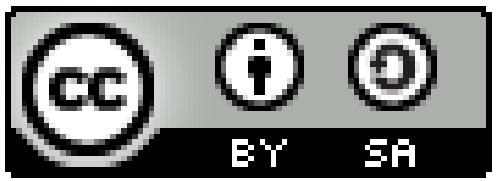
Software build and packaging

Matthieu Herrb

December 2012

`http://homepages.laas.fr/matthieu/cours/build-packaging.pdf`

License



Parts of this document have been re-used and translated from Johan Moreau, « Outils de construction » et de Konrad Hinsen « Packaging en Python » for the ENVOL 2010 CNRS school.

Agenda

1 Introduction

2 Autotools

3 CMake

4 Conclusion

Agenda

1 Introduction

2 Autotools

3 CMake

4 Conclusion

Introduction

Build and packaging:

From a source code:

- provide a set of installable and runnable binaries
- provide packages;
  - source
  - binary

The user's point of view

Users:

- look for simple instructions
  (a `README` or `INSTALL` file)
- likes to follow well-known procedures
  - `configure ; make ; make install`
  - `dpkg -i supersoft.deb`
- hates having 3 tons of dependencies to manually install before being able to test a piece of software.

Complex task

Often the first contact with a new software

- must be robust
- variety of environments
- variety of users
- must fit the user's expectations
- depends on the kind of targeted audience
- depends on technical but also cultural factors
- issues caused by the software license

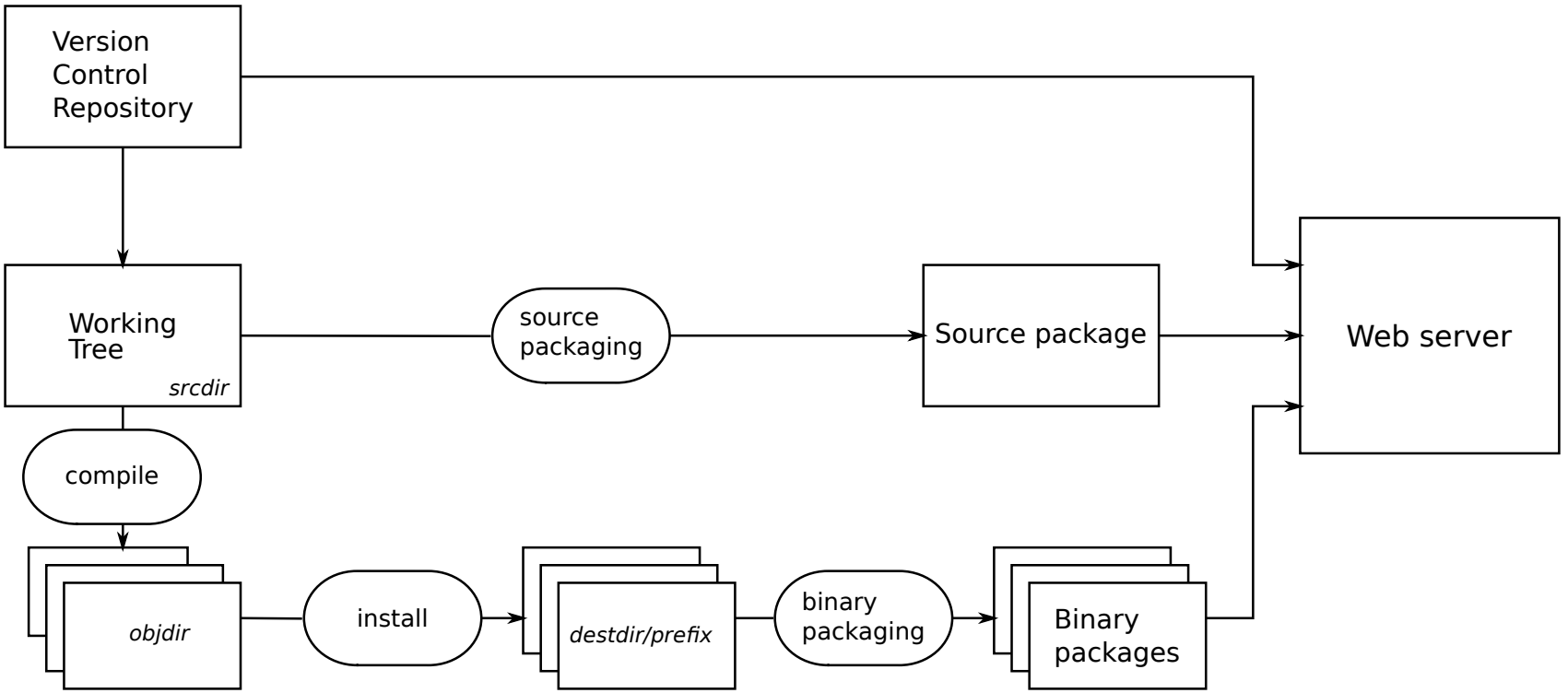
$\rightarrow$ important aspect should not be neglected.

