Compilation avec Make

Slides basées sur celles de Laurent Reveillère

Rohan Fossé

2021-2022

Objectifs

- **Automatiser** la reconstruction d'un exécutable composé d'un ou plusieurs :
  - Modules objets ;
  - Librairies ;
  - Fichier entêtes ...
- **Les avantages**
  - Assure la compilation séparée des différentes ressources ;
  - Utilise des macro-commandes et des variables ;
  - Permet de recompiler uniquement le code nécessaire ;
  - Permet d'utiliser des commandes shell arbitraires
    - Scripts d'installation ;
    - Scripts de désinstallation ;
    - Scripts de nettoyage...

Introduction

Quelques définitions

Cible

- Exécutable à (re)construire ;
- Commande à exécuter (installation, nettoyage,..).

Dépendances

- Élément(s) dont dépend la cible.

Exemple en C

- Cible dépendant d'un fichier *source.c* ;
- Cible construite si le fichier source est plus récent que la cible ;

Fichier Makefile

Fichier contenant la définition des dépendances et des règles de reconstruction des cibles.

Règles de reconstruction d'une cible

Forme générale

CIBLE(s) :DEPENDANCES

<tab> COMMANDE(s)

Contraintes syntaxiques

- Cibles et dépendances sont sur une même ligne ;
- Les commandes commençent **toujours** par une tabulation ;
- Commandes sur plusieurs lignes
  - Chaque ligne est séparée par le caractère \ sauf la **dernière ligne**.

Exemples

Makefile

```
1    dir/prog: a.o b.o
2        cd dir
3        gcc -o prog a.o b.o
4
```

Est-ce-que cette cible est correctement écrite ?

Exemples

Makefile

```
1    dir/prog: a.o b.o
2        cd dir
3        gcc -o prog a.o b.o
4
```

Est-ce-que cette cible est correctement écrite ?

Exemples

Makefile

```
1    dir/prog: a.o b.o
2        cd dir ;\
3        gcc -o prog a.o b.o
```

Est-ce-que cette cible est correctement écrite ?

Exemples

Makefile

```
1    dir/prog: a.o b.o
2        cd dir ;\
3        gcc -o prog a.o b.o
```

Est-ce-que cette cible est correctement écrite ?

Prenons un vrai exemple

C main.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include "hello.h"

int main(int argc, char const *argv[])
{
    Hello();
    return EXIT_SUCCESS;
}
```

Hello > C hello.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include "hello.h"

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

Hello > C hello.h > ...

```c
#ifndef __HELLO_H__
#define __HELLO_H__

void Hello();

#endif
```

Figure 1 – Les fichiers considérés pour l'exemple

A quoi va ressembler le Makefile ?

Hello > M Makefile

```
1  hello: hello.o main.o
2      gcc -o hello hello.o main.o
3
4  hello.o: hello.c hello.h
5      gcc -o hello.o -c hello.c
6
7  main.o: main.c main.h
8      gcc -o main.o -c main.c
```

Evaluation d'un fichier Makefile

Hello > M Makefile

```
1  hello: hello.o main.o
2      gcc -o hello hello.o main.o
3
4  hello.o: hello.c hello.h
5      gcc -o hello.o -c hello.c
6
7  main.o: main.c main.h
8      gcc -o main.o -c main.c
```

- Commande make sans argument
  => La première ligne est évaluée ;
- Commande make avec argument
  => Le nom de la règle passée en argument est évaluée ;
  - exemple : *make hello*
- Évaluation d'une règle :
  - Analyse des dépendances
  - Si une dépendance est la cible d'une autre règle du Makefile, cette règle est à son tour évaluée ;
  - Sinon, les différentes commandes de la règle sont exécutées

Et si je fais un *make* dans notre exemple ?

Hello > M Makefile

```
1  hello: hello.o main.o
2      gcc -o hello hello.o main.o
3
4  hello.o: hello.c hello.h
5      gcc -o hello.o -c hello.c
6
7  main.o: main.c main.h
8      gcc -o main.o -c main.c
```

1. La première règle du Makefile est la règle hello

Et si je fais un *make* dans notre exemple ?

Hello > M Makefile

```
1  hello: hello.o main.o
2      gcc -o hello hello.o main.o
3
4  hello.o: hello.c hello.h
5      gcc -o hello.o -c hello.c
6
7  main.o: main.c main.h
8      gcc -o main.o -c main.c
```

1. La première règle du Makefile est la règle hello
2. Evaluation de la règle hello

Et si je fais un *make* dans notre exemple ?

Hello > M Makefile

```
1  hello: hello.o main.o
2      gcc -o hello hello.o main.o
3
4  hello.o: hello.c hello.h
5      gcc -o hello.o -c hello.c
6
7  main.o: main.c main.h
8      gcc -o main.o -c main.c
```

1. La première règle du Makefile est la règle hello
2. Evaluation de la règle hello
3. Analyse des dépendances

Et si je fais un *make* dans notre exemple ?

1. La première règle du Makefile est la règle hello
2. Evaluation de la règle hello
3. Analyse des dépendances
   3.1 Evaluation des cibles hello.o et main.o

Et si je fais un *make* dans notre exemple ?

Hello > M Makefile

