Build facili con CMake

Il sistema di build -definitivo-

Carlo Nicolini

December 6, 2012

Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.

Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.
- Gestione di molti tipi di build diversa

Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.
- Gestione di molti tipi di build diversa
- Supporto a librerie esterne

Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.
- Gestione di molti tipi di build diversa
- Supporto a librerie esterne
- Facilità di mantenimento di un progetto

Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.
- Gestione di molti tipi di build diversa
- Supporto a librerie esterne
- Facilità di mantenimento di un progetto
- Chiarezza ed intuitività, **velocità** di build

Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.
- Gestione di molti tipi di build diversa
- Supporto a librerie esterne
- Facilità di mantenimento di un progetto
- Chiarezza ed intuitività, **velocità** di build
- Build parallele

Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.
- Gestione di molti tipi di build diversa
- Supporto a librerie esterne
- Facilità di mantenimento di un progetto
- Chiarezza ed intuitività, **velocità** di build
- Build parallele
- Miglior rapporto con chi userà il nostro codice

Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.
- Gestione di molti tipi di build diversa
- Supporto a librerie esterne
- Facilità di mantenimento di un progetto
- Chiarezza ed intuitività, **velocità** di build
- Build parallele
- Miglior rapporto con chi userà il nostro codice
- Packing, unit testing, profiling e debugging ALL-IN-ONE

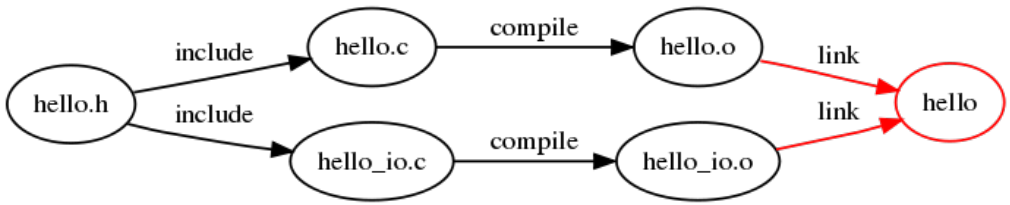
Motivazioni

Un sistema di build per C/C++ ci permette di essere più efficienti, produttivi e chiari. Vogliamo un sistema di build che ci permetta

- Portabilità su molti O.S.
- Gestione di molti tipi di build diversa
- Supporto a librerie esterne
- Facilità di mantenimento di un progetto
- Chiarezza ed intuitività, **velocità** di build
- Build parallele
- Miglior rapporto con chi userà il nostro codice
- Packing, unit testing, profiling e debugging ALL-IN-ONE
- *Scriptabilità*

Sistemi di build

Un sistema di build si occupa di generare eseguibili/librerie partendo dai sorgenti [1]



Problema delle dipendenze = punti d'articolazione su grafo DAG.
Limite al parallelismo (CMake supporta build multiprocessore).

[1] http://www.cs.virginia.edu/~dww4s/articles/build_systems.html

In passato

In passato sono apparsi tanti sistemi, ognuno con i suoi pro e contro. Dimentichiamo la compilazione manuale (anni 70)

```
gcc -c mylibrary.cpp mylibrary.h -o mylibrary.o
```

e passiamo ai sistemi di build automatici:

- Scons *basato su Python, cross-platform*

[2] http://en.wikipedia.org/wiki/Perforce_Jam
[3] http://docs.waf.googlecode.com/git/book_17/waf.pdf

In passato

In passato sono apparsi tanti sistemi, ognuno con i suoi pro e contro. Dimentichiamo la compilazione manuale (anni 70)

```
gcc -c mylibrary.cpp mylibrary.h -o mylibrary.o
```

e passiamo ai sistemi di build automatici:

- Scons *basato su Python, cross-platform*
- Autotools *molto usato, sintassi complicata (Autohell), solo Unix*

[2] http://en.wikipedia.org/wiki/Perforce_Jam
[3] http://docs.waf.googlecode.com/git/book_17/waf.pdf

In passato

In passato sono apparsi tanti sistemi, ognuno con i suoi pro e contro.
Dimentichiamo la compilazione manuale (anni 70)

```
gcc -c mylibrary.cpp mylibrary.h -o mylibrary.o
```

e passiamo ai sistemi di build automatici:

