Autoconf & Automake

Thomas Claveirole

Présentation EPX, 14 Janvier 2005
http://www.epita.fr/~ epx

Copying this document

Introduction

- Simplifier l'écriture des `Makefile`'s.
- Augmenter la portabilité à moindre coût.

Automake facilite l'écriture des fichiers `Makefile`.

Autoconf facilite la générations de `Makefile`'s portables.

Autotool désigne la combinaison des deux outils.

En somme, les Autotools forment un ensemble d'outils pour simplifier et rendre portable les processus de compilations d'un projet.

Table des matières

Pourquoi utiliser les Autotools ? [1/4]

Différents problèmes de portabilité

- Où est le compilateur ?
- Quelle version de `make` est-elle utilisée ?
  Beaucoup de `make` sont pauvres en fonctionnalités.
- Sur quel(s) shell(s) puis-je compter ?
  Un script shell n'est pas portable !
- Les fonctions de bibliothèques dont j'ai besoin sont-elles présentes ?
- Si oui, ou ça ?

Pourquoi utiliser les Autotools ? [2/4]

Différentes nécessités de configuration

- Optimisations en fonction du système (`sizeof`, assembleur inline, ...)
- Adapter l'interface aux périphériques (OSS, ALSA, *DirectSound*, ...)
- ...

Pourquoi utiliser les Autotools ? [3/4]

Premières approches

Portabilité Forcer l'utilisation d'un `make` riche (ex : GNU `make`).

- Contraignant pour l'utilisateur.

Configuration Approche de *X11* : entretenir une base de donnée de `Makefile`'s et de `#define` pour chaque système.

- Difficile à maintenir.
- Inadapté pour des nouveaux systèmes ou des systèmes « bidouillés ».

Pourquoi utiliser les Autotools ? [4/4]

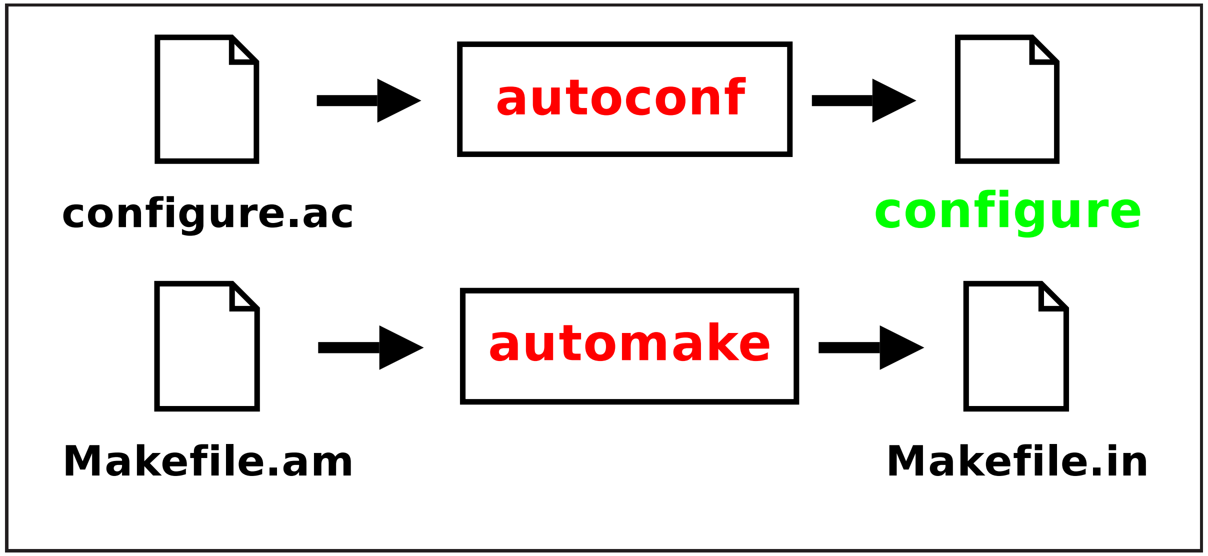
L'approche des Autotools

- Un script `configure` teste la configuration et génère les `Makefile`'s.
  - Compatible avec tous les `make`.
  - Marche sur tous les systèmes, même ceux qui n'existent pas encore…
  - Coûts de maintenance faibles.
- Portabilité du script `configure` :
  - `configure` est généré par `autoconf`.
  - Il n'utilise que des fonctionnalités *basiques* que l'on trouve sur tous les shells.

