BTS SIO SI 1	Les scripts BASH sous LINUX *Point du programme :* *Architecture et fonctions d'un système d'exploitation*	Fiche de synthèse N° 13

I- Qu'est-ce que le Bash ?

Bash est une version évoluée du **shell sh** (le "**Bourne shell**"). Le shell peut être utilisé comme un simple interpréteur de commande, mais il est aussi possible de l'utiliser comme langage de programmation interprété (scripts).

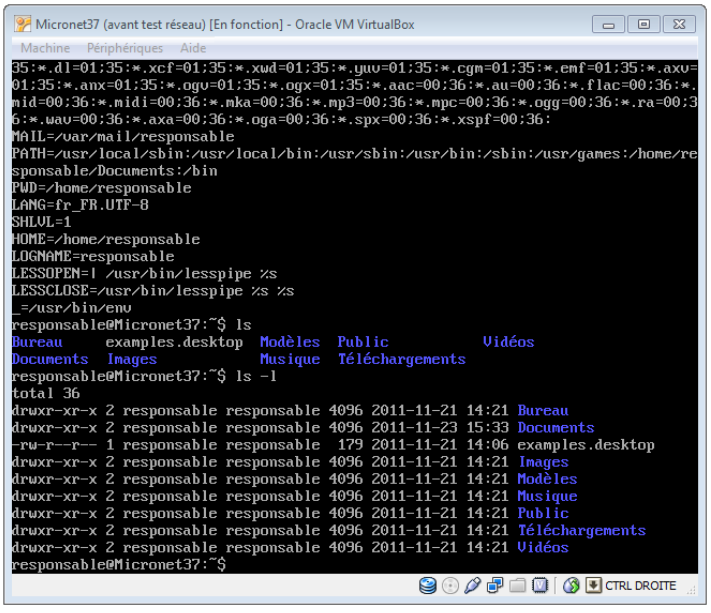
La connaissance du shell est indispensable au travail de l'administrateur unix :

- le travail en "ligne de commande" est souvent beaucoup plus efficace qu'à travers une interface graphique;
- dans de nombreux contextes (serveurs, liaisons distantes lentes) on ne dispose pas d'interface graphique;
- le shell permet l'automatisation aisée des tâches répétitives (scripts);
- de très nombreuses parties du système UNIX sont écrites en shell, il faut être capable de les lire pour comprendre et éventuellement modifier leur fonctionnement.

Il existe plusieurs versions de shell : ***sh*** (ancêtre de bash), ***csh*** (C shell), ***ksh*** (Korn shell), ***zsh***, etc. Nous avons choisi d'enseigner **bash** car il s'agit d'un logiciel libre, utilisé sur toutes les distributions récentes de Linux et de nombreuses autres variantes d'UNIX.

Connaissant bash, l'apprentissage d'un autre shell sur le terrain ne devrait pas poser de difficultés.

Pour de nombreuses tâches simples, c'est effectivement très commode le shell est bien adapté. Le langage shell est forcément assez limité; pour des programmes plus ambitieux il est recommandé d'utiliser des langages plus évolués comme Python ou Perl, voire des langages compilés (C, C++) si l'on désire optimiser au maximum les performances (au prix de coûts de développement plus importants).

