ENVOL 2012 - TP

Création de composants logiciels avec CMake

Jean-Christophe Souplet

jcsouplet@lri.fr

Selon Présentation : « Introduction à CMake »

De Alexandre Abadie (IR SED INRIA-Saclay)

CMake

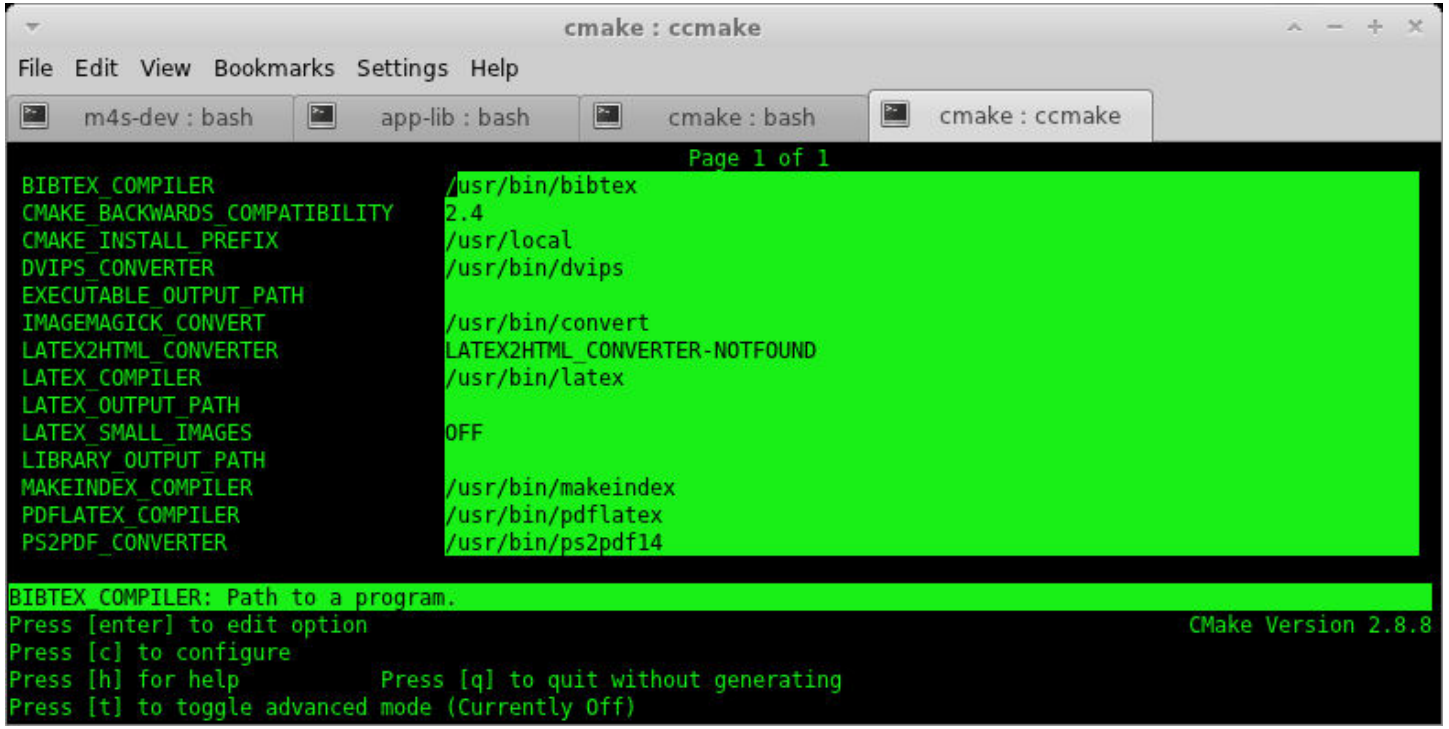
- Un générateur de Makefile
- Multiplateforme (Linux, Unix, Windows, MacOSX, Android, iOS, etc)
- Principalement utilisé pour les projets C++, C, Fortran
- Indépendant du compilateur (gcc, llvm, VC, f90, etc)
- Opensource (Kitware)
- Projets utilisant CMake
  - Kde, Boost, VTK, ITK, …

Installation

- Sous windows/mac en utilisant l'installeur
- Sous Linux en utilisant le gestionnaire de paquets
- A partir des sources (exemple pour linux/Mac) :
  - $ ./bootstrap
  - $ make
  - $ sudo make install

