IRISA

Journées bonnes pratiques pour le développement
19-20 octobre 2009

Outils d'aide au développement pour plates-formes hétérogènes

Fabien Spindler

SED-REN

Equipe-projet Lagadic

INRIA Rennes-Bretagne Atlantique - IRISA

http://www.irisa.fr/lagadic

Plan

A. Build
- CMake

B. Automatisation des tests
- CTest
- CDash
- Pipol

C. Packaging
- CPack

D. Conclusion

E. Pour aller plus loin

A. Build: Utilisation de CMake

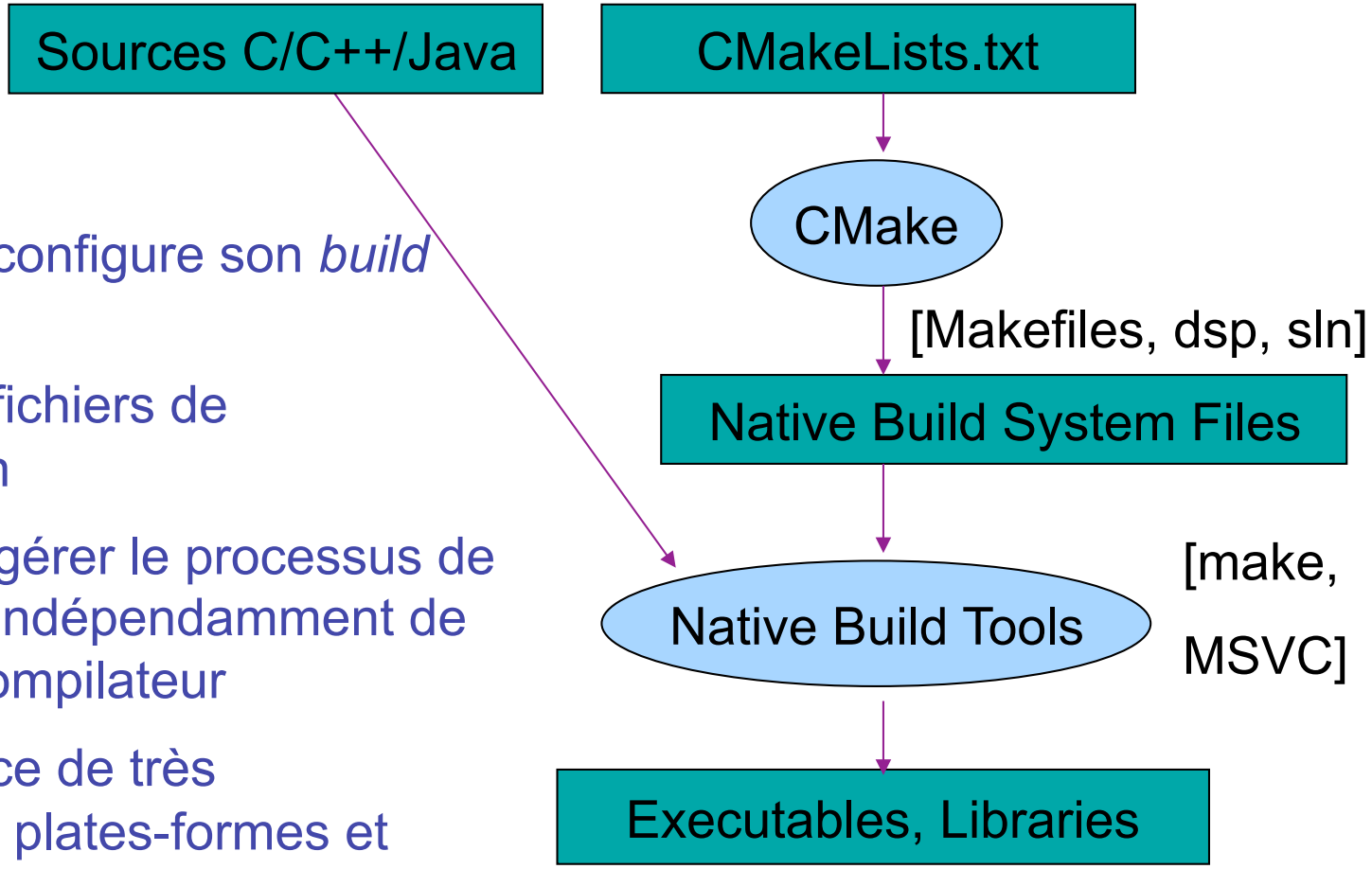
1. Introduction
2. Principe de fonctionnement
3. Eléments de syntaxe
4. Exemple
5. Configuration
   - Via interface graphique
   - En ligne de commande
   - *In-source* et *out-of-source build*
6. Exploitation des fichiers de configuration pour le *build*
7. Démonstration

