Système d'Exploitation - Travaux Dirigés n°8.

« Programmation en shell »

CORRIGÉ

Notions vues dans ce TD : la programmation en shell (variables de position et autres variables).

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

L'instruction « shift » permet de décaler les paramètres de position ($0, $1, $2, ...), sauf le premier qui ne bouge pas. Ce dernier ($0) correspond au nom de la commande en cours.

Q1. Ecrire une première version d'un script (mon_script1) qui affiche la valeur des paramètres de position, jusqu'à « $10 » compris, passés à l'appel de ce script. Vous pouvez utiliser l'instruction « if [-n "$i"] ; then … ; fi ».

prompt> ***cat mon_script1***
#! /bin/bash
echo 0 : $0
if [-n "$1"] ; then echo 1 : $1 ; fi
if [-n "$2"] ; then echo 2 : $2 ; fi
if [-n "$3"] ; then echo 3 : $3 ; fi
if [-n "$4"] ; then echo 4 : $4 ; fi
if [-n "$5"] ; then echo 5 : $5 ; fi
if [-n "$6"] ; then echo 6 : $6 ; fi
if [-n "$7"] ; then echo 7 : $7 ; fi
if [-n "$8"] ; then echo 8 : $8 ; fi
if [-n "$9"] ; then echo 9 : $9 ; fi
if [-n "${10}"] ; then echo 10 : ${10} ; fi
prompt>

Q2. Cette première version ne permet pas d'afficher un nombre variable de paramètres passés à l'appel du script. Proposer une deuxième version (mon_script2) qui le permet. Vous utiliserez l'instruction « while [-n "$i"] ; do … done » et la commande « shift ».

prompt> ***cat mon_script2***
#! /bin/bash
while [-n "$1"] ; do
echo $1
shift
done
prompt>

Q3. Que se passe-t-il si vous passez les arguments suivants aux scripts « mon_script1 » et « mon_script2 » ? Qu'en concluez-vous ?

prompt> **mon_script1 un "" deux trois**
0 : mon_script1
1 : un

3 : deux
4 : trois
prompt> **mon_script2 un deux "" trois quatre**
un
deux
prompt>

Le script « mon_script1 » n'a pas affiché $2 car il l'a vu vide, alors que « mon_script2 » s'est arrêté (sortie du while) au premier $i vide, pensant être arrivé à la fin des arguments.

Q4. Pour corriger le problème vu à la question précédente, modifier « mon_script2 » en « mon_script3 » en utilisant le paramètre « $# ». Afficher la valeur de la variable « $# » dans un script auquel vous passerez des paramètres, pour comprendre ce qu 'elle contient.

prompt> ***cat mon_script3***
#! /bin/bash
while [$# -ne 0] ; do
echo $1
shift
done
prompt>

Le paramètre « $@ » permet de récupérer l'ensemble des arguments passés à l'appel d'un script. Par exemple la commande suivante affiche les informations sur les fichiers passés en paramètre :

prompt> **cat ll.sh**
#! /bin/bash
ls -l "$@"
prompt> **ll *.c**
-rw-rw-r-- 1 user1 user1 7 oct. 21 01:47 fich1.c
-rw-rw-r-- 1 user1 user1 7 oct. 21 01:47 fich2.c
prompt>

La protection d'un caratère afin qu'il ne soit pas interprété par le shell, peut se faire soit par un « \ » (barre oblique inverse - backslash), soit par des apostrophes « ' ' », soit encore par des guillemets « " " ».

Le backslash protège uniquement le caractère suivant, y compris lui-même.

prompt> **echo \$a**
$a
prompt> **echo Fa \# ou Do \#**
Fa # ou Do #
prompt> **echo *****
*****
prompt> **echo un \\ précède \$5**
un \ précède $5
prompt> **echo début **
> et fin
début et fin.
prompt>

Les apostrophes protègent toute une expression en une seule fois. Un backslash devient un caractère comme un autre qui ne peut protéger une apostrophe. L'apostrophe ne peut pas être protégée entre deux apostrophes.

prompt> **echo '#\$"&>|'**
#\$"&>|
prompt>

Les guillemets permettent de garder protéger tous les caractéres spéciaux sauf « $ », « ' » et « \ », qui conservent leur signification particulière. Les guillemets permettent aussi de garder l'unité d'une chaîne sans la fragmenter en différents mots. Enfin ils préservent les espaces, les tabulations et les retours chariot contenus dans une expression.

prompt> **var=$(ls /dev/sda1*)**
prompt> **echo $var**
/dev/sda1 /dev/sda10 /dev/sda11 /dev/sda12
prompt> **echo "$var"**
/dev/sda1
/dev/sda10
/dev/sda11
/dev/sda12
prompt>

On peut manipuler des tableaux en shell. Comme en « C » les index des tableaux sont numérotés à partir de 0.

tableau[i]=valeur # affectation
${tableau[i]} # consultation
${tableau[@]} # la liste de tous les membres du tableau
${#tableau[i]} # longueur du ième membre du tableau
${#tableau[@]} # nombre des membres du tableau

prompt> **var="valeur originale"**
prompt> **echo $var**
valeur originale
prompt> **echo ${var[0]}**
valeur originale
prompt> **var[1]="nouveau membre"**
prompt> **echo ${var[0]}**
valeur originale
prompt> **echo ${var[1]}**
nouveau membre
prompt> **echo ${#var[@]}**
2
prompt> **echo ${#var[0]}**
16
prompt> **echo ${#var[1]}**
14
prompt>

On peut évaluer une expression avec la commande « eval ».

