Chapitre 8
Programmation Shell

M1

Programmation Shell

- Interpréteur de commande ou langage de programmation ?
- Tout interpréteur de commande permet de définir des séquences de commandes réutilisables
- Le shell offre un maximum de puissance et de souplesse
- Un fichier contenant des commandes est appelé un script (ou par extension un shell)
- Un script peut être assimilé à un programme

Programmation Shell

- Langage interprété
  - Pas de compilation
- Un script est écrit pour fonctionner sous un type précis de shell
- Mais il peut invoquer un script fonctionnant sous un autre shell
- Un script commence par le chemin d'accès au shell qui doit interpréter le script
  - #!/bin/xxx
- Un shell doit être exécutable
  - Obligation de changer les droits

Eléments d’un shell

Un shell contient :

- Des variables
  - mavariable=bonjour # assignation
  - echo $mavariable #référence
- Des instructions
  - Toutes les commandes Linux;
  - Invocation de programmes exécutables avec passage de paramètres;
  - Assignation et référencement de variables;
  - Instructions conditionnelles et itératives;
  - Instructions d’entrée/sortie.

Différents modes d'exécution d'un script

- Mode trace : trace le déroulement du script
  - Recopie sur la sortie standard chaque ligne telle qu'elle est
- interprétée
  - sh –x nomshell
- Mode verbose : commente le déroulement du script
  - Recopie sur la sortie standard chaque ligne avant interprétation
  - sh –v nomshell
- Possibilité de spécifier le mode dans le shell
  - set –x ou set -v

Passage de paramètres

- Paramètres = arguments d'un programme
  - Ex : ls ./toto → ./toto est un paramètre de ls
- Un paramètre s'utilise comme une variable
- Les paramètres sont numéroté séquentiellement de 1 à 9
  - Ex : $1 correspond à ./toto
- shift permet de supprimer le premier paramètre
  - Décalage des paramètres si plus de 9

Passage de paramètres

Variables spéciales :

- $0 le nom du script invoqué
- $# Le nombre de paramètres passés en arguments (variable chaîne de caractères, non numérique)
- $* La liste des paramètres passés en arguments
- $? Le code de retour de la dernière commande exécutée
- $$ Le numéro du processus du shell
- Exemple
  - script affichParam :
  - echo $0 a ete appele avec $# parametres qui sont : $*
  - Résultat (affichParam a b c d):
  - affichParam a ete appele avec 4 parametres qui sont a b c d

Instructions d’entrée/sortie

- Commande **read**
  - Lecture entrée standard
  - Affecte chaque mot lu à une variable
- Commande **echo**
  - Ecriture sortie standard
- Exemple :
  - echo « Nom du fichier à afficher : »
  - read nomfic
  - echo $nomfic

Structures de contrôle : instructions conditionnelles – le cas du if

- sélection avec une alternative : if … then … fi

  if commande
  then commandesv
  fi

- Sélection à deux alternatives : if … then … else …fi

  if commande
  then commandes1
  else commandes2
  fi

Structures de contrôle : instructions conditionnelles – le cas du if

- Sélection à n alternative

  if commande1
  then commandes1
  elif commande2
  then commandes2
  …
  else
  commandes0
  fi

Structures de contrôle : instructions conditionnelles – le cas du if

Exemple :

```
if [ $# eq 1 ]
then
echo « un paramètre »
elif [ $# eq 2 ]
then
echo « deux paramètres »
else
echo « plus de deux paramètres »
fi
```

Structure de contrôle : instructions conditionnelles – le cas du test

- Permet de :
  - Reconnaître les caractéristiques d'un fichier;
  - Comparer des chaînes de caractères;
  - Comparer algébriquement des nombres.
  - Deux syntaxes différentes :
- Syntaxes :
  - test expression
  - [expression]
- Répond à l'interrogation formulée dans expression par :
  - Un code nul en cas de réponse positive;
  - Un code différent de zéro sinon.