1. Introduction

- Logiciel open source multi plate-forme: http://www.cmake.org
- Outil d'aide au processus de compilation (*build*) pour des cibles et des compilateurs hétérogènes
- Développé depuis 2001 par Kitware (VTK)
- Adopté par les développeurs de KDE
- Produit des fichiers de configuration: capable de générer des makefile unix, des projets MSVC, Borland… qui seront utilisés par les outils natifs lors de la compilation
- Dispose de 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)

2. Principe de fonctionnement

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



3. Eléments de syntaxe – CMakeLists.txt

Langage de type script

- Commentaire
- Commande
- Liste
- Variable
- Structure de contrôle

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

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

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

```
${VAR}
```

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

Configuration dynamique

- config.h.in

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

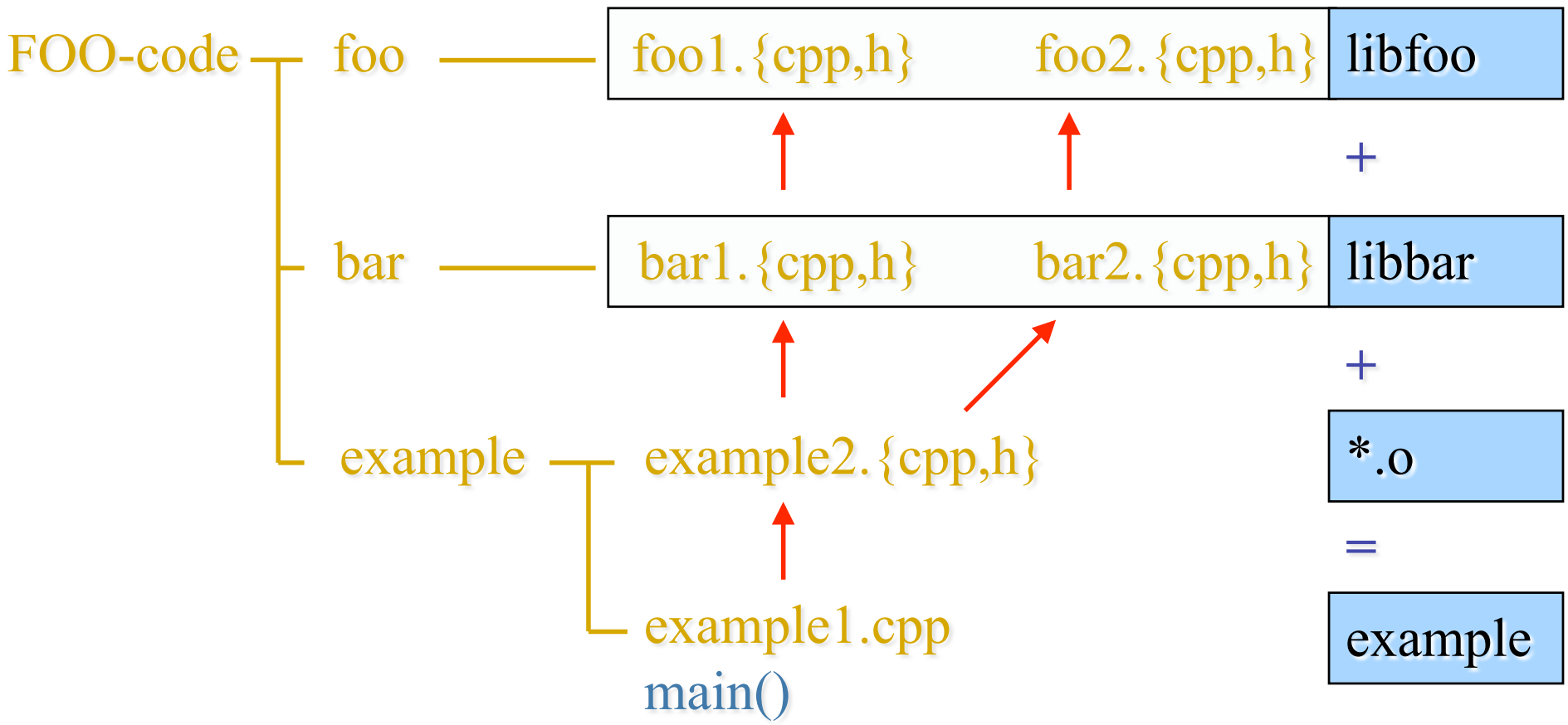
```
#cmakedefine FOO_VERSION ${FOO_VERSION}
#cmakedefine BUILD_SHARED_LIBS
```

