Scripts SHELL 2

Matière de Max Mignotte

Modifications: Alena Tsikhanovich

Scripts SHELL, Exemple 1

- Spécification: écrire un script qui cherche une chaîne spécifiée comme premier argument (**$1**) dans les fichiers se trouvant dans l'hiérarchie des répertoires commençant par le répertoire argument 2 (**$2**)

Usage:

$0	$1	$2
findInFile.sh	**string**	**dirname**

- Commande **find point de départ options [expression]**
- **find** – cherche les fichiers dans une hiérarchie des répertoires à partir d'un point de départ selon les règles définies par une expression
- Ex.: options **-name** et **-print : find . -name "*.py" -print**
- Rechercher à partir du répertoire courant et afficher tous les noms d'entrées se terminant par la chaîne **.py**

Scripts SHELL, Exemple 1

```
#!/bin/sh
#findInFile.sh script
find $2 -type f -exec grep -l $1 {} \;
```

find args

- Va chercher tous les fichiers dans le répertoire spécifié en arg (**$2**) de type fichier (option **-type f**)

-exec commande

- Exécute la commande: envoie le nom du fichier en cours de traitement comme argument à la commande qui suit
- **{}** – espace réservé pour mettre un nom de fichier dedans

grep -l arg1 arg2

- Recherche le texte arg1 dans arg2. Les lignes qui contiennent le texte recherché s'afficheront entièrement.
- **grep -l arg1 arg2**
- Option **-l** de **grep** affiche les noms des fichiers trouvés

Scripts SHELL, Exemple 1


```
#!/bin/sh
#findInFile.sh script
find $2 -type f -exec grep -l $1 {} \;
```

```
tsikhana@arcade02:~/Scripts$ ./findInFile.sh Allo!  repl
repl/toto.txt
tsikhana@arcade02:~/Scripts$ ./findInFile.sh Allo! .
./repl/toto.txt
```

```
Allo!
Petit fichier texte
```

-exec commande

- Exécute la commande
- Il est impossible d'ajouter directement la commande à exécuter
- La commande doit être suivie de la chaîne **{}** qui sera remplacée par le nom du fichier en cours d'examen suivie de **\;**
- Si on omet l'antislash, l'interprète de commandes considérera que **;** marque la fin de la commande **find**
- Finalement, il faut mettre un espace entre les accolades et l'antislash

Scripts SHELL, Exemple 1

Solution plus efficace

- Solution précédente provoque l'exécution de **grep** pour chaque fichier trouvé
- Utilisons **xargs** à la place, afin d'invoquer **grep** sur plusieurs fichiers à la fois
- **xargs** ajoute la liste des fichiers à la fin de la commande spécifiée
- **xargs** chaîne des commandes qui prennent des arguments comme entrée; avec **xargs**, si vous donnez une commande tout ce qui est passé à l'entrée standard sera passé en une fois comme argument de la commande

```
#!/bin/sh

#findInFile.sh script

find $2 -type f -print | xargs grep $1
```

```
tsikhana@arcade02:~/Scripts$ ./findInFile1.sh Allo! .
./repl/toto.txt:Allo!
```

```
Allo!
Petit fichier texte
```

Scripts SHELL, Conditions-Tests


```
if [ expression ]
then
   commande 1
   commande 2
fi
```

```
if [ expression ]; then
   commande 1
   commande 2
fi
```

```
if [ $day == "Monday" ]
then
echo First day of the week
fi
```

Conditions sur les fichiers/répertoires

Syntaxe	Correspondance
-d nom	nom existe et c'est un répertoire
-f nom	nom existe et c'est un fichier
-r nom	nom existe et l'objet (fichier ou rép.) est lisible
-w nom	nom existe et l'objet (fichier ou rép.) peut être écrit
-x nom	nom existe et l'objet (fichier ou rép.) est exécutable

Scripts SHELL, Conditions-Tests

Conditions sur les entiers

Syntaxe	Correspondance
`n1 eq n2`	`n1 == n2 ?`
`n1 -ne n2`	`n1 ≠ n2 ?`
`n1 -gt n2`	`n1 > n2 ?`
`n1 -ge n2`	`n1 ≥ n2 ?`
`n1 -lt n2`	`n1 < n2 ?`
`n1 -le n2`	`n1 ≤ n2 ?`

