GNU Autotools

Que sont-ce ?

- **Autoconf** simplifie la configuration
- **Automake** simplifie l'écriture des `Makefile`s
- **Libtool** simplifie la création de bibliothèques dynamiques

⇒ Équipement complet d'un projet (tests, installation, cross-compilation...)
⇒ Portabilité
⇒ Respect des standards de projets GNU ou assimilés

Plan

- Background : problèmes à surmonter et historique
- Principe de fonctionnement
- Applications

Background

Configuration :

- Mais où est le compilateur ?
- Quelles fonctions sont disponibles ?
- Quelles sont les caractéristiques du CPU ?

Make :

- Où sont les bibliothèques ?
- Où installer ?
- Comment connaître les dépendances ?

Autres : chemins d'installation, génération automatique de documentation, batteries de tests...

Portabilité 1/4

Assurer la compilation et le bon fonctionnement sur tous les systèmes

- conformité avec les conventions d'appel (`RETSIGTYPE`, membres de `struct stat`...)
- vérification des fonctions et types disponibles (`strdup`, `alloca`, `roundf`, `long long`...)
- retrouver les *headers* et bibliothèques où sont définies les fonctions (nécessité ou non de `-I`, `-L`, `-lm`, `-ldl` ...)
- bonne utilisation d'extensions spécifiques (`C99`, `__attribute__((unused))`...)

Portabilité 2/4

Changer de comportement selon les systèmes

- optimisation par compilation conditionnelle (ex: `sizeof`, assembleur inline...)
- utiliser le jeu d'appels systèmes local (ex: `exec`/`spawn`)
- adapter le traitement des chemins (ex: `C:\`, `/`, `:`, `.exe`)
- adapter l'interface aux périphériques (ex: son avec OSS, DirectSound, USS...)

Portabilité 3/4

Solutions possibles :

- Répertorier toutes les combinaisons architecture/OS, entretenir des jeux de `Makefile` / `#define` pour chacun

  $\Rightarrow$ difficile à maintenir

- **Autoconf :** tester chaque fonctionnalité requise au moment de la compilation, générer `Makefile` / `#define` en fonction du résultat.

Portabilité 4/4

Pauvreté du `make` standard :

- pas de syntaxe de réécriture de macros universelle
- pas de gestion automatique des dépendances
- peu de règles prédéfinies

$\Rightarrow$ `Makefile` verbeux, beaucoup d'information redondante

Solutions :

- imposer l'utilisation d'une version de `make` riche (GNU Make, dmake...)

  $\Rightarrow$ contraignant pour l'utilisateur

- **Automake :** automatiser l'écriture de `Makefile`'s standards

Historique

David J. MacKenzie (Autoconf/Automake), Akim Demaille (Autoconf), Gordon Matzigkeit (Libtool)

Préhistoire un script shell `configure` transforme `Makefile.in` en `Makefile`

Autoconf 1 (1992) `autoconf` transforme `configure.in` en `configure`

Autoconf 2 (1994) cross-compilation, sous-répertoires, cache des résultat, plus de tests...

