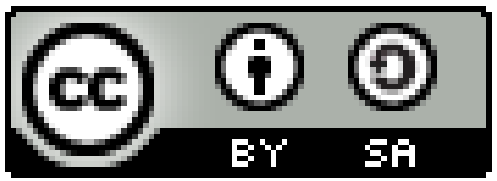
Construction et empaquetage de logiciel

Matthieu Herrb

Ecole ENVOL, La Rochelle, 30 novembre 2016

http://homepages.laas.fr/matthieu/envol-16/

Licence



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

Agenda

Agenda

Introduction

Construction et empaquetage *(packaging)* **:**

À partir d'un code source :

- fournir un ensemble exécutable et installable
- fournir des paquets :
  - sources,
  - binaires
- intégration continue, containers,...

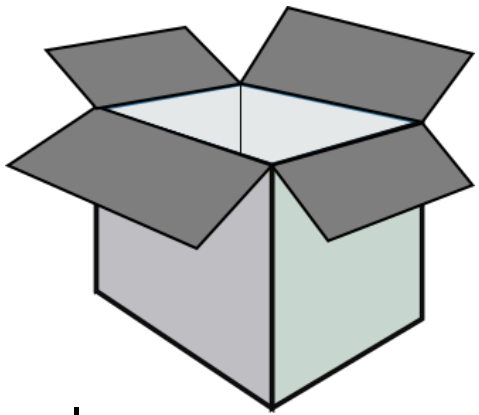
Le point de vue de l'utilisateur

L'utilisateur :

- recherche des instructions simples (fichier `README` ou `INSTALL`)
- aime bien suivre des procédures connues
  - `configure ; make ; make install`
  - `cmake .. ; make ; make install`
  - `dpkg -i supersoft.deb`
  - `curl http://blah/... | bash`, (Non!!!)
- préfère ne pas avoir des dizaines de dépendances à installer manuellement avant de pouvoir tester un logiciel.

Une tâche complexe

Le paquet logiciel est le premier contact avec le logiciel.
Il faut donc le soigner pour donner envie d'aller plus loin;



- robustesse face à :
  - variabilité des environnements,
  - variabilité des utilisateurs,
  - attentes de l'utilisateur,
- dépend du type de public ciblé,
- dépend de facteurs techniques mais aussi « culturels »,
- contraintes liées à la licence d'utilisation

