lagadic

IRISA

institut de recherche en informatique et systèmes aléatoires

Builds et tests
Utilisation de CMake et de Dart

Séminaire Code & Travaux ASCII

Fabien Spindler
Projet Lagadic
IRISA / INRIA Rennes
http://www.irisa.fr/lagadic

16 novembre 2006

INRIA

CNRS

UNIVERSITE DE RENNES 1

Plan

1. CMake
   - Installation, principe de fonctionnement
   - Structure des sources
   - Exemple
   - *In-source* et *out-of-source build*
   - Utilisation, le cache, démonstration
2. CTest
   - Exemple, utilisation
3. Dart
   - Exemple, utilisation, *daily builds,* démonstration
4. CPack
   - Exemple, utilisation, démonstration

CMake

- http://www.cmake.org
- Outil d'aide au processus de compilation (*build*), multi plates-formes et open source
- Développé depuis 2001 par Kitware (VTK)
- Adopté par les développeurs de KDE
- Capable de générer des makefile unix, des projets MSVC, Borland… qui seront utilisés par les outils natifs lors de la compilation
- Nombreuses commandes permettant de localiser include, lib, exe
- Propose de nombreuses extensions pour localiser X, Qt, OpenGL…
- Propose des interfaces de test (CTest) et de *packaging* (CPack)

Installation

- Dernière version: 2.4.3
- Téléchargement: http://www.cmake.org/HTML/Download.html
- Version en cours de développement:

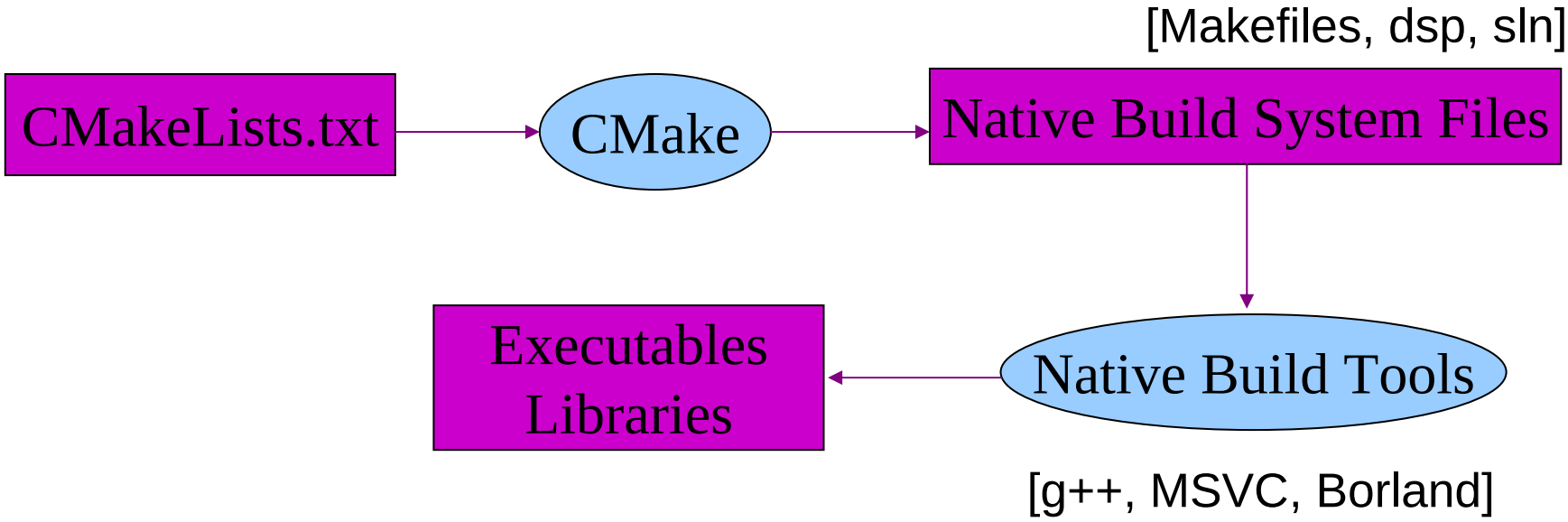
```
cvs -d :pserver:anonymous@www.cmake.org:/cvsroot/CMake login
(respond with password cmake)

cd /tmp
cvs -d :pserver:anonymous@www.cmake.org:/cvsroot/CMake co CMake

cd /tmp/CMake
./bootstrap –prefix=/usr
make
make install
```

