B1 Ynov - Linux Essentials

Xavier Morel

Opus Solutions

Entree/Sorties

Entree/Sorties

Par defaut, tout processus possede

- 1 entree
- 2 sorties

Pour les programmes interactifs (comme les shells)

- les entrees proviennent du clavier
- les sorties s'affichent sur l ecran

Entrees / Sorties

Entree : STDIN (entre standard) -> ce qui est envoye vers le processus

Sortie STDOUT (sortie standard) -> Ce qui est envoye par le processus

Sortie STDERR (sortie erreur standard) -> les erreurs renvoyees par le processus

Descripteurs Entree-Sortie

STDIN, STDOUT et STDERR sont des noms symboliques correspondant a des “descripteurs de fichiers”

- STDIN (INput) : descripteur de fichier 0
- STDOUT (OUTput) : descripteur de fichier 1
- STDERR (ERRor) : descripteur de fichier 2

Entrees/Sorties : exemple de cat

```
cat etc/hosts
cat etc/nexistepas
```

Redirections d'E/S

Les E/S peuvent etre redirigees de ou vers un fichier :

```
# STDIN provient du fichier (et non du clavier)
processus < fichier

# STDOUT est ecrit dans un fichier (et non dans un terminal)
processus > fichier

# STDERR est ecrit dans un fichier (et non plus sur le terminal)
processus 2> fichier

# STDOUT est ecrit dans fic1 et STDERR dans fic2
processus > fic1 2> fic2
```

Exemples de redirection d'E/S avec cat

```
cat < /etc/hostname
cat /etc/hostname

cat /etc/hostname > /tmp/test
cat /tmp/test

cat /etc/hostname > dev/null # Rappel : c'est un trou noir.

cat /etc/hostname /etc/nexistepas > out.txt 2> err.txt
cat err.txt
```

Doubles redirections

“>” ecrase le fichier-cible de la redirection.

La double-redirection “»” evite ce probleme -> il concatene !

```
echo "hey" > fic1
echo "hey again" > fic1
cat fic1
```

```
echo "hey" > fic1
echo "hey again" >> fic1
cat fic1
```

Codes de retour

Le paramètre spécial ? du shell (à ne pas confondre avec le caractère générique ? du shell) contient le code de retour de la dernière commande exécutée

- OK : 0
- Not OK : 1-255

NB : Chaque commande positionne « à sa manière » les codes de retour différents de 0.
-> Un code de retour égal à 1 positionné par `ls` n'a pas la même signification qu'un code de retour à 1 positionné par `grep`.
-> Une solution : lire le manuel correspondant à la commande ...

Programmation Shell

Programmation Shell

Shell != simple interpreteur de commandes ...

Veritable langage de programmation, avec notamment :

- gestion des variables
- tests
- boucles
- operations sur variables
- fonctions
- ...

Structure et execution d'un script

La premiere ligne ("Shebang") permet de preciser quel shell va executer le script

```
#!/usr/bin/sh
#!/usr/bin/ksh
```

Structure et execution d' un script

Instructions et commandes sont regroupees au sein d'un - script.

A l'execution d'un script, chaque ligne sera lue une a une et executee.

Une ligne peut :

- etre composee de commande internes
- etre composee de commandes externes
- etre un commentaire
- etre vide

Structure et execution d'un script

Plusieurs instructions par lignes sont possibles :

- separees par le ";" (equivalent au saut de ligne)
- ou liees conditionnellement par "&&" ou "||"

Structure et execution d'un script

Par convention, les fichiers shell scripts se terminent generalement :

- par .sh pour les Bourne Shell et Bourne Again Shell
- par .ksh pour le Korn Shell
- par .csh pour le C shell

Structure et execution d'un script

Pour rendre un script executable :

```
[Bob@machine]$ chmod u+x monscript
```

Pour l'executer :

```
[Bob@machine]$ ./monscript
```

Pour eviter le ./ :

```
[Bob@machine]$ PATH=&PATH:.
[Bob@machine]$ monscript
```

Arguments d'un script

De la meme maniere que certaines commandes ont des arguments necessaires a leurs executions, il est possble de fournir a un script des arguments (=parametres) lors de son appel.

