Usando cmake para compilar c/c++

Israel Junior Follow
Sep 18, 2017 · 3 min read

Como usar o cmake para compilar projetos em c/c++

CMake — Cross-Platform Makefile Generator

O que é o cmake

O CMake é um "Cross-Platform Makefile Generator" ou uma ferramenta open-source que permite gerar automaticamente scripts de construção de aplicação em diferentes plataformas, como por exemplo "Unix Makefiles, Xcode, Visual Studio, CodeBlocks etc …", uma lista completa pode ser encontrada aqui.

Instalando o cmake

Para instalar o cmake abra o terminal e siga as instruções abaixo.

```
# For Ubuntu
$ sudo apt-get install cmake

# For Redhat
$ yum install cmake
```

```
# For Mac OS X with brew
$ brew install cmake
```

Criando um hello world em cpp

Para facilitar nosso exemplo vamos criar um pequeno hello world em cpp e gerar o build com o cmake.

Criando um diretório para o projeto.

- $ mkdir sampleCmake
- $ cd sampleCmake
- $ vi main.cpp

Dentro do arquivo **main.cpp** coloque o conteúdo abaixo.

```
#include <iostream>

using namespace std;

int main(void) {

     cout << "Hello World" << endl;

     return(0);

}
```

Agora temos que criar o nosso CMakeLists.txt que será o roteiro usado pelo cmake para criar os scripts de compilação da aplicação. dentro do CMakeLists.txt coloque o conteúdo listado abaixo.

```
cmake_minimum_required(VERSION 2.8)
project(sampleCmake)
add_executable(hello main.cpp)
```

Uma vez criado os arquivos do nosso projeto vamos testar para saber se está tudo funcionando como planejado para isso siga as instruções abaixo.

```
$ mkdir build && cd build
$ cmake ../
$ make
$ ./hello
```

Nos criamos um diretório aparte para compilar o projeto convencionalmente chamado de build e depois executamos o cmake ../ mas informações para que ele procure o CMakeLists.txt no diretório anterior. Uma vez que encontre o CMakeLists.txt o cmake começa a construção dos scripts de compilação necessários no caso do exemplo acima um Makefile. Terminado a construção sem erros temos um arquivo Makefile pronto para compilação nesse caso é só executar um "make" para compilar e depois executar o binário "./hello".

Melhorando um pouco nosso projeto.

Agora digamos que eu queira controlar a versão do meu binário, com o cmake também podemos fazer essa tarefa de forma simples segue abaixo algumas alterações que teremos que fazer no nosso projeto.

Altere o arquivo CMakeLists.txt conforme abaixo :

```
cmake_minimum_required(VERSION 2.8)
project(sampleCmake)

set (Sample_VERSION_MAJOR 0)
set (Sample_VERSION_MINOR 0)
set (Sample_VERSION_PATCH 1)

configure_file(
  "${PROJECT_SOURCE_DIR}/SampleConfig.h.in"
  "${PROJECT_BINARY_DIR}/SampleConfig.h"
  )

include_directories("${PROJECT_BINARY_DIR}")

add_executable(hello main.cpp)
```

Crie o arquivo chamado "SampleConfig.h.in" com o conteúdo listado abaixo.

```
#define SAMPLE_VERSION_MAJOR @Sample_VERSION_MAJOR@
#define SAMPLE_VERSION_MINOR @Sample_VERSION_MINOR@
```

```
#define SAMPLE_VERSION_PATCH @Sample_VERSION_PATCH@
```

Agora altere o arquivo main.cpp para ficar como o conteúdo listado abaixo.

```
#include <iostream>
#include "SampleConfig.h"

void version()
{
  std::cout << "Version : " << SAMPLE_VERSION_MAJOR <<
    "." << SAMPLE_VERSION_MINOR <<
    "." << SAMPLE_VERSION_PATCH << std::endl;
}

int
main(void)
{
  version();
  std::cout << "Hello World by CMake !!!" << std::endl;
  return(0);
}
```

Uma vez feitas estas alterações compile novamente o projeto usando o cmake conforme descrito anteriormente.

```
$ rm -rf build
$ mkdir build && cd build
$ cmake ../
$ make
$ ./hello
Version : 0.0.1
Hello World by CMake !!!
```

O resultado deve ser uma saída como mostrado acima. Com o cmake podemos fazer muito mais no próximo artigo vou mostrar como usar o cmake para compilar um projeto usando biblioteca de terceiros como a libcurl.

Para um melhor entendimento estou disponibilizando uma versão desse projeto no

C C Programming Cpp Cpp11 Cmake