iNFO

IUT GRAND OUEST NORMANDIE

R 1.04

2023 - 2024

Introduction aux systèmes d'exploitation et à leur fonctionnement

TD N°7

« Programmation en Shell »

ANNE Jean-François

D'après le TD de F. BOURDON

Le but de ce TD est de se familiariser avec les systèmes d'exploitation et avec leur fonctionnement.

« Programmation en Shell »

Notions vues dans ce TD :

la programmation en Shell (login, affectation et portée des variables, ...).

Nombre de séance de 2h00 prévu pour faire ce TD : 1.

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

1. Le PATH

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 « . ».*

Pour cela il faut remettre dans la variable son ancien contenu suivi de la nouvelle valeur. Vous observerez que le séparateur n'est pas le caractère blanc, mais le « : ».

✍ *Vérifier la prise en compte de votre modification à l'aide de la commande « echo ».*

✍ *Que constatez-vous si vous faites cette vérification dans un autre Shell ?*

Une autre solution consiste à mettre le répertoire courant en début de variable « PATH ». Expliquez pourquoi cette solution est dangereuse.

✍ *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. Attention à ne pas faire d'erreurs c'est un fichier sensible !*

2. Les scripts

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 interpréter les instructions contenues dans le script. Attention, les caractères « # » et « ! » doivent être respectivement le premier et le deuxième caractère 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>

✍ *Construisez ce fichier « mon_script1 » à partir d'un éditeur et essayez de le lancer dans la fenêtre du Shell. Que constatez-vous ? Comment remédier à ce problème ? Vous pouvez utiliser deux techniques distinctes (modification des droits ou lancement par un mécanisme particulier).*

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

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

✍ *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 pouvez y faire des ajouts, placer cette commande (« mv ») 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.*

✍ *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 lancé dans votre terminal, expliquez le résultat. Qu'obtenez-vous en faisant les commandes suivantes ?*

```
prompt> echo $ma_variable
prompt> echo $$
```

✍ *Que se passe-t-il si vous lancez votre script de la façon suivante ? Expliquez.*

```
prompt> . mon_script2
```

Ou encore :

```
prompt> source mon_script2
```

3. Les variables

✍ *Réalisez la séquence d'instructions suivante et commentez les résultats obtenus. Vous devez en conclure quelque chose sur le type des variables en Shell.*

```
prompt> a=3
prompt> echo $a
???
prompt> a=a+1
prompt> echo $a
???
prompt> a=3
prompt> a=$a+1
prompt> echo $a
???
prompt> singulier=mot
```

Nous allons travailler sur le traitement des chaînes de caractères. Les accolades permettent de délimiter la portée du nom d'une variable.

```
prompt> pluriel=${singulier}s
prompt> echo $pluriel
???
```

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

```
prompt> variable=ABCDEFGHIJKLMNOPQRSTUVWXYZ
prompt> echo ${variable:5:2}
???
prompt> echo ${variable:20}
???
```

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}
???
prompt> echo ${variable#*L}
???
prompt> echo ${variable#*[QRSPZ]}
???
```

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

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

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*}
???
prompt> echo ${variable%%BLABLA*}
???
prompt> echo ${variable%[P-Z]*}
???
prompt> echo ${variable%%[P-Z]*}
???
```

L'opérateur « / » permet de remplacer dans la variable, la première occurrence du motif par la chaîne fournie en troisième position : $ {variable/motif/remplacement}.

```
prompt> cd /etc/X11
prompt> echo ${PWD/$HOME/"~"}
???
prompt> cd
prompt> echo ${PWD/$HOME/"~"}
```

```
???
prompt> var=aaabbbccc
prompt> echo ${var/bbb/111}
???
```

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

```
prompt> adresse=utilisateur@machine.org
prompt> echo ${???} # retrouver le nom de login dans une adresse e-mail
utilisateur
prompt>
prompt> adresse=machine.entreprise.com
prompt> echo ${???} # retrouver le nom d'hôte dans une adresse complète de machine
machine
prompt>
prompt> fichier=/usr/src/linux/kernel/sys.c
prompt> echo ${???} # obtenir le nom d'un fichier débarrassé de son chemin d'accès
sys.c
prompt>
prompt> fichier=module1.c
prompt> echo ${???} # 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
???
prompt> variable=$(grep bash /etc/passwd)
prompt> echo ${#variable}
???
prompt>
```

4. Calculs arithmétiques simples en Shell :

Nous allons travailler sur des expressions numériques en utilisant l'opérateur : $((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))
???
prompt> echo $((7 % 3))
???

prompt> a=1+2
prompt> echo $a
???
prompt> echo $((a))
???
prompt> echo $(($a))
???
prompt> echo $(($a*2))
???
prompt> echo $((a*2))
???
```

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))
???
prompt> echo $(((12 + 4) == 17))
???
prompt> echo $(((1 == 1) && (2 < 3)))
???
```

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
???
prompt> B=1+1
prompt> echo $B
???
prompt> A=4*7 + 67
???
prompt> A="4*7 + 67"
prompt> echo $A
???
prompt> C=5*7
prompt> echo $C
???
prompt> A=C
prompt> echo $A
???
prompt> A="ABC"
prompt> echo $A
???
prompt>
```

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

```
prompt> echo $$
???
prompt> C=immuable
prompt> echo $A
???
prompt> declare -r C
prompt> C=modifiée
???
prompt> echo $C
???
prompt> unset C
```

```
???
prompt> export C # transmission de la variable A aux sous-processus
prompt> bash # création d'un sous-processus bash (shell)
prompt> echo $$
???
prompt> echo $C
???
prompt> C=changée
prompt> echo $C
???
prompt> exit # fin du sous-processus bash (shell)
???
prompt> echo $$
???
prompt> echo $C
???
```

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 $variable
???
prompt> echo "$variable"
???
```

5. 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 interpréter.

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
 ???
prompt> var=456
prompt> affiche_var
 ???
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
 ???
prompt>
```

✍ *Lancer le script suivant « var_locales » et conclure 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>
```

✍ *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 $$
???
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
??? ???
prompt> /bin/bash
```

```
prompt> echo $$
???
prompt> echo $var1
???
prompt> echo $var2
???
prompt> echo $var3
???
prompt> var2="variable modifiée dans le second shell"
prompt> /bin/bash
prompt> echo $$
???
prompt> echo $v1 $v2
??? ???
prompt> v1=10
prompt> echo $v1 $v2
??? ???
prompt> echo $var2
???
prompt> echo $var3
???
prompt> exit
???
prompt> echo $$
???
prompt> exit
exit
prompt> echo $$
???
prompt> echo $var2 $v1 $v2
??? ??? ???
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
???
prompt> exit
???
prompt> export -n var1
prompt> echo $var1
???
prompt> /bin/bash
prompt> echo $var1
???
prompt> exit
???
prompt> echo $var1
???
prompt>
```

II. Webographie :

- https://bourdon.users.info.unicaen.fr/cours/IUT-1A/index.html
-