CMake build system
Distribute your software easily
Outline

1. Motivations of a build system
2. CMake build system
3. Test integration
4. Packaging
5. Release engineering @ Inria
1 Motivations of a build system
What problems do build system solve?

- For a developer:
  - reduce the time spent in the cycle “edit / compile / test” cycle
  - compile only what is necessary in the source code
- For a development team:
  - generate packages
  - run tests
  - generate documentation
- For a user:
  - install software easily
  - have understandable error during install phase
  - tune installation
Build a software: a lot of evil ways

Examples:

• “I will do a script to launch all my command and it will be ok”
  • system-dependent, all path dependent, etc.
  • high cost for developers and users

• “I will do a makefile with a make.inc, my software earns portability”
  • costly for the user: manual configuration
  • portable ≠ customizable

• Etc.
Features of a build system (1)

• automatic dependency management of source code
  • compile only the modified sources files and thiers dependencies

• software portability:
  • use native build environment
  • determine available OS/compiler features : foo.h, libbar, strndup, -Wall, etc.
  • name correctly the library: .so / .dylib / .dll

• adaptability according user environment:
  • auto-configuration of the project
  • determine the availability and location of libraries, commands, etc...
Features of a build system (2)

• customize installation:
  • cross-compiling
  • give some information: --help
  • possibility to set information: --prefix, --libdir, --disable-shared, etc.
  • have some target: make all, make install…

• launch tests:
  • without installation: link with generated library
  • after an installation: link with installed library
  • give a report of the build
2 CMake build system
Introduction

- Open-source, cross-platform build system (New BSD Licence)
- Develop by Kitware since 2001
- Using compiler-independent method
- Can be used with native build environments (Eclipse, Xcode, Visual Studio…)
- Give some extensions to locate libraries, headers…
- Give some interfaces for generate a test suite and packaging
- Notable applications using CMake: KDE, Blender, LLVM, OGRE
Get and install CMake

- Get and install from web:
  
  http://www.cmake.org/cmake/resources/software.html
  
  >./configure --prefix=<path>
  
  > make
  
  > make install

- Or install form your distribution

- Be careful:
  - about the version of CMake
  - CMake is needed to build and install your software
Manage a project with CMake

- CMakeLists.txt describes the project:
  - list of source files,
  - library to link with…

- CMakeLists.txt is:
  - machine-independent
  - common for all users

- CMakeCache.txt is:
  - generated by calling: cmake <path_to_source>
  - GUI: ccmake or cmake-gui
  - machine-specific
Configuration, build and install step

- Two way to configure the project:
  - In-source
    ```
    > cd <path_to_source>
    > cmake . -DOPTION=<VALUE>
    ```
  - Out-of-source
    ```
    > cd <path_to_build>
    > cmake <path_to_source>
    -DOPTION=<VALUE>
    ```

- Possibility to choose makefile generator during configuration
  ```
  > cmake ../ -G "Unix Makefiles" or -G "Xcode" etc...
  ```

- After configuration, build and install step can be launched
  ```
  > make
  > make install
  ```
Configuration with GUI

- `ccmake <path_to_source>`
- `cmake-gui <path_to_source>`
Build and install step

Some important variables to:

- control the build type:
  
  ```
  CMAKE_BUILD_TYPE=[Debug, Release]
  ```

- control the install directory

  ```
  CMAKE_INSTALL_PREFIX=[/usr/local, home/toto/my_project]
  ```

- activate the verbosity of makefiles

  ```
  CMAKE_VERBOSE_MAKEFILE=ON
  ```

- produce shared or static library

  ```
  CMAKE_SHARED_LIBS=[OFF, ON]
  ```

- etc…
A simple syntax (1)

- Look like script language
  - note
  - variable
  - list
  - Command
- Control structure
  ```cmake
  IF(${VAR})
  ENDIF()
  FOREACH(VAR VAL1 VAL2)
  ENDFOREACH()
  ```
- Dynamic configuration
  ```cmake
  # Describe what I have done
  SET(VAR "toto")
  LIST(KEYWORD list iostream)
  COMMAND(ARG1 ARG2)
  CONFIGURE_FILE(config.h.in config.h)
  #cmakedefine FOO_VER ${FOO_VER}
  #cmakedefine @BUILD_SHARED_LIBS@
  ```
A simple syntax (2)

- Library detection
  - `FIND_LIBRARY(MY_LIB lib PATH path)`
  - `FIND_PACKAGE(CUDA REQUIRED)`

- Feature validation
  - `INCLUDE(CheckCCompilerFlag)
    CHECK_C_COMPILER_FLAG(flag HAVE_FLAG)`
  - `INCLUDE(CheckFunctionExists)
    CHECK_FUNCTION_EXISTS(func HAVE_FUNC)`
  - `INCLUDE(CheckIncludeFile)
    CHECK_INCLUDE_FILE(header HAVE_HEADER)`
  - `INCLUDE(CheckSourceCompiles)
    CHECK_C_SOURCE_COMPILES(code VAR)`
Exercise: helloworld-cmake (1)
Exercise: helloworld-cmake (2)

