Make

J.-F. Lalande

1/31

INSA CVL

3 mai 2016

Mon premier Makefile

Exemple à deux dépendances :

```
prog : foo.o bar.o
⟶gcc -o prog foo.o bar.o

foo.o : foo.c
⟶gcc -c foo.c

bar.o : bar.c
⟶gcc -c bar.c
```

Exécution de l'outil Make :

```
jf@lalande:make/makefiles> make prog
gcc -c foo.c
gcc -c bar.c
gcc -o prog foo.o bar.o
```


Règle d'un Makefile

Format des règles d'un Makefile :

```
cible : prerequis1 prerequis2 prerequis3
⟶commande shell 1
⟶commande shell 2

cible2 : prerequis4 prerequis5 prerequis6
⟶commande shell 3
⟶commande shell 4 \
⟶suite de la commande shell 4 \
```

Une cible peut être :

- Un fichier
- Un mot-clef du makefile

Une pré-requis peut être :

- Un fichier existant
- Un fichier non existant : devient une cible !

Exécuter un makefile

L'exécutable de l'outil Makefile :

- Est le binaire `/usr/bin/make`
- Charge le fichier par défaut du répertoire courant
  - GNUmakefile
  - makefile
  - Makefile
- Charge un fichier explicit

```
jf@lalande:make/makefiles> make prog
gcc -c foo.c
gcc -c bar.c
gcc -o prog foo.o bar.o

jf@lalande:make/makefiles> make -f baba prog
jf@lalande:make/makefiles> make --file baba prog
jf@lalande:make/makefiles> make --makefile baba prog
```


Dépendances temporelles

Makefile construit une cible si :

- la cible n'existe pas
- une dépendance est plus récente que la cible
- avec l'option `-B` (unconditionally make all targets)

```
jf@lalande:make/makefiles> make -f baba prog
gcc -c foo.c
gcc -c bar.c
gcc -o prog foo.o bar.o
jf@lalande:make/makefiles> make -f baba prog
make: « prog » est à jour.
jf@lalande:make/makefiles> touch foo.c
jf@lalande:make/makefiles> make -f baba prog
gcc -c foo.c
gcc -o prog foo.o bar.o
jf@lalande:make/makefiles> make -f baba -B prog
gcc -c foo.c
gcc -c bar.c
gcc -o prog foo.o bar.o
```


Contenu d'un makefile

Un makefile contient :

- Des règles explicites (cf exemple)
- Des règles implicites (donc invisibles !)
- Des définitions de variables
- Des commentaires
- Des directives

Commentaires et inclusions

Commentaires et directives d'inclusion :

```
# Regle de generation du programme
prog : foo.o bar.o
	gcc -o prog foo.o bar.o

# Regle pour foo
foo.o : foo.c
	gcc -c foo.c

include inclu.mk
```

Avec, dans inclu.mk :

```
bar.o : bar.c
	gcc -c bar.c
```


Les variables

Utilisation de variables :

- pour factoriser les répétitions
- paramétrer le makefile

```
OBJ = foo.o bar.o
CC = gcc

prog : $(OBJ)
	$(CC) -o prog $(OBJ)
```

Les variables sont récursivement étendues :

```
OBJfoo2 := foo.o
OBJfoo := $(OBJfoo2)
OBJ := $(OBJfoo) bar.o

prog : $(OBJ)
	echo $(OBJ)
```


Les assignation de variables

Les variables sont des chaînes :

- = Assigne une valeur
- += Ajoute une valeur à la variable
- ?= Assigne si la variable est non définie
- := Force l'expansion de la variable
- != Expansion et exécution du résultat dans le shell
- L'expansion est faite lorsque la variable est utilisée

```
# Pas d'expansion a cette ligne:
OBJ = *.c
# Expansion forcee a cette ligne:
OBJ2 := *.c

prog : $(OBJ)
	echo $(OBJ)
	echo $(OBJ2)
```

echo *.c
bar.c foo.c
echo *.c
bar.c foo.c

Les variables prédéfinies

Pour le C et C++ principalement :

```
CC = cc
CPP = $(CC) -E
LINK.cc = $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(
    LDFLAGS) $(TARGET_ARCH)
LD = ld
CXX = g++
RM = rm -f
CURDIR := /home/jf/cours/make
```

cf *cf make -p*

Ecrasement de variables à l'appel de make :

```
make CC='gcc' CFLAGS='-g' -f baba
```


Les variables automatiques

Règle du type :

cible : dépendance1 dépendance2

Contenu des variables :

- *$@* nom de fichier de la cible
- *$<* la première dépendance de la règle
- *$^* toutes les dépendances
- *$?* dépendances plus récentes que la cible

Accès aux répertoires, fichiers :

