Les outils d'aide au processus de “build”

Nicolas Chleq
DREAM
http://www-sop.inria.fr/dream

Processus de “build”

Etapes :

- Compilation et édition de liens : passer des sources aux exécutables
- Installation : déployer l'application
- Génération des distributions sources ou binaires
- Génération de la documentation
- Exécution des suites de tests

Automatisation de ces étapes

Les outils

Make

- L'ancêtre

Autoconf/automake

- GNU

tmake/qmake

- Trolltech (Qt, KDE)

cmake

- Kitware (VTK)

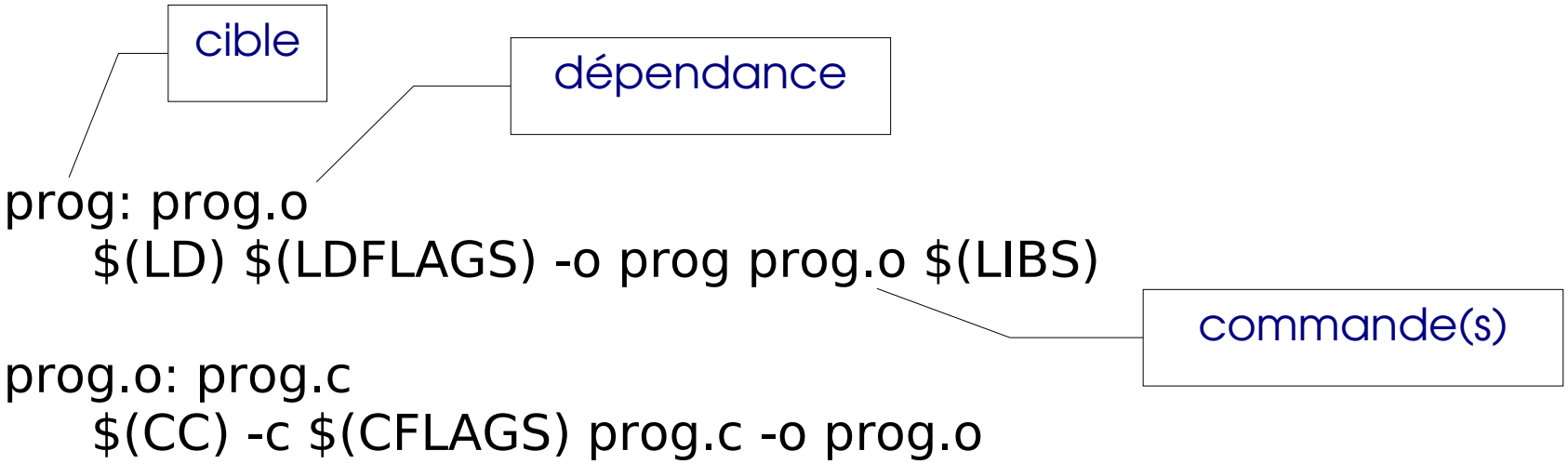
Ant

- Apache

Make

Principe :

- Un fichier (Makefile) décrit les commandes à exécuter pour construire une ou plusieurs “cible(s)”
- Une “cible” est un exécutable, une librairie, une documentation, etc.
- Les cibles ont une/des dépendances : autres cibles à construire avant



Make

Avantages

- Outil standard
- Toutes les implantations ont une base commune
- Principes de fonctionnement simples
- Bien documenté

