Système d'Exploitation - Travaux Dirigés n°7

« Programmation en shell »

Notions vues dans ce TD : la programmation en shell (login, affectation et portée des variables, ...).

Nombre de séances de 2h prévues pour faire ce TD : 2.

Prochain TD : Suite du shell.

PS : Les parties correspondant à du travail à faire sont toutes en italiques ; le restant étant du complément au cours.

La variable PATH permet de spécifier au shell la liste ordonnée des chemins qui seront parcourus pour trouver les commandes lancées sur la ligne de commande ou à partir de scripts.
Modifier cette variable PATH en lui ajoutant en fin de liste le répertoire courant « . ». Vérifier la prise en compte de votre modification à l'aide de la commande « echo ».

prompt> ***PATH=$PATH:.***
prompt> ***echo $PATH***
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:.
prompt>

Que constatez-vous si vous faites cette vérification dans un autre shell ?

La variable PATH n'a pas été modifiée. Elle est locale à chaque shell.

Pour que cette modification soit effective inscrivez-la à la fin du fichier de configuration du bash « $HOME/.bashrc ». Le fichier « $HOME/.bashrc » devra être réactivé (« . .bashrc » ou « source .bashrc ») dans votre shell courant, afin de tenir compte des dernières modifications. Elles seront prises en compte automatiquement pour les nouveaux shells lancés.

Un fichier script est un fichier construit à partir d'un éditeur, dans lequel se trouve des commandes ou des instructions compréhensibles par le shell. Le caractère « # » permet de mettre des commentaires dans le fichier. La première ligne commençant par la séquence « #! » indique au noyau quel est le shell à utiliser pour interprêter les instructions contenues dans le script. Attention, les caractères « # » et « ! » doivent être respectivement le premier et le deuxième caractères du fichier.

prompt> **cat mon_script1**
#! /bin/bash
echo "Voici les variables d'environnement du shell :"
echo " " # permet de sauter une ligne à l'affichage
env | tee /dev/pts/0 | wc -l > f1.txt
echo " "
echo "Il en existe `cat f1.txt`"
prompt>

Construiser ce fichier « mon_script1 » à partir d'un éditeur et essayer de le lancer dans la fenêtre du shell . Que constatez-vous ?

Le script n'a pas les droits d'exécution pour pouvoir être lancé directement dans un shell.

Comment remédier à ce problème ? Vous pouvez utiliser deux techniques distinctes (modification des droits ou lancement par un mécanisme particulier).

prompt> ***Chmod u+x mon_script1***
ou
prompt> ***. mon_script1***

Que fait ce script ? Expliquer en particulier la ligne utilisant les tubes (|), ainsi que la dernière ligne du script. Attention cette dernière ligne utilise les quotes inverses (back-quote ou « ` »).

Ce script affiche les variables d'environnement du shell courant dans le terminal /dev/pts/0, grâce au « tee » et envoie ces mêmes variables (une par ligne) grâce au « | » à la commande « wc » qui compte le nombre de lignes obtenues, donc de variables ; cette dernière envoie ce nombre dans le fichier « f1.txt ». La dernière ligne du script permet, grâce aux quotes inversées, l'affichage du contenu du fichier « f1.txt » dans la chaîne affichée par « echo ».

Ce script fonctionne uniquement si vous le lancer à partir du terminal qui possède le nom « /dev/pts/0 ». Ouvrez plusieurs fenêtres (/dev/pts/?) et lancer le script après avoir remplacé « /dev/pts/0 » par « /dev/pts/? » ou « /dev/pts/ ». Que constatez-vous ?*

Le résultat du « tee » va dans tous les terminaux « /dev/pts/ » ouverts, y compris le terminal qui a lancé le script.

Modifier le script afin qu'il fonctionne depuis n'importe quel terminal et uniquement pour ce terminal. Vous pourrez utiliser une variable.

Il suffit de créer une variable « var » dans le script et de lui affecter le résultat de la commande « tty » (var=`tty`), pour ensuite utiliser le contenu de cette variable « $var » dans la commande « tee ».

