ISS - Initiation aux Systèmes d'exploitation et au Shell
LU2IN020

TP 03 – Les points d'entrée pour les données d'un script

Julien Sopena

septembre 2023

Le but de cette troisième semaine est d'approfondir l'étude des arborescences de processus commencée la semaine précédente. Il aussi l'occasion de faire le tour des différentes sources d'entrée pour les données d'un script : les paramètres et les variables d'environnement.

Exercice 1 : Un peu de C

Question 1

La semaine dernière, on a vu en TP comment utiliser les arguments passés à un script. Mais ce concept d'argument est général et vient de la façon dons l'OS lance un nouvel exécutable.

Pour commencer, implémentez un programme `helloYou` en *C* qui affiche un bonjour pour chaque personne passée en paramètre.

```
moi@pc /home/moi $ helloYou Pierre Sylvain Étienne Manon Maxime
Pierre
Sylvain
Étienne
Manon
Maxime
```

Question 2

Ajoutez à votre programme un message d'erreur s'il n'y a personne à saluer. Votre message devra contenir un exemple d'utilisation (à la manière d'un *usage* du *man*) qui contiendra le nom de votre exécutable.

Vous testerez votre solution en compilant votre code sous différent nom (option `-o` de *gcc*)

Question 3

Que se passe-t-il si certaines personnes comportent un nom et un prénom, comme dans l'exemple suivant ? Comment peut-on régler le problème ?

```
moi@pc /home/moi $ helloYou Pierre Sylvain Étienne Moustache Manon Lanza Maxime Biaggi
```

Exercice 2 : L'addition s'il vous plait

Question 1

Reprenez le code de l'exercice précédent pour implémenter un programme `somme` qui affiche l'addition de tous les paramètres passés en argument.

Question 2

Dans cet exercice nous allons utiliser une boucle infinie en *Bash*, celle-ci va utiliser un `while`. Dans beaucoup de langage ce dernier peut être vu comme un *"tant que vrai"*, mais *Bash* il doit être compris comme un *"tant que la commande suivante s'est bien passée"*.

Pour simplifier l'écriture des scripts, *Bash* fournit deux commandes `true` et `false`. Utilisez la commande `type` pour vérifier que ces deux booléens ne sont pas des mots clés, mais bien des commandes.

Quelles valeurs retournent-elles ?

Question 3

Nous allons maintenant implémenter en shell une boucle infinie qui lit sur le clavier des valeurs, puis les affiches comme dans l'exemple suivant). Pour ce faire, vous utiliserez la commande `read maVariable` qui lit une chaine de caractères sur le clavier et l'affecte dans la variable `maVariable` (une utilisation plus avancée de cette commande vous sera présentée la semaine prochaine).

Pour quitter la boucle `while`, vous testerez la valeur saisie et vous en sortirez avec un `break` si elle est égale à 'q'. Notez que si vous avez fait une erreur dans votre implémentation, vous pouvez toujours arrêter votre programme avec un `Ctrl+c` (dont nous étudierons le mécanisme plus tard).

```
moi@pc /home/moi $ ./addition.sh
Saisissez un nombre ou 'q' pour quitter : 12
Vous avez ajouté 12 à la somme
Saisissez un nombre ou 'q' pour quitter : 30
Vous avez ajouté 30 à la somme
Saisissez un nombre ou 'q' pour quitter : q
```

Question 4

On veut maintenant calculer la somme de tous les nombres saisis. Combien de processus seraient créés si l'on faisait un appel au programme `somme` à chaque tour de boucle ?

Question 5

Pour augmenter les performances de ce script, utilisez votre programme `somme` une seule fois en dehors de la boucle pour faire le calcul de l'addition.

Exercice 3 : L'environnement c'est durable

Question 1

En TD nous avons introduit le concept de variables d'environnement. Il s'agit bien d'un concept système et non de quelque chose de spécifique au *Bash*. Ainsi, la `libstd` nous offre la fonction `char *getenv(const char *name)` qui prend en paramètre le nom d'une variable d'environnement sous forme d'une chaine de caractères et retourne la chaine quelle contient.

Implémentez en *C* un programme *monEnvironnement* qui affiche le contenu des variables `HOME`, `PATH` et `PWD`, puis lancez-le depuis plusieurs répertoires en utilisant un chemin vers le binaire correspondant à votre programme. Que remarquez-vous ?

Question 2

Modifiez votre programme pour afficher le contenu des variables d'environnement `TEST_1`, `TEST_2` et `TEST_3`. Puis lancez les commandes suivantes :

```
moi@pc /home/moi $ monEnvironnement
moi@pc /home/moi $ export TEST_1="J'existe"
moi@pc /home/moi $ monEnvironnement
moi@pc /home/moi $ monEnvironnement
moi@pc /home/moi $ TEST_2="J'existe" monEnvironnement
moi@pc /home/moi $ monEnvironnement
```

Que remarquez-vous ?

Question 3

Si la commande `echo` permet d'afficher le contenu d'une variable d'environnement, il fait connaitre a priori son nom.

Pour lister l'ensemble des variables d'environnement, on peut utiliser la commande `env`. Lancez-la pour vérifier la présence des variables `TEST_1` et `TEST_2`. Cela devrait confirmer votre précédente observation.

Question 4

Vérifiez que cette commande `env` n'est pas un *builtin*. Pourquoi est-ce possible ?

Question 5

Les variables d'environnement pourraient finir par s'accumuler. Il faut donc un mécanisme permettant de les supprimer. Le *Bash* nous offre donc la commande `unset`.

Supprimer la variable `TEST_1` puis vérifier qu'elle a bien disparu à l'aide de la commande `env`.

Question 6

Quel est le type de cette commande (*builtin* ou commande extérieure) ? Expliquez ce choix.

