就会产生很多麻烦的地方。
- CMake 可以帮我们有效地解决这个问题。它能根据我们的需求来帮我们生成适用于不同平台的文件。

为什么要使用 CMake

- CMake 最核心的优势就体现在它的名字中——Cross Platform。
- 如果我们需要在不同的系统下使用不同的编译器平台来运行我们的代码，就会产生很多麻烦的地方。
- CMake 可以帮我们有效地解决这个问题。它能根据我们的需求来帮我们生成适用于不同平台的文件。
- 但就算我们只使用 `make`，不涉及其它平台，CMake 也是一个很好的简化 `makefile` 文件的工具。

安装与配置

安装与配置

- 首先我们需要从 <https://www.cmake.org/> 上下载对应系统的 CMake，并依照不同的指示进行安装。

安装与配置

- 首先我们需要从 <https://www.cmake.org/> 上下载对应系统的 CMake，并依照不同的指示进行安装。
- 如果你使用的是 mac，你也可以使用 `brew install cmake` 来进行安装（不过你需要先安装 homebrew）。

安装与配置

- 首先我们需要从 <https://www.cmake.org/> 上下载对应系统的 CMake，并依照不同的指示进行安装。
- 如果你使用的是 mac，你也可以使用 `brew install cmake` 来进行安装（不过你需要先安装 homebrew）。
- 在完成安装之后，如果你发现不能找到 `cmake` 命令，那么可能是路径没有配置成功所导致的。这时，你需要手动地配置路径。比如在 mac 下，你需要在终端执行 `export PATH=/Applications/CMake.app/Contents/bin:$PATH`。

一个最简单的例子

一个最简单的例子

- 比如我们有一个文件 `main.cpp`，放在一个名叫 `home` 的目录下。

一个最简单的例子

- 比如我们有一个文件 main.cpp，放在一个名叫 home 的目录下。

```
#include<iostream>
using namespace std;
int main()
{
    cout<<"Hello World"<<endl;
}
```

一个最简单的例子

- 比如我们有一个文件 `main.cpp`，放在一个名叫 `home` 的目录下。

```
#include<iostream>
using namespace std;
int main()
{
    cout<<"Hello World"<<endl;
}
```

- 现在，我们需要制作上面这份代码的 `make` 文件。

一个最简单的例子

一个最简单的例子

- 在 home 这个目录下，我们新建一个文件 CMakeLists.txt，并写入如下内容：

一个最简单的例子

- 在 home 这个目录下，我们新建一个文件 CMakeLists.txt，并写入如下内容：

```
project ( test )  
set ( SRC  main . cpp )  
add_executable ( test  ${ SRC } )
```


一个最简单的例子

- 在 home 这个目录下，我们新建一个文件 CMakeLists.txt，并写入如下内容：

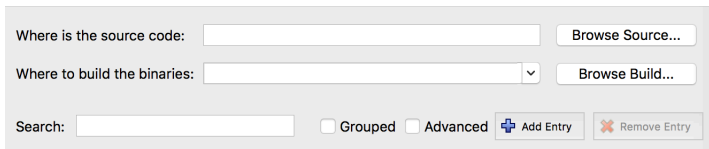
```
project ( test )  
set ( SRC main . cpp )  
add_executable ( test  ${SRC} )
```

- 接着，我们需要通过对 CMakeLists.txt 的编译来生成 makefile 文件。

一个最简单的例子

一个最简单的例子

- 一种最直接的办法是点开 CMake。



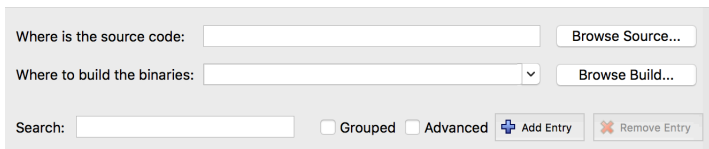
Where is the source code: [Browse Source...](#)

Where to build the binaries: [Browse Build...](#)

Search: ☐ Grouped ☐ Advanced [+ Add Entry](#) [✖ Remove Entry](#)

一个最简单的例子

- 一种最直接的办法是点开 CMake。

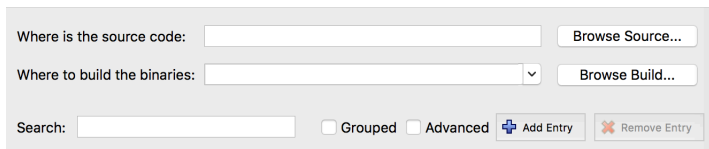


The image shows a portion of the CMake GUI. It features two text input fields: 'Where is the source code:' and 'Where to build the binaries:'. To the right of the first field is a 'Browse Source...' button. To the right of the second field is a 'Browse Build...' button. Below these fields is a 'Search:' label followed by another text input field. To the right of the search field are two checkboxes labeled 'Grouped' and 'Advanced'. Further right are two buttons: '+ Add Entry' and '- Remove Entry'.