#include <foo.h>
int main(int ac, char *av[])
{
    print_message();
    return 0;
}

#include <stdio.h>
void print_message(void);

#include <foo.h>
void print_message(void) {
    printf("Hello World!\n");
}

CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
PROJECT(helloworld C)
SET(SRC
    src/main.c
    include/foo.h
    lib/foo.c
)
ADD_EXECUTABLE(test ${SRC})

• The quickest way to compile the project
• Feature test are not here!
• “install” phase not defined…
Exercise: helloworld-cmake (3)

CMAKE_MINIMUM_REQUIRED(VERSION 3.6)
PROJECT(helloworld C)
INCLUDE(CheckIncludeFile)
CHECK_INCLUDE_FILE(stdio.h
    HAVE_STDIO)
IF(NOT HAVE_STDIO)
    MESSAGE(FATAL_ERROR "Looking
    for stdio.h - not found")
ENDIF()
INCLUDE(CheckFunctionExists)
CHECK_FUNCTION_EXISTS(printf
    HAVE_PRINTF)
IF(NOT HAVE_PRINTF)
    MESSAGE(FATAL_ERROR "Looking
    for printf - not found")
ENDIF()
INCLUDE_DIRECTORIES(include)
ADD_SUBDIRECTORY(lib)
ADD_SUBDIRECTORY(src)

CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
ADD_LIBRARY(foo foo.c)
INSTALL(TARGETS foo
    DESTINATION lib)

CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
ADD_EXECUTABLE(my_helloworld main.c)
TARGET_LINK_LIBRARIES(my_helloworld
    foo)
INSTALL(TARGETS foo
    DESTINATION bin)
3

Test integration
About CTest

- CTest comes with CMake
- It can be used without CMake
- It allows to:
  - automate updating from a repository
  - configuration and build
  - execute unit or regression tests
  - execute advanced tests (coverage, purify, valgrind…)
- Results can be submitted to a CDash server
Introduction to CTest

• Modify `CMakeLists.txt` in the top directory:

```
PROJECT(FOO)
INCLUDE(CTest)
INCLUDE_DIRECTORIES(tests)
ENABLE_TESTING()
```

• `tests/CMakeLists.txt` looks like:

```
ADD_EXECUTABLE(example example.cpp)
ADD_TEST(test1 example)
```
Using CTest

- Get the list of tests
  ```
  > ctest -N
  ```

- Launch tests
  ```
  > make test  
  > ctest  
  > ctest -I Start,End,Stride
  ```

- Get log files
  ```
  LastTest.log  
  LastTestsFailed.log
  ```
4

Packaging
about CPack

- CPack comes with CMake
- It can be used without CMake
- It allows to:
  - generate a source distribution
  - generate different binary package
Introduction to CPack without CMake

- Write a file named CPackConfig.cmake or CPackSourceConfig.cmake that looks like:

```cmake
SET(CPACK_GENERATOR                   "TGZ")
SET(CPACK_PACKAGE_NAME                "MY_SOFT")
SET(CPACK_PACKAGE_VERSION_MAJOR       "1")
SET(CPACK_PACKAGE_VERSION_MINOR       "2")
SET(CPACK_PACKAGE_VERSION_PATCH       "0")
SET(CPACK_PACKAGE_DESCRIPTION_FILE    "${SOURCE_DIRECTORY}/COPYRIGHT")
SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "Summary")
SET(CPACK_INSTALLED_DIRECTORIES       "${SOURCE_DIRECTORY};/")
SET(CPACK_INSTALL_CMAKE_PROJECTS      "")
SET(CPACK_PACKAGE_FILE_NAME           "my-soft")
SET(CPACK_PACKAGE_VENDOR             "Inria")
```

- Generate package:

```bash
> cpack -D OPTION=VALUE
```
Introduction to CPack with CMake

- Add in your CMakeLists.txt

```
INCLUDE(InstallRequiredSystemLibraries)
SET(CPACK_GENERATOR "TGZ")
...
SET(CPACK_PACKAGE_VENDOR "Inria")
INCLUDE(Cpack)
```

- Generate package:

```bash
> make && cpack
> make && make package
> make && make package_source
```
5

Release engineering @ Inria
Some platform to help you

- Continuous integration:
  - Hydra: local platform | status: OK
    contact: sed-bordeaux@inria.fr
  - CI@Inria: national platform | status: standby
    contact: sed-lille@inria.fr
  - CDash: national platform | status: OK
    contact: http://cdash.inria.fr/CDash/

- Porting:
  - PIPOL: national platform | status: ON (OFF soon???)
    contact: http://pipol.inria.fr/
6

To conclude
Some conclusions

• About build system
  • manage the relationship: developer(s) / user(s)

• About CMake / CTest / CPack
  • easy-to-develop
  • multi-platform
  • warning: reinventing the wheel, and making it square
Thank you

Inria Bordeaux – Sud-Ouest

http://sed.bordeaux.inria.fr