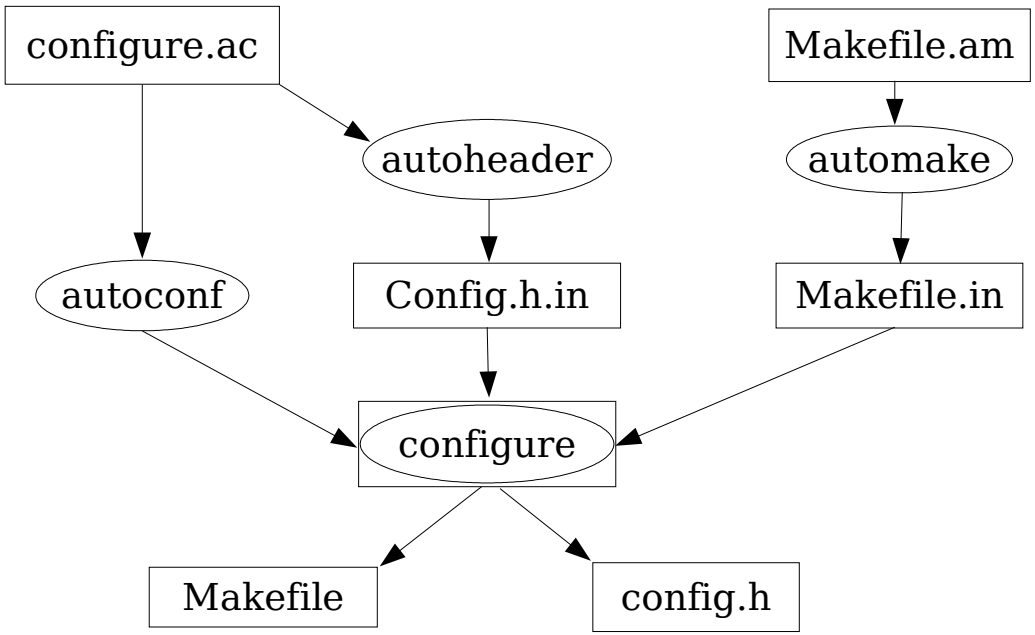
Inconvénients

- Syntaxe des commandes (shell)
- Portabilité des commandes

Autoconf/Automake

Principe :

- Automake génère un/des squelettes de Makefile (Makefile.in)
- Autoconf génère un script (appelé configure)
- L'exécution de configure génère un/des Makefile



Autoconf/Automake

Inconvénients :

- Apprentissage difficile et pénible
- Une syntaxe pour automake, une autre pour autoconf
- Extension : m4/shell/make...
- Unix. Windows avec cygwin

Avantages :

- Macros de tests des caractéristiques/capacités de la plateforme
- Support (automake) pour l'exécution de suites de tests
- Supporte les builds en dehors des sources
- Bien documenté
- Archive de macros pour Autoconf : http://ac-archive.sourceforge.net/

CMake

Principe :

- Génère un makefile (Unix) ou un projet Visual C++ (Windows) à partir d'un fichier de description CMakeLists.txt
- Utilisable en ligne de commande (cmake) ou avec GUI (ccmake)

```
SET(CMAKE_INSTALL_PREFIX /usr/)
ADD_EXECUTABLE(test test.c)
INSTALL_TARGETS(bin test)

$ cmake .          # génère le fichier Makefile
$ make
$ make install
```

CMake

Exécution de suite de tests

```
ENABLE_TESTING()
ADD_EXECUTABLE(test1 test1.c)
ADD_EXECUTABLE(test2 test2.c)
ADD_TEST(test1 test1)
ADD_TEST(test2 test2)
```

```
$ cmake .
$ make test
Building tests test...
Test project
  1/  2 Testing test1                    Passed
  2/  2 Testing test2                    Passed

100% tests passed, 0 tests failed out of 2
```

CMake

Extensions

- Fichiers d'extension à utiliser ainsi :

```
INCLUDE (${CMAKE_ROOT}/Modules/FindOpenGL.cmake)
```

- Adapte la compilation à l'utilisation de librairies externes

Inconvénients

- Chemins absolus dans le/les Makefile(s) générés : il faut réexécuter cmake si on déplace le directory

Avantages

- Une seule syntaxe simple à apprendre
- Portabilité du build
- Supporte les builds en dehors des sources