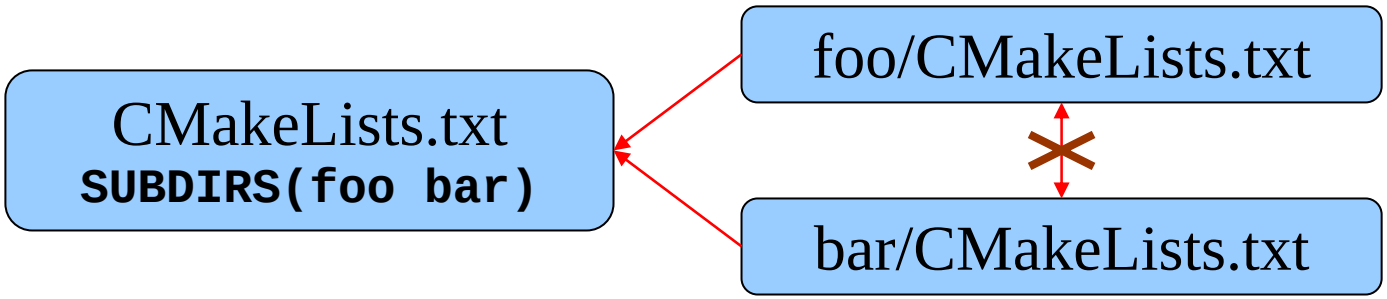
Principe de fonctionnement

- Capable de gérer le processus de compilation indépendamment de l'OS et du compilateur
- Connaissance de très nombreuses plates-formes et outils
- L'utilisateur configure son *build* avec CMake



Structure des sources

- Certains répertoires contiennent un fichier de description CMakeLists.txt
- Sous-répertoires spécifiées par la commande SUBDIRS
- Les répertoires dépendent uniquement des parents
- Certaines commandes sont transmises aux descendants



Eléments de syntaxe – CMakeLists.txt

- Langage de type script
  - Commentaire
  - Commande
  - Liste
  - Variable
  - Sructure de contrôle
- Configuration dynamique
  - config.h.in
- Détection de librairie

```
# Comment ends at a newline
```

```
COMMAND(arg1 arg2 ...)
```

```
A;B;C # semicolon separated
```

```
${VAR}
```

```
IF(CONDITION) FOREACH(v A B C)
```

