

Construction de logiciel et packaging

Matthieu Herrb



Ecole IDL2011, La Londe Les Maures, 5 au 9 décembre 2011

<http://homepages.laas.fr/matthieu/idl2011/>



Ce document est sous licence

Creative Commons Paternité - Partage à l'Identique 3.0 non transposé.

Pour accéder à une copie de cette licence, merci de vous rendre à l'adresse suivante :

<http://creativecommons.org/licenses/by-sa/3.0/>

Des parties de document ont été empruntés aux présentations de Johan Moreau, « Outils de construction » et de Konrad Hinsén « Packaging en Python » à l'école ENVOL 2010.

Agenda

- 1 Introduction
- 2 Outils généraux
 - Autotools
 - CMake
 - Autres
- 3 Outils spécifiques
 - Outils liés à une distribution
 - Outils liés à un langage
- 4 Conclusion

Agenda

1 Introduction

2 Outils généraux

- Autotools
- CMake
- Autres

3 Outils spécifiques

- Outils liés à une distribution
- Outils liés à un langage

4 Conclusion

Construction et packaging :

À partir d'un code source :

- fournir un ensemble exécutable et installable
- fournir des paquets :
 - source
 - binaire



Le point de vue de l'utilisateur

L'utilisateur :

- recherche des instructions simples (fichier README ou INSTALL)
- aime bien suivre des procédures connues
 - `configure ; make ; make install`
 - `dpkg -i supersoft.deb`
- préfère ne pas avoir 3 tonnes de dépendances à installer manuellement avant de pouvoir tester un logiciel.

Tâche complexe

Souvent le premier contact avec le logiciel

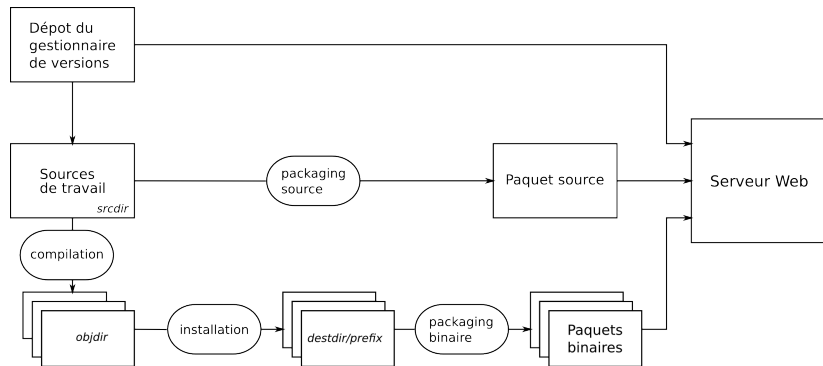
- doit être robuste,
- variabilité des environnements,
- variabilité des utilisateurs,
- doit répondre aux attentes de l'utilisateur,
- dépend du type de public ciblé,
- dépend de facteurs techniques mais aussi « culturels »,
- aspects liés à la licence d'utilisation

→ aspect important à ne pas négliger

Automatiser :

- Configuration
- Compilation, génération de fichiers annexes (documentation,...)
- Installation
- Désinstallation

Démarche générale



Intégration continue

But : tester en continu l'ensemble de son projet :

- configuration
- compilation
- installation
- exécuter tests unitaires
- désinstallation

Utilise les mêmes outils.

Permet de produire des instantanés (*snapshots*) quotidiens du développement

Interfaces, numéros de version, release, etc.

Release : notion *marketing* : nom, numéro, logo, évènementiel,...

Besoin des développeurs et des utilisateurs :

- identifier les modifications importantes pour eux,
- dans le cas d'une bibliothèque :
 - versions de l'API (majeur.mineur) correspond à la compatibilité binaire avec les versions précédentes
- pour les autres cas : documenter les changements de comportement, de format de fichiers, prévoir les procédures de migration,...

Gérer un planning de releases et tenter de s'y tenir.

Configuration

Adaptation du logiciel à une plateforme

2 modes :

- pré-sélection à partir d'une base de connaissances
- détection automatique des fonctionnalités de la plateforme

Toujours privilégier les tests sur les fonctionnalités

exemple : `#ifdef HAVE_STRLCPY`

Produit :

- fichiers d'entête
- Makefiles

Compilation

Programme :

- compilation proprement dite : traduction source → assembleur
- assemblage : traduction → code machine
- édition des liens : résolution des références aux bibliothèques externes.

