

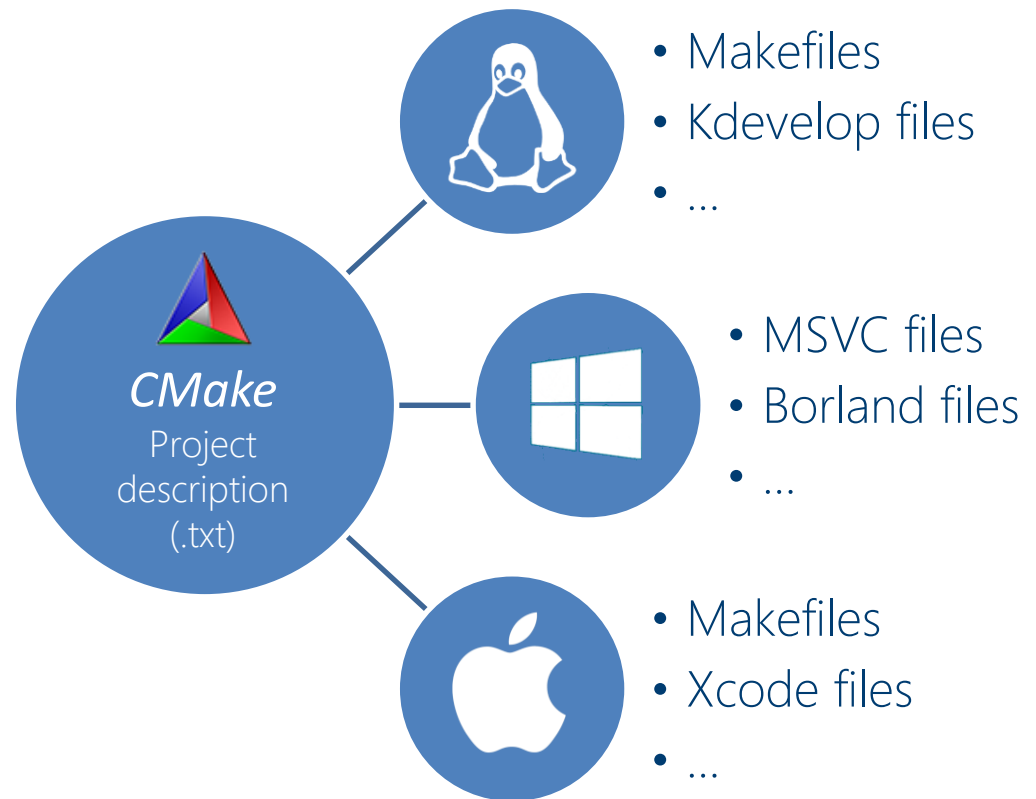
Automating Software Build Process using CMake

Armin Sobhani
asobhani@sharcnet.ca

SHARCNET

University of Ontario Institute
of Technology (UOIT)

September 14, 2016



Outline

- CMake Overview
 - What is CMake
 - Why CMake
 - CMake history
 - Features
 - Concepts
- Using CMake
- Developing with CMake

What is CMake?

- CMake is the cross-platform, open-source build system that lets you use the native development tools
- It's a build system generator
- It takes plain text files as input that describe your project and produces project files or make files for use with a wide variety of native development tools.

Why CMake?

- It's easy and works well
 - Certainly way simpler than Makefiles!
- It's cross-platform
- It's popular (scientific and commercial)
- It can build a directory tree outside of the source tree

CMake History

- Built for the Insight Segmentation and Registration Toolkit (ITK) (<http://www.itk.org>)
- Funded by National Library of Medicine (NLM): part of the Visible Human Project
- Release 1.0 branch created in late 2001

CMake Features

- Native tools
 - Unix Makefiles
 - Borland Makefiles
 - MSYS Makefiles
 - MinGW Makefiles
 - NMake Makefiles
 - Watcom WMake
 - Ninja (Google)
 - MSBuild
- IDE Support
 - Code::Blocks
 - CodeLite
 - Eclipse CDT
 - KDevelop
 - Visual Studio
 - Xcode
 - Kate (KDE Text Editor)
 - Sublime Text 2

CMake Features

OS Support

Linux

OS X

Windows

Solaris / SunOS

HPUX

IRIX

Platform inspections commands can:

- Search for
 - Programs
 - Libraries and header files
 - Packages
- Determine hardware specifics
 - Byte order
 - Number of bits