Détection de librairie

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

Evalué lors des phases de configuration de CMake

4. Exemple : Le projet FOO



4. Exemple : Configuration du projet FOO

```
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 foo1.h foo2.h)
```

bar - CMakeLists.txt

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

example - CMakeLists.txt

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

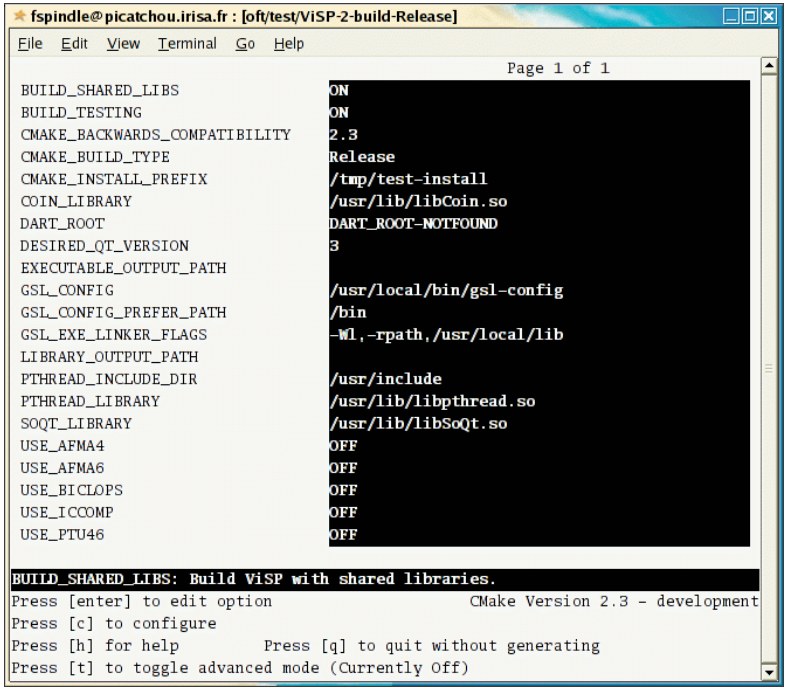
5. Configuration: via interface graphique

Editer les variables du cache pour configurer le *build*

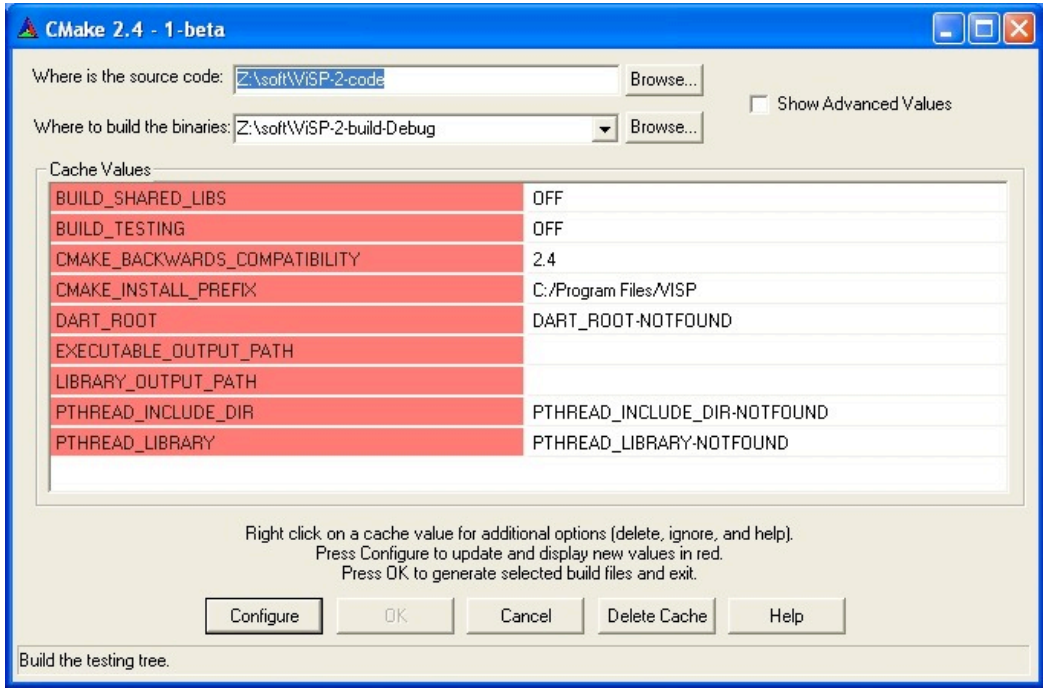
Configuration après un changement: touche “c” ou bouton “Configure”