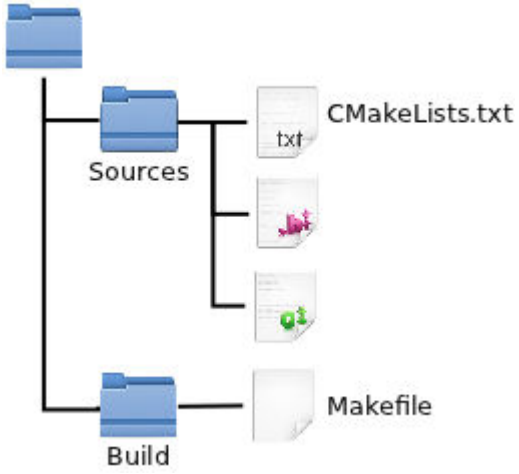
3 manières d'utiliser CMake

- En ligne de commande : cmake
- Avec l'interface curses : ccmake
- Avec l'interface graphique : cmake-gui



In source/out source builds

- On préférera toujours utiliser le outsource builds. Pourquoi ?
  - Permet de garder son dossier source “propre”
  - Pour avoir plusieurs types de compilations (options différentes, cross-compilation, etc)
  - Evite les problèmes avec les gestionnaires de versions
- Exemple :



Commandes cmake associées :
$ cd Build
$ cmake ../Sources
$ cmake .

Utilisation d'un IDE

- CMake peut générer des projets pour certains IDE couramment utilisés
  - Visual Studio, Eclipse, Kdevelop, …
- Aujourd'hui, certains IDE supportent l'import de projets CMake
  - QtCreator, KDevelop4, …
- Pour connaître la liste des générateurs disponibles pour votre plateforme :
  - $ cmake -h

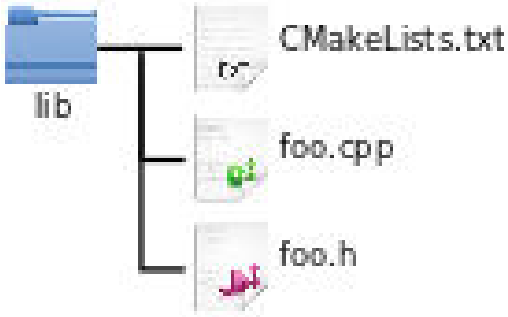
Exemple d'un exécutable


```
1 cmake_minimum_required (VERSION 2 . 8 )
2 p r o j e c t (MYAPP)
3
4 add_executable (myapp main . cpp )
```

- Compilation :

```
[ubuntu]:/cmake-exemples/app> make
Scanning dependencies of target app
[100%] Building CXX object CMakeFiles/app.dir/main.cpp.o
Linking CXX executable bin/app
[100%] Built target app
```

Exemple d'une bibliothèque




```
1 cmake_minimum_required (VERSION 2 . 8 )
2 p r o j e c t (MYLIB)
3
4 add_l ibrar y ( foo SHARED foo . cpp )
```

- Compilation

```
[ubuntu]:/cmake-exemples/lib> make
[100%] Building CXX object CMakeFiles/foo.dir/foo.cpp.o
Linking CXX shared library libfoo.so
[100%] Built target foo
```

Exercice 1 : Utiliser CMake

- But
  - Comprendre l'utilisation de CMake en tant qu'outil de génération de Makefile
  - Avoir un aperçu des fichiers générés par CMake
  - Se familiariser avec la notion d'outsource build
- En mode graphique
  - Aller dans le dossier exercice1 et prendre connaissance de son contenu
    - un fichier main.cpp
    - un fichier CMakeLists.txt
  - Lancer CMake (en mode GUI)
  - Remplir le champ Source directory

Exercice 1 : Utiliser CMake

- Remplir le champ Build directory (créer un dossier build, cf outsource build)
- Cliquer sur Configure et choisir parmi les générateurs proposés
  - Linux/Mac => Unix Makefile,
  - Windows => VisualStudio ou Mingw
- Cliquer sur Generate
- Compiler (make dans le dossier de build sous mac et linux, générer la solution sous visual studio)

- En ligne de commande
  - Reprendre les étapes précédentes en ligne de commande. On utilisera ccmake pour générer le makefile.

