Introducción a CMake

Jesús Nuevo Chiquero

Department of Electronics , Univerisity of Alcalá.

3 de diciembre de 2009

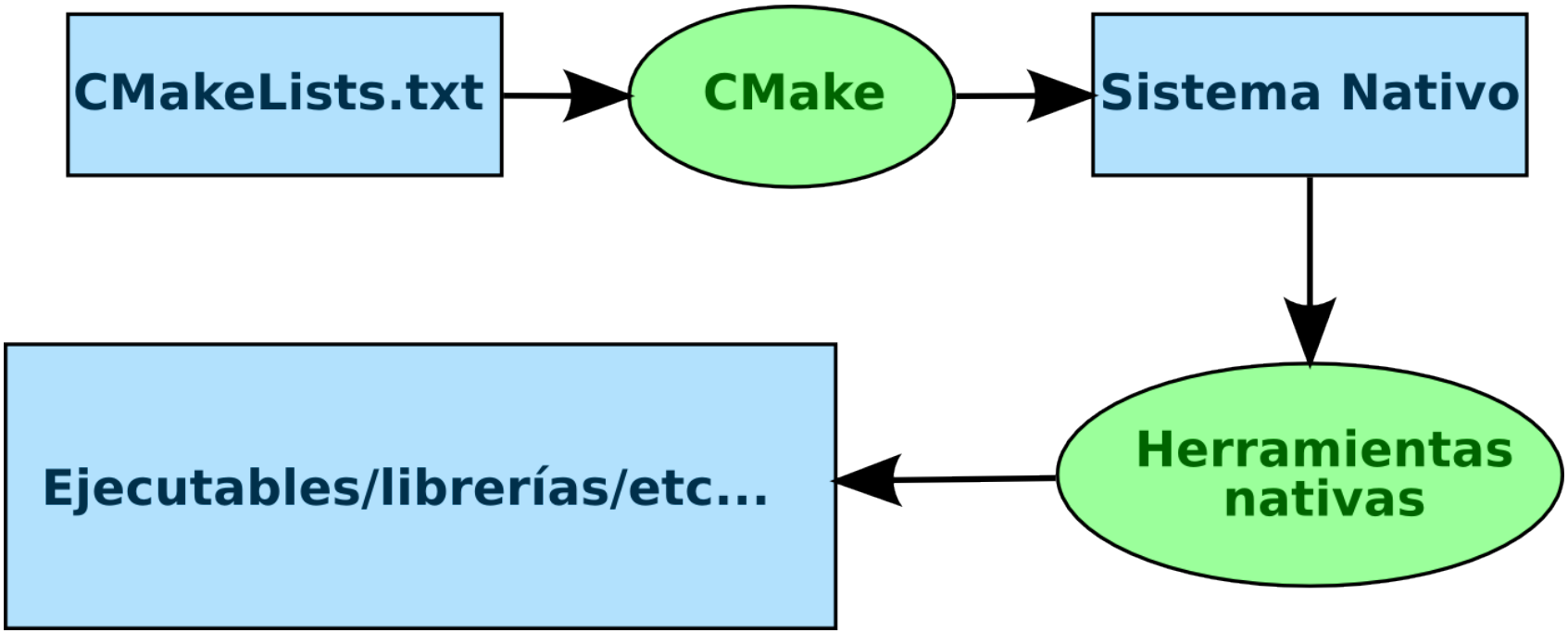
Outline

¿Qué es CMake?

CMake = Cross-Platfrom Make

- Es un sistema de *construcción*: compilar, testear y distribuir software
- No reemplaza los sistemas nativos. **Genera** entornos nativos
  - Makefiles - Unix, Linux
  - VStudio proyectos/Workspaces - MS Windows
  - Xcode - Mac OS X
  - Eclipse - Multiplataforma
- Open-source (y gratis)
- Multi-plataforma: configura 1 vez, compila en múltiples entornos.