Créer un répertoire « bin » sous votre répertoire dédié aux cours systèmes, ou sous votre répertoire de connexion (« $HOME » ou « ~ »). Lorsque votre commande/script « mon_script1 » fonctionnera correctement, vous pourrez y faire des ajouts ; placer cette commande (« cp ») dans ce répertoire « bin » et modifier la variable PATH du shell afin que vous puissiez appeler directement cette commande, sans préciser où elle se trouve.

prompt> pwd
/home/user1/dev
prompt> mkdir $HOME/bin
prompt> cp mon_script1 ../bin
prompt> PATH=$PATH:$HOME/bin:.
prompt> echo $PATH
/home/user1/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user1/bin:.
prompt>

Les variables en shell se créent en déclarant leur nom (chaîne de caractères sans blanc) et en y affectant une valeur non typée à l'aide du signe « = » sans blanc autour. Créer le script « mon_script2 » contenant les instructions suivantes :

prompt> **cat mon_script2**
#! /bin/bash
echo $$
ma_variable=2011
echo $ma_variable
trap "echo fin du script" EXIT # la commande « trap » permet de lancer la commande passée en paramètre à l'arrivée du
 # pseudo signal « EXIT » (signal simulé par le shell), reçu par le shell lorsqu'il se termine
prompt>

Après avoir modifié les droits d'accès de cette commande, et l'avoir lancée dans votre terminal, expliquer le résultat.

prompt> chmod u+x mon_script2

La variable « $$ » affiche le PID du processus courant en l'occurrence ici le shell qui a lancé le script. Suivant la façon dont est lancé le script (« . » ou directement) le pid affiché correspond respectivement au pid du shell de départ ou à celui

du shell créé par celui de départ pour interpréter le script.

Qu'obtenez-vous en faisant les commandes suivantes ?

prompt> **echo $ma_variable**
prompt>**echo $$**

La variable « ma_variable » n'est pas définie dans le shell courant, car en lançant « mon_script2 » comme une commande, le shell courant crée un autre shell pour interpréter le script.

Que se passe-t-il si vous lancer votre script de la façon suivante ? Expliquer.

prompt> **. mon_script2**
Ou encore :
prompt> **source mon_script2**

Dans ce cas c'est bien le shell courant qui interprête le script, donc la variable « ma_variable » reste définie dans le shell courant après l'interprétation du script.

Les variables

R éaliser la séquence d'instruction suivante et commenter les résultats obtenu. Vous devez en conclure quelque chose sur le type des variables en shell.

prompt> **a=3**
prompt> **echo $a**
3
prompt> **a=a+1**
prompt> **echo $a**
a+1
prompt> **a=3**
prompt> **a=$a+1**
prompt> **echo $a**
3+1
prompt> **singulier=mot**
prompt> **pluriel=${singulier}s**
prompt> **echo $pluriel**
mots

L'expression ${variable:debut:longueur} permet d'extraire automatiquement une partie de la variable en question.

prompt> **variable=ABCDEFGHIJKLMNOPQRSTUVWXYZ**
prompt> **echo ${variable:5:2}**
FG
prompt> **echo ${variable:20}**
UVWXYZ

L'expression ${variable#motif} est remplacée par la valeur de la variable, de laquelle on ôte la chaîne initiale la plus courte qui correspond au motif. Le caractère « # » permet de travailler sur le préfixe des variables.

prompt> **variable=ABLABLACDEFGHIJKLMNOPQRBLABLASTUVWXYZ**
prompt> **echo ${variable#BLABLA}**

ABLABLACDEFGHIJKLMNOPQRBLABLASTUVWXYZ
prompt> **echo ${variable#*L}**
ABLACDEFGHIJKLMNOPQRBLABLASTUVWXYZ
prompt> **echo ${variable#*[QRSPZ]}**
QRBLABLASTUVWXYZ