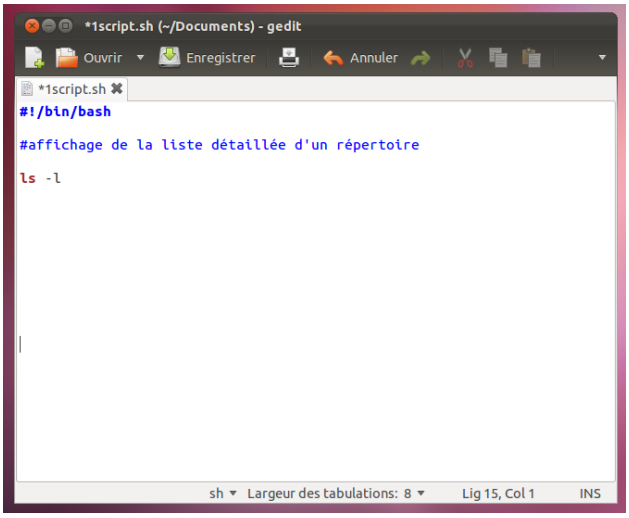


II- Création d'un script Bash

Pour créer un script bash nous devons passer par un éditeur de texte. **Vi** et **Gedit** sous linux sont les plus utilisés. On peut les utiliser une ligne de commande mais nous privilégierons dans un premier temps l'utilisation de l'interface graphique pour ne pas ajouter des difficultés de compréhension. L'éditeur que nous utiliserons sera Gedit.

1ère étape : Création d'un script sous Gedit (exemplecours1.sh)

Dans l'éditeur de texte voilà à quoi ressemble un script bash :



La **première ligne** du script indique le shell utilisé, ici le **bash**.

2ème étape : enregistrement du script

Les extensions des fichiers ne sont pas gérées par Linux mais par convention on rajoute l'extension **.sh** à la suite du nom du script pour que les autres développeurs connaissent le langage utilisé.

Deux possibilités d'enregistrement :

- Soit un enregistrement dans un dossier quelconque et pour lancer le script il faut utiliser : **./1script.sh**
- Soit un enregistrement dans un dossier **PATH** qui permet de lancer une application directement de n'importe où dans l'arborescence : **1script.sh**

Pour avoir la liste des dossiers PATH il faut saisir la commande suivante : **echo $PATH**

Si vous désirez inclure votre espace personnel dans les dossiers PATH il faut utiliser la commande suivante : PATH= $PATH:/home/utilisateur

3ème étape : changer les droits d'exécution du fichier

Grâce à la commande déjà étudiée **chmod** vous devez changer les droits du fichier que vous venez de créer pour accepter les droits d'exécution.

4ème étape : le débuggage

Plus tard, vous ferez probablement de gros scripts et vous risquez de rencontrer des bugs. Il faut donc dès à présent que vous sachiez comment débugger un script. Il faut utiliser la commande : **bash -x 1script.sh**

III- Quelques éléments de programmation *(voir site : http://www.siteduzero.com)*

A- Les variables

1- Déclaration de variable

Affectation d'une variable :
nom = 'Florient'

Afficher une variable :
echo Florient *ou* **echo $**nom

Si vous voulez inclure une variable dans un texte utilisez les double quotes :
echo ''Bonjour **$nom**''

Si vous désirez activer une commande bash il faut utiliser les back quotes :
echo ''Voici les fichiers situés dans le répertoire home **`ls`** ''

2- Saisie d'une variable

```
read –p 'entrez votre nom :' nom
echo "Bonjour $nom !"
```

3- Effectuer des opérations

En bash toutes les variables sont considérées comme des chaînes de caractères. Il faut donc utiliser la commande « **let »**pour lui indiquer qu'il va devoir faire un calcul.

```
let "a = 5"
let "b = 2"
let "c = a * b"
echo $c
```

B- Les conditions (exemplecours2.sh)

```
nom1="Florient"
nom2="Nabil"

if [ $nom1 = $nom2 ]
then
    echo "même prénom !"
else
    echo "prénom différent !"
fi
```

Comme tout langage de programmation possibilité d'imbriquer les conditions avec **elif (**sinon si…).