- Utilisation d’un système de macros : `m4`.
- Le développeur ne se soucie pas de la portabilité et appelle les macros.

Fonctionnement global (très simplifié) [1/3]

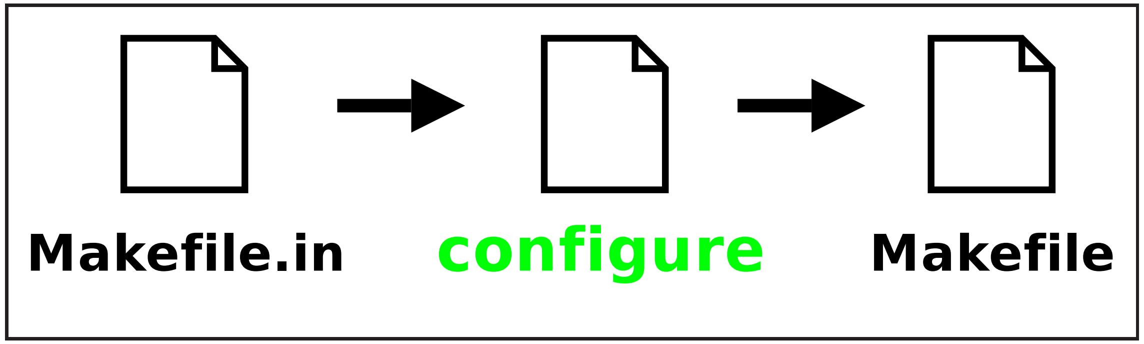
*Travail fourni par le **développeur***



- La première fois, `autoreconf` génère tous les fichiers.
- Ensuite, les `Makefile`'s générés invoquent `autoconf/automake` lorsque c'est nécessaire.

Fonctionnement global (très simplifié) [2/3]