Generador de CMake



Características básicas

- Multi-plataforma
- Sencillo, sintáxis intuitiva
- Flexible
  - Macros
  - Módulos para encontrar y configurar software
  - Comandos y objetivos de compilación personalizables
  - Lanzamiento de programas externos
  - *RegEx*, estilo Unix
  - ...
- Compilación cruzada
- Herramientas de instalación
- Herramientas de test y empaquetado (CTest, CPack).

¿Y cómo se configura esto?

Dos componentes fundamentales

- **CMakeLists.txt**
  - Describen los proyectos
  - Control de parámetros de compilación
  - Flujo de compilación
  - Configuración de instalación, etc...

- **Módulos CMake**

¿Y cómo se configura esto?

Dos componentes fundamentales

- **CMakeLists.txt**
  - Describen los proyectos
  - Control de parámetros de compilación
  - Flujo de compilación
  - Configuración de instalación, etc...
  - **Estos son los archivos que vamos a escribir**
- **Módulos CMake**

¿Y cómo se configura esto?

Dos componentes fundamentales

- **CMakeLists.txt**
  - Describen los proyectos
  - Control de parámetros de compilación
  - Flujo de compilación
  - Configuración de instalación, etc...
  - **Estos son los archivos que vamos a escribir**
- **Módulos CMake**
  - Archivos que encuentran bibliotecas de funciones, includes, definiciones, etc..
  - Ej.: FindQt4.cmake, FindBLAS.cmake,...

Configurando un software (ya hecho)

- Nos bajamos un software que usa CMake
- y queremos configurarlo e instalarlo.

Configurando un software (ya hecho)

- Nos bajamos un software que usa CMake
- y queremos configurarlo e instalarlo.

Veamos un ejemplo con las OpenCV 2.0.

- Utilizando CMake 2.6 en GNU/Linux.
- Herramienta `ccmake` (Unix/Linux),
- `CMakeSetup.exe` (Windows)
- `qtmake` (a partir de CMake 2.8)

Estas herramientas generan la caché de CMake.

Configurando un software (ya hecho)

- Nos bajamos un software que usa CMake
- y queremos configurarlo e instalarlo.

Veamos un ejemplo con las OpenCV 2.0.

- Utilizando CMake 2.6 en GNU/Linux.
- Herramienta `ccmake` (Unix/Linux),
- `CMakeSetup.exe` (Windows)
- `qtmake` (a partir de CMake 2.8)

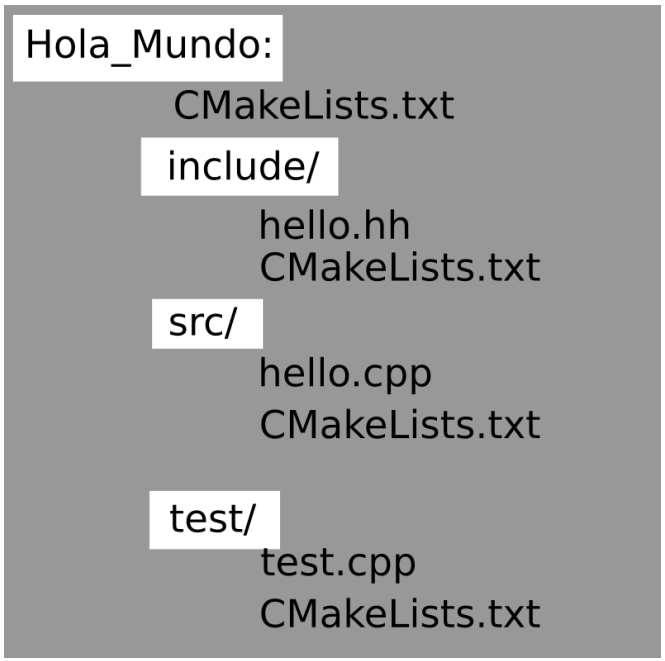
Estas herramientas generan la caché de CMake.

- Configuración con `cmake`.
- Generamos un Makefile
- Compilamos con `make`

Hola Mundo con CMake

Un proyecto se puede distribuir en múltiples directorios y sub-directorios.