L'expression ${variable##motif} sert à éliminer le plus long préfixe correspondant au motif transmis.

prompt> **variable=ABLABLACDEFGHIJKLMNOPQRBLABLASTUVWXYZ**
prompt> **echo ${variable##BLABLA}**
ABLABLACDEFGHIJKLMNOPQRBLABLASTUVWXYZ
prompt> **echo ${variable##*L}**
ASTUVWXYZ
prompt> **echo ${variable##*[QRSP]}**
TUVWXYZ

Symétriquement les expressions ${variable%motif} et ${variable%%motif} correspondent au contenu de la variable indiquée, qui est débarrassée, respectivement, du plus court et du plus long suffixe correspondant au motif transmis.

prompt> **variable=ABLABLACDEFGHIJKLMNOPQRBLABLASTUVWXYZ**
prompt> **echo ${variable%BLABLA*}**
ABLABLACDEFGHIJKLMNOPQR
prompt> **echo ${variable%%BLABLA*}**
A
prompt> **echo ${variable%[P-Z]*}**
ABLABLACDEFGHIJKLMNOPQRBLABLASTUVWXY
prompt> **echo ${variable%%[P-Z]*}**
ABLABLACDEFGHIJKLMNO

L'opérateur « / » permet le remplacement d'une portion de la chaîne par un motif : ${variable/motif/remplacement}.

prompt> **cd /etc/X11**
prompt> **echo ${PWD/$HOME/~}**
/etc/X11
prompt> **cd**
prompt> **echo ${PWD/$HOME/~}**
/home/user1
prompt> **cd SE**
prompt> **echo ${PWD/$HOME/~}**
/home/user1/SE

Voici quelques cas concrets d'utilisation de ces opérateurs, à compléter.

prompt> **adresse=utilisateur@machine.org**
prompt> **echo ${adresse%%@*}** *# retrouver le nom de login dans une adresse e-mail*
utilisateur
prompt>
prompt> **adresse=machine.entreprise.com**
prompt> **echo ${adresse%%.*}** *# retrouver le nom d'hôte dans une adresse complète de machine*
machine

prompt>
prompt> **fichier=/usr/src/linux/kernel/sys.c**
prompt> **echo ${fichier##*/}** *# obtenir le nom d'un fichier débarrassé de son chemin d'accès*
sys.c
prompt>
prompt> **fichier=module1.c**
prompt> **echo ${fichier%.*}** *# éliminer l'extension éventuelle d'un nom de fichier*
module1
prompt>

L'opérateur « ${#variable} » permet également de calculer la longueur d'une variable. Cette longueur exprime le nombre de caractères contenus dans la variable.

prompt> **grep bash /etc/passwd**
root:x:0:0:root:/root:/bin/bash
couchdb:x:106:113:CouchDB Administrator,,,:/var/lib/couchdb:/bin/bash
user1:x:1000:1000:user1,,,:/home/user1:/bin/bash
prompt> **grep bash /etc/passwd | wc -c**
151
prompt> **variable=$(grep bash /etc/passwd)**
prompt> **echo ${#variable}**
150

Calculs arithmétiques simples, en shell : $((opérations))

*+, -, *, / pour l'addition, la soustraction, la multiplication et la division*
% pour le modulo
<< et >> pour les décalages binaires à gauche et à droite
&, |, ^ pour les opérateurs binaires ET, OU et OU exclusif
~ pour la négation binaire

Expliquer les résultats obtenus par les expressions suivantes :

prompt> **echo $((2 * (4 + (10/2)) - 1))**
17
prompt> **echo $((7 % 3))**
1

prompt> **a=1+2**
prompt> **echo $a**
1+2
prompt> **echo $((a))**
3
prompt> **echo $(($a))**
3
prompt> **echo $(($a*2))**
5 # (1+(2*2))
prompt> **echo $a**
1+2
prompt> **echo $((a*2))**
6