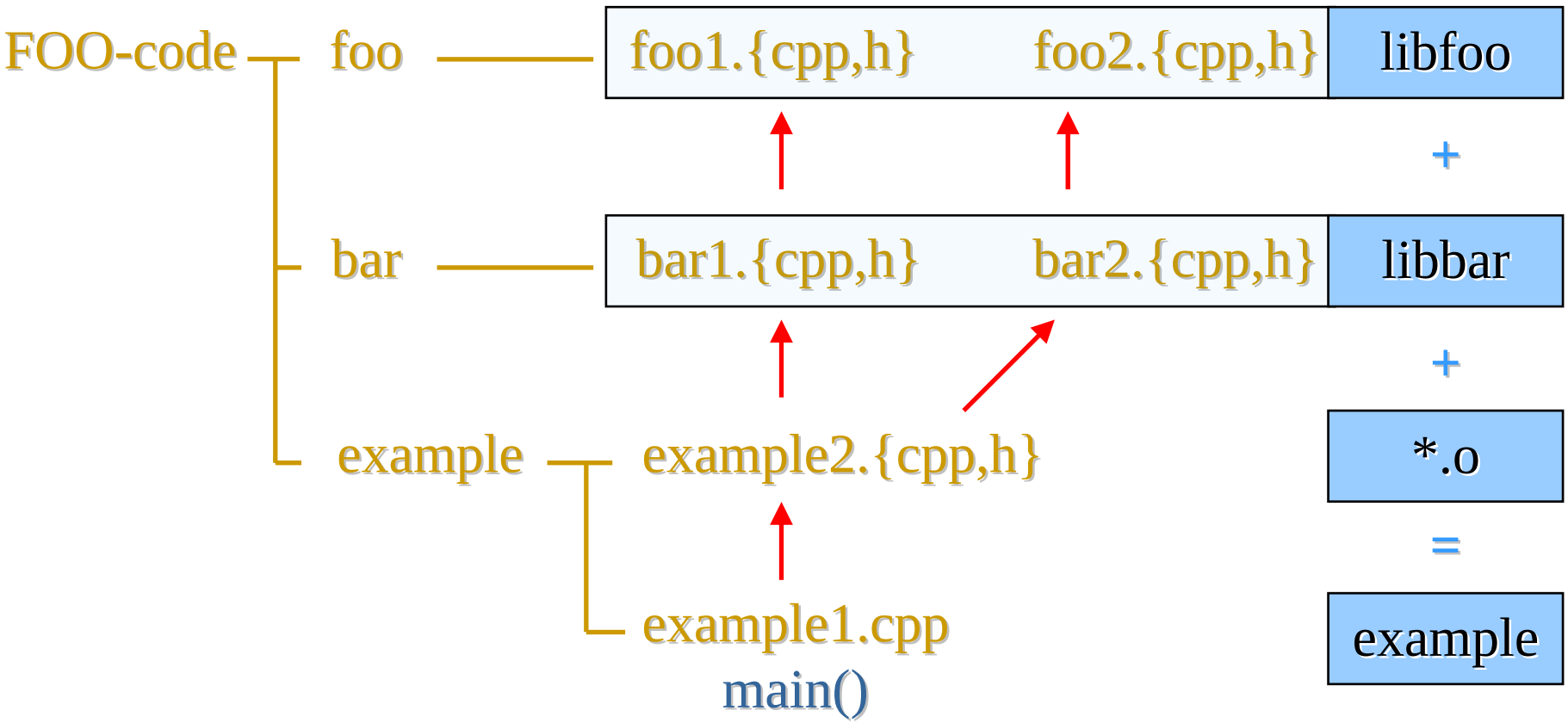
```
CONFIGURE_FILE(config.h.in config.h)
```

```
#cmakedefine FOO_VERSION ${FOO_VERSION}
#cmakedefine BUILD_SHARED_LIBS
```

```
FIND_LIBRARY(MY_LIB NAMES my1 my2
                         PATHS /foo c:/bar)
```

- Evalué lors des phases de configuration de CMake

Exemple : Le projet FOO 1/4



Exemple : Configuration minimale 2/4

```
PROJECT(projectname [CXX] [C] [JAVA])
```

FOO-code — CMakeLists.txt

```
PROJECT(FOO)
SUBDIRS(foo bar example)
INCLUDE_DIRECTORIES(${CMAKE_SOURCE_DIR}/foo
  ${CMAKE_SOURCE_DIR}/bar)
SET(LIBRARY_OUTPUT_PATH ${FOO_BINARY_DIR}/lib)
```

foo - CMakeLists.txt

```
ADD_LIBRARY(foo foo1.cpp foo2.cpp)
```

bar - CMakeLists.txt

```
ADD_LIBRARY(bar bar1.cpp bar2.cpp)
TARGET_LINK_LIBRARIES(bar foo)
```

example - CMakeLists.txt

```
ADD_EXECUTABLE(example example1.cpp example2.cpp)
TARGET_LINK_LIBRARIES(example bar)
```

Exemple : Lib statique / dynamique 3/4

FOO-code — CMakeLists.txt

```
PROJECT(FOO)
...
SET(FOO_VERSION_MAJOR “1”)
SET(FOO_VERSION “1.2.3”)
OPTION(BUILD_SHARED_LIBS "Build shared libraries." OFF)
```

foo - CMakeLists.txt

```
ADD_LIBRARY(foo foo1.cpp foo2.cpp)
SET_TARGET_PROPERTIES(foo PROPERTIES
  VERSION ${FOO_VERSION}
  SOVERSION ${FOO_VERSION_MAJOR})
```

bar - CMakeLists.txt

```
ADD_LIBRARY(bar bar1.cpp bar2.cpp)
SET_TARGET_PROPERTIES(bar PROPERTIES
  VERSION ${FOO_VERSION}
  SOVERSION ${FOO_VERSION_MAJOR})
```

