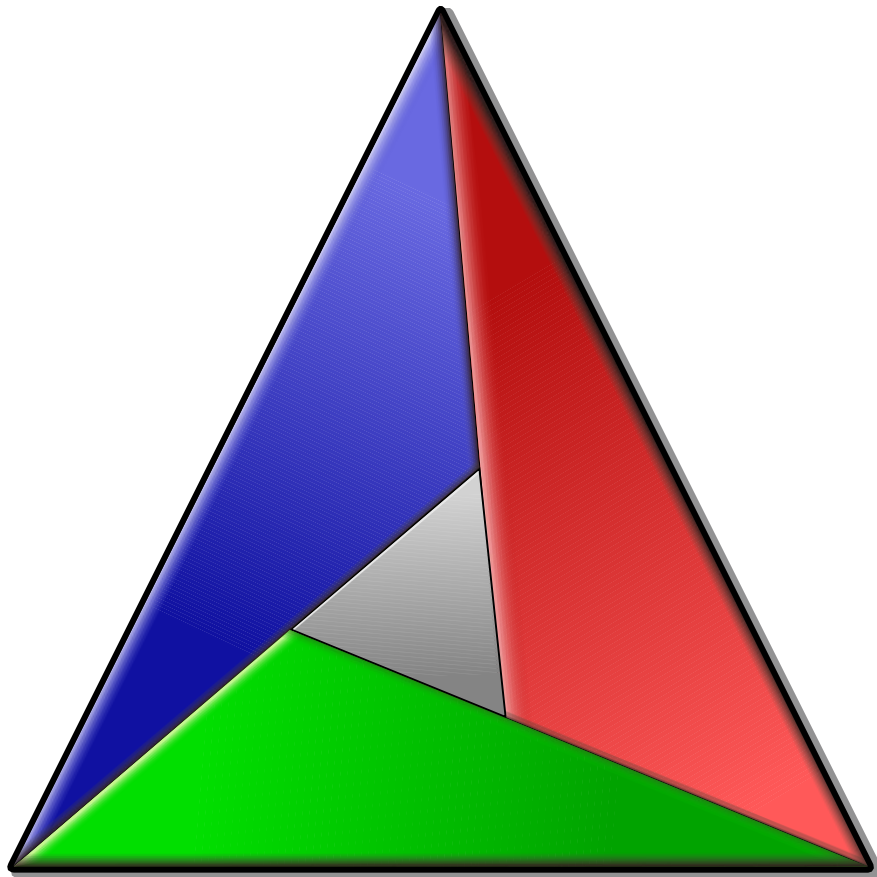


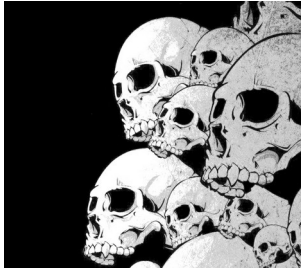
Cmake  
<https://cmake.org/>



# Introduction à CMake

Y. Collette





# Plan

## Introduction à CMake

### Utilisation standard de CMake

- Compiler GLPK

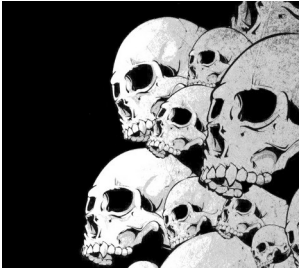
### Utilisation avancée de Cmake

- Télécharger des dépendances

### Gestion des tests de non régression

- Ctest

### Le packaging



# Exemple d'application

On utilisera cmake pour compiler GLPK sur Linux et Windows:

<https://ftp.gnu.org/gnu/glpk/glpk-4.65.tar.gz>

C'est un outil d'optimisation numérique (programmation linéaire) écrit en C.

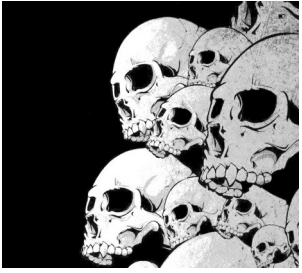
On utilisera ctest pour ajouter des tests de non régression

On utilisera cpack pour générer un installeur

Prérequis :

- installer cmake → <https://cmake.org/>
- installer un compilateur C ou C++ sous Windows (<https://www.msys2.org/> ou Visual Studio par exemple) ou Linux (gcc et / ou gcc-c++)





# Objectif d'un système de build

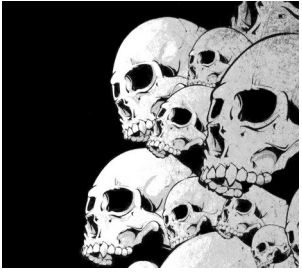
**Efficacité:** Économie de temps à la recompilation

Analyse des modifications et des dépendances

**Flexibilité:** configuration des paramètres de compilation

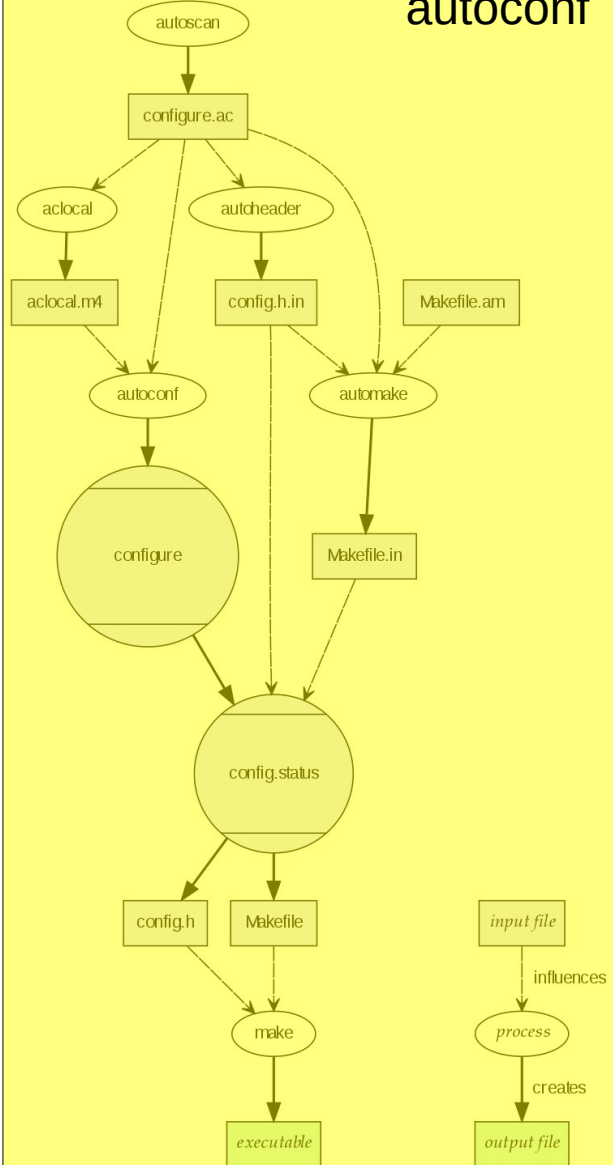
**Scalabilité:** gestion des sous-projets et des bibliothèques