General framework

Automate:

- Configuration
- Compilation, generation of auxiliary files (documentation,...)
- Installation
- Un-installation

General framework



Continuous Integration

Goal: continuously test the whole project:

- configure
- compile
- install
- run unit tests
- un-install

Use the same tools.
Allows to produce *daily snapshots* of the development.

Interfaces, version numbers, release, etc.

Release: *marketing* notion: name, number, logo, event,...

Needs of developers and end-users:

- identify important changes and incompatibilities
- in the case of a library:
  - API version (major.minor) corresponds to binary compatibility with previous versions
- other cases: document behavior, file format changes, provide migration procedures...

Manage a release planning and attempt to follow it.

Configuration phase

Adapt the software to a platform

2 modes:

- pre-selection from a knowledge base
- automatic detection of the platform's features

Always prefer tests on features

example: `#ifdef HAVE_STRLCPY`

produces:

- header files
- Makefiles

Compilation phase

3 steps:

- compiler: source $\rightarrow$ assembly language conversion,
- assembler: $\rightarrow$ machine code,
- link editor: solves references to external libraries.

Library:

- compile individual components
- bind them into the resulting library (static or shared)

objdir or **builddir**
separate directory where build files are stored.

Managing compilation

Quasi-universal tool: **make**
Production rules:

TARGETS : *DEPENDENCIES*
COMMANDS
...

Targets: list of files to build

Dependencies: list of files on which targets depend.

Commands: commands to run to build the targets.

Really powerful, but:

- painful to maintain in large projects
- not well suited to handle configuration (portability)

Tools that produce `Makefiles`

- Automake/autoconf/configure
- CMake
- Ant
- Scons
- ...

Install phase

Copy files to their final destination in the system tree.

- don't skip this phase
- respect conventions of the target system
  - Linux: FHS, LFS
  - Mac OS X: Bundles, or Unix-like
  - Windows: "Program Files"...
- makes it possible to create binary packages
- better integration with other software
- **destdir**: root of the installation file system
- **prefix**: base directory for installation inside *destdir*

Auxiliary operations (optional): create a dedicated user, register installed components,...

Un-installation

Inverse operation of installation

- remove all installed components
- undo auxiliary operations

"Well behaving" packages:
leave the system in the same state it was before they were installed.

Distribution formats

- Source
  - source files,
  - configuration, compilation and installation tools
  - documentation

  Format: TAR or ZIP archive, VCS
- Binary
  - binary executables + auxiliary files
  - installation program
  - documentation
  - meta-data: dependencies, list of installed files,...

  Binary package formats depend on the operating system and on its package manager(s).

Which format to choose ?

- Free and Open Source software:
  - prefer a source distribution
  - let binary packaging to those who know how
- Closed software:
  - limited diffusion: provide sources, simple packaging
  - broad diffusion: prefer standard binary packages for targeted platforms

Get help...

OpenRobots at LAAS...

Packaging tool: `robotpkg`

- source packaging system
- dependencies tracking
- suited both for end-users and developers

See separate course...

Licenses...

A package must

- Clearly display its license and terms of use.
- Respect licenses from third party software it uses and depends on

→ Provide a `COPYING` or `LICENSE` file that groups all applicable licenses.

→ In the case of binary distribution of software under the GPL,
prepare a source package and provide information on how to get it.

Generic tools

- relatively independents from the programming language and the system
- manage configuration, compilation and packaging
- packaging is the last (optional) step.