Exemple : Installation 4/4

FOO-code

foo - CMakeLists.txt

```
ADD_LIBRARY(foo foo1.cpp foo2.cpp)
...
INSTALL(TARGETS foo
  DESTINATION lib
  PERMISSIONS OWNER_READ GROUP_READ WORLD_READ)
INSTALL(FILES foo1.h foo2.h
  DESTINATION include
  PERMISSIONS OWNER_READ GROUP_READ WORLD_READ)
```

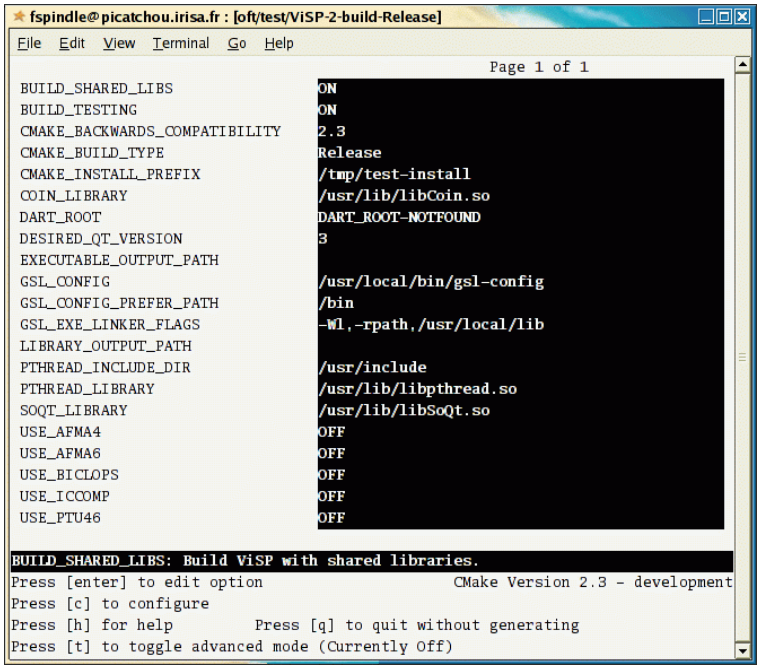
bar - CMakeLists.txt

```
...
INSTALL(TARGETS bar
  DESTINATION lib
  PERMISSIONS OWNER_READ GROUP_READ WORLD_READ)
INSTALL(FILES bar1.h bar2.h
  DESTINATION include
  PERMISSIONS OWNER_READ GROUP_READ WORLD_READ)
```