Structure de contrôle : instructions conditionnelles – le cas du test

- Exemple d'utilisation :

  if [expression]
  then commandes
  fi

- Remarques
  - Il faut obligatoirement un espace entre les [] et l'expression
  - Si then est sur la même ligne, il faut mettre un ; entre le crochet
  - fermant et then.
- Option du test
  - **-d** nom vrai si le répertoire existe
  - **-f** nom vrai si le fichier existe
  - -s nom vrai si le fichier existe et est non vide

Structure de contrôle : instructions conditionnelles – le cas du test

- -r nom vrai si fichier existe et est accessible en lect.
- -w nom vrai si fichier existe et est accessible en écrit.
- -x nom vrai si le fichier existe et est exécutable
- -z chaîne vrai si la chaîne de caractères est vide
- -n chaîne vrai si la chaîne de caractères est non vide
- C1 = C2 vrai si C1 et C2 sont identiques
- C1 != C2 vrai si C1 et C2 sont différentes
- N1 -eq N2 vrai si les entiers N1 et N2 sont égaux
- N1 -ne N2 vrai si N1 est différent de N2
- N1 -gt N2 vrai si N1 est supérieur à N2
- N1 -ge N2 vrai si N1 est supérieur ou égal à N2
- N1 -lt N2 vrai si N1 est inférieur à N2
- N1 -le N2 vrai si N1 est inférieur ou égal à N2

Structure de contrôle : instructions conditionnelles – le cas du test

- ! Opérateur logique de négation
- -a Opérateur logique traduisant le « et »
- -o Opérateur logique traduisant le « ou »

- **Exemple** :

```
if [ $1 –lt $2 ] ; then
echo $1 plus petit que $2
elif [ $1 –gt $2 ] ; then
echo $1 plus grand que $2
else
echo $1 égal à $2
fi
```

Structure de contrôle : instructions conditionnelles – le cas du case

- Case permet la gestion des évaluations de type choix multiple
- Le shell recherche parmi les différentes chaînes de caractères motif1, motif2, … motifn la première qui correspond à chaine et il exécute les commandes correspondantes.
- Syntaxe :

```
case chaine in
motif1) commandes 1 ;;
motif2) commandes 2 ;;
motif3) commandes 3 ;;
…
motifn) commandes n ;;
esac
```

Structure de contrôle : instructions conditionnelles – le cas du case

- chaine peut prendre les formes suivantes :
  - Un chiffre, une lettre ou un mot
  - Des caractères spéciaux du shell
  - Une combinaison des éléments précédents
- chaine peut être :
  - Lue ou passée en paramètre
  - le résultat d'une commande exécutée avec l'opérateur backquote `` ou $().
- Exemple :

```
case $# in
0) echo $0 sans argument ;;
1) echo $0 possède un argument ;;
*) echo $0 possède plus d'un argument;;
esac
```

Itérations bornées – la boucle for

- **Forme 1** : variable prend les valeurs énumérées

  for variable in chaine1 chaine2 … chainen
  do
  commandes
  done

- **Exemple**

  for i in un deux
  trois
  do
  echo $i
  done

bash>echofor1
un
deux
trois
bash>

Itérations bornées – la boucle for

- **Forme 2** : variable prend ses valeurs dans la liste des paramètres du script

```
for variable
do
commandes
done
```

- **Exemple** :

```
for i
do
echo $i
Done
```

```
bash>echofor2 le systeme linux
le
systeme
linux
bash>
```

Itérations bornées – la boucle for

- **Forme 3** : variable prend ses valeurs dans la liste des fichiers du répertoire courant
  - for variable in *
  - do
  - commandes
  - Done
- **Exemple** :

```
for i in *
do
echo $i
Done
```

```
bash>echofor3
fic1
fic2
fic3
bash>
```

Itérations non bornées – les boucles until

- Exécution d'une série de commande jusqu'à ce qu'une commande spécifique soit vérifiée :

```
until commande1
do commandes2
done
```

- Exemple :

```
until [ $1 = fin ];do
echo $1
shift
Done
```

```
Bash>scunt 1 2 3 fin 5 6
1
2
3
Bash>
```

Arithmétique entière sur des variables

- Le Bourne-shell ne permet pas la définition de variables numériques.
  - Les opérateurs –eq –ne –lt –le –gt –ge existent
  - Toutes les variables écrites en Bourne-shell sont des chaînes de caractères
  - Si elles représentent des valeurs numériques, les opérateurs s'y appliquent
- En Bash, les opérations arithmétiques sur variables sont plus faciles grâce à 2 mécanismes :
  - Évaluation arithmétique :
    - $((variable1 opérateur variable2))
  - Test arithmétique :
    - if ((variable1 opérateur variable2))

L’arithmétique entière : ((…))

- L’arithmétique entière est peu commode en Bourne-shell
- Facilité en Bash
  - Reconnaissance des < > ( ) *
  - Priorités de calcul habituelles
- Exemple:

```
declare –i n1 n2 n3 c
n1=17
n2=3
n3=$((17/3))                #division
n3=$((17%3))                #reste de la division entière
n1=$((n2*(n1+27)-5))
```

Le test arithmétique if((...))

