Introduction à CMake

SED
INRIA Saclay
Alexandre.Abadie@inria.fr

10 décembre 2012

Sommaire

1 Généralités

CMake est...

- Un generateur de Makefile
- Multiplateforme (Linux, Unix, Windows, MacOSX, Android, iOS, etc)
- Principalement utilisé pour les projets C++, C, Fortran
- Indépendent du compilateur (gcc, llvm, VC, f90, etc)
- Opensource (Kitware)

Projets utilisant CMake

- Kde
- Compiz
- Boost
- VTK, ITK
- Second Life

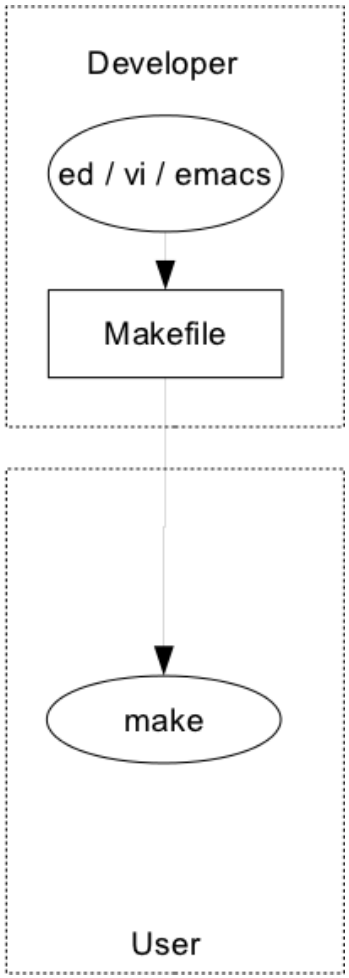
Installation

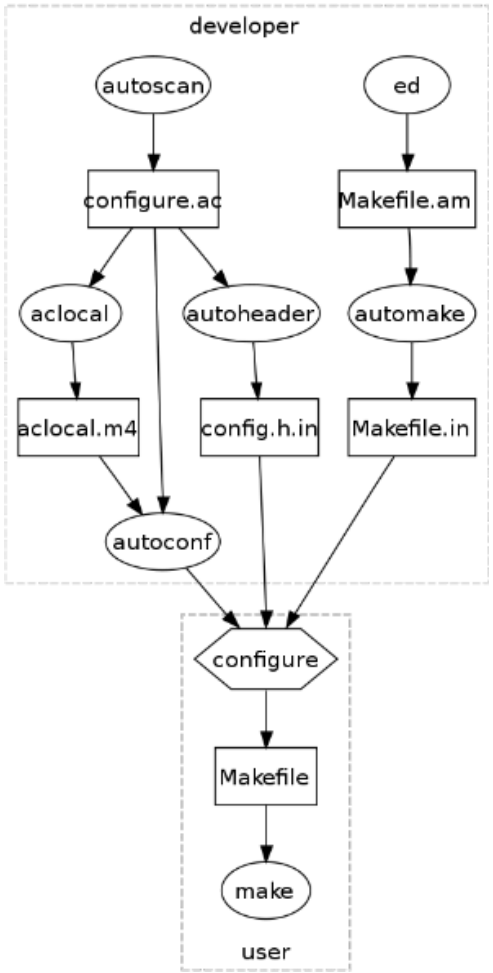
- Sous windows/mac en utilisant l'installeur
- Sous Linux en utilisant le gestionnaire de paquets
- A partir des sources (exemple pour linux/Mac) :

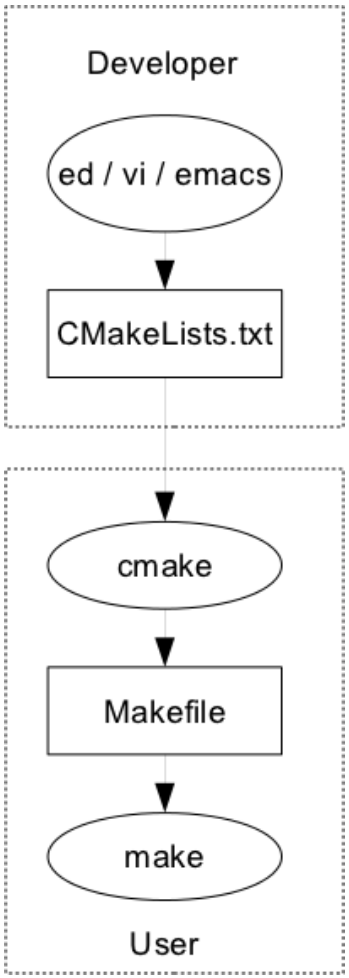
```
$ ./bootstrap
$ make
$ sudo make install
```

CMake et les autres

Comparaison avec d'autres générateurs de makefile :