Scripts SHELL, Conditions-Tests

Conditions sur les chaînes de caractères

Syntaxe	Correspondance
`string`	`string n'est pas nulle`
`-n string`	`string n'est pas de longueur nulle`
`-z string`	`string est de longueur nulle`
`s1 == s2`	`string s1 identique au s2 ?`
`s1 != s2`	`string s1 différente de s2 ?`

Scripts SHELL, Conditions-Tests

Conditions composées

Syntaxe	Correspondance
(expr)	Vrai si l'expression entre parenthèses est vrai
!expr	Vrai si l'expression est fausse
expr1 -a expr2	Vrai si les 2 expressions sont Vrai
expr1 -o expr2	Vrai si l'une des 2 expressions est vrai

Scripts SHELL, Conditions-Tests

```
if [ $# -lt 1 ]
```

Si le nb d'arguments est < 1

```
if [ -f $1 ]
```

Si l'argument 1 est le nom d'un fichier

```
if [ -d /tmp ]
```

Si le répertoire **/tmp** existe

Scripts SHELL, Conditions-Tests

```
if [ $x ]
```

Si la valeur de la variable x est non nulle ou définie

```
if [ $x!=error ]
```

Si la valeur de la variable x n'est pas error

```
if [ $height -gt 64 -a $weight -ge 120]
```

Si ((height > 64)&&(weight>=120)

```
if [ $day==Sat -o $day==Sun ]
```

Si la valeur de day est Sat ou Sun

Scripts SHELL, Conditions-Tests

```
if [ !\(-f $1 -a -r $1\) ]
```

Si le premier argument n'est pas un fichier lisible

```
if [ : ]
```

Toujours Vrai

```
if [ file1 -nt file 2 ]
```

Si file1 est plus recent (newer than) que file 2

Scripts SHELL, Conditions-Tests

Utilisation: **`mylist.sh dirname`**

Le script affiche le contenu d'un répertoire passé en paramètre

```
if [ expression1 ]; then
   commande 1
elif [ expression2 ]; then
   commande 2
else
   commande 3
fi
```

```
#!/bin/sh
#mylist.sh script
if [ $# -ne 1 ]; then
   echo Mauvais nombre d'arguments
   exit
fi
if [ -d $1 ]; then
   cat $1
else
  echo $1 n'est pas un répertoire de
          fichiers
fi
```

Scripts SHELL, Conditions-Tests

Utilisation: **octaldisplay.sh nombre (octal)**

Le script interprète un nombre octal représentant les droits d'un répertoire ou un fichier

```
#!/bin/sh
#octaldisplay.sh script
if [ $# -ne 1 ]; then
 echo Mauvais nombre d'arguments
 exit
fi
case $1 in
0) echo Pas de permission;;
1) echo Permission exécuté;;
2) echo Permission écriture;;
3) echo Permission exécuté et écriture;;
4) echo Permission lecture;;
5) echo Permission lecture et exécuté;;
6) echo Permission lecture et écriture;;
7) echo Permission lecture, écriture et exécuté;;
*) echo Ce n'est pas un nombre octal d'1 digit;;
esac
```

```
case value in
pattern1)
   cmd1;;
pattern2)
   cmd2;;
…
…
esac
```

Scripts SHELL, Boucles

```
for var in list
do
    cmds
    …
done
```

```
for name in Harry Susan Bob Jane
do
    echo Hello $name
done
```

```
> Hello Harry
> Hello Susan
> Hello Bob
> Hello Jane
```

Scripts SHELL, Boucles, Exemples

```
for var in list
do
    cmds
    …
done
```

```
#!/bin/sh
#listc.sh script
for name in *.c
do
    echo $name
done
```

```
#!/bin/sh
#pargc.sh script
for arg in $
do
    echo $arg
done
```

Scripts SHELL, Boucles

```
while [ expression ]
do
    cmds
    …
done
```

```
#!/bin/sh
#count.sh script
i=1
while [ $i -le 10 ]; do
    echo $i
    let i=$i + 1
done
```

```
#!/bin/sh
#countdown.sh script
a=100
while [ $a -gt 0 ]; do
    echo -n "$a"
    a=$(($a-1))
done
echo "Booum!"
```

```
#!/bin/sh
#countdownarg.sh script
while [ $# -gt 0 ]; do
    echo $1
    shift
done
```

Scripts SHELL, Boucles


```
until [ expression ]
do
    cmds
    …
done
```

```
#!/bin/sh
#count.sh script
for (( i=0; i<100; i++))
do
    echo $i
done
```

```
for (( init; cond. d'arrêt; incr.))
do
    cmds
    …
done
```

Scripts SHELL, **break**


```
#!/bin/sh
# option.sh script
YES=1 Ot=0 Oa=0 Ol=0
for arg in $
do
    case $arg in
        -t) Ot=$YES;;
        -a) Oa=$YES;;
        -l) Ol=$YES;;
         *) echo argument invalide
         break
    esac
done
echo option t-a-l : $Ot $Oa $Ol
```

Utilisation: **option.sh option (-l ou –a ou -t)**

Le script analyse 3 options de la commande **ls**:

-l, format long de l'affichage,
-a, n'est pas ignoré les entrées commençantes par « . »

-t, les entrées sont triées selon leur date de dernière utilisation (jour et temps)

Scripts SHELL, boucles


```
#!/bin/sh
# loop.sh script
i=1
while [ : ]
do
   echo $i
   let i=$i + 1
   if [ $i -gt 10 ]; then
            break
   fi
done
```

Scripts SHELL, boucles

```
#!/bin/sh
# convert.sh script
for filename in *.jpg *.gif *.tif
do
   ppmfile=${filename%.*}.ppm
   echo conversion de $filename en $ppmfile
   convert $filename $ppmfile
done
```

${param%pattern}

- Efface le pattern de fin à la string param

${param##pattern}

- Efface le pattern de début à la string param

Scripts SHELL, variables avancées

VARIABLES AVANCÉES

Syntaxe de la substitution de variables avancées

Syntaxe	Correspondance
`${var:-val}`	Si var existe on l'utilise sinon utilise val
`${var:=val}`	Si var existe on l'utilise sinon var=val
`${var:?message}`	Si var existe on l'utilise afficher le message est exit

Scripts SHELL, variables avancées

```
echo je suis ${moi:-personne}   > Je suis personne
moi=Max
echo je suis ${moi:-personne}   > Je suis max
echo $today does not exist      > does not exist
echo ${today:-`date`}           > Wed Dec 17 16:52:10 EST 2003
```

Utilisation: **`findfile.sh string dirName`** (ou 1 seul paramètre)

Le script cherche les fichiers contenant **`string`** à partir de **`dirName`** ou à partir du répertoire de travail (`.`) si arg2 n'est pas fourni

```
#!/bin/sh
#findfile.sh script
find ${2:-.} -type f -exec grep $1 {} \;
```

Scripts SHELL, exemples (1)


```
#!/bin/sh
# devine.sh script
sec=`date + %s`
let num=$sec%10
if [ $num -eq 0 ]; then
   num=10
fi
while [ : ]
do
   echo Devine un nombre entre 1 et 10
   read x
   if [ $x -eq $num ]; then
      echo Gagné
      break
   else
      echo Désolé, essaie encore!
   fi
done
```

```
#!/bin/sh
# nbfiles.sh script
nbfiles=`ls *|wc -l`
echo $nbfiles
```

Scripts SHELL, exemples (2)


```
#!/bin/sh
# edg.sh script
ed $3 <<%
g/$1/s//$2/g
w
%
```

- Utilisation: edg.sh string1 string2 file1
- Substitue toute string1 dans file1 par string2

ed – éditeur de texte orienté ligne
Commandes: **g/string/** - imprime toutes les lignes contenantes string ($1)
/s/ – "substitute" du texte résultat de **g/$1/** par le texte **/$2/g**
w – écrire
% - end of file dans ed

```
#!/bin/sh
# rmexec.sh script
for x in *
do
   [ -x $x -a -f $x.c ] && echo $x; done | xargs rm -f
```

Calcul sur les flottants

- En natif, Bash ne propose que des fonctionnalités de calcul limitées (sur les entiers)
- **bc** permet des calculs plus complexes, avec gestion des décimales
- Syntaxe:

variable=$echo "expresion" | bc ou
variable=`echo expression | bc`

```
$ echo "1/3" |bc -l
.33333333333333333333
```

```
$ echo "sqrt(2)" |bc -l
1.41421356237309504880
```

- Option **-l** définie la librairie standard de fonctions mathématiques