Linux: Le Shell

Table of Contents

Le shell (qui signifie coquille en anglais) est un programme qui sert d'interface en mode texte entre le noyau et l'utilisateur. Le shell est un interpréteur de commande et un langage de programmation . Le shell est une interface en mode texte dont le clavier est l'entrée et l'écran la sortie. Le shell de connexion est le premier shell dont l'utilisateur dispose.

A l'époque des terminaux, les shells étaient la seule manière d'accéder au système. Aujourd'hui, les shells ouverts dans une fenêtre de terminal restent l'interface préférée des administrateurs systèmes, notamment pour écrire des scripts shell.

Présentation des shells

Les terminaux

L'utilisateur qui se connecte au système dispose d'un shell dès sont authentification. Le shell par défaut de l'utilisateur est lancé automatiquement à partir d'un terminