Sistemas Informáticos Industriales

Tema 8. Herramientas de desarrollo

Centro: Escuela Técnica Superior de Ingeniería y Diseño Industrial

Departamento: Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

11 de Septiembre de 2015

Tema 8. Herramientas de desarrollo

Tema 8. Índice

1. Sistemas de control de versiones
2. CMake
3. Pruebas unitarias (CTest)

CMake

¿Por qué hablar de "build systems"?

- Los desarrolladores escriben el código fuente de una aplicación y necesitan:
  - Compilar el código fuente
  - Enlazar otras librerías
  - Distribuir la aplicación como fuente y/o binario

- Y además:
  - Ejecutar tests
  - Ver los resultados de ello

CMake

Compilación y linkado:

- Manual:

  gcc –options –c myapp.o myapp.cpp –lc –lmylib

- Pero se hace imposible cuando:
  - Hay muchos ficheros
  - Algunos ficheros deberían compilarse únicamente en una plataforma particular
  - Hay diferente código dependiendo de la plataforma / compilador / debug/ release, etc.

Distribución de software:

- Los desarrolladores desarrollan código
- Una vez el software está finalizado, otros se encargan de empaquetarlo
- Hay muchos formatos de empaquetado dependiendo de la versión del SO, plataforma, distribución Linux, etc.: .deb, .rpm, .msi, .dmg, .targ.gz, InstallShield, etc.

CMake

Testing:

- Se desarrollan (o se debería hacer) pruebas unitarias para testear el software
- ¿Cuándo y cómo se ejecutan los tests unitarios? Normalmente se sigue un proceso de tres pasos:
  - Se invoca el proceso de construcción (p. ej. make)
  - Cuando termina, se ejecuta manualmente la suite de test (colección de casos de test)
  - Cuando termina, se mira los resultados y se buscan los errores y/o warnings

Se necesitan sistemas de construcción modernos que sean capaces de construir el software, empaquetarlo y testearlo

CMake

Lo usan: Kitware, VTK, ParaView, KDE 4, Biicode …

CMake

CMake

- Abreviatura de: Cross Platform Make
- Herramienta software libre multiplataforma de generación o automatización de código
- Familia de herramientas para construir, probar y empaquetar software
- Características principales:
  - Lenguaje de script muy simple
  - Descubrimiento óptimo de dependencias
  - Admite jerarquía de directorios complejas y detecta librerías
  - Crea makefiles nativos y workspaces para Makefile, Visual C++, Kdevelop, Eclipse, Code::Blocks, Borland, MinGW, etc.
    - Por tanto, los usuarios pueden seguir usando las herramientas que suelen utilizar

- El flujo se define por medio de ficheros de configuración, llamados **CMakeLists.txt**, en cada directorio (y subdirectorios)

COMANDO (argumentos …)

CMake

Pasos para crear un Makefile de Linux

- Preparar el fichero CMakeLists.txt
  - Con editor favorito
  - Visualmente: cmake-gui (en Linux)
- Ejecutar *cmake* → Se genera Makefile
- Compilar *make*

Formato de un fichero **Makefile**:

objetivo: dependencia dependencia …
→ accion
→ accion
…
objetivo: dependencia …
→ accion …

CMake

Ejecutable muy simple

```
PROJECT (helloworld)
SET (hello_SRCS hello.c)
ADD_EXECUTABLE (hello ${hello_SRCS})
```

- PROJECT no es obligatorio pero se debería utilizar
- ADD_EXECUTABLE crea un ejecutable desde los códigos fuente listados

SET (var1 13)

Establece una variable “var1” con el valor 13

CMake

Para Makefiles de UNIX

Se crea la carpeta de construcción:

$ mkdir build

$ cd build

$ cmake ..

Para Code::Blocks (se crea el workspace con el fichero del proyecto)

$ mkdir buildcb

$ cd buildcb

$ cmake –G 'CodeBlocks – Unix Makefiles' ..

CMake

Estructura

hello.c

CMakeLists.txt

Makefile

CMakeFiles

CMakeCache.txt

cmake_install.cmake

hello

CMake

Estructura

hello.c

CMakeLists.txt

Makefile

CMakeFiles

CMakeCache.txt

cmake_install.cmake

hello → **Ejecutable**

CMake

Estructura

hello.c

CMakeLists.txt

Makefile → **Fichero autogenerado**

CMakeFiles

CMakeCache.txt

cmake_install.cmake

hello

CMake

Estructura

hello.c

CMakeLists.txt

Makefile

CMakeFiles → **Directorio con ficheros objeto (.o)**

CMakeCache.txt

cmake_install.cmake

hello

CMake

Estructura

hello.c

CMakeLists.txt

Makefile

CMakeFiles

CMakeCache.txt → **Fichero caché (evitar recompilar innecesariamente)**

cmake_install.cmake

hello

CMake

Estructura

hello.c

CMakeLists.txt

Makefile

CMakeFiles

CMakeCache.txt

cmake_install.cmake → **Fichero de instalación/desinstalación (cuando se usa)**

hello

CMake

Librería muy simple

```
PROJECT (mylibrary)
SET (mylib_SRCS library.cpp)
ADD_LIBRARY (my SHARED ${mylib_SRCS})
```

- ADD_LIBRARY crea una librería estática desde los códigos fuente listados
- Se añade SHARED si se quiere generar librerías dinámicas (Windows) o compartidas (UNIX)

CTest

- Características de una buena prueba unitaria:
  - Se tienen que poder ejecutar sin necesidad de intervención manual
  - Deben poder cubrir la totalidad de código de la aplicación
  - Una prueba no puede afectar la ejecución de la otra

- CTest es una herramienta de test que forma parte de CMake
- Se puede utilizar por medio de comandos especiales en el CMakeLists.txt para crear tests

CTest

Estructura

- CMake permite añadir tests a un proyecto:
  enable_testing()
  Esto añade el target “test” en los generadores de Makefile

- Se usa el comando ADD_TEST para añadir un test al proyecto:
  add_test (testname exename arg1 arg2 …)
- Se usa el comando SET_TESTS_PROPERTIES para añadir propiedades de los tests:
  set_tests_properties (testname PROPERTIES prop1 value1 prop2 value2 …)

- Una vez se ha construido el proyecto, se pueden ejecutar todos los tests con dos opciones:
  make test
  ctest

Referencias

- Libro "**Mastering CMake**". Publisher: Kitware, Inc. 5ª edición (Marzo 2010).
  ISBN-13: 978-1-930934-22-1

- Web oficial de **CMake** (tutoriales, documentación, comandos, etc.):
  http://www.cmake.org

- **Ejemplo** paso a paso que cubre todas las fases de un proyecto (CMake, CTest, CPack, CDash) – Tutorial incluido en el libro "Mastering CMake"
  www.cmake.org/cmake-tutorial/

- Ayuda en **línea de comandos**:
  man cmake