UNIX (TP3)

Samba Ndojh NDIAYE
IUT Lyon 1
Laboratoire d'InfoRmatique en Image et Systèmes d'information
LIRIS UMR 5205 CNRS/Université Claude Bernard Lyon 1
samba-ndojh.ndiaye@univ-lyon1.fr

Sommaire

Les scripts

Shell

- Interpréteur de commandes
- Bourne Shell : syntaxe proche des premiers shells unix (/bin/sh)
- C Shell : syntaxe ≪ proche ≫ de celle du C (/bin/csh)

Sous Linux

- Bourne Again Shell : Bourne Shell augmenté de toutes les fonctionnalités du C Shell
- Tcsh : extension du C shell d'origine

Les scripts

Commandes internes

- Commandes intégrées au Shell
- Traitées directement par le Shell sans créer de processus
- Selon le type de Shell, les commandes internes peuvent être différentes

Commandes externes

- Fichiers exécutables
- Emplacement spécifié dans une variable d'environnement nommée PATH
- PATH est modifiable : vous pouvez créer vos propres commandes et y accéder depuis le Shell

Les scripts

Un script

- fichier texte exécutable
- contient les commandes à exécuter
- nécessité de donner des droits d'exécution avec chmod

Commande

- Si la commande est interne : exécution au sein du Shell courant (pas de création de processus)
- Sinon : le Shell recherche dans les répertoires situés dans PATH pour trouver cette commande (un/des nouveaux processus sont créés)

Sommaire

Les variables

Généralités

- Création affectation d'une variable dans un Shell
- Affectation (bash) avec la commande "=" (variable=valeur)
- La commande read permet de lire au clavier la valeur à affecter à une variable (read a)
- On peut affecter plusieurs variables à la fois (read b c d)
- La valeur d'une variable est désignée par son nom précédé du caractère '$'
- Une variable Shell n'est pas typée
- Affichage de la valeur d'une variable avec echo (echo $test)
- Une variable est locale au Shell où elle a été créée

Les variables

Variables

Il est possible de récupérer le résultat d'une commande dans une variable avec le caractère `

Exemples

- datedujour=`date` ; echo $datedujour
- datedujour=la date du jour est `date` ; echo $datedujour
- datedujour=' la date du jour est `date` ' ; echo $datedujour
- datedujour=" la date du jour est `date` " ; echo $datedujour

Les variables

Variables définies

- la commande set : permet de voir la liste des variables actuellement définies
- la commande unset : permet de détruire une variable

Variables automatiques (prédéfinies par le système)

- $PATH donne le chemin d'accès par défaut aux exécutables
- $_ : la dernière commande
- $PWD : le chemin absolu du répertoire courant

Sommaire

Paramètres de scripts

Paramètres

- Une commande exécutable (script ou autre) peut avoir des arguments
- Pour un script, les arguments sont accessibles sous la forme de variables nommées $0, $1, $2 ...
- $0 correspond à la commande elle-même, $1 au premier argument, $2 au deuxième ...
- S'il y a plus de 9 arguments, il faut utiliser la segmentation avec les accolades : ${10} ...
- $* l'ensemble des paramètres sous la forme d'une seule chaîne de caractères
- $@ l'ensemble des paramètres sous la forme d'une liste itérable
- $# le nombre de paramètres

Sommaire

Opérateurs

et

cmd1 && cmd2 : **cmd2** sera exécutée sauf si la **cmd1** se termine sur un code d'erreur

ou

cmd1 || **cmd2** : **cmd2** sera exécutée uniquement si la **cmd1** se termine sur un code d'erreur

Boucle for

for

for var **in** liste
do
commandes
done

- liste est un itérable
- si aucune liste n'est fournie, celle-ci est constituée par la liste des arguments (du script)
- liste peut être contenue dans une variable
- il est possible, en utilisant ` d'utiliser le résultat d'une commande comme liste

Structure if

if

if commande1
then commande2
else commande3
fi

- La commande1 est évaluée par rapport à son code retour

Structure if

if

if commande1
then commande2
elif commande3
then commande4
elif commande5
...
else commandek
fi

Structure case

case

case chaine **in**
motif1) commande1 ; ;
motif2) commande2 ; ;
...
motifn) commanden ; ;
esac

- permet de sélectionner une action en fonction de certains motifs
- Un motif peut être une expression régulière
- Le symbole "|" permet de simuler le "ou" pour un motif

Structure while

while

while commande1
do commande2
done

- permet de répéter commande2 tant que le code retour de commande1 est nul

Structure until

until

until commande1
do commande2
done

- permet de répéter commande2 tant que le code retour de commande1 est non nul

Commande test

test

évalue une expression et retourne vrai ou faux

test expression ou **[expression]**

- **test -w** fichier : vrai si fichier existe et est autorisé en écriture
- **test -r** fichier : idem en lecture
- **test -x** fichier : idem en exécution
- **test -d** fichier : vrai si fichier existe et est un répertoire
- **test -f** fichier : vrai si fichier existe et n'est pas un répertoire
- **test -s** fichier : vrai si fichier existe et n'a pas une taille nulle
- **test -z s1** : vrai si la chaîne s1 est vide
- **test -n s1** : vrai si la chaîne s1 n'est pas vide

Commande test

test

évalue une expression et retourne vrai ou faux

test expression ou **[expression]**

- **test s1 = s2** : vrai si les chaînes s1 et s2 sont identiques
- **test s1 != s2** : vrai si les chaînes s1 et s2 sont différentes
- **test n1 -eq n2** : vrai si les nombres $n1$ et $n2$ sont égaux
- **test n1 -ne n2** : vrai si $n1 \neq n2$
- **test n1 -gt n2** : vrai si $n1 > n2$
- **test n1 -lt n2** : vrai si $n1 < n2$
- **test n1 -ge n2** : vrai si $n1 \geq n2$
- **test n1 -le n2** : vrai si $n1 \leq n2$

Commande expr

expr

évalue une expression et retourne le résultat

- **expr** $e1 + e2$
- **expr** $e1 - e2$
- **expr** $e1 \setminus * e2$
- **expr** $e1/e2$
- **expr** $e1\%e2$

Opérateurs

opérateurs logiques

- ! : négation
- -a : ET logique
- -o : OU logique
- () : parenthèses
- Les parenthèses doivent être protégées (“ \ ” ou “ { } ”) pour éviter d'être interprétées par le Shell

Sommaire

Fonctions

Fonction

```
Fonction()
{
Commande1
Commande2
...
}
```

- L'instruction "return n" permet de quitter la fonction en spécifiant le statut n
- exit ?
- à l'image d'un script, il est possible d'appeler une fonction (nom de la fonction sans les parenthèses) en rajoutant des arguments qui sont accessibles par $1, $2...