- Scons *basato su Python, cross-platform*
- Autotools *molto usato, sintassi complicata (Autohell), solo Unix*
- Jam *(cross-platform,cross-language), buggy, poco automatico*[2]

[2] http://en.wikipedia.org/wiki/Perforce_Jam
[3] http://docs.waf.googlecode.com/git/book_17/waf.pdf

In passato

In passato sono apparsi tanti sistemi, ognuno con i suoi pro e contro.
Dimentichiamo la compilazione manuale (anni 70)

```
gcc -c mylibrary.cpp mylibrary.h -o mylibrary.o
```

e passiamo ai sistemi di build automatici:

- Scons *basato su Python, cross-platform*
- Autotools *molto usato, sintassi complicata (Autohell), solo Unix*
- Jam *(cross-platform,cross-language), buggy, poco automatico*[2]
- Waf *python un singolo file da redistribuire* [3]

[2] http://en.wikipedia.org/wiki/Perforce_Jam
[3] http://docs.waf.googlecode.com/git/book_17/waf.pdf

In passato

In passato sono apparsi tanti sistemi, ognuno con i suoi pro e contro. Dimentichiamo la compilazione manuale (anni 70)

```
gcc -c mylibrary.cpp mylibrary.h -o mylibrary.o
```

e passiamo ai sistemi di build automatici:

- Scons *basato su Python, cross-platform*
- Autotools *molto usato, sintassi complicata (Autohell), solo Unix*
- Jam *(cross-platform,cross-language), buggy, poco automatico*[2]
- Waf *python un singolo file da redistribuire* [3]
- Altri?

[2] http://en.wikipedia.org/wiki/Perforce_Jam
[3] http://docs.waf.googlecode.com/git/book_17/waf.pdf

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)
- Supporto a tanti ambienti di sviluppo (IDE) tramite *generatori*[5]: Kdevelop3, Eclipse, XCode, Code::Blocks ,VisualStudio, Makefiles (Unix, NMake, Borland, MinGW, Cygwin)

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)
- Supporto a tanti ambienti di sviluppo (IDE) tramite *generatori*[5]: Kdevelop3, Eclipse, XCode, Code::Blocks ,VisualStudio, Makefiles (Unix, NMake, Borland, MinGW, Cygwin)
- Cross plattform (veramente).

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)
- Supporto a tanti ambienti di sviluppo (IDE) tramite *generatori*[5]: Kdevelop3, Eclipse, XCode, Code::Blocks ,VisualStudio, Makefiles (Unix, NMake, Borland, MinGW, Cygwin)
- Cross plattform (veramente).
- Dipendenze soddisfatte sempre (veramente).

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)
- Supporto a tanti ambienti di sviluppo (IDE) tramite *generatori*[5]: Kdevelop3, Eclipse, XCode, Code::Blocks ,VisualStudio, Makefiles (Unix, NMake, Borland, MinGW, Cygwin)
- Cross plattform (veramente).
- Dipendenze soddisfatte sempre (veramente).
- Un linguaggio di scripting che da libertà (vera).

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)
- Supporto a tanti ambienti di sviluppo (IDE) tramite *generatori*[5]: Kdevelop3, Eclipse, XCode, Code::Blocks ,VisualStudio, Makefiles (Unix, NMake, Borland, MinGW, Cygwin)
- Cross plattform (veramente).
- Dipendenze soddisfatte sempre (veramente).
- Un linguaggio di scripting che da libertà (vera).
  - `#define` a compile-time tramite variabili scriptabili.

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)
- Supporto a tanti ambienti di sviluppo (IDE) tramite *generatori*[5]: Kdevelop3, Eclipse, XCode, Code::Blocks ,VisualStudio, Makefiles (Unix, NMake, Borland, MinGW, Cygwin)
- Cross plattform (veramente).
- Dipendenze soddisfatte sempre (veramente).
- Un linguaggio di scripting che da libertà (vera).
  - `#define` a compile-time tramite variabili scriptabili.
  - Supporto menu Gnome, icone e creazione setup personalizzati.

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)
- Supporto a tanti ambienti di sviluppo (IDE) tramite *generatori*[5]: Kdevelop3, Eclipse, XCode, Code::Blocks ,VisualStudio, Makefiles (Unix, NMake, Borland, MinGW, Cygwin)
- Cross plattform (veramente).
- Dipendenze soddisfatte sempre (veramente).
- Un linguaggio di scripting che da libertà (vera).
  - `#define` a compile-time tramite variabili scriptabili.
  - Supporto menu Gnome, icone e creazione setup personalizzati.
  - Centinaia di librerie esterne supportate.