Génération des fichiers utiles à la compilation: touche “g” ou bouton “OK”

ccmake (unix)



CMake (windows)



5. Configuration: entrées principales du cache

Lors de la configuration, le cache est mis à jour

- Fichier CMakeCache.txt
- Il contient des variables exploitées par CMake pour produire les fichiers de configuration utiles au *build*

Quelques variables importantes

- Type de build

```
CMAKE_BUILD_TYPE=[Debug, Release]
```

- Répertoire d'installation des binaires produits

```
CMAKE_INSTALL_PREFIX=[/usr/local, C:\Program Files\FOO]
```

- Visualisation de la commande utilisée pour compiler (Makefile)

```
CMAKE_VERBOSE_MAKEFILE=ON
```

- Production de librairie statique ou dynamique

```
CMAKE_SHARED_LIBS=[OFF, ON]
```

5. Configuration: 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
```

5. Configuration: in-source et out-of-source build

In-source *build*

- Cohabitation des fichiers source, ceux utiles au *build et binaires produits*

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

— FOO-code – [sources, binaires]

Out-of-source *build*

- Fichiers source séparés de ceux utiles au *build* et binaires produits
- Permet de gérer plusieurs versions de binaires

```
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]

6. *Build*

Exploitation des fichiers de configuration produits par CMake

- Makefile, Microsoft Visual Studio project files .dsp, .sln, …

Utilisation des outils natifs de *build*

- GNU Make, MSVC, Borland

Processus de *build*

```
make
```

```
MSVC projet "ALL_BUILD"
```

Processus d'installation

```
make install
```

```
MSVC projet "INSTALL"
```

7. Démonstration

Sous Windows

Configuration du projet FOO avec CMake

Compilation avec MSVC9

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

B. Automatisation des tests

1. CTest
   - Introduction
   - Exemple
   - Utilisation
   - Démonstration
2. CDash
   - Introduction
   - Exemple
   - Utilisation
3. Pipol
   - Introduction
   - Démonstration
4. Comment automatiser les tests
   - Exemple
   - Démonstration

Objectif: exécuter les tests aussi souvent que possible pour assurer la qualité du développement

1. CTest: Introduction

Fait partie de la distribution CMake: http://www.cmake.org

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 unitaires, non régression (binaires, scripts)
- Exécution de tests avancés:
  - Couverture de code: uniquement avec GNU g++ en ajoutant les options de compilation: -fprofile-arcs -ftest-coverage
  - Etat de la mémoire: utilisation de purify, valgrind

Les résultats peuvent être soumis à un serveur CDash

1. CTest: 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)
```

1. CTest: Utilisation

Lancement des tests

- Liste des tests

```
ctest -N
```

- Tests automatiques

```
make test
```

```
ctest
```

```
MSVC projet "RUN_TESTS"
```

- Tests individuels

```
ctest -I 2,2
```

Fichiers de log dans Testing/Temporary

`LastTest.log` `LastTestsFailed.log`

1. CTest: Démonstration

Sous Windows

Exécution des tests du projet FOO avec MSVC9

2. CDash: Introduction

Logiciel open source: http://www.cdash.org

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

Un exemple de serveur CDash à l'Irisa: http://cdash.inria.fr

2. CDash: Exemple

FOO-code

CMakeLists.txt