- 找到”Where is the source code”一栏，并且把 home 文件夹的路径填入栏中。

一个最简单的例子

- 一种最直接的办法是点开 CMake。



The image shows a screenshot of the CMake GUI. It features two main input fields: 'Where is the source code:' and 'Where to build the binaries:'. Each field has a 'Browse' button next to it. Below these fields is a 'Search:' field and two checkboxes labeled 'Grouped' and 'Advanced'. At the bottom right, there are two buttons: '+ Add Entry' and '- Remove Entry'.

- 找到”Where is the source code”一栏，并且把 home 文件夹的路径填入栏中。
- ”Where to build the binaries”一栏则决定了你的文件会被生成在哪里。

一个最简单的例子

一个最简单的例子

- 当然，我们可以直接把文件生成在 home 文件夹中。不过这会使得源文件和 cmake 所生成的文件混在一起，使得文件的管理比较混乱。

一个最简单的例子

- 当然，我们可以直接把文件生成在 home 文件夹中。不过这会使得源文件和 cmake 所生成的文件混在一起，使得文件的管理比较混乱。
- 因此，我们更推荐的做法是在 home 文件夹中新建一个 build 文件夹，并且把文件都生成在那里。

一个最简单的例子

一个最简单的例子

- 除了上面的做法以外，我们还可以直接在终端下执行

一个最简单的例子

- 除了上面的做法以外，我们还可以直接在终端下执行

```
cmake .. -G "Unix Makefiles"  
make
```

一个最简单的例子

- 除了上面的做法以外，我们还可以直接在终端下执行

```
cmake .. -G "Unix Makefiles"  
make
```

- 需要注意的是，执行上述命令之前，我们需要 cd 到 home 目录下的 build 文件夹中，而不是 cd 到 home 文件夹中。

一个最简单的例子

- 除了上面的做法以外，我们还可以直接在终端下执行

```
cmake .. -G "Unix Makefiles"  
make
```

- 需要注意的是，执行上述命令之前，我们需要 cd 到 home 目录下的 build 文件夹中，而不是 cd 到 home 文件夹中。
- 实际上，在我所看到的一些 CMake 入门材料中，上述命令的第一行中的“Unix”实际上写为“MinGW”。但如果按 MinGW 来写，在执行命令时会出现奇怪的错误而无法通过。如果有同学了解其中的原因，或是和我一样产生了相同的问题，欢迎课下与我交流。

多个文件下 Cmake 的使用

多个文件下 Cmake 的使用

- 以第一次作业中的 Problem1 为例：在 home 文件夹下，我们分别有 main.cpp, polinomial.cpp, posynomial.cpp 以及 func.h 四个文件。

多个文件下 Cmake 的使用

- 以第一次作业中的 Problem1 为例：在 home 文件夹下，我们分别有 main.cpp, polinomial.cpp, posynomial.cpp 以及 func.h 四个文件。
- 实际上我们只需要对上面例子中的 CMakeLists.txt 文件进行一个小的修改，就能生成正确的 make 文件。修改后的代码如下：

多个文件下 Cmake 的使用

- 以第一次作业中的 Problem1 为例：在 home 文件夹下，我们分别有 main.cpp, polynomial.cpp, posynomial.cpp 以及 func.h 四个文件。
- 实际上我们只需要对上面例子中的 CMakeLists.txt 文件进行一个小的修改，就能生成正确的 make 文件。修改后的代码如下：

```
project(test)
set(SRC main.cpp polynomial.cpp posynomial.cpp)
add_executable(test ${SRC})
```

CMake 的一些其它应用

CMake 的一些其它应用

- 前面所介绍的一些 CMake 的使用方法只是 CMake 的强大功能中的一小点。除了能够非常方便地实现源文件之间的连接以外，它还能支持很多其它的功能。

CMake 的一些其它应用

- 前面所介绍的一些 CMake 的使用方法只是 CMake 的强大功能中的一小点。除了能够非常方便地实现源文件之间的连接以外，它还能支持很多其它的功能。
- 比如，CMake 可以支持生成库文件，还能支持把 `main.cpp` 文件及其它的文件放在不同的文件夹中。

CMake 的一些其它应用

- 前面所介绍的一些 CMake 的使用方法只是 CMake 的强大功能中的一小点。除了能够非常方便地实现源文件之间的连接以外，它还能支持很多其它的功能。
- 比如，CMake 可以支持生成库文件，还能支持把 `main.cpp` 文件及其它的文件放在不同的文件夹中。
- 由于上述操作实现起来比较繁杂，而且涉及到了有关于库文件的构建，所以在此不再展开。有兴趣的同学可以在参考资料中找到详细的讲解。

参考资料

- <http://www.liuxiao.org/2015/01/mac-cmake-解决-command-not-found-问题/>
- <http://blog.csdn.net/dbzhang800/article/details/6314073>
- <https://www.cmake.org/>

Thanks for listening!

谢谢！