Q5. En vous aidant du cours (« while », « read » et « if […]), écrivez ce script et lancez-le. Que fait-il ? Vous pourrez le tester avec les instructions suivant le script ou d'autres à votre choix.

prompt> **cat script**
#! /bin/bash
while true ; do
echo -n "? "
read ligne
if [-z "$ligne"] ; then
break;
fi
eval $ligne
done
prompt>

prompt> **script**
? **ls**
f1.c f2.c f3.c
? **A=123456**
? **echo $A**
123456
? **B='A = $A'**
? **echo $B**
A = $A
? **eval echo $B**
A = 123456
? **(entrée)**
prompt>

Q6. Ecrire un script « mes_carres » qui affiche les carrés des différentes valeurs entières passées en paramètre. Vous utiliserez la structure de contrôle « for ».
prompt> **mes_carres 1 2 3**
$1^2 = 1$
$2^2 = 4$
$3^2 = 9$
prompt>

prompt> **cat mes_carres**
#! /bin/bash
for i ; do echo "$i² = $((i * i))"
done
prompt>

Q7. Ecrire une fonction « ma_somme » qui affiche les somme des différentes valeurs entières passées en paramètre. Vous utiliserez la structure de contrôle « for ».
prompt> **ma_somme 1 2 3**

6
prompt>

prompt> **cat ma_somme**
function ma_somme () {
ii=0
for i ; do ii=$(($ii+$i))
done
echo $ii
}
prompt>

Q8. Ecrire une fonction gco qui prend en paramètre un fichier source « C » (par exemple « mon_prog.c ») et qui appelle le compilateur C (cc ou gcc) avec l'option " -o nom-source " (dans l'exemple « mon_prog »). On pourra mettre cette fonction dans le fichier " $HOME/.bashrc ". Vous utiliserez la fonction « basename ».
prompt> **gco mon_prog.c**
prompt> **ls**
mon_prog.c mon_prog
prompt>

gco() {
source=$1
c=$(basename $1 .c)
if [-f $1]
then cc $source -o $c
fi
}

Q9. Ecrire la procédure « fdate » qui affiche la date en français. Vous utiliserez la commande « set », l'instruction « $(cmde) » et la structure « case ».

prompt> **chmod u+x fdate**
prompt> **fdate**
Lundi 8 fevrier 2099 20:29:10
prompt>

set $(date)
case $1 in
Mon) j=Lundi;;
Tue) j=Mardi;;
Wen) j=Mercredi;;
Thu) j=Jeudi;;
Fri) j=Vendredi;;
Sat) j=Samedi;;
Sun) j=Dimanche;;
esac

```
case $2 in
Jan) m=Janvier;;
Feb) m=Fevrier;;
Mar) m=Mars;;
Apr) m=Avril;;
May) m=Mai;;
Jun) m=Juin;;
Jul) m=Juillet;;
Aug) m=Aout;;
Sep) m=Septembre;;
Oct) m=Octobre;;
Nov) m=Novembre;;
Dec) m=Decembre;;
esac
echo $j $3 $m $6 $4
```

Q10. Ecrire un script (archiv) qui renomme tous les fichiers du répertoire courant dont l'extension est « .tgz », en fichiers « .tar.gz ». Vous utiliserez la structure de contrôle « for » et l'instruction « ${%}.

prompt> **ls *.tgz**
a.tgz b.tgz

prompt> **cat archiv**

```
#! /bin/bash
for i in *.tgz ; do mv $i ${i%tgz}tar.gz ; done
```

prompt> **archiv**
prompt> **ls *.tar.gz**
a.tar.gz b.tar.g
prompt>

Q11. Ecrire la procédure « new_suff » qui permet de modifier le suffixe de un ou plusieurs fichiers. Le premier paramètre décrit l'ancien suffixe, le second décrit le nouveau et le troisième les fichiers touchés par cette modification. Vous utiliserez les structures « case » et « for », la fonction « shift », les fonctions « mv » et « basename ».

Exemple :
prompt> **ls**
f1.c f2.c f3.c g1.c
prompt> **new_suff .c .old_c f*.c**
prompt> **ls**
f1.old_c f2.old_c f3.old_c g1.c
prompt>

prompt> **cat new_suff**

```
case $# in
0 | 1 | 2 ) echo manque parametre >&2 ;;
*) old=$1
shift
new=$1
shift
for i
do
```

```
mv $i $(basename $i $old)$new
done;;
esac
# on peut aussi faire *.c) et *.cc) dans case
```
prompt>

Q12. Ecrire la procédure « comp » qui permet de compiler un fichier. Cette procédure doit contrôler l'existence du fichier source, appeler le bon compilateur (on utilisera l'extension « .c » pour le compilateur C, cc et « .cc » pour le compilateur C++, gcc) et créer un exécutable dont le nom est celui du fichier source sans extension.

prompt> **cat comp**
```
source=$1
c=$(basename $1 .c)
cc=$(basename $1 .cc)
if [ -f ${c}.c ]
then cc $source -o $c
elif [ -f ${cc}.cc ]
then gcc $source -o $cc
elif [ -f $source ]
then echo "mauvais nom "
else echo "fichier inexistant "
fi
```
prompt>