Using CMake

Installing CMake

- Windows
 - Download: <https://cmake.org/download/>
- OS X
 - Download
 - MacPorts: `port install cmake`
- Linux
 - Download
 - Use package manager

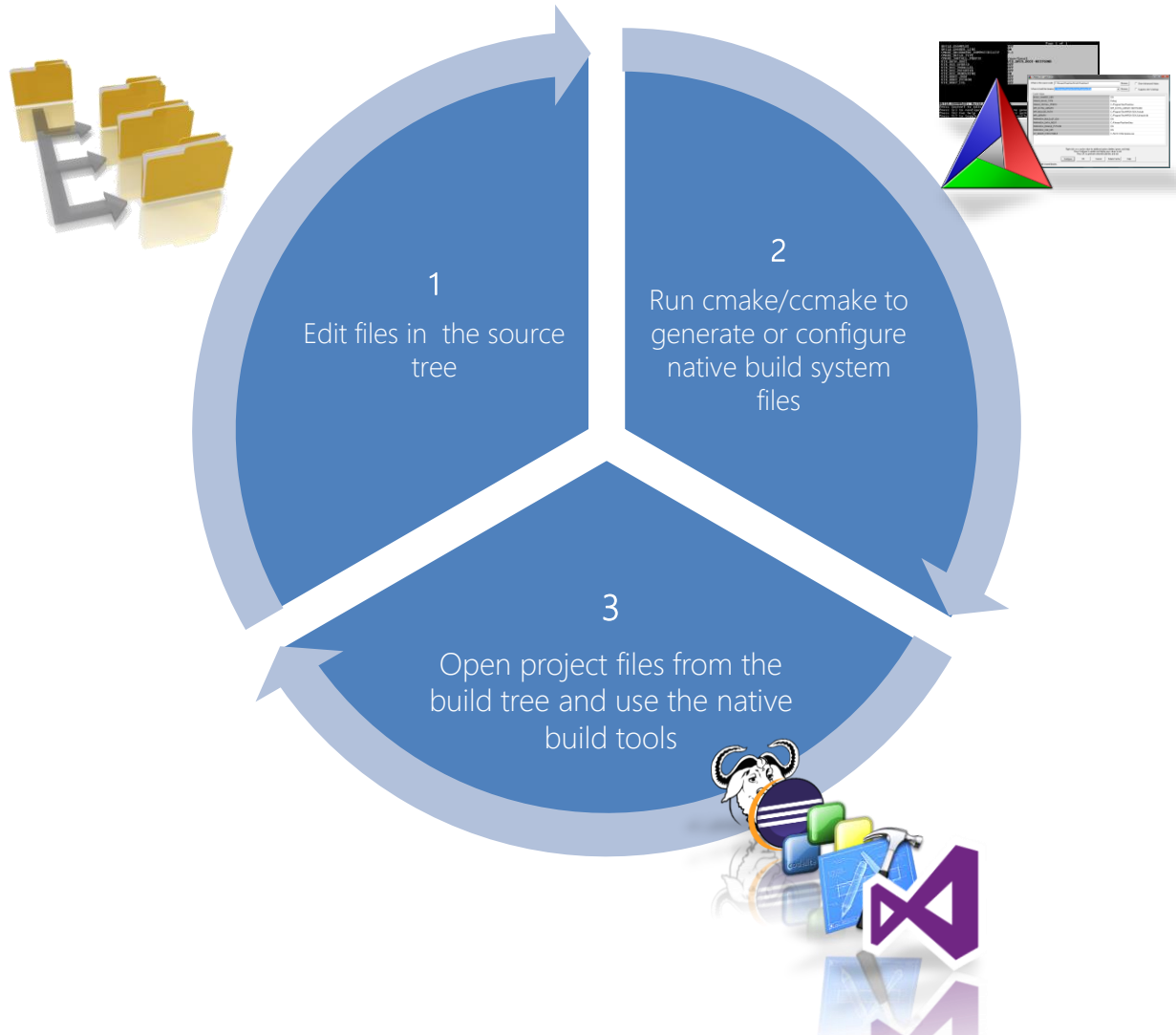
CMake on SHARCNET's Systems

- <https://www.sharcnet.ca/my/software/show/122>
- <https://www.sharcnet.ca/help/index.php/CMAKE>

- Default (CentOS 6.0): 2.8.12.2
- Using modules: 3.3.2, 3.4.3

Developing Software with CMake

CMake Workflow



Editing CMakeLists.txt

- Editor support and syntax highlighting:
 - [https://cmake.org/Wiki/CMake Editors Support](https://cmake.org/Wiki/CMake%20Editors%20Support)
- Emacs
- vim
- Notepad++
- Sublime Text
- ...