**Portabilité:** Compile pour différentes plateformes



# Les autres systèmes de build

## autoconf



WAF (Python)

SCONS (Python)

Imake (X11)

Qmake (Qt)

Makefile  
nmake / make

premake

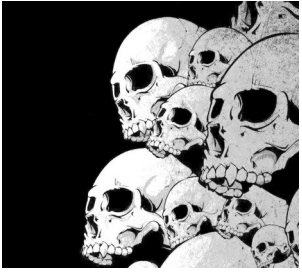
MSBuild (VS)

HomeBrew (MacOS)

MacPort (MacOS)

Ninja

Meson



# Exemple de build

## Meson / Ninja

```
$ mkdir build  
$ meson setup build  
$ meson configure build  
$ cd build  
$ ninja
```

## CMake

```
$ mkdir build  
$ cd build  
$ cmake ..  
$ make
```

## Autoconf / Automake

```
$ autoreconf --install  
$ ./configure  
$ make
```

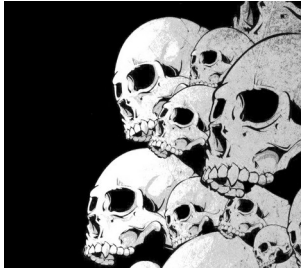
## SCons

```
$ scons -2
```

## Premake

```
$ premake4 gmake  
$ cd buildpremake  
$ make
```

Il y a un exemple de projets  
dans l'archive d'exemples  
venant avec les slides



# Points importants de CMake

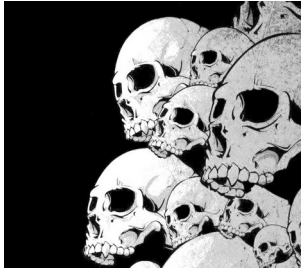
Génère un environnement de build pour:

- UNIX/Linux → Makefiles
- Windows → VS Projects/Workspaces (et autres)
- Apple → Xcode
- Projets Eclipse
- Etc.

Beaucoup de macros pour détecter des éléments de plateforme (bibliothèques, headers, outils)

Cross-Platform (Linux / MacOS / Windows / x86 / amd64 / arm)

Fournit des outils de packaging (via CPack) et de test (via CTest)



# Qui utilise CMake

Quelques exemples de projets qui utilisent cmake :

**Linden Lab** (pour Second Life)

**KDE**

**Boost** (récemment arrivé dans le dépôt git)

**MySQL**

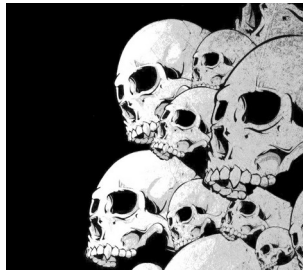
**Half-Life 2 SDK**

**Rosegarden** (outil pour la MAO sous Linux)

Et plein d'autres projets (voir <https://cmake.org/success/>) pour plus de détails

