INFO-F-201 – Systèmes d'exploitations 1

TP6: Introduction à Linux: Shell scripting

avec l'accord de Sébastien Collette

Résumé

Suite à la séance précédente sur l'utilisation du shell `bash`, nous allons voir comment automatiser des séquences de commandes sous la forme de scripts et approfondir certains aspects du shell.

1 Globbing

Le *globbing* ou *filename expansion*, est une méthode pour désigner un ensemble de fichiers. C'est un concept proche, mais différent, des expressions régulières, sur lesquelles nous reviendrons plus tard. En voici la syntaxe :

`*`	n'importe quelle chaine de caractères, même vide (sauf `.` en début de mot)
`?`	n'importe quel caractère (sauf `.` en début de mot)
`[liste]`	n'importe quel caractère de la liste
`[^liste]`	n'importe quel caractère hors de la liste
`[a-z]`	n'importe quel caractère dans l'interval

La commande `ls` permet de lister les fichiers contenus dans un répertoire [1]. Vous obtiendrez tous les fichiers dont le nom commence par a en tappant `ls a*`.

Exercice 1 *Comment obtenir tous les fichiers dont le nom*
- *commence par* `var` *et est suivi de deux lettres*
- *se termine par tar, par gz ou par tgz*
- *contient au moins un chiffre*
- *indique un fichier caché (fichier dont le nom commence par* `.`*)*

en utilisant uniquement le globbing.

---

2 Caractères spéciaux

Exercice 2 *Testez les commandes suivantes :*

1. N'oubliez pas d'utiliser `man` pour obtenir plus de détails.

```
echo un essai
echo  un   essai
echo "un   essai"
echo "
echo \"
echo \
echo \\
echo \'
echo \\
echo **
echo \*\*
echo ‘date‘
echo "date: ‘date‘"
echo 'date: ‘date‘'
```

et déterminez quel est leur effet.

Exercice 3 *Utilisez la commande* `echo` *et la commande* `date` *pour obtenir* `Bonjour! La date au format jj/mm/aa est <date>. Obtenue par la commande "<commande>".`

---

3 Encore quelques commandes utiles

Exercice 4 *A quoi servent* `du` *et* `df` *? Donnez la taille de tous les répertoires présents dans / sous format "human-readable". Que fait la commande* `whereis` *?*

---

4 Introduction au shell scripting

Les scripts shells sont reconnaissables à l'extension `.sh` qui leur est généralement attribuée. Ils commencent par une ligne déterminant quel shell devra être utilisé. Dans notre cas ce sera :

```
#!/bin/bash
```

Ensuite, ils sont composés d'un ensemble de commandes. Autrement dit, tout ce que nous avons fait jusqu'ici en ligne de commande peut être placé dans un script. Il s'agit d'un vrai langage de programmation, avec des variables, des boucles, des conditions, *etc.*

Note *Une fois votre script écrit dans un fichier* `monscript.sh`*, il vous faudra taper* `chmod a+x monscript.sh` *pour le rendre exécutable.*

5 Variables

Le shell permet la création, la consultation et la modification de variables. Il existe deux types de variables, les locales et les globales (appelées aussi variables d'environnement). Les locales ne sont visibles qu'à l'intérieur du shell ou du script qui les a déclarées, tandis que les globales sont également visibles dans tous les processus fils de ce shell [2]. Les variables ne doivent pas être déclarées, elles le sont automatiquement à leur première assignation. Voici les opérations possibles :

- `mavariable=valeur` : assigne la valeur à mavariable, après l'avoir déclarée si nécessaire.
- `$mavariable` : représente la valeur de la variable mavariable.
- `echo $mavariable` : affiche le contenu de la variable.
- `unset mavariable` : efface la variable (la valeur est perdue).
- `export mavariable` : transforme une variable locale en variable globale (la valeur ne change pas).
- `set` : affiche toutes les variables actuellement définies et leur valeur.

Exercice 5 *Listez l'ensemble des variables. Que contiennent et à quoi servent* `PATH`*,* `HOME`*,* `PS1`*,* `GROUP`*,* `USER`*,* `PWD`*,* `PPID`*,* `TERM`*? N'oubliez pas que* `man bash` *est là pour vous aider.*

Exercice 6 *Quel est l'effet de chacune de ces commandes (dans l'ordre) ?*

```
VAR=essai
echo $VAR
echo $VARs ratés
echo ${VAR}s ratés
unset VAR
echo $VAR
VAR=essai
bash
echo $VAR
exit
export VAR
bash
echo $VAR
VAR=essai2
echo $VAR
exit
echo $VAR
```

2. Ce qui peut s'avérer utile pour passer des informations à des scripts invoqués par d'autres.

Soyez précis dans vos réponses.

Exercice 7 *Modifiez le prompt*[3] *pour qu'il affiche* `login: dossier: date>`.

---

6 Paramètres

Les variables suivantes sont automatiquement assignées par le shell et ont une signification particulière :

`$0`	Le nom du script en cours d'exécution
`$1, $2, ...`	Le 1er, 2ème, etc... paramètre du script
`$#`	Le nombre de paramètres du script
`$*`	Les paramètres, séparés par le premier caractère de la variable `IFS`
`$@`	Les paramètres, chacun entouré de double quotes, et séparés par des blancs

Les variables `$*,$1,$2, ...`, quand elles sont utilisées dans une fonction, se rapportent aux paramètres de cette fonction, mais jamais la variable `$0`, qui fait toujours référence au nom du script global.

