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

TD 01 – Introduction au Bash

Julien Sopena

septembre 2022

Le but de cette première semaine est de comprendre le fonctionnement de l'interprète de commandes bash et d'apprendre à réaliser un petit script.

Exercice 1 : Langage interprété VS langage compilé

Question 1

Quelles différences y a-t-il entre un compilateur et un interprète ?

Question 2

Beaucoup de langages peuvent être à la fois interprétés et compilés, mais on appelle *langage compilé* un langage qui est généralement exécuté après une phase de compilation. Citez des *langages compilés*.

Question 3

Citez des *langages interprétés*.

Question 4

Qu'est-ce qu'un langage de script ?

Exercice 2 : Terminal VS Shell

Par abus de langage, on confond trop souvent ces deux termes. Mais même s'ils sont complémentaires, ils correspondent à deux couches logicielles complètement différentes.

Question 1

Quel est le rôle d'un terminal ?

Question 2

Pourquoi parle-t-on d'émulateur de terminal ou de terminal virtuel ?

Question 3

Citez des exemples de terminaux.

Question 4

Quel est le rôle d'un Shell Unix ?

Question 5

Citez des exemples de Shell Unix.

Exercice 3 : L'automate de traitement de *Bash*

Dans la suite de ce TD, comme pour le reste de l'année, nous considérerons le Shell Unix Bash.

Question 1

Après chaque lecture sur son entrée standard (le clavier), Bash va enchaîner une série de traitements avant d'exécuter la commande demandée. Listez dans l'ordre ces différentes étapes.

Question 2

Lorsque le *Bash* traite à nouveau une ligne, par exemple dans une boucle, doit-il refaire toutes ces étapes à chaque fois qu'il exécute la ligne ?

Exercice 4 : Recherche de la prochaine commande

La première action réalisée par Bash lorsqu'il reçoit une chaîne de caractères est d'extraire la première commande à exécuter.

Dans cet exercice, on utilisera la commande `echo` qui affiche sur la sortie standard (par défaut la console) l'ensemble des arguments.

Question 1

Quel est le résultat des commandes suivantes :

1.
```
moi@pc /home/moi $ echo Hello world
```

2.
```
moi@pc /home/moi $ echo Hello world
moi@pc /home/moi $ echo Hello world
```

3.
```
moi@pc /home/moi $ echo Hello world echo Hello world
```

4.
```
moi@pc /home/moi $ echo Hello world ; Hello world
```

5.
```
moi@pc /home/moi $ echo Hello world ; echo Hello world
```

6.
```
moi@pc /home/moi $ echo Hello world;echo Hello world
```

7.
```
moi@pc /home/moi $ echo Hello world ;
moi@pc /home/moi $ echo Hello world ;
```

Question 2

Peut-on écrire tout un programme shell en une seule ligne ? Quel en serait l'avantage ?

Question 3

Y a-t-il d'autres moyens de séparer les commandes ?

Exercice 5 : Substitution des écritures factorisées

Après avoir trouvé la commande à exécuter, le *Bash* va appliquer une série de substitution en commençant par l'écriture factorisée à l'aide d'accolades.

Question 1

Quels sont les résultats des commandes suivantes :

1.
```
moi@pc /home/moi $ echo fichier_{a,b,c,d}
```

2.
```
moi@pc /home/moi $ echo fichier_{abcd}
```

3.
```
moi@pc /home/moi $ echo fichier_{un,deux,trois}
```

4.
```
moi@pc /home/moi $ echo fichier_{un deux trois}
```

5.
```
moi@pc /home/moi $ echo fichier_{a,b} repertoire_{c,d}
```

6.
```
moi@pc /home/moi $ echo fichier_{a,b} {c,d}
```

7.
```
moi@pc /home/moi $ echo fichier_{a,b}{c,d}
```

8.
```
moi@pc /home/moi $ echo fichier_{a,b,c{1,2}}
```

9.
```
moi@pc /home/moi $ echo fichier_{a,b,{c,d}}
```

Question 2

Proposez trois commandes factorisées qui affichent sur le terminal les listes de paquets suivantes :

1. `libreoffice-en-US libreoffice-fr`
2. `python python-scipy python-matplotlib`
3. `vim-runtime vim-spell-fr vim-spell-en`

Question 3

Proposez une commande qui efface dans le répertoire courant les 6 fichiers suivants sans risquer d'en effacer d'autres :
`list.c list.h list.o tab.c tab.h tab.o`

Dans cet exercice, on utilisera la commande `rm` qui supprime les fichiers dont les noms sont passés en arguments à la commande.

Question 4

Les accolades permettent aussi de générer facilement des séquences de nombre. La syntaxe est la suivante :

`{debut..fin}` : énumère tous les nombres compris entre `debut` et `fin` ;

`{debut..fin..inc}` : énumère tous les nombres compris entre `debut` et `fin` avec un pas `inc`.

En utilisant les séquences, donnez des équivalents des trois commandes suivantes :

1.
```
moi@pc /home/moi $ echo fichier_{0,1,2,3,4,5,6,7,8,9}
```

2.
```
moi@pc /home/moi $ echo fichier_{0,2,4,6,8}
```

3.
```
moi@pc /home/moi $ echo fichier_{0,1,2,3,6,8,10,12}
```

Exercice 6 : La substitution des variables

En *Bash*, on accède à la valeur des variables en utilisant le caractère `$` accolé au nom de la variable. L'interprète va alors les substituer par leur valeur en partant de la gauche vers la droite. Pour définir le contenu d'une variable, on écrit simplement au début d'une commande son nom suivi d'un = sans le signe `$`.