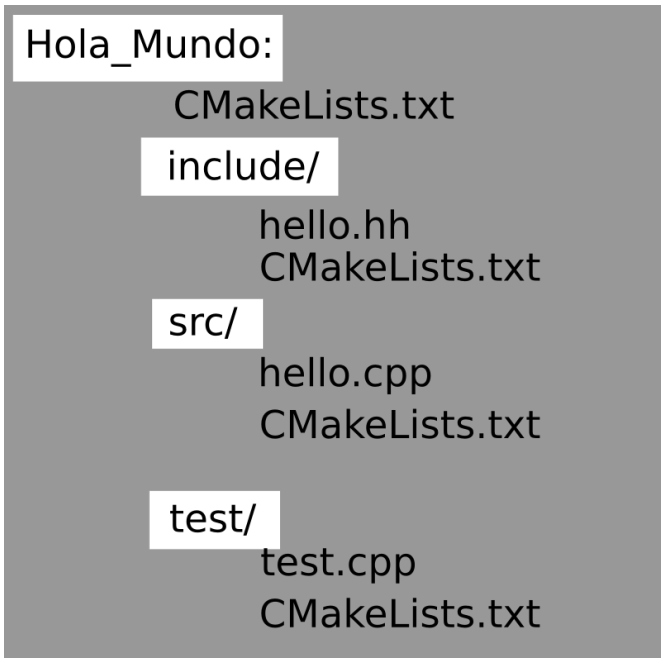
- Los comandos de CMake se guardan al bajar subdirectorios.



Hola Mundo con CMake

Un proyecto se puede distribuir en múltiples directorios y sub-directorios.

- Los comandos de CMake se guardan al bajar subdirectorios.



Creemos una librería y un ejecutable para testearla.

Hola Mundo: añadiendo opciones

Queremos modificar los parámetros de compilación:

- Añadir símbolos de depuración,
- Activar optimizaciones y otras opciones del compilador
- Activar definiciones,
- etc, etc...

Hola Mundo: añadiendo opciones

Queremos modificar los parámetros de compilación:

- Añadir símbolos de depuración,
- Activar optimizaciones y otras opciones del compilador
- Activar definiciones,
- etc, etc...

Y queremos hacerlo para cada objetivo por separado

Hola Mundo: incluyendo librerías

Queremos incluir dos librerías, necesarias para nuestro software.

- Incluiremos las OpenCV 2.0, y las librearías JPEG

Hola Mundo: incluyendo librerías

Queremos incluir dos librerías, necesarias para nuestro software.

- Incluiremos las OpenCV 2.0, y las librearías JPEG
- `FIND_PACKAGE(<package> [VERSION] [REQUIRED|component] ... )`
- Este comando busca módulos en ciertos directorios, y fija ciertas variables. Comúnmente:
  - <package>_INCLUDE_DIR: directorios de las cabeceras
  - <package>_LIBRARY_DIR: directorios de las librerías
  - <package>_LIBRARIES: las librerías para linkar
  - <package>_VERSION: versión encontrada
  - ...
- Hay otras opciones: *FIND_FILE()*, *FIND_LIBRARY()*, ...

Agrupando archivos, fijando variables

Cuando hay muchos archivos o librerías, es útil incluirlos en una lista

- `FILE(GLOB <variable>` *regEx*`)`
  - `FILE(GLOB MY_SOURCES “*.cpp”)`
  - `FILE(GLOB MY_MAIN “main.cpp”)`
  - `FILE(GLOB MY_HEADERS “include/*.hh”)`

Agrupando archivos, fijando variables

Cuando hay muchos archivos o librerías, es útil incluirlos en una lista

- `FILE(GLOB <variable>` *regEx*`)`
  - `FILE(GLOB MY_SOURCES “*.cpp”)`
  - `FILE(GLOB MY_MAIN “main.cpp”)`
  - `FILE(GLOB MY_HEADERS “include/*.hh”)`
- Las variables son una lista, que es modificable:
  - `LIST(REMOVE_ITEM MY_SOURCES ${MY_MAIN})`
  - `LIST(APPEND MY_SOURCES “foo.c”)`
  - y otros comandos: `INSERT`, `LENGTH`, `GET`, `REMOVE_ITEM`, `SORT`, ...

