CMake

Sommaire

Thierry Vaira - <tvaira@free.fr> - version v1.0 - 05/01/2021

1. CMake

CMake (abréviation de « *cross platform make* ») est un **système de construction logicielle multiplateforme**. Il permet de vérifier les prérequis nécessaires à la construction, de déterminer les dépendances entre les différents composants d'un projet, afin de planifier une construction ordonnée et adaptée à la plateforme. La construction du projet est ensuite déléguée à un logiciel spécialisé dans l'ordonnancement de taches et spécifique à la plateforme, Make, Ninja ou Microsoft Visual Studio.

Malgré l'utilisation de « *make* » dans son nom, **CMake** est une application séparée et de plus haut niveau que l'outil `make`.

Liens :

- https://cmake.org/ et https://cmake.org/documentation/
- https://fr.wikipedia.org/wiki/CMake
- https://cmake.org/cmake/help/latest/guide/tutorial/index.html
- https://alexandre-laurent.developpez.com/tutoriels/cmake/

2. Installation

- Téléchargements pour Windows et Mac OS X
- Sous GNU/Linux Ubuntu :

```
$ sudo apt-get install cmake

$ cmake --version
cmake version 3.10.2

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

$ cmake --help
Usage

  cmake [options] <path-to-source>
  cmake [options] <path-to-existing-build>

Specify a source directory to (re-)generate a build system for it in the
current working directory.  Specify an existing build directory to
re-generate its build system.

Options
  -C <initial-cache>           = Pre-load a script to populate the cache.
  -D <var>[:<type>]=<value>    = Create a cmake cache entry.
  -U <globbing_expr>           = Remove matching entries from CMake cache.
  -G <generator-name>          = Specify a build system generator.
  -T <toolset-name>            = Specify toolset name if supported by
                                 generator.
  -A <platform-name>           = Specify platform name if supported by
                                 generator.
...

Generators

The following generators are available on this platform:
  Unix Makefiles               = Generates standard UNIX makefiles.
  Ninja                        = Generates build.ninja files.
  Watcom WMake                 = Generates Watcom WMake makefiles.
  CodeBlocks - Ninja           = Generates CodeBlocks project files.
  CodeBlocks - Unix Makefiles  = Generates CodeBlocks project files.
  CodeLite - Ninja             = Generates CodeLite project files.
  CodeLite - Unix Makefiles    = Generates CodeLite project files.
  Sublime Text 2 - Ninja       = Generates Sublime Text 2 project files.
  Sublime Text 2 - Unix Makefiles
                               = Generates Sublime Text 2 project files.
  Kate - Ninja                 = Generates Kate project files.
  Kate - Unix Makefiles        = Generates Kate project files.
  Eclipse CDT4 - Ninja         = Generates Eclipse CDT 4.0 project files.
  Eclipse CDT4 - Unix Makefiles= Generates Eclipse CDT 4.0 project files.
  KDevelop3                    = Generates KDevelop 3 project files.
  KDevelop3 - Unix Makefiles   = Generates KDevelop 3 project files.
```

Il existe des versions GUI pour CMake :

```
$ sudo apt-get install cmake-curses-gui
$ sudo apt-get install cmake-qt-gui
```

3. Exemple n°1

```
$ mkdir exemple1
$ cd exemple1
```

main.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "hello world!\n";
    return 0;
}
```

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 3.0) # au moins 2.8
PROJECT(hello)
ADD_EXECUTABLE(hello main.cpp)
```

Pour activer la prise en charge d'une norme C++ spécifique, il faut la préciser dans la variable `CMAKE_CXX_STANDARD` et placer la variable `CMAKE_CXX_STANDARD_REQUIRED` à `True` :

```
CMAKE_MINIMUM_REQUIRED(VERSION 3.0) # au moins 2.8
PROJECT(hello)
SET(CMAKE_CXX_STANDARD 11) # pour C++11
SET(CMAKE_CXX_STANDARD_REQUIRED True)
ADD_EXECUTABLE(hello main.cpp)
```

```
$ mkdir build
$ cd build

$ cmake ..
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- 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/exemples/exemple1/build

$ make
Scanning dependencies of target hello
[ 50%] Building CXX object CMakeFiles/hello.dir/main.cpp.o
[100%] Linking CXX executable hello
[100%] Built target hello

$ ./hello
hello world!
```

Il est possible d'activer le mode verbeux (*verbose*) :

```
$ cmake --trace ..

$ make VERBOSE=1
```

