Shell



Le "shell" (interpréteur de commandes) est un programme qui accepte les commandes à partir du clavier et demande au système d'exploitation de les exécuter. Au début avec UNIX, c'était la seule interface disponible. Le shell sert d'intermédiaire entre l'utilisateur et le système d'exploitation.

Utilisateurs

Le SHELL

LINUX

Aujourd'hui, nous avons l'environnement graphique GUI "Graphical User Interfaces" et l'environnement en ligne de commandes CLI "Command Line Interface".

Script

Un script est un fichier texte qui contient une suite de commandes. Un script est interprété par un shell de la même façon que sur l'invite de commandes.

Shell d'un usager

Lors de la création d’un compte utilisateur, un shell lui est associé. Dans le fichier **/etc/passwd**, le dernier champ contient le nom du shell qui est associé à l'utilisateur, pour CentOS (/bin/bash)

Pour connaître le shell utilisé par un utilisateur :

cat /etc/passwd | grep *utilisateur*

Pour afficher le shell courant :

echo $SHELL

Enchaînement inconditionnel des commandes

En général, on place une commande par ligne que ce soit en ligne de commande

ou dans un script. Le point-virgule **;** joue le rôle de séparateur inconditionnel. On peut alors mettre plus d’une commandes sur une même ligne (ex: clear; ls -l)

Enchaînement conditionnel des commandes

Les séparateurs **&&** (AND) et || (OR) sur la ligne de commande sont des

séparateurs qui jouent les rôles d'opérateurs conditionnels.

|  |  |
| --- | --- |
| **&&** | La commande qui suit le symbole **&&** n'est exécutée que si l'exécution de la commande précédente a réussi. **exemple: mkdir /tmp/toto/ && cd /tmp/toto/** |
| **||** | La commande qui suit le symbole **||** n'est exécutée que si l'exécution de la commande précédente n'a pas réussi.**exemple: cd /toto/ || cd /tmp/toto/** |

Redirections des entrées-sorties

Par défaut les canaux d'entrées et de sorties communiquent avec le clavier et l'écran.

|  |  |
| --- | --- |
| **<** | redirection de l'entrée standard à partir d'un fichier**exemple: more < /etc/passwd** |
| **>** | redirection de la sortie standard en direction d'un fichier. Le fichier est créé et écrase sans avertissement le fichier existant qui porte le même nom**exemple: ls -l /tmp/ > /tmp/info.txt** |
| **>>** | redirection de la sortie standard à la fin du fichier s'il existe déjà**exemple: ls -l /root/ >> /tmp/info.txt** |
| **|** | enchaînement de commandes (appelé aussi tube ou pipe).La sortie de la commande de gauche est envoyée en entrée à la commande de droite**exemple: cd /toto/ | cd /tmp/toto/** |

Redirections des messages d'erreurs

|  |  |
| --- | --- |
| **2>** | redirige le message d'erreur dans un nouveau fichiersi le fichier existe déjà, il écrase sans avertissement l'ancien contenu**exemple: cd /toto/ 2> /tmp/erreur.txt** |
| **2>>** | reridige le message d’erreur à la fin du fichierexemple: cd /toto/ 2> /tmp/erreur.txt |
| **2 > &1** | Redirige les erreurs dans la sortie standard |

Création d'un script

La première ligne d'un script BASH doit être: **#!/bin/bash**

|  |
| --- |
| **#!/bin/bash***# script Salut**# affiche le message Salut à l'utilisateur qui l'a lance**# la variable d'environnement $USER contient le nom de login***echo Salut $USER** |

Les commentaires dans un script

Une ligne de commentaire commence par un # sauf la première ligne qui n'est pas considéré comme un commentaire (#!/bin/bash).

Forcer la fin de l'exécution d'un script

* exit

Exécution d'un script

Avant d'exécuter le script pour la première fois, il faut attribuer les autorisations (X):

* **chmod a+x *nomduscript***

**chemin relatif (répertoire courant)**

* **./*nomduscript***
* **bash *nomduscript***

**chemin absolu**

* **/tmp/*nomduscript***
* **bash /tmp/*nomduscript***

On peut modifier le fichier **/root/.bash\_profile** pour modifier la variable $PATH afin d'y ajouter un répertoire qui contient nos scripts.