- parametres positionnels
- parametres speciaux

Parametres positionnels

Arguments passes en “parametre” sur la ligne de commande.

Ils sont affectes aux variables reservees 1, 2, 3, ..., et sont accessibles via la syntaxe :

```
$1
$2
...
${15}
```

Rappel : De maniere generale, on accede a la valeur d’une variable VAR par la syntaxe $VAR ! ## Exemple

```
#!/bin/bash
# affiche_param.sh

echo "Le 1er paramètre est : $1"
echo "Le 3ème paramètre est : $3"
```

```
./affiche param.sh 1 deux trois 4 cinq
Le 1er paramètre est : 1
Le 3ème paramètre est : trois
```

Parametres et caracteres speciaux

Si un parametre/argument contient un caractere special ou un espace, il faut utiliser les quotes ””

```
./affiche_param.sh un 2 "le troisieme parametre" quatre
```

Parametres speciaux

Variables reservees. Tres utiles pour effectuer des traitements sur les parametres eux memes.

- $0 : contient le nom du script
- $* : l'ensemble des parametres sous la forme d'un seul argument
- $@ : l'ensemble des parametres, sous forme de liste
- $# : le nombre de parametres passes au script
- $? : le code retour de la derniere commande
- $$: le PID du shell executant le script

Exemple :

```
$ vi ./script.sh
1  #! /bin/bash
2  # Le script doit recevoir au minimum 2 arguments
3  if [ $# -lt 2 ]
4  then
5          # Si le nombre d'arguments est inférieur à 2
6          # on retourne le code erreur 1
7          echo "Nombre arguments incorrect"
8          exit 1
9  else
10       # Si le nombre d'arguments est supérieur ou égal à 2
11          # on retourne le code erreur 0
12          echo "Nombre arguments correct"
13          exit 0
14  fi
$
```

Tests et structures de controle

Les boucles et structures de controle determinent le flux d'execution d'un programme : choix entre l'execution d'un bloc de code ou d'un autre, repetition d'execution de code, sortie forcee de bloc, ...

Exemple : if [condition] then [execution]

Structure IF

```
# Structure simple
if [condition]
then
  echo "OK"
fi # Ne pas oublier de fermer la condition par un "fi"

# Structure composee
if [test1]
then
  echo "Premiere condition OK"
elif [test2]
  echo "Deuxieme condition OK"
else
  echo "All KO"
fi
```

Structure WHILE

Executent “instructions” tant que “condition” est vraie. Si “condition” est fausse des l’entree de la boucle, celle ci n‘est pas executee.

```
while [condition]
do
  instructions
done
```

Attention donc aux conditions de sortie ...

Structure FOR

Ces boucles executent “instructions” successivement pour chaque valeur de “variable” contenue dans “suite”

```
for variable in 'suite'
do
  instructions
done
```

```
for fichier in `ls`
do
  echo "J'ai trouve $fichier"
done
```

La commande test

Commande permettant de faire un test et de renvoyer 0 si tout est OK , et 1 en cas d'erreur.

-> Fondamentale pour les structures de controle, en particulier IF.

-> man test pour connaitre tout les operateurs

Test : Operateurs sur fichiers (liste non-exhaustive)

Les operateurs suivant renvoient 0 si ...

-e fichier	fichier existe
-d fichier	fichier est un repertoire
-f fichier	fichier est “normal”
-w fichier	fichier existe et est en ecriture
-x fichier	fichier existe et est executable
fic1 -ot fic2	fic1 est plus vieux que fic2

Test : Operateurs de comparaison numeriques

Les operateurs suivant renvoient 0 si ...