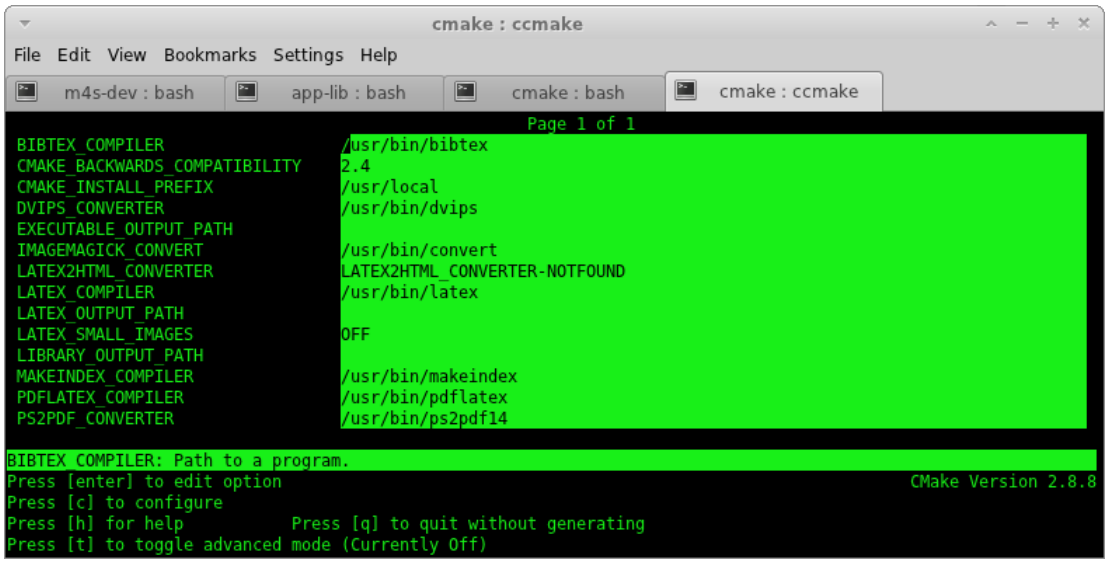


2
Les bases de CMake

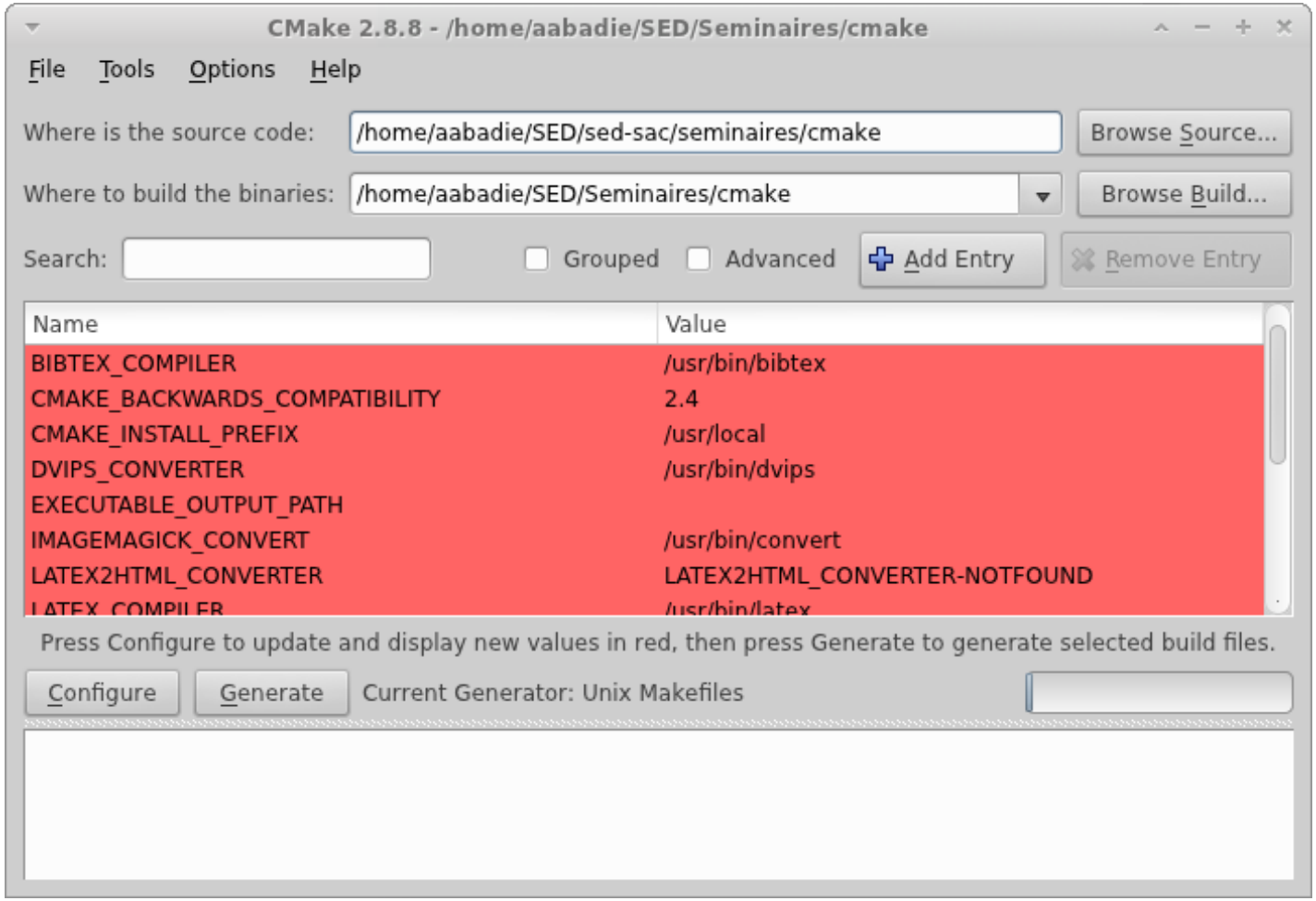
Utilisation de CMake

CMake peut s'utiliser de 3 manières :

- En ligne de commande : cmake
- Avec l'interface curses : ccmake
- Avec l'interface graphique (Qt)