* PATH=$PATH:$HOME/bin**:/tmp**

Mais étant donné que la variable d'environnement PATH inclus automatiquement un dossier bin pour l'utilisateur: /root/bin ou /home/***utilisateur***/bin

Débogage

Il est possible de faire la trace d'un script avec la syntaxe suivante:

* **bash -x *nom\_du\_script***

Pour aider à la mise au point d'un script, on peut insérer des lignes temporaires:

* **echo $var** pour afficher la valeur de la variable
* **exit 1** pour forcer l'arrêt du script à cet endroit

Commande pour effectuer la trace d'un script

**set -x active la trace du script**

**set +x désactive la trace du script**

|  |
| --- |
| **Code d'un script avec trace****#!/bin/bash****set -x****a=5****echo $a****set +x** |
| **Résultat du script avec trace****+ a=5****echo 5****5****+ set +x** |

Entrées-sorties

**echo** 🡺 affiche un message à la console

echo change de ligne

echo "message"

echo –n "message" ne change pas de ligne

echo $# affiche le nombre de paramètres lors de l’appel

echo $BASH affiche le contenu d’une variable d’environnement

echo $LINENO affiche le numéro de ligne courante dans le script

echo -e "Un saut de ligne \n" permet l'utilisation des métacaractères

* \n saut ligne
* \b retour arrière
* \t tabulation
* \a alarme
* \c fin sans saut de ligne

**printf** 🡺 affiche un message formatté à la console (voir man printf)

printf "Votre nom est %s et vous avez %d ans\n" "$USER" 20

**read** 🡺 permet d'affecter une valeur lors de la saisie au clavier

echo "Entrer votre nom"

read nom

echo "Bonjour: " $nom

Déclaration des variables

variable=valeur

**Le signe = NE DOIT PAS être entouré d'espace(s)**

On peut initialiser une variable à une chaîne vide:

* chaine\_vide=

Il est possible de déclarer une variable avec le mot clé "declare".

declare -i a=12 a est un entier

declare -r b=5 b est en lecture seulement

declare -ri c=10 c est un entier en lecture seulement

declare -a ordinateur=(cpu memoire disque boitier)

* echo ${ordinateur} affiche le premier élément de la liste
* echo ${ordinateur[0]} affiche le premier élément de la liste
* echo ${ordinateur[1]} affiche le deuxième élément de la liste
* echo ${ordinateur[2]} affiche ...
* echo ${ordinateur[3]} affiche ...

Assigner une valeur à une variable

* nom=jean
* nom='richard jean'
* noyau=`uname –a`

Lire la valeur d'une variable

Pour lire la valeur d'une variable il faut placer un $ devant le nom de la variable.

* echo $ORDINATEUR affiche le contenu de la variable à l'écran

Les variables exportées

Si on déclare une variable avec "export" dans le shell, il sera possible de l'utiliser à l'intérieur d'un script.

* nom=richard déclaration de la variable dans le shell
* export nom il sera possible d'utiliser la variable dans un script
* unset nom pour éliminer la variable dans le shell

**Utilisation des guillemets (" 🡺 alt-34)**

* servent à délimiter une chaîne de caractères en protégeant les espaces et la plupart des méta-caractères qu'elle contient
* le shell va interpréter les variables contenues à l'intérieures des guillemets

DIR=repertoire

mkdir "Ce $DIR avec des espaces"

**Utilisation des apostrophes (' 🡺 alt-39)**

* servent à délimiter une chaîne de caractères en protégeant l'ensemble des caractères qu'elle contient. Aucun de ceux-ci ne sera interprété.

DIR=repertoire

mkdir 'Ce $DIR avec des espaces'

**Utilisation des apostrophes inversées (` 🡺 alt-96)**

* servent à substituer le résultat d'une commande, la commande que l'on place entre les apostrophes inversées sera exécutée et le résultat sera envoyé dans la variable

noyau=`uname –a`

echo $noyau

Pour effectuer des calculs

Le contenu des variables est considéré comme étant alphanumérique.

note: ne pas laissé d'espace entre la variable et l'opérateur

NB=5

RES=$NB\*2

echo $RES affiche 5\*2

NB=5

let RES=$NB\*2

echo $RES affiche 10

echo $((30+2)) affiche 32

echo $((30+2\*10/4)) affiche 35

echo $(( (30+2) \* ((10-7) /4 ))) affiche 0

Bash est inadapté aux calculs. Il est préférable d’utiliser la commande « let »

Les variables système

Ce sont des variables qui sont automatiquement affectées lors d'un appel de script.

$0 contient le nom du script

$1 à $9 les paramètres passés à l'appel du script

$# le nombre de paramètres passés au script

$\* la liste des paramètres à partir de $1

$$ affiche le numéro du PID du processus courant

|  |
| --- |
| #!/bin/bash# appel du script : **./salut nom prénom**if test $# -eq 2then echo "Salut $2 $1"else echo "Syntaxe : $0 nom prénom"fiecho "nombre de paramètres :"$#echo "nom du programme :"$0echo "liste des paramètres :"$\* |

|  |
| --- |
| [root@localhost tmp]# salut.sh jean richardSalut richard jean et bonne journée !nombre de paramètres :2nom du programme :/tmp/salut.shliste des paramètres :jean richard |

Les différents opérateurs pour les tests

La commande **test** sert à vérifier des conditions. Ces conditions portent sur des fichiers ou des chaînes ou une expression numérique.

