Makefile

Comment automatiser la chaine de compilation

Yves Stadler

Codasystem, UPV-M

22 mars 2012

Historique et description

- Il existe de multiples utilitaires makefile.
- Ce n'est pas normalisé
- Nous parlerons de GNU make.

Utilité

- Permet d'exécuter des commandes séquentiellement
- A la différence des scripts, make n'exécute les commandes que si nécessaire
- Make facilite la compilation et les éditions de liens

Historique

- Docteur Stuart Feldman, en 1977.
- Bell Labs
- Deux principaux, BSD et GNU
- Maintenant pour les grands projets, on utilise des outils comme autoconf

Règles

```
cible1: dependance1 dependance2 ...
  commande1
  commande2
  ...

cible2: ...
  ...
```

Projet

- Fichiers des commandes mathématiques
- lmath.c lmath.h main.c

Fonctionnement

lmath.h

```
/* Fonction mathématique : addition */
int ajoute(int a, int b);
/* Fonction mathématique : multiplication */
int multiplie(int a, int b);
/* Fonction mathématique : soustraction */
int soustrait(int a, int b);
/* Fonction mathématique : division */
int divise(int a, int b);
```


Introduction

lmath.c

```
/* Corps de la fonction addition */
int addition(int a, int b) {
  return a + b;
}

/* Corps de la fonction multiplication */
int multiplie(int a, int b) {
  return a * b;
}

/* Corps de la fonction soustraction */
int soustrait(int a, int b) {
  return a - b;
}

/* Corps de la fonction division */
int divise(int a, int b) {
```



```
return a / b;
}
```

Introduction

main.c

```
#include "lmath.h"

int main(void) {
  addition(1, 1);
  multiplication(addition(1,3), 5);
  division(9, 3);
}
```


Première règles

Chaîne

- Compiler lmath.c si on a modifier le corps d'une fonction en fichier .o (sans édition de lien)
- Compiler main.c sans édition de lien si modifié
- Faire l'édition de lien entre les deux fichiers objets

```
gcc -c -o lmath.o lmath.h
gcc -c -o main.o main.c
gcc -o executable main.o lmath.o
```

Makefile de base

main : lmath.o main.o gcc -o executable main.o lmath.o
lmath.o : lmath.c gcc -c -o lmath.o lmath.h
main.o : main.c lmath.h gcc -c -o main.o main.c

Explication

- La première règle est la règle par défaut
- Pour construire la cible main (nom symbolique) nous avons besoin des fichiers lmath.o et main.o
- Si on a ces deux fichiers, on peut exécuter la commande en dessous
- Si il manque un de ces fichiers, on va chercher la règle correspondante
- Si lmath.o n'existe pas, on a une règle nommée lmath.o
- Si lmath.o existe, on vérifie quand même la règle, car lmath.o nécéssite lmath.c. Est-ce que lmath.c à été modifié ?

Problèmes

- Les fichiers temporaires restent sur le disque
- On ne peut pas forcer une génération complète du projet
- On ne peut rien personnaliser

Makefile avancé

- Règle dites "standards"
- all : regroupe toutes les dépendances
- clean : supprime les fichiers intermédiaires
- mrproper : supprime tout ce qui peut-être regénéré

makefile

```
all: main
  #vide

main: lmath.o main.o
  gcc -o executable main.o lmath.o

lmath.o: lmath.c
  gcc -c -o lmath.o lmath.h

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

clean:
  rm -rf *.o

mrproper:
  rm -rf executable
```


Variables

```
#Compilateur
CC=gcc
#Option de compilation
CFLAGS=-Wall
#Option de linkage
LDFLAGS=-lm
#Nom de l'exécutable
EXEC=main

all: $(EXEC)
  #vide

main: lmath.o main.o
  $(CC) -o executable main.o lmath.o $(LDFLAGS)
```

Lexical

```
lmath.o: lmath.c
  $(CC) -c -o lmath.o lmath.h $(CFLAGS)

main.o: main.c lmath.h
  $(CC) -c -o main.o main.c $(CFLAGS)

clean:
  rm -rf *.o

mrproper:
  rm -rf executable
```


On peut écrire

```
main: lmath.o main.o
  $(CC) -o $@ $^ $(LDFLAGS)
```

Va compiler avec gcc -o nom_de_la_cible dépendances flags_de_compilation

Variable propre a makefile

- `$@` est le nom de la cible en cours
- `$@` est le nom du fichier sans extension
- `$<` est le nom de la première dépendance
- `$^` est le nom de toutes les dépendances
- `$?` est le nom des dépendances plus récentes que la cible

Vocabulaire

- Si on regarde notre makefile on verra qu'il y a des règles qui fonctionnent de la même manière

```
lmath.o: lmath.c
  $(CC) -c -o lmath.o lmath.h $(CFLAGS)

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

- On peut généraliser

Règle globale

- `%.o:%.c`
  `$(CC) -c -o $@ $< $(CFLAGS)`
- Cette règle compile tout les fichiers .o de la même manière. (gcc -c -o nom_de_la_cible nom_du_fichier.c flags)
- Attention main.o ne dépend plus de lmath.h
- `main.o: lmath.h`

Petites substitutions

- `SRC=main.c lmath.c OBJ=$(SRC:.c=.o)`
- `src=$(wildcard *.C)`