Bibliothèque :

- compilation des composants
- création du fichier bibliothèque (ar ou ld)

objdir ou **builddir**

répertoire séparé des sources où sont produits les fichiers machine.

Gestion de la compilation

Outil quasi-universel : **make**

Mécanisme de règles de production :

CIBLES: DÉPENDANCES
COMMANDES
...

Cibles : liste de fichiers à construire

Dépendances : liste de fichiers dont dépendent les cibles

Commandes : commandes à exécuter pour construire les cibles

Très puissant, mais

- lourd à maintenir pour de gros projets
- difficile de gérer la portabilité

Outils pour produire Makefile

- Automake/autoconf/configure
- CMake
- Ant
- Scons
- ...

Copier les fichiers à leur place finale dans l'arborescence système.

- ne pas sauter cette étape
- respecter les conventions du système cible
 - Linux : FHS, LFS
 - Mac OS X : Bundles, ou alors Unix-like
 - Windows : "Program Files"...
- permet de créer des paquets binaires
- meilleure intégration avec d'autres logiciels
- **destdir** : racine de l'installation
- **prefix** : répertoire de base d'une installation

Opérations annexes (optionnelles) : création d'un utilisateur dédié, enregistrement des composants installés, ...

Opération inverse de l'installation

- supprimer tous les composants installés
- défaire les opérations annexes

Paquets « bien élevés » :

laissent le système dans l'état où ils l'ont trouvé.

Formats de distribution

■ Source

- Archive des sources,
- outils de configuration, compilation, installation
- documentation

Format : archive TAR ou ZIP, système de gestion de version

■ Binaire

- exécutables binaires + fichiers annexes
- programme d'installation
- documentation
- méta-données : dépendances, liste de ce qui est installé,...

Formats de paquets binaires dépendants du système d'exploitation, du gestionnaire de paquets,

Quel format choisir ?

- Logiciels libres :
 - favoriser la distribution source
 - laisser faire les paquets binaires à ceux qui savent
- Logiciels non libres :
 - faible diffusion : envisager le transfert des sources, packaging simple
 - diffusion large : privilégier les paquets binaires standards des plateformes visées.

Se faire aider...

Quelques caractéristiques des formats de paquets binaires

- méta-données
 - description résumée du paquet
 - dépendances
 - liste des fichiers installés
 - commandes à exécuter
 - mots-clés décrivant la fonctionnalité fournie
 - Auteurs, licence
- fichiers proprement dits
- hash, signature

En général, lors de l'installation, les métadonnées sont stockées dans la base de données des paquets installés.

Outils pour produire des paquets binaires

Dépendants de la plateforme...

Système	format	outils
Debian/Ubuntu	.deb	dpkg-build
Redhat/Fedora/SuSE	.rpm	rpmbuild
Mac OS X	.pkg, .dmg	PackageMaker (XCode)
Windows	MSI	WinInstall (?)

Un paquet doit :

- Mettre en évidence sa licence d'utilisation
- Respecter les licences des composants tierce inclus

→ Fournir un fichier COPYING ou LICENSE qui regroupe toutes les licences applicables.

→ Dans le cas de distributions binaires de logiciels sous GPL, prévoir un paquet source et indiquer clairement où le trouver.

Dépend du type de diffusion...

Si choix d'un contrôle des droits :

- ne pas bricoler un système « maison »
- utiliser des outils standard (flexlm,...)
- nécessite intégration fine dans le code source pour être efficace
- prévoir les licences pour les développeurs
- nécessite un politique de sécurité rigoureuse
- se faire aider

Un projet a besoin :

- d'un nom
- d'un logo (ou d'une charte graphique)
- d'un site web (du nom de domaine associé?)
- d'une liste de diffusion
- du code...

⇒ Forges, plateformes de diffusion,...