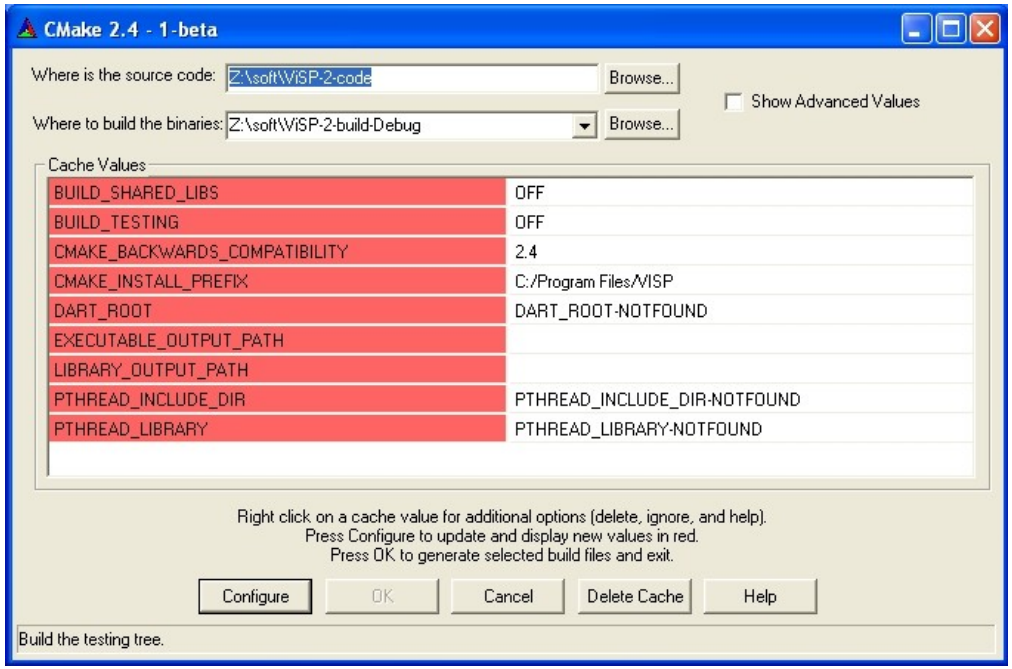
Utilisation via interface graphique

- Editer les variables du cache pour configurer le *build*
- Configuration après un changement: touche “c” ou bouton “Configure”
- Génération des fichiers utiles à la compilation: touche “g” ou bouton “OK”

ccmake (unix)



CMake (windows)



Utilisation en ligne de commande

- Peut être utilisé par des scripts (crontab, tâches planifiées)
- Premier argument: localisation des sources
- Option -G pour sélectionner le générateur utilisé lors du *build*

```
cmake –G”KDevelop3” ou –G”Unix Makefiles” ou –G”Xcode”
```

- Option -D pour positionner des variables du cache

```
cmake –DCMAKE_VERBOSE_MAKEFILE=ON
```

```
cd FOO-build-debug
cmake ../FOO-code -DCMAKE_BUILD_TYPE=Debug
make

cd FOO-build-release
cmake ../FOO-code -DCMAKE_BUILD_TYPE=Release
make
```

In-source et out-of-source build 1/2

- Les sources contiennent
  - Le code source C, C++ (myClass.cpp, myClass.h), Java
  - Fichiers de description (CMakeLists.txt) utiles à CMake
- Les *builds* contiennent
  - Les fichiers produits par CMake (.dsp, .sln, Makefile)
  - Les résultats de la compilation: librairies (.a, .so) et binaires
- *In-source build*
  - Code source et produits de la compilation cohabitent dans la même arborescence
- *Out-of-source build*
  - Code source et produits de la compilation dans des répertoires distincts

In-source et out-of-source build 2/2

- *In-source build*
  - Cohabitation des sources et *build*

```
cd FOO-code
ccmake .
make
```

— FOO-code – [sources, binaires]

- *Out-of-source build*
  - Séparation des sources et *build*
  - Permet de gérer plusieurs versions

```
mkdir FOO-build
cd FOO-build
ccmake ../FOO-code
make
```

FOO-code – [sources]
FOO-build – [binaires]
FOO-build-g++4.x – [binaires]
FOO-build-g++3.x – [binaires]

Le cache

- Résume l'ensemble de la configuration du *build*
- Fichier CMakeCache.txt situé à la racine du projet
- Créé / mis à jour lors des phases de configuration réalisées par CMake
- Les variables sont typées
- Les variables incontournables

```
CMAKE_BUILD_TYPE=[Debug, Release]
CMAKE_INSTALL_PREFIX=[/usr/local, C:\Program Files\ViSP]
CMAKE_VERBOSE_MAKEFILE=[OFF, ON]
```

Démonstration

Sous Windows

Configuration du projet FOO avec CMake

Compilation avec MSVC71

http://www.irisa.fr/lagadic/prive/fspindle/cmake/example/FOO.tar.gz

CTest

- Fait partie de la distribution CMake
- Peut-être utilisé avec ou sans CMake
- Permet de réaliser plusieurs opérations sur le code source
  - Checkout CVS ou Subversion depuis un référentiel,
  - configuration, *build*,
  - Exécution de tests prédéfinis (binaires, scripts),
  - Exécution de tests avancés:
    - Couverture de code: uniquement avec g++ **-fprofile-arcs -ftest-coverage**
    - Etat de la mémoire: utilisation de purify, valgrind
- Les résultats peuvent être soumis à un serveur DART

