Learning CMake

Pau Garcia i Quiles <pgquiles@elpauer.org>

Arisnova Ingeniería de Sistemas
<arisnova@arisnova.com>

The Kitware build and test chain

- Cmake
- CPack
- Ctest + BullsEye/gcov
- CDash

Part I
Meeting CMake

What is CMake

- Think of it as a meta-Make
- CMake is used to control the software compilation process using simple platform and compiler independent configuration files
- CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice: gcc, Visual C++, MingW, Cygwin, Eclipse, Borland, etc
- Projects are described in CMakeLists.txt files (usually one per subdir)

In-tree vs out-of-tree

- Where to place object files, executables and libraries?
- In-tree:
  - helloapp/hello.cpp
  - helloapp/hello.exe
- Out-of-tree:
  - helloapp/hello.cpp
  - helloapp-build/hello.exe
- CMake prefers out-of-tree builds

The CMake workflow

- Have this tree:
  - `myapp`
    `build`
    `trunk`
- `cd myapp/build`
- `cmake ../trunk`
- `make` (Unix) or open project (VC++)
- On Windows, you can also use CMakeSetup (GUI). A multiplatform Qt version is in development (3rd party)

Very simple executable

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

- PROJECT is not mandatory but you should use it
- ADD_EXECUTABLE creates an executable from the listed sources
- Tip: add sources to a list (hello_SRCS), do not list them in ADD_EXECUTABLE

Showing verbose info

- To see the command line CMake produces
- SET(CMAKE_VERBOSE_MAKEFILE on)
- Tip: only use it if your build is failing and you need to find out why

Very simple library

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

- ADD_LIBRARY creates an static library from the listed sources
- Add SHARED to generate shared libraries (Unix) or dynamic libraries (Windows)

Shared vs static libs

- Static libraries: on linking, add the used code to your executable
- Shared/Dynamic libraries: on linking, tell the executable where to find some code it needs
- If you build shared libs in C++, you should also use soversioning to state binary compatibility (too long to be discussed here)

The CMake cache

- Cmake is very fast on Unix but noticeably slow on Windows
- The Cmake cache stores values which are not usually changed
- Edit the cache using ccmake (Unix) or CMakeSetup (Windows)

Regular expressions

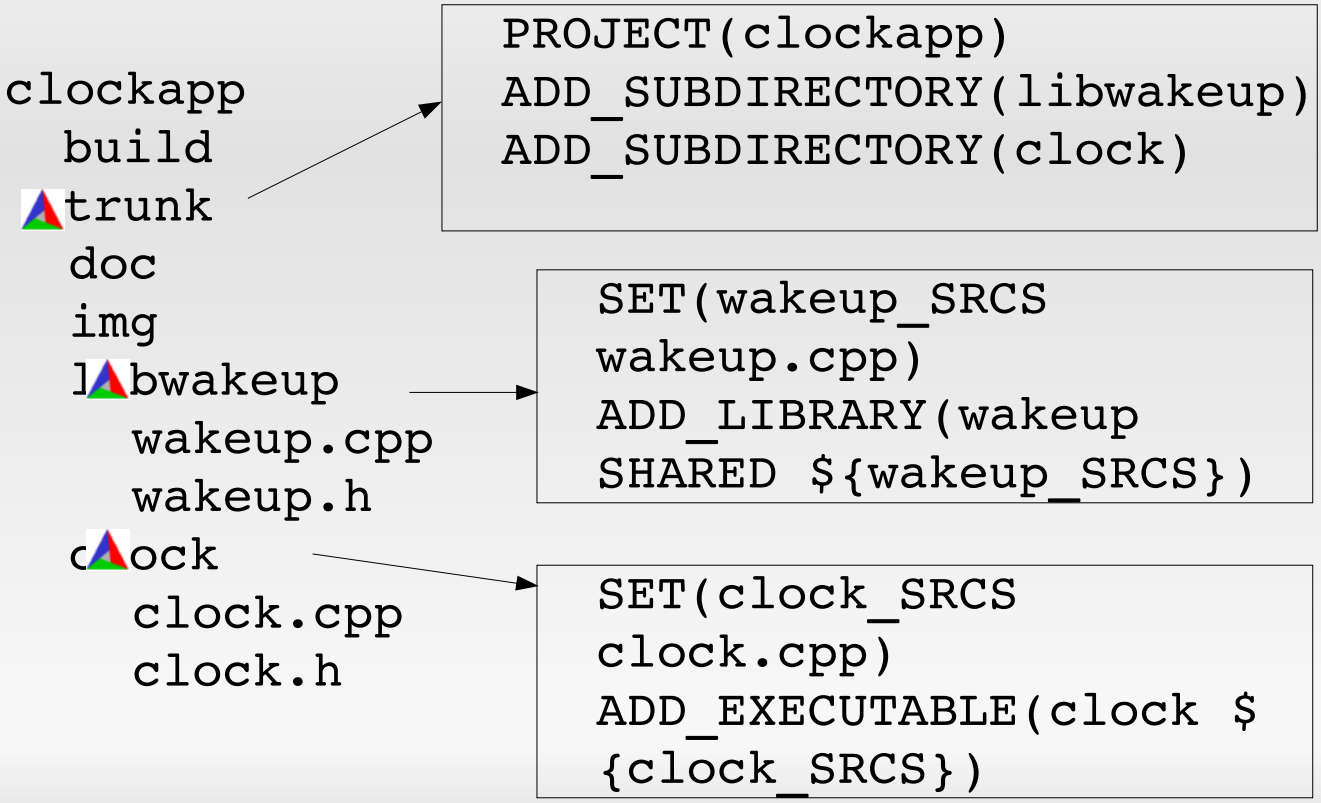
- Worst side of Cmake: they are non-PCRE
- Use STRING(REGEX MATCH ...), STRING (REGEX MATCHALL ...), STRING(REGEX REPLACE ...)
- You will need to try once and again until you find the right regex
- I'm implementing STRING(PCRE_REGEX MATCH ...), etc based on PCRE. Not sure if it will be on time for Cmake 2.6.0 – It won't be

