Ant *vs Make*

Jean-Noël Albert

19 Avril 2005

LAL

Ant *vs* Make

- **Make** construction d'application
  standard UNIX
  orienté « tache à accomplir »
- **Ant** construction d'application
  vise la portabilité
  orienté « enchaînement de taches »
  extensible
  écrit en Java
  bien adapté aux projets JAVA

L'approche Make

- ***Principe*** *:* on décrit le but à atteindre, pas la manière d'y parvenir

- Exemple

  *% **make hello***

  cc hello.c –o hello

  Cherche un fichier permettant de construire ***hello***, trouve ***hello.c***, le compile avec le compilateur adapté (CC).

- Description des étapes de construction dans un fichier ***Makefile***

L'approche Make

- Ne compile que ce qui est nécessaire

  *% **make***

  make: `hello' is up to date

- Gagne du temps,
  appréciable pour les projets complexes
- L'application est toujours à jour
- Une étape peut dépendre d'une ou de plusieurs autres
- Une étape peut ne pas être liée
  à la création d'un fichier

Makefile

- Exemple très simplifié

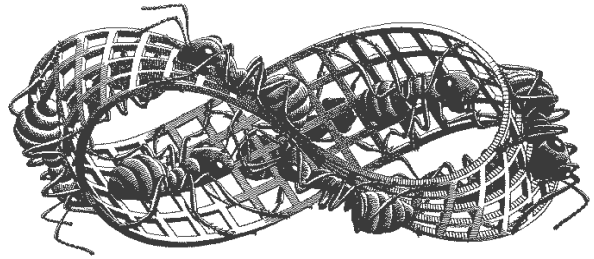
```
hello : hello.o
        $(CC) –o $@  $^

hello.o : hello.c

clean :
        rm hello hello.o *~
```

- Make exécute la première cible (hello) ou la cible indiquée
  - ex : ***make clean***

Inconvénients de Make

- Cheminement parfois difficile à comprendre
- Différentes variantes avec de nombreuses extensions difficilement portables
- Support difficiles des options de CC/C++ sur les différentes plates-formes UNIX
  - GMake : beaucoup d'extensions très pratiques
- Difficilement transportable vers Windows
  - Principalement à cause des noms des fichiers
- Mal adapté aux projets avec de nombreux répertoires

Ant



- Succession de taches
- Les taches visent à masquer les commandes effectivement réalisées
- Une tache peut dépendre d'autres taches
  - Mais une tache ne sera exécutée qu'une seule fois pour un appel à Ant
  - Évite des bouclages survenant avec Make
- Conditions pour contrôler l'exécution des taches
- Support l'itération sur un ensemble de répertoires et de fichiers
- Portable
- Bien adapté à Java

Ant et Java

- 3 taches : **javac**, **jar** et **javadoc**

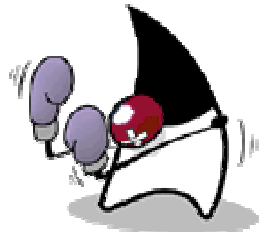
```
<javac srcdir="sources/"
       destdir="classes/" />
<jar destfile="lib/hello.jar"
     basedir="classes/" />
<javadoc packagenames="demo"
         sourcepath="sources/"
         destdir="docs/" />
```

- Plus les taches pour la création de répertoires, la suppression de fichiers intermédiaires, ...

Un exemple

```
<?xml version="1.0"?>
<project name="Hello" default="build" basedir=".">
  <target name="build">
    <javac srcdir="sources/" destdir="classes/" />
    <jar destfile="lib/hello.jar" basedir="classes/" />
  </target>
  <target name="doc" depends="build">
    <javadoc packagenames="demo"
          sourcepath="sources/"
          destdir="docs/" />
  </target>
  <target name="clean">
    <delete>
      <fileset dir="." defaultexcludes="no">
        <include name="**/*~"/>          <!-- XEmacs -->
        <include name="**/*.class"/>     <!-- Classes Java -->
        <include name="lib/*.jar"/>      <!-- Library Java -->
        <include name="docs/**/*" />     <!-- Javadoc -->
      </fileset>
    </delete>
   </target>
</project>
```

Extensions



- Possibilité d'ajouter de nouvelles taches en fournissant les librairies Jar
- Exemple : C++, Doxygen

```
<taskdef resource="net/sf/antcontrib/antcontrib.properties"/>
<taskdef resource="cpptasks.tasks"/>
<typedef resource="cpptasks.types"/>
<target name="build">
   <cc link="executable" outfile="demo.exe">
     <fileset dir=".">
          <include name="*.cc"/>
     </fileset>
     <syslibset libs="c,stdc++"/>
   </cc>
</target>
```

Conclusions

- Ant est portable
  extensible
  bien adapté à Java
    dans ce cas, il est *très* efficace
  bien intégré à des IDE comme Eclipse
- Make est plus standard, plus répandu
- Recommandations
  - Essayer sur un nouveau projet, si possible simple
  - Ne pas essayer de migrer un Makefile compliqué, du moins au début