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

CMake

Un sistema di build moderno gestisce building, testing e packaging tutto insieme, scritto in C++, supporta progetti C/C++[4].

- Sistema di meta-make, multiprogetto (targets)
- Supporto a tanti ambienti di sviluppo (IDE) tramite *generatori*[5]: Kdevelop3, Eclipse, XCode, Code::Blocks ,VisualStudio, Makefiles (Unix, NMake, Borland, MinGW, Cygwin)
- Cross plattform (veramente).
- Dipendenze soddisfatte sempre (veramente).
- Un linguaggio di scripting che da libertà (vera).
  - `#define` a compile-time tramite variabili scriptabili.
  - Supporto menu Gnome, icone e creazione setup personalizzati.
  - Centinaia di librerie esterne supportate.
- **Out-of-source** builds: fare una build senza "sporcare" la codebase

[4] `www.cmake.org`

[5] `http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Generators`

Moduli supportati

Forniti insieme a CMake

ALSA, Armadillo, ASPELL, AVIFile, BISON, **BLAS**, **Boost**, Bullet, BZip2, CABLE, Coin3D, **CUDA**, Cups, CURL, Curses, CVS, CxxTest, **Cygwin**, Dart, DCMTK, DevIL, Doxygen, EXPAT, FLEX, FLTK2, FLTK, Gettext, GIF, Git, GLU, **GLUT**, Gnuplot, GnuTLS, GTest, GTK2, GTK, HDF5, HSPELL, HTMLHelp, **ImageMagick**, ITK, Jasper, Java, JNI, JPEG, KDE3, **KDE4**, LAPACK, LATEX, LibArchive, LibXml2, LibXslt, Lua50, Lua51, Matlab, MFC, Motif, MPEG2, MPEG, MPI, OpenAL, OpenGL, OpenMP, OpenSceneGraph, OpenSSL, OpenThreads, osgAnimation, PackageMessage, **Perl**, PerlLibs, PHP4, PhysFS, Pike, PkgConfig, PNG, PostgreSQL, PythonInterp, **PythonLibs**, Qt3, **Qt4**, QuickTime, Ruby, SDL, SelfPackers, Subversion, SWIG, TCL, Tclsh, TclStub, **Threads**, TIFF, UnixCommands, VTK, Wget, Wish, wxWidgets, wxWindows, X11, XMLRPC, ZLIB

e tanti altri si trovano sul web

CMake tree e primi passi

Tree di un progetto base

- **src**/
  - myapp.cpp
  - myapp.h
  - CMakeLists.txt
- **build**/
- *cmake*/
- *doc*/
- *deps*/
- *doc*/
- CMakeLists.txt

Opzioni di compilazione

- Definizione di opzioni di compilazione
- Switch su opzioni
- Switch su compilatori
- Switch su librerie
- etc...

```
//CMakeLists.txt
option(MyOption "MyOption" OFF)
//Da linea di comando
cmake -DMyOption=ON .
```

Esempio 07-qt+opengl

Compatibilità con versioni precedenti

- Molto importante impostare la compatibilita con le versioni precedenti, cambi di sintassi, bug-fix e nuovi moduli.
- Mantenere sempre CMake all'ultima versione (attuale 2.8.10).
- Variabile di sistema apposita.

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6.0 FATAL_ERROR)
```

Variabili in CMake

- Non serve dichiararle (stringa vuota se non esistono).
- Atipizzate.
- SET crea e modifica variabili.
- SEPARATE_ARGUMENTS spezza argomenti separati da spazio in una LIST.
- Da Cmake $>$ 2.6 variabili scoped.
- Dereferenza stile bash (attenzione all'assegnazione!)

Attenzione

```
FOO="fooval"
BAR=FOO
QUX=${FOO}
```

BAR e QUX hanno valori diversi! (tutte variabili sono stringhe)

Comando SET

Settare una variabile in CMake con il comando **SET**[6]

Esempio di uso del comando SET

```
SET(USE_BOOST_LIBRARIES "ON")
message(STATUS "Are you using Boost libraries?
    ${USE_BOOST_LIBRARIES})