```
1  hello: hello.o main.o
2      gcc -o hello hello.o main.o
3
4  hello.o: hello.c hello.h
5      gcc -o hello.o -c hello.c
6
7  main.o: main.c main.h
8      gcc -o main.o -c main.c
```

1. La première règle du Makefile est la règle hello
2. Evaluation de la règle hello
3. Analyse des dépendances
   - 3.1 Evaluation des cibles hello.o et main.o
   - 3.2 Analyses des dépendances pour la cible hello.o

Et si je fais un *make* dans notre exemple ?

Hello > M Makefile

```
1  hello: hello.o main.o
2      gcc -o hello hello.o main.o
3
4  hello.o: hello.c hello.h
5      gcc -o hello.o -c hello.c
6
7  main.o: main.c main.h
8      gcc -o main.o -c main.c
```

1. La première règle du Makefile est la règle hello
2. Evaluation de la règle hello
3. Analyse des dépendances
   - 3.1 Evaluation des cibles hello.o et main.o
   - 3.2 Analyses des dépendances pour la cible hello.o
   - 3.3 analyses des dépendances pour la cible main.o

Et si je fais un *make* dans notre exemple ?

1. La première règle du Makefile est la règle hello
2. Evaluation de la règle hello
3. Analyse des dépendances
   - 3.1 Evaluation des cibles hello.o et main.o
   - 3.2 Analyses des dépendances pour la cible hello.o
   - 3.3 analyses des dépendances pour la cible main.o
   - 3.4 Exécution des commandes si nécessaires

Et si je fais un *make* dans notre exemple ?

Hello > M Makefile

```
1  hello: hello.o main.o
2      gcc -o hello hello.o main.o
3
4  hello.o: hello.c hello.h
5      gcc -o hello.o -c hello.c
6
7  main.o: main.c main.h
8      gcc -o main.o -c main.c
```

1. La première règle du Makefile est la règle hello
2. Evaluation de la règle hello
3. Analyse des dépendances
   - 3.1 Evaluation des cibles hello.o et main.o
   - 3.2 Analyses des dépendances pour la cible hello.o
   - 3.3 analyses des dépendances pour la cible main.o
   - 3.4 Exécution des commandes si nécessaires
4. Exécution de la commande si nécessaire

Cibles standard

all
Première règle explicite pour définir ce qui est fait par défaut

install
Création de la structure du répertoire, copie des fichiers, ...

uninstall
Défait ce que install à fait mais ne supprime pas les fichiers temporaires

clean
Supprime du répertoire courant tous les fichier temporaires

mrproper
Remet les répertoires sources dans leur état initial

Exemple des cibles standard

Hello > M Makefile

```
1   # la règle all qui défini hello comme première règle à exécuter
2   all: hello
3
4   hello: hello.o main.o
5       gcc -o hello hello.o main.o
6
7   hello.o:
8       gcc -o hello.o -c hello.c
9
10  main.o:
11      gcc -o main.o -c main.c
12
13  # On crée un dossier exec et on y déplace hello
14  install:
15      mkdir exec ;\
16      mv hello exec
17
18  #On supprime le dossier exec
19  uninstall:
20      rm -rf exec
21
22  # On supprime tous les fichiers .o et le fichier hello
23  clean:
24      rm -f *.o hello
25
26  # La règle rmproper est équivalente dans ce cas
27  rmproper:
28      rm -fr exec ;\
29      rm -f *.o hello
```

Notion de variables

Motivations

- Simplifie l'évolution des règles
- Factorise la définition des outils utilisés

Définition

NOM_VARIABLE = VALEUR

Utilisation

$(*NOM_VARIABLE*)

Exemple

Hello > M Makefile

```
1  # On défini la variable de compilation
2  CC = gcc
3
4  foo.o: foo.c
5      $(CC) -c foo.c  # On compile le fichier foo.c
```

Noms de variables standards

CC
Compilateur C

RM
Commande pour effacer un fichier

CFLAGS
Paramètres à passer au compilateur C

LfIAGS
Paramètres à passer à l'éditeur de lien

EXEC
Nom de l'exécutable principal à générer

Exemple

Hello > M Makefile

```
# On défini la variable de compilation
CC = gcc
CFLAGS = -Wall -g -O2 -std=c99
EXEC = hello

# la règle all qui défini hello comme première règle à exécuter
all: hello

$(EXEC): hello.o main.o
    $(CC) -o $(EXEC) hello.o main.o $(CFLAGS)

hello.o:
    gcc -o hello.o -c hello.c $(CFLAGS)

main.o:
    gcc -o main.o -c main.c $(CFLAGS)

# On crée un dossier exec et on y déplace hello
install:
    mkdir exec ;\
    mv $(EXEC) exec

#On supprime le dossier exec
uninstall:
    rm -rf exec

# On supprime tous les fichiers .o et le fichier hello
clean:
    rm -f *.o $(EXEC)

# La règle rmproper est équivalente dans ce cas
rmproper:
    rm -fr exec ;\
    rm -f *.o $(EXEC)

```

Variables automatiques

$@
Nom de la cible de la règle

$<
Nom de la première dépendance

$?
Nom de toutes les dépendances qui sont plus récentes que la cible

Règles implicites

Objectif

- Construction d'un module objet (.o) à partir d'un fichier source (.c)
- Règles appelées automatiquement pour construire toutes les cibles ayant pour suffixe .o

Syntaxe

%.o : %.c

commandes