

Outils de développement et compilation

Construction de projets avec Make, les Autotools,
CMake

Timothée Ravier

LIFO, INSA-CVL, LIPN

1^{re} année cycle ingénieur STI
2013 - 2014

Make
oooooooooooooooo

Autotools
oooooo

CMake
oo

Les autres...
ooo

Plan global

- 1 Make
- 2 Autotools
- 3 CMake
- 4 Les autres...

1/31

Make
oooooooooooooooo

Autotools
oooooo

CMake
oo

Les autres...
ooo

Plan

- 1 Make
- 2 Autotools
- 3 CMake
- 4 Les autres...

2/31

Make
●oooooooooooooooo

Autotools
oooooo

CMake
oo

Les autres...
ooo

Objectifs

- Automatiser la construction d'un projet :
 - Ne plus avoir à taper 'gcc -o myprog main.c' à répétition ;
 - Éviter les erreurs dans la commande du dessus ;
 - Remplacer les scripts écrits à la main pour chaque projet.
- Gérer dépendances entre les fichiers pour éviter de recompiler les fichiers qui n'ont pas changé depuis la compilation ;
- Ne se limite pas à la compilation de code source.

3/31

Make Autotools CMake Les autres...

Makefile : Exemple

```

$ cat Makefile
all: main.c
    gcc -o main main.c -Wall -Wextra -lpthread

clean:
    rm -f main
$ ls
main.c      Makefile
$ make
gcc -o main main.c -Wall -Wextra -lpthread
$ ls
main.c      main      Makefile
$ make clean
rm -f main
$ ls
main.c      Makefile

```

4/31

Make Autotools CMake Les autres...

Concepts : Cibles (targets) & règles (rules) I

- Un Makefile est constitué d'un ensemble de règles (avec des dépendances) qui visent à construire ou mettre à jour des cibles ;
- Syntaxe d'une règle (tabulation importante, ne pas utiliser d'espaces) :


```

cible1: dependance1 dependance2
    commande1 argument1 argument2
    ...

```
- Lorsque l'on fait appel à `make` sans préciser quelle cible il doit construire, c'est la première cible rencontrée dans le fichier qui est construite. Sinon il construit les cibles indiquées en argument :


```

$ make
$ make cible1 cible2

```

5/31

Make Autotools CMake Les autres...

Concepts : Cibles (targets) & règles (rules) II

- Par défaut, ces cibles correspondent à des fichiers que `make` va essayer de construire ;
- Il est aussi possible d'utiliser des cibles avec des noms ne correspondant pas à des fichiers ;
- Il faut alors l'indiquer à `make` en utilisant une variable spéciale :


```

.PHONY: cible_qui_n_est_pas_un_fichier clean all

```

6/31

Make Autotools CMake Les autres...

Concepts : Cibles (targets) & règles (rules) III

- Pour chaque cible, `make` essaie de construire (ou de mettre à jour) en premier les dépendances listées ;
- Celles ci peuvent correspondre à d'autres cibles ou à des fichiers ;
- `make` peut construire ces dépendances en parallèle :


```

$ make -jX

```
- Avec `X` le nombre de tâches que `make` va pouvoir effectuer en parallèle (généralement on utilise le nombre de cœurs sur un système).

7/31

Concepts : Cibles (targets) & règles (rules) IV

- L'ordre de construction des dépendances n'est pas fixé ;
- Pour les construire dans l'ordre, il faut utiliser le symbole | avant les dépendances concernées (extension spécifique à GNU Make) ;

```
.PHONY: clean build rebuild
```

```
build: main.c
    gcc -o main main.c -Wall -Wextra -lpthread
```

```
clean:
    rm -f main
```

```
rebuild: | clean build
```

Concepts : Cibles (targets) & règles (rules) V

- Une fois les dépendances construites, make exécute les commandes définies dans la règle dans l'ordre et ce tant que les processus ne retournent pas de code d'erreur ;
- De nombreuses règles implicites sont incluses par défaut dans Make et permettent de construire automatiquement certains fichiers ;
- Exemple sans aucun Makefile :

```
$ ls
main.c
$ make main
cc main.c -o main
$ ls
main.c      main
```