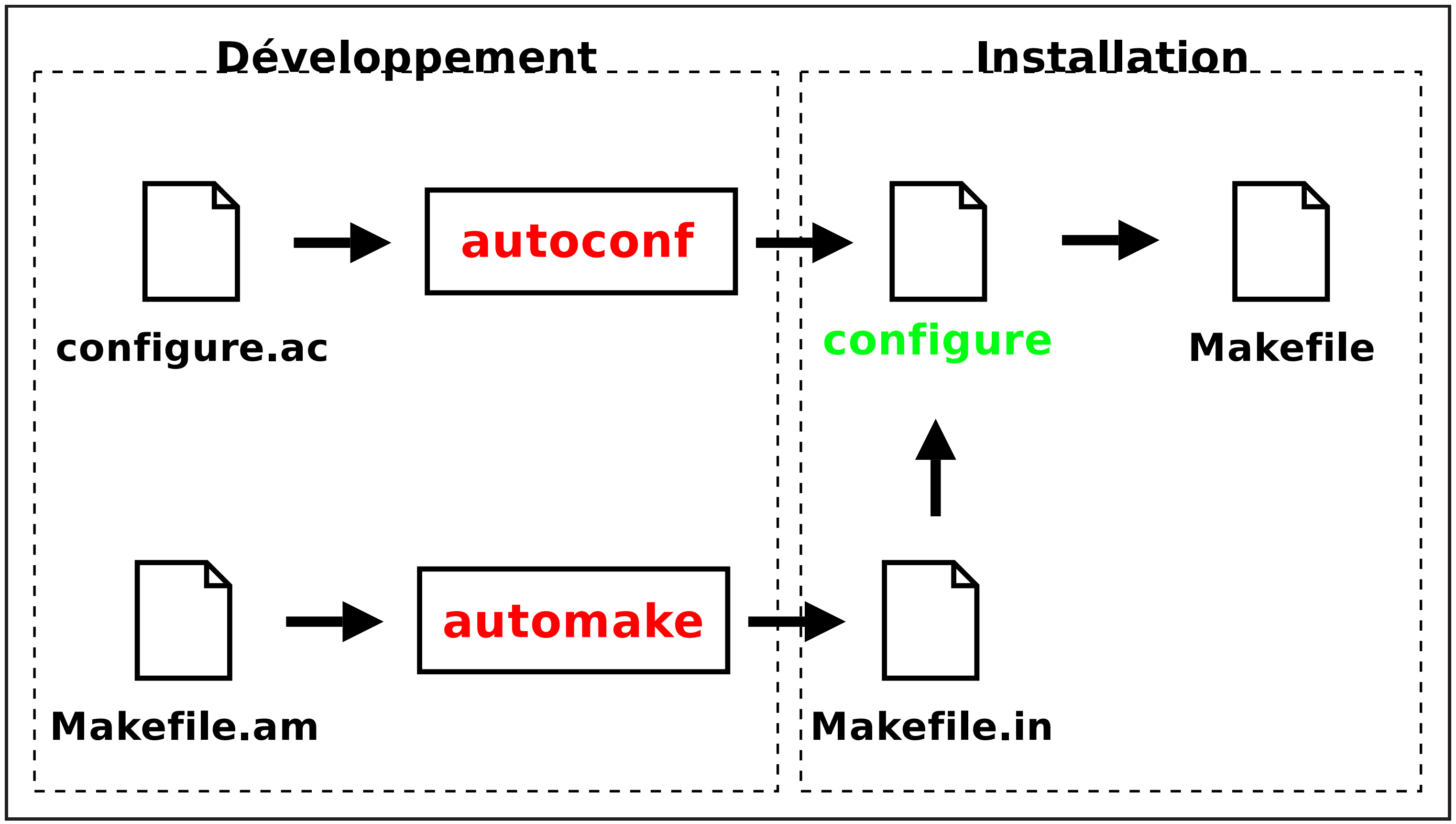
*Travail fourni par l'**installateur***



- A l'installation, les Autotools **ne sont pas nécessaire**.
- Pas de contrainte pour l'utilisateur.

Fonctionnement global (très simplifié) [3/3]

Vision d'ensemble



Autoconf

Fichiers mis en jeu

configure.ac Fichier d’entrée, fourni par le développeur.

- C’est un « méta script shell ».
- En clair, un script shell enrichi avec des macros `m4`.

configure C’est le fichier `configure.ac` après passage de GNU `m4`.
C’est un script shell.

⋆.in Fichiers utilisés par le `configure`.

- Contient des variables à substituer : `@FOO@`.
- `@FOO@` sera substitué au lancement de `configure`.
- `bar.in` va servir à générer `bar`.
- Ces substitutions sont contrôlées par la macro `AC_SUBST`.

Avertissement

- Certaines versions d'Autoconf/Automake sont incompatibles entre elles.
- Le passé, c'est Autoconf 2.13 et Automake 1.4.
- Pour être à jour, vous devez utiliser :
  - Autoconf >2.50 (2.59.)
  - Automake >1.5 (1.9.4.)
- Pour vérifier cela dans un `configure.ac` :

```
AC_PREREQ([2.50])
AM_INIT_AUTOMAKE([1.5])
```

GNU `m4` et Autoconf [1/2]

- Les commentaires se font :
  - Avec `#` (on les retrouve dans `configure`.)
  - Avec `dnl` (`m4` ne les recopie pas dans `configure`.)
- Toutes les macros Autoconf commencent par `AC_`.
- Toutes les macros Autoconf sont en majuscules : `AC_INIT`.
- Un appel de macros se fait de manière « naturelle » :

```
AC_LANG_PUSH(C)   dnl Un argument.
AC_LANG_COMPILER dnl Pas d'arguments.
```

GNU `m4` et Autoconf [2/2]

Les arguments des macros

- Il est conseillé de mettre un argument par ligne !
- Chaque argument est « quoté »avec « `[` »et « `]` »
- Attention, le « quoting »est récursif :

```
AC_CHECK_FUNCS([snprintf fprintf],
               [],
               [AC_MSG_FAILURE([Some functions are missing.])])
```

`configure.ac` minimal

```
AC_PREREQ([2.59])
AC_INIT([foobar-project], [1.0])

...

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

- `AC_PREREQ` demande une certaine version d'Autoconf.
- `AC_INIT` initialise Autoconf.
- `AC_CONFIG_FILES` indique quels sont les fichiers à générer.
- `AC_OUTPUT` effectue réellement les substitutions et génère les fichiers.

Aspect général d'un `configure.ac`

La documentation d'Autoconf recommande la structure suivante :

```
autoconf requirements
AC_INIT(PACKAGE, VERSION, BUG-REPORT-ADDRESS)
information on the package
checks for programs
checks for libraries
checks for header files
checks for types
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services
AC_CONFIG_FILES([FILE...])
```

AC_OUTPUT

Quelques macros utiles…

- `AC_LANG_PUSH` et `AC_LANG_POP` permettent de choisir le langage utilisé.
- `AC_LANG_COMPILER` permet de trouver un compilateur pour le langage en cours :

