Build facili con CMake

Il sistema di build -definitivo-

Carlo Nicolini

November 26, 2012

Outline

Outline

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

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

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

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

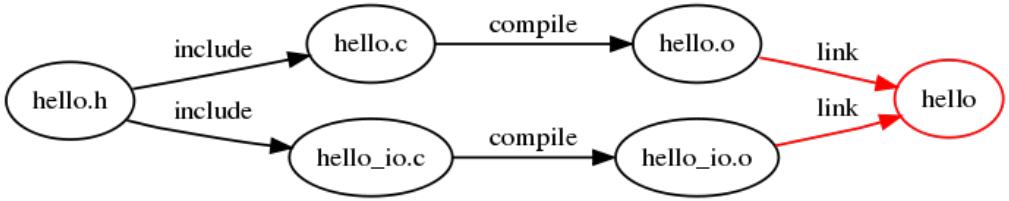
Motivazioni

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

Il passato

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



Problema delle dipendenze è un problema su grafo DAG. Il grafo direzionato informa il sistema di build cosa ricompilare se un file cambia. Se `hello.c` cambia allora tutti i nodi da lì in poi devono essere ricompilati Un buon sistema di build affronta e risolve i seguenti problemi:

- Soluzione delle dipendenze
- Build parallele
- Analisi dei punti di articolazione (limite al parallelismo)

[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*
- 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]
- eccetera...

In genere, poco supporto, piccola comunità, tanti bachi.

[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]
- eccetera...

In genere, poco supporto, piccola comunità, tanti bachi.

[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]
- eccetera...

In genere, poco supporto, piccola comunità, tanti bachi.

[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]
- eccetera...

In genere, poco supporto, piccola comunità, tanti bachi.

[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]
- eccetera...

In genere, poco supporto, piccola comunità, tanti bachi.

[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)
- 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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

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.
- Build multiprocessore
- **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`

CMake tree e primi passi

Supponiamo di avere questo tree:

- **src**
  - myapp.cpp
  - myapp.h
  - CMakeLists.txt
- **build**
- CMakeLists.txt

Si fa la build con:

```
cmake .
make
```

CMake tree e primi passi

Supponiamo di avere questo tree:

- **src**
  - myapp.cpp
  - myapp.h
  - CMakeLists.txt
- **build**
- CMakeLists.txt

Si fa la build con:

```
cmake .
make
```

CMake tree e primi passi

Supponiamo di avere questo tree:

- **src**
  - myapp.cpp
  - myapp.h
  - CMakeLists.txt
- **build**
- CMakeLists.txt

Si fa la build con:

```
cmake .
make
```

Linguaggio - Variabili

- Non serve dichiararle (stringa vuota se non esistono)
- Atipizzate
- SET crea e modifica variabili
- SET si affianca a LIST
- SEPARATE_ARGUMENTS spezza argomenti separati da spazio in una LIST
- In Cmake 2.4: globali (name clashing problems) In Cmake 2.6: scoped
- CMake corrente 2.9

Linguaggio - Variabili

- Non serve dichiararle (stringa vuota se non esistono)
- Atipizzate
- SET crea e modifica variabili
- SET si affianca a LIST
- SEPARATE_ARGUMENTS spezza argomenti separati da spazio in una LIST
- In Cmake 2.4: globali (name clashing problems) In Cmake 2.6: scoped
- CMake corrente 2.9

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

```
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

Complicate ma possibili

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

Esempio:

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

stampa a stdout:

```
-- result= me
```

(il “–” è stampato ad ogni riga di default)

Espressioni regolari

How to convert a semicolon separated list to a whitespace separated string?