Utilisation de CMake

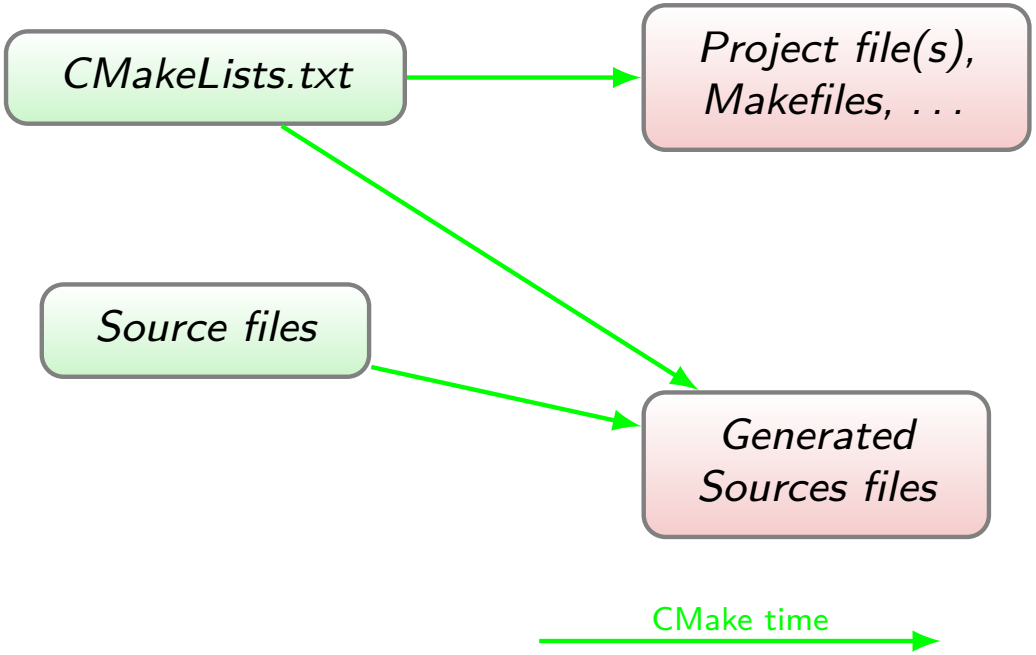


Utilisation de CMake

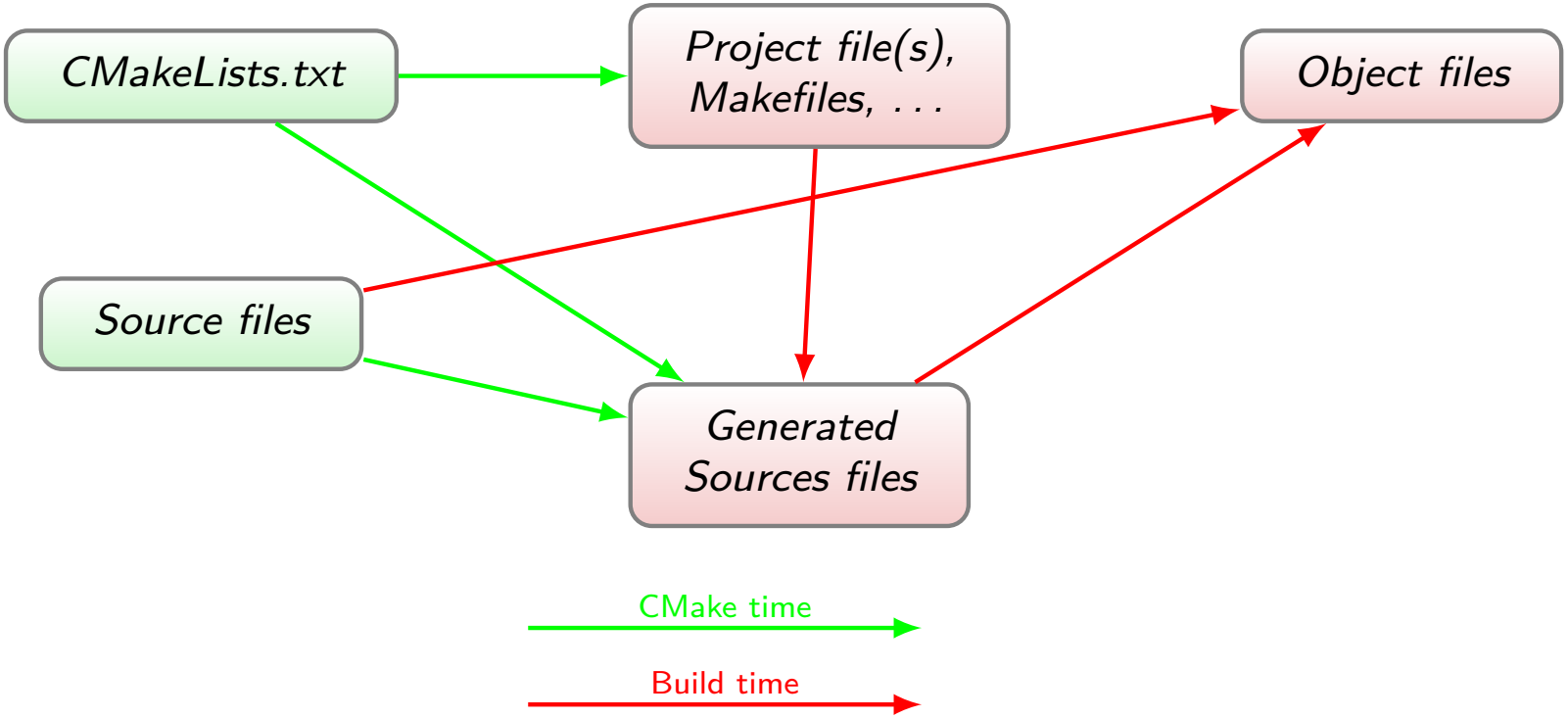
CMakeLists.txt

Source files

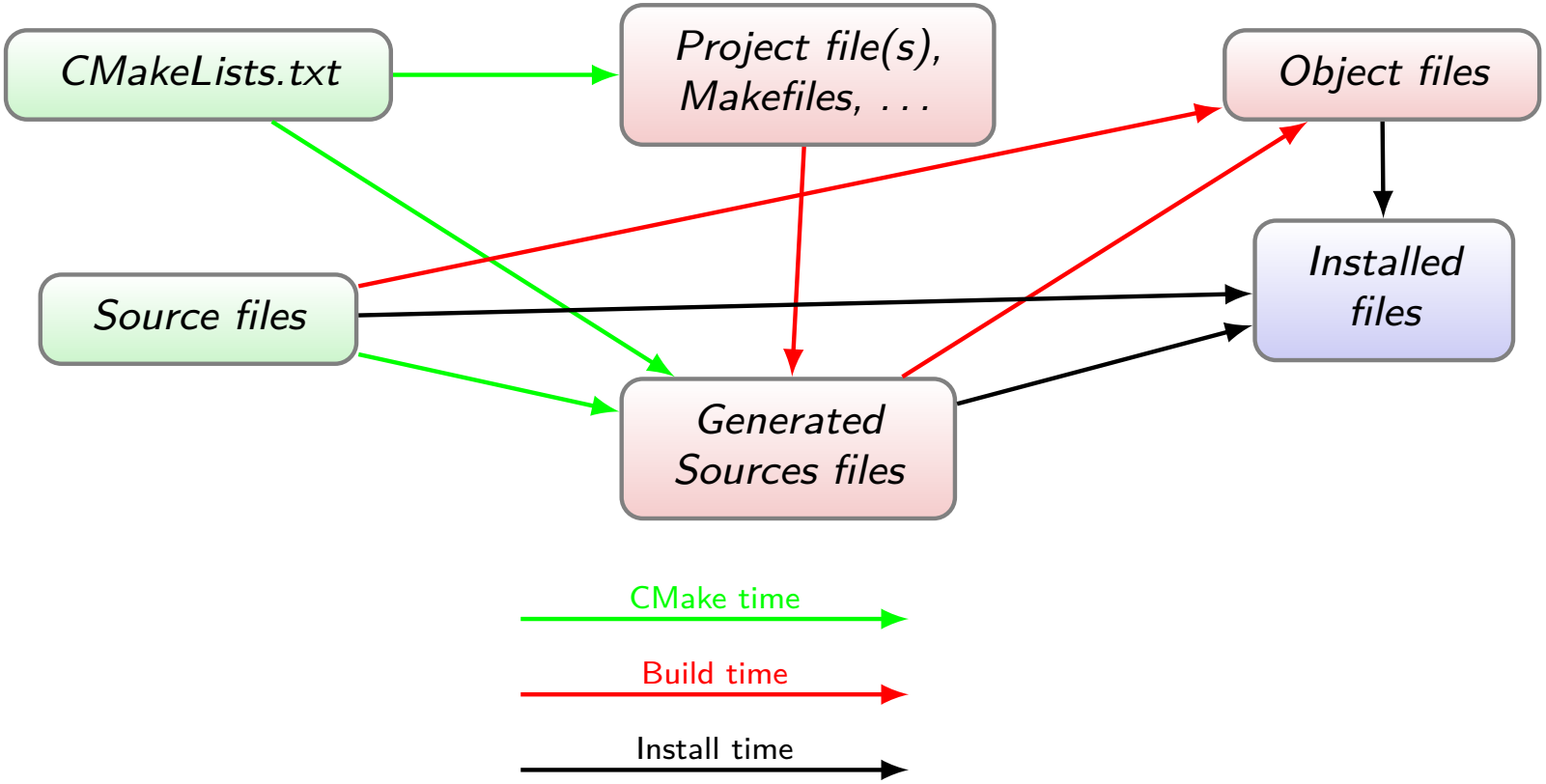
Utilisation de CMake



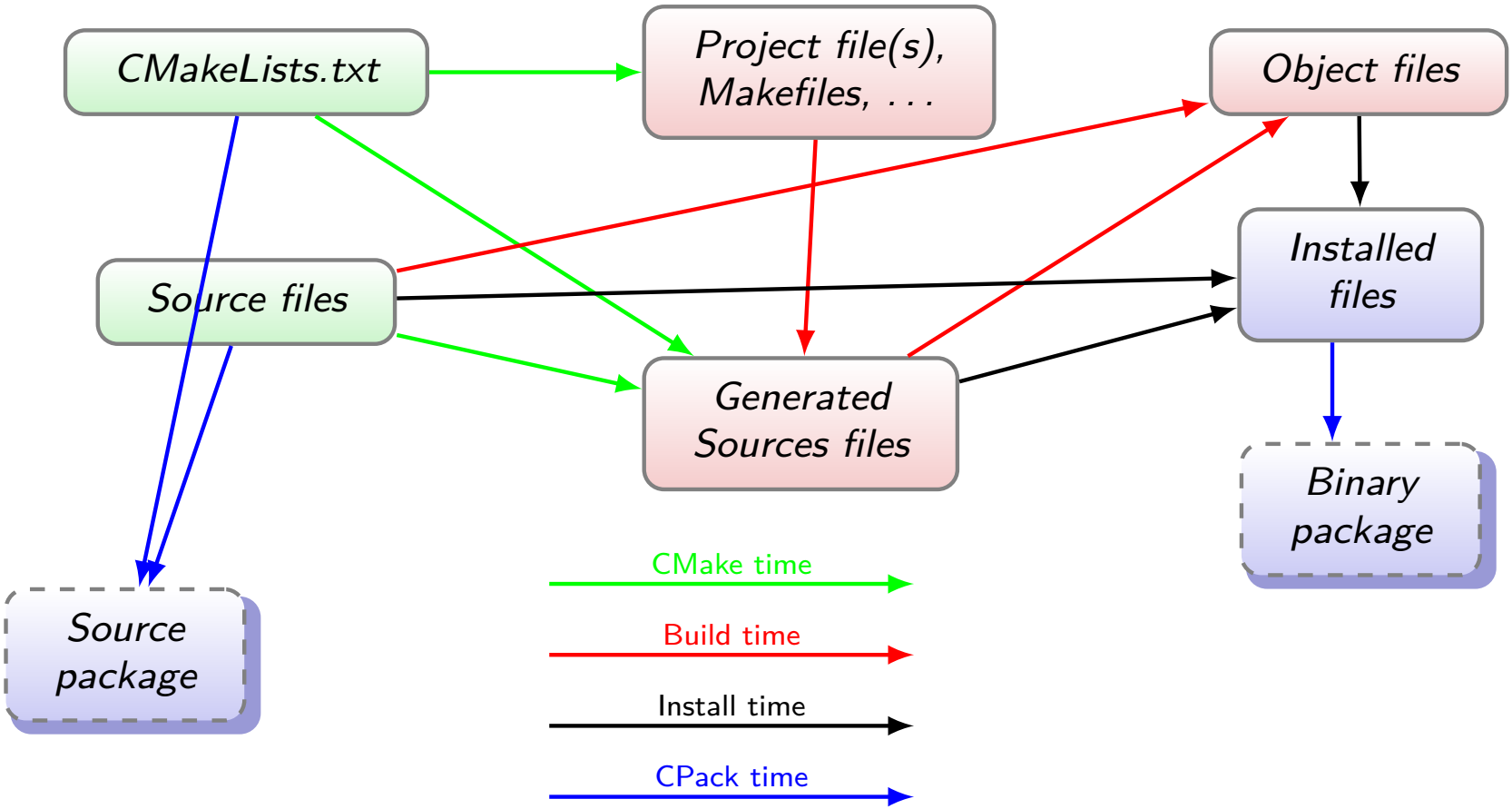
Utilisation de CMake



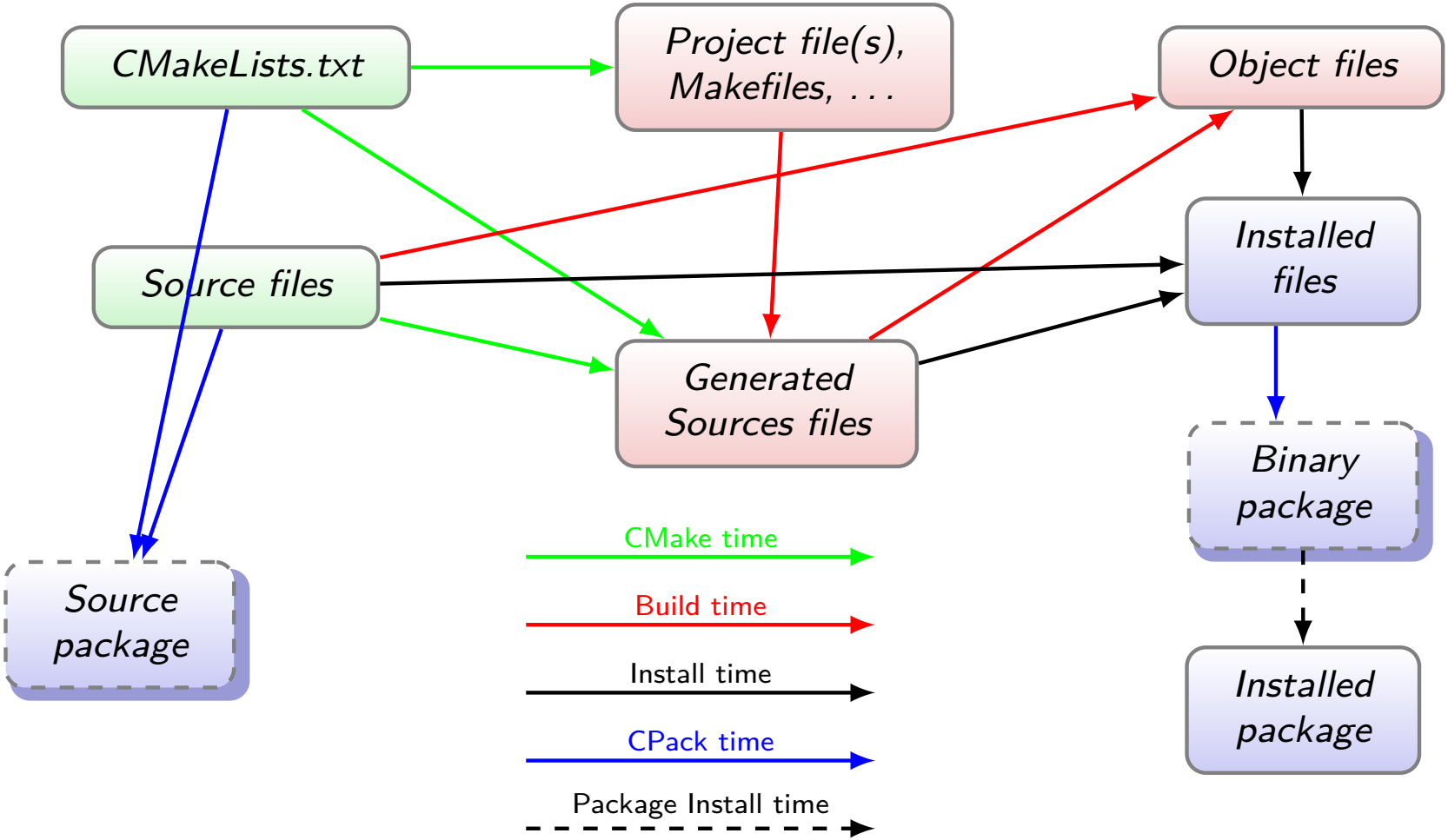
Utilisation de CMake



Utilisation de CMake



Utilisation de CMake

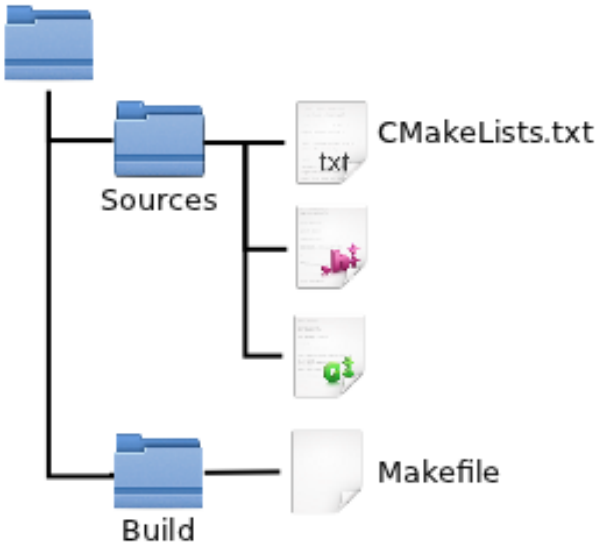


In source/out source builds

On préférera toujours utiliser le outsource builds. Pourquoi ?

- Permet de garder son dossier source “propre”
- Pour avoir plusieurs types de compilation (options différentes, cross-compilation, etc)
- Evite les problèmes avec les gestionnaires de versions

Exemple :



Commandes cmake associées :

```
$ cd Build
$ cmake ../Sources
$ cmake .
$ make
```

Utilisation d'un IDE

CMake peut générer des projets pour certains IDE couramment utilisés :

- Visual studio
- Eclipse
- Xcode
- Kdevelop

Aujourd'hui, certains IDE supporte l'import de projets CMake :

- QtCreator
- KDevelop4

Pour connaitre la liste des générateurs disponibles pour votre plateforme :

```
$ cmake -h
```

Exercice 1

Cas pratiques

Quelques exemples simples de projets gérés par CMake :

- Un executable
- Une bibliothèque

Un exécutable simple avec CMake


```
1 cmake_minimum_required (VERSION 2.8)
2 project (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