```
AC_LANG_PUSH([C++])
AC_LANG_COMPILER
AC_LANG_POP
```

- `AC_CHECK_LIB` permet de tester l'existence d'une bibliothèque :

```
AC_CHECK_LIB([GL], [glClearIndex], [], [],
             [$OPENGL_LIBS -lXext -lX11 -lGL])
```

- `AC_CHECK_HEADER` permet de tester l'existence d'un header :

```
AC_CHECK_HEADER([GL/gl.h])
```

- `AC_PATH_X` permet de rechercher les bibliothèques de *X11*.

Quelques macros (moins) utiles. . .[1/2]

- `AC_CONFIG_SUBDIR` permet de lancer la configuration d'un sous projet, possédant son propre script `configure` et situé dans un sous-répertoire.

- `AC_SUBST` permet d'indiquer des variables à substituer :
  ```
  AC_SUBST([FOOBAR]) dnl Will substitute @FOOBAR@ with $FOOBAR
  ```

Quelques macros (moins) utiles. . .[2/2]

- `AC_ARG_ENABLE` permet de rajouter une option au `configure` pour activer/désactiver certaines fonctionnalités :

```
AC_ARG_ENABLE([bar],
              [AC_HELP_STRING([--enable-bar], [Enable bar])],
              [enable_bar=$enableval],
              [enable_bar=no])
```

- `AC_ARG_WITH` est similaire à `AC_ARG_ENABLE` :

```
AC_ARG_WITH([foo],
            [AC_HELP_STRING([--with-foo=bar],
                            [Build with foo located in bar.])],
            [foo=$withval],
            [foo=/usr/share/foo])
```

Autoheader

- Permet de générer des headers avec des définitions en fonction du système :

  ```
  #define HAVE_FOO 0
  #define HAVE_BAR 1
  ...
  ```

- Génère un fichier `config.h.in`.
- Enrichi Autoconf avec ses propres macros pour substituer les headers.
- `info autoheader` pour en savoir plus. . .

Automake

Fichiers mis en jeu

Makefile.am Fichier d'entrée, fourni par le développeur.

- C'est un « méta `Makefile` ».
- En clair, un `Makefile` avec des variables spéciales pour Automake.
- Tout le contenu de ce fichier sera recopié tel quel dans le `Makefile` final.

Makefile.in Fichier généré par `automake`, fourni en entré du script `configure`.

- C'est *toujours* un « méta `Makefile` » !
- En effet, on trouve des variables prêtes pour la substitution par `configure` : `@FOO@`.

Makefile Fichier final, produit par `configure`.

C'est le `Makefile` final.

Les règles des `Makefile`'s générés

all Compile le programme.

check Effectue les tests.

clean Efface les fichiers temporaires.

distclean Efface les fichiers temporaires, ainsi que les fichiers générés par `configure`.

dist Crée une tarball du projet.

distcheck Crée une tarball du projet, puis la décompresse, compile le programme depuis un autre répertoire et lance la batterie de tests.

install Installe le programme sur le système.

uninstall Désinstalle le programme.

Compilation depuis un autre répertoire

```
$ cd /tmp/foobar
$ /path/to/foobar/sources/configure
[...]
$ make
[...]
```

- Cela ne modifiera pas le répertoire avec les sources.
- Cela peut être utilisé pour déporter une partie des accès aux fichiers sur un système local (comprendre : pas sur une NFS qui rame.)
- C'est pour cela qu'il faut utiliser `$(top_srcdir)` et `$(top_builddir)` dans les `Makefile.am`.

Automake et Autoconf

Pour faire fonctionner Automake correctement, il faut ajouter une macro dans `configure.ac` :

```
AM_INIT_AUTOMAKE([1.9 foreign])
```

- `AM_INIT_AUTOMAKE` prends un seul argument.
- Il s'agit d'une liste d'options séparées par un espace.
- Parmi les options classiques, citons :
  - `1.9` : Requiert une version d'`automake` supérieure à 1.9.
  - `foreign` : Certains fichiers, nécessaires pour un projet GNU, ne seront plus obligatoires.
  - `dist-bzip2` : `make dist` générera une tarball au format `bzip2`.

Variables spéciales

- Affectation comme des variables de `Makefile` classiques :
  `bin_PROGRAMS = foobar`
- On dispose en plus d'une opération « += » :
  `LDFLAGS += -ldl`
- Le nom des variables suit **une syntaxe spéciale** :
  - Un nom de variable est composé de deux parties, séparées par un « _ ».
  - La première partie a une signification qui dépends de la seconde.
  - La seconde partie indique « de quoi »l'on parle.
  - Exemple : `bin_PROGRAMS`, `foobar_SOURCES`, ...

Un `Makefile.am` minimal

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

Un Makefile.am plus compliqué