C- Les boucles

1- Tant que

```
while   [ test ]
do
        echo 'Action en boucle'
done
```

2- For

```
for     variable in 'valeur1' 'valeur2' 'valeur3'
do
        echo "La variable vaut $variable"
done
```

Tests binaires pour les conditions :

- chaine1 = chaine2 : vraie si chaine1 est égale a chaine2
- chaine1 != chaine2 : vraie si chaine1 n'est pas égale à chaine2
- n1 -eq n2 : vraie si n1 est égal a n2
- n1 -ne n2 : vraie si n1 est diffèrent de n2
- n1 -gt n2 : vraie si n1 est plus grand strictement a n2
- n1 -ge n2 : vraie si n1 est plus grand ou égal à n2
- n1 -lt n2 : vraie si n1 est plus petit strictement a n2
- n1 -le n2 : vraie si n1 est plus petit ou égal à n2

D- **Les fonctions*(exemplecours3.sh)***

On peut définir des fonctions en Bash, cela peut être très utile pour structurer ses programmes. La syntaxe est la suivante :

ma_fonction( ) { corps }

Pour appeler la fonction dans le script ce sera :

ma_fonction param1 param2 param3 ...

A l'intérieur du corps de la fonction, les paramètres sont disponibles dans les variables $0, $1, $2 ... Le nombre de paramètres étant toujours $#. Une variable utilisée à l'intérieur d'une fonction est globale! Pour éviter ce phénomène, il faut rajouter **local** à la déclaration de la variable :

local MA_VARIABLE

Exemple :

```
#!/bin/bash

#déclaration de la fonction
valeur=0
calcul() {
somme=$(( $1 + $2 ))
local valeur=$somme
}

#execution de la fonction
calcul 5 9
#retourne "14"
echo $somme
#retourne "0"
echo $valeur
```

Remarque : la déclaration de la fonction dans le script doit ABSOLUMENT se faire avant son appel.

E- La structure case : *(exemplecours4.sh)*

Supposons que le script doive réagir différemment selon la valeur d'une variable; on va faire plusieurs cas selon la valeur de cette variable :

```
case valeur in
 expr1) commandes ;;
 expr2) commandes ;;
 ...
esac
```

Exemple :

```
read langue
case $langue in
francais) echo Bonjour ;;
anglais) echo Hello ;;
espagnol) echo Buenos Dias ;;
esac
```

F- Précision sur les boucles :

Si vous voulez que ***for*** boucle un certain nombre de fois, la syntaxe du script shell peut s'avérer quelque peu fastidieuse :

```
#!/bin/bash

for i in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
    echo $i
done
```

Pour itérer par exemple 250 fois, vous devez saisir tout cela au clavier ! Heureusement, il y a un « raccourci », la commande « **seq »**, qui affiche une séquence de nombres allant de 1 jusqu'au maximum indiqué, par exemple

```
#!/bin/bash

for i in $(seq 15)
do
    echo $i
done
```

IV- Quelques commandes utiles pour la programmation

A- Filtre : la commande « grep » (exemplecours5.sh)

`grep` est un filtre. Il peut trouver un mot dans un fichier, par exemple :

```
grep –r read /home/sio1/Bureau/Batch/*
```

Cherche la chaine de caractères *read* dans tous les fichiers du répertoire /home/sio1/Bureau/Batch/*

On peut insérer une variable dans le critère de recherche (utile pour les scripts shells) :

```
grep –r $variable /home/sio1/Bureau/Batch/*
```

Par ailleurs, plusieurs options peuvent être utilisées.

Options courantes de la commande **grep**.

Option	Signification
-c	Nombre de ligne trouvées (sans les afficher).
-i	Ne fait pas la différence entre majuscule et minuscule.
-n	Affiche le numero de la ligne.
-l	Affiche le nom du fichier contenant la ligne (et pas la ligne).
-v	Affiche toutes les lignes qui ne contiennent pas le mot en question.

B- Les tubes : « | » (exemplecours6.sh)

Utiliser un tube pour filtrer la sortie d'une commande :

cd /home/sio1/Bureau/Batch/ | grep –r *read*

C- La redirection : « >> » (exemplecours7.sh)

cd /home/sio1/Bureau/Batch/ | grep –r *read* >> texte.txt

Attention si vous utilisez seulement “>” et non “>>” vous allez écraser le fichier de destination.