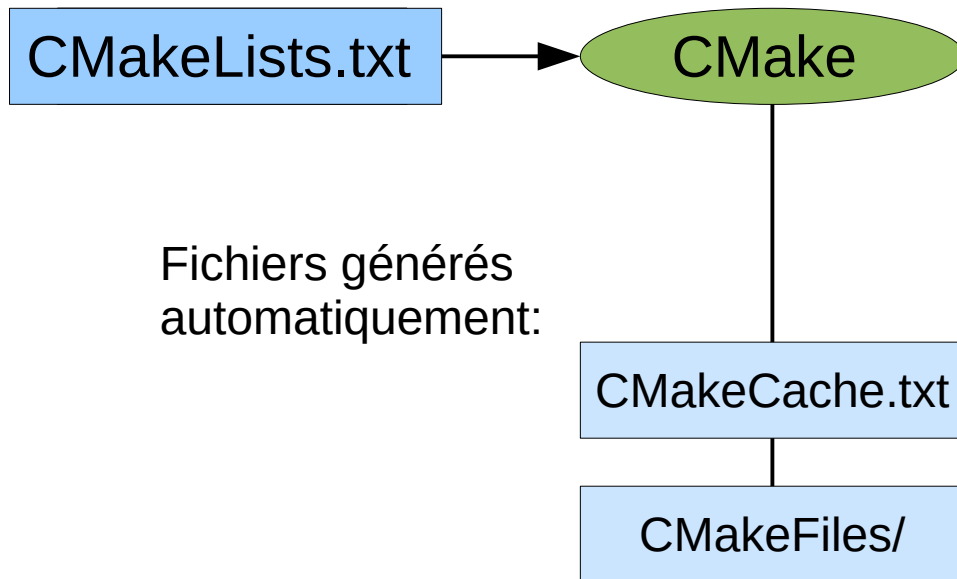






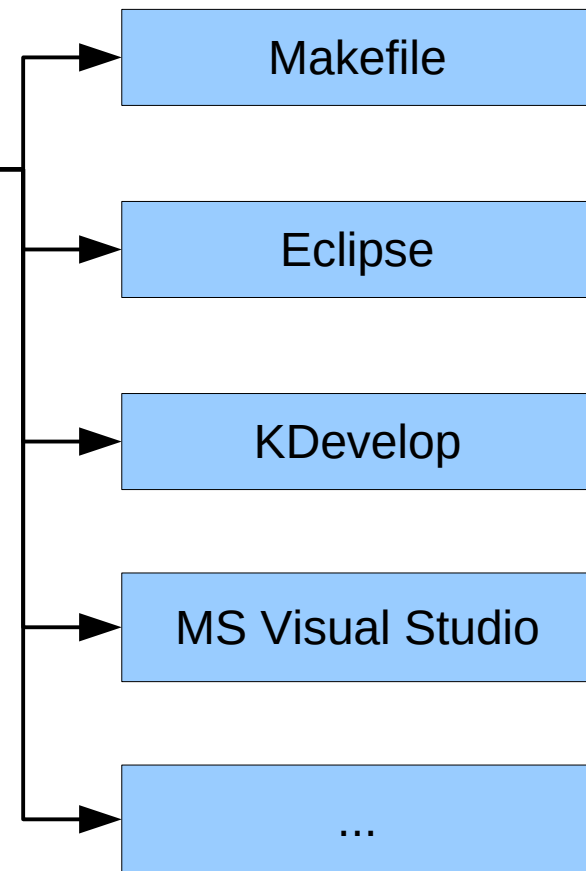
# Build process

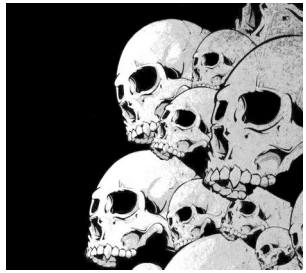
Fichier source CMake:



Fichiers générés automatiquement:

Projets cible:





# Syntaxe

## Le fichier CmakeLists.txt

### CMakeLists.txt

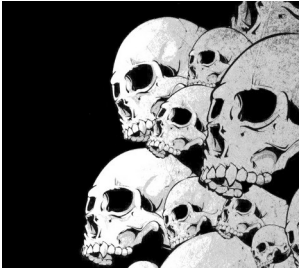
Le fichier principal qui contient les paramètres du projet et qui décrit les différentes séquences de compilation en utilisant le langage CMake. Le langage CMake n'est pas sensible à la casse.

Exemple de syntaxe:

**# Commentaire**

**command**(argument1 argument2 ...argumentN)

**COMMAND**(argument1 argument2 ...argumentN)



# Obtenir de l'aide

```
cmake -help-command-list
```

```
cmake -help-command add_executable
```

```
cmake -help-variable-list
```

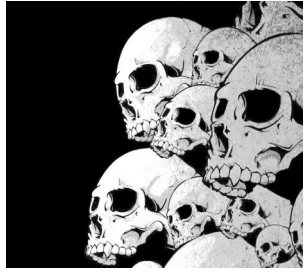
```
cmake -help-variable CMAKE_INSTALL_PREFIX
```

```
cmake -help-module-list
```

```
cmake -help-module FindZLIB
```

Et :

<https://cmake.org/cmake/help/v3.17/>



# Syntaxe

## Etude d'un exemple

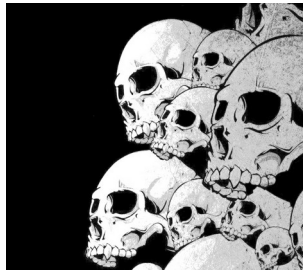
Analyse des fichiers CMakeLists.txt d'un exemple.

Nous allons voir la syntaxe cmake à travers le CmakeLists.txt destiné à compiler la librairie Zlib.

Le code source de Zlib :

<https://zlib.net/>








# GLPK

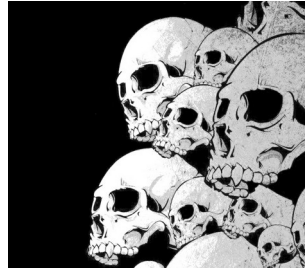
## Architecture du projet

```
[glpk-4.65] $ ls -1 | xargs -n 2
```

aclocal.m4	AUTHORS
autogen.sh	ChangeLog
config.guess	config.h.in
config.sub	configure
configure.ac	COPYING
depcomp	doc
examples	INSTALL
install-sh	ltmain.sh
m4	Makefile.am
Makefile.in	missing
NEWS	README
src	THANKS
w32	w64

-  → Fichiers du système de build Linux
-  → Fichiers du système de build Windows
-  → Répertoire à utiliser pour la compilation





# Hello World

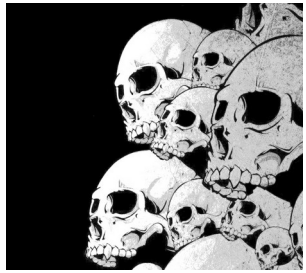
## Step 1

```
cmake_minimum_required(VERSION 3.0)
project(GLPK C)
message(STATUS "Hello World !")
```

Détection du compilateur C

Affichage de "Hello World !"

```
[build] $ cmake ..
-- The C compiler identification is GNU 9.2.1
-- Check for working C compiler: /usr/lib64/ccache/cc
-- Check for working C compiler: /usr/lib64/ccache/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Hello World !
-- Configuring done
-- Generating done
-- Build files have been written to: /home/formation/glpk/build
```



# “Hello world!” Step 2

**hello.c :**

```
int main() { printf("Hello, world!\n"); }
```

**CMakeLists.txt :**

Target hello

```
project(hello C)
```

```
add_executable(hello hello.c)
```

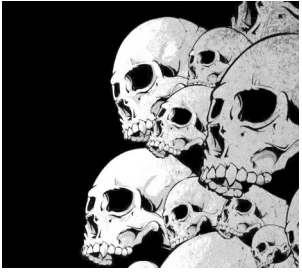
**Shell:**

```
$ mkdir build && cd build && cmake ..
```

```
$ make
```

```
$ ./hello
```

```
Hello, world!
```



# Utilisation de librairies libpng

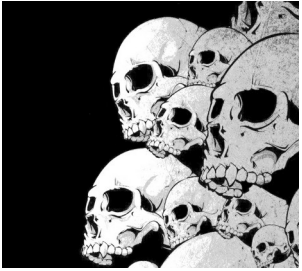
## Code source C:

```
...  
png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING,  
                                NULL, NULL, NULL);  
...
```

