

Unix et réseaux

4. Processus

4 novembre 2009

Dans toutes les questions qui suivent, il s'agit d'écrire un script. Pour respecter les conventions, faites en sorte que, dans chaque cas, si les arguments de la ligne de commande sont invalides (par exemple s'il manque un argument), le script affiche un message sur sa sortie d'erreur et termine avec un code de retour non nul.

Exercice 1 – *Encore de la programmation*

1. Écrire un script « **first** » qui prend un nombre n en premier argument et affiche les n arguments suivants de sa ligne de commande. Par exemple, « `./first 3 ceci est une liste de mots` » doit afficher

```
ceci
est
une
```

2. Faire en sorte que le script précédent accepte en plus une option « `-n` » permettant de numéroter les lignes. C'est-à-dire qu'ils se comportera pareil sur l'exemple précédent et que « `./first -n 4 ceci est une liste de mots` » affichera

```
1. ceci
2. est
3. une
4. liste
```

3. Écrire un script « **hello** » qui obéisse à la spécification suivante :
 - Sans argument, il affiche « **Hello user!** » en remplaçant « **user** » par le login de l'utilisateur courant.
 - Avec un argument « **A** », il affiche « **Hello A!** ».
 - Avec plusieurs arguments « **A B C D** », il affiche « **Hello A, B, C and D!** » en plaçant correctement les virgules et le mot “and”.

De plus, on demande que si le nom du script commence par « **bonjour** », le texte soit en français et pas en anglais. C'est-à-dire que le comportement doit être le suivant :

```
$ ./hello John
Hello John!
$ ./hello Paul George Ringo
Hello Paul, George and Ringo!
$ ln -s hello bonjour
$ ./bonjour Maurice Patapon
Bonjour Maurice et Patapon!
```

Exercice 2 – Priorités

1. Écrire un script « `ackerman` » qui prend deux arguments m et n et affiche la valeur de la fonction d’Ackerman sur le couple (m, n) . Si vous ne connaissez pas la définition de cette fonction, utilisez Google!
2. Lancer le script avec de grandes valeurs, par exemple « `ackerman 4 4` ». Lancer « `top` » et constater ce qui se passe.
3. Sans arrêter la commande précédente, lancer le script avec d’autres grandes valeurs, observer ce qui se passe avec « `top` ».
4. Donner au premier des deux une priorité plus grande avec « `renice` » et observer.
5. Tuer le deuxième Ackerman avec « `kill` ».

Exercice 3 – Sur un thème de Léonard de Pise

Vous connaissez sans aucun doute la suite de Fibonacci :

$$F_0 = 1 \qquad F_1 = 1 \qquad F_{n+2} = F_{n+1} + F_n$$

On va chercher à comparer différentes façons d’implémenter cette fonction en shell. On pourra utiliser la commande « `time` » pour comparer l’efficacité des différentes implémentations.

1. Écrire un script « `fib01` » qui calcule la fonction en se relançant lui-même pour faire l’appel récursif.
2. Écrire un script « `fib02` » qui calcule la fonction en utilisant une fonction shell, comparer. Dans les deux questions suivantes, on veut que le script, appelé avec la valeur n , affiche sur la sortie standard un nombre de lignes égal à F_n .
3. En supposant que la commande « `fib0 n` » produit en sortie F_n lignes de texte, donner une façon de faire afficher F_n (il est pertinent d’utiliser « `wc` »)
4. Écrire un script « `fib03a` » tel que « `./fib03a n` » affiche F_n lignes de texte. Lorsqu’il y a des appels récursifs, les faire l’un après l’autre.
5. Écrire un script « `fib03b` » qui a le même effet mais qui fait ses appels récursifs en parallèle.
6. Comparer l’exécution de ces deux scripts (« `time` » donne le temps d’exécution, « `top` » permet de surveiller le nombre de processus existants et leur état). Expliquer ce qui se passe pour de grandes valeurs de n (par exemple 30).