- Ce test permet des branchements conditionnels sur calculs arithmétiques
- Exemple
- Bourne-Shell
  - if [$x –gt 1000] ; then
- Bash
- if ((x>1000)) ; then

Variables prédéfinies

- PPID : numéro du processus père
- PWD : répertoire de travail
- RANDOM : un nombre aléatoire
- SECONDS : temps écoulé depuis le lancement du shell
- ! : numéro du dernier processus alloué en arrière-plan
- _ : dernier mot de la dernière commande exécutée

Définition de variables : la commande declare

- Commande générique de déclaration de variable
  declare [+/-option] [-p] [variable[=valeur]]
- -p affiche les variables et leur valeur
- -f ou –F affiche les fonctions avec ou sans leur définition
- -a variable est de type tableau
- -i variable est de type numérique entier
- -r variable en lecture seule
- -x variable exportée (placée dans l'environnement)
- Exemple
  declare –a tab=(1 2 3)

Définition de variables : la commande declare

Exemples

```
Bash> declare –a vtt=(sunn cannondale giant scott)
Bash> echo $ vtt
sunn
Bash> echo ${vtt[2]}
giant
Bash> echo ${vtt[*]}
sunn cannondale giant scott
Bash> echo ${#vtt[1]}
10
Bash> echo ${#vtt[*]}
4
Bash> unset vtt
```

Extension de la commande test

- Test a un pouvoir d'expression plus riche
- L'opérateur == peut remplacer = (préférable)
- Opérateurs sur fichier
  - -a fich fich existe
  - -L fich fich est un lien symbolique
  - fich1 –ef fich2 fich1 est un lien sur fich2
  - fich1 –nt fich2 fich1 est plus récent que fich2
  - fich1 –ot fich2 fich1 est plus ancien que fich2

Extension de la commande test

- Le test [[…]]
  - Permet la comparaison de chaînes de caractères avec les opérateurs == et !=
  - Entre simple crochet, comparaison de chaîne à chaîne
  - Entre double crochet, comparaison de chaîne à motif
- Exemple
  - if [$TERM == vt100] correct
  - if [$TERM == vt*] KO
  - if [[$TERM == vt*]] correct
  - if [$TERM == vt100 –o $TERM == vt220]

L’écriture de script

- Parenthésage
  - (commande1 ; commande2 ; commande3)
    - Les commandes sont exécutées sous un sous-shell
  - { commande1 ; commande2 ; commande3 ; }
    - Les commandes sont exécutées dans le shell courant
- Substitution
  - $variable ou ${variable} #la 2ème est préférable
  - ${#variable} longueur de la variable
  - ${#variable[*]} nombre d’élément du tableau
  - ${variable:-chaine} affectation sans évaluation

  ex : TERM=${TERM:-vt100}
  - ${variable:=chaine} affectation avec évaluation
  - ${variable:?chaine} affectation ou envoie sortie std.
  - ${variable:+chaine} chaine ou vide

L’écriture de script

- Substitution de commande
- ${commande}
- Imbrication de commandes
- Exemple :
- Bash> cp $(find $(echo $PATH | tr ‘:` ‘ ‘) \ –type –f –name ‘*log* -print) /tmp/backup

Extensions Bash dans la génération de nom

- **Syntaxe :**

*(motif) 0 ou 1 ou plusieurs occurrences de motif

+(motif) 1 ou plusieurs occurrences de motif

?(motif) 0 ou 1 occurrence de motif

@(motif1|motif2) motif1 ou motif2

!(motif) tout sauf motif

Exemple :

rm !(*.c|*.h|[Mm]akefile*|README*)

if [$fic = @(*.c|*.h|[Mm]akefile*|README*)]

Fin