Créer une communauté

Agenda

1 Introduction

2 Outils généraux

- Autotools
- CMake
- Autres

3 Outils spécifiques

- Outils liés à une distribution
- Outils liés à un langage

4 Conclusion

- relativement indépendants du langage de programmation et du système
- gèrent la configuration, la compilation, le packaging
- le packaging est une étape parmi d'autres



GNU Autotools

Références :

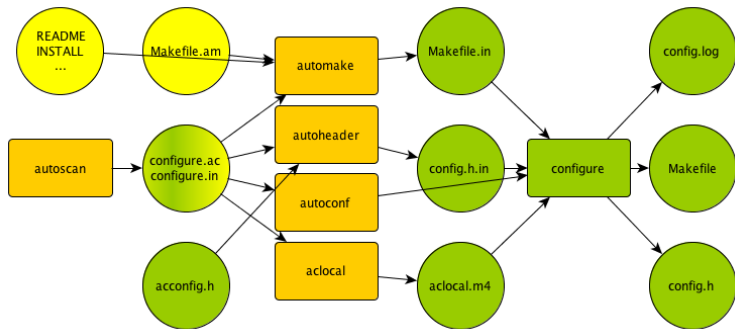
- <http://www.gnu.org/software/autoconf/>
- <http://sourceware.org/autobook/>
- <http://www.lrde.epita.fr/adl/autotools.html>

```
configure; make; make install
```

Ensemble d'outils

- automake, autoheader, autoconf, libtool,...
- écrits en **shell** et pré-processeur **m4**
- génération automatique de Makefiles
- configuration en fonction de la plateforme
- portables

Autotools - un ensemble d'outils



Autotools - pour démarrer

- création de `configure.ac`
- création des `Makefile.am`
- exécution de `autoreconf -force -install`
- exécution de `configure`
- exécution de `make`
- exécution de `make install`

Autotools - Hello World

src/main.c

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    print("Hello, World\n");
    return 0;
}
```


Autotools - fichiers à créer

configure.ac

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

Makefile.am

```
SUBDIRS=src
```

src/Makefile.am

```
bin_PROGRAMS = hello
hello_SOURCES = main.c
```

Notre `configure.ac`

```
configure.ac
```

```
AC_INIT([hello], [1.0], [bug-report@example.org])  
AM_INIT_AUTOMAKE([foreign])  
AC_PROG_CC  
AC_CONFIG_HEADERS([config.h])  
AC_CONFIG_FILES([Makefile src/Makefile])  
AC_OUTPUT
```

- Initialisation Autoconf (nom, version, @ de support, ...)

Notre `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Initialisation Autoconf (nom, version, @ de support, ...)
- Initialisation Automake (paquet non GNU)

Notre `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Initialisation Autoconf (nom, version, @ de support, ...)
- Initialisation Automake (paquet non GNU)
- Utilisation du compilateur C

Notre `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Initialisation Autoconf (nom, version, @ de support, ...)
- Initialisation Automake (paquet non GNU)
- Utilisation du compilateur C
- Déclare `config.h` pour les entêtes

Notre `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Initialisation Autoconf (nom, version, @ de support, ...)
- Initialisation Automake (paquet non GNU)
- Utilisation du compilateur C
- Déclare `config.h` pour les entêtes
- Déclare `Makefile` et `src/Makefile`

Notre `configure.ac`

`configure.ac`

```
AC_INIT([hello], [1.0], [bug-report@example.org])
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

- Initialisation Autoconf (nom, version, @ de support, ...)
- Initialisation Automake (paquet non GNU)
- Utilisation du compilateur C
- Déclare `config.h` pour les entêtes
- Déclare `Makefile` et `src/Makefile`
- Génère les fichiers spécifiés

autotools - exemples de vérifications de programmes

AC_PROC_CC, AC_PROG_CXX, AC_PROG_F77

Vérifications pour les compilateurs

AC_PROG_SED, AC_PROG_YACC, AC_PROG_LEX

Vérification pour des utilitaires

AC_CHECK_PROGS(VAR, PROG, [VAL-IF-NOT-FOUND])

VAR est positionné au chemin de *PROG* si trouvé ou à *VAL-IF-NOT-FOUND* sinon.

exemple

```
AC_CHECK_PROGS([TAR], [tar gtar], "not found")
if [ "$TAR" = "not found" ]; then
    AC_MSG_ERROR([Ce paquet a besoin de tar.])
fi
```


autotools - macros utiles

`AC_MSG_ERROR(ERROR-DESCRIPTION, [EXIT-STATUS])`

Affiche *ERROR-DESCRIPTION* et arrête configure.