Automake (1994) `automake` transforme `Makefile.am` en `Makefile`

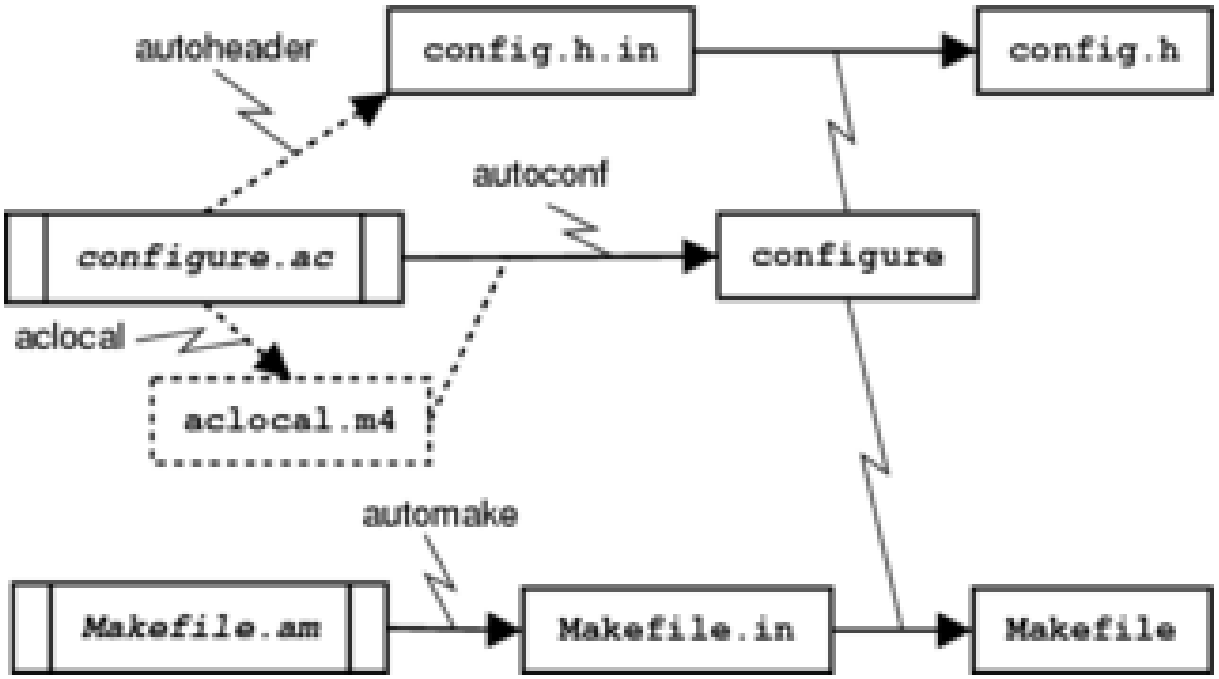
Libtool (1996) change le comportement d'Automake

Anachronismes

Attention aux changements entre Autoconf 2.13 et 2.50, Automake 1.4 et 1.5, Libtool 1.4.2 et 1.5

- Autoconf 2.13 a duré très longtemps $\Rightarrow$ Automake 1.4 répare et complète Autoconf 2.13
- Libtool 1.4.2 adapté à Automake 1.4 et Autoconf 2.13
- Automake 1.5 change l'architecture d'Automake
- Autoconf $>=$ 2.50 incompatible avec Automake 1.4 et Libtool 1.4.2

Principe Général



Autoconf 1/4

configure.ac

- préprocessé par GNU m4
- macros de contrôle : `AC_INIT`, `AC_PREREQ`...
- macro tests : `AC_PROG_CC`...
- macros d'actions résultantes : `AC_SUBST`, `AC_DEFINE`...

configure (shell script)

- dépend d'un shell « standard »
- teste et génère **config.status**
- **config.status** effectue les actions résultantes

Autoconf 2/4

config.h.in

- contient des lignes

  `#undef HAVE_XXX`

- transformé en **config.h** par `config.status` avec `#undef`→`#define`
- contrôle par `AC_DEFINE`
- générable automatiquement par `autoheader`

Makefile.in et autres substitués

- contient des occurences de @VAR@
- occurences de @VAR@ substituées par `config.status`
- contrôle par AC_SUBST

Autoconf 3/4

config.h.in

Makefile.in et autres substitués

Autoconf 4/4

config.h.in

Makefile.in et autres substitués

Automake 1/3

- **Makefile.am** préprocessé par Automake (Perl)
- syntaxe Make *étendue*
- génère des « patrons » pour l'entrée de **config.status**
- génère des règles de regénération récursive

Automake 2/3

Règles de construction :

```
bin_PROGRAMS = 42sh
42sh_SOURCES = main.c
42sh_LDADD = -L. -lshell
42sh_DEPENDENCIES = libshell.a

noinst_LIBRARIES = libshell.a
libshell_a_SOURCES = shell.c global.c
```

Règles de contrôle :

```
CLEANFILES = \*~ \#*
```

Changement du préfixe : `bin_`, `dist_`, `nobase_`

Automake 3/3

Dans le **Makefile** généré :

- `all`
- `dist`
- `check`, `distcheck`
- `doc`
- `install`, `uninstall`
- `clean`
- `distclean`
- `maintainer-clean`
- `Makefile`, `Makefile.in`, `configure`...

Macros disponibles

Dans les pages info de Autoconf et Automake :

Fichiers annexes

`install-sh`, `missing`, `texinfo.tex`, `mkinstalldirs`, `depcomp`...

$\Rightarrow$ utilisés par Automake pour certaines règles

$\Rightarrow$ auto-importés par `autoreconf -i`

Applications

(démonstration à côté)

- utilisation de Lex et Yacc
- utilisation de Texinfo
- utilisation sommaire de `make check`

Ouvertures

- Autotest
- Libtool