Programmation Procédurale – make & Makefile

Polytech'Nice Sophia Antipolis

Erick Gallesio

2015 – 2016

Principes

- Une commande: **make**
- Un fichier de specifications: **Makefile**
- Le fichier **Makefile** explicites les dépendances d'un projet
- La commande make:
  - simplifie le processus de compilation
  - assure que les composants d'un projet sont dans un état cohérent
  - permet d'éviter les compilations inutiles

```c
/* Fichier main.c */

#include "hello.h"


int main(void) {
  Hello(); return 0;
}
```

```c
/* Fichier hello.c */

#include <stdio.h>
#include "hello.h"

void Hello(void) {
   printf("Hello, world!\n");
}
```

Makefile

```
# Makefile du programme précédent
main:   main.o hello.o
        gcc -o main main.o hello.o

main.o: main.c hello.h
        gcc -c main.c

hello.o: hello.c hello.h
        gcc -c hello.c
```

Usage

```
$ make
gcc -c main.c
gcc -c hello.c
gcc -o main main.o hello.o
$ emacs main.c
.....
$ make
gcc -c main.c
gcc -o main main.o hello.o
```

Make: macros

make permet de définir des macros

- définition avec `macro=valeur`
- Valeur peut être lue avec `$(macro)`

```
OBJ    = main.o hello.o
CC     = gcc
CFLAGS = -DDEBUG -g

main: $(OBJ)
        $(CC) -o main $(OBJ)
main.o: main.c hello.h
        $(CC) $(CFLAGS) -c main.c
hello.o: hello.c hello.h
         $(CC) $(CFLAGS) -c hello.c
```

Usage

```
$ touch hello.c; make
gcc -DDEBUG -g helloc.c
cc -o main main.o hello.o
```

Make: cibles fictives

Les cibles d'un *makefile*

- fichier (la plupart du temps)
- nom fictif (on dénote une action dans ce cas)

```
# Ajouter ces lignes au Makefile précédent
clean:
	/bin/rm -f main $(OBJ)
print:
	lpr main.c hello.c
```

Usage:

```
$ make clean
/bin/rm -f main main.o hello.o
$ make print
lpr main.c hello.c
```

Make: macro spéciales

- $@ nom de la cible
- $< nom de la première dépendance
- $^ liste des dépendances
- $? liste des dépendances plus récentes que la cible
- $* nom du fichier sans suffixe

```
# Makefile (nouvelle version)
....
main.o: main.c hello.h
        $(CC) $(CFLAGS) -c $*.c
hello.o: hello.c hello.h
        $(CC) $(CFLAGS) -c $<

print: .print

.print: hello.c main.c
        lpr $?
        touch .print
```

Make: règles implicites

Make connaît un certain nombre de règles implicites

- Nom du compilateur C rangé dans `CC`
- Option de compilations rangées dans ‘CFLAGS“
- Passage de `.c` → `.o`:

```
    $(CC) $(CFLAGS) -c fich.c
```

- Simplifications du *Makefile* précédent:

```
main.o : main.c hello.h
hello.o: hello.c hello.h
```

ou encore

```
main.o : hello.h
hello.o: hello.h
```

ou encore

```
$(OBJ): hello.h
```

Note: Les dépendances peuvent être calculées par `gcc` (options `-M` ou `-MM`)

Make: définition de règle implicites

Les règles implicites peuvent être définies avec `.SUFFIXES`

```
.SUFFIXES: .c .c.gz
.c.c.gz:
    gzip -9 $<
.c.gz.c:
    gunzip $<
```

Usage

```
$ make hello.c.gz main.c.gz
gzip -9 hello.c
gzip -9 main.c
```

$\longrightarrow$

```
clean: hello.c.gz main.c.gz
    /bin/rm -f main $(OBJ)
```

GNU-make goodies

- La commande **make** de GNU permet de définir autrement les règles implicites

```
%.c.gz: %.c
    gzip -9 $<

%.c: %.c.gz
    gunzip -9 $<
```

- Autres commandes GNU utiles (voir la doc pour une liste complète)
  - (*wildcard*.c)
  - $(VAR:.c=.exe)

Voir un exemple d'utilisation dans le transparent suivant

Le Makefile complet

Un makefile réaliste pour notre *hello world* pourrait être:

```
CC=gcc
CFLAGS=-Wall -std=gnu99 -O3

SRC=$(wildcard *.c)
OBJ=$(SRC:.c=.o)
EXE=helloworld

$(EXE):  $(OBJ)
    $(CC) -o $(EXE) $(OBJ)

$(OBJ): hello.h

clean:
    rm -f $(OBJ) $(EXE) *~
```

Exemple d'utilisation:

```
$ make
gcc -Wall -std=gnu99 -O3   -c -o hello.o hello.c
gcc -Wall -std=gnu99 -O3   -c -o main.o main.c
gcc -o helloworld hello.o main.o
$ touch hello.c; make
gcc -Wall -std=c99 -O3   -c -o hello.o hello.c
gcc -o helloworld hello.o main.o
$ make clean
rm -f hello.o main.o helloworld *~
```

Un Makefile pour les TDs de C:

```
CC=gcc
CFLAGS=-Wall -std=gnu99 -g

SRC=$(wildcard *.c)
EXE=$(SRC:.c=)

all: $(EXE)

clean:
    rm -f $(EXE) *~
```

Exemple d'utilisation:

```
$ ls
Makefile exo1.c exo2.c
$ make
gcc -Wall -std=gnu99 -g    exo1.c   -o exo1
gcc -Wall -std=gnu99 -g    exo2.c   -o exo2
$ ls
Makefile  exo1  exo1.c  exo2  exo2.c
$ make clean
rm -f exo1 exo2 *~
$ ls
Makefile exo1.c exo2.c
```