- *$(@D)* répertoire de la cible (sans /)
- *$(@F)* nom du fichier cible sans répertoire
- *$(<D)$(<F)* rép. ou fichier de la 1ère dépendance
- *$(^D)$(^F)* rép. ou fichier de toutes dépendances

Exemple de variables automatiques

Exemple du makefile "baba" :

```
# BAba makefile
CC=gcc
prog : foo.o bar.o
→$(CC) -o $@ $^

foo.o : foo.c
→$(CC) -c $<

bar.o : bar.c
→$(CC) -c $<
```

Règles génériques

Règles avec token génériques :

- % : “token” anonyme remplace n’importe quoi
- $\star$: demande l’expansion par n’importe quoi
  - expansion dans les dépendances
  - pas d’expansion dans les variables
- Utiliser "wildcard" pour forcer l’expansion

```
# Pas d'expansion:
objects=*.o
# Pas d'expansion non plus !
prog : *.o
	echo $(objects)
	$(CC) -o $@ $^

# Ok
%.o : %.c
	$(CC) -c $<
```


Règles génériques

```
fichiers=$(wildcard *.c)

# Ok
prog : $(fichiers)
	$(CC) -o $@ $^

# Ok
prog2 : *.c
	$(CC) -o $@ $^

# Imprimer les fichiers C
# plus recents que la derniere impression
print: *.c
	lpr -p $?
	touch print
```


Règles génériques

```
CC=gcc
OBJ=foo.o bar.o

prog : $(OBJ)
	$(CC) $^ -o $@

# Regle implicite
%.o : %.c
	$(CC) $(CFLAGS) $(CPPFLAGS) $(
	    TARGET_ARCH) -c -o $@  $<
```

Règles génériques

Idée : faire des substitutions (remplacement de chaînes)
$(subst .c,.o,$(OBJ))
d'où le Makefile final :

```
CC=gcc
OBJ=$(subst .c,.o, $(wildcard *.c))

prog : $(OBJ)
	echo $(OBJ)
	$(CC) $^ -o $@
```


Cibles virtuelles

.PHONY :

```
.PHONY : clean cleanall

cleanall : clean
	rm -f prog
clean :
	rm -f *.o
```

```
all : server client
.PHONY : all

server : $(SERV_OBJ)
	$(CC) -o server $(SERV_OBJ) $(CFLAGS)
client : $(CLI_OBJ)
	$(CC) -o client $(CLI_OBJ) $(CFLAGS)
```


Conventions pour les cibles

Cibles virtuelles standards :

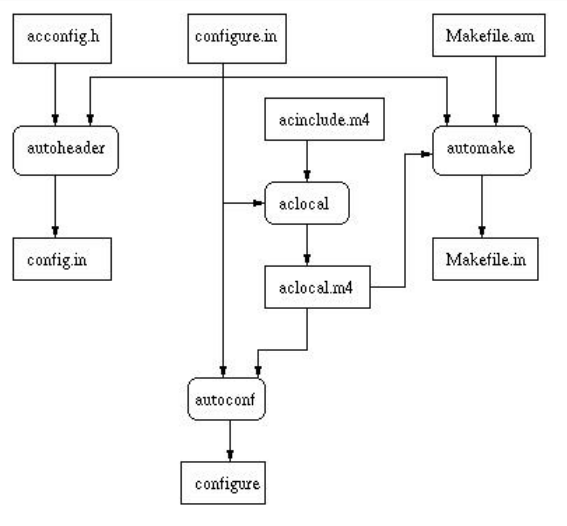
- all : compile entièrement le programme,
- install : compile et installe le programme,
- clean : supprime tous les fichiers créés lors de la compilation,
- dist : créer une archive tar du programme,
- print : affiche la liste des fichiers source ayant changés,

Options de make

Options intéressantes :

- `-f` utiliser tel fichier de makefile
- `-n` simulation des commandes à effectuer
- `-t` marque les cibles comme à jour (sans les reconstruire)
- `--assume-new file` suppose le fichier comme nouveau
- `--assume-old file` suppose le fichier comme non à jour

Principe : un beau bordel tout de même I



Principe : un beau bordel tout de même II

- configure.in : This is the configuration script. This script contains invocations of autoconf macros. It may also contain ordinary shell script code. This file will contain feature tests for portability issues
- Makefile.am : This is the automake input file. It describes how the code should be built. It consists of definitions of automake variables. It may also contain ordinary Makefile targets
- configure : This is the configure script which will be run when building the package. This is generated by ‘autoconf’ from ‘configure.in’ and ‘aclocal.m4’. This is a shell script.

Principe : un beau bordel tout de même III

- Makefile.in : This is the file which the configure script will turn into the ‘Makefile’ at build time. This file is generated by ‘automake’ from ‘Makefile.am’. If you aren’t using automake, you must write this file yourself. This file is pretty much a normal ‘Makefile’, with some configure substitutions for certain variables.