`AC_MSG_WARN(ERROR-DESCRIPTION)`

Pareil, mais n'arrête pas

`AC_DEFINE(VARIABLE, VALUE, DESCRIPTION)`

Ajoute une définition dans `config.h` :

```
/* DESCRIPTION */  
#define VARIABLE VALUE
```

`AC_SUBST(VARIABLE, [VALUE])`

Affecte la valeur *VALUE* à *VARIABLE* dans `Makefile`

Vérification pour les bibliothèques

```
AC_CHECK_LIB(LIBRARY, FUNCT, [ACT-IF-FOUND],  
[ACT-IF-NOT])
```

Vérifie si *LIBRARY* existe et contient *FUNCT*. Si oui exécute *ACT-IF-FOUND*, sinon *ACT-IF-NOT*.

exemple

```
AC_CHECK_LIB([z], [gzopen64], [LIBZ64=-lz])  
AC_SUBST([ZLIB64])
```

permet d'utiliser `$(LIBZ64)` dans les Makefiles.

Vérification pour les entêtes

`AC_CHECK_HEADERS(HEADERS...)`

vérifie la présence de *HEADERS* et définit `HAVE_HEADER_H`

exemple

```
AC_CHECK_HEADERS([sys/param.h unistd.h])
AC_CHECK_HEADERS([wcar.h])
```

produit dans `config.h` :

```
#define HAVE_SYS_PARAM_H
#define HAVE_HUNISTD_H
#undef HAVE_WCAR_H
```

On écrit ensuite :

```
#ifdef HAVE_UNISTD_H
# include <unistd.h>
#endif
```

Autotools - utilisation de pkg-config

`PKG_CHECK_MODULES(BASE, PKG-SPEC...)`

vérifie avec `pkg-config` que les paquets décrit par `PKG-SPEC` sont installés. Définit alors `BASE_CFLAGS` et `BASE_LIBS` avec les options pour utiliser ces paquets.

exemple

```
PKG_CHECK_MODULE([GTK], [gtk+-2.0])
```

Dans `Makefile.am` :

```
AM_CFLAGS = $(GTK_CFLAGS)
```

```
AM_LDFLAGS = $(GTK_LIBS)
```

Autotools - écriture de Makefile.am

Makefile.am

```
SUBDIRS=src
```

src/Makefile.am

```
bin_PROGRAMS = hello  
hello_SOURCES = main.c
```

- SUBDIRS pour définir les sous-répertoires ou aller
- bin_PROGRAMS : exécutable(s) à générer
- program_SOURCES : liste des fichiers sources

Autotools - cibles dans Makefile.am

Makefile.am

*lieu*_TYPE = cible ...

lieu : répertoire d'installation

bin_ → \$(bindir)

lib_ → \$(libdir)

TYPE : défini les règles utilisées

_PROGRAMS

_LIBRARIES

_LTLIBRARIES

_HEADERS

_SCRIPTS

_DATA

Autotools - construction de bibliothèque

- Ajouter AC_PROG_LIBTOOL dans `configure.ac`

Makefile.am

```
lib_LTLIBRARIES = libfoo.la
libfoo_la_sources = foo.c foopriv.h
include_HEADERS = foo.h
```

- bibliothèque installée dans `$(libdir)`
- en-têtes publics installés dans `$(includedir)`
- en-têtes privés pas installés

Autotools - fichier pkg-config pour une bibliothèque

Permet aux autres d'utiliser `pkg-config`

foo.pc.in

```
prefix=@prefix@
exec_prefix=@exec_prefix@
libdir=@libdir@
includedir=@includedir@

Name: foo
Description: Library foo providing foo functions
Version: @PACKAGE_VERSION
Cflags: -I${includedir}
Libs: -L${libdir} -lfoo
```


Autotools - production et installation du fichier .pc

configure.ac

```
AC_CONFIG_FILES([Makefile
                 src/Makefile
                 foo.pc])
AC_OUTPUT
```

Makefile.am

```
pkgconfigdir = $(libdir)/pkgconfig
pkgconfig_DATA = foo.pc
```

Autotools - packaging source

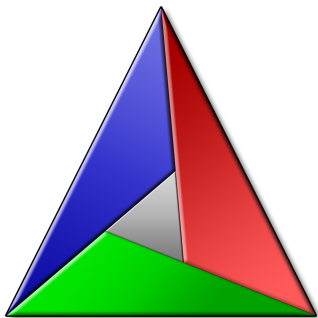
`make distcheck`

- produit une archive source au format `tar.gz`
- vérifie qu'elle se compile et s'installe sans problème

Autotools - packaging binaire

Pas vraiment supporté.

outil de base : `make install` avec `DESTDIR=/staging`
permet de récupérer dans `/staging/` l'ensemble des fichiers installés.



