Système utilisateur
.:: Système UNIX ::

semaine 5 – les scripts shell

Petit rappel sur les Quotes

- Le simple quote « ' » : interdit toutes interprétations par le shell, y compris les caractères " et $

  > echo 'affiche "le $PATH'
  affiche "le $PATH

- Le double quote « " » : les symboles $, ` et " seront interprétés

  > echo "affiche le $PATH"
  affiche le /bin:/usr/bin ...

- Le back-quote ou anti-quote « ` » : évalue la commande placée à l'intérieur après expansion desvariables :

  > echo "affiche le `echo $PATH`"
  affiche le /bin:/usr/bin ...
  a – interprète $PATH
  b – évalue la commande echo placée entre `
  c – exécute la commande echo principale

Autres rappels

Le caractère « \ » protège les caractères spéciaux :

> echo " affiche un double quote \" "
affiche un double quote "

Les variables locales :

-> Non transmises aux processus fils
-> Fixées par l'affectation : *maVar*=1 ou **set** *maVar* 1
-> Listées par la commande **set**

Les variables exportées :

-> Transmises aux fils
-> Fixées par les commande **set** -a *maVar* 1 ou **export** *maVar*=1
-> Listées par la commande **env**

Suppression d'une variable : **unset** *maVar*

Les scripts Shell – Kézako ?

=> C'est un fichier texte contenant une liste de commandes shell
=> C'est un programme shell exécutable

=> C'est un programme qui comprend une liste de commandes s'exécutant les unes à la suite des autres comme si elles étaient saisies par l'utilisateur.

=> Comme tout langage, le shell inclus des schémas de programmation standards : l'alternative et la répétition

=> Les scripts shell ne sont pas compilés, ils sont interprétés ligne à ligne exactement comme lorsque vous tapez des commandes unes à unes.

Mon premier script SHELL

1. Un script est créé avec un éditeur : vi, emacs ...
2. Il contient comme première ligne le shell à exécuter car chaque shell a ses spécificités en terme de programmation.

   #!/bin/bash
3. Il contient la liste des commandes à exécuter par exemple :

   echo "Hello World !"
4. Avant exécution, il doit être rendu exécutable (commande chmod)
5. Enfin il peut être lancé comme n'importe quelle commande

=> Exemple complet

```
#!/bin/bash
# exemple de script shell pour commencer
echo "Hello World !"
```

Note : les commentaires sont identifiés pas un symbole « # » en début de ligne

Le passage de paramètres

=> Un script peut recevoir des paramètres depuis la ligne de commande : ils sont alors mis à la suite de la commande et séparés par des espaces

=> La chaîne de caractère affichée par l'exemple précédent peut donc être paramétrable :

```
#!/bin/bash
# exemple de programme avec paramètre
echo $1
```

Appel : ./monProg "Hello World"

=> Le paramètre est associé automatiquement par le shell à la variable "1" utilisée par $1

Le passage de paramètres

=> Les arguments sont identifiés par les variables suivantes

$1 : premier argument

$2 : Second argument

...

=> Les autres variables associées sont les suivantes :

$0 : Le nom du script

$* : Tous les arguments

$# : Le nombre des arguments

=> Une commande associée :

shift : décale les arguments d'un rang ($2 devient $1 et $1 est perdu)

La structure alternative

=> "Si *condition* alors *action1* sinon *action2* finsi" s'écrit :

```
if   expression          --> expression retournant 0 si vraie
then                         --> peut être mis sur la même ligne que
     instruction1                le if mais doit alors être précédé d'un ;
     instruction2                ex : if expression; then
     ...
else                         --> en cas de nouvelle alternative (else if)
     instruction1                il est possible d'utiliser elif exemple :
     instruction2                if expression; then
fi                                 action1
                                elif expression2; then
                                      action2
                                else
                                      action3
                                fi
```

Les expressions de tests

=> Les expressions peuvent être le résultat d'une commande comme vu en TP :

```
if egrep "chaine" fichier >/dev/null ; then ...
```

=> Les expressions peuvent être le résultat de tests notés : [expression]
-> les caractères « [» et «] » doivent être précédés et suivis d'espaces
-> retourne 0 lorsque l'expression est vraie
-> exemple de tests d'égalité : if [3 -eq 3] ; then ...

=> Variante par l'usage de la commande *test* : if test 3 -eq 3 ; then ...
-> les [ ] indiquent l'appel à la commande *test*.

Les tests

=> Test d'égalité entre 2 nombres	[*nb1* -eq *nb2*]
=> Test de différence entre 2 nombre	[*nb1* -ne *nb2*]
=> Test de supériorité stricte nb1 > nb2	[*nb1* -gt *nb2*]
=> Test de supériorité nb1 >= nb2	[*nb1* -ge *nb2*]
=> Test d'infériorité stricte nb1 < nb2	[*nb1* -lt *nb2*]
=> Test d'infériorité nb1 =< nb2	[*nb1* -le *nb2*]
=> Test d'égalité entre 2 chaînes	[*ch1* == *ch2*]
=> Test de différence entre 2 chaînes	[*ch1* != *ch2*]
=> Test de la chaîne vide	[-z *ch1*]
=> Test de la chaîne non vide	[-n *ch1*]

Les tests

Tests sur les fichiers :

=> Le fichier est un répertoire :	[-d *fic*]
=> Le fichier existe :	[-e *fic*]
=> Le fichier est un fichier simple :	[-f *fic*]
=> Le fichier existe et est exécutable :	[-x *fic*]
=> Le fichier existe et est accessible en écriture :	[-w *fic*]
=> Le fichier existe et peut être lu :	[-r *fic*]

...