SET( MyProjectHeaders "Ciccio.h Pippo.h Mimmo.h")
set( MyProjectSources "Ciccio.cpp Mimmo.cpp" )
```

[6] I comandi sono case-insensitive

Sintassi

- Costrutto condizionale IF

```
IF ( expression )
...
ELSE ( expression )
...
ENDIF ( expression )
```

- Costrutto FOREACH (comodo per liste)

```
FOREACH ( loopvariable )
...
ENDFOREACH ( loopvariable )
```

- Ciclo WHILE

```
WHILE ( condition )
...
ENDWHILE ( condition )
```

Esempio di foreach e wildcards su files

Il comando `file` è una chiamata al sistema

Esempio

```
file(GLOB mytestfiles "test*.cpp")
foreach(testfile ${mytestfiles})
  message(STATUS "This is a test file ${testfile}")
endforeach(testfile ${mytestfiles })
```

Switch condizionali di sistema

```
IF ( MSVC )
ENDIF ( MSVC )

IF (WIN32)
ENDIF (WIN32)

IF ( UNIX )
ENDIF (UNIX)

IF (APPLE)
ENDIF (APPLE)
```

- Tutte le variabili sono scoped nel singolo CMakeLists.txt
- Non esiste un costrutto **switch**

Espressioni regolari

Regexp

```
STRING( REGEX MATCH ... )
STRING (REGEX MATCHALL ... )
STRING(REGEX REPLACE ... )
```

Esempio:

Esempio regex

```
SET(test "hello world ! catch: me if you can")
STRING(REGEX REPLACE ".*catch: ([^ ]+).*" "\\1"
  result "${test}" )
MESSAGE(STATUS "result= ${result}")
```

stampa a stdout:

Cmake cache

- CMake salva le variabili non variate in un file CMakeCache.txt
- Veloce su Unix, lento su Windows (MSVC)
- Utile ripulire la cache

  `make rebuild_cache}`
- Dalla GUI, menu apposito `Reload cache`
- `rm CMakeCache.txt`, oppure
- Cancellare manualmente il file `CMakeCache.txt`
- Rigenerare il progetto

Gestione di Debug e Release

- Una variabile definisce il tipo di build

Gestione di Debug e Release

- Una variabile definisce il tipo di build
- `SET(CMAKE_BUILD_TYPE XXX)`
  - `Debug`
  - `Release`
  - `RelWithDebInfo`
  - `MinSizeRel`
  - `Profile`

Gestione di Debug e Release

- Una variabile definisce il tipo di build
- `SET(CMAKE_BUILD_TYPE XXX)`
  - `Debug`
  - `Release`
  - `RelWithDebInfo`
  - `MinSizeRel`
  - `Profile`
- oppure da linea di comando:

```
cmake -DCMAKE_BUILD_TYPE=Debug .
```

Gestione di Debug e Release

- Una variabile definisce il tipo di build
- `SET(CMAKE_BUILD_TYPE XXX)`
  - `Debug`
  - `Release`
  - `RelWithDebInfo`
  - `MinSizeRel`
  - `Profile`
- oppure da linea di comando:

  ```
  cmake -DCMAKE_BUILD_TYPE=Debug .
  ```
- `Debug` $\rightarrow$ gdb+valgrind, grosse dimensioni

Gestione di Debug e Release

- Una variabile definisce il tipo di build
- `SET(CMAKE_BUILD_TYPE XXX)`
  - `Debug`
  - `Release`
  - `RelWithDebInfo`
  - `MinSizeRel`
  - `Profile`
- oppure da linea di comando:

  ```
  cmake -DCMAKE_BUILD_TYPE=Debug .
  ```

- `Debug` → gdb+valgrind, grosse dimensioni
- `Profile` utile quando accoppiata con gprof/KCacheGrind

Gestione di Debug e Release

- Una variabile definisce il tipo di build
- `SET(CMAKE_BUILD_TYPE XXX)`
  - `Debug`
  - `Release`
  - `RelWithDebInfo`
  - `MinSizeRel`
  - `Profile`
- oppure da linea di comando:

  ```
  cmake -DCMAKE_BUILD_TYPE=Debug .
  ```
- `Debug` → gdb+valgrind, grosse dimensioni
- `Profile` utile quando accoppiata con gprof/KCacheGrind
- Deploy si effettua sempre in `Release`

Informazioni revisione compile-time

Viene in aiuto il modulo FindSubversion (ripulire cache prima)

Working copy informations

```
FIND_PACKAGE(Subversion)
IF(SUBVERSION_FOUND)
Subversion_WC_INFO(${PROJECT_SOURCE_DIR} Project)
MESSAGE("Current revision is ${Project_WC_REVISION}")
Subversion_WC_LOG(${PROJECT_SOURCE_DIR} Project)
MESSAGE("Last changed log is ${Project_LAST_CHANGED_LOG}")
ENDIF(SUBVERSION_FOUND)
```

Usare ADD_DEFINITIONS:

Generare la documentazione

Aggiungere un target doc ed usare FindDoxygen.cmake

Esempio

```
find_package(Doxygen)
if(DOXYGEN_FOUND)
configure_file(${CMAKE_CURRENT_SOURCE_DIR}/Doxyfile
    ${CMAKE_CURRENT_BINARY_DIR}/Doxyfile @ONLY)
# Aggiunge un target doc al Makefile
add_custom_target(doc
${DOXYGEN_EXECUTABLE} ${CMAKE_CURRENT_BINARY_DIR}/Doxyfile
WORKING_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}
COMMENT "Generating API documentation with Doxygen"
    VERBATIM )
endif(DOXYGEN_FOUND)
```

Vedi http://bit.ly/9eel8b

HelloWorld eseguibile

Hello world!

```
#Creiamo il nome del progetto
PROJECT( helloworld )
# Impostiamo la variabile hello_SRCS a contenere la hello.cpp
SET( hello_SRCS hello.cpp )
# Crea l'eseguibile di nome hello dal file contenuto
# nella variabile hello_SRCS
ADD_EXECUTABLE( hello ${hello_SRCS} )
```

Esempio 01-helloworld

Creazione librerie statica/dinamica

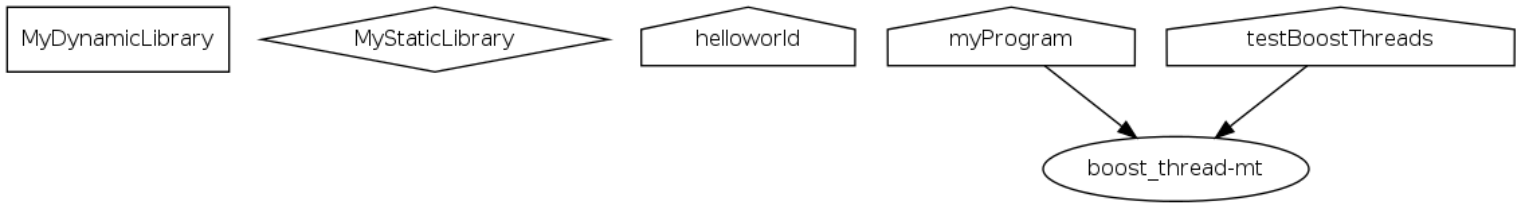
```
#Creiamo il nome del progetto
PROJECT( mylibrary )
# Impostiamo la variabile mylibrary_SRCS a contenere tutti i
# file che definiscono la libreria
SET( mylibrary_SRCS Foo.cpp Bar.cpp Qux.cpp)
# Crea una libreria STATICA (di default in CMake)
# in Linux con gcc genera un file libMyLibrary.a
ADD_LIBRARY( MyLibrary ${mylibrary_SRCS} )
# in Linux con gcc genera un file libMyLibrary.so
ADD_LIBRARY( myLibrary ${mylibrary_SRCS} )
```

Esempi 02-staticlib, 03-dynamiclib

Generazione grafo delle dipendenze

CMake supporta la generazione visual del grafo delle dipendenze fra files, sfruttando il pacchetto di graph-drawing **Graphviz**

```
cmake --graphviz=dependencies.dot .
dot -Tpng dependencies.dot > dependencies.png
```



Boost

Boost è una libreria estensione del C++:

- "...one of the most highly regarded and expertly designed C++ library projects in the world." — Herb Sutter and Andrei Alexandrescu, C++ Coding Standards
- "Item 55: Familiarize yourself with Boost." — Scott Meyers, Effective C++, 3rd Ed.

Boost contiene supporto headers-only e anche librerie bimap, containers generici, interfacce IO, socket, **threading** e molto altro ancora.

CMake + Boost

Nel CMakeLists.txt di base