Exemple

FOO-code

CMakeLists.txt

```
PROJECT(FOO)
SUBDIRS(foo bar example)
ENABLE_TESTING()
```

example/CMakeLists.txt

```
ADD_EXECUTABLE(example example1.cpp example2.cpp)
TARGET_LINK_LIBRARIES(example bar)
ADD_TEST(test1 example)
ADD_TEST(test2 example –foo)
```

Utilisation

- Lancement des tests
  - Liste des tests

```
ctest -N
```

  - Tests automatiques

```
make test
```

```
ctest
```

```
MSVC projet “RUN_TESTS”
```

```
Running tests...
Start processing tests
Test project /udd/fspindle/FOO-build
  1/ 2 Testing test1  Passed
  2/ 2 Testing test2  Passed
100% tests passed, 0 tests failed out of 2
```

  - Tests individuels

```
ctest –I 2,2
```

- Fichiers de log dans Testing/Temporary

```
LastTest.log
```

```
LastTestsFailed.log
```

Démonstration

Sous Windows

Exécution des tests du projet FOO avec MSVC71

Dart

- http://public.kitware.com/Dart
- Serveur permettant de gérer les rapports de tests de logiciels
  - Synthèse de tous les tests sous forme d'un tableau de bord HTML
  - Notification automatique par email en cas d'erreur
  - Facilite le déploiement de tests sur plusieurs plates-formes / configurations
- Client CMake ou CTest
  - Envoie les rapports de *build* et de tests (XML) au serveur
- Catégories de *builds* et de tests
  - *Continuous builds* : lancés si modification du référentiel CVS ou SVN
  - *Nightly builds* : lancés la nuit (utilisés pour suivre l'évolution du logiciel)
  - *Experimental builds*: lancés à la demande pour valider une fonctionnalité
- Serveur Dart à l'Irisa: http://dart.irisa.fr
  - Création de votre projet sur simple demande via helpdesk

Exemple

FOO-code

CMakeLists.txt

```
PROJECT(FOO)
SUBDIRS(foo bar example)
ENABLE_TESTING()
INCLUDE(DartConfig.cmake)
INCLUDE(Dart)
IF(CMAKE_COMPILER_IS_GNUCXX AND NOT BUILD_SHARED_LIBS)
  SET(CMAKE_CXX_FLAGS "-fprofile-arcs -ftest-coverage")
ENDIF(CMAKE_COMPILER_IS_GNUCXX AND NOT BUILD_SHARED_LIBS)
```

DartConfig.cmake

```
SET (DROP_METHOD "xmlrpc")
SET (DROP_SITE "http://dart.irisa.fr")
SET (DROP_LOCATION "FOO")
SET (COMPRESS_SUBMISSION ON)
SET (NIGHTLY_START_TIME "9:00PM")
```

