GCC/Make : génération automatique des dépendances

M Billaud

13 septembre 2018

Table des matières

Dernière révision 15 novembre 2021.

1 Objectif

L'objectif est de produire, au moindre effort, des fichiers `Makefile` qui fonctionnent correctement pour des projets qui mettent en oeuvre la compilation séparée : un exécutable obtenu par édition des liens de plusieurs modules, dont les sources incluent des headers qui éventuellement en incluent eux mêmes.

Explicitation des critères

- `Makefile` qui marche correctement : recompile **tout** ce qui doit l'être, et **rien d'autre**,
- moindre effort : fabrication **automatisée** autant que possible.

Moyens

- on se repose sur la possibilité (option `-MMD`) qu'offre `gcc/g++` de générer, pendant la compilation d'un source, un fichier contenant la liste de dépendances (les fichiers qu'il inclut) ;
- la possibilité pour un `Makefile` d'inclure d'autres makefiles.

2 Exemple de projet

Le projet contient deux sources (`hello.c` et `salutations.c`), et un fichier d'en-tête `salutations.h`.

2.1 Les sources

Programme principal

```c
// hello.c
#include <stdlib.h>
#include "salutations.h"

int main(void)
{
    dire_bonjour();
    return EXIT_SUCCESS;
}
```

Fonctions auxiliaires

```c
// salutations.c
#include <stdio.h>
#include "salutations.h"

void dire_bonjour() {
    printf("Bonjour !\n");
}
```

Prototypes des fonctions auxiliaires

```c
// salutations.h
#ifndef SALUTATIONS_H
#define SALUTATIONS_H

void dire_bonjour();

#endif
```

2.2 Le Makefile

```
CFLAGS = --std=c11
CFLAGS += -Wall -Wextra -pedantic
CFLAGS += -MMD                          # génération des dépendances
CFLAGS += -g

EXECS = hello                           # les exécutables à fabriquer

all: $(EXECS)

hello: hello.o salutations.o            # composition de hello
```

```
-include $(wildcard *.d)                # inclusion des dépendances

clean:
    $(RM) *.o *.d *~

mrproper: clean
    $(RM) $(EXECS)
```

2.3 Principe

1. La ligne "`-include $(wildcard *.d)`" du `Makefile` demande la lecture des fichiers `*.d` qui sont présents dans le répertoire lors du lancement de `make`. Et ne fait rien si il n'y en a pas.
2. La compilation d'un source avec l'option `-MMD` fabrique un fichier de dépendances.

Le premier lancement de `make` fabrique l'exécutable `hello` en laissant aussi comme "sous-produits" des fichiers `hello.d` `salutations.d` (en plus des fichiers objets suffixés "`.o`").

Ces fichiers, qui contiennent respectivement

```
hello.o: hello.c salutations.h
```

et

```
salutations.o: salutations.c salutations.h
```

sont des *makefile*s basiques, contenant seulement une règle de dépendance, qui liste (de façon transitive) tous les fichiers dont dépend un module objet.

Le lancement suivant de `make`, si on a modifié par exemple `salutations.h`, provoquera donc la recompilation des deux sources, comme il est souhaitable, et la mise à jour des dépendances.

3 Conclusion

La méthode présentés ci-dessus ne demande que d'ajouter

- une option `-MMD` pour la compilation
- une ligne `-include $(wildcard *.d)`

et elle évite d'avoir à écrire la liste des dépendances à la main, ce qui est une source habituelle d'erreurs lors des changements

Elle est donc parfaitement recommandable, pour les petits comme pour les grands.

Remarques

- avec `-MMD`, ne figurent que les inclusions "utilisateurs". L'option `-MM` fournit aussi la liste des d'en-têtes système.
- ne pas placer la ligne "`-include`" avant la première cible, sinon la commande `make` va lancer l'exécution de la cible du premier fichier de dépendance.

Problèmes éventuels

- quand on *enlève* des fichiers sources au projet, on peut avoir des soucis avec des dépendances qui citent des fichiers qui n'existe plus.
- Solution radicale : nettoyer le répertoire de ses fichiers `.o` et `.d` par un `make clean`. La prochaine compilation régénère les dépendances.