La structure $(( )) peut servir à vérifier les conditions arithmétiques. Les opérateurs de comparaison

renvoient 1 quand la condition est vérifiée, 0 sinon.

prompt> **echo $(((25 + 2) < 28))**
1
prompt> **echo $(((12 + 4) == 17))**
0
prompt> **echo $(((1 == 1) && (2 < 3)))**
1

Dans les shells bash et Ksh, il est possible de typer une variable comme arithmétique. Pour cela on utilisera respectivement les instructions « declare -i variable » et « typeset -i variable ». Pour le bash, toute affectation du type « A=xxx » sera évaluée sous la forme «A=$((xxx)). Si la valeur affectée à une telle variable est une chaîne de caractères ou est indéfinie, la valeur retenue sera zéro.

prompt> **declare -i A**
prompt> **A=1+1**
prompt> **echo $A**
2
prompt> **B=1+1**
prompt> **echo $B**
1+1
prompt> **A=4*7 + 67**
+ : commande introuvable
prompt> **A="4*7 + 67"**
prompt> **echo $A**
95

L'option « r » permet avec l'instruction « declare » de figer le contenu d'une variable (cela devient une constante). Ceci ne concerne que le processus en cours.

prompt> **echo $$**
2836
prompt> **C=immuable**
prompt> **echo $C**
immuable
prompt> **declare -r C**
prompt> **C=modifiée**
bash: C : variable en lecture seule
prompt> **echo $C**
immuable
prompt> **unset C**
bash: unset: C : « unset » impossible : variable est en lecture seule
prompt> **export C # transmission de la variable C aux sous-processus**
prompt> **bash # création d'un sous-processus bash (shell)**
prompt> **echo $$**
2931
prompt> **echo $C**
immuable
prompt> **C=changée**
prompt> **echo $C**
changée
prompt> **exit # fin du sous-processus bash (shell)**
exit
prompt> **echo $$**

2836
prompt> **echo $C**
immuable
prompt>

L'opérateur « $(commande) » permet de substituer la commande passée en paramètre. On obtient le même effet qu'avec l'instruction « `commande` ».

prompt> **ls -l**
total 4
-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f1
-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f2
-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f3
drwxr-xr-x 2 francois francois 4096 2011-11-15 23:18 rep2
prompt> **variable=$(ls -l)**
prompt> **echo $variable**
total 4-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f1-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f2
-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f3drwxr-xr-x 2 francois francois 4096 2011-11-15 23:18 rep2
prompt>
prompt> **echo "$variable"**
total 4
-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f1
-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f2
-rw-r--r-- 1 francois francois 0 2011-11-15 23:18 f3
drwxr-xr-x 2 francois francois 4096 2011-11-15 23:18 rep2
prompt>

Portée des variables

La portée d'une variable du shell se caractérise par la visibilité que les sous-processus et les fonctions ont sur ces variables. On peut très facilement définir une fonction en shell, en utilisant le mot-clé « function » suivi du nom de la fonction que vous voulez créer. En passant à la ligne le shell affiche ce que l'on appelle un prompt de deuxième niveau (ici « > »). L'apparition du caractère « } » fera revenir le shell à son prompt de premier niveau, marquant ainsi son attente d'une nouvelle commande à interprêter. Une variable définie simplement, comme dans l'exemple ci-dessous, est accessible (en lecture et en écriture) dans l'ensemble du processus courant, y compris les fonctions.

prompt> **var=123**
prompt> **function affiche_var ( )**
> **{**
> **echo $var**
> **}**
prompt> **affiche_var**
123
prompt> **var=456**
prompt> **affiche_var**
456
prompt>

Une nouvelle variable définie dans une fonction, restera définie jusqu'à la fin du processus, y compris en dehors de la dite fonction.

prompt> **set -u** *# affiche un message d'erreur si la variable invoquée n'existe pas*
prompt> **function definit_var ( )**
> **{**
> **nouv_var=123**
> **}**
prompt> **echo $nouv_var**
bash : nouv_var : unbound variable
prompt> **definit_var**
prompt> **echo $nouv_var**
123
prompt>