```

Une bibliothèque simple avec CMake


```
1 cmake_minimum_required (VERSION 2.8)
2 project (MYLIB)
3
4 add_library (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
```

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 contôle

Variables CMake

Le type de base dans CMake est la **chaine de caractère** (string)
Pour définir une variable on utilise le mot clé **set** :

```
1 set(Foo "abc")
2 set(Foo a b c)
```

L'accès à la valeur d'une variable se fait avec la syntaxe suivante :

```
1 ${Foo}
```

Quelques variables importantes

CMake définit des variables internes utiles :

- **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 générés les bibliothèques
- **BUILD_SHARED_LIB** : Compilation des bibliothèques en mode statique ou partagées (OFF ou ON)
- **CMAKE_BUILD_TYPE** : Type de compilation du projet (Release, Debug, RelWithDebInfo)

Les variables internes de CMake :

Voir la liste complète :

http://www.cmake.org/Wiki/CMake_Useful_Variables

Les structures de controle

Le langage de script de CMake fournit les structures de contrôle basiques :

- **if then else**
- **while**
- **foreach**
- **macro, function**

Exemple :

```
1 set(VAR a b c)
2 foreach(f ${VAR})
3     message(${f})
4 endforeach(f)
```

Les structures de controle

Exemple avec **if** :

```
1 if(var)
2     some_command(...)
3 endif(var)
```