Note : certaines variables nécessitent la notation plus générale `${nom}`, par exemple `$10` doit être plutôt écrit `${10}`.

7 Pattern matching

Les opérateurs suivants permettent de faire du pattern matching[4] dans une chaine contenue dans une variable. Les patterns peuvent contenir des wildcards[5].

- `${variable#pattern}`
  Si le pattern matche le début de la valeur de la variable, effacer la plus courte partie de la variable qui matche, et renvoyer le reste.
- `${variable%pattern}`
  Si le pattern matche la fin de la valeur de la variable, effacer la plus courte partie de la variable qui matche, et renvoyer le reste.
- `${variable/pattern/string}`
  Le plus long match du pattern dans la variable est remplacé par `string`. Seul le premier match est remplacé. Disponible à partir de bash 2.0.
- `${variable//pattern/string}`
  Idem, sauf que toutes les occurences sont remplacées.

3. Le prompt est l'indicateur de début de ligne affiché par `bash` dans l'attente de l'entrée d'une commande.
4. Le pattern matching consiste à trouver une sous-chaine dans une autre.
5. La syntaxe a été vue lorsque nous avons parlé du *globbing*.

8 Contrôle de séquence

8.1 if/else

```
if condition
then
     statements
[elif condition
     then statements...]
[else
     statements]
fi
```

Le `then` doit être sur une nouvelle ligne, sauf si la condition est suivie d'un point-virgule (caractère séparateur du shell). La condition testée peut correspondre à différentes choses :

- Test du "exit status" : si le if est suivi d'une série de commandes, le test est vrai si la dernière commande renvoie 0 et faux sinon. Les commandes peuvent également être liées par un `||` ou par un `&&`.
- Tests de strings :

Opérateur	Vrai si...
`[ str1=str2 ]`	
`[ str1!=str2 ]`	
`[ str1<str2 ]`	
`[ str1>str2 ]`	
-n str1	str1 pas vide
-z str1	t str1 vide

  Attention : le crochet ouvrant (suivi d'un espace) et le crochet fermant (précédé et suivi d'un espace) sont obligatoires.
- Les tests d'attributs de fichiers [6]

Opérateur	Vrai si...
`[ -d fichier ]`	*fichier* existe et est un répertoire
`[ -e fichier ]`	*fichier* existe
`[ -f fichier ]`	*fichier* existe et est régulier
`[ -r fichier ]`	permission en lecture
`[ -s fichier ]`	*fichier* existe et est non-vide
`[ -w fichier ]`	permission en écriture
`[ -x fichier ]`	permission en exécution
`[ -O fichier ]`	est-on propriétaire de *fichier* ?
`[ -G fichier ]`	est-on du groupe de *fichier* ?
`fichier1 -nt fichier2`	*fichier1* est-il plus récent que *fichier2* ?
`fichier1 -ot fichier2`	*fichier1* est-il moins récent que *fichier2* ?

6. Les informations se rapportant aux droits et utilisateurs sont ici à titre indicatif. Une session ultérieure présentera ces aspects.

– Les expressions conditionnelles entre crochets peuvent être combinées par `||` et par `&&`. Une expression conditionnelle peut également être combinée avec une commande (test du “exit status”). Les expressions entre crochets peuvent également contenir un `!` (négation). Finalement, des expressions logiques peuvent être groupées par des parenthèses précédées d'un backslash, et connectées par des `-a` (AND) et des `-o` (OR).
– Conditions entières :

Test	Comparaison
`-lt`	less than
`-le`	lower or equal
`-eq`	equal
`-ge`	greater or equal
`-gt`	greater than
`-ne`	not equal

8.2 for

Le `for` du shell itère sur une variable dont les valeurs sont prises, dans l'ordre, dans une liste de valeurs spécifiées :

```
for name [in list]
do
    statements that can use $name...
done
```

Si `in list` est omis, la liste est fixée par défaut à `"$@"`. Une liste est un string d'éléments séparés par le premier caractère contenu dans la variable `IFS` (l'epace par défaut).

8.3 while et until

```
while condition
do
   statements...
done
```

```
until command; do
   statements...
done
```

8.4 Fonctions

Il existe deux syntaxes pour la déclaration de fonctions en bash :

```
function name                    name()
{                                {
  shell commands                    shell commands
}                                }
```

Ces fonctions peuvent être récursives. La valeur de retour d'une fonction est la valeur de retour de la dernière commande, sauf si un `return` est spécifiquement mentionné. Note : une valeur de retour de 0 est considérée comme vraie dans les tests, et une valeur de retour différente comme fausse.

9 Input

Il est également possible d'interagir avec l'utilisateur au travers de `read` pour lui demander certaines informations.

```
read nom_variable
```

Exercice 8 *Ecrire un script affichant "Bonjour " accompagné d'un nom demandé à utilisateur.*

---

10 Exercices

Exercice 9 *Ecrire un script* lsD.sh *qui va lister uniquement les dossiers contenus dans un dossier passé en paramètre du script*

Exercice 10 *Ecrire une script qui liste récursivement le contenu d'un répertoire (à la* `ls -R`*) d'une manière arborescente. Donnez également le nombre de fichiers contenus et la taille du répertoire.*

Exemple :

```
home 2 file(s) for a total of 5MB
    user1 2 file(s) for a total of 3MB
          tp-admin.tex
          tp-ag1.html
    user2 1 file(s) for a total of 2MB
          tp-archi.cpp
```