Back/Forward compatibility

- Since Cmake 2.0, ask for at least a certain version with CMAKE_MINIMUM_REQUIRED
- Since Cmake 2.6, tell Cmake to behave bug-by-bug like a certain version with CMAKE_POLICY(VERSION major.minor[.patch])

Part II
Real world CMake:
dependencies between targets

Adding other sources



Variables

- No need to declare them
- Usually, no need to specify type
- SET creates and modifies variables
- SET can do everything but LIST makes some operations easier
- Use SEPARATE_ARGUMENTS to split space-separated arguments (i.e. a string) into a list (semicolon-separated)
- In Cmake 2.4: global (name clashing problems)
- In Cmake 2.6: scoped

Changing build parameters

- Cmake uses common, sensible defaults for the preprocessor, compiler and linker
- Modify preprocessor settings with ADD_DEFINITIONS and REMOVE_DEFINITIONS
- Compiler settings: CMAKE_C_FLAGS and CMAKE_CXX_FLAGS variables
- Tip: some internal variables (CMAKE_*) are read-only and must be changed executing a command

Flow control

- `IF(expression)`
  `...`
  `ELSE(expression)`
  `...`
  `ENDIF(expression)`
- Process a list:
  `FOREACH(loop_var)`
  `...`
  `ENDFOREACH(loop_var)`
- `WHILE(condition)`
  `...`
  `ENDWHILE(condition)`

Always repeat the expression/condition
It's possible to avoid that but I won't tell you how

Visual Studio special

- To show .h files in Visual Studio, add them to the list of sources in ADD_EXECUTABLE / ADD_LIBRARY
- ```
  SET(wakeup_SRCS wakeup.cpp)
  IF(WIN32)
    SET(wakeup_SRCS ${wakeup_SRCS}
  wakeup.h)
  ENDIF(WIN32)
  ADD_LIBRARY(wakeup SHARED ${wakeup_SRCS})
  ```
- Use SOURCE_GROUP if all your sources are in the same directory