```
PROJECT(FOO)
SUBDIRS(foo bar example)
ENABLE_TESTING()
INCLUDE(CTestConfig.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)
```

CTestConfig.cmake

```
SET(CTEST_PROJECT_NAME "FOO")
set(CTEST_NIGHTLY_START_TIME "9:00PM")

SET(CTEST_DROP_METHOD "http")
SET(CTEST_DROP_SITE "cdash.inria.fr")
SET(CTEST_DROP_LOCATION "/CDash/submit.php?project=FOO")
SET(CTEST_DROP_SITE_CDASH TRUE)
```

2. CDash: 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 tests sont envoyés sur le serveur CDash

Ils sont synthétisés sous forme de tableau de bord

2. CDash: Démonstration

Sous Windows

Envoi des rapports de tests expérimentaux sur CDash

pour le projet FOO avec MSVC9

3. Pipol: Introduction

Plate-forme Inria de Portage Logiciel: http://pipol.inrialpes.fr

Objectifs: mettre à disposition des équipes projets différentes architectures matérielles et logicielles (systèmes, librairies, outils de développement) afin de valider la portabilité des logiciels

Pipol permet :

- l'accès, après réservation, à des systèmes Linux (principales distributions libres et à licences), Windows et Mac OS X.
- de travailler sur des architectures 32 ou 64 bits (EMT64 et IA64)
- de passer administrateur sur tous les systèmes.
- d'automatiser les configurations et d'effectuer des compilations cycliques («nightly build»).

3. Pipol: Démonstration

Visite de l'interface web

http://pipol.inrialpes.fr

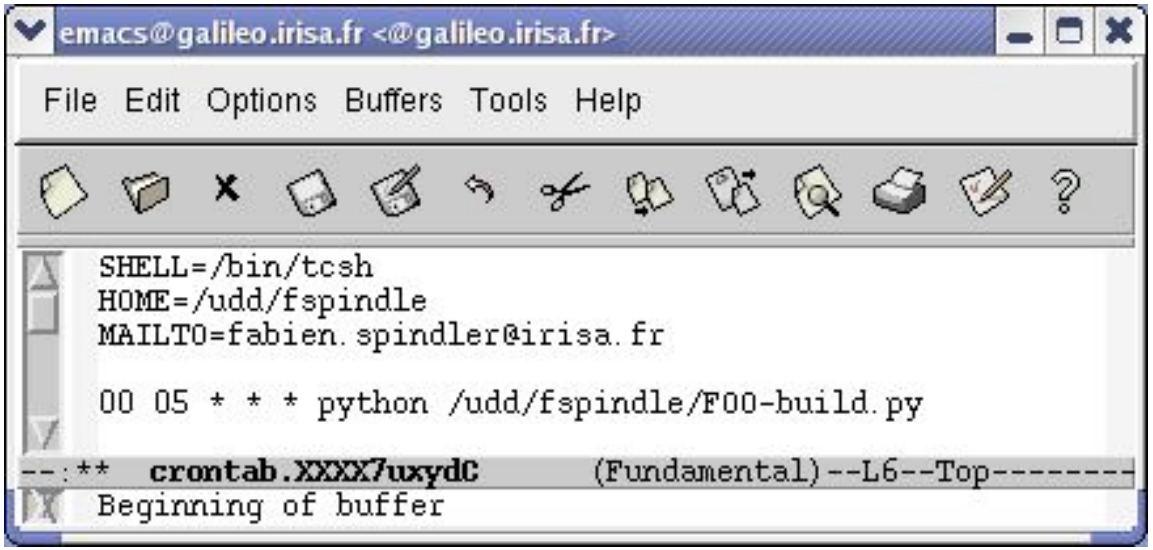
4. Comment automatiser les tests: Exemple

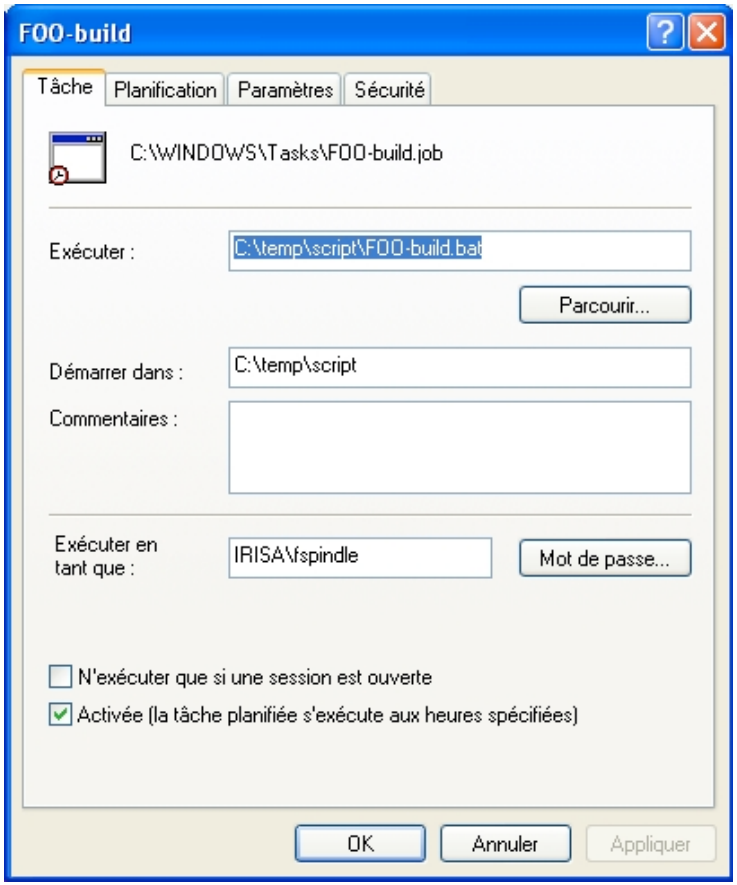
On parle également de *daily builds*

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

4. Exemple de script python: FOO-build.py

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

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

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

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

C. Packaging: Utilisation de CPack

1. Introduction
2. Exemple
3. Utilisation
4. Démonstration

1. CPack: Introduction

La distribution CMake inclus un outil de packaging: CPack

- Distribution de code source ou de binaires sous forme d'archive
- Peut-être utilisé avec ou sans CMake

Package type	Packaging tool	Linux	OSX	Win
STGZ	Self extracting Tar GZip	■	■	
TBZ2	Tar BZip2	■	■	
TGZ	Tar GZip	■	■	
TZ	Tar Compress	■	■	
ZIP	ZIP file format	■	■	■
PackageMaker	OSX Package Maker		■	
NSIS	Nullsoft Scriptable Install System			■
DEB	Debian packages	■		
RPM	RPM packages	■		

2. CPack: Exemple

FOO-code — CMakeLists.txt

