TD n°8 – Bash, encore

1) Présentation

On voudrait programmer une commande permettant d'afficher les options de compilation nécessaires pour certains logiciels.

Par exemple, mettons que l'un de vos programmes utilise la librairie mathématique, par exemple la fonction sqrt (racine carrée). Vous avez donc un programme source calcul.c qui doit être compilé comme ceci

```
cc calcul.c -o calcul -lm
```

Il faut lui rajouter l'option -lm pour que la fonction sqrt soit ajoutée à votre programme, sinon vous aurez une erreur de type _sqrt : reference not found.

De plus, si votre programme doit être rapide, alors il faut demander au compilateur d'optimiser le code machine, et ça se demande à l'aide de l'option -O3 :

```
cc calcul.c -o calcul -lm -O3
```

Pour finir, il existe un grand nombre de librairies qui demandent chacune leurs options. Par exemple, pour compiler avec la librairie gtk3, voici une toute petite partie ce qu'il faut mettre :

```
cc projet.c -o projet -pthread -I/usr/include/gtk-3.0
-I/usr/include/pango-1.0 -lgtk-3 -lgdk-3 -lgdk_pixbuf-2.0 -lpango-
1.0 -lcairo -lgobject-2.0 -lglib-2.0
```

Le problème est de se souvenir de toutes ces options, c'est impossible. D'autre part, toute erreur empêche la compilation. On voudrait donc un mécanisme pour ne pas devoir mémoriser autant d'options et ne mettre que quelques mots clés clairs comme ceci :

```
cc calcul.c -o calcul options: maths optim
```

Notre souhait est le suivant : on tape la commande de compilation normale puis le mot Options: suivi de mots-clés indiquant quelles options on veut.

2) Réalisation

a) Lancement de la compilation avec les options

Le problème de cette approche, c'est que le mot `options:` est un mot ordinaire et le compilateur ne va pas l'accepter. Il faudrait que ce mot devienne *actif*. Alors l'idée est qu'il soit en fait le nom d'un script et que ce qui est en gras dans le précédent exemple soit remplacé avant l'exécution de la commande cc par toutes les options demandées.

☞ Comment fait-on pour lancer une commande ou un script et faire que ce qu'elle ou il affiche soit mis à la place de son appel ? Que doit-on taper sur la ligne de commande de compilation ?

b) Comment gérer les options de compilation ?

On va donc devoir écrire un script appelé `options:` qui reçoit des mots en paramètres. Ce sont des noms de modules tels que `maths` ou `optim`, et qui affiche des listes d'options à l'écran telles que `-lm` ou `-O3`.

☞ Écrire la première version du script `options:` dans laquelle les options sont codées en dur, c'est à dire qu'en fait le script est une boucle sur ses paramètres et selon chacun, on affiche les options.

c) Séparation entre le script et les options

Une idée qui vient est de placer les options, par exemple `-lm` dans un fichier appelé `maths` et placé dans `/usr/lib/options`. Pareil pour optim : on écrit `-O3` dans un fichier appelé `/usr/lib/options/optim`. C'est alors très simple, il suffit que le script `options:` affiche les contenus des fichiers dont on lui fournit les noms., un peu comme more ou cat, mais en rajoutant un chemin.

Par exemple, si on fait `options: optim` alors il affiche le contenu du fichier `/usr/lib/options/optim`. Si on fait `options: optim maths`, alors il affiche le contenu de `/usr/lib/options/optim` suivi de celui de `/usr/lib/options/maths`.

☞ écrire la deuxième version du script `options:` Il doit vérifier qu'on lui donne bien les noms de fichiers d'options situés dans `/usr/lib/options`.

d) Dépendances en cascade

En fait, c'est plus compliqué : il y a des dépendances, c'est à dire que, par exemple, si on emploie la librairie fftw3 (option -lfftw3), alors il faut aussi rajouter la librairie math

```
cc calcul.c -o calcul $(options: fftw3)
```

doit produire

```
cc calcul.c -o calcul -lm -lfftw3
```

Une première idée serait d'éditer le fichier `/usr/lib/options/fftw3` et d'y rajouter l'option `-lm`, mais ce n'est pas propre car si jamais un jour les options de maths changent, alors il faudrait éditer tous les fichiers d'options qui en contiennent.

On va donc devoir changer le format des fichiers qui sont dans `/usr/lib/options` : au lieu d'une unique ligne contenant les options, voici ce qu'on est obligé de faire pour `/usr/lib/options/fftw3` :

```
nom: fftw3
inclure:   <fftw3.h>
options: -lfftw3
depend: math
descr: librairie pour faire des transformées de Fourier
```

Et voici par exemple le fichier `/usr/lib/options/maths` :

```
nom: math
inclure:   <math.h>
options: -lm
depend:
descr: librairie mathématique de base
```

Et voici par exemple le fichier `/usr/lib/options/optim` :

```
nom: optim
inclure:
options: -O3
depend:
descr: optimisation pour la vitesse du programme
```

Il y a donc plusieurs lignes, attribut: valeur. Les options sont dans la ligne `options:` et les dépendances dans la ligne `depend:` On a aussi une courte description des options, ce qui est

agréable pour savoir si on emploie les bonnes. Et on a aussi les `#include` à faire pour que ça compile bien.

☞ Proposer ce qu'il faut pour rendre le script `options:` capable de gérer les dépendances.

e) Affichage d'informations

☞ Écrire un second script appelé `InfosOption` auquel on passe le nom d'option comme pour `options:` et qui affiche le nom, les inclus et la description et aussi les informations des dépendances.

NB : cette commande `options:` existe vraiment, mais elle s'appelle `pkg-config`.