Question 1

Quel affichage est produit par les commandes suivantes ?

```
moi@pc /home/moi $ ue=ISS
moi@pc /home/moi $ annee=2019
moi@pc /home/moi $ echo $ue td1 - $annee
moi@pc /home/moi $ note_partiel=18
moi@pc /home/moi $ echo partiel:$note_partiel et examen:$note_examen
```

Question 2

Quels sont les types des variables `ue` et `annee` ?

Question 3

À quel mécanisme du C s'apparente le traitement des variables en *Bash*.

Question 4

Trouvez et tentez de corriger les erreurs dans les commandes suivantes :

```
moi@pc /home/moi $ annee = 2010
moi@pc /home/moi $ nom=Aloe Blacc
moi@pc /home/moi $ echo $Nom_$Annee : I need a $
```

Question 5

La syntaxe d'accès aux variables avec des accolades est très puissante, car elle permet de faire un prétraitement sur la valeur de variable avant sa substitution. Nous en verrons plusieurs exemples à la semaine 5.

Pour aujourd'hui, nous allons utiliser `${#nom_var}` qui retourne la longueur de la chaîne de caractères contenue dans la variable `$nom_var`. Écrire la commande qui affiche la chaîne contenue dans une variable `x` sous la forme du message suivant :

```
La variable $x contient la chaine "Hello" qui a 5 caractères.
```

Exercice 7 : Découpage des paramètres

Une autre tâche de l'interprète Bash est de découper en mot la commande une fois toutes les substitutions effectuées. Le premier mot constituera la commande et les autres ses paramètres.

On parle de mot, car le séparateur considéré par défaut est le caractère espace. Ce comportement peut être redéfini en modifiant la variable `$IFS`, mais nous n'aborderons pas ce point au cours de ce semestre.

Question 1

Pour expérimenter ce découpage, nous allons utiliser la commande `cat` qui affiche sur sa sortie standard le contenu de tous les fichiers dont les noms sont passés en paramètre.

En supposant l'existance des deux fichiers `cv.txt` et `ma lettre.txt`, donnez l'affichage produit par les commandes suivantes :

```
moi@pc /home/moi $ cat cv.txt
moi@pc /home/moi $ cat ma lettre.txt
```

Question 2

Proposez une solution (la plus simple possible) qui permette d'aboutir à l'affichage des deux fichiers.

Question 3

Le découpage en mot est particulièrement important pour la boucle for. En effet, la boucle for du *Bash* ressemble davantage au `for` du `Python` ou au `foreach` du `Java`, qu'au `for` du `C` : une variable va successivement contenir chaque mot présent dans une liste.

Après avoir expliqué l'utilisation du `;`, donnez l'affichage produit par les commandes suivantes :

```
moi@pc /home/moi $ val="un␣deux"
moi@pc /home/moi $ for x in zero $val trois "et␣cetera"; do
   > echo $x
   > done
```

Question 4

Peut-on écrire cette boucle sur une seule ligne dans le terminal. Si oui, proposez une solution ; sinon pourquoi ?

Exercice 8 : Substitution des métacaractères

La dernière étape avant le lancement de la commande est la substitution des métacaractères par l'ensemble des fichiers dont le nom correspond au motif demandé.

Bash offre 3 métacaractères pour décrire ces motifs :

- `*` : représente n'importe quelle suite de caractères (vide ou non). Seule exception, la chaîne ne doit pas commencer par un point si l'étoile est en début de motif (nous reviendrons sur ce point dans 2 semaines).
- `?` : représente n'importe quel caractère (un et un seul)
- `[liste de caractères]` : représente exactement un caractère de la liste. Cette liste peut être une succession de caractères (sans séparateur) ou d'intervalle ([x-z] ou [4-9] par exemple).

Ces métacaractères peuvent être composés à l'envie, seuls ou accompagnés de caractères classiques.

En cas d'échec, i.e. absence d'au moins un fichier correspondant au motif, celui ci est laissé tel que dans la ligne de commande.

Question 1

Dans cet exercice, nous allons utiliser la commande `ls` qui liste et affiche des informations :
— de l'ensemble des fichiers du répertoire courant en absence de paramètre ;
— des fichiers contenus par les répertoires passés en paramètre ;
— des fichiers dont le nom est passé en paramètre.
Donnez l'affichage produit par les commandes suivantes :

1.
```
moi@pc /home/moi $ ls *.pdf
```
2.
```
moi@pc /home/moi $ ls ?
```
3.
```
moi@pc /home/moi $ ls td[1-3]
```
4.
```
moi@pc /home/moi $ ls td[13]
```
5.
```
moi@pc /home/moi $ ls td[1-13]
```
6.
```
moi@pc /home/moi $ ls [a-z]*
```

Question 2

Pourquoi est-il horrible d'écrire la commande `ls *` pour lister des fichiers ?

Question 3

Pour étudier l'ordre de la substitution, on propose de faire l'expérience suivante :

```
moi@pc /home/moi $ a=*
moi@pc /home/moi $ echo x$a
```

Qu'est ce que cette expérience permet de vérifier.

Question 4

Proposez une autre expérience qui mette en évidence le fait que la substitution des métacaractères ait lieu après le découpage en mot des paramètres.

Question 5

Est-ce que l'utilisation de métacaractères peut conduire à avoir trop de paramètres si les fichiers sont trop nombreux.