Les scripts SHELL

1. Enchainer plusieurs commandes en écrivant un script

C'est quoi un script

Les shells sont interactifs, c'est à dire qu'ils acceptent des commandes et les exécutent. Mais s'il vous voulez utilisez une séquence de commande alors vous pouvez les écrire dans un fichier et dire au shell d'exécuter le fichier plutôt que d'entrer les commandes.

Les scripts shell sont une série de commande dans un fichier texte.

Les scripts peuvent prendre des paramètres entrés par l'utilisateur, par des fichiers et afficher la sortie à l'écran.

Les scripts sont utiles pour :

- Créer vos propres commandes,
- Gagner du temps,
- Automatiser certaines tâches de l'utilisateur et surtout des administrateurs système.

Les outils pour écrire mes scripts

Les scripts shell sont des fichiers textes tout ce qu'il y a de plus classique.

Pour créer un fichier et y insérer du texte, on a deux possibilités :

1. La méthode « geek » tout en ligne de commande :

```
~$ echo « Contenu de mon fichier texte » > fichier.txt
```

Ici vous créez un fichier nommé « fichier.txt ». Si vous voulez ajouter du texte, il faudra concaténer le message précédent :

```
~$ echo « Suite de mon fichier texte » >> fichier.txt
```

Pour lire le contenu complet de votre fichier, vous pouvez utiliser les commandes :

- more,
- cat,
- less.

Pour modifier le fichier, on pourra utiliser les commandes suivantes :

- sed,
- awk,
- tr :

```
cat fichier.txt | tr « fichier » « petit fichier » > fichier.txt
```

Cette méthode laisse peu de place à la modification de fichier, c'est pourquoi on préférera utiliser la méthode suivante :

2. La méthode « classique » : l'utilisation d'un éditeur de texte.

   Vous pouvez utiliser différents éditeurs :

- gedit, kedit, nedit
- geany
- nano
- vim
- gvim
- emacs
- xemacs
- eclipse.

2. Ecrire un premier script shell

Comment exécuter un fichier script

Pour exécuter une commande shell, il existe deux alternatives.

La première méthode est la moins pratique : il faut lancer un shell en lui donnant en paramètre le nom d'un fichier contenant les commandes à exécuter.

```
~$ bash monScript.sh
```

La seconde méthode consiste à indiquer directement dans le fichier quelle est la commande (le shell) qui permettra de l'interpréter. Pour mettre en place cette solution, il faut indiquer sur la première ligne du fichier de script quel est l'interpréteur qui permettra d'exécuter les commandes des lignes suivantes. Cette ligne possède une syntaxe particulière : les caractères « # ! » (Prononcer shebang) suivi du chemin absolu (depuis la racine) vers l'interpréteur.

Dans le cas du bash, nous aurons comme première ligne du fichier :

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

Toutefois, pour que ce mécanisme fonctionne, vous devez rendre ce script exécutable (attention à la gestion des droits utilisateur (u), groupe (g) et autres (o). Il faut exécuter la commande « chmod » pour modifier les droits d'un fichier en lecture (r), écriture (w) et exécution (x).

```
chmod u+x o-rw monScript.sh
```

On ajoute les droits d'exécution pour l'utilisateur et on retire les droits de lecture et écriture pour les autres.

Il ne reste plus qu'à taper le nom de votre dans un terminal pour que celui-ci soit exécuté.

Premier script « Hello Linux »

Nous allons pouvoir maintenant réaliser notre 1er script shell.

Pour que l'exemple reste le plsu simple possible, nous allons seulement utiliser la commande d'affichage « echo ».

Le caractère # suivi de n'importe qu'elle texte est considéré comme un commentaire (excepté la 1ère ligne).

3. Rendre un script paramétrable en utilisant une variable

Qu'est-ce qu'une variable

Une variable n'est jamais qu'un espace réservé en mémoire dans lequel on peut écrire des données. Pour accéder plus simplement à cette espace, on lui donne un nom (utiliser une variable du type 0x278faa0 est nettement moins pratique).

Créer une variable

Pour définir un variable « utilisateur », on utilisera la syntaxe suivante :

```
variable_name=value
```

Pour utiliser la valeur contenue dans la variable, il faudra la faire précéder du caractère « $ ».

```
#!/bin/bash
msg=« Hello World ! »
echo $msg
```

Quelques règles :

1. Le nom d'une variable doit commencer par un caractère alphanumérique ou underscore (_) et être suivi par un ou plusieurs caractères alphanumériques.
2. Ne pas mettre d'espace entre le caractère égal
3. Les variables sont « case sensitive »
4. Vous pouvez définir des variables NULL :

```
$ variable=
$ variable=«»
```

5. Ne pas utiliser ?, *, etc. pour nommer vos variables.

Il est possible d'exécuter des opérations sur les variables. Si les variables sont des entiers, on pourra utiliser les opérateurs arithmétiques classiques, plus « % » pour le reste de la division entière et « ** » pour la puissance.

