

Cours 5 : Makefile

1 Structure des fichiers Makefile

Un fichier `Makefile` est constitué de règles de la forme suivante :

<code>cible : liste de dépendances actions</code>

Le programme `make <cible>` cherche à exécuter la règle `cible` du fichier `Makefile` du répertoire courant. Si aucune règle n'est donnée en argument, c'est la première règle de ce fichier qui est exécutée.

L'exécution d'une règle consiste à exécuter d'abord les dépendances de cette règle (dans l'ordre dans lequel elles apparaissent), puis à effectuer les actions correspondantes. Les dépendances sont soit des noms d'autres règles, soit des noms de fichiers ; les actions sont des commandes shell.



Les lignes contenant les actions doivent commencer par une tabulation.

Exemple : la commande `make foo` avec les deux `Makefile` suivant permet de compiler le programme `foo`. Celui de droite apporte en outre de la compilation séparée.

<code>foo : a.ml b.ml ocamlc -c a.ml ocamlc -c b.ml ocamlc -o foo a.cmo b.cmo</code>	<code>foo : a.cmo b.cmo ocamlc -o foo a.cmo b.cmo a.cmo : a.ml ocamlc -c a.ml b.cmo : b.ml ocamlc -c b.ml</code>
--	--

2 Macros et règles d'inférences

Afin de simplifier l'écriture des fichiers `Makefile` et l'obtention de la compilation séparée, l'outil `make` permet de définir des macros de la manière suivante :

<code>NOM = valeur</code>

où `NOM` est le nom de la macro et `valeur` une chaîne de caractères quelconque. On peut ensuite écrire `$(NOM)` n'importe où dans le fichier comme raccourci pour `valeur`.

L'outil `make` comprend un certain nombre de macros prédéfinies, parmi lesquelles :

<code>\$\$</code>	Nom de la cible à reconstruire
<code>\$(</code>	Nom de la dépendance à partir de laquelle la cible est reconstruite
<code>^</code>	Liste de toutes les dépendances
<code>*</code>	Le nom de la cible sans suffixe (cf. règles génériques)

Exemple : Le Makefile de droite ci-dessus peut se récrire de la manière suivante :

```
OCAMLC = ocamlc -c
CMO = a.cmo b.cmo
foo : $(CMO)
    ocamlc -o $@ $^
a.cmo : a.ml
    $(OCAMLC) $<
b.cmo : b.ml
    $(OCAMLC) $<
```

On peut voir que les règles `a.cmo` et `b.cmo` sont très similaires. Afin de les factoriser, `make` autorise la définition de règles génériques de la forme suivante :

<pre>%.xxx : %.yyy actions</pre>

qui permettent de construire, à partir d'un fichier quelconque `foo.yyy`, un fichier `foo.xxx`.

Exemple : On peut encore simplifier notre Makefile de la manière suivante :

```
OCAMLC = ocamlc -c
CMO = a.cmo b.cmo
foo : $(CMO)
    ocamlc -o $@ $^
%.cmo : %.ml
    $(OCAMLC) $<
```

3 Calcul automatique des dépendances

Afin d'appliquer les règles génériques dans le bon ordre, il faut fournir les dépendances mutuelles des fichiers `.cmo`. Cela est calculé automatiquement pour OCaml avec l'outil `ocamldep` dont le résultat peut être inclus directement dans le Makefile avec l'instruction `include`.

```
.PHONY: depend          # permet de forcer l'exécution de la règle depend
OCAMLC = ocamlc -c
CMO = a.cmo b.cmo
foo : depend $(CMO)
    ocamlc -o $@ $^
%.cmo : %.ml
    $(OCAMLC) $<
depend:                  # calcule les dépendances via ocamldep
    ocamldep *.ml > .depend
include .depend         # ajoute les dépendances calculées par ocamldep
```