Le langage CMake

- CMake fournit un langage de script simple pour :
  - Gérer les sources de son projet
  - Importer des bibliothèques externes
  - Assurer le portabilité de son code, etc

=> Liste de variables, commandes internes, structures de contrôle

Variables CMake

- Le type de base dans CMake est la **chaine de caractères** (string)
- Pour définir une variable on utilise le mot clé **set**
  - **set** (Foo "abc")
  - **set** (Foo a b c)
- L'accès à la valeur d'une variable se fait avec la syntaxe suivante : ${Foo}
- Voir la liste complète
  http://www.cmake.org/Wiki/CMake_Useful_Variables

Variables importantes

- **PROJECT_SOURCE_DIR** : Le dossier contenant les sources de votre projet
- **PROJECT_BINARY_DIR** : Le dossier de build de votre projet
- **EXECUTABLE_OUTPUT_PATH** : Le dossier où seront générés les exécutables
- **LIBRARY_OUTPUT_PATH** : Le dossier où seront générées les bibliothèques
- **BUILD_SHARED_LIB** : Compilation des bibliothèques en mode statique ou dynamique (OFF ou ON)
- **CMAKE_BUILD_TYPE** : Type de compilation du projet (Release, Debug, RelWithDebInfo)

Les structures de contrôle

- Le langage de script de CMake fournit les structures basiques de contrôle
  - **if then else**
  - **while**
  - **foreach**
  - **macro, function**
- Exemple **foreach**
  - **s e t** (VAR a b c)
  - **foreach** (f ${VAR})
    - **message** (${ f })
  - **endforeach** (f)

Les structures de contrôle

- Exemple **if**
  - **i f** (v a r)
    - some_command (. . .)
  - **e n d i f** (v a r)
  - Les commandes seront appelées si var est différent de vide, 0, N, NO, OFF, FALSE, NOTFOUND, or –NOTFOUND
- Exemple **macro**
  - **macro** (h e l l o **MESSAGE**)
    - **message** (${**MESSAGE**})
  - **endmacro** (h e l l o)
  - h e l l o ("hello world") *# appel de la macro*

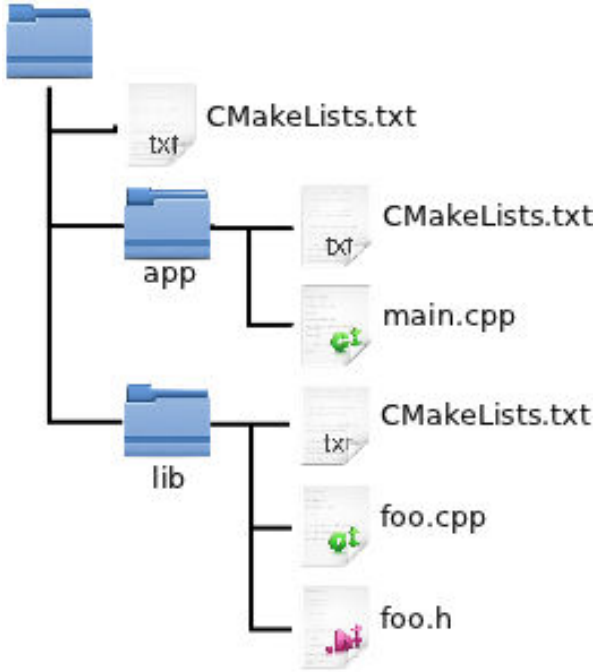
Quelques commandes utiles

- **add_subdirectory**
- **target_link_libraries**
- **link_directories**
- **option**
- **add_definitions**
- **configure_file**
- **message**
- **include**

- Voir la documentation

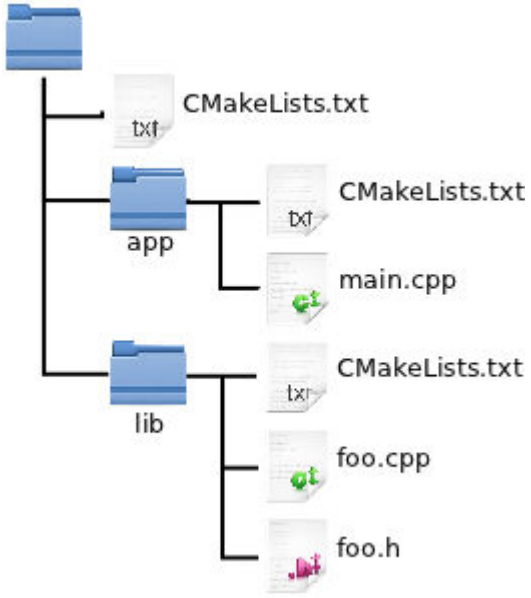
http://www.cmake.org/cmake/help/v2.8.8/cmake.html#section_Commands

Exemple plus complet



- CMakeLists.txt (lib) :
- **add_library** (foo SHARED foo.cpp)

Exemple plus complet



- CMakeLists.txt (app) :
- **add_executable** (app main.cpp)
- **target_link_libraries** (app foo)