CMake

Références :

- <http://www.cmake.org/cmake/help/documentation.html>
- http://www.cmake.org/cmake/help/cmake_tutorial.html
- http://www.elpauer.org/stuff/learning_cmake.pdf
- Mastering CMake, Kitware Inc, 5e édition ISBN 978-1-930934-22-1

CMake - présentation

- Écrit en C++
- Syntaxe spécifique, fichier `CMakeLists.txt`
- Multi-plateformes (Unix, MacOS X, Windows)
- Génération du système de construction en fonction de la plateforme
 - Makefiles sur systèmes Unix
 - Projets Visual C++
 - Projets eclipse
- Outils supplémentaires : CDash, CPack, CTest

CMake - principe d'utilisation

- Créer un fichier `CMakeLists.txt` dans chaque répertoire du projet
- Utiliser la commande `cmake`
- Utiliser les outils natifs (`make; make install`) avec le système de construction généré
- La configuration utilise un mécanisme de cache (`CMakeCache.txt`) pour mémoriser les informations.

CMake - Hello World

src/main.c

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    print("Hello, World\n");
    return 0;
}
```

CMakeLists.txt

```
PROJECT(hello)
SUBDIRS(src)
```

src/CMakeLists.txt

```
SET(hello_SRCS main.c)
ADD_EXECUTABLE(hello ${hello_SRCS})
```


CMake - exemple de bibliothèque

CMakeLists.txt

```
PROJECT(foo)  
SUBDIRS(src)
```

src/CMakeLists.txt

```
SET(foo_SRCS foo.c  
ADD_LIBRARY(foo SHARED ${foo_SRCS})
```

Remplacer **SHARED** par **STATIC** pour bibliothèque statique.

CMake - installation

CMakeLists.txt

```
INSTALL(TARGET hello DESTINATION bin)
INSTALL(TARGET foo DESTINATION lib)
```

Installe `hello` et `libfoo.so` dans
`$prefix/bin` et `$prefix/lib` respectivement.

CMakeLists.txt

```
INSTALL(FILE README.foo DESTINATION share/doc)
```

Installe le fichier `README.foo`.

CMake - langage de CMakeLists.txt

- Langage de scripts avec une syntaxe simple
- Commentaires : #
- Commandes : **COMMAND**(*arg1 arg2 ...*)
- Listes : **A**; **B**; **C**
- Variables : **\$VAR**

CMake - structures de contrôle

Condition

```
IF (CONDITION)
    MESSAGE ("Yes")
ELSE (CONDITION)
    MESSAGE ("No")
ENDIF (CONDITION)
```

Macros

```
MACRO (MY_MACRO arg1 arg2)
    SET ($arg1 "$$arg2")
ENDMACRO (MY_MACRO)
MY_MACRO (A B)
```

Boucles

```
FOREACH (c A B C)
    MESSAGE ("$c: $$c")
ENDFOREACH (c)
```

CMake - composition de macros

exemple

```
MACRO (CREATE_EXECUTABLE NAME SOURCES LIBRARIES)
    ADD_EXECUTABLE($NAME $SOURCES)
    TARGET_LINK_LIBRARIES($NAME $LIBRARIES)
ENDMACRO(CREATE_EXECUTABLE)

ADD_LIBRARY(foo foo.c)
CREATE_EXECUTABLE(hell main.c foo)
```

CMake - création de variables

Création d'options booléennes, utilisables en ligne de commande

exemple

```
OPTION(DEBUG "Programme en mode DEBUG" OFF)

IF(DEBUG)
    SET_SOURCE_FILES_PROPERTIES(main.c COMPILE_FLAGS -DDEBUG)
ENDIF(DEBUG)
```

cmake -DDEBUG:BOOL=ON

CMake - gestion des dépendances

`FIND_PACKAGE(BAR)`

- utilise `FindBAR.cmake`

utilisation

```
PROJECT(myProject)
FIND_PACKAGE(PNG)
IF(PNG_FOUND)
    INCLUDE($PNG_USE_FILE)
ENDIF(PNG_FOUND)
ADD_EXECUTABLE(myProject myProject.cxx)
TARGET_LINK_LIBRARIES(myProject PNG)
```

CMake - pkg-config

- pkg-config n'est pas disponible sous Windows
→ à éviter avec CMake si support Windows
- module `FindPkgConfig.cmake`

exemple