Concepts : Variables I

- Il est possible de définir des variables que l'on peut utiliser par la suite dans le Makefile ;
- Un seul type : chaînes de caractères ;
- Exemple :

```
MON_PROGRAMME = programme
SOURCES = main.c fichier1.c fichier2.c
COMPILATEUR = cc
OPTIONS = -Wall -Werror
BIBLIOTHEQUES = -lpthread
```

```
$(MON_PROGRAMME): $(SOURCES)
    $(COMPILATEUR) -o $(MON_PROGRAMME) $(SOURCES) \
        $(OPTIONS) $(BIBLIOTHEQUES)
```

Concepts : Variables II

- Certaines variables sont conventionnellement utilisées :
- ARCH="x86_64"
- CHOST="x86_64-unknown-linux-gnu"
- CPPFLAGS="-D_FORTIFY_SOURCE=2"
- CFLAGS="-march=x86-64 -mtune=generic -O2 -pipe -fstack-protector -param=ssp-buffer-size=4"
- CXXFLAGS="-march=x86-64 -mtune=generic -O2 -pipe -fstack-protector -param=ssp-buffer-size=4"
- LDFLAGS="-Wl,-O1,-sort-common,-as-needed,-z,relro"

Concepts : Variables III

- Les variables peuvent être redéfinies lors de l'appel à make :

```
$ cat Makefile
CC = cc

main: main.c
    $(CC) -Wall -Wextra -o main main.c
$ make main CC=clang
clang -Wall -Wextra -o main main.c
```

Makefile complet

```
.PHONY: clean build rebuild
CC = cc
CFLAGS = -Wall -Wextra
LD_LIBS = -lpthread -lm
FICHIERS = fichier1.c fichier2.c
PROG = monprogramme

build: $(FICHIERS)
    $(CC) -o $(PROG) $(FICHIERS) $(CFLAGS) $(LD_LIBS)

clean:
    rm -f *.~ $(PROG)

rebuild: | clean build
```

Redéfinitions et ajouts de règles

- Il est possible d'ajouter des dépendances à une cible existante sans la redéfinir :

```
ma_cible_deja_definie: dependance_en_plus1 dep2
```

- Il est possible d'ajouter des éléments dans les variables :

```
FICHIERS += fichier_oublie.c
```

Recursive Make

- On peut être tenté de faire des appels à make dans un Makefile ;
- Malheureusement c'est une très mauvaise idée :
 - make a besoin de connaître l'état complet du projet pour pouvoir prendre certaines décisions sur la construction ou la mise à jour d'une cible ;
 - Ces appels récursifs ralentissent la construction du projet sans aucun avantage.
- Explications détaillées en [2] ;

Make Autotools CMake Les autres...

Makefile et inclusions successives

- Évite les problèmes créés par les appels récursifs à make ;
- Solution : utiliser l'instruction #include.
- Exemple :

```
all: monprog

clean:
    rm -f src/monprog

#include src/Makefile.inc
```

16/31

Make Autotools CMake Les autres...

Règles basées sur les motifs (pattern)

- Une règle est basé sur un motif si la cible contient exactement un caractère '%' ;
- make essaie de faire correspondre une chaine de caractères non nulle avec '%' ;
- Fonctionne exactement comme les autres règles ;
- Utilise souvent des variables automatiques :
 - \$@ : le nom de la cible ;
 - \$< : le nom de la première dépendance ;
 - \$^ : toutes les dépendances.
- Exemple :

```
% : %.c
    gcc -Wall -Wextra -O0 -o $@ $< -lpthread
```

17/31

Make Autotools CMake Les autres...

Plan

- 1 Make
- 2 Autotools
- 3 CMake
- 4 Les autres...

18/31

Make Autotools CMake Les autres...