Exemple plus complet

- CMakeLists.txt (principal) :
- **cmake_minimum_required** (**VERSION** 2.8)
- **project**(MYPROJECT)
- **set**(**EXECUTABLE_OUTPUT_PATH** ${**PROJECT_BINARY_DIR**}/bin)
- **set**(**LIBRARY_OUTPUT_PATH** ${**PROJECT_BINARY_DIR**}/lib)
- **include_directories**(${**PROJECT_SOURCE_DIR**}/foo)
- **add_subdirectory** (foo)
- **add_subdirectory** (app)

- Compilation et exécution :

```
[ubuntu]:/cmake-exemples/app-lib> cmake .
-- Configuring done
-- Generating done
-- Build files have been written to: /home/aabadie/SED/Seminaires/cmake-exemples/app-lib
[ubuntu]:/cmake-exemples/app-lib> make
[ 50%] Building CXX object foo/CMakeFiles/foo.dir/foo.cpp.o
Linking CXX shared library ../lib/libfoo.so
[ 50%] Built target foo
[100%] Building CXX object app/CMakeFiles/app.dir/main.cpp.o
Linking CXX executable ../bin/app
[100%] Built target app
[ubuntu]:/cmake-exemples/app-lib> bin/app
Ma première application avecCMake utilisant la lib foo
Nouvelle description de la lib foo
[ubuntu]:/cmake-exemples/app-lib>
```

Exercice 2

- But
  - Ecrire son premier CMakeLists.txt pour générer un makefile (ou une solution Visual Studio).
- On utilisera les commandes CMake
  - add_executable,
  - include_directories,
  - add_library,
  - option,
  - link_directories et target_link_libraries

Exercice 2

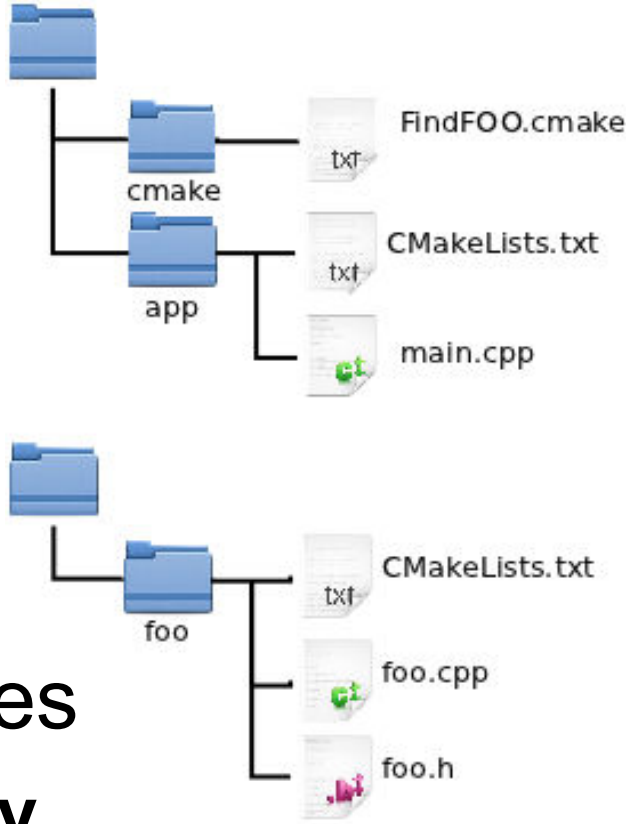
1. Prendre connaissance du contenu du dossier
   - fichiers main.cpp, foo.h, foo.cpp
2. Ecrire un fichier CMakeLists.txt pour générer un binaire à partir de main.cpp et foo.cpp
3. Générer le makefile, compiler, exécuter
4. Reprendre les 2 points précédents en générant une bibliothèque foo à partir de foo.cpp et foo.h
5. (On pourra tester avec $BUILD_SHARED_LIBS à ON et OFF).

Import de biblio externes

- Se fait par la commande **find_package**.
- Exemple :
  - **find_package** (VTK REQUIRED)
  - **if**(VTK_FOUND)
  - **include**(${VTK_USE_FILE})
  - **endif**(VTK_FOUND)
- La distribution de CMake fournit des scripts de recherche de packages pour un certain nombre de bibliothèques externes :
  - VTK, ITK, Boost, Qt, OpenCV, etc,
  - Mais parfois il faut écrire son propre script de recherche !