CMake Concepts

- Source tree
- Binary tree = Build tree
- Install tree
- Generator
- Cache Entry or Cache Variable
- Target (library, executable, custom)

Build Configurations

Make file generators

- CMAKE_BUILD_TYPE
- Known values
 - Debug
 - Release
 - MinSizeRel
 - RelWithDebInfo

Multi-config generators

- Use multiple build trees

Our First CMake Project

- Only 2 lines of code!

CMake Syntax 101

Basic CMake Syntax

- CMake language components
 - Commands (with arguments)
 - Variables
 - All values are strings
 - Comments (start with '#')

Basic CMake Syntax – Commands

- Case insensitive
- Arguments are case sensitive and space separated
- Quoted argument is always one value
- List of valid commands:
 - <https://cmake.org/cmake/help/v3.0/manual/cmake-commands.7.html>
 - **cmake --help-command-list**

Basic CMake Syntax – Commands

- Multiple argument to the SET command are combined with semicolon:

```
set (VAR a b c)      # VAR="a;b;c"  
set (VAR "a;b;c")   # VAR="a;b;c"
```

Basic CMake Syntax – Commands

- Examples:

```
set(name myexe)
```

```
set(srcs src1.c src2.c s3.c)
```

```
set(srcs "src1.c;src2.c;s3.c")
```

```
add_executable("${name}" ${srcs})
```

```
add_executable("myexe" src1.c src2.c s3.c)
```

```
add_executable("myexe" src1.c;src2.c;s3.c)
```

Basic CMake Syntax – Variables

- Variable names are case sensitive:

```
set(VAR value)
```

```
set(var value)
```

- Use alpha-numeric and underscores

- Variables are strings:

```
set(VAR "a b c") # VAR holds one thing
```

```
set(var a;b;c) # VAR holds three things
```

```
set(var a b c) # VAR holds three things
```

Basic CMake Syntax – Variables

- Use the `list` or `foreach` commands to access list elements
- Special syntax for setting environment variables:

```
set (ENV{ROOT_DIR} "/home/username/root")
```

Basic CMake Syntax – Variables

- Variable (de-) referencing: `${VAR}`

```
set(my_dir "${CMAKE_SOURCE_DIR}/my_dir")  
message("my_dir='${my_dir}'")
```

- For environment variables: `$ENV{VAR}`

```
set(my_path "$ENV{PATH}")  
message("my_path='${my_path}'")
```


Basic CMake Syntax – Variables

- Escaping – \ is the escape character used in string literals:

```
set(VAR "a\\b\\c and \"embedded quotes\"")  
message(${VAR}) # "a\b\c and "embedded quotes"
```

```
set(VAR a b c)  
message(${VAR}) # "abc"  
message("${VAR}") # "a;b;c"
```

```
set(VAR)  
message("${VAR}") # ""  
message(${VAR}) # error!
```

Flow Control (if)

```
if (VAR)
  code
endif (VAR) # arguments to endif must match if
```

```
if (NOT VAR)
if (VAR AND VAR2)
if (VAR OR VAR2)
if (VAR MATCHES regular_expression)
if (COMMAND command)
if (EXISTS file)
if (VAR LESS VAR2)
```

`cmake --help-command if` for more details

Flow Control (if) – FALSE Values

- `""` (the empty string)
- `OFF`
- `0` (the number zero)
- `NO`
- `FALSE`
- `N`
- `NOTFOUND` exactly or ends in `"-NOTFOUND"`
- `IGNORE`

Flow Control (FOREACH, WHILE)

```
foreach(F a b c)
    message(${F})
endforeach(F)
```

```
while(VAR)
    message(${VAR})
    set(VAR FALSE)
endwhile(VAR)
```

Macros and Functions

```
macro(MYMACRO arg1 arg2)
    command1 (...)
    command2 (...)
endmacro(MYMACRO)
```

```
# cmake 2.6 and later
# dynamically scoped, any variables set are local to func
function(myfunction arg1 arg2)
    command1 (...)
    command2 (...)
endfunction(myfunction)
```

CMake Commands

Common commands

```
cmake_minimum_required(VERSION 2.6)
project(projectname CXX)
set(libname sumdulib)
add_library(${libname} src1.cpp src2.cpp ...)
add_executable(exename src1.cpp src2.cpp ...)
target_link_libraries(exename ${libname} ...)
add_subdirectory(tests)
```