Agenda

1 Introduction

2 Autotools

3 CMake

4 Conclusion

GNU Autotools

Autotools

References:

- `http://www.gnu.org/software/autoconf/`
- `http://sourceware.org/autobook/`
- `http://www.lrde.epita.fr/adl/autotools.html`

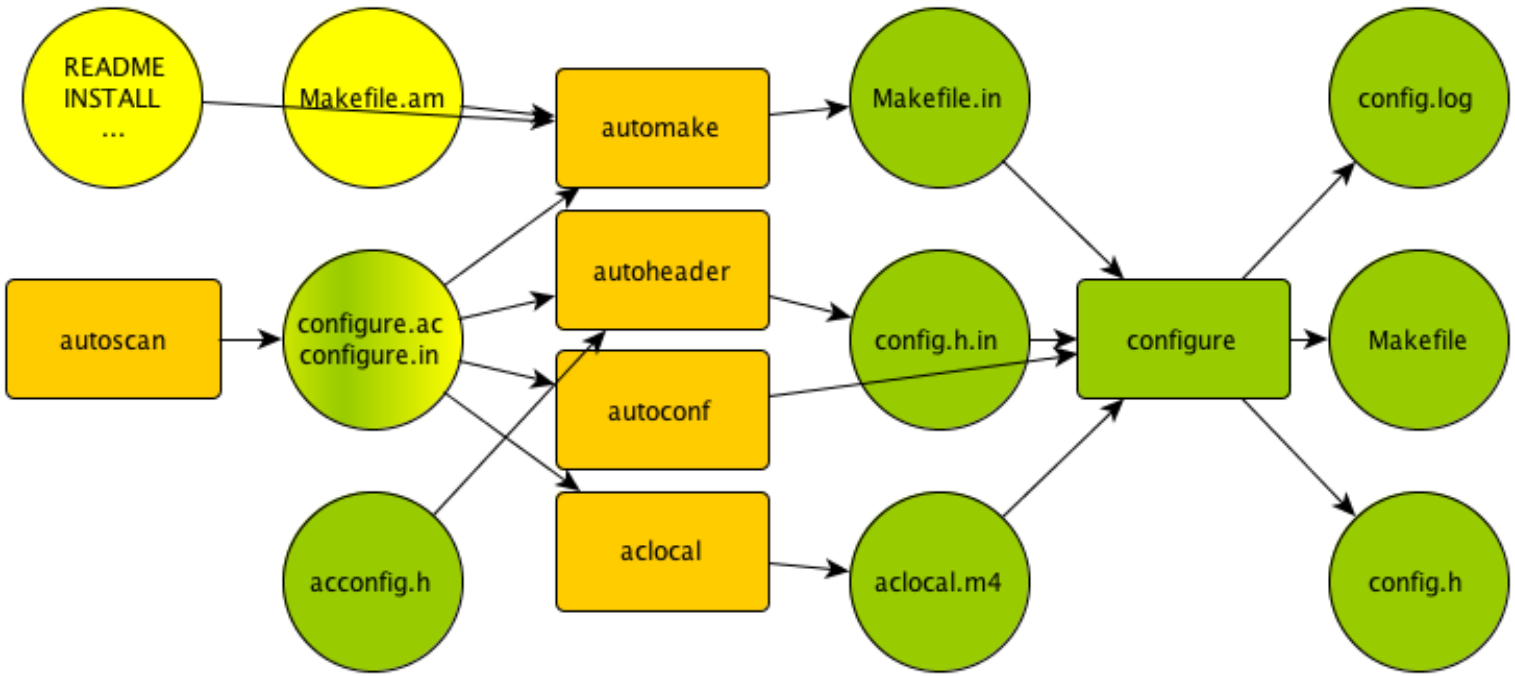
Autotools

```
configure; make; make install
```

Set of tools:

- automake, autoheader, autoconf, libtool,...
- written in **shell**, **m4** macro processor and **perl**
- automated generation of Makefiles
- configuration conforming to target platform
- portables

Autotools - a toolset



Autotools - getting started

- create `configure.ac`
- create `Makefile.am`
- run `autoreconf -force -install`
- run `configure`
- run `make`
- run `make install`

Autotools - Hello World

src/main.c

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  print("Hello, World\n");
  return 0;
}
```

Autotools - files to create

configure.ac

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

Makefile.am