Managing debug and release builds

- `SET(CMAKE_BUILD_TYPE Debug)`
- As any other variable, it can be set from the command line:
  `cmake -DCMAKE_BUILD_TYPE=Release ../trunk`
- Specify debug and release targets and 3rdparty libs:

```
TARGET_LINK_LIBRARIES(wakeup RELEASE $
{wakeup_SRCS})
TARGET_LINK_LIBRARIES(wakeupd DEBUG $
{wakeup_SRCS})
```
```

Standard directories... not!

- Libraries built in your project (even if in a different CmakeLists.txt) is automatic (in rare occasions: ADD_DEPENDENCIES)
- If the 3rd party library or .h is in a “standard” directory (PATH and/or LD_LIBRARY_PATH) is automatic
- If in a non-standard dir, add that directory to LINK_DIRECTORIES (library) and INCLUDE_DIRECTORIES (headers)

make install

- INSTALL(TARGETS clock wakeup RUNTIME DESTINATION bin LIBRARY DESTINATION lib)
- Would install in /usr/local/bin and /usr/local/lib (Unix) or %PROGRAMFILES%\projectname (Windows)

Part III
Platform checks and external dependencies

Finding installed software

- `FIND_PACKAGE( Qt4 REQUIRED )`
- Cmake includes finders (FindXXXX.cmake) for ~130 software packages, many more available in Internet
- If using a non-CMake FindXXXX.cmake, tell Cmake where to find it by setting the CMAKE_MODULE_PATH variable
- Think of FIND_PACKAGE as an #include

Qt with CMake

```
PROJECT( pfrac )
FIND_PACKAGE( Qt4 REQUIRED )
SET( DESIRED_QT_VERSION GREATER 4.2 )
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} )
```

Platform includes

- `CONFIGURE_FILE(InputFile OutputFile [COPYONLY] [ESCAPE_QUOTES] [@ONLY])`
  - Your source may need to set some options depending on the platform, build type, etc
  - Create a `wakeup.h.cmake` and:
    - `#cmakedefine VAR` will be replaced with `#define VAR` if VAR is true, else with `/* #undef VAR */`
    - `@VAR@` will be replaced with the value of VAR
  - Also useful for .conf files

Platform includes (II)

- CHECK_TYPE_SIZE (needs INCLUDE(CheckTypeSize))
- TEST_BIG_ENDIAN (needs INCLUDE(CheckBigEndian))
- CHECK_INCLUDE_FILES (needs INCLUDE(CheckIncludeFiles))

Platform Includes (III)

CmakeLists.txt

```
...
INCLUDE(CheckIncludeFiles)
CHECK_INCLUDE_FILES (
malloc.h HAVE_MALLOC_H )
...
```

wakeup.cpp

```
#include "wakeup.h"
#include “wakeup2.h”
#ifdef HAVE_MALLOC_H
#include <malloc.h>
#else
#include <stdlib.h>
#endif
void do_something() {
void *buf=malloc(1024);
...
}
```

Part IV
Macros and functions

Macros

- ```
  MACRO( <name> [arg1 [arg2 [arg3 ...]]] )
  COMMAND1(ARGS ...)
  COMMAND2(ARGS ...)
  ...
  ENDMACRO( <name> )
  ```
- They perform text substitution, just like `#define` does in C
- Danger! Variable-name clashing is possible if using too generic names. Hint: prefix your varnames with the macro name: MACRO_VARNAME instead of VARNAME
  ```

Functions

- New in Cmake 2.6
- Real functions (like C), not just text-replace (a-la C preprocessor)
- Advantages:
  - Cmake processes CmakeLists.txt faster
  - Avoid variable-scope trouble (hopefully)

New targets

- Targets defined with ADD_CUSTOM_TARGET are always considered outdated (i. e. rebuilt)
- Two signatures for ADD_CUSTOM_COMMAND:
  - Same as ADD_CUSTOM_TARGET but do not rebuild if not needed
  - Execute a target before build, after build or before link
- For example, you can create GENERATE_DOCUMENTATION

GENERATE_DOCUMENTATION (I)

```
MACRO(GENERATE_DOCUMENTATION DOXYGEN_CONFIG_FILE)
FIND_PACKAGE(Doxygen)
SET(DOXYFILE_FOUND false)
IF(EXISTS ${PROJECT_SOURCE_DIR}/${DOXYGEN_CONFIG_FILE})
    SET(DOXYFILE_FOUND true)
ENDIF(EXISTS ${PROJECT_SOURCE_DIR}/${DOXYGEN_CONFIG_FILE})

IF( DOXYGEN_FOUND )
   IF( DOXYFILE_FOUND )
        # Add target
        ADD_CUSTOM_TARGET( doc ALL ${DOXYGEN_EXECUTABLE} "$
{PROJECT_SOURCE_DIR}/${DOXYGEN_CONFIG_FILE}" )

        # Add .tag file and generated documentation to the list
of files we must erase when distcleaning

        # Read doxygen configuration file
        FILE( READ ${PROJECT_SOURCE_DIR}/${DOXYGEN_CONFIG_FILE}
DOXYFILE_CONTENTS )
        STRING( REGEX REPLACE "\n" ";" DOXYFILE_LINES $
{DOXYFILE_CONTENTS} )
...
```