les commandes seront appelées si var est différent de :
vide, 0, N, NO, OFF, FALSE, NOTFOUND, or -NOTFOUND

Les structures de controle

Exemple avec **macro** :

```
1 macro(hello MESSAGE)
2       message(${MESSAGE})
3 endmacro(hello)
4
5 # appel de la macro
6 hello("hello world")
```

Quelques commandes utiles de CMake

```
1 add_subdirectory
2 target_link_libraries
3 link_directories
4 option
5 add_definitions
6 configure_file
7 message
8 include
```

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

Configurer son projet en ligne de commandes

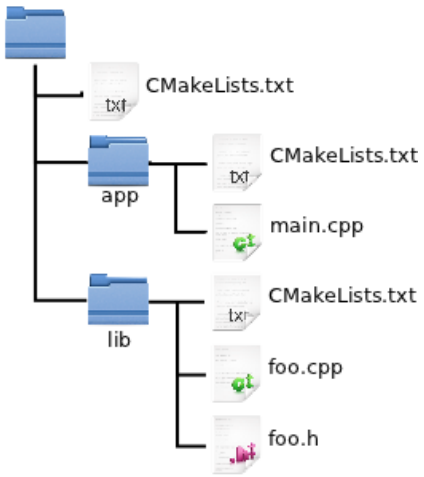
Il est possible de changer des variables de votre projet CMake à partir de la ligne de commande :

```
cmake -DVARIABLE:TYPE=VALUE
```

Exemple :

```
$ cmake -DBUILD_SHARED_LIBS=ON .
$ cmake -DCMAKE_BUILD_TYPE=Release .
```

Exemple plus complet



CMakeLists.txt (lib) :

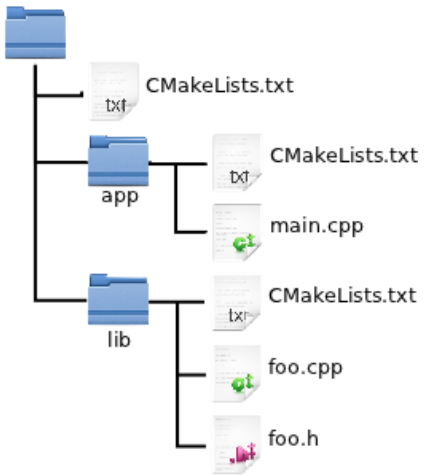
```
1 add_library(foo SHARED foo.cpp)
```

Exemple plus complet

foo.h :

```
1  #include <string>
2
3  class Foo {
4  public:
5    Foo() : m_description("description␣par␣defaut") {};
6    ~Foo() {};
7
8    void setDescription(const std::string & description);
9    std::string description(void) const;
10
11 private:
12   std::string m_description;
13 };
```

Exemple plus complet



CMakeLists.txt (app) :

```
1 add_executable(app main.cpp)
2 target_link_libraries(app foo)
```