→ aspect important à ne pas négliger

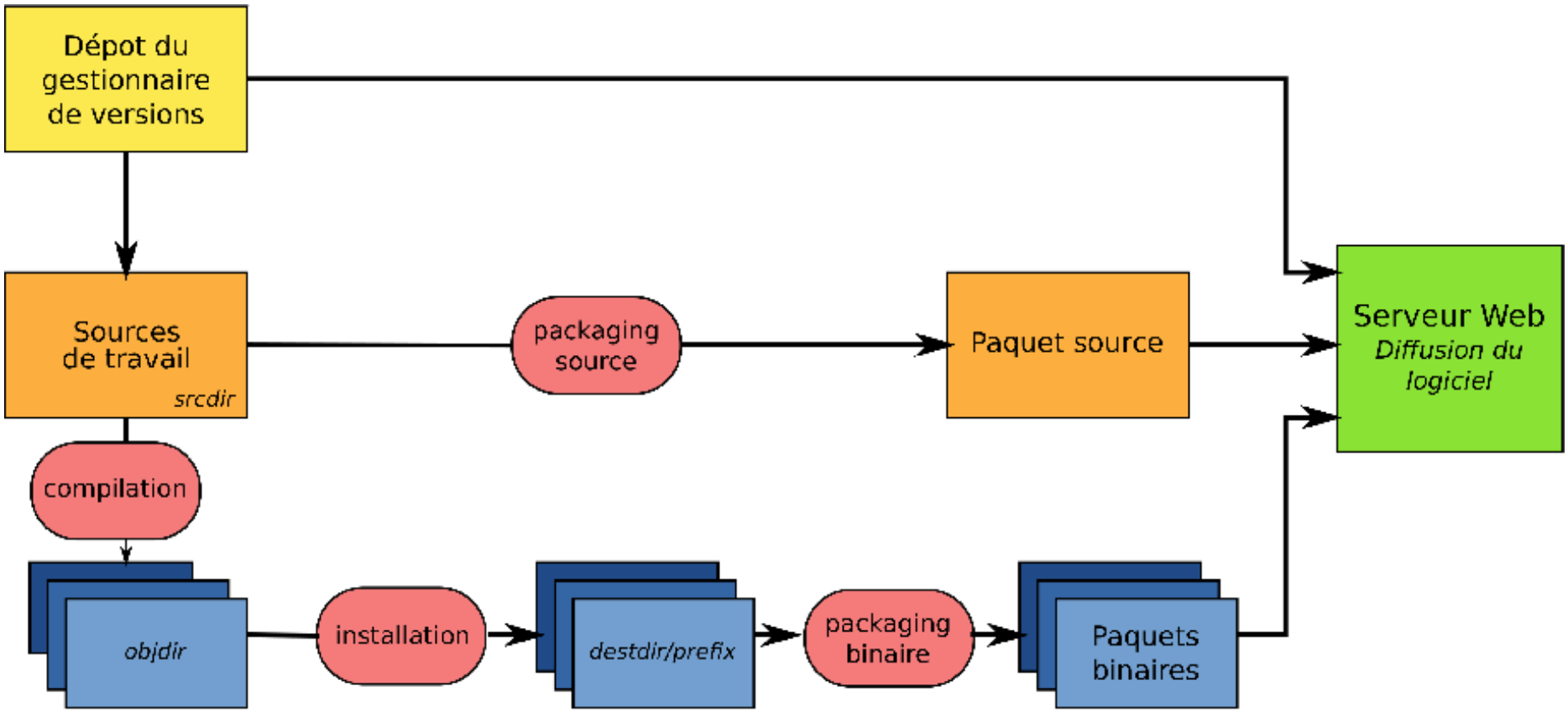
Agenda

Démarche générale

Automatiser :

- Configuration
- Compilation, génération de fichiers annexes (documentation,...)
- Installation
- Construction des paquets sources et binaires
- Mise en ligne des paquets

Démarche générale



Intégration continue

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

- configuration
- compilation
- installation
- exécuter tests unitaires
- desinstallation

Utilise les mêmes outils.
Permet de produire des instantanés (*snapshots*) quotidiens du développement

Bibliothèques

Collection de fichiers objets (compilation de fichiers source)

Trois types :

- statique : archive simple
- dynamique : permet le partage en mémoire (*shared*)
- modules : permet le chargement dynamique (`dlopen()`)

Fichiers d'entête `.h`, `.hpp`,... décrivent les interfaces de la bibliothèque (prototypes, définition des classes, templates,...)

Méta-données pour trouver la bibliothèque :

- `pkg-config`,
- module CMake,...

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 autes 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.

Interfaces : versions d'une API

majeur.mineur

- Incompatibilité binaire : augmenter le **majeur**
  - suppression d'une fonction
  - changement incompatible de la signature d'une fonction
  - changement incompatible d'une structure, d'une classe
- Compatibilité ascendante : augmenter le **mineur**
  - ajout d'une fonction
  - ajout d'un membre à la fin d'une stucture, d'une classe
  - ...

Exprimées dans le nom de fichier (`.so.M.m`) ou via le `soname` de l'objet.

Les liens inter-bibliothèques permettent de spécifier la dépendance sur une version particulière.

Configuration

Adaptation du logiciel à une plateforme

2 modes:

- pré-selection 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 $\rightarrow$ assembleur
- assemblage: traduction $\rightarrow$ code machine
- édition des liens : résolution des références aux bibliothèques externes.

Bibliothèque:

- compilation des composants
- création du ficher 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
- ...

Installation

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'instalation
- **prefix**: répertoire de base d'une installation

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

Désinstallation

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 compressée ou ZIP,
  système de gestion de version (« fork me on github »)
- 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, empaquetage 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 (?)

Licences...

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.

Logiciels payants et contrôle des droits

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

Sécurité

Fournir des paquets signés

- paquets sources : signature détachée PGP
- paquets binaires : utilise le mécanisme de signature existant
- mis à disposition via **https** avec un certificat valide

→ intégrité et garantie d'origine du paquet

Fournir du code sans bugs

- impossible... :-)
- nouvelles dispositions législatives (données personnelles)

→ prévoir de gérer des avis de sécurité (cf. envol 2008)

Diffusion des paquets

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...

$\Rightarrow$ Forges, plateformes de diffusion,...