Agrupando archivos, fijando variables

Cuando hay muchos archivos o librerías, es útil incluirlos en una lista

- `FILE(GLOB <variable> regEx)`
  - `FILE(GLOB MY_SOURCES "*.cpp")`
  - `FILE(GLOB MY_MAIN "main.cpp")`
  - `FILE(GLOB MY_HEADERS "include/*.hh")`
- Las variables son una lista, que es modificable:
  - `LIST(REMOVE_ITEM MY_SOURCES ${MY_MAIN})`
  - `LIST(APPEND MY_SOURCES "foo.c")`
  - y otros comandos: `INSERT`, `LENGTH`, `GET`, `REMOVE_ITEM`, `SORT`, ...
- Estas variables se usan fácilmente:
  - `ADD_EXECUTABLE(mi_exec ${MY_SOURCES} ${MY_MAIN} )`

Opciones y condicionales

Las opciones como las que vimos al principio (OpenCV), se declaran como:

- `OPTION(var “descripción” [ON|OFF])`

Otra posibilidad es:

- `SET( VAR valor [CACHE <type> <docstring> [FORCE]])`
  - `CACHE` incluye la variable en la caché de CMake
  - `<type>` puede ser FILEPATH, PATH, STRING, BOOL, INTERNAL.
  - `FORCE` sobreescribe siempre la caché.
- ej.: `set(INSTALL_TOOLS FALSE CACHE BOOL "Install tools")`

Opciones y condicionales 2

Las variables se pueden usar en expresiones condicionales: IF, WHILE, FOREACH, ...

```
IF(WIN32) # o  APPLE
#hacer algo
ELSEIF(UNIX)
#hacer otra cosa
ENDIF()
%%%%%%%%%%%%
OPTION(WITH\_DEBUG  "Build with debug code" ON)
IF(WITH\_DEBUG)
   ADD_DEFINITIONS(-D\_DEBUG)
ELSE(WITH\_DEBUG)
   REMOVE_DEFINITIONS(-D\_DEBUG)
ENDIF()
```

Opciones y condicionales 2

Las variables se pueden usar en expresiones condicionales: IF, WHILE, FOREACH, ...

```
IF(WIN32) # o  APPLE
#hacer algo
ELSEIF(UNIX)
#hacer otra cosa
ENDIF()
%%%%%%%%%%%%
OPTION(WITH\_DEBUG  "Build with debug code" ON)
IF(WITH\_DEBUG)
   ADD_DEFINITIONS(-D\_DEBUG)
ELSE(WITH\_DEBUG)
   REMOVE_DEFINITIONS(-D\_DEBUG)
ENDIF()
```

Hay maneras mejores de hacer esto... $\longrightarrow$ CMAKE_BUILD_TYPE

Instalación de archivos

Queremos instalar nuestra librería y el test en nuestro ordenador.

- Comando `INSTALL`

Instalación de archivos

Queremos instalar nuestra librería y el test en nuestro ordenador.

- Comando `INSTALL`

Veamos un ejemplo: *hola mundo 4*

Instalación de archivos

Queremos instalar nuestra librería y el test en nuestro ordenador.

- Comando `INSTALL`

¿Y desistarlo?

- No es automático: requiere crear un *target* específico.
- Viene en las FAQ, y es sencillo: vamos a verlo.

¿Y las otras arquitecturas?

- CMake tiene un *generador* para cada sistema de construcción.
- `cmake . -G "generador"`

¿Y las otras arquitecturas?

- CMake tiene un *generador* para cada sistema de construcción.
- `cmake . -G “generador”`
- Veamos un ejemplo para Eclipse.
- `cmake -G"Eclipse CDT4 - Unix Makefiles"`

Más información

Esto sólo ha sido una introducción:

- CMake tiene muchas más opciones. Compilar archivos LaTeX, generar documentación con Doxygen, etc... es muy sencillo.

Algunos sitios con más información:

- CMake.org - referencia oficial de CMake
- CMake Wiki - muy útil
- Tutoriales varios:
  - CMakeTutorial
  - Learning CMake