## Ajout dans **CMakeLists.txt** :

```
find_package(PNG)  
include_directories(${PNG_INCLUDE_DIRS})  
add_definitions(${PNG_DEFINITIONS})  
add_executable(testpng testpng.c)  
target_link_libraries(testpng ${PNG_LIBRARIES})
```





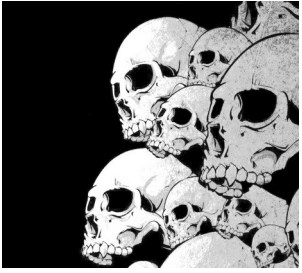
# Utilisation de librairies

## Step 3 et Step 3 bis

```
cmake ../Step3/  
-- The C compiler identification is GNU 9.2.1  
-- Check for working C compiler: /usr/lib64/ccache/cc  
-- Check for working C compiler: /usr/lib64/ccache/cc -- works  
-- Detecting C compiler ABI info  
-- Detecting C compiler ABI info - done  
-- Detecting C compile features  
-- Detecting C compile features - done  
-- Found ZLIB: /usr/lib64/libz.so (found version "1.2.11")  
-- Found PNG: /usr/lib64/libpng.so (found version "1.6.37")  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/formation/build
```

```
$ make  
Scanning dependencies of target testpng  
[ 50%] Building C object  
CMakeFiles/testpng.dir/testpng.c.o  
[100%] Linking C executable testpng  
[100%] Built target testpng
```

```
$ ldd testpng  
linux-vdso.so.1 (0x00007ffcadbe8000)  
libpng16.so.16 => /lib64/libpng16.so.16 (0x00007f6b270f5000)  
libz.so.1 => /lib64/libz.so.1 (0x00007f6b270db000)  
libc.so.6 => /lib64/libc.so.6 (0x00007f6b26f12000)  
libm.so.6 => /lib64/libm.so.6 (0x00007f6b26dcc000)  
/lib64/ld-linux-x86-64.so.2 (0x00007f6b27163000)
```



## Module Find<name>.cmake

Variables standards:

- <name>\_FOUND
- <name>\_LIBRARIES
- <name>\_INCLUDE\_DIRS

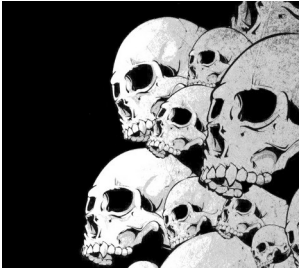
Utilisation d'un module:

-> Ajouter: find\_package(<nom du module>)  
exemple: find\_package(Eigen)

Exemple d'implémentation d'un script de détection :

```
if (GMP_INCLUDE_DIR AND GMP_LIBRARIES)
  # Already in cache, be silent
  set(GMP_FIND_QUIETLY TRUE)
endif()
find_path(GMP_INCLUDE_DIR NAMES gmp.h )
find_library(GMP_LIBRARIES NAMES gmp )
if(GMP_INCLUDE_DIR AND GMP_LIBRARIES)
  set(GMP_FOUND 1)
endif()
mark_as_advanced(GMP_INCLUDE_DIR GMP_LIBRARIES)
```





# Création de bibliothèques

Les éléments importants du CmakeLists.txt:

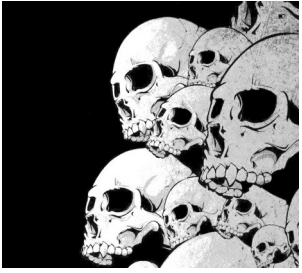
Les fichiers à compiler:  
mylibrary.c  
mylibrary.h  
hello.c

```
include_directories(${CMAKE_SOURCE_DIR})  
add_library(myhellolib SHARED mylibrary.c)
```

```
add_executable(hello hello.c)  
target_link_libraries(hello myhellolib)
```

```
$ make  
Scanning dependencies of target myhellolib  
[ 25%] Building C object  
CMakeFiles/myhellolib.dir/mylibrary.c.o  
[ 50%] Linking C shared library libmyhellolib.so  
[ 50%] Built target myhellolib  
Scanning dependencies of target hello  
[ 75%] Building C object CMakeFiles/hello.dir/hello.c.o  
[100%] Linking C executable hello  
[100%] Built target hello
```

```
$ ldd hello  
linux-vdso.so.1 (0x00007ffcbba56000)  
libmyhellolib.so => /home/formations/build/libmyhellolib.so (0x00007f3396653000)  
libc.so.6 => /lib64/libc.so.6 (0x00007f3396454000)  
/lib64/ld-linux-x86-64.so.2 (0x00007f339665a000)
```



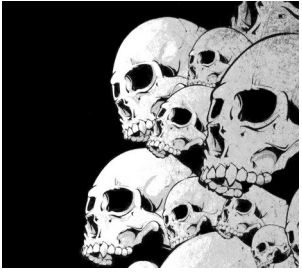
# Utiliser des bibliothèques 1/3

**Objectif** : compiler une partie de code si un élément a été trouvé par cmake

```
Image::Image(){  
    qDebug() << "Le système supporte les images de type :" << "png," << "tiff";  
}
```

Ajout dans CMakeLists.txt :

```
find_package(PNG)  
if (PNG_FOUND)  
    set(HAVE_PNG 1)  
endif()  
configure_file(config.h.cmake ${CMAKE_CURRENT_BINARY_DIR}/config.h)
```



# Utiliser des bibliothèques 2/3

Fichier config.h.cmake:

```
#cmakedefine HAVE_PNG 1
```

il deviendra lors de la génération si PNG trouvé:

```
#define HAVE_PNG 1
```

```
sinon
```

```
// #define HAVE_PNG 1
```

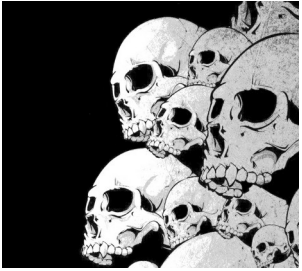
Autre alternative de syntaxe dans config.h.cmake :

```
#define CBC_SVN_REV @CBC_SVN_REV@
```

```
#include <config.h>
```

```
Image::Image() {  
    qDebug() << "Le système supporte les images:"  
#ifdef HAVE_PNG  
    << "png"  
#endif  
    << "tiff";  
}
```

cmake --help-command configure\_file  
Pour plus d'informations



# Utiliser des bibliothèques 3/3

Recherche de header:

```
include(CheckIncludeFiles)
check_include_files(sys/stat.h HAVE_SYS_STAT_H)
```