Utilisation

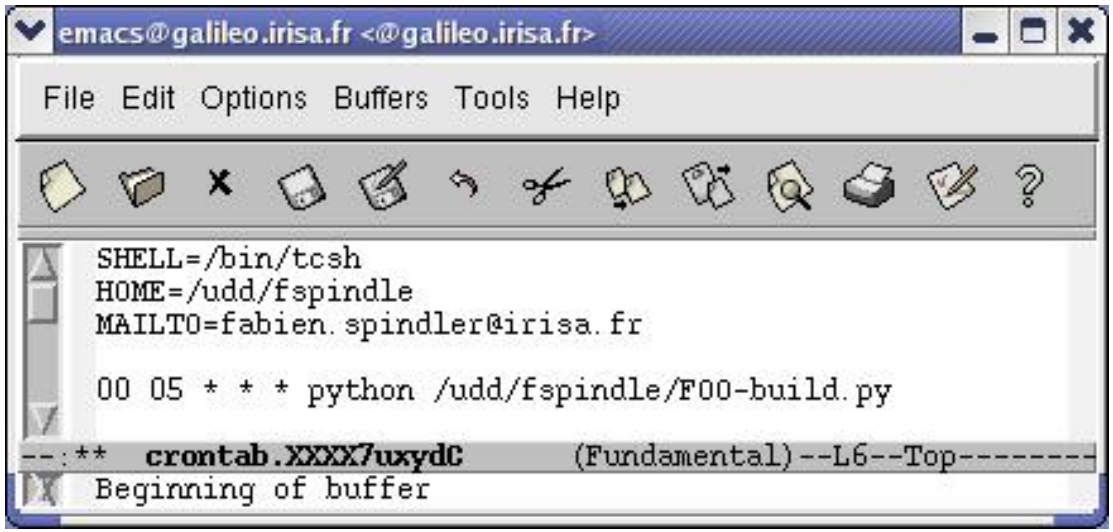
- Lancement des tests

```
ctest –D Experimental
ctest –D Nightly
ctest –D [Experimental|Nightly]Build
ctest –D [Experimental|Nightly]Test
ctest –D [Experimental|Nightly]Coverage
ctest –D [Experimental|Nightly]MemCheck
```

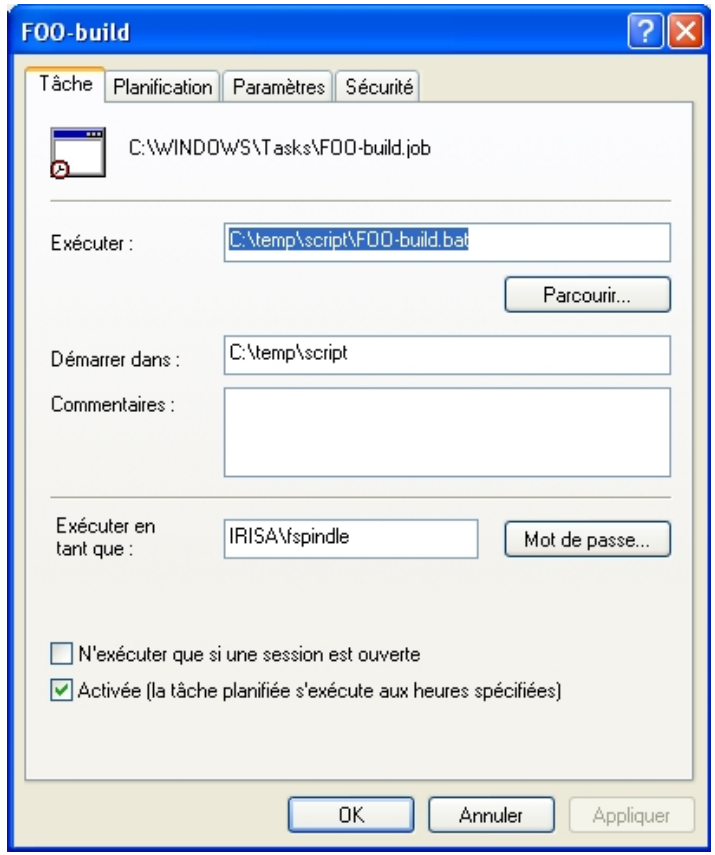
- Les tableaux de bord mettent 10 min pour se mettre à jour

Daily builds

- Exécution de test automatiques
- Utilisation d'un script python: FOO-build.py
  - Crontab Unix
  - Tâches planifiées Windows
  - *checkout*, *build*, test



Édition: crontab -e Consultation: crontab -l



Panneau configuration / Tâches planifiées

Exemple de script python: ViSP-build.py

```
import os, shutil, sys
if os.path.isdir("ViSP"):
  shutil.rmtree("ViSP")
os.system("cvs –d
:pserver:anonymous@scm.gforge.inria.fr:/cvsroot/visp checkout ViSP")

if os.path.isdir("ViSP-build"):
  shutil.rmtree("ViSP-build")
os.mkdir("ViSP-build")
os.chdir("ViSP-build")

if sys.platform == 'win32':
  os.system("cmake ../ViSP -G\"Visual Studio 7 .NET 2003\"")
  os.system("devenv.exe /build debug /project ALL_BUILD VISP.sln")
else:
  os.system("cmake ../ViSP")
  os.system("make")

os.system("ctest -D Experimental")
```