```
SUBDIRS=src
```

src/Makefile.am

```
bin_PROGRAMS = hello
hello_SOURCES = main.c
```

Our configure.ac

configure.ac

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Autoconf initialization (name, version, support @, ...)

Our `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Autoconf initialization (name, version, support @, ...)
- Automake initialization (non-GNU package)

Our `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Autoconf initialization (name, version, support @, ...)
- Automake initialization (non-GNU package)
- Using the C compiler

Our `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Autoconf initialization (name, version, support @, ...)
- Automake initialization (non-GNU package)
- Using the C compiler
- Declare the `config.h` headers

Our `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Autoconf initialization (name, version, support @, ...)
- Automake initialization (non-GNU package)
- Using the C compiler
- Declare the `config.h` headers
- Declare `Makefile` et `src/Makefile`

Our `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Autoconf initialization (name, version, support @, ...)
- Automake initialization (non-GNU package)
- Using the C compiler
- Declare the `config.h` headers
- Declare `Makefile` et `src/Makefile`
- Generate specified files

autotools - sample checks for programs

`AC_PROC_CC`, `AC_PROG_CXX`, `AC_PROC_F77`
checks for compilers

`AC_PROG_SED`, `AC_PROG_YACC`, `AC_PROG_LEX`
Checks for various tools

`AC_CHECK_PROGS(`*`VAR`*`,` *`PROG`*`,` `[`*`VAL-IF-NOT-FOUND`*`])`
`VAR` is set to the pathname for *`PROG`* if found or to *`VAL-IF-NOT-FOUND`* otherwise.

example

```
AC_CHECK_PROGS([TAR], [tar gtar], "not found")
if [ "$TAR" = "not found" ]; then
    AC_MSG_ERROR([This package requires tar.])
fi
```

autotools - useful macros

`AC_MSG_ERROR(`*ERROR-DESCRIPTION*`, [`*EXIT-STATUS*`])`
Displays *ERROR-DESCRIPTION* and stops configure.

`AC_MSG_WARN(`*ERROR-DESCRIPTION*`)`
Same, without stopping

`AC_DEFINE(`*(*VARIABLE`,` *VALUE*`,` *DESCRIPTION*`)`
Adds a definition in `config.h`:

```
/* DESCRIPTION */
#define VARIABLE VALUE
```

`AC_SUBST(`*VARIABLE*`, [`*VALUE*`])`
Sets *VARIABLE* to *VALUE* in `Makefile`

Checks for libraries

`AC_CHECK_LIB(`*`LIBRARY`*`, `*`FUNCT`*`, [`*`ACT-IF-FOUND`*`], [`*`ACT-IF-NOT`*`])`
Checks if *LIBRARY* is present and contains *FUNCT*. If yes executes *ACT-IF-FOUND*, otherwise *ACT-IF-NOT*.

example

```
AC_CHECK_LIB([z], [gzopen64], [LIBZ64=-lz])
AC_SUBST([ZLIB64])
```

allows to use `$(LIBZ64)` in Makefiles.

Checks for header files

`AC_CHECK_HEADERS(`*`HEADERS...`*`)`
Checks for the presence of *HEADERS* and sets `HAVE_HEADER_H`

example

```
AC_CHECK_HEADERS([sys/param.h unistd.h])
AC_CHECK_HEADERS([wcar.h])
```

produces, in `config.h`:

```
#define HAVE_SYS_PARAM_H
#define HAVE_HUNISTD_H
#undef HAVE_WCAR_H
```

One can then write:

```
#ifdef HAVE_UNISTD_H
# include <unistd.h>
#endif
```

Autotools - use of pkg-config

`PKG_CHECK_MODULES(`*BASE, PKG-SPEC...*`)`
checks using **pkg-config** that packages described by *PKG-SPEC* are installed. Sets *BASE_CFLAGS* and *BASE_LIBS* with correct options to use those packages.

example

```
PKG_CHECK_MODULE([GTK], [gtk+-2.0])
```

In `Makefile.am`:

```
AM_CFLAGs = $(GTK_CFLAGS)
AM_LDFLAGS = $(GTK_LIBS)
```

Autotools - Writing Makefile.am

Makefile.am

```
SUBDIRS=src
```

src/Makefile.am

```
bin_PROGRAMS = hello
hello_SOURCES = main.c
```

- SUBDIRS to set sub-directories in which to build
- bin_PROGRAMS : programs to generate
- program_SOURCES : list of source files

Autotools - targets for Makefile.am

Makefile.am

*where*_*TYPE* = target ...

where: installation directory

`bin_` → `$(bindir)`

`lib_` → `$(libdir)`

TYPE: defines the kind of rules used

`_PROGRAMS`

`_LIBRARIES`

`_LTLIBRARIES`

`_HEADERS`

`_SCRIPTS`

`_DATA`

Autotools - building a library

- Add AC_PROG_LIBTOOL in `configure.ac`

`Makefile.am`

