CMake: Sistema de Compilación para C/C++

Daniel Molina

Universidad of Cádiz, Departamento de Lenguaje y Sistemas Informáticos, OSLUCA

29 de Junio 2009, Cádiz

Presentación

Qué es CMake

- CMake:
  - Genera ficheros de compilación.

Qué es CMake

- CMake:
  - Genera ficheros de compilación.
    - Unix/Linux → Makefiles.
    - Windows → Proyectos Visual Studio.
    - Apple → Xcode.

Qué es CMake

- CMake:
  - Genera ficheros de compilación.
    - Unix/Linux → Makefiles.
    - Windows → Proyectos Visual Studio.
    - Apple → Xcode.
  - Multiplataforma.

Qué es CMake

- CMake:
  - Genera ficheros de compilación.
    - Unix/Linux → Makefiles.
    - Windows → Proyectos Visual Studio.
    - Apple → Xcode.
  - Multiplataforma.
  - Software Libre :-).

Qué es CMake

- CMake:
  - Genera ficheros de compilación.
    - Unix/Linux → Makefiles.
    - Windows → Proyectos Visual Studio.
    - Apple → Xcode.
  - Multiplataforma.
  - Software Libre :-).

Pero si mi aplicación no es multiplataforma

Qué es CMake

- CMake:
  - Genera ficheros de compilación.
    - Unix/Linux → Makefiles.
    - Windows → Proyectos Visual Studio.
    - Apple → Xcode.
  - Multiplataforma.
  - Software Libre :-).

Pero si mi aplicación no es multiplataforma

- Más sencillo que Makefiles *a mano*.

Qué es CMake

- CMake:
  - Genera ficheros de compilación.
    - Unix/Linux → Makefiles.
    - Windows → Proyectos Visual Studio.
    - Apple → Xcode.
  - Multiplataforma.
  - Software Libre :-).

Pero si mi aplicación no es multiplataforma

- Más sencillo que Makefiles *a mano*.
- ¿Para eso no están las autotools?

Ventajas frente a Makefile y Autotools

CMake Versus Makefile

- Más cómodo y fácil.

CMake Versus Autotools

- Menor curva de aprendizaje.

Ventajas frente a Makefile y Autotools

CMake Versus Makefile

- Más cómodo y fácil.
- ¿He dicho que es más fácil?

CMake Versus Autotools

- Menor curva de aprendizaje.
- No usa M4 :-).

Ventajas frente a Makefile y Autotools

CMake Versus Makefile

- Más cómodo y fácil.
- ¿He dicho que es más fácil?
- Portable.

CMake Versus Autotools

- Menor curva de aprendizaje.
- No usa M4 :-).
- Más portable.

Ventajas frente a Makefile y Autotools

CMake Versus Makefile

- Más cómodo y fácil.
- ¿He dicho que es más fácil?
- Portable.
- Más opciones (búsqueda de librerías, ...).

CMake Versus Autotools

- Menor curva de aprendizaje.
- No usa M4 :-).
- Más portable.
- Más fácil de extender.

Ventajas frente a Makefile y Autotools

CMake Versus Makefile

- Más cómodo y fácil.
- ¿He dicho que es más fácil?
- Portable.
- Más opciones (búsqueda de librerías, ...).

CMake Versus Autotools

- Menor curva de aprendizaje.
- No usa M4 :-).
- Más portable.
- Más fácil de extender.
- Mejor documentado.

Características de CMake

- Utilizado en entornos complejos (VTK+, KDE4, ...).
- Flexible y Extensible.
- Soporte de macros (buscar/configurar software).
- Ejecutar programar externos.
- Syntaxis Intuitiva.
- Modos de compilación (Debug, Release, ...).
- Admite jerarquía de directorios complejas, y detecta librerías.

