

Guide_{make} Pour l'Impatient(e)

“make - GNU make utility to maintain groups of programs”, /usr/bin/man 2004.

“Le Guide *make* Pour l'Impatient(e) est un tas de mots gérés par un tas de commandes elles-même gérées par un fichier *Makefile* écrit pour la commande *make*.”, nj 2004.

Point de départ : un ensemble de fichiers, un ensemble de dépendances relatives à ces fichiers.

Objectif : utiliser un fichier de configuration appelé *Makefile* pour automatiser la gestion des dépendances et éviter de retenir des commandes à rallonge. Il est courant pour cela d'utiliser la commande *make*.

1 fichier *Makefile*

Ce fichier contient une liste de dépendances et d'actions. Les dépendances maintiennent la bonne constitution des programmes. Les actions décrivent les commandes à exécuter. Il est spécifié à la commande *make* par l'option `-f nomDuFichierMakefile`. Par défaut, *make* recherche un fichier nommé *Makefile*.

Dans ce fichier :

```
# commente tout ce qui suit sur cette ligne,  
CC = gcc indique à make de remplacer toutes les expressions $(CC) par gcc,  
A.o : définit une référence-cible,  
A.o : A.c A.h définit une référence-cible avec deux dépendances.
```

Le fichier cible **A.o** est valide si les dates de modification des fichiers **A.c** et **A.h** ne sont pas postérieures à celle du fichier **A.o** (**a11** est la cible par défaut de la commande *make*),

```
<tab>$(CC) -o A.c est un exemple de commande.
```

Les commandes sont toujours précédées d'une tabulation. Pour associer des commandes à une référence, il suffit de les placer sur les lignes suivant cette référence,

```
AB.o : A.o B.o impose aux références A.o et B.o invalides d'être reconstruites  
avant l'exécution des commandes associées à AB.o. Une (ou plusieurs) référence(s) est (sont) éva-  
luée(s) par la commande make ref (ref2 ...) (ici make AB.o et make appli propre ci-dessous).
```

exemple :

```
CC = gcc  
CFLAGS = -g -Wall  
obj = A.o B.o  
all : appli  
  
appli : $(obj)  
        $(CC) $(CFLAGS) -o appli $(obj)  
A.o : A.c A.h  
        $(CC) $(CFLAGS) -c A.c  
B.o : B.c B.h  
        $(CC) $(CFLAGS) -c B.c  
propre :  
        rm -f *~ $(obj)
```

lien utile :

– <http://www.gnu.org/software/make/make.html>

Autres options de la commande **make** :

-p	affiche la liste des règles par défaut,
-n	affiche les commandes à exécuter sans les exécuter,
-j 3	tente d'exécuter simultanément au plus 3 commandes indépendantes,
-i	ignore les codes d'erreur des commandes dans l'exécution des commandes,
-r	ignore toutes les règles de dépendances par défaut,
-d	affiche la séquence des traitements testés par la commande make .

2 macro-définitions standard

<code>\$\$</code>	est le nom de la référence-cible en cours d'examen,
<code>\$(</code>	est le nom de la première dépendance de la liste des dépendances.
<code>\$(?</code>	est la liste des dépendances plus récentes que la référence-cible,
<code>\$(^</code>	est la liste de toutes les dépendances,
<code>\$(*</code>	est le nom de la référence-cible en cours de traitement privée de son suffixe.

3 règles génériques et fonctions

Une règle générique est définie par :

`%.b : %.a` tout fichier de suffixe `.b` dépend du fichier de même nom de suffixe `.a`,

La syntaxe d'une fonction est : `$(fonction paramètres)`.

Pour toutes les fonctions suivantes, la valeur de retour est affectée à la variable `RES` :

<code>RES=\$(wildcard *.a)</code>	retourne la liste des fichiers de suffixe <code>.a</code> ,
<code>RES=\$(subst from,to,text)</code>	retourne le texte <code>text</code> en substituant toutes les occurrences de <code>from</code> par <code>to</code> dans <code>text</code> ,
<code>RES=\$(findstring text,find)</code>	retourne le texte <code>find</code> si <code>find</code> est dans <code>text</code> ,
<code>RES=\$(strip text)</code>	retourne le texte <code>text</code> en supprimant les espaces au début et à la fin de <code>text</code> ,
<code>RES=\$(shell ls -l)</code>	retourne le résultat de l'exécution de la commande <code>ls -l</code> ,
<code>RES=\$(VAR: .a=.b)</code>	comme <code>RES=\$(subst .a,.b,\$(VAR))</code> ,
<code>RES=\$(dir src/file.c)</code>	retourne <code>src</code> ,
<code>RES=\$(notdir src/file.c)</code>	retourne <code>file.c</code> ,

Il est possible de vérifier des conditions avec la syntaxe

`ifeq CONDITIONAL-DIRECTIVE <tab>TEXT-IF-TRUE else <tab>TEXT-IF-FALSE endif`.

exemples :

<pre>TEX = latex TEXTOPS = dvips FICHERSTEX = \$(wildcard *.tex) all : \$(FICHERSTEX:.tex=.ps) %.dvi : %.tex \$(TEX) \$(< rm \$(@:.dvi=.aux) rm \$(@:.dvi=.log) %.ps : %.dvi \$(TEXTOPS) \$(< -o \$@</pre>	<pre>OS=\$(shell uname) all : deuxttests deuxttests : ifeq (\$(OS),Linux) echo OS est Linux else echo OS pas Linux mais \$(OS) endif ifeq (1.tex,\$(wildcard 1.tex)) echo 1.tex dans rep courant else echo 1.tex pas dans rep courant endif</pre>
--	---