GENERATE_DOCUMENTATION (II)

```
        # Parse .tag filename and add to list of files to
delete if it exists
        FOREACH( DOXYLINE ${DOXYFILE_CONTENTS} )
            STRING( REGEX REPLACE ".*GENERATE_TAGFILE *= *([^
^\n]+).*" "\\1" DOXYGEN_TAG_FILE ${DOXYLINE} )
        ENDFOREACH( DOXYLINE )
        ADD_TO_DISTCLEAN( ${PROJECT_BINARY_DIR}/$
{DOXYGEN_TAG_FILE} )

        # Parse doxygen output doc dir and add to list of files
to delete if it exists
        FOREACH( DOXYLINE ${DOXYFILE_CONTENTS} )
            STRING( REGEX REPLACE ".*OUTPUT_DIRECTORY *= *([^
^\n]+).*" "\\1" DOXYGEN_DOC_DIR ${DOXYLINE} )
        ENDFOREACH( DOXYLINE )
        ADD_TO_DISTCLEAN( ${PROJECT_BINARY_DIR}/$
{DOXYGEN_DOC_DIR} )
        ADD_TO_DISTCLEAN( ${PROJECT_BINARY_DIR}/$
{DOXYGEN_DOC_DIR}.dir )
...
```

GENERATE_DOCUMENTATION (III)

```
    ELSE( DOXYFILE_FOUND )
        MESSAGE( STATUS "Doxygen configuration file not found -
Documentation will not be generated" )
 ENDIF( DOXYFILE_FOUND )
ELSE(DOXYGEN_FOUND)
        MESSAGE(STATUS "Doxygen not found - Documentation will
not be generated")
ENDIF(DOXYGEN_FOUND)
ENDMACRO(GENERATE_DOCUMENTATION)
```

Calling the outside world

- EXECUTE_PROCESS
- Execute and get output from a command, copy files, remove files, etc
- Cross-platform: avoid calling /bin/sh or cmd.exe if EXECUTE_PROCESS suffices

Part V
Creating your own finders

What is a finder

- When compiling a piece of software which links to third-party libraries, we need to know:
  - Where to find the .h files (`-I` in gcc)
  - Where to find the libraries (.so/.dll/.lib/.dylib/...) (`-L` in gcc)
  - The filenames of the libraries we want to link to (`-l` in gcc)
- That's the basic information a finder needs to return

MESSAGE

- Show status information, warnings or errors

```
MESSAGE( [SEND_ERROR | STATUS | FATAL_ERROR]
                    "message to display" ... )
```

STRING

- Manipulate strings or regular expressions
- Many signatures

Files and Windows registry

- GET_FILENAME_COMPONENT interacts with the outside world
  - Sets a Cmake variable to the value of an environment variable
  - Gets a value from a Windows registry key
  - Gets basename, extension, absolute path for a filename

FILE

- Read from / write to files
- Remove files and directories
- Translate paths between native and Cmake: \ ↔ /

Find libraries

- FIND_LIBRARY and the CMAKE_LIBRARY_PATH variable

Find header files

- FIND_FILE

Find generic files

- FIND_PATH and the CMAKE_INCLUDE_PATH variable

PkgConfig support

- PkgConfig is a helper tool used when compiling applications and libraries
- PkgConfig provides the `-L`, `-l` and `-I` parameters
- If some software package has PkgConfig support, use it: the finder will be easier to develop and less error-prone
- PKGCONFIG(package includedir libdir linkflags cflags) (needs INCLUDE(UsePkgConfig))
- Mostly Unix, available for Win32 but seldomly used