```
FIND_PACKAGE(PkgConfig)
IF(PKG_CONFIG_FOUND)
  PKG_CHECK_MODULE(GTK REQUIRED gtk+-2.0)
  IF(GTK_FOUND)
    # GTK_CFLAGS GTK_LIBRARIES sont definis
  ENDIF(GTK_FOUND)
ENDIF(PKG_CONFIG_FOUND)
```


CMake - packaging avec CPack

- CPack génère des packages source et binaires
 - RPM, DEB paquets binaires pour Linux
 - Mac OS X .pkg et .dmg
 - installeurs NSIS pour Windows
 - archives sources .tgz et .tar.bz2
- Utilise les déclarations **INSTALL** de CMake
- Important : définir les variables **avant INCLUDE(CPack)**

CPack - exemple

CMakeLists.txt

```
INCLUDE(InstallRequiredSystemLibraries)

SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "My Package")
SET(CPACK_PACKAGE_VENDOR "Matthieu Herrb")
SET(CPACK_RESOURCE_FILE_LICENSE
    "$CMAKE_CURRENT_SOURCE_DIR/COPYING")
SET(CPACK_PACKAGE_VERSION_MAJOR "0")
SET(CPACK_PACKAGE_VERSION_MINOR "1")
SET(CPACK_GENERATOR "DEB")
SET(CPACK_DEBIAN_PACKAGE_MAINTAINER "Matthieu Herrb")
...
INCLUDE(CPack)
```

Utilisation : **make package**

- Licence MIT
- Créé en 2000 (suite au projet Cons en Perl)
- Écrit en python
- Syntaxe python dans des fichiers SConstruct
- Outil interprété
- Multiplateforme (sur toutes les plateformes avec Python)
- Remplace make, nmake.exe, msbuild.exe, etc ...
- Détection des outils de compilation, d'édition des liens, ...
- Génération des "espaces de travail" Microsoft
- Petits "extras" : SWIG, Graphviz, CDash, CPack, CTest
- Support pour C, C++, D, Java, Fortran, Yacc, Tex, Qt, SWIG, ...

- <http://fr.wikipedia.org/wiki/SCons>
- <http://www.scons.org/documentation.php>



Ant : Another Neat Tool²



- Ant¹ est sous licence Apache et a été créé en 2000.
- Le projet est hébergé par la fondation Apache et est écrit en Java.
- Il permet de faire l'appel aux compilateurs, aux générateurs de documentation (Javadoc), aux générateurs de rapports, aux vérificateurs de style (checkstyle), ...
- Il peut être utilisé avec C/C++ via des contributions au projet
- Il est devenu omnipresent dans le monde Java.

1. <http://www.projet-plume.org/fr/fiche/ant>

2. http://fr.wikipedia.org/wiki/Apache_Ant

maven

- Maven est sous licence Apache et a été créé en 2002.
- En gros, il reprend tout ce que fait Ant, mais essaye d'en simplifier la configuration pour les gros projets.
- Il essaye de standardiser les projets.
- Il apporte une très bonne gestion des dépendances.
- Il a une orientation réseau (si les sources ne sont pas là, il va les chercher)
- Il n'y a plus de support pour C/C++ mais il en existe pour .Net
- La version 3 devrait support Ruby et Groovy

3. http://fr.wikipedia.org/wiki/Apache_Maven

Agenda

- 1 Introduction
- 2 Outils généraux
 - Autotools
 - CMake
 - Autres
- 3 Outils spécifiques
 - Outils liés à une distribution
 - Outils liés à un langage
- 4 Conclusion

Utilisé sur Debian et dérivés (Ubuntu,...)

<http://www.debian.org/doc/manuals/maint-guide/index.en.html>

<http://www.lucas-nussbaum.net/blog/?p=676>

Utilisé par RedHat, Fedora, SuSE, Mandriva, et dérivés

<http://www.rpm.org/wiki/Docs>

Outils liés à un langage

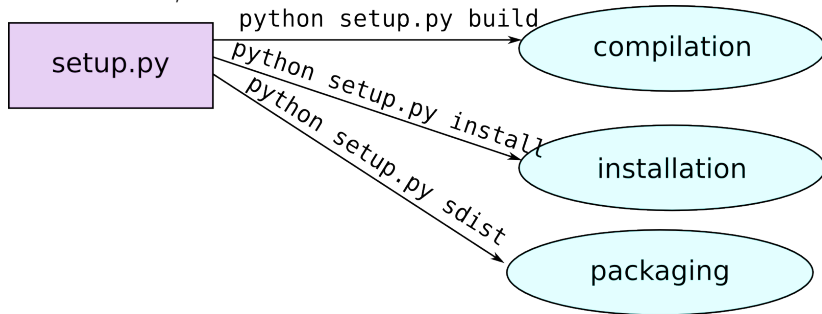
Python - distutils

`http://docs.python.org/distutils/`

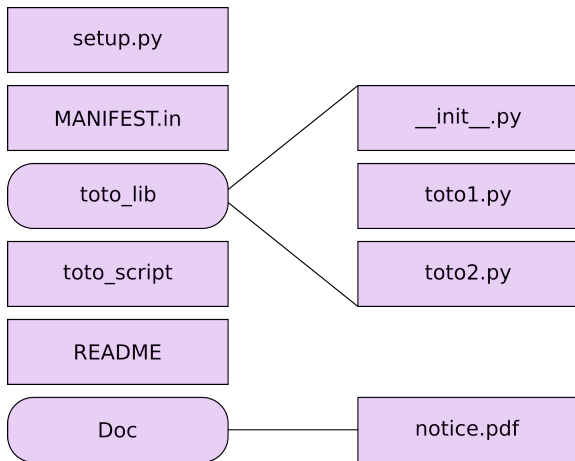
`http://peak.telecommunity.com/DevCenter/setuptools`

distutils - principe

Description des modules Python, C/Cython, scripts, fichiers de documentation,...



distutils - arrangement des fichiers



setup.py