Tmake

Historique

- Initialement développé par TrollTech pour Qt et KDE versions < 2
- Remplacé par qmake dans Qt et KDE versions 3
- Repris et étendu : http://tmake.sourceforge.net/

Principe

- Un fichier *projet*.pro décrit le projet à compiler :

```
SOURCES = hello.c
CONFIG += warn_on release
```

- L'exécution de tmake génère le Makefile (unix)

```
$ tmake -o Makefile  projet.pro
$ make
```

Tmake

Projet pour VisualC++

- $ tmake -lib win32-msdev *projet*.pro

Projet Kdevelop

- $ tmake -lib unix-kdev *projet*.pro
  $ kdevprj2kdevelop *projet*.kdevprj

Librairie partagé (unix)

- $ tmake -t lib *projet*.pro

Librairie statique

- Dans *projet*.pro :
  CONFIG += staticlib
- Commande tmake :
  $ tmake -t lib *projet*.pro

Tmake

Avantages

- Apprentissage facile
- Tutorial
- Utilitaire de génération de fichiers .pro (progen)
- Expressions conditionnelles
  - Plateforme
  - Configuration (variable CONFIG)

Inconvénients

- Pas de support des suites de tests
- Pas de support lex et yacc : écrire un template (voir documentation)
- Pas de build en dehors des sources

Qmake

Principe identique à Tmake

Plus :

- Mode génération de projet intégré (option -project)
- Variable QMAKESPEC et option -spec plutôt qu'option -lib de tmake
- Support lex et yacc intégré
- Expressions conditionnelles plus étendues
- Documentation plus complète (manuel de référence)

Ant

Outil pour :

- Compilation
- Test
- Déploiement

des programmes Java

Principe

- Un fichier (par défaut build.xml) décrit le projet et les tâches correspondantes
- Exécute :
  - $ ant [-buildfile truc.xml] *cible*

Ant

Fichier build.xml

- Trois niveaux : projet / cible / tâche

```
<project name="build" default="all" basedir=".">

   <property name="src" value="java/" />
   <property name="build" value="classes" />
   <property name="manifest" value="meta/META-INF/MANIFEST.MF" />

   <target name="compile">
      <mkdir dir="${build}" />
      <javac srcdir="${src}" destdir="${build}" />
      <jar destfile="myapp.jar" manifest="${manifest}">
         <fileset dir="${build}"> <exclude name="**/CVS/*" /> </fileset>
      </jar>
   </target>
</project>
```

cible

Tâche

- Exécute : $ ant compile

Ant

Avantages

- Presque un standard...
- Complet
  - Tâches prédéfinies : javac, jar, javadoc, zip, java, exec, mkdir, ...
- Extensible : une tâche est un objet java
  - Voir les pages :
    - “External Tools and Tasks” : http://ant.apache.org/external.html
    - “Related Projects” : http://ant.apache.org/projects.html
- Bien documenté (manuels, tutoriaux, livres, etc.)
- Utilisable en ligne de commandes et dans tous les bons IDE

Inconvénients

- Syntaxe XML
- Conditions

Comparatif

	Langages	***Plateformes***	***Apprentissage***	***Documentation***
Make	Tous	Toutes	Moyen	Bonne
Automake/ Autoconf	C/C++	unix/cygwin	Difficile	Bonne
Cmake	C/C++	unix/win32	Facile	Bonne
Tmake	C/C++	unix/win32	Facile	Moyenne
Qmake	C/C++	unix/win32/ macosx	Facile	Bonne
Ant	Java	Toutes	Facile	Bonne

Liens

Autoconf/Automake

- http://www.gnu.org/software/autoconf
- http://ac-archive.sourceforge.net/

Cmake

- http://www.cmake.org/

Tmake

- http://tmake.sourceforge.net/

Qmake

- http://doc.trolltech.com/3.3/qmake-manual.html

Ant

- http://ant.apache.org/
- http://www-interne.irisa.fr/atelier/documentations/ASCII/seminaires/ant.pdf