```
PROJECT(FOO)
SUBDIRS(foo bar example)
...
INCLUDE(InstallRequiredSystemLibraries)

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

INCLUDE(CPack)
```

3. CPack: Utilisation

Liste des outils de packaging disponibles : `cpack --help`

Production de fichiers de configuration par: `ccmake <source dir>`

- CPackConfig.cmake
- CPackSourceConfig.cmake

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`

4. CPack: Démonstration

Sous Windows

Création d'un auto-installeur du projet FOO

à l'aide de NSIS avec MSVC9

D. Conclusion

CMake: outil de configuration pour produire des *builds*

- Multi plates-formes (Linux, OSX, Windows, …)
- Multi-compilateurs (g++, MS cl, Borland)

Supporte les *builds* en dehors des sources (*out-of-source build*)

- Gestion de plusieurs versions de binaires pour un même code source

Syntaxe du langage assez simple s'appuyant sur des fichiers texte

- CMakeLists.txt

Interface vers plusieurs IDE

- KDevelop, Xcode, MSVC, Eclipse

D. Conclusion

CMake intègre un outil de test: CTest

- Permet de lancer des tests sur des cibles hétérogènes
- Tests unitaires, de non régression (binaires, scripts)
- Intègre des outils de test de l'utilisation de la mémoire
- Possibilité d'automatiser les tests
- Peut-être utilisé avec ou sans CMake

Possibilité d'envoyer les résultats des tests sur CDash un serveur web

- Synthèse des tests
- Historique
- En cas d'erreur notification par mél

CMake intègre également un outil de packaging: CPack

Un des outils indispensable pour mener à bien un développement de qualité

E. 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. disponible à la bibliothèque

- 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, IGSTK, BIAS…

4. Comment automatiser les tests: Démonstration

Visite du serveur CDash utilisé pour ViSP

http://cdash.inria.fr/CDash/index.php?project=ViSP