Soit on encadre l'opération par $((…))

Soit on utilise la commande let « … »

Dans le cas de chaînes de caractères, pour réaliser une concaténation, il suffit de mettre les variables côte à côte.

COMMANDE	VALEUR DE LA VARIABLE VAR
var=7+8	7+8
let « var=7+8 »	15
var=$((7+8))	15
a=«Hello» var=$a« World !»	Hello World !

```
a=«Hello»                Hello World !
b= « World ! »
var=$a$b
```

La commande echo :

echo [options] [string, variables...]
Displays text or variables value on screen.
Options
-n Do not output the trailing new line.
-e Enable interpretation of the following backslash escaped characters in the strings:
\a alert (bell)
\b backspace
\c suppress trailing new line
\n new line
\r carriage return
\t horizontal tab
\\ backslash

Les différents types de variables

Les variables d'environnement

Sans revenir sur le cours précédent, les variables d'environnement sont notées en majuscules.

Les variables spéciales

$0	Le nom du script.
$1, $2	Les arguments passés au script ($1 est le 1er, ...).
$*	La liste de tous les arguments passés au script, séparés par un espace.
$#	Le nombre d'arguments passé au script.
$?	Le code de retour de la dernière commande exécutée.
$!	Le numéro de processus de la dernière commande lancée en tâche de fond
$$	Le numéro de processus du script lui-même.

La commande read

Syntaxe :

```
Read variable1, variable2, … variableN
```

La commande read peut être utilisée pour réaliser une pause dans un programme en demandant à l'utilisateur d'appuyer sur une touche pour continuer :

```
echo « Appuyer sur une touche pour continuer »
read
```

4. Exécuter des commandes suivant des conditions précises

Les structures conditionnelles

Pour conditionner l'exécution de lignes de code, on utilise l'instruction if.

```
if condition ; then
…
fi
```

L'instruction then étant une nouvelle instruction, il faut la séparer du début de la ligne par un point virgule. Sans ce caractère, il aurait fallu utiliser une notation moins compacte :

```
if condition
then
…
fi
```

Cette structure permet d'exécuter des instructions au cas où la condition est vérifiée. Dans le cas où elle est invalidée, si vous souhaitez exécuter un autre traitement, il faudra exécuter l'instruction else :

```
if condition ; then
…
else
…
fi
```

Enfin, on peut imbriquer les tests dans des structures de type :

```
if condition_1 ; then
…
elif condition_2 ; then
…
elif condition_3 ; then
…
fi
```

Il reste à voir comment s'écrit la condition.

Une condition, encore appelée test, peut s'écrire de deux manière différentes :

- soit en l'encadrant par des crochets : […],
- soit en utilisant la commande « test ».

Une fois la syntaxe choisie, en conjuguant cette écriture avec un opérateur de test, on obtient une condition.

Exemple : L'opérateur plus petit que s'écrit « –lt ». Ce qui donne :

```
[2 -lt 3]
test 2 -lt 3
```

Ces 2 lignes sont identiques. Pour vous en assurer, vous pouvez exécuter :

```
~$ [2 -lt 3]
~$ echo $?
```

Les opérateurs de test

Les opérateur de test se répartissent en 3 catégories : les opérateur arithmétiques dont nous avons vu un exemple avec « –lt » , les opérateurs de test de fichiers, et les opérateurs logiques permettant de créer des conditions complexes.

Les opérateurs arithmétiques

-lt	lower than (inférieur à)
-gt	greater than (supérieur à)
-eq	equal (égal à)
-ne	not equal (différent de)
-ge	greater or equal (supérieur ou égal)
-le	lower or equal (inférieur ou égal)
==	égal
!=	différent

Les opérateurs de test de fichiers

Les opérateurs de test de fichiers permettent d'obtenir très facilement des informations sur les fichiers. Il s'agit la plupart du temps d'opérateurs unaires portant sur le nom d'un fichier :

```
if [ -f $file ]
```

-e	(exist) Pour vérifier l'existence d'un fichier.
-f	(file) Pour vérifier l'existence d'un fichier et qu'il s'agisse bien d'un fichier et pas d'un répertoire.

-d	(directory) Pour vérifier l'existence d'un répertoire.
-r	(readable) Pour vérifier si un fichier est accessible en lecture.
-w	(writable) en écriture.
-x	(executable) en exécution.

Les opérateurs logiques

Les opérateurs logiques permettent de composer des conditions plus « complexes » à l'aide des opérateurs booléens « et », « ou » et « non ».

L'opérateur « non » est un opérateur unaire : il n'accepte qu'un seul argument dont il va inverser la valeur.

Le symbole de l'opérateur est !

Exemple :

[! 10 –lt 5] testera si 10 est supérieur ou égal à 5.

Les opérateur « et » et «ou » sont des opérateur binaire.