```
lib_LTLIBRARIES = libfoo.la
libfoo_la_SOURCES = foo.c foopriv.h
include_HEADERS = foo.h
```

- library installed in `$(libdir)`
- public headers installed in `$(includedir)`
- private headers not installed

Autotools - pkg-config file for a library

Allow other packages to use **pkg-config**

foo.pc.in

```
prefix=@prefix@
exec_prefix=@exec_prefix@
libdir=@libdir@
includedir=@includedir@

Name: foo
Description: Library foo providing foo functions
Version: @PACKAGE_VERSION@
Cflags: -I${includedir}
Libs: -L${libdir} -lfoo
```

Autotools - produce and install the .pc file

configure.ac

```
AC_CONFIG_FILES([Makefile
                 src/Makefile
                 foo.pc])
AC_OUTPUT
```

Makefile.am

```
pkgconfigdir = $(libdir)/pkgconfig
pkgconfig_DATA = foo.pc
```

Autotools - source packaging

`make distcheck`

- produces a source archive in the `tar.gz` format,
- checks that it builds and installs without error.

Autotools - binary packaging

Not really supported, but provides a useful base block:

- `make install DESTDIR=/staging`

generates in `/staging/` all installed files.

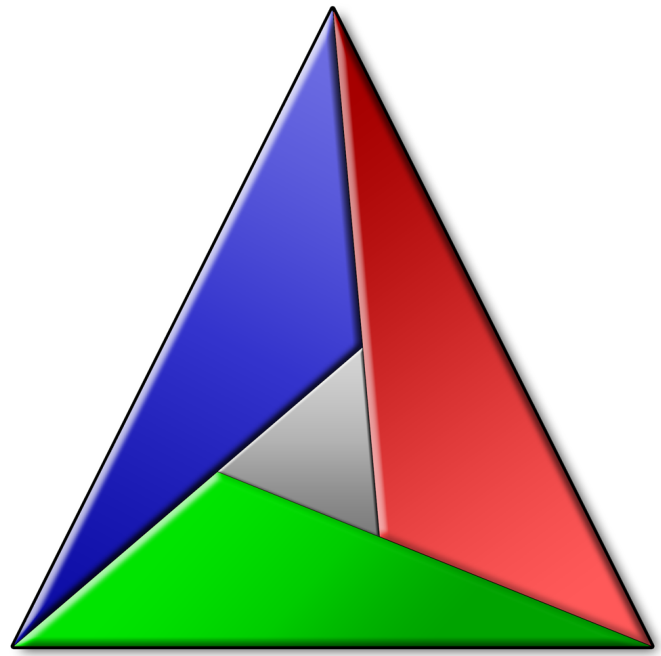
Agenda

1 Introduction

2 Autotools

3 CMake

4 Conclusion



CMake

CMake

References:

- `http://www.cmake.org/cmake/help/documentation.html`
- `http://www.cmake.org/cmake/help/cmake_tutorial.html`
- `http://www.elpauer.org/stuff/learning_cmake.pdf`
- Mastering CMake, Kitware Inc, 5th edition ISBN 978-1-930934-22-1

CMake - presentation

- Written in C++
- Specific syntax, in `CMakeLists.txt` files.
- Multi-platform (Unix, MacOS X, Windows)
- Generates a build system depending on the platform
  - Makefiles on Unix systems
  - Visual C++ projects
  - Eclipse projects
- Additional tools: CDash, CPack, CTest

CMake - principle

- Create a `CMakeLists.txt` file in each directory of the project
- Run the `cmake` command
- Use native tools "`make; make install`" with the generated build system
- Configuration is using a cache mechanism (`CMakeCache.txt`) to store probed informations between runs.

CMake - Hello World

src/main.c

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  print("Hello, World\n");
  return 0;
}
```

CMakeLists.txt

```
PROJECT(hello)
SUBDIRS(src)
```

src/CMakeLists.txt

```
SET(hello_SRCS main.c)
ADD_EXECUTABLE(hello ${hello_SRCS})
```

CMake - sample library

CMakeLists.txt

```
PROJECT(foo)
SUBDIRS(src)
```

src/CMakeLists.txt

```
SET(foo_SRCS foo.c
ADD_LIBRARY(foo SHARED ${foo_SRCS})
```

Replace `SHARED` by `STATIC` for a static library.