4. Exemple n°2

```
$ mkdir exemple2
$ cd exemple2
```

thread.cpp

```cpp
#include <iostream>
#include <thread>
#include <chrono>

using namespace std;

void etoile()
{
    for(int i=0; i < 10; ++i)
    {
        this_thread::sleep_for(chrono::duration<int,milli>(250));
        cout << "*";
    }
}

void diese()
{
    for(int i=0; i < 10; ++i)
    {
        this_thread::sleep_for(chrono::duration<int,milli>(250));
        cout << "#";
    }
}

int main()
{
    setbuf(stdout, NULL);

    thread t1(etoile); // création et lancement du thread
    thread t2(diese); // création et lancement du thread

    t1.join(); // attendre la fin du thread
    t2.join(); // attendre la fin du thread

    cout << endl;

    return 0;
}
```

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 3.1)
PROJECT(threads)
SET(CMAKE_CXX_STANDARD 11)
SET(CMAKE_CXX_STANDARD_REQUIRED True)
SET(THREADS_PREFER_PTHREAD_FLAG ON)
FIND_PACKAGE(Threads REQUIRED)
ADD_EXECUTABLE(threads thread.cpp)
#TARGET_LINK_LIBRARIES(threads pthread)
TARGET_LINK_LIBRARIES(threads PRIVATE Threads::Threads)
```

```
$ mkdir build
$ cd build

$ cmake ..
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- 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
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Check if compiler accepts -pthread
-- Check if compiler accepts -pthread - yes
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: $HOME/exemples/exemple2/build

$ make
Scanning dependencies of target threads
[ 50%] Building CXX object CMakeFiles/threads.dir/thread.cpp.o
[100%] Linking CXX executable threads
[100%] Built target threads

$ ./threads
*#*#*#*#*#*##*#*#*#*
```

5. Exemple n°3

```
$ mkdir exemple3

$ tree exemple3
exemple3
├──── build
├──── CMakeLists.txt
├──── include
│    ├──── CMakeLists.txt
│    ├──── DeConfig.h.in
│    └──── de.h
├──── install
└──── src
     ├──── CMakeLists.txt
     ├──── lib
     │    ├──── CMakeLists.txt
     │    └──── de.cpp
     └──── main.cpp

$ cd exemple3
```

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 3.1)
PROJECT(DE VERSION 1.0)
SET(CMAKE_CXX_STANDARD 11) # pour C++11
SET(CMAKE_CXX_STANDARD_REQUIRED True)
#SET(DE_VERSION_MAJOR 1)
#SET(DE_VERSION_MINOR 0)
ADD_SUBDIRECTORY(include)
ADD_SUBDIRECTORY(src)
CONFIGURE_FILE(${CMAKE_SOURCE_DIR}/include/DeConfig.h.in ${CMAKE_SOURCE_DIR
}/include/DeConfig.h)
MESSAGE("-- Repertoire source du projet : ${CMAKE_SOURCE_DIR}")
```

- Dans `src` :

main.cpp

```
#include <iostream>
#include <stdlib.h>
#include <limits.h>
#include <stdio.h>
#include <errno.h>
#include "de.h"
#include "DeConfig.h"
```

```
using namespace std;

int main(int argc, char *argv[])
{
    char *endptr, *str;
    long val;

    if (argc == 2)
    {
        /* cf. man strtol */
        str = argv[1];
        errno = 0;    /* Pour distinguer la réussite/échec après l'appel */
        val = strtol(str, &endptr, 10);

        /* Vérification de certaines erreurs possibles */
        if ((errno == ERANGE && (val == LONG_MAX || val == LONG_MIN)) || (errno != 0
&& val == 0))
        {
           perror("strtol");
           exit(EXIT_FAILURE);
        }

        if (endptr == str)
        {
           cerr << "Erreur: il faut un nombre !" << endl;
           exit(EXIT_FAILURE);
        }

        if (*endptr != '\0')
        {
           cerr << "Erreur: il faut juste un nombre !" << endl;
           exit(EXIT_FAILURE);
        }
        /* Si nous sommes ici, strtol() a analysé un nombre avec succès */

        De de(val);
        de.lancer();
        cout << de.getValeur() << endl;
        exit(EXIT_SUCCESS);
    }
    else
    {
        cout << argv[0] << " version " << DE_VERSION_MAJOR << "." << DE_VERSION_MINOR
<< endl;
        cout << "Usage: " << argv[0] << " nombre" << endl;
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}
```

