Learning CMake

Pau Garcia i Quiles <pgquiles@elpauer.org>
Arisnova Ingeniería de Sistemas
<arisnova@arisnova.com>

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
- 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

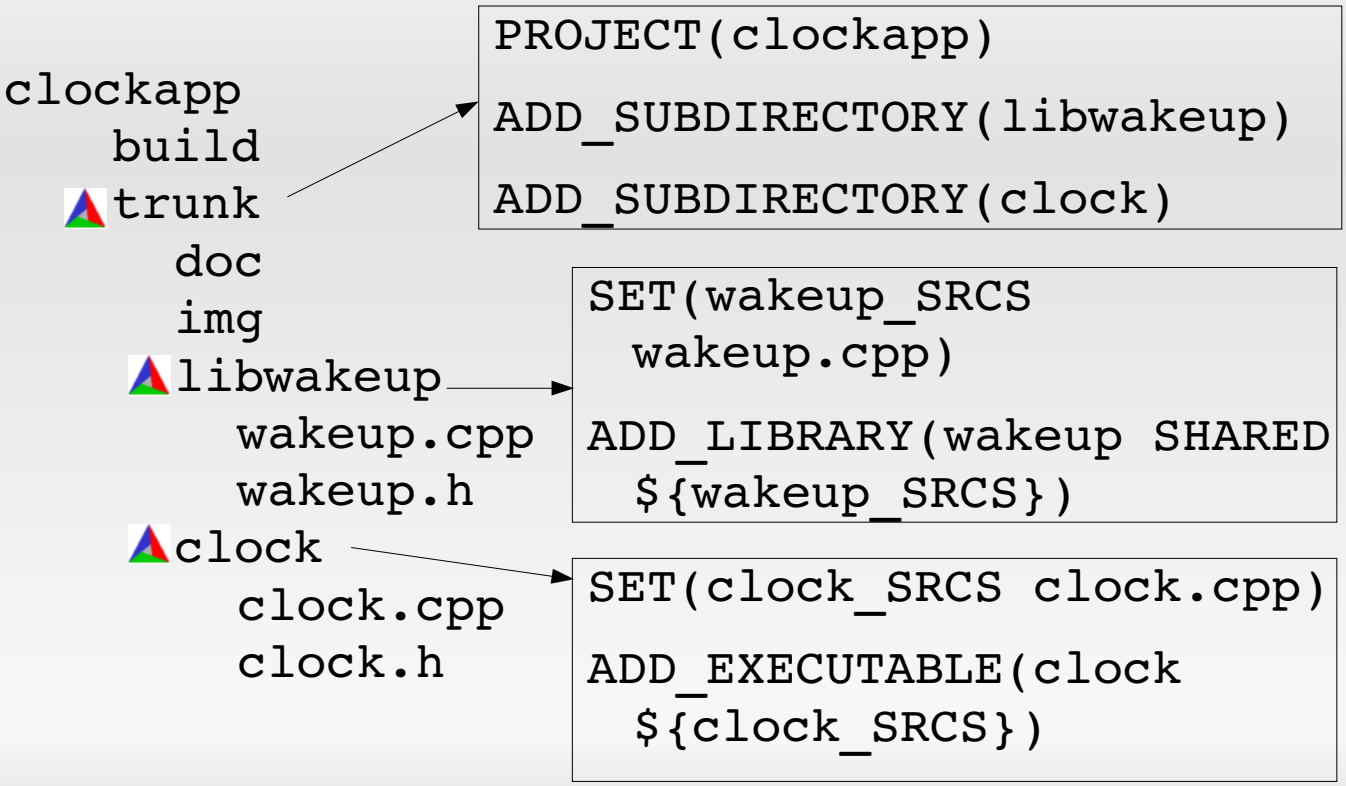
- Worst side of Cmake: they are non-PCRE
- Use STRING(REGEX MATCH ...), STRING (REGEX MATCHALL ...), STRING(REGEX REPLACE ...)
- You will need to try once and again until you find the right regex
- I'm implementing STRING(PCRE_REGEX MATCH ...), etc based on PCRE. Not sure if it will be on time for Cmake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

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)

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 and 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)