Exemple plus complet

main.cpp :

```
1  #include <iostream>
2  #include <foo.h>
3
4  int main (int argc, char * argv[]) {
5    std::cout << "Ma premiere application avec"
6              << "CMake utilisant la lib foo"
7              << std::endl;
8
9    Foo * foo = new Foo();
10   foo->setDescription("Nouvelle description
11 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣de la lib foo");
12
13   std::cout << foo->description() << std::endl;
14
15   delete foo;
16   return 0;
17 }
```

Exemple plus complet

CMakeLists.txt (principal) :

```
1  cmake_minimum_required (VERSION 2.8)
2
3  project (MYPROJECT)
4
5  set (EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/bin)
6  set (LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
7
8  include_directories (${PROJECT_SOURCE_DIR}/foo)
9
10 add_subdirectory (foo)
11 add_subdirectory (app)
```

Exemple plus complet

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

3
Utilisation avancée

Utilisation avancée

- Import de bibliothèques externes
- CMake pour une application Qt

Import de bibliothèques externes

L'import de bibliothèques externes se fait par l'utilisation de la commande **find_package**.
Exemple :

```
1 find_package(VTK  REQUIRED)
2 if(VTK_FOUND)
3       include(${VTK_USE_FILE})
4 endif(VTK_FOUND)
```

La distribution de CMake fournit des scripts de recherche de package pour un certains nombre de bibliothèques externes : VTK, ITK, Boost, Qt, OpenCV, etc, etc

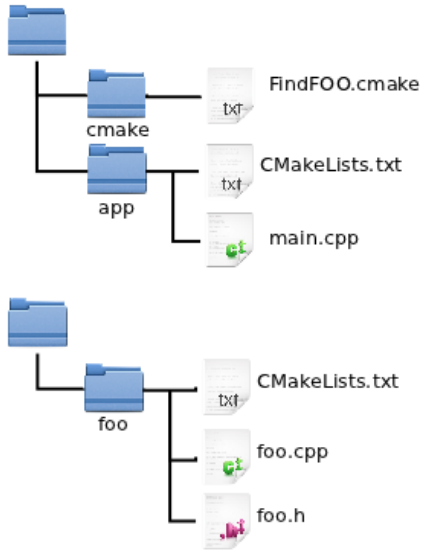
Mais parfois il faut écrire son propre script de recherche !

Import de bibliothèques externes

Marche à suivre :

1. Ecrire un fichier FindXXX.cmake
2. Dire à cmake où aller le chercher
3. Importer votre lib

Commandes cmake utiles : **find_path** et **find_library**
Exemple avec lib foo



Ecriture d'un fichier FindXXX.cmake

1. FindFOO.cmake :

```
1  find_path(LIBFOO_INCLUDE_DIR foo.h
2            HINTS /usr/include/foo
3            PATH_SUFFIXES foo )
4
5  find_library(LIBFOO_LIBRARY foo
6               HINTS /usr/lib)
7
8  set(LIBFOO_LIBRARIES ${LIBFOO_LIBRARY} )
9  set(LIBFOO_INCLUDE_DIRS ${LIBFOO_INCLUDE_DIR} )
```

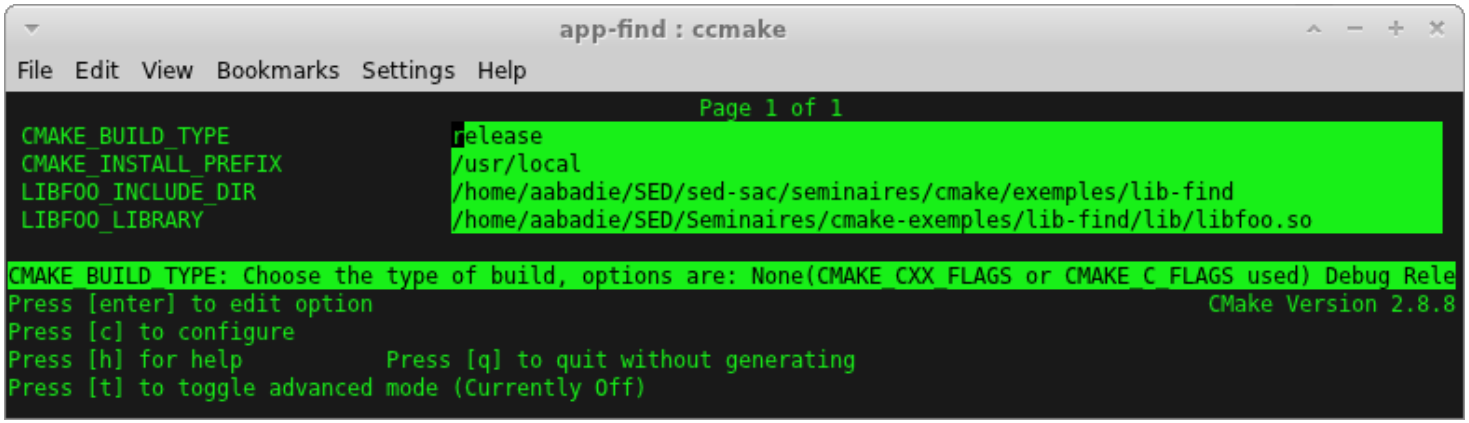
Ecriture d'un fichier FindXXX.cmake

2. Dire cmake où aller le chercher : on utilise la variable **CMAKE_MODULE_PATH**

```
1 set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH}
2                       "${CMAKE_SOURCE_DIR}/cmake")
```

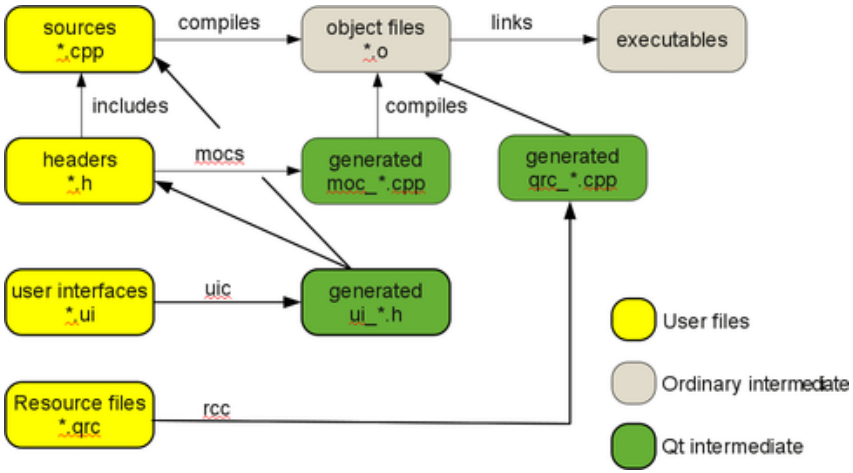
3. Importer la lib FOO :

```
1 find_package(FOO)
```