Ecriture FindXXX.cmake

- Marche à suivre :
  - Ecrire un fichier FindXXX.cmake
  - Dire à cmake où aller le chercher
  - Importer votre lib
- Commandes cmake utiles
  - **find_path** et **find_library**
  - Exemple avec lib foo

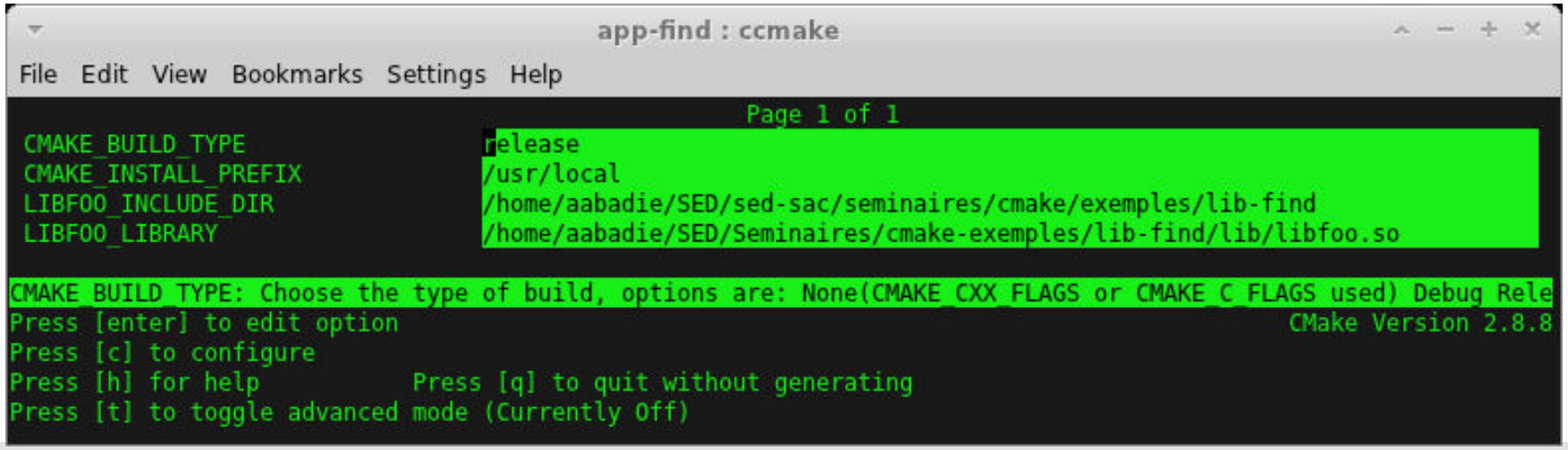


Ecriture FindXXX.cmake

- FindFOO.cmake
  - **find_path** (LIBFOO_INCLUDE_DIR foo.h HINTS /usr/**include**/foo PATH_SUFFIXES foo)
  - **find_library**(LIBFOO_LIBRARY foo HINTS /usr/lib)
- **set**(LIBFOO_LIBRARIES ${LIBFOO_LIBRARY})
- **set**(LIBFOO_INCLUDE_DIRS ${LIBFOO_INCLUDE_DIR})

Ecriture FindXXX.cmake

- Dire à cmake où aller le chercher
  - On utilise **CMAKE_MODULE_PATH**
  - **set** (**CMAKE_MODULE_PATH** ${**CMAKE_MODULE_PATH**} "${CMAKE_SOURCE_DIR}/cmake")
- Importer la lib FOO
  - **find_package** (FOO)



Exercice 3

- But
  - Définir la bibliothèque Foo comme une bibliothèque externe qui sera importée pour compiler le projet exercice3
- On utilisera les commandes find_package, find_path et find_library
- Aller dans le dossier exercice3 et prendre connaissance de son contenu
  - un dossier app qui contient le projet exo3
  - un dossier lib qui contient le projet FOO

Exercice 3

- Ecrire le fichier CMakeLists.txt nécessaire pour générer le projet FOO qui correspond à la bibliothèque foo (on préfèrera une bibliothèque partagée)
- Générer et compiler la bibliothèque foo dans un sous dossier 'build du dossier foo
- Vérifier que tout est compilé (il doit y avoir un fichier libfoo.so quelque part dans le dossier de build, ou dans un sous dossier de build appelé lib)
- Remplir le fichier app/FindFOO.cmake

Exercice 3

- Ecrire le fichier CMakeLists.txt pour générer le projet exercice3 (l'exécutable portera le même nom)
- Dans un sous dossier build du dossier app, générer et compiler le makefile (attention, il faut renseigner des chemins pour que la configuration et la compilation marchent)

Reference

- Alexandre Abadie

IR SED INRIA-Saclay
Alexandre.Abadie@inria.fr