```
from distutils.core import setup

setup (name = "Toto",
       version = "1.0",
       description = "Un logiciel exceptionnel",
       author = "Konrad Hinsén",
       author_email = "konrad.hinsén@cnsr-orléans.fr",
       url = "http://forge.cnsr.fr/toto",

       packages = ['toto_lib'],
       scripts = ['toto_script'])
```

MANIFEST.IN

```
include README  
include Doc/*.pdf
```

Sert à spécifier les fichiers non-exécutables à inclure dans la distribution.

TOTO_SCRIPT

```
#!/usr/bin/python  
  
from toto_lib.toto1 import a  
from toto_lib.toto2 import b  
  
print a + b
```

`http://www.perlmonks.org/?node_id=158999`

`http://docs.rubygems.org/read/book/1`

Agenda

- 1 Introduction
- 2 Outils généraux
 - Autotools
 - CMake
 - Autres
- 3 Outils spécifiques
 - Outils liés à une distribution
 - Outils liés à un langage
- 4 Conclusion

Conclusion

- construction et packaging sont importants
- bien identifier le public visé
- choisir les formats et les outils adaptés
- liens avec les méthodes agiles (intégration continue)
- liens avec les forges et leurs outils

Impact de la taille de l'équipe

(informaticien travaillant seul ou à plusieurs sur le projet)

- Seul → favoriser la distribution source
- A plusieurs
 - rôle de « *release manager* »
 - un spécialiste par plateforme cible

Type de développement

- travail développé uniquement en interne
 - distribution source
 - outils maison
- avec de la main d'oeuvre externe (CDD, stagiaire, société de service, autres services ...)
 - importance des outils standard
 - paquets binaires peuvent être nécessaires en fonction de la licence...
- le projet démarre et est nouveau,
 - occasion de mettre en place un système d'intégration continue moderne ?
 - mais : rester modeste au début
- c'est une reprise/maintenance d'un logiciel existant (pas, peu ou bien documenté, développeurs initiaux joignables ou non ...)
 - réutiliser le système de build existant si possible
 - ne pas hésiter à le refaire si mauvais, trop exotique,...

Types d'utilisateurs

- utilisateurs uniquement propres à l'équipe qui développe,
 - partage uniquement via le gestionnaire de versions ?
- pour une autre équipe, pour un département, l'INRA,
 - Fournir des versions stables et validées
 - utilisation d'une forge interne...
- diffusé à l'extérieur ?
 - Importance de la qualité des paquets diffusés
- logiciel libre/propriétaire,
- gratuit ou payant ?

Questions ?