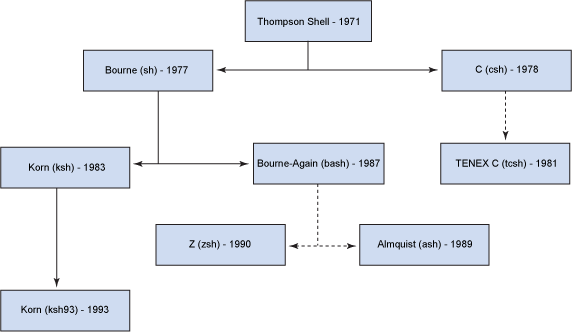
**Shell**



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 shelll 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, **/bin/bash** est la valeur par défaut.

Pour connaître le shell utilisé par un utilisateur:

* cat /etc/passwd | grep nom\_utilisateur

Le nom du shell qui est utilisé par défaut est dans la variable d'environnement SHELL.

* echo $SHELL

La commande **printenv** permet de d'afficher la liste complète des variables d'environnement.

**Exécution de plusieurs commandes sur la même ligne**

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 des commandes**

|  |  |
| --- | --- |
| **|** | 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/** |

**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, toutes les SORTIES (résultat d'une commande) sont dirigées vers 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 /tmp/ >> /tmp/info.txt** |

**Redirections des messages d'erreurs**

|  |  |
| --- | --- |
| **2>** | redirige le message d'erreur dans un nouveau fichier  si le fichier existe déjà, il écrase sans avertissement l'ancien contenu  **exemple: ls -l /tmmp/ 2> /tmp/erreurs.txt** |
| **2>>** | redirige le message d’erreur à la fin du fichier  **exemple: ls -l /tmmp/ 2>> /tmp/erreurs.txt** |

**Combinaison des symboles de redirection**

ls -l /tmp /tmmp >2 erreurs.txt > resultat.txt

**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** |

**Comment avoir une ligne en commentaire dans un script**

Une ligne en commentaire commence par un **#**

**Comment avoir plusieurs lignes en commentaire dans un script**

**<< --COMMENT--**

**Les lignes de code**

**--COMMENT--**

note: **--COMMENT--** est à votre discrétion

**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**

* ./*nomduscript*
* bash *nomduscript*

**chemin absolu**

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

Le fichier **/home/tech/.profile** contient du code pour la variable d'environnement PATH.

Si vous créez le dossier **bin** sous **/home/tech/** la variable d'environnement va inclure **/home/tech/bin/** à condition d'exécuter le code qui est dans le fichier **.profile**

La prochaine commande permet d'exécuter le code du fichier **.profile**

**source .profile**

**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
* \033c <ESC>c

Imprime le caractère représenté par le nombre octal 33 suivi d'un 'c'.

Dans la table ASCII, le nombre octal 33 est le caractère ESC (escape).

**printf** 🡺 affiche un message formaté 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

Le paramètre -p permet d'afficher un message lorsqu'on utilise read

read -p "Entrer votre nom: " nom

echo "Bonjour: " $nom

**Déclaration des variables**

variable=valeur

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

**Lire la valeur d'une variable**

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

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

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`
* noyau=$(uname -a)

**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"

résultat: le nom du dossier sera **Ce repertoire 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'

résultat: le nom du dossier sera **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 –r`

echo $noyau

**Effectuer des calculs avec let**

Avec let, il ne faut pas laisser 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

i=5

let i=$i+1

echo $i affiche 6

i=5

let i++

echo $i affiche 6

a=2

b=7

let c=$a-$b

echo $c affiche -5

i=5

let i--

echo $i affiche 4

a=8

b=2

let c=$a/$b

echo $c affiche 4

a=8

b=2

let c=$b/$a

echo $c affiche 0

**Bash effectue des calculs avec des nombres entier !!!**

a=10

let b=$a%3

echo $b affiche 1

**Effectuer des calculs avec (( ))**

Avec (( )), on peut laisser des espaces entre la variable et l'opérateur

echo $((30+2)) affiche 32

echo $(( 30 + 2 )) affiche 32

c=$((30+2))

echo c$ affiche 32

c=$(( 30 + 2 ))

echo c$ affiche 32

**priorité des opérateurs mathématiques**

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

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

**Bash effectue des calculs avec des nombres entiers !!!**

**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.sh nom prénom**  if test $# -eq 2  then  echo "Salut $2 $1"  else  echo "Syntaxe : $0 nom prénom"  fi  echo "nombre de paramètres :"$#  echo "nom du programme :"$0  echo "liste des paramètres :"$\* |

|  |
| --- |
| [tech@ubuntu-100]# **salut.sh jean richard**  Salut richard jean  nombre de paramètres :2  nom du programme :/tmp/salut.sh  liste 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 de caractères**

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

**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

**Les classes de caractères**

Linux utilise plusieurs classes de caractères pour faciliter les tests.

[[:alnum:]]

[[:alpha:]]

[[:blank:]]

[[:digit:]]

[[:lower:]]

[[:space:]]

[[:upper:]]

...

|  |
| --- |
| #!/bin/bash  read -p "Votre nom: " NOM  case $NOM in  [[:digit:]]\*)  printf "Le NOM $NOM débute par un chiffre.\n"  ;;  \*)  printf "Le NOM $NOM ne débute pas par un chiffre.\n"  esac |

La version moderne de "**test**"

La syntaxe est:

**[[ expression ]]**

|  |  |  |
| --- | --- | --- |
| **Opération** | **test** | **[[ expression ]]** |
| **AND** | -a | && |
| **OR** | -o | || |
| **NOT** | ! | ! |

**exemple**

if [[ $1 = "n" || $1 = "N" ]]

then

echo "NON"

fi

**L’instruction IF**

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

**L’instruction CASE**

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

**L’instruction FOR**

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

|  |
| --- |
| for VAR in 1 2 3 4 5 6 7 8 9 10  do  echo $VAR  …  done |

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

ou

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

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

ou

|  |
| --- |
| LISTE=$(ls)  for VAR in $LISTE  do  echo $VAR  …  done |

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

**L’instruction WHILE**

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

|  |
| --- |
| REP=o  while test $REP = o  do  echo "Entrer o pour répéter la boucle : "  read REP  done |

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

**L’instruction UNTIL**

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

|  |
| --- |
| REP=o  until test $REP != o  do  echo "Entrer o pour répéter la boucle : "  read REP  done |

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 script  echo "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**

**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** |