Recherche de prototype de fonction :

```
include(CheckPrototypeDefinition)
check_prototype_exists(mkstemps "stdlib.h;unistd.h" HAVE_MKSTEMPS_PROTO)
```

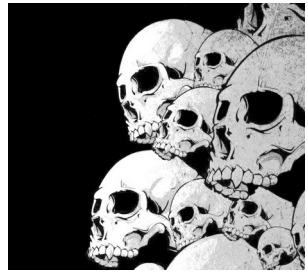
Compilation :

```
include(CheckCXXSourceCompiles)
check_cxx_source_compiles("
#include <sys/types.h>
#include <sys/statvfs.h>
int main(){
    struct statvfs *mntbufp;
    int flags;
    return getmntinfo(&mntbufp, flags);
}
" GETMNTINFO_USES_STATVFS )
```

Pour plus d'aide :

```
cmake --help-module CheckIncludeFiles
cmake --help-module CheckPrototypeDefinition
cmake --help-module CheckCXXSourceCompiles
```





# Syntaxe 1/8

## Travailler avec des sous répertoires

Pour ajouter un sous-projet:

**add\_subdirectory(sub)**

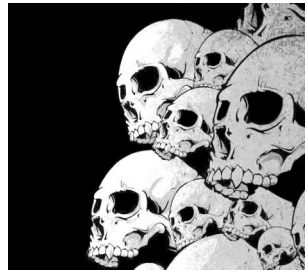
CMake ajoute le sous répertoire aux dépendances.  
L'enfant hérite du parent.

Rechercher des fichiers d'include dans un sous  
répertoire:

**include\_directories(sub)**

Rechercher des bibliothèques dans un sous-répertoire:

**link\_directories(sub)**



# Syntaxe 2/8

## Commandes additionnelles

### **cmake\_minimum\_required** (VERSION 2.6)

Règle la version minimum de cmake pour le projet.

### **project** (projectname [CXX] [C] [Java])

Crée les variables projectname\_BINARY\_DIR et projectname\_SOURCE\_DIR.

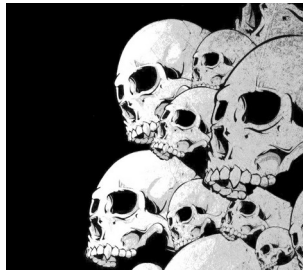
### **add\_definitions** (-g)

Arguments passés au compilateur (comme **CFLAGS**)

### **message** (text)

Affiche du texte





# Syntaxe 3/8

## Résumé des commandes importantes

```
project(name C)
```

```
cmake_minimum_required(VERSION 3.3)
```

```
add_definitions(-g) # ajouté à CFLAGS
```

```
add_library(mylibrary SHARED mylibrary.c)
```

```
add_subdirectory(sub)
```

```
include_directories(/usr/include) # Facultatif car ajouté par défaut
```

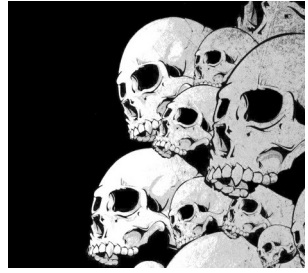
```
link_directories(/usr/lib64) # Facultatif car ajouté par défaut
```

```
add_executable(hello hello.c)
```

```
target_link_libraries(hello mylibrary)
```

```
message("Hello World !")
```

**Les 10 commandes les plus importantes pour 90% des projets.**



# Syntaxe 4/8

## Expressions conditionnelles

Les plus utiles:

**if** (exp)

**elseif** (exp2)

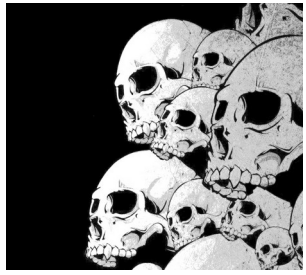
**else** ([exp])

**endif** ([exp])

Il en existe d'autres (cmake --help-command-list):

**foreach**

**while**



# Syntaxe 5/8

## Code réutilisable

**include** (<file|module>)

**macro** (<name> [arg1 [arg2 [arg3 ...]]])

command1 (ARGS ...)

command2 (ARGS ...)

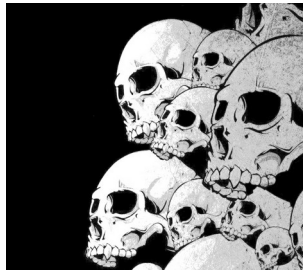
**endmacro** (<name>)

**function** (<name> [arg1 [arg2 [arg3 ...]]])

command1 (ARGS ...)

command2 (ARGS ...)

**endfunction** (<name>)



# Syntaxe 6/8

## Les expressions régulières 1/2

Attention: ne sont pas Perl compatible

```
set (S "abc")
```

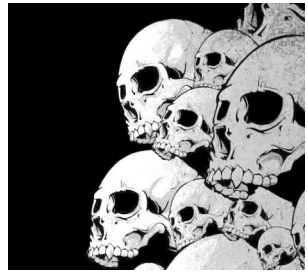
```
string (REGEX REPLACE "b.*" "BC" S ${S})
```

```
# S → aBC
```

```
set (VER 1.2)
```

```
string (REGEX MATCHALL "[0-9]+" NUMS "${VER}")
```

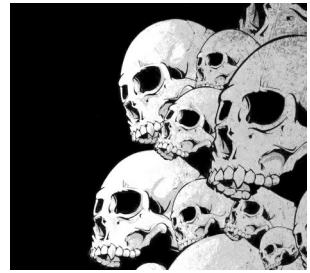
```
# NUMS est une liste : NUMS → 1 ;2
```