**les tests sur les fichiers**

test -e fichier vrai si **fichier** existe

test -f fichier vrai si **fichier** est un fichier

test -d fichier vrai si **fichier** est un répertoire

test -r fichier vrai si **fichier** est accessible en lecture

test -w fichier vrai si **fichier** est accessible en écriture

test -x fichier vrai si **fichier** est accessible en exécution

test -s fichier vrai si **fichier** est de taille non nulle

exemples

* test -e /boot/grub/menu.lst
* test -f /boot/grub/menu.lst
* test -e /boot/grub/
* test -d /boot/grub/

**les tests sur les chaînes**

test -z chaine vrai si **chaine** est de taille nulle

test -n chaine vrai si **chaine** n'est pas de taille nulle

test chaine1 = chaine2 vrai si **chaine1** et **chaine2** sont égales

test chaine1 != chaine2 vrai si **chaine1** et **chaine2** ne sont pas égales

 ex : *if test "$variable" != "oui"*

**les tests numériques**

test int1 -eq int2 vrai si **int1** = **int2**

test int1 -ne int2 vrai si **int1** != **int2**

test int1 -lt int2 vrai si **int1** < **int2**

test int1 -le int2 vrai si **int1** <= **int2**

test int1 -gt int2 vrai si **int1** > **int2**

test int1 -ge int2 vrai si **int1** >= **int2**

**combinaison booléenne de tests**

test -r fichier -a -w fichier vrai si le **fichier** est en lecture ET en écriture

test -r fichier -o -w fichier vrai si le **fichier** est en lecture OU en écriture

exemple

if test $1 = "n" -o $1 = "N"

then

 echo "NON"

fi

L’instruction IF

|  |  |  |
| --- | --- | --- |
| echo $#if test $# -gt 1then echo "Trop de paramètre" exitfi | read REPif test $REP = othen echo "oui"else echo "non"fi | read REPif test $REP –eq 5then echo "5"elif test $REP –eq 10then echo "10"else echo "oups"fi |

L’instruction CASE

|  |
| --- |
| read REPcase $REP in o|O) echo "oui" ;; n|N) echo "non" ;; \*) echo "Les autres options !!!" ;; esac |

L’instruction FOR

|  |
| --- |
| for VAR in 1 2 3 4 5do echo $VAR …done |

|  |
| --- |
| LISTE=`ls`for VAR in $LISTEdo echo $VAR …done |

|  |
| --- |
| for VAR in `seq 1 1 10`do echo $VAR …done |

|  |
| --- |
| for ((i=1;i<=5;i++))do echo $i …done |

L’instruction WHILE

La répétition se poursuit TANT QUE la dernière commande de la liste est vraie

|  |
| --- |
| REP=owhile test $REP = odo  echo "Entrer o pour répéter la boucle : " read REPdone |

L’instruction UNTIL

La répétition se poursuit JUSQU'A CE QUE la dernière commande de la liste devienne vraie

|  |
| --- |
| REP=ountil test $REP != odo  echo "Entrer o pour répéter la boucle : " read REPdone |

note: l'instruction **break** permet de quitter rapidement une boucle (for, while, until)

Les fonctions

|  |  |
| --- | --- |
| **# exemple d'une fonction****# qui n'utilise pas de parametres****f1\_test(){** **echo "Pas de parametres !!!"****}****f1\_test** | **# exemple d'une fonction****# qui utilise des parametres****f2\_test(){** **echo "Le parametre est $1"****}****f2\_test "SALUT"** |

Exemple d'une fonction

|  |
| --- |
| **function today() {** **echo "Today's date is:"** **date +"%A, %B %-d, %Y"****}** |

Créer et exécuter un script

* Écrire un script
* Donner la permission d'exécuter le script
* Le placer dans un répertoire ou le "shell" peut le trouver

Le premier script

|  |
| --- |
| #!/bin/bash# Mon premier scriptecho "Salut !!!" |

Enregistrer le fichier "**mon\_script.sh**"

Donner les permissions au fichier "mon\_script": **chmod a+x mon\_script.sh**

Exécuter le script: **./mon\_script.sh**

Afficher le contenu actuel de la variable PATH

* **echo $PATH**

Ajouter votre répertoire à la variable PATH

* **export PATH=$PATH:directory**

SUGGESTION: ajouter l'export du PATH dans le fichier .bash\_profile de l'utilisateur

Suggestion, créer un répertoire script dans votre répertoire usager

* **mkdir /home/u1/script**

Maintenant le script s'exécute simplement avec la commande: **mon\_script.sh**