Lancer le script suivant « var_locales » et concluer sur la signification du mot-clé « local » affecté à une variable dans une fonction.

prompt> **cat var_locales**
#! /bin/bash
function ma_fonction ( )
{
local var="dans fonction"
echo " entrée dans ma_fonction"
echo " var = " $var
echo " appel de sous_fonction"
sous_fonction
echo " var = " $var
echo " sortie de ma_fonction"
}
function sous_fonction ( )
{
echo " entrée dans sous_fonction"
echo " var = " $var
echo " modification de var"
var="dans sous_fonction"
echo " var = " $var
echo " sortie de sous_fonction"
}
echo "entrée dans le script"
var="dans le script"
echo "var = " $var
echo "appel de ma_fonction"
ma_fonction
echo "var = " $var
prompt>
prompt> **var_locales**
entrée dans le script
var = dans le script
appel de ma_fonction
entrée dans ma_fonction
var = dans fonction
appel de sous_fonction
entrée dans sous_fonction
var = dans fonction
modification de var
var = dans sous_fonction
sortie de sous_fonction
var = dans sous_fonction

sortie de ma_fonction
var = dans le script
prompt>

On voit sur le résultat du lancement du script qu'une variable locale, n'est accessible que pour la fonction où elle a été définie, mais également dans les sous-fonctions appelées par cette fonction. Toutes les autres fonctions n'ont pas accès à cette variable. Contrairement aux autres variables, non locales, qui sont accessibles par toutes les fonctions du script.

Lorsqu'un processus « parent » crée un processus « enfant », les variables d'environnement du « parent » sont héritées dans « l'enfant ». « Héritée » pour une variable, signifie recopiée dans le processus « enfant ». Le principe fondamental pour un processus, est qu'il ne partage pas ses propres variables avec d'autres processus. Au moment de l'héritage ou encore de la recopie, la variable est créée avec le même nom dans « l'enfant » que dans son « parent », avec la valeur au moment de la recopie. C'est le mot-clé « export » qui permet ce mécanisme d'héritage. Observons ce qui se passe dans les instructions suivantes. Expliquer ce principe d'héritage de variables entre processus. Qu'observe-ton pour la variable « var2 » de retour dans le shell de départ ?

prompt> **echo $$**
2836
prompt> **var1="variable non-exportée"**
prompt> **var2="première variable exportée"**
prompt> **export var2**
prompt> **export var3="seconde variable exportée"**
prompt> **export v1=1 v2=2**
prompt> **echo $v1 $v2**
1 2
prompt> **/bin/bash**
prompt> **echo $$**
3058
prompt> **echo $var1**

prompt> **echo $var2**
première variable exportée
prompt> **echo $var3**
seconde variable exportée
prompt> **var2="variable modifiée dans le second shell"**
prompt> **/bin/bash**
prompt> **echo $$**
3073
prompt> **echo $v1 $v2**
1 2
prompt> **v1=10**
prompt> **echo $v1 $v2**
10 2
prompt> **echo $var2**
variable modifiée dans le second shell
prompt> **echo $var3**
seconde variable exportée
prompt> **exit**
exit
prompt> **echo $$**
3058
prompt> **exit**
exit
prompt> **echo $$**

2836
prompt> **echo $var2 $v1 $v2**
première variable exportée 1 2
prompt>

Notons que l'instruction « export -p » affiche la liste des variables exportées, alors que l'instruction « export -n variable » rend la variable non-exportable pour les processus « enfants » du processus en cours. Vous pouvez le tester sur l'exemple suivant :

prompt> **export var1="exportée"**
prompt> **/bin/bash**
prompt> **echo $var1**
exportée
prompt> **exit**
exit
prompt> **export -n var1**
prompt> **echo $var1**
exportée
prompt> **/bin/bash**
prompt> **echo $var1**

prompt> **exit**
exit
prompt> **echo $var1**
exportée
prompt>