Créer une communauté

Agenda

Outils généraux

- relativement indépendants du langage de progammation et du système
- gèrent la configuration, la compilation, l'empaquetage
- l'empaquetage est une étape parmi d'autres

GNU Autotools

Autotools

Références :

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

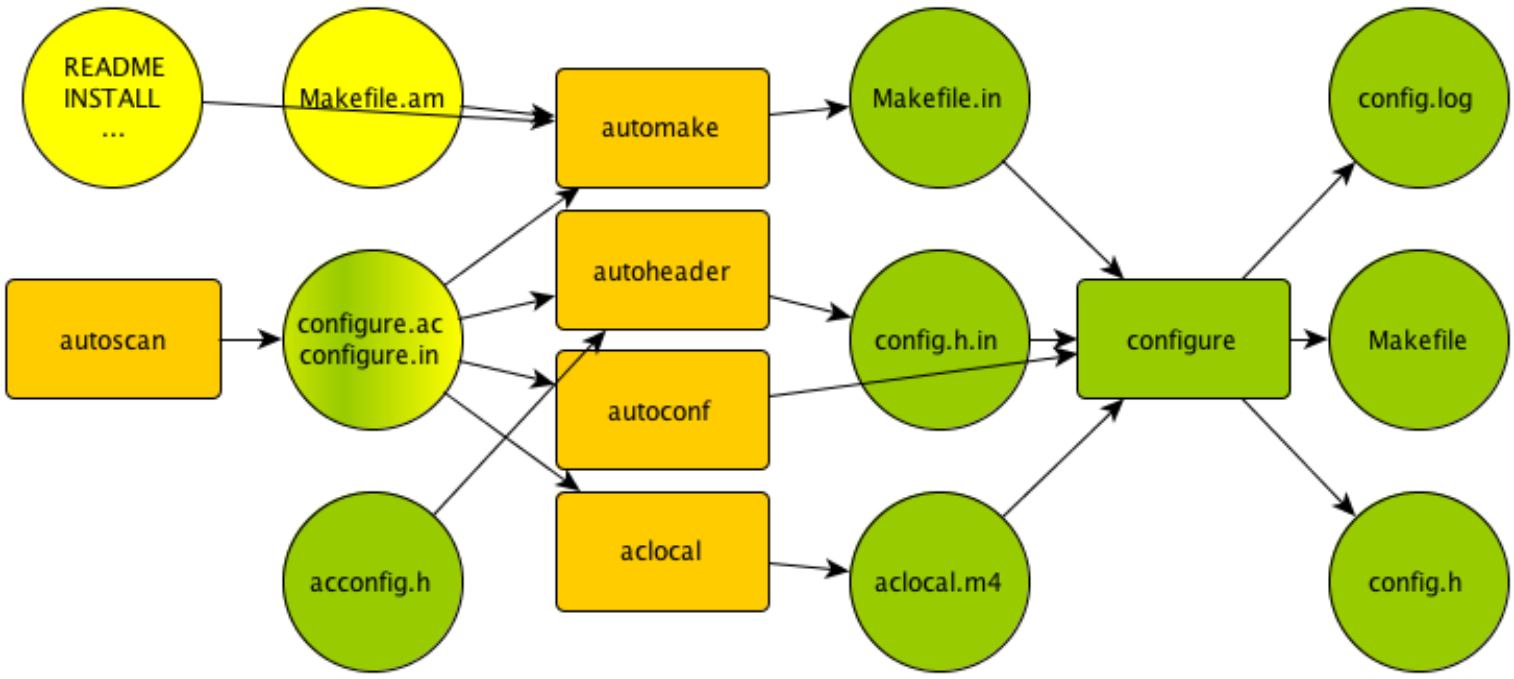
Autotools

```
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_PROC_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 - empaquetage source

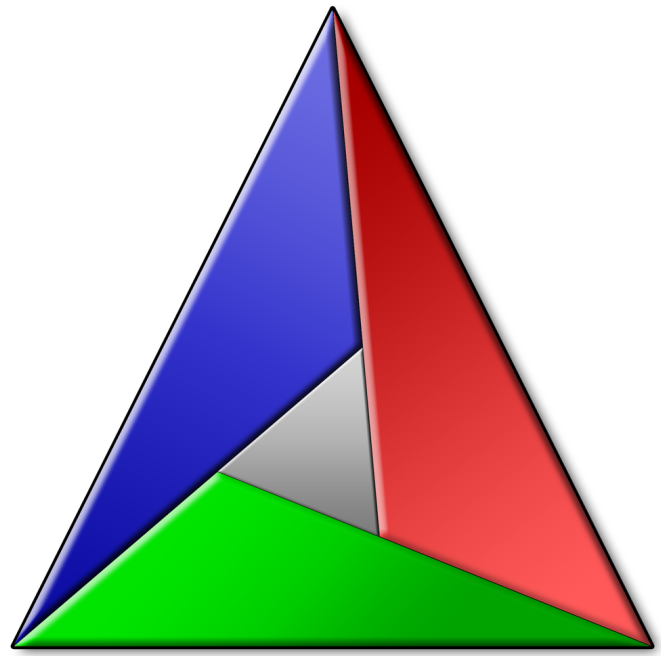
`make distcheck`

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

Autotools - empaquetage binaire

Pas vraiment supporté.

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



CMake

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
- file:///usr/share/doc/cmake-data/html/index.html
  (paquet Ubuntu / Debian `cmake-doc`)
- Mastering CMake, Kitware Inc, édition 3.1 ISBN 978-1930934313

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
  - Ninja
- 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 - fonctions

factorielle !