# Syntaxe 7/8

## Les expressions régulières 2/2

```
macro (today RESULT)
  if (WIN32)
    execute_process(COMMAND "cmd" " /C date /T" OUTPUT_VARIABLE ${RESULT})
    string(REGEX REPLACE "(..)/(..)/..(..).*" "\\1/\\2/\\3" ${RESULT} ${${RESULT}})
  elseif (UNIX)
    execute_process(COMMAND "date" "+%d/%m/%Y" OUTPUT_VARIABLE ${RESULT})
    string(REGEX REPLACE "(..)/(..)/..(..).*" "\\1/\\2/\\3" ${RESULT} ${${RESULT}})
  else (WIN32)
    message(SEND_ERROR "date not implemented")
    set(${RESULT} 000000)
  endif ()
endmacro ()
```



# Syntaxe 8/8 Installers

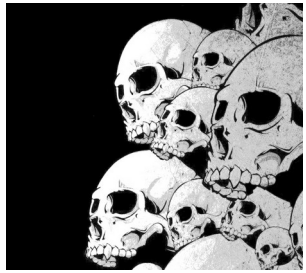
Ca peut être très simple:

```
cmake -help-command install
```

**install** (TARGETS hello DESTINATION bin)

Ou un peu plus compliqué:

```
install (TARGETS targets... [EXPORT <export-name>]  
  [[ARCHIVE|LIBRARY|RUNTIME|FRAMEWORK|BUNDLE|  
  PRIVATE_HEADER|PUBLIC_HEADER|RESOURCE]  
  [DESTINATION <dir>]  
  [PERMISSIONS permissions...]  
  [CONFIGURATIONS [Debug|Release|...]]  
  [COMPONENT <component>]  
  [OPTIONAL] [NAMELINK_ONLY|NAMELINK_SKIP]  
  ] [...])
```



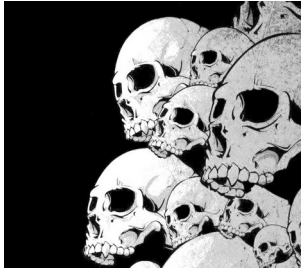
# Syntaxe

## Trouver des bibliothèques

Un fichier cmake spécial écrit dans le but de trouver un outil, un header ou une bibliothèque ou des définitions et écrit les différents résultats dans des variables cmake de façon à être utilisé dans le build du projet. (ex: FindJava.cmake, FindZLIB.cmake, FindQt4.cmake)

Plus de 150 Find\* modules se trouvent dans le répertoire d'installation de cmake.

Plus de 250 macros diverses se trouvent aussi dans ce répertoire d'installation (/usr/share/cmake/Modules sous Linux).



# Plan

## Utilisation avancée de Cmake

Paramétrage du build

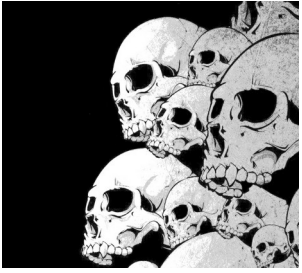
Variables et lists

Variables du cache

Out-of-source build

Cross-compilation





# Paramétrage du build

Il est possible d'ajouter des options modifiables via cmake-gui et ccmake.  
Pour une option booléenne :

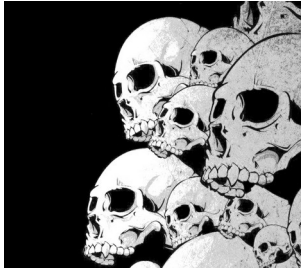
```
option(MYOPTION "Descriptive text" ON)
```

Pour une option d'un autre type :

```
set(MYPATHOPTION "/tmp" CACHE PATH "The TMPDIR path")
```

Pour masquer une option et le la découvrir que si « Avancé » est coché :

```
mark_as_advanced(MYOPTION  
                 MYPATHOPTION)
```



# Le cache CMake

Créé dans le répertoire de build (*CMakeCache.txt*)

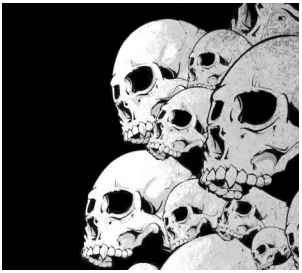
Contient des entrées **VAR:TYPE=VALUE**

Rempli/Mis à jour pendant la phase de configuration

Accélère le build

Peut être initialisé avec *cmake -C <fichier>*

**cmake-gui** ou **ccmake** peuvent être utilisés pour changer des valeurs



# Editer le cache CMake

ccmake (interface graphique ncurses)

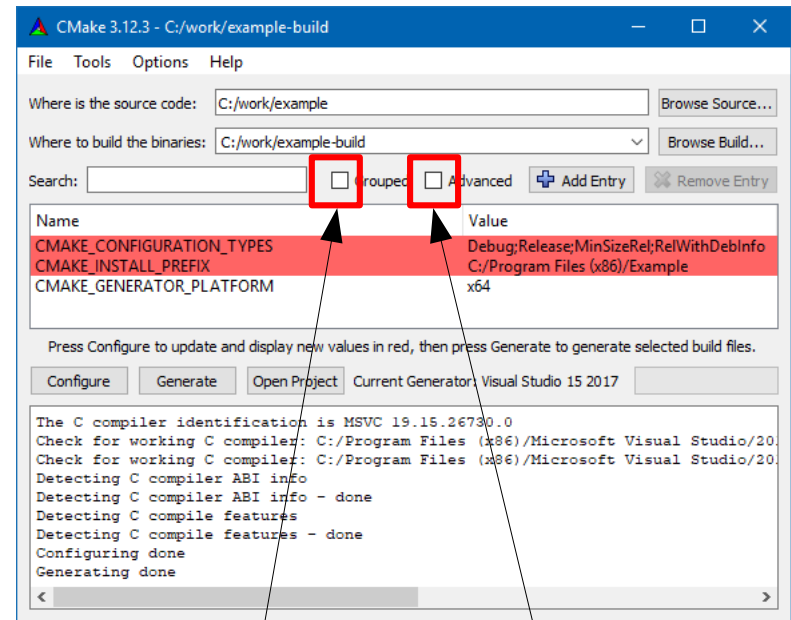
```
Page 1 of 1
BUILD_EXAMPLES      OFF
BUILD_SHARED_LIBS   ON
CMAKE_BACKWARDS_COMPATIBILITY 2.1
CMAKE_BUILD_TYPE
CMAKE_INSTALL_PREFIX /usr/local
UTK_DATA_ROOT       UTK_DATA_ROOT-NOTFOUND
UTK_USE_HYBRID      OFF
UTK_USE_PARALLEL    OFF
UTK_USE_PATENTED    OFF
UTK_USE_RENDERING   ON
UTK_WRAP_JAVA       OFF
UTK_WRAP_PYTHON     OFF
UTK_WRAP_ICL        OFF

BUILD_EXAMPLES: Build UTK examples.
Press enter to edit option
Press ctrl to configure
Press ctrl to generate and exit
Press ctrl for help
Press ctrl to quit without generating
Press ctrl to toggle advanced mode <Currently Off>

CMake Version 2.0 - patch 2
```