CMakeLists.txt

```
INCLUDE_DIRECTORIES(${CMAKE_SOURCE_DIR}/include)
ADD_SUBDIRECTORY(lib)
SET(LIB de)
ADD_EXECUTABLE(dice main.cpp)
TARGET_LINK_LIBRARIES(dice ${LIB})
INSTALL(TARGETS dice DESTINATION bin)
```

- Dans `src/lib` :

de.cpp

```
#include "de.h"
#include <iostream>
#include <chrono>
#include <random>

De::De(long nbFaces) : nbFaces(nbFaces), valeur(1), seed
(std::chrono::system_clock::now().time_since_epoch().count())
{
}

De::~De()
{
}

long De::getValeur() const
{
  return valeur;
}

void De::lancer()
{
    std::default_random_engine generator(seed);
    std::uniform_int_distribution<long> distribution(1, this->nbFaces);
    valeur = distribution(generator);  // generates number in the range 1..nbFaces
}
```

CMakeLists.txt

```
INCLUDE_DIRECTORIES(${CMAKE_SOURCE_DIR}/include)
FILE(GLOB SRC . *.cpp)
ADD_LIBRARY(de-static STATIC ${SRC})
ADD_LIBRARY(de SHARED ${SRC})
#TARGET_COMPILE_OPTIONS(de PRIVATE -DDEBUG)
INSTALL(TARGETS de DESTINATION lib)
INSTALL(TARGETS de-static DESTINATION lib-static)
```

- Dans `include` :

de.h

```
#ifndef DE_H
#define DE_H

class De
{
    private:
        const long nbFaces;
        long valeur;
        unsigned seed;

    public:
        explicit De(long nbFaces = 6);
        ~De();

        long getValeur() const;
        void lancer();
};

#endif
```

DeConfig.h.in

```
#ifndef CONFIG_H
#define CONFIG_H

#define DE_VERSION_MAJOR @DE_VERSION_MAJOR@
#define DE_VERSION_MINOR @DE_VERSION_MINOR@

#endif
```

CMakeLists.txt

```
FILE(GLOB HEADERS . *.h)
INSTALL(FILES ${HEADERS} DESTINATION include)
```

Fabrication :

```
$ cd build/

$ cmake -DCMAKE_INSTALL_PREFIX=../install ..
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- 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
-- Repertoire source du projet : $HOME/exemples/exemple3
-- Configuring done
-- Generating done
-- Build files have been written to: $HOME/exemples/exemple3/build

$ make
Scanning dependencies of target de
[ 16%] Building CXX object src/lib/CMakeFiles/de.dir/de.cpp.o
[ 33%] Linking CXX shared library libde.so
[ 33%] Built target de
Scanning dependencies of target dice
[ 50%] Building CXX object src/CMakeFiles/dice.dir/main.cpp.o
[ 66%] Linking CXX executable dice
[ 66%] Built target dice
Scanning dependencies of target de-static
[ 83%] Building CXX object src/lib/CMakeFiles/de-static.dir/de.cpp.o
[100%] Linking CXX static library libde-static.a
[100%] Built target de-static
```

Installation :

```
$ make install
[ 33%] Built target de
[ 66%] Built target dice
[100%] Built target de-static
Install the project...
-- Install configuration: ""
-- Installing: $HOME/exemples/exemple3/install/include/de.h
-- Installing: $HOME/exemples/exemple3/install/bin/dice
-- Set runtime path of "$HOME/exemples/exemple3/install/bin/dice" to ""
-- Installing: $HOME/exemples/exemple3/install/lib/libde.so
-- Installing: $HOME/exemples/exemple3/install/lib-static/libde-static.a
```

Tests :

```
$ cd ../install
$ tree
.
├─── bin
│    └─── dice
├─── include
│    └─── de.h
├─── lib
│    └─── libde.so
└─── lib-static
     └─── libde-static.a

$ ldd ./bin/dice
    linux-vdso.so.1 (0x00007ffe3d9f3000)
    libde.so => ./lib/libde.so (0x00007f4574e66000)
    libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f4574add000)
    libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f45748c5000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f45744d4000)
    libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f4574136000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f457526c000)

$ ./bin/dice
./bin/dice version 1.0
Usage: ./bin/dice nombre

$ ./bin/dice 6
3
$ ./bin/dice 6
5
$ ./bin/dice 6
6

$ ./bin/dice a
Erreur: il faut un nombre !

$ ./bin/dice 6a
Erreur: il faut juste un nombre !
```

6. Voir aussi

Les exemples de ce document : cmake-exemples.zip.

Site : tvaira.free.fr

Thierry Vaira - <tvaira@free.fr> - version v1.0 - 05/01/2021