```
SUBDIRS = doc

noinst_LIBRARIES = foobar.a
foobar_a_SOURCES = foobar.c foobar.h
foobar_a_LIBADD = @FOOBAR_LIBADD@

bin_PROGRAMS = foo
foo_SOURCES = foo.c foobar.h
foo_LDADD = foobar.a

sbin_PROGRAMS = bar
bar_SOURCES = bar.c lexer.l parser.y
bar_CFLAGS += -W -Wall -pedantic -Werror
```

Variables spéciales, préfixes intéressants, . . .

- `EXTRA_DIST` permet d'inclure dans la distribution des fichiers supplémentaires :
  `EXTRA_DIST = README ChangeLog NEWS AUTHORS`
- La primaire `SCRIPTS` peut être utilisée pour les scripts.
- Les fichiers qui ne doivent pas être installés, peuvent être préfixés avec `noinst_` :
  `noinst_PROGRAMS = foobar`
- Il ne faut jamais faire référence à un fichier avec des chemins relatifs.
  Pour cela, certaines variables sont réglées par Automake :
  – `$(srcdir)` et `$(top_srcdir)` : répertoires des sources.
  – `$(top_builddir)` : répertoire racine de construction.

Tests

- `TESTS` contient le nom de programmes de tests.
- `XFAIL_TESTS` contient le nom des tests qui sont supposés échouer (retourne non-zéro.)
- La primaire `check` indique les programmes à construire pour `check`.

Exemple :

```
TESTS = misc-tests.sh parsing-test lexing-test
XFAIL_TESTS = globing-test
dist_noinst_SCRIPTS = misc-tests.sh
check_PROGRAMS = parsing-test lexing-test globing-test
parsing_test_SOURCES = parsing-test.c
lexing_test_SOURCES = lexing-test.c
globing_test_SOURCES = globing-test.c
```

Récapitulatif

- Autoconf simplifie l'écriture de scripts `configure` et améliore la portabilité.
- Automake simplifie l'écriture de `Makefile` « sains »et portables.
- Il ne faut jamais appeler ces outils directement.
- Lorsque les `Makefile`'s n'ont jamais été générés, appeler :
  `autoreconf -fvim`
- Les `Makefile`'s générés gèrent correctement les dépendances.
- Il n'est donc plus nécessaire par la suite d'appeler `autoconf`, `automake` ou `autoreconf` directement.

Conclusion

- Très efficace pour obtenir une portabilité forte.
- Automake permet de générer des `Makefile`'s **très simplement**.
- Pour les bibliothèques dynamiques : `info libtool`.
- Autoconf peut s'utiliser individuellement.
- Automake nécessite Autoconf.
- Conçu uniquement pour régler des problèmes de portabilité.
  Ne pas « faire le café »avec.

Conclusion [2/2]

- C'est une artillerie lourde.
- C'est très coûteux en temps d'exécution : `sh`, appels de programmes auxiliaires, nombreux accès aux fichiers, . . .

  Il vaut donc mieux l'utiliser sur un système de fichier local.
- Ce n'est pas la panacée : à éviter sur des petits projets.
  - Les dépendances sont faciles à gérer.
  - Pas de problèmes de portabilité, car `make` est sous-utilisé.

Documentation

- Il existe une documentation très riche.
- Les **pages info** sont la principale source de documentation.
  - `info autoconf`
  - `info automake`
- Sous emacs : `C-h i`.
- Dans un terminal : `apt-get install pinfo && pinfo`.
- Pour les pauvres : `info`.

Références

- autoconf, http://www.gnu.org/software/autoconf/.
- Manuel autoconf.
- automake, http://www.gnu.org/software/automake/.
- Manuel automake.
- Les news EPITA, dans epita.alt.linux.
- Raphaël « Raph »Poss (raph@lrde.epita.fr), grand gourou automaniaque.
- Thomas « Jokeman »Claveirole (thomas@lrde.epita.fr), wannabe gourou.