CMake - installation

CMakeLists.txt

```
INSTALL(TARGET hello DESTINATION bin)
INSTALL(TARGET foo DESTINATION lib)
```

Installs `hello` and `libfoo.so` in
`$prefix/bin` and `$prefix/lib` respectively.

CMakeLists.txt

```
INSTALL(FILE README.foo DESTINATION share/doc)
```

Installs the `README.foo` file.

CMake - language of CMakeLists.txt

- Scripting language with a simple syntax
- Comments: #
- Commands: `COMMAND(arg1 arg2 ...)`
- Lists: `A; B; C`
- Variables: `$VAR`

CMake - control structures

Condition

```
IF(CONDITION)
    MESSAGE("Yes")
ELSE(CONDITION)
    MESSAGE("No")
ENDIF(CONDITION)
```

Macros

```
MACRO(MY_MACRO arg1 arg2)
    SET($arg1 "$$arg2")
ENDMACRO(MY_MACRO)
MY_MACRO(A B)
```

Loops

```
FOREACH(c A B C)
    MESSAGE("$c: $$c")
ENDFOREACH(c)
```

CMake - macros composition

example

```
MACRO (CREATE_EXECUTABLE NAME SOURCES LIBRARIES)
    ADD_EXECUTABLE($NAME $SOURCES)
    TARGET_LINK_LIBRARIES($NAME $LIBRARIES)
ENDMACRO(CREATE_EXECUTABLE)

ADD_LIBRARY(foo foo.c)
CREATE_EXECUTABLE(hell main.c foo)
```

CMake - variables creation

Creation of boolean options, usable on the command line

example

```
OPTION(DEBUG "Program in DEBUG mode" OFF)

IF(DEBUG)
   SET_SOURCE_FILES_PROPERTIES(main.c COMPILE_FLAGS -DDEBUG)
ENDIF(DEBUG)
```

```
cmake -DDEBUG:BOOL=ON
```

CMake - dependencies handling

`FIND_PACKAGE(`*`BAR`*`)`

- uses `FindBAR.cmake`

Example

```
PROJECT(myProject)
FIND_PACKAGE(PNG)
IF(PNG_FOUND)
    INCLUDE($PNG_USE_FILE)
ENDIF(PNG_FOUND)
ADD_EXECUTABLE(myProject myProject.cxx)
TARGET_LINK_LIBRARIES(myProject PNG)
```

CMake - pkg-config

- pkg-config is not available under MS-Windows
  $\rightarrow$ avoid using it with CMake on Windows targets
- module `FindPkgConfig.cmake`

example

```
FIND_PACKAGE(PkgConfig)
IF(PKG_CONFIG_FOUND)
  PKG_CHECK_MODULE(GTK REQUIRED gtk+-2.0)
  IF(GTK_FOUND)
    # GTK_CFLAGS GTK_LIBRARIES sont definis
  ENDIF(GTK_FOUND)
ENDIF(PKG_CONFIG_FOUND)
```

CMake - packaging with CPack

- CPack generates source and binary packages
  - RPM, DEB binary (and source) packages for Linux
  - Mac OS X `.pkg` and `.dmg`
  - NSIS installers for Windows
  - sources archives `.tgz` and `.tar.bz2`
- Reuses `INSTALL` declarations from CMake
- Important: set variables **before** `INCLUDE(CPack)`

CPack - example

CMakeLists.txt

```
INCLUDE(InstallRequiredSystemLibraries)

SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "My Package")
SET(CPACK_PACKAGE_VENDOR "Matthieu Herrb")
SET(CPACK_RESOURCE_FILE_LICENSE
   "$CMAKE_CURRENT_SOURCE_DIR/COPYING")
SET(CPACK_PACKAGE_VERSION_MAJOR "0")
SET(CPACK_PACKAGE_VERSION_MINOR "1")
SET(CPACK_GENERATOR "DEB")
SET(CPACK_DEBIAN_PACKAGE_MAINTAINER "Matthieu Herrb")
...
INCLUDE(CPack)
```

Use: `make package`

Agenda

1 Introduction

2 Autotools

3 CMake

4 Conclusion

Conclusion

- build and packaging are important steps
- clearly identify the target audience
- choose adapted formats and tools
- links to continuous integration methods
- links to software forges (GitHub, SourceForge, Trac,..) and their tools

Questions ?