Démonstration

Visite du serveur Dart utilisé pour ViSP

http://dart.irisa.fr/ViSP2/Dashboard/

CPack

- Fait partie de la distribution CMake, intégré à CMake depuis CMake 2.4
- Outil de *packaging* et de distribution de binaires ou de sources
- Peut-être utilisé avec ou sans CMake
- http://www.cmake.org/Wiki/CMake

Package type	*Packaging tool*	Linux	OSX	Win
STGZ	Self extracting Tar GZip	X	X	
TBZ2	Tar BZip2	X	X	
TGZ	Tar GZip	X	X	
TZ	Tar Compress	X	X	
ZIP	ZIP file format	X	X	X
PackageMaker	OSX Package Maker		X	
NSIS	Nullsoft Scriptable Install System			X

Exemple

FOO-code — CMakeLists.txt

```
PROJECT(FOO)
...
INCLUDE(InstallRequiredSystemLibraries)

SET(CPACK_PACKAGE_VERSION ${FOO_VERSION})
SET(CPACK_PACKAGE_VERSION_MAJOR ${FOO_VERSION_MAJOR})
SET(CPACK_PACKAGE_VERSION_MINOR ${FOO_VERSION_MINOR})
SET(CPACK_PACKAGE_VERSION_PATCH ${FOO_VERSION_PATCH})

IF(WIN32 AND NOT UNIX)
  SET(CPACK_NSIS_MODIFY_PATH ON)
ENDIF(WIN32 AND NOT UNIX)

INCLUDE(CPack)
```

Utilisation

- Liste des générateurs en lançant : `cpack --help`
- Fichiers de configuration créés par : `ccmake <source dir>`
  - CPackConfig.cmake
  - CPackSourceConfig.cmake
  - Configurables via CMakeLists.txt
- Utilisation

`make package`	→	`FOO-1.2.3-Linux.sh`
`cpack –G ZIP`	→	`FOO-1.2.3-Darwin.zip`
`MSVC projet “PACKAGE”`	→	`FOO-1.2.3-win32.exe`
`cpack –G TGZ –config CPackSourceConfig.cmake`		
`make package_source`	→	`FOO-1.2.3-Source.tar.gz`

Démonstration

Sous Windows

Création d'un package NSIS du projet FOO

Préalable: installation de NSIS

http://nsis.sourceforge.net/Main_Page

Conclusion

- Avantages
  - CMake: outil de *build* multi plates-formes (Linux, OSX, Windows, …)
  - Supporte les *builds* en dehors des sources (*out-of-source build*)
  - Intègre des outils
    - de test (CTest)
    - de synthèse des rapports de tests (Dart)
    - de packaging et d'installation (CPack)
  - Syntaxe du langage assez simple
- Inconvénients
  - Chemins absolus dans les fichiers générés
  - Exécuter cmake si déplacement du répertoire sources ou *build*
  - Lors d'un *build* indispensable d'avoir CMake d'installé

Pour aller plus loin…

- Documentation des commandes
  - "**`cmake --help`**" pour un résumé
  - "**`cmake --help`** ***`COMMAND`***" pour une aide détaillée
  - "**`cmake --help IF`**"
  - "**`[cmake | ctest | cpack] --help`**"
- Documentation, FAQ, wiki:
  - http://www.cmake.org
- Ouvrages édités par Kitware, Inc.
  - Mastering CMake User's Developers Guide, Août 2003
  - Matering CMake 2.2, Fév. 2006
- Exemples
  - Le projet FOO: http://www.irisa.fr/lagadic/prive/fspindle/cmake/example/FOO.tar.gz
  - A l'Irisa: projets ViSP (Lagadic), Vistal (Visages) http://gforge.inria.fr
  - VTK, ITK, BIAS…