- FIND_PROGRAM

- TRY_COMPILE

- TRY_RUN

Part VI
Properties

- CMAKE_MINIMUM_REQUIRED

- OPTION
-

- GET_CMAKE_PROPERTY

- GET_TARGET_PROPERTY

- SET_TARGET_PROPERTIES

Part VII
Useful variables

- CMAKE_BINARY_DIR/CMAKE_SOURCE_DIR

- CMAKE_CURRENT_BINARY_DIR /CMAKE_CURRENT_SOURCE_DIR

- PROJECT_BINARY_DIR/PROJECT_SOURCE_DIR

- EXECUTABLE_OUTPUT_PATH/LIBRARY_OUTPUT_PATH

- ENV ($ENV{name})

- CMAKE_SKIP_RPATH (important in Debian and Debian-derivatives) (follow http://www.cmake.org/Wiki/CMake_RPATH_handling)

More variables

- Use this snippet to list all variables and their values:

```
get_cmake_property( P VARIABLES )
    foreach( VAR in ${P} )
       message( STATUS
              "  ${VAR}=${${VAR}}" )
    endforeach()
```

Part VIII
CPack

Features

- CPack generates installing packages:
  - RPM, DEB, GZip and Bzip2 distributions of both binaries and source code
  - NSIS installers (for Microsoft Windows)
  - Mac OS X packages (.dmg)
- In Cmake 2.4, .rpm and .deb support works but is not good
- It can be used without Cmake
- If used with Cmake, takes advantage of the INSTALL declarations

Variables in CPack

- There are bundle-specific variables: NSIS needs some vars a ZIP does not need
- Important: set variable values BEFORE you INCLUDE(CPack)

Example

```
INCLUDE(InstallRequiredSystemLibraries)

SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "Alarm clock")
SET(CPACK_PACKAGE_VENDOR "Pau Garcia i Quiles")
SET(CPACK_PACKAGE_DESCRIPTION_FILE
"$CMAKE_CURRENT_SOURCE_DIR}/ReadMe.txt")
SET(CPACK_RESOURCE_FILE_LICENSE
"$CMAKE_CURRENT_SOURCE_DIR}/Copyright.txt")
SET(CPACK_PACKAGE_VERSION_MAJOR "0")
SET(CPACK_PACKAGE_VERSION_MINOR "0")
SET(CPACK_PACKAGE_VERSION_PATCH "1")
SET(CPACK_PACKAGE_INSTALL_DIRECTORY "CMake $
{Cmake_VERSION_MAJOR}.${CMake_VERSION_MINOR}")
...
```

Example (cont.)

```
IF(WIN32 AND NOT UNIX)
SET(CPACK_PACKAGE_ICON "$
{Cmake_SOURCE_DIR}/Utilities/Release\\\\InstallIcon.bmp"
)
SET(CPACK_NSIS_INSTALLED_ICON_NAME
"bin\\\\MyExecutable.exe")
SET(CPACK_NSIS_DISPLAY_NAME "$
{CPACK_PACKAGE_INSTALL_DIRECTORY} My Famous Project")
SET(CPACK_NSIS_HELP_LINK "http:\\\\\\\\elpauer.org")
SET(CPACK_NSIS_URL_INFO_ABOUT
"http:\\\\\\\\elpauer.org")
SET(CPACK_NSIS_CONTACT "pgquiles@elpauer.org")
...

INCLUDE(CPack)
```

Part IX
CTest

Features

- Cross-platform testing system which:
  - Retrieves source from CVS, Subversion or Perforce (git support currently being worked on)
  - Configures and build the project
  - Configures, build and runs a set of predefined runtime tests
  - Sends the results to a Dart/CDash dashboard
- Other tests:
  - code coverage (using BullsEye $$$)
  - memory checking

Example

- Very easy!
  - `ENABLE_TESTING()`
  - `ADD_TEST( testname testexecutable args )`
- Some scripting needed to:
  - Download sources from a VC system (CVS, SVN and Perforce templates available, git in progress)
  - Upload to Dart/CDash dashboard (templates available for HTTP, FTP, SCP and XML-RPC)
- It can be used with non-CMake projects

Part X
CDash

Features

- CDash aggregates, analyzes and displays the results of software testing processes submitted from clients.
- Replaces Dart
- For example, build a piece of software on Linux, Windows, Mac OS X, Solaris and AIX
- Usually, you want two kinds of information:
  - Build results on all platforms
  - Test (Ctest) results on all platforms
- Customizable using XSL

Example