Exercice 3

CMake pour une application Qt

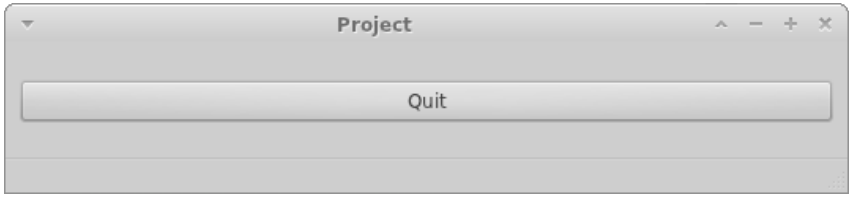


Commandes CMake :

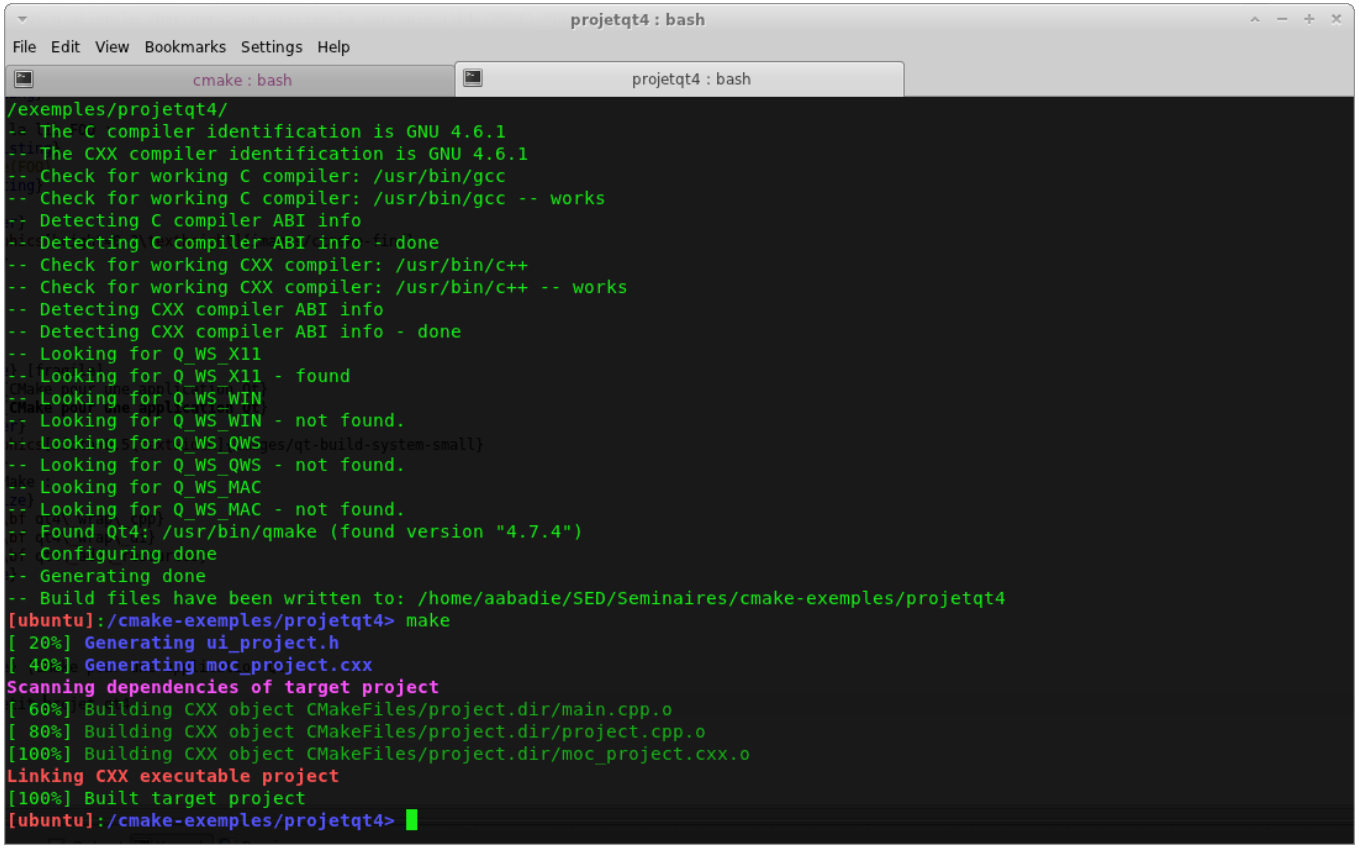
- **qt4_wrap_cpp**
- **qt4_wrap_ui**
- **qt4_add_resources**

CMake pour une application Qt

Exemple : petit projet qt4



Compilation



3
Tests et intégration continue avec CMake

Tests et intégration continue avec CMake

- CTest
- Ajout d'un test
- Integration continue

CTest

CTest est fourni avec CMake. Il permet d'automatiser :

- La configuration
- La compilation
- L'exécution des tests
- Les tests de mémoire
- Le calcul de la couverture du code
- La soumission des résultats sur un Dashboard (CDash)

Ajout d'un test

Commandes cmake :

- **enable_testing**
- **add_test**

Exemple :

```
1 include(CTest)
2 enable_testing()
3 add_test(myTest ${CMAKE_BINARY_DIR}/bin/myTest)
```

Pour écrires des classes de test :

- CppUnit
- QTest
- CxxTest
- assert

Paramètres d'exécution de CTest

- **$ ctest -R myTest** : lancer seulement le test myTest
- **$ ctest -V** : mode verbose
- **$ ctest -N** : obtenir la liste des tests

```
[ubuntu]:/cmake-exemples/tests> ctest -N
Test project /home/aabadie/SED/Seminaires/cmake-exemples/tests
  Test #1: myTest

Total Tests: 1
[ubuntu]:/cmake-exemples/tests> ctest -R myTest
Test project /home/aabadie/SED/Seminaires/cmake-exemples/tests
    Start 1: myTest
1/1 Test #1: myTest ...........................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 1
```

Les différents types de builds

CTest propose plusieurs type de soumissions (4 familles) :

- Experimental
- Continuous
- Nightly
- MemoryCheck

Exemple :

```
$ ctest -D Experimental
$ ctest -D Nightly
$ ctest -D Continuous
```

Test experimental

```
[ubuntu]:/cmake-exemples/tests> ctest -D Experimental
   Site: ubuntu
   Build name: Linux-c++
Create new tag: 20120620-1409 - Experimental
Configure project
   Each . represents 1024 bytes of output
    . Size of output: 0K
Build project
   Each symbol represents 1024 bytes of output.
   '!' represents an error and '*' a warning.
    . Size of output: 0K
   0 Compiler errors
   0 Compiler warnings
Test project /home/aabadie/SED/Seminaires/cmake-exemples/tests
    Start 1: myTest
1/1 Test #1: myTest ........................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.19 sec
Performing coverage
 Cannot find any coverage files. Ignoring Coverage request.
Submit files (using http)
   Using HTTP submit method
   Drop site:http://cdash.inria.fr/CDash/submit.php?project=cmake-tutorial
   Uploaded: /home/aabadie/SED/Seminaires/cmake-exemples/tests/Testing/20120620-1409/Build.xml
   Uploaded: /home/aabadie/SED/Seminaires/cmake-exemples/tests/Testing/20120620-1409/Configure.xml
   Uploaded: /home/aabadie/SED/Seminaires/cmake-exemples/tests/Testing/20120620-1409/Test.xml
   Submission successful
[ubuntu]:/cmake-exemples/tests>
```