```
set(foo abc.c abc.b abc.a)

foreach(arg ${foo})
    set(bar "${bar} ${arg}")
endforeach(arg ${foo})

message("foo: ${foo}")
message("bar: ${bar}")
```

Compatibilità con versioni precedenti

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

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

Cmake cache

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

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
- Si rilascia il pacchetto sempre in `Release`
- `Profile` utile quando accoppiata con gprof/KCacheGrind

Versionamento compile-time

- Utilizzare il modulo `UseSubversion` `http://bit.ly/WjwiRP` (ripulire il repo prima)
- Vengono definite alcune variabili utili:
  - `SUBVERSION_REPO_REVISION`
    `SUBVERSION_REPO_LAST_CHANGED_DATE`
    `etcetera`
  - Passabili al compilatore come flags:

    ```
    ADD_DEFINITIONS(-DREV_NUMBER=
    "${SUBVERSION_REPO_REVISION}")
    ```
- Nel codice C/C++ quindi:

```
int revision=REV_NUMBER;
printf("La versione corrente è %d\n", revision);
```

Generare la documentazione

Aggiungere un target doc ed usare FindDoxygen.cmake

```
# Aggiunge un target al Makefile per generare
# la doc usando Doxygen
find_package(Doxygen)
if(DOXYGEN_FOUND)
configure_file(${CMAKE_CURRENT_SOURCE_DIR}/Doxyfile
    ${CMAKE_CURRENT_BINARY_DIR}/Doxyfile @ONLY)
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)
```

6

[6] http://bit.ly/9eel8b

Outline

HelloWorld eseguibile

```
#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 varia
ADD\_EXECUTABLE( hello \$\{hello\_SRCS\} )
```

- Tutte le variabili sono **stringhe**
- Le variabili si dereferenziano bash-style

  `${NOMEVARIABILE}`

- Le variabili si impostano con SET

Creazione libreria statica .a

#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) a partire dai sorgenti
in Linux con gcc genera un file *lib*myLibrary.a
ADD_LIBRARY(myLibrary ${mylibrary_SRCS})
Oppure crea una libreria SHARED (o DINAMICA in Windows) a partire dai sorgenti
in Linux con gcc genera un file *lib*myLibrary.so **ADD_LIBRARY(myLibrary SHARED ${mylibrary_SRCS})**

Outline

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 (g++ -fopenmp, icc -openmp, MSVC /OMP)
- 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

```
set(MyApp_HDR  Foo.h)
# MyApp source files
set(MyApp_SRC main.cpp mainwindow.cpp Foo.cpp )
# User interface files
set(MyApp_FORMS mainwindow.ui )
set(MyApp_RESOURCES "myapp_images.qrc")

QT4_ADD_RESOURCES( MyApp_RESOURCES_SOURCES
    ${MyApp_RESOURCES} )
QT4_WRAP_UI( MyApp_FORMS_HEADERS ${MyApp_FORMS} )
QT4_WRAP_CPP( MyApp_HEADERS_MOC ${MyApp_HDR} )

add_executable(MyApp WIN32 ${MyApp_SRC} ${MyApp_HDR}
    ${MyApp_HEADERS_MOC} ${MyApp_RESOURCES_SOURCES}
    ${MyApp_FORMS_HEADERS} ${MyApp_RCS})
```

Semplice progettino con 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} )
```

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.
- "The obvious solution for most programmers is to use a library that provides an elegant and efficient platform independent to needed services. Examples are BOOST..." — Bjarne Stroustrup, Abstraction, libraries, and efficiency in C++

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

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(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`

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

```
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})
```

Chiamate al sistema

Chiamate al sistema - Esempio

```
include( UseCython )
add_custom_target( ReplicatePythonSourceTree ALL ${CMAKE_COMMAND
  ${CMAKE_CURRENT_SOURCE_DIR}/cmake/ReplicatePythonSourceTree.cm
  ${CMAKE_CURRENT_BINARY_DIR}
  WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR} )
```

Outline

Highlighting

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!

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 (in biblioteca di ingegneria a Mesiano)
  `http://www.kitware.com/products/books/CMakeBook.html`
- Slides fatte con il tema beamer offerto da KDE
  `http://www.kde.org/kdeslides/`

Domande ?

Carlo Nicolini
nicolini.carlo@gmail.com