Quién lo está usando

- Kitware, `http://www.kitware.com/`
- The Visualization ToolKit (VTK), `http://www.vtk.org/`
- ParaView, `http://www.paraview.org/`
- KDE 4, `https://lwn.net/Articles/188693/`

Estructura

- CMakefiles.txt: Formato del fichero, define el flujo de control en sintaxis CMake.
- Módulos CMake: Módulos que extienden la funcionalidad de CMake, principalmente en buscar aplicaciones/herramientas.
  - Ejemplo: FindQt4.cmake, FindJava.cmake

Ejecutando CMake

- Preparar el fichero CMakeLists.txt:
  - Con editor favorito.
  - Visualmente: cmake-gui.
- Ejecutar *cmake .* $\rightarrow$ Makefile
- Compilar (*make*)

Ejemplo: "Hello, World"

Estructura

- hello_src
  - hello.c
  - CMakeLists.txt

CMakeLists.txt

```
PROJECT(hello C)
SET(SRC
    hello.c)

ADD_EXECUTABLE(hello ${SRC})
```

Ejemplo: "Hello, World"

Estructura

- hello_src
  - hello.c
  - CMakeLists.txt
- Makefile

CMakeLists.txt

```
PROJECT(hello C)
SET(SRC
    hello.c)

ADD_EXECUTABLE(hello ${SRC})
```

Ejemplo: "Hello, World"

Estructura

- hello_src
  - hello.c
  - CMakeLists.txt
- Makefile
- CMakeFiles
- CMakeCache.txt
- cmake_install.cmake
- hello

CMakeLists.txt

```
PROJECT(hello C)
SET(SRC
    hello.c)

ADD_EXECUTABLE(hello ${SRC})
```

Ejemplo: "Hello, World"

Estructura

- hello_src
  - hello.c
  - CMakeLists.txt
- Makefile
- CMakeFiles
- CMakeCache.txt
- cmake_install.cmake
- hello → Ejecutable

Ejemplo: "Hello, World"

Estructura

- hello_src
  - hello.c
  - CMakeLists.txt
- Makefile → Fichero autogenerado
- CMakeFiles
- CMakeCache.txt
- cmake_install.cmake
- hello → Ejecutable

Ejemplo: "Hello, World"

Estructura

- hello_src
  - hello.c
  - CMakeLists.txt
- Makefile → Fichero autogenerado
- CMakeFiles → Directorio con ficheros objeto (.o)
- CMakeCache.txt
- cmake_install.cmake
- hello → Ejecutable

Ejemplo: "Hello, World"

Estructura

- hello_src
  - hello.c
  - CMakeLists.txt
- Makefile → Fichero autogenerado
- CMakeFiles → Directorio con ficheros objeto (.o)
- CMakeCache.txt → Fichero caché (evitar recompilar innecesariamente)
- cmake_install.cmake
- hello → Ejecutable

Ejemplo: "Hello, World"

Estructura

- hello_src
  - hello.c
  - CMakeLists.txt
- Makefile → Fichero autogenerado
- CMakeFiles → Directorio con ficheros objeto (.o)
- CMakeCache.txt → Fichero caché (evitar recompilar innecesariamente)
- cmake_install.cmake → Fichero de instalación/desistalación (cuando se usa)
- hello → Ejecutable

Ejemplo: "Hello, World"

- Otra alternativa:
  - mkdir build
  - cd build
  - cmake ..
- Así, todos los ficheros adicionales están en bin
- También se puede hacer configurando adecuadamente CMakeLists.txt

Otras versiones

hellolib.c

```
#include <stdio.h>
#include <string.h>

void get_msg(char *msg) {
    strcpy(msg, "hello, world\n");
}
```

hellolib.h

```
#ifndef _HELLO_LIB_H

#define _HELLO_LIB_H 1

void get_msg(char *msg);

#endif
```

hello.c

```
/* hello.c: display a message
   on the screen */
#include <stdio.h>
#include "hellolib.h"

int main(void) {
   char msg[30];

   get_msg(msg);
   printf(msg);
   return 0;
}
```

Usando Librería Dinámica Versus Sin Usar

Sin ibrería

```
PROJECT(hello C)
SET(SRC
    hello
    hellolib)

ADD_EXECUTABLE(hello ${SRC})
```

Con Librería Dinámica

```
PROJECT(hello C)
SET(LIBSRC
    hellolib)

SET(SRC
    hello)

ADD_LIBRARY(hellolib SHARED ${LIBSRC})

ADD_EXECUTABLE(hello ${SRC})

TARGET_LINK_LIBRARIES(hello hellolib)
```

Espero que esta introducción motive para el taller

¿Empezamos? :-)