Exemple I

testautomake.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char ** argv)
{
  int a,b;

  a = -1;
  printf("Hello world");
  b = abs(a);

  return 0;
}
```


Exemple II

Makefile.am

```
bin_PROGRAMS = testautomake
testautomake_SOURCES = testautomake.c
```

configure.ac

```
dnl Process this file with autoconf to produce a
    configure script.
AC_INIT(testautomake, 1.5)
AC_CONFIG_SRCDIR(testautomake.c)
AM_INIT_AUTOMAKE

    dnl Set of available languages.
    ALL_LINGUAS="de fr es ko nl no pl pt sl sv"

    dnl Checks for programs.
    AC_PROG_CC
    AC_ISC_POSIX

    dnl Checks for libraries.
```

Exemple III

```
dnl Checks for header files.
AC_STDC_HEADERS
AC_HAVE_HEADERS(stdlib.h stdio.h)

dnl Checks for library functions.
AC_FUNC_ALLOCA

dnl Check for st_blksize in struct stat
AC_ST_BLKSIZE

dnl internationalization macros
AC_OUTPUT([Makefile],
    [chmod +x testautomake])
```


Exemple IV

"Dans le shell"

```
jf@lalande:~/cours/c/sti/cours11> automake
automake: configure.ac: 'PACKAGE' not defined in '
    configure.ac'
automake: configure.ac: 'VERSION' not defined in '
    configure.ac'
automake: configure.ac: required file './install-sh
    ' not found
automake: configure.ac: required file './
    mkinstalldirs' not found
automake: configure.ac: required file './missing'
    not found
automake: Makefile.am: required file './INSTALL'
    not found
automake: Makefile.am: required file './NEWS' not
    found
automake: Makefile.am: required file './README' not
     found
automake: Makefile.am: required file './COPYING'
    not found
```


Exemple V

```
automake: Makefile.am: required file './AUTHORS'
    not found
automake: Makefile.am: required file './ChangeLog'
    not found
automake: configure.ac: AC_ARG_PROGRAM must be used
     in 'configure.ac'
automake: configure.ac: AC_PROG_INSTALL must be
    used in 'configure.ac'

jf@lalande:~/cours/c/sti/cours11> ls Makefile.in
-rw-r--r--  1 jf jf 9,6K 2006-01-06 17:12 Makefile.
    in

autoconf
configure.ac:5: error: possibly undefined macro:
    AM_INIT_AUTOMAKE
      If this token and others are legitimate,
          please use m4_pattern_allow.
      See the Autoconf documentation.
```

Exemple VI

```
ls configure
-rwxr-xr-x  1 jf jf 128K 2006-01-06 17:12 configure

jf@lalande:~/cours/c/sti/cours11> ./configure
./configure: line 1300: AM_INIT_AUTOMAKE: command
    not found
checking for gcc... gcc
checking for C compiler default output file name...
     a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler...
     yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none
    needed
checking for library containing strerror... none
    required
```


Exemple VII

```
checking how to run the C preprocessor... gcc -E
checking for egrep... grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for stdlib.h... (cached) yes
checking stdio.h usability... yes
checking stdio.h presence... yes
checking for stdio.h... yes
checking for working alloca.h... yes
checking for alloca... yes
checking for struct stat.st_blksize... yes
configure: creating ./config.status
```


Exemple VIII

```
config.status: creating Makefile
config.status: executing default commands

ls Makefile
-rw-r--r--  1 jf jf 10K 2006-01-06 17:12 Makefile

jf@lalande:~/cours/c/sti/cours11> make clean
test -z "testautomake" || rm -f testautomake
rm -f *.o core *.core

jf@lalande:~/cours/c/sti/cours11> make
```

Exemple IX

```
gcc -DPACKAGE_NAME=\"testautomake\" -
    DPACKAGE_TARNAME=\"testautomake\" -
    DPACKAGE_VERSION=\"1.5\" -DPACKAGE_STRING=\"
    testautomake\ 1.5\" -DPACKAGE_BUGREPORT=\"\" -
    DSTDC_HEADERS=1 -DHAVE_SYS_TYPES_H=1 -
    DHAVE_SYS_STAT_H=1 -DHAVE_STDLIB_H=1 -
    DHAVE_STRING_H=1 -DHAVE_MEMORY_H=1 -
    DHAVE_STRINGS_H=1 -DHAVE_INTTYPES_H=1 -
    DHAVE_STDINT_H=1 -DHAVE_UNISTD_H=1 -
    DHAVE_STDLIB_H=1 -DHAVE_STDIO_H=1 -
    DHAVE_ALLOCA_H=1 -DHAVE_ALLOCA=1 -
    DHAVE_STRUCT_STAT_ST_BLKSIZE=1 -
    DHAVE_ST_BLKSIZE=1 -I. -I.      -g -O2 -c
    testautomake.c
gcc -g -O2 -o testautomake testautomake.o
```