Introduction aux systèmes d'exploitation (IS1)

TP 8 – Des boucles

Tout comme en Java, tous les shells Unix, et en particulier `bash` que vous utilisez, sont capables de réaliser plusieurs types de boucles ou itérations.

Boucles `for`. La première syntaxe possible, et la plus simple, a la forme suivante :

```
for var in liste ; do commandes ; done
```

Ici, *var* est le nom d'une variable, *liste* une liste de chaînes de caractères séparées par des espaces et *commandes* une liste de commandes. Lors de l'exécution, *var* prend successivement chaque valeur apparaissant dans *liste*, et *commandes* est executée pour chacune de ces valeurs. Voici un exemple de boucle :

```
for prenom in Yvonne J{ean{ne,},acques} ; do echo Bonjour $prenom ; done
```

La sortie produite par cette boucle sur le terminal est :

```
Bonjour Yvonne
Bonjour Jeanne
Bonjour Jean
Bonjour Jacques
```

Il est également possible d'utiliser une syntaxe plus proche d'autres langages, tels Java :

```
for (( expr1 ; expr2 ; expr3 )) do commandes ; done
```

Ici, *expr1*, *expr2* et *expr3* sont des expressions arithmétiques comme nous en avons déjà rencontrées. À l'initialisation de la boucle, *expr1* est évaluée. En fin de boucle, si *expr2* est évaluée à 0 alors *expr3* est évaluée et un nouveau tour de boucle commence. Sinon, la boucle se termine. Voici un exemple utilisant cette syntaxe :

```
for (( i=1 ; i <= 1024 ; i*=2 )) do echo $i ; done
```

La sortie produite par cette boucle est la suite des puissances de 2 jusqu'à 2^{10}.

Boucles `while` et `until`. Un autre type de boucles, qui existe également en Java, est la boucle `while` ou « tant que ». Elle suit la syntaxe suivante :

```
while condition ; do commandes ; done
```

Dans une boucle while, la liste de commandes *commandes* est exécutée de façon répétée tant que la commande *condition* donne un code de retour égal à 0. Il existe une autre syntaxe, `until` ou « jusqu'à ce que », qui fonctionne de manière quasiment identique :

```
until condition ; do commandes ; done
```

Exercice 1 – *Échauffement.*

1. Écrivez une boucle `while`, une boucle `until` et une boucle `for` qui affichent le message « Bonjour ! » indéfiniment. Est-ce possible avec les deux types de boucles `for` ?[1]
2. Écrire à l'aide d'une boucle `while` puis `until` une commande qui demande un nombre entier à l'utilisateur et affiche toutes les puissances de 3 inférieures à ce nombre.
3. Même question avec une boucle `for (( ... ))`.
4. Même question avec une boucle `for ... in ...`.

Exercice 2 – *Premières vraies boucles.*

Dans le TP6, exercice 6, question 3, on demandait de trouver parmi 100 couples de fichiers celui ou ceux qui correspondent à un calcul erroné. Plutôt que de tous les tester un par un, on souhaite écrire une commande qui réalise tous les tests automatiquement et affiche les numéros de **tous** les couples erronés (attention, il pourrait y en avoir plusieurs !).

1. Placez vous dans le répertoire `~monitIS1/tp6` et résolvez le problème en utilisant les caractères spéciaux du shell et une boucle de type `for ... in ...`
2. Comme la première solution est un peu trop facile, on souhaite maintenant utiliser une boucle de type `for (( ... ))` pour énumérer les numéros de couples.
   Indice : Pour afficher un nombre entier ***n*** en utilisant au moins deux chiffres (c'est à dire en rajoutant un zéro à gauche si nécessaire), on peut utiliser la commande
   `printf %.2d`*n*
3. En utilisant le même type de boucles, trouvez maintenant `deux` moyens possibles de vous passer de la commande `printf`.
4. Comme vous adorez faire des boucles, refaites les questions 2 et 3 en utilisant à la place de `for` une boucle `while`.
5. Pour prouver une fois pour toute que vous maîtrisez, refaites encore une fois les questions 2 et 3, cette fois ci avec une boucle `until`.
6. On souhaite à présent que notre commande s'interrompe dès qu'une mauvaise réponse est trouvée, et ne teste pas les couples de fichiers restants. Modifiez vos réponses aux questions précédentes afin de prendre en compte cette modification.
   Indice : On pourra dans certains cas utiliser la commande `break` pour forcer la sortie prématurée d'une boucle (ou `break` *n* pour sortir de *n* boucles imbriquées...), mais ce n'est pas toujours la meilleure solution.

Exercice 3 (Exercice complémentaire) – *Tri automatique.*

Un chercheur de la NASA maîtrisant mal Unix a malencontreusement rangé tous ses programmes dans un même dossier. Comme il possède de très nombreux fichiers, il souhaite y voir plus clair en les triant par date.

1. Téléchargez et extrayez l'archive `http://www.liafa.jussieu.fr/~ameyer/a-trier.tar.gz`.
2. Écrivez une boucle qui permette de trier chaque fichier du répertoire créé dans un sous-répertoire `annee/mois`, où `annee` et `mois` correspondent à la date de dernière modification du fichier concerné.

[1] **Rappel :** Pour terminer un programme qui boucle indéfiniment, utilisez la combinaison `Ctrl-C`.