```
# Set the needed boost libraries
set(BOOST_LIBS thread date_time system program_options
  filesystem regex serialization iostreams)
set(Boost_USE_STATIC_LIBS        ON)
set(Boost_USE_MULTITHREADED      ON)
set(Boost_USE_STATIC_RUNTIME    OFF)

find_package(Boost COMPONENTS ${BOOST_LIBS} REQUIRED)
include_directories(${Boost_INCLUDE_DIR})
find_package(Threads REQUIRED)
```

CMake + Boost

Linkare una libreria a Boost

```
add_library(FooBar Foo.cpp Bar.cpp)
target_link_libraries(FooBar ${BOOST_LIBRARIES})
```

Linkare un eseguibile a Boost:

```
add_executable(myApplication myApplication.cpp
    Foo.cpp Bar.cpp)
target_link_libraries(myApplication ${BOOST_LIBRARIES})
```

Viene **automagicamente** selezionata la versione della libreria corretta:

- **build statica multithread** `/usr/lib/libboostthread-mt.a`
- **build dinamica non-multithread** `/usr/lib/libboostthread.so`

Supporto a OpenMP

```
include(FindOpenMP)
if(OPENMP_FOUND)
   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}
         ${OpenMP_CXX_FLAGS}")
   set(CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS}
         ${OpenMP_EXE_LINKER_FLAGS}")
endif()
```

- Sceglie il flag appropriato del compilatore (`-fopenmp` su g++ , -openmp su Intel icc, /openmp su MSVC)
- Problemi con `vcompd.dll`, modificare il file manifest con MSVC9 (2008)[7]
- `http://public.kitware.com/Bug/view.php?id=12964`

[7] `http://kitware.com/blog/home/post/4`

CMake e Qt

CMake interagisce molto bene con Qt (e viceversa)

```
set(QT_MIN_VERSION "4.6.0")
set(QT_USE_QTMAIN TRUE)
set(QT_USE_OPENGL TRUE)
find_package(Qt4 4.6.0 COMPONENTS QtGui QtCore
    QtOpenGL REQUIRED )
INCLUDE(${QT_USE_FILE})
```

CMake e Qt

Ingredienti

- Cercare Qt4 nel sistema

```
Find_Package(Qt4 REQUIRED)
INCLUDE( ${QT_USE_FILE} )
```

CMake e Qt

Ingredienti

- Cercare Qt4 nel sistema

```
Find_Package(Qt4 REQUIRED)
INCLUDE( ${QT_USE_FILE} )
```

- Indicare i sorgenti .cpp ed i forms .ui

```
QT4_WRAP_CPP, QT4_WRAP_UI
```

CMake e Qt

Ingredienti

- Cercare Qt4 nel sistema

```
Find_Package(Qt4 REQUIRED)
INCLUDE( ${QT_USE_FILE} )
```

- Indicare i sorgenti .cpp ed i forms .ui

```
QT4_WRAP_CPP, QT4_WRAP_UI
```

- Indicare il file risorsa (icone, suoni, immagini etc)

```
QT4_ADD_RESOURCES
```

CMake e Qt

Ingredienti

- Cercare Qt4 nel sistema

```
Find_Package(Qt4 REQUIRED)
INCLUDE( ${QT_USE_FILE} )
```

- Indicare i sorgenti .cpp ed i forms .ui

```
QT4_WRAP_CPP, QT4_WRAP_UI
```

- Indicare il file risorsa (icone, suoni, immagini etc)

```
QT4_ADD_RESOURCES
```

- Linkare il tutto ed includere le directory di Qt

```
link_libraries(${QT_QTCORE_LIBRARY} ${QT_QTGUI_LIBRARY} )
include_directories(${QT_INCLUDE_PATH}
  ${QT_QTGUI_INCLUDE_DIR}
  ${QT_QTCORE_INCLUDE_DIR} )
```

Semplice progettino con Qt

Esempio di Qt