Tester les fuites mémoire

CTest permet d'exécuter les outils **valgrind** ou **purify** pour détecter les fuites mémoires.

```
$ ctest -D MemoryCheck
$ ctest -D NightlyMemCheck
```

Calculer la couverture du code par les tests (Linux uniquement)

Pour calculer la couverture du code, il faut ajouter deux options de compilation supplémentaires :

```
1  add_definitions("-fprofile-arcs -ftest-coverage")
```

Et linker avec la bibliothèque **gcov** :

```
1  target_link_libraries(testedlib gcov)
```

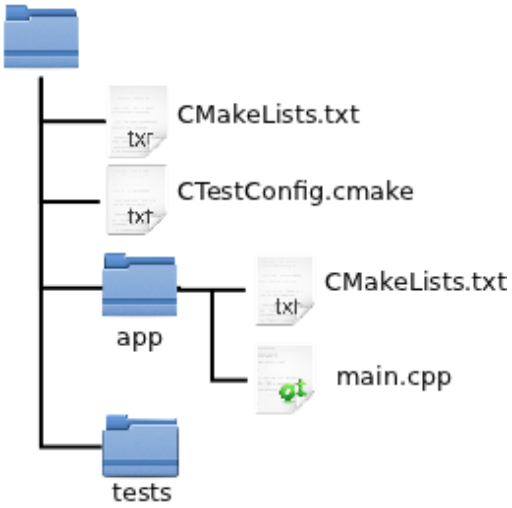
Integration continue

CTest permet de publier les résultats des tests sur un dashboard.
Plusieurs dashboard sont compatibles avec CMake :

- CDash
- Dart (obsolète)
- Jenkins (nécessite un plugin pour CMake)

On prendra l'exemple de CDash !
Voir http://cdash.inria.fr

Integration continue



CMakeLists.txt (top) :

```
1 include(CTest)
2 enable_testing()
```

CTestConfig.cmake :

```
1 set(CTEST_PROJECT_NAME "cmake-tutorial")
2 set(CTEST_NIGHTLY_START_TIME "00:00:00 EST")
3 set(CTEST_DROP_METHOD "http")
4 set(CTEST_DROP_SITE "cdash.inria.fr")
5 set(CTEST_DROP_LOCATION
6       "/CDash/submit.php?project=cmake-tutorial")
7 set(CTEST_DROP_SITE_CDASH TRUE)
```

CDash

No update data as of **Wednesday, June 20 2012 07:00:00 CEST** Help

[Show Filters]

Nightly

Site	Build Name	Update		Configure			Build			Test				Build Time
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min	
ubuntu	Linux-c++			0	0	0	0	0	0	0	0	1	0	2012-06-20T15:18:34 CEST
Totals	**1 Builds**	0	0	**0**	**0**	0	**0**	**0**	0	**0**	**0**	**1**	0	

Continuous

Site	Build Name	Update		Configure			Build			Test				Build Time
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min	
ubuntu	Linux-c++			0	0	0	0	0	0	0	0	1	0	2012-06-20T15:20:00 CEST
Totals	**1 Builds**	0	0	**0**	**0**	0	**0**	**0**	0	**0**	**0**	**1**	0	

Experimental

Site	Build Name	Update		Configure			Build			Test				Build Time
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min	
ubuntu	Linux-c++			0	0	0	0	0	0					2012-06-20T15:19:42 CEST
ubuntu	Linux-c++			0	0	0	0	0	0					2012-06-20T15:18:56 CEST
ubuntu	Linux-c++			0	0	0	0	0	0	0	0	1	0	2012-06-20T15:15:39 CEST
Totals	**3 Builds**	0	0	**0**	**0**	0	**0**	**0**	0	**0**	**0**	**1**	0	

No Coverage

Dynamic Analysis

Site	Build Name	Checker	Defect Count	Date
ubuntu	Linux-c++	Valgrind	0	2012-06-20T15:19:42 CEST
ubuntu	Linux-c++	Valgrind	0	2012-06-20T15:18:56 CEST

http://cdash.inria.fr

Exercice 4

3
Packaging avec CMake

Packaging avec CMake

CPack

CPack est un outil de packaging de logiciel **multiplateforme**
Il supporte plusieurs types de générateurs :

- Windows : NSIS, Cygwin
- MacOSX : DragNDrop, PackageMaker, OSXX11, Bundle
- Linux : Deb, RPM
- Archive contenant les binaires : zip, tz, tgz, stgz, tbz2

Exemple avec Linux

```
1  cmake_minimum_required(VERSION 2.8)
2
3  project(CMAKE_TUTORIAL)
4
5  add_executable(app main.cpp)
6  install(TARGETS app DESTINATION cmake-tutorial)
7
8  set(CPACK_GENERATOR "DEB")
9  # Mainteneur requis pour les paquets Debian
10 set(CPACK_DEBIAN_PACKAGE_MAINTAINER "Alexandre Abadie")
11
12 include(CPack)
```

Puis pour générer le package :

```
$ make package
$ sudo dpkg -i CMAKE-TUTORIAL-0.1.1-Linux.deb
```

CPack

```
[ubuntu]:/cmake-exemples/cpack> make
-- Configuring done
-- Generating done
-- Build files have been written to: /home/aabadie/SED/Seminaires/cmake-exemples/cpack
Scanning dependencies of target app
[100%] Building CXX object CMakeFiles/app.dir/main.cpp.o
Linking CXX executable bin/app
[100%] Built target app
[ubuntu]:/cmake-exemples/cpack> make package
[100%] Built target app
Run CPack packaging tool...
CPack: Create package using DEB
CPack: Install projects
CPack: - Run preinstall target for: CMAKE-TUTORIAL
CPack: - Install project: CMAKE-TUTORIAL
CPack: Create package
CPack: - package: /home/aabadie/SED/Seminaires/cmake-exemples/cpack/CMAKE-TUTORIAL-0.1.1-Linux.deb generate
d.
```

Exercice 5

3
Documentation et liens utiles

- CMake Wiki : www.cmake.org/Wiki/CMake
- Trouver les bibliothèques externes : http://www.cmake.org/Wiki/CMake:How_To_Find_Libraries
- Exposer/Packager sa bibliothèque : http://www.cmake.org/Wiki/CMake/Tutorials#CMake_Packages
- Utilisation de CTest : http://www.cmake.org/Wiki/CMake_Testing_With_CTest
- Packaging avec CPack : http://www.cmake.org/Wiki/CMake:Packaging_With_CPack
- CDash : http://www.cdash.org

MERCI

INRIA

SED SACLAY

www.inria.fr