Makefile, projet avec sous-projet dans une bibliothèque

Michel Billaud

9 janvier 2022

Résumé

Quand un projet contient de nombreux fichiers sources, il est intéressant de produire les fichiers intermédiaires et les exécutables dans des répertoires séparés des sources.

Table des matières

Résumé. Quand un projet atteint une certaine taille, il arrive qu'on puisse en isoler une "bibliothèque" (sous-projet) qu'on espère réutiliser dans d'autres projets. Ici on montre comment organiser et compiler un projet (en C) qui contient un sous-projet. Et surtout on explique les `Makefile`s qui vont avec.

Dernière mise à jour : 9 janvier 2022.

1 Exemple

Le programme principal suivant

```
// main.c

#include <stdio.h>
```

```c
#include "MusicLib.h"

int main() {
    music_play("Pouet-Pouet.mp3");
    music_stop();
}
```

fait appel à deux fonctions d'un sous-projet "MusicLib". Elles sont déclarées dans `MusicLib.h` qui se trouve dans le répertoire du sous-projet.

2 Makefile du projet principal

Le `Makefile` du projet principal

```
CFLAGS = -std=c11 -Wall -Wextra

CPPFLAGS = -IMusicLib/include
LDLIBS   = -LMusicLib/lib/ -lmymusic

all: updates main

updates:
    (cd MusicLib ; make)

main : main.o

clean:
    $(RM) *~ *.o

mrproper: clean
    $(RM) main
    (cd MusicLib ; make mrproper)
```

réalise la fabrication de l'exécutable `main`.

- Pour la compilation, le fichier d'entête `MusicLib.h` est pris dans `MusicLib/include/MusicLib.h` ;
- Pour l'édition des liens, les fonctions `music_play()` et `music_stop()` sont dans la bibliothèque statique `MusicLib/lib/libmymusic.a`.

Déroulement : si on lance la commande `make` avec la cible par défaut (`all`), le sous-projet est d'abord mis à jour pour produire éventuellement une nouvelle version de la bibliothèque `MusicLib/lib/libmymusic.a`. Ensuite, le `main` est fabriqué.

Attention si le `main` existe déjà et est à jour par rapport à `main.c`, il n'est pas recréé si la bibliothèque a été modifiée (on verra peut être plus loin comment arranger ça).

3 Organisation générale des sources

Le projet et le sous-projet sont organisés ainsi

```
.
|-- Makefile
|-- MusicLib
|   |-- Makefile
|   |-- include
|   |   |-- MusicLib.h
|   |   |-- music_play.h
|   |   `-- music_stop.h
|   |-- lib
|   |-- obj
|   `-- src
|       |-- music_play.c
|       `-- music_stop.c
`-- main.c

5 directories, 8 files
```

Le fichier `MusicLib/src/music_play.c` contient le source d'une fonction

```c
#include <stdio.h>

#include "music_play.h"

void music_play(char *filename) {
    printf("vous entendez maintenant %s\n", filename);
}
```

dont l'entête est dans `MusicLib/include/music_play.h`

```c
#ifndef MUSIC_PLAY_H
#define MUSIC_PLAY_H

void music_play(char *filename);

#endif
```

Même chose pour `music_stop`. le fichier `MusicLib/include/MusicLib.h` regroupe les entêtes "exportées" du sous-projet :

```c
#ifndef MUSIC_LIB_H
#define MUSIC_LIB_H

#include "music_play.h"
#include "music_stop.h"

#endif
```

4 Makefile du sous-projet

```
#
# Makefile pour le sous projet MusicLib
#

CFLAGS = -std=c11 -Wall -Wextra
CFLAGS += -MMD
CPPFLAGS = -Iinclude
ARFLAGS = rv

OBJ = music_play music_stop

lib/libmymusic.a: $(addprefix obj/,$(addsuffix .o,$(OBJ)))
    $(AR) $(ARFLAGS) $@ $^

obj/%.o: src/%.c
    $(COMPILE.c) -o $@ $<

-include $(wildcard obj/*.d)

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

mrproper: clean
    $(RM) lib/*.a
```

5 Compilation du sous-projet

Si on exécute ce `Makefile`, on voit passer les étapes de fabrication de la cible par défaut (`lib/libmusic.a`) :

```
$ make
cc -std=c11 -Wall -Wextra -MMD -Iinclude  -c -o obj/music_play.o src/music_play.c
cc -std=c11 -Wall -Wextra -MMD -Iinclude  -c -o obj/music_stop.o src/music_stop.c
ar rv lib/libmymusic.a obj/music_play.o obj/music_stop.o
ar: création de lib/libmymusic.a
a - obj/music_play.o
a - obj/music_stop.o
```

à savoir

1. fabrication (`cc`) des fichiers `obj/*.o` à partir des sources `src/*.c`,
2. fabrication (`ar`) de la bibliothèque à partir des `obj/*.o`.

Remarque : l'option `-MMD` jointe à la directive `-include` permet la gestion automatique des dépendances

6 Compilation du projet complet, exécution

Si on lance maintenant la commande `make` dans le répertoire principal :

```
$ make
(cd MusicLib ; make)
make[1] : on entre dans le répertoire « /xxx/MusicLib »
make[1]: « lib/libmymusic.a » est à jour.
make[1] : on quitte le répertoire « /xxx/MusicLib »
cc -std=c11 -Wall -Wextra -IMusicLib/include  -c -o main.o main.c
cc   main.o  -LMusicLib/lib/ -lmymusic -o main
```

1. le sous-projet est mis à jour et recompilé au besoin (ici, non) ;
2. le source `main.c` est compilé ;
3. l'exécutable est produit par édition des liens avec la bibliothèque.

Il ne reste plus qu'à l'exécuter

```
$ ./main
vous entendez maintenant Pouet-Pouet.mp3
fin de la musique
```

7 Conclusion

Ce qui est présenté ci dessus, c'est le makefile quand le sous-projet est encore instable, c'est à dire en cours de co-développement avec le projet principal.

Quand le sous-projet est stable, il est "sorti" de l'arborescence du projet, et le `Makefile` du projet principal ne demande plus sa re-compilation.

8 Annexe : vue générale des fichiers

```
.
|-- Makefile
|-- MusicLib
|   |-- Makefile
|   |-- include
|   |   |-- MusicLib.h
|   |   |-- music_play.h
|   |   `-- music_stop.h
|   |-- lib
|   |   `-- libmymusic.a
|   |-- obj
|   |   |-- music_play.d
|   |   |-- music_play.o
|   |   |-- music_stop.d
|   |   `-- music_stop.o
|   `-- src
|       |-- music_play.c
|       `-- music_stop.c
|-- main
|-- main.c
`-- main.o
```

```
5 directories, 15 files
```