```
PROJECT( pfrac )
FIND_PACKAGE( Qt4 REQUIRED )
INCLUDE( ${QT_USE_FILE} )
SET( pfrac_SRCS main.cpp client.h client.cpp )
SET( pfrac_MOC_HEADERS client.h )
QT4_ADD_RESOURCES( pfrac_SRCS
     ${PROJECT_SOURCE_DIR}/pfrac.qrc )
QT4_WRAP_CPP( pfrac_MOC_SRCS
     ${pfrac_MOC_HEADERS} )
ADD_EXECUTABLE( pfrac ${pfrac_SRCS} $
{pfrac_MOC_SRCS}
TARGET_LINK_LIBRARIES( pfrac ${QT_LIBRARIES} )
```

CMake + OpenGL

Esistono diversi modi di supportare OpenGL con CMake, sono tutti cross-platform

- FindOpenGL.cmake
- FindGlut.cmake

Su Linux è molto facile (previa installazione di GLU/GLUT/FreeGLUT)

Librerie OpenGL

```
find_package(OpenGL REQUIRED)
include_directories(${OPENGL_INCLUDE_DIR})
find_package(GLUT REQUIRED)
set(GL_LIBS ${OPENGL_LIBRARIES} ${GLUT_LIBRARIES})
...
target_link_libraries(myApp ${GL_LIBS})
```

Scrivere il proprio FindXXX.cmake

E' possibile scrivere il proprio FindXXX.cmake per includere e linkare una libreria

- WIN32 XXX.lib
- UNIX libXXX.a

Esempio: FindGLEW.cmake

```
FIND_PATH( GLEW_INCLUDE_DIR glew.h wglew.h
NAMES gl/glew.h GL/glew.h
    PATHS /usr/local/include /opt/local/include /usr/include
    NO_DEFAULT_PATH NO_CMAKE_ENVIRONMENT_PATH
    NO_CMAKE_PATH NO_SYSTEM_ENVIRONMENT_PATH
    NO_CMAKE_SYSTEM_PATH
)

FIND_LIBRARY( GLEW_LIBRARY
NAMES "GLEW glew"
    PATHS /usr/lib /usr/local/lib /opt/local/lib
    NO_DEFAULT_PATH NO_CMAKE_ENVIRONMENT_PATH
```

CPack

CPack è il progetto cugino di CMake con cui è strettamente integrato e permette di creare pacchi:

- Linux generici (.sh, .tgz)
- Distro-based (.deb, .rpm,)
- OSX (creazione .app o dischi .dmg)
- Windows (creazione setup.exe grazie a NSIS installer e 7Zip)
- Pacchi architettura-specifici
- Cross compilazione!

CPack

Creazione di un installer per una libreria **MyLib** [8]

```
install(TARGETS MyLib
  ARCHIVE
  DESTINATION lib)

install(TARGETS MyLibApp
  RUNTIME
  DESTINATION bin)

install(FILES MyLibrary.h
  DESTINATION include)
```

[8] http://cmake.org/Wiki/CMake:Component_Install_With_CPack

Installer per la libreria MyLib

```
set(CPACK_PACKAGE_NAME "MyLib")
set(CPACK_PACKAGE_VENDOR "CMake.org")
set(CPACK_PACKAGE_DESCRIPTION_SUMMARY "Esempio CPack")
set(CPACK_PACKAGE_VERSION "1.0.0")
set(CPACK_PACKAGE_VERSION_MAJOR "1")
set(CPACK_PACKAGE_VERSION_MINOR "0")
set(CPACK_PACKAGE_VERSION_PATCH "0")
set(CPACK_PACKAGE_INSTALL_DIRECTORY "CPack Example")

# Ultimo comando da includere nel CMakeLists.txt
include(CPack)
```

Hands on CMake/CPack

Ho preparato alcune risorse per potere meglio capire i concetti

Pacchetti necessari

```
sudo apt-get git install build-essentials cmake
libbboost-dev
```

GitHub repository

```
git clone
https://github.com/CarloNicolini/cmake-slides.git
```

Risorse online

Mailing list ufficiale degli utenti di CMake

`http://www.cmake.org/mailman/listinfo/cmake`

Sito di domande e risposte, frequentato da molti utenti di CMake

`www.stackoverflow.com`

CMake tutorial online

`http://www.cmake.org/cmake/help/cmake_tutorial.html`

Il libro ufficiale, insostituibile

`http://www.kitware.com/products/books/CMakeBook.html`

Domande ?

Carlo Nicolini
nicolini.carlo@gmail.com