$A -lt 5	$A est strictement inferieur a 5
$A -le 5	$A est inferieur ou egal a 5
$A -gt 5	$A est strictement superieur a 5
$A -ge 5	$A est superieur ou egal a 5
$A -eq 5	$A est egal a 5
$A -ne 5	$A est different de 5

Test : Syntaxe crochets

On peut raccourcir la commande test par des crochets.

```
test -f /etc/passwd
# Syntaxe equivalente
[-f /etc/passwd]
```

Test : OR et AND logiques

Syntaxe des operateurs OR et AND : -o et -a.

exemple :

```
if [$1 -lt 1 -o $2 -eq 100]
then
  echo "Le premier argument est inferieur a 1 ou egal a 100"
fi

if [$2 -lt 1 -a $2 -gt 2]
then
  echo "Ceci ne sera jamais affiche ..."
fi
```

Les Variables

Les Variables

Trois types de variables :

- variables utilisateur
- variables systeme
- variables speciales

Principe : affecter un contenu a un nom de variable (chaine de caracteres, valeurs numeriques, ...)

Nomenclature

Regles pour nom de variables :

- Compose de lettres minuscules, majuscules, chiffres, caracteres de soulignement
- Le premier caractere ne peut etre un chiffre
- Taille d'un nom en principe limitee
- Convention : variables utilisateurs en minuscules pour les differencier des variables systemes

Declaration et affectation

Une variable est declaree des qu'une valeur lui est affectee.

Affection effectuee avec le signe "="

```
var = Hello
```

Acces et affichage

Acces au contenu d'une variable en placant le signe "$" devant le nom de la variable.

Quand le shell rencontre "$", il tente d'interpreter le mot suivant comme etant une variable.

Si elle existe, alors "$nom_variable" est remplace par son contenu. Sinon, texte vide.

Acces et affichage

```
$ chemin=/tmp/
$ ls $chemin
$ cd $chemin
```

Pour afficher la liste de toutes les variables (systemes et utilisateurs), on utilise la commande set.

Acces et affichage

La commande echo permet d'afficher le contenu de variables specifiques

```
$ a=Jules
$ b=Cesar
$ echo $a $b a conquis la Gaule
Jules Cesar a conquis la Gaule
```

Caracteres speciaux

Une variable peut contenir des caracteres speciaux. Par exemple. l'espace. Mais ...

```
$ c=Salut les copains
les : not found
$ echo $c
```

... ne marche pas ...

Verrouillage de caracteres speciaux

Pour resoudre ce probleme, il faut verrouiller les caracteres speciaux :

- soit un par un
- soit de les mettre entre guillements ou apostrophes

```
$ c=Salut\ les\ Copains # Solution lourde
$ c="Salut les copains" # Solution correcte
$ c='Salut les copains' # Solution correcte
```

Difference entre guillemets et apostrophes

Les deux solutions precedentes ont une subtile difference -> interpretation des variables et substitions.

```
$ a=Jules
$ b=Cesar
$ c="$a $b a conquis la Gaule"
$ d='$a $b a conquis la Gaule'
$ echo $c
Jules Cesar a conquis la Gaule
$ echo $d
$a $b a conquis la Gaule
$ echo "Unix c'est top"
Unix c'est top
$ echo 'Unix "trop bien"
'Unix "trop bien"
```

Suppression et protection

On supprime une variable avec la commande unset.

On protege une variable en ecriture avec la commande readonly

Une variable protegee en lecture seule, meme vide, est figee. Il n'existe aucun moyen de la remplacer ou de la supprimer, sauf quitter le shell.

Suppression et protection

```
$ a=Jules
$ b=Cesar
$ echo $a $b
Jules Cesar
$ unset b
$ echo $a $b
Jules
$ readonly a
$ a=Neron
a: is read only
$ unset a
a: is read only
```

Exportation

Par defaut, une variable n'est accessible que depuis le shell ou elle a ete definie.

```
$ a=Jules
$ echo 'echo "a=$a"' > voir_a.sh
$ chmod u+x voir_a.sh
$ ./voir_a.sh
a=
```

La commande export permet d'exporter une variable pour que son contenu soit visible hors de son shell de definition.

Exportation

Les variables exportees peuvent etre modifiees dans le script, mais ces modifications ne s'appliquent qu'au script ou au sous-shell.

```
$ export a
$ ./voir_a.sh
a=Jules
$ echo 'a=Neron ; echo "a=$a"' >> voir_a.sh
$ ./voir_a.sh
a=Jules
a=Neron
$ echo $a
Jules
```