&& ou –a pour l'opérateur « et »

|| ou –o pour l'opérateur « ou »

Exemple :

Pour tester si la variable var est comprise entre 1 et 10, on utilisera le test :

```
[$var –ge 1] && [$var –le 10]
```

Enfin une commande permet d'obtenir la liste complète des opérateurs arithmétiques, de test de fichiers et logique pour la création de conditions :

```
~$ info coreutils 'test invocation'
```

La structure case

Il n'y a pas que l'instruction if pour créer des structures conditionnelles : pour les structures à choix multiples, il existe la structure case permettant d'effectuer un branchement sur des lignes de code en fonction de la valeur contenue dans une fonction.

Exemple où l'on demande à l'utilisateur d'afficher un chiffre entre 1 et 4 :

```
#!/bin/bash

echo « Donnez un chiffre entre 1 et 4 : »
read choix

case $choix in
      1) echo « un »;;
      2) echo « deux »;;
      3) echo « trois »;;
      4) echo « quatre »;;
      *) echo « ERREUR »
esac
```

5. Exécuter plusieurs fois une commande sans avoir à la réécrire

Les structures de base

L'instruction while

Exemple pour l'écriture des 100 premiers résultats de la table de multiplication par 7 :

```
#!/bin/bash

i=1

while [ $i -le 100 ]; do
    echo $i 7= »$(($i*7))
    i=$(($i+1))
done
```

L'instruction until

La structure until se rapproche énormément de la boucle while.

Until signifie « jusqu'à ce que » alors que while signifie « tant que »

```
#!/bin/bash

i=1

until [ $i -gt 100 ]; do
    echo $i 7= »$(($i*7))
    i=$(($i+1))
done
```

Parcourir une liste

L'instruction for

Cette structure s'éloigne un peu des 2 types de boucles vus précédemment : ici, nous allons parcourir une liste d'éléments (chaîne de caractères, entier ou mélange de plusieurs types). Une liste est spécifiée par une suite d'éléments séparés par un espace.

Voici un exemple de liste

```
« chaine » 1 2 « autre chaine » 3 4 5
```

L'utilisation du caractère * permet d'obtenir un élément de type liste : les fichiers du répertoire courant.

La construction de la boucle se fait à l'aide d'une variable (la variable de boucle) qui va prendre consécutivement pour valeur les éléments de la liste.

La syntaxe est la suivante

```
for variable in liste ; do
```

Voici par exemple comment afficher et compter les fichiers du répertoire courant :

```
# !/bin/bash
i=1
for file in * ;do
    echo $i « .  » $file
    i=$(($i+1))
done
```

L'instruction select

L'instruction select est une sorte d'amélioration du for pour des cas particuliers. En effet, elle permet de proposer à l'utilisateur de saisir un choix dans une liste qui sera affichée à l'écran. Sans commande d'arrêt, la demande de saisie sera infinie.

Voici un exemple simple :

```
# !/bin/bash
select choix in « entree 1 » « entree 2 » ; do
    echo « Vous avez choisi » $choix
done
```

Lors de l'exécution, ce script va réaliser l'affichage suivant :

```
1) entree 1
2) entree 2
# ?
```

Vous pourrez saisir votre choix après le point d'interrogation et la variable choix prendra pour valeur votre saisie (à condition que votre choix fasse partie des choix possible).

Attention, vous devez taper le numéro correspondant à votre choix.

Comme dit précédemment, ce script boucle à l'infini et demandera toujours à l'utilisateur de saisir un choix. Pour arrêter la boucle, il faut utiliser l'instruction break. Le menu ressemblera alors à :

```
# !/bin/bash
select choix in « entree 1 » « entree 2 » « q »; do
    case $choix in
        « entree 1 » | « entree 2 ») echo « Vous avez choisi » $choix ;;
        « q ») echo « Au revoir… »
             break ;;
        *) echo « Saisie non valide !»
    esac
done
```

Si vous souhaitez interrompre la boucle dès qu'un choix a été sélectionné, vous pouvez également insérer un break.

Pour faire la même chose avec des instructions for et while, se serait plus complexe :

```
i=1
#Affichage des choix
for choix in « entree 1 » entree 2 » « q » ; do
    echo $i ») » $choix
    i=$(($i+1))
done
#Attente et Traitement de la saisie
while [ 1 ] ; do
    echo –n « # ? »
    read choix
    case $choix in
        1 | 2) echo « Vous avez choisi l'entree n.» $choix ;;
        3) echo « Au revoir… »
             break ;;
        *) echo « Saisie non valide !»
    esac
done
```

6. Debugger un script bash

Il existe 2 solutions pour pouvoir débugger un script bash :

Soit vous rajouter un –x à la fin de la 1ère ligne de votre script :

```
# !/bin/bash –x
```

Soit dans un terminal vous lancez:

```
~ $ bash –x mon_script.sh
```