Inverser un test « **!** » :

=> Le fichier n'existe pas :	[**!** -e *fic*]

Les tests

Chaîner plusieurs tests :

=> Exactement comme enchaîner des commandes ... Le résultat du test est toujours celui de la dernière commande exécutée.

=> effectuer un second test si le premier est réussi :

if [test1] && [test2] && [test3] ...

exemple : si le fichier peut être lu et écrit :

if [-r *fic*] && [-w *fic*] ; then ...

=> effectuer un second test si le premier est un echec :

if [test1] || [test2] || [test 3] ...

exemple : si le nombre est > 10 ou < 3

if [*nb* -gt 10] || [*nb* -lt 3] ; then

Exo

=> écrire un script qui reçoit en paramètre l'heure :

./monscript 09 30 45 pour 9h30 et 45 secondes

Retourne 0 si l'heure est au bon format (heure [0,23], minutes [0,59] et secondes [0,59])
Retourne 1 sinon et affiche sur la sortie erreur : "format incorrect"

=> écrire une script qui retourne 0 si le nombre passé en paramètre est pair, 1 sinon

Exo

=> Script heure :

```
#!/bin/bash
ret=0
if [ $1 -lt 0 ] || [ $1 -gt 23 ] ; then ret=1 ; fi
if [ $2 -lt 0 ] || [ $2 -gt 59 ] ; then ret=1 ; fi
if [ $3 -lt 0 ] || [ $3 -gt 59 ] ; then ret=1 ; fi
if [ $ret -eq 1 ] ; then
   echo "Format incorrect" >&2
fi
exit $ret;
```

Exo

=> Script pair/impair :

```
#!/bin/bash
nb= $(( $1 / 2 ))
nb= $(( $nb * 2 ))
if [ $nb -eq $1 ] ; then
    exit 0
else
    exit 1
fi
```

Les boucles

=> Répéter jusqu'à :

until *expression*
do
    commandes
    ...
done

i=0; **until** *[$i -eq 10]*
do
    i=$(($i + 1))
done

=> Tant que :

while *expression*
do
    commandes
    ...
done

i=0; ***while*** *[$i -lt 10]*
do
    i=$(($i + 1))
done

Les boucles

=> Parcourir une liste

for *variable* **in** *mots*
do
    commandes
    ...
done

for i **in** *0 1 2 3 4 5 6 7 8 9*
do

done

=> Si dans for « **in** *mots* » ne sont pas précisés alors c'est la liste des arguments qui est prise. exemple : le script echo

```
#!/bin/sh
for m ; do
   echo $m
done
```

Exos

> écrire un script effectuant la copie de tous les fichiers d'un répertoire passé en paramètre dans le répertoire courant.

> écrire un script comptant le temps : il compte les secondes puis les minutes puis les heures dans 3 boucles imbriquées et affiche chaque seconde la nouvelle heure. Avec 3 types de boucles.

Exos

> écrire un script effectuant la copie de tous les fichiers d'un répertoire, passé en paramètre, dans le répertoire courant :

```
#!/bin/bash
for fic in `ls $1` ; do
      if [ ! -d $1/$fic ] ;then
        cp $1/$fic .
      fi
done
```

Exos

> écrire un script comptant le temps : il compte les secondes puis les minutes puis les heures dans 3 boucles imbriquées et affiche chaque seconde la nouvelle heure :

```
#!/bin/bash
while [ 1 -eq 1 ] ; do
        for h in 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 ; do
          m = 0
          while [ $m – lt 60 ] ; do
               s=0
               until [ $s -eq 60 ] ; do
                     echo $h:$m:$s
                     s=$(( $s + 1))
                     sleep 1
              done
              m=$(( $m + 1 ))
          done
        done
done
```

Quelques commandes en plus

=> Lire une chaîne saisie au clavier : read *variable*
Charge dans la variable *variable* la chaîne saisie par l'utilisateur

```
echo -n "Votre nom :" ; read nom
echo "Votre nom est $nom"
```

Si la variable n'est pas précisée le résultat est envoyé dans REPLY

```
echo -n "Votre nom :" ; read
echo "Votre nom est $REPLY"
```

Il est possible de récupérer chaque mot dans une variable :

```
echo -n "Votre nom et prenom :" ; read nom prenom suite
echo "Vous êtres $prenom $nom"
```

Ici la variable suite contient tous les mots suivants (au cas où...)

Calculs mathématiques

=> i=`expr $i + 1` => fait appel à une commande externe : lent

=> i=$(($i+1)) => beaucoup plus rapide (mais moins portable)

=> r=`echo "scale=8000; sqrt(2)" | bc`
=> permet des calculs de type flottant

TD

=> écrire un script acceptant une liste de nombre en argument et n'affichant que ceux >= 0

=> écrire un script lisant son entrée standard et n'affichant que les lignes :
- paires
- contenant au moins 2 chiffres

en les passant en majuscule (par la commande tr a-z A-Z) et en affichant leur numéro (1 pour la première)

TD

=> écrire un script acceptant une liste de nombre en argument et n'affichant que ceux >= 0

```
#!/bin/bash
for nb
do
   if [ $nb -gt 0 ] ; then
      echo $nb
   fi
done
```

TD

=> écrire un script lisant son entrée standard et n'affichant que les lignes :
- paires
- contenant au moins 2 chiffres

en les passant en majuscule (par la commande tr a-z A-Z) et en affichant leur numéro (1 pour la première)

```
#!/bin/bash
l=1
while [ 1 -eq 1 ]; do
  read
  l0=$(( $l / 2))
  if [ $(( $l0 * 2 )) -eq $l ] ; then
    if echo $REPLY | grep "^[^0-9]*[0-9][^0-9]*[0-9][^0-9]*$" > /dev/null
    then
      echo $l : `echo $REPLY | tr a-z A-Z`
    fi
  fi
  l=$(( $l + 1 ))
done
```