```
function(fact n result)
  if(${n} LESS 2)
    set(${result} 1 PARENT_SCOPE)
  else()
    math(EXPR a ${n}-1)
    fact(${a} aa)
    math(EXPR r ${n}*${aa})
    set(${result} ${r} PARENT_SCOPE)
  endif()
endfunction()

fact(5 x)
message("Fact(5): ${x}")
```

CMake - création de variables

Création d'options booléenes, utilisables en ligne de commande

exemple

```
option(DEMO "Programme en mode DEMO" OFF)

if(DEMO)
   set_source_files_properties(main.c COMPILE_FLAGS -DDEMO)
endif(DEMO)
```

```
cmake -DDEMO=ON
```

CMake - Modules

Moyen d'étendre CMake. Inclus par la commande `include()`

- « Find » pour recherche de paquets externes
- « System Introspection » pour configuration du système
- « Utilitaires » fonctions d'aide pour tout le reste...

CMake - gestion des dépendances tierces

`find_package(`*BAR*`)`

- utilise le module `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
  $\rightarrow$ 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 - configuration système

Exemple

```
include(CheckIncludeFiles)
include(CheckTypeSize)
include(CheckFunctionExists)

set(INCLUDES "")
check_include_files("${INCLUDES};winsock.h" HAVE_WINSOCK_H)
check_type_size(int SIZEOF_INT)
set(CMAKE_REQUIRED_LIBRARIES m)
check_function_exists(atan2)
```

CMake - config.h

Definition (dans CMakeLists.txt)

```
configure_file(config.h.in config.h)
```

Fichier config.h.in

```
#ifndef _CONFIG_H
#define _CONFIG_H

#cmakedefine HAVE_STDINT_H

#if HAVE_STDINT_H
#include <stdint.h>
#endif

#cmakedefine SIZEOF_INT @SIZEOF_INT@
#cmakedefine HAVE_ATAN2 @ATAN2@

#endif
```

CMake - empaquetage 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`

Introduction à Ninja

https://ninja-build.org/

Ninja est un système de construction de logiciel conçu :

- pour aller aussi vite que possible,
- pour que ses fichiers d'entrée soient produits automatiquement par un mécanisme de plus haut niveau

→ ninja est l'**assembleur** des systèmes de construction.

Ninja - écosystème

- Développé par Google pour Chrome et les projets associés (v8)
- Outils qui produisent des `build.ninja` :
  - gyp pour Google Chrome
  - CMake
  - autres

Ninja - hello world

ninja.build

```
cflags = -g -Wall

rule cc
  command = gcc $cflags -c $in -o $out

build hello: cc hello.c
```

Ninja - utilisation avec CMake

Utilisation du générateur `Ninja`

