TP6/7-IUP1
Structures de contrôles
fonctions, récursivité
(Unix externe)

January 12, 2009

1 Rappel et compléments sur les structures de contrôles

[a]

Les commandes importantes :

- `if test_command ; then instr ; else instr ; fi`
- `until test_command ; do instr ; done`
- `while test_command ; do instr ; done`
- `for NAME [in word] ; do instr ; done`

[a] Un fichier shell doit toujours commencer par la ligne `#/dev/sh`

La structure de contrôle *conditionnelle* a été vu au précédent TP. L'ensemble des structures de contrôle *itératives* sont définiess en shell comme dans les langages traditionnels, excepté la structure de `for ...` qui est différente. On peut remarquer par ailleurs, que l'opérateur utilisé pour composer des instructions est soit le `;` soit le retour chariot. Ainsi la ligne de commande, et le script suivant sont interprétés par le shell de la même manière :

```
>if ‘ls core > /dev/null 2> /dev/null‘; then echo
"fichier core" ; else echo "pas de fichier core"; fi
```

et

```
#!/bin/sh
if ‘ls core > /dev/null 2> /dev/null‘
then echo "fichier core"
else echo "pas de fichier core"
fi
```

La structure itérative `for` permet de réaliser des boucles sur des ensembles, comme par exemple :

```
for a in fic1 fic2 ; do ls $a ; rm $a; done
```

qui permet de lister les deux fichiers `fic1` et `fic2`, puis de les supprimer.

2 Évaluation d'expressions arithmétiques

La commande importante

- `expr`

La commande `expr` permet d'évaluer des expressions arithmétiques.

➥**Exercice 2.1** *Ecrivez un script shell qui prend comme argument le chemin relatif d'un répertoire, et qui permet de compter le nombre de fichiers et de répertoires contenus dans ce répertoire, en utilisant la commande* `expr` *et la structure itérative* `for`*. Si le répertoire n'existe pas la commande renvoie un message d'erreur.*

➥**Exercice 2.2** *Ecrivez un script shell qui prend comme argument le chemin relatif d'un fichier, et si le fichier existe affiche son contenu sur les terminaux des utilisateurs qui sont connectés sur la machine sur laquelle vous travaillez. Si le fichier n'existe pas la commande renvoie un message d'erreur.*

➥**Exercice 2.3** *Ecrivez un script shell qui prend en argument un entier, et renvoie sa factorielle.*

3 Les fonctions et la récursivité

La commande importante

- `function NAME() ;`
  `liste de commande;`

➥**Exercice 3.1** *Écrire une commande nommée* `rech` *qui prend un ou deux arguments. Le premier argument est le chemin relatif d'un répertoire et le second le nom d'un fichier. Si la commande est lancée avec un seul argument, c'est un nom fichier. La commande* `rech`*, recherche dans tout les répertoires issus du répertoire passé en argument le fichier passé en second argument, et affiche son chemin relatif. Si la commande est lancée sans argument, celle-ci effectue la recherche dans le répertoire courant.*

Vous écrirez deux versions de cette commande l'une avec des fonctions, et l'autre sans fonction.

➥**Exercice 3.2** *Ecrire une commande* `efface` *qui permet de supprimer les fichiers de taille nulle situés dans le répertoire dont le chemin relatif est passé en argument et dans tout ses sous-répertoires.*

➥**Exercice 3.3** *Ecrire une commande* `tree` *qui prend 1 ou 0 argument. Si la commande prend un argument, c'est le chemin relatif d'un répertoire, et elle affiche sous la forme d'un arbre le contenu de ce répertoire et de ses sous-répertoires. Si cette commande est lancée sans arguments, alors la commande*

affiche le contenu du répertoire courant et de ses sous-répertoires. L'exemple suivant illustre ce qui est demandé.

```
>tree
\---IUP
| \---tp3
| +---TP3.aux
| +---TP3.dvi
| +---TP3.log
| +---TP3.tex
| \---tp4
| +---TP4.aux
| +---TP4.dvi
| +---TP4.log
| +---TP4.tex
| +---TP4.tex~
| \---tp67
| +---#tp67.tex#
| +---core
| \---dir
| +---fic1
| +---fic2
| +---fic1
| +---fic2
| +---script2
```