t : Faire apparaître les variables avancées

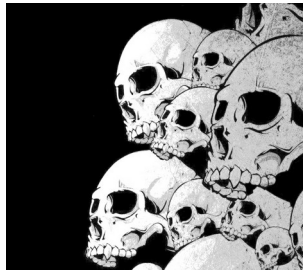
cmake-gui



Regrouper les variables par préfixe

Faire apparaître les variables avancées





# Syntaxe

## Variables et Listes

### Variables:

```
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -g")
```

```
message(STATUS "CMAKE_C_FLAGS: ${CMAKE_C_FLAGS}")
```

cmake -D <var>:<type>=<value> # définit une variable et la stocke dans le *cache*

### Listes :

```
set(L 1;2;3) # le point virgule est le séparateur de liste
```

```
list(APPEND L a b c )
```

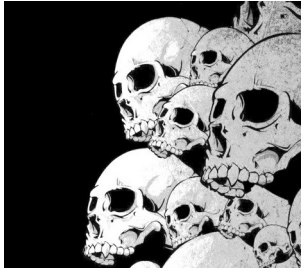
```
message(STATUS "\${L}=" "${L}")
```

```
message(STATUS "\"\${L}\"=" "${L}")
```

### Sortie:

```
${L}=123abc
```

```
"${L}"=1;2;3;a;b;c
```



# Compilation Out-of-source

Une compilation Out-of-source est utilisée pour séparer le résultat du build des fichiers source. Très utile pour le développement multi-plateformes. Se mettre dans le répertoire de build et indiquer à cmake le répertoire de sources comme argument:

```
mkdir -p ../build; cd ../build
```

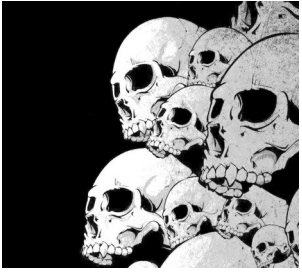
```
cmake ../hello
```

```
make
```

Tous les éléments du build doivent rester dans le répertoire de build :

- les binaires compilés
- les headers générés par cmake
- etc ...

Le répertoire des sources doit rester le plus propre possible.



# Compilation croisée

On définit la localisation d'outils dans le fichier arm-toolchain.cmake:

```
set(CMAKE_SYSTEM_NAME Linux)

set(CMAKE_C_COMPILER /opt/arm-2009q1/bin/arm-none-linux-gnueabi-gcc)

set(CMAKE_CXX_COMPILER /opt/arm-2009q1/bin/arm-none-linux-gnueabi-gcc)

set(CMAKE_FIND_ROOT_PATH /opt/arm-2009q1/arm-none-linux-gnueabi/)

# ajuste le comportement par défaut des commandes FIND_XXX():

set(CMAKE_FIND_ROOT_PATH $ENV{HOME}/rspi/rootfs)

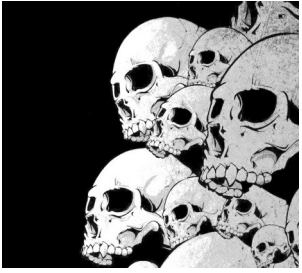
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)

set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)

set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)

$ cmake -DCMAKE_TOOLCHAIN_FILE=arm-toolchain.cmake
```





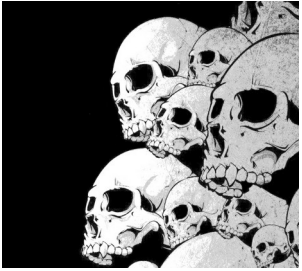
# External Project

Télécharger et installer un fichier de données:

```
include(ExternalProject)

if (NOT EXISTS ${CMAKE_BINARY_DIR}/DATA_TEST)
  make_directory(${CMAKE_BINARY_DIR}/DATA_TEST)
endif ()

ExternalProject_Add(EP_DATA_TEST
  PREFIX ${CMAKE_BINARY_DIR}/DATA_TEST
  URL http://www.coin-or.org/download/source/Data/Data-1.0.7.tgz
  PATCH_COMMAND ""
  CONFIGURE_COMMAND ""
  BUILD_COMMAND ""
  INSTALL_COMMAND ""
)
```



# External Project

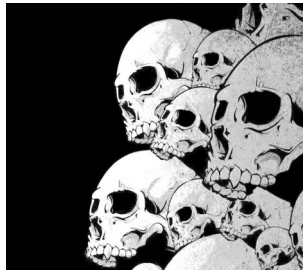
Télécharger / compiler / installer Zlib:

`cmake --help-module ExternalProject`

```
set(InstDir ${CMAKE_BINARY_DIR}/_dep)

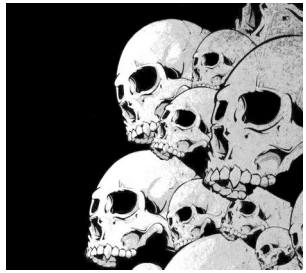
ExternalProject_Add(Zlib
    DEPENDS          EP_DATA_TEST
    PREFIX           ${InstDir}/Zlib
    URL              http://zlib.net/zlib-1.2.11.tar.gz
    PATCH_COMMAND   cp
    ${CMAKE_SOURCE_DIR}/cmake/CMakeLists_zlib.txt
    ${InstDir}/Zlib/src/Zlib/CMakeLists.txt
    CONFIGURE_COMMAND mkdir ${InstDir}/Zlib/src/Zlib/build
    & cd ${InstDir}/Zlib/src/Zlib/build cmake -
    DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=${InstallDir}/install ..
    BUILD_COMMAND   cd ${InstDir}/Zlib/src/Zlib/build make
    INSTALL_COMMAND cd ${InstDir}/Zlib/src/Zlib/build make
install
)
```