Objectifs

- Remplacer les Makefile écrits à la main pour chaque projet ;
- Utiliser plus de fonctionnalités de Make par défaut ;
- Standardiser le processus :
 - \$ autoreconf -fiv && ./configure && make && make install

19/31

Make Autotools CMake Les autres...

autoreconf : autoconf et automake

- Généralement inclus dans un script appelé `autogen.sh` par convention ;
- Génère :
 - le script `configure` à partir d'un fichier `configure.ac` qui décrit quel programmes et bibliothèques sont nécessaires pour construire le projet (autoconf) ;
 - le fichier `Makefile.in`, pour générer le `Makefile` final (automake) ;

20/31

Make Autotools CMake Les autres...

configure

- Script shell qui génère :
 - le fichier `config.h` qui contient une suite de `#define` à utiliser dans le projet qui défini quelle fonctions sont disponibles suivant le système d'exploitation par exemple ;
 - le fichier `Makefile` à partir des information stockées dans le fichier `Makefile.in`.
- De nombreuses options : on utilise `--help` pour les afficher et utiliser ainsi celles qui sont pertinentes ;
- Exemple : `./configure --libexecdir=/usr/lib --localstatedir=/var --sysconfdir=/etc --enable-feature1 ...`

21/31

Make Autotools CMake Les autres...

configure.ac : exemple

```
AC_INIT(helloworld, 0.1, foo@bar.com)
AC_CONFIG_SRCDIR([hello.c])
AC_CONFIG_HEADER([config.h])
AM_INIT_AUTOMAKE(helloworld, main)

# Checks for programs.
AC_PROG_CC
AC_PROG_INSTALL
AC_PROG_MAKE_SET
# Checks for libraries.
AM_PROG_LIBTOOL
# Checks for header files, typedefs, structures, compiler
# characteristics, library functions...

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

22/31

Make Autotools CMake Les autres...

Makefile

- Fichier `Makefile` classique (généré à partir du fichier `Makefile.am`) ;
- Penser à utiliser les options courante pour changer le répertoire d'installation par exemple :
 - `make DESTDIR="~/build/project/" install`

23/31

Make Autotools CMake Les autres...

Makefile.am : exemple

```
AUTOMAKE_OPTIONS = foreign

CFLAGS = -Wall -pedantic
include_HEADERS = hello.h

lib_LTLIBRARIES = libhello.la
libhello_la_SOURCES = hello.c

bin_PROGRAMS = hello
hello_SOURCES = hello.c
hello_LDADD = .libs/libhello.a
```

24 / 31

Make Autotools CMake Les autres...

Plan

- 1 Make
- 2 Autotools
- 3 CMake
- 4 Les autres...

25 / 31

Make Autotools CMake Les autres...

CMake

- Générateur de fichiers de règles de construction de projets ;
- Supporte les Makefiles, les projets VisualStudio, XCode... (Cross-platform) ;
- Nouveau processus :
 - \$ mkdir build && cd build && cmake (-i) .. && make && make install
- Configuration regroupée dans les fichiers CMakeLists.txt. Un par dossier ;

26 / 31

Make Autotools CMake Les autres...

CMakeLists.txt

- Exemple :

```
cmake_minimum_required(VERSION 2.8)

project(app_project)

add_executable(myapp main.c)

install(TARGETS myapp DESTINATION bin)
```

27 / 31

Make Autotools CMake Les autres...

Plan

- 1 Make
- 2 Autotools
- 3 CMake
- 4 Les autres...

28 / 31

Make Autotools CMake Les autres...

Autres outils

- Il existe plein d'outils pour construire des projets, avec des cas d'usage particuliers :
 - ninja : Make est « lent » pour des gros projets avec beaucoup de fichiers ;
 - scons, ...
 - qbuild : Qt ;
 - Outils de build/packaging lié à un langage de programmation (python, nodejs...).

29 / 31

Make Autotools CMake Les autres...

L'excuse de base du développeur



THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF: "MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.

<http://xkcd.com/303/>

30 / 31

Make Autotools CMake Les autres...

Références I

- 1 Documentation officielle de make : http://www.gnu.org/software/make/manual/html_node/index.html
- 2 Recursive Make Considered Harmful : <http://miller.emu.id.au/pmiller/books/rmch/>
- 3 Autotools Mythbuster : <http://www.flameeyes.eu/autotools-mythbuster/>

31 / 31