```
$ mkdir build
$ cd build
$ cmake -G Ninja ..
$ ninja
```

C'est tout...

SCons

- 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.html

Ant : Another Neat Tool[2]

- Ant[1] 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[3]

- 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)

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

Agenda

dpkg

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

http://www.debian.org/doc/manuals/maint-guide/index.en.html
https://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.fr.pdf

rpm/yum

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

http://rpm.org/documentation.html

Snap

Ubuntu 16.04 et après

http://snapcraft.io/docs/build-snaps/

Un paquet snap :

- est un système de fichiers squashFS qui contient le code de l'application et un fichier `snap.yaml` contenant les méta-données spécifiques.
- il a une partie en lecture seule, et une fois installé une zone accessible en écriture.
- est autonome. Il regroupe une grande partie des bibliothèques dont il dépend et peut être mis à jour ou rétrogradé sans affecter le reste du système.
- est isolé du système et des autres applications par des mécanismes de sécurité, mais peut accéder à d'autres snaps via des politiques à grain fin contrôlées par l'utilisateur.

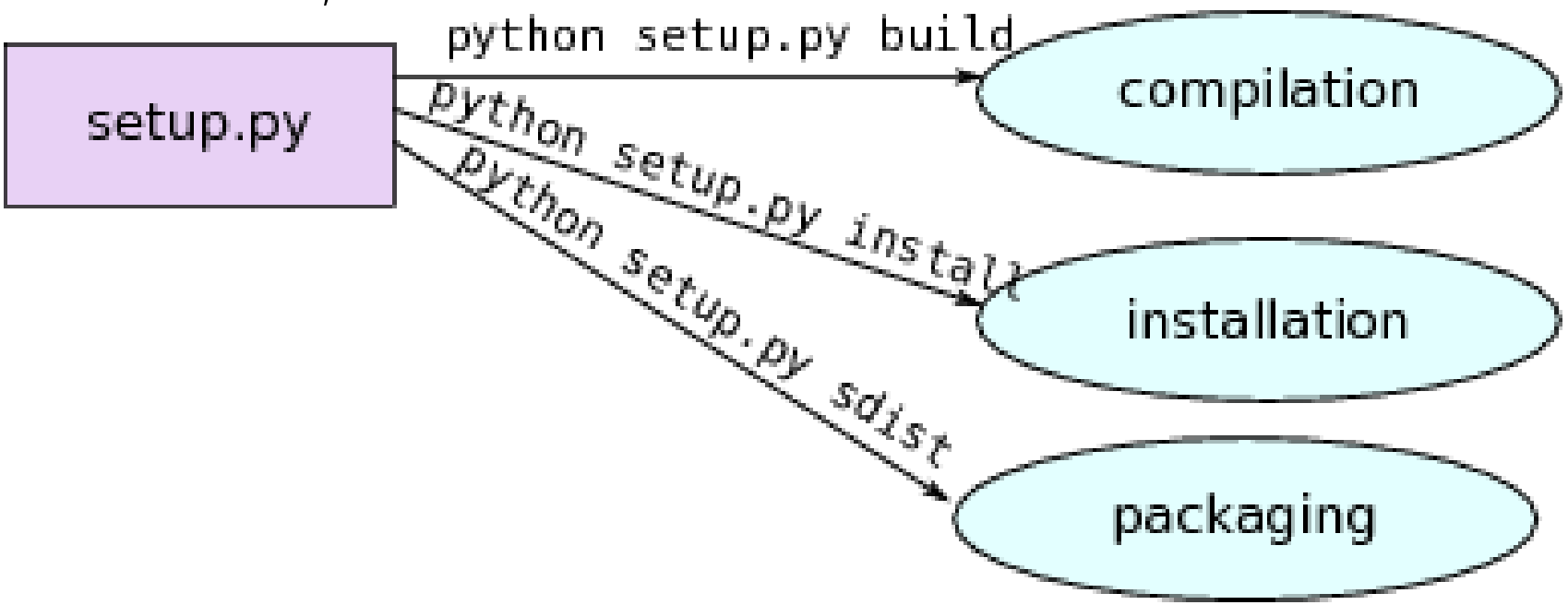
Outils liés à un langage

Python - distutils / setuptools

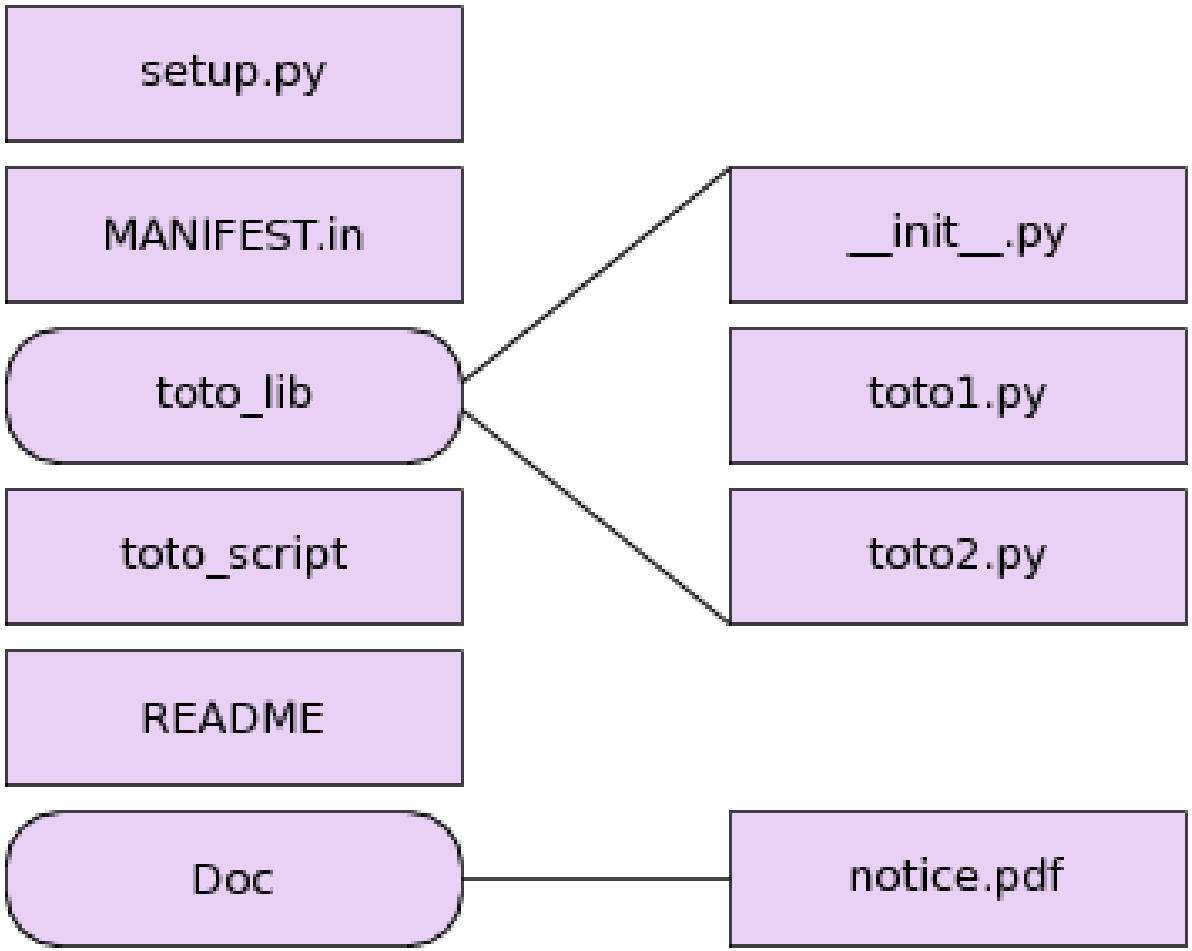
https://packaging.python.org/current/
http://docs.python.org/distutils/
http://peak.telecommunity.com/DevCenter/setuptools

distutils - principe

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



distutils - arrangement des fichiers



distutils - setup.py

setup.py

```
from distutils.core import setup

setup (name = "Toto",
       version = "1.0",
       description = "Un logiciel execeptionel",
       author = "Konrad Hinsen",
       author_email = "konrad.hinsen@cnrs-orleans.fr",
       url = "http://forge.cnrs.fr/toto",

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

distutils - MANIFEST.IN

MANIFEST.IN

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

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

distutils - TOTO_SCRIPT

TOTO_SCRIPT

```
#!/usr/bin/python

from toto_lib.toto1 import a
from toto_lib.toto2 import b

print a + b
```

perl - cpan

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

ruby - gems

http://guides.rubygems.org/make-your-own-gem/

Agenda

Conclusion

- Construction et empaquetage 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 $\rightarrow$ favoriser la distribution source
- A plusieurs
  - $\rightarrow$ rôle de « *release manager* »
  - $\rightarrow$ un spécialiste par plateforme cible

Type de développement

- travail développé uniquement en interne
  - distribtion 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 ?