# CTest

# CTest

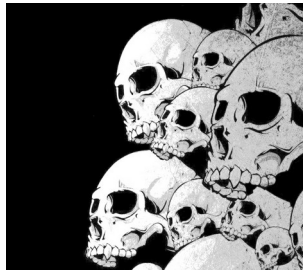


# CTest

A ajouter en début de CmakeLists.txt

```
Enable_Testing ()
```

Cette fonction va inclure et définir la variable BUILD\_TESTING



# CTest

Ajout d'un test avec quelques propriétés standards:

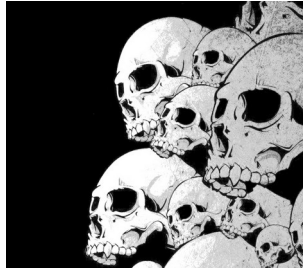
```
add_test(NAME infeas_chemcom
         COMMAND $<TARGET_FILE:glpsol> --mps ${FileData} -
         o ${FileOut})

set_tests_properties(infeas_chemcom PROPERTIES TIMEOUT 30)
set_tests_properties(infeas_chemcom PROPERTIES LABELS
"MPS,INFEAS")
set_tests_properties(infeas_chemcom PROPERTIES
PASS_REGULAR_EXPRESSION "Optimal Solution Found")
```

Lancement des tests dans le répertoire de build:

```
$ ctest -R infeas_chemcom -verbose
$ ctest -L INFEAS --verbose
```





# CTest

Définir des tests dépendants l'un de l'autre:

```
add_test(NAME infeas_chemcom
          COMMAND $<TARGET_FILE:glpsol> --mps ${FileData} -
o ${FileOut})

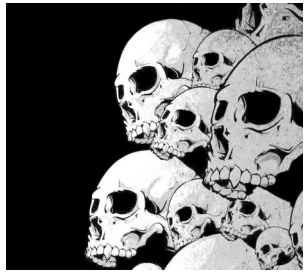
add_test(NAME infeas_chemcom_analyze
          COMMAND python3 analyze.py ${FileOut})

set_tests_properties(infeas_chemcom_analyze PROPERTIES
DEPENDS infeas_chemcom)
```



# CPack

# CPack



# CPack

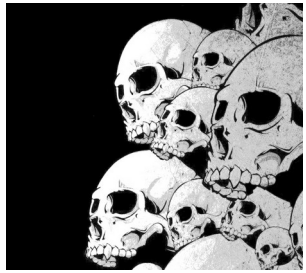
CPACK_GENERATOR	(ZIP, NSIS, RPM, DEB)
CPACK_PACKAGE_VENDOR	(GLPK)
CPACK_PACKAGE_NAME	(GLPK)
CPACK_PACKAGE_FILE_NAME	(g l p k - 4 . 6 5)
CPACK_PACKAGE_INSTALL_DIRECTORY	(/opt/g l p k - 4 . 6 5)
CPACK_PACKAGE_VERSION	(4.65.0)
CPACK_PACKAGE_VERSION_MAJOR	(4)
CPACK_PACKAGE_VERSION_MINOR	(65)
CPACK_PACKAGE_VERSION_PATCH	(0)
CPACK_PACKAGE_DESCRIPTION_FILE	(<PathTo>/releaseNotes.txt)
CPACK_RESOURCE_FILE_LICENSE	(<PathTo>/licenseAgreement.txt)
CPACK_RESOURCE_FILE_README	(<PathTo>/releaseNotes.txt)

Sous Windows : installer NSIS ou Wix pour le packaging. Ensuite :

```
$ make package
```

Ou

```
$ cpack -G "NSIS"
```



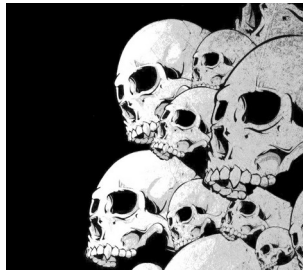
# CPack

Les fichiers qu'il est possible de customiser pour ajuster le comportement de l'installeur NSIS sous Windows:

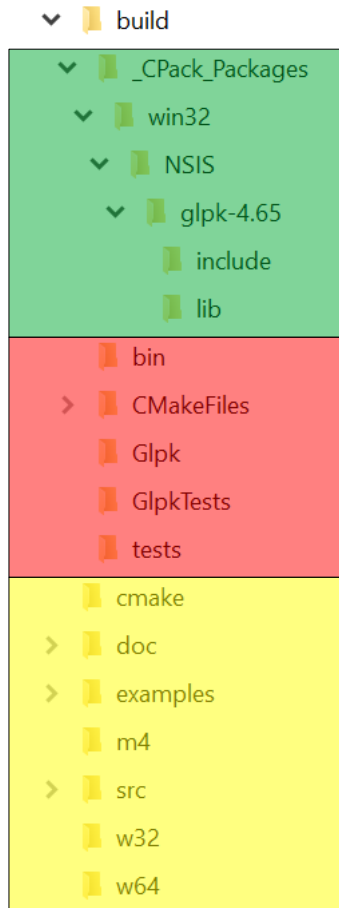
NSIS.InstallOptions.ini.in  
NSIS.template.in  
WIX.template.in  
CPack.NuGet.nuspec.in

Les générateurs disponibles :

7Z	= 7-Zip file format
DEB	= Debian packages
External	= CPack External packages
IFW	= Qt Installer Framework
NSIS	= Null Soft Installer
NSIS64	= Null Soft Installer (64-bit)
NuGet	= NuGet packages
STGZ	= Self extracting Tar GZip compression
TBZ2	= Tar BZip2 compression
TGZ	= Tar GZip compression
TXZ	= Tar XZ compression
TZ	= Tar Compress compression
WIX	= MSI file format via Wix tools
ZIP	= ZIP file format



# CPack



Répertoire Cmake de préparation du packaging.  
Sous Windows, en cas de souci, voir le fichier de log ainsi que le fichier project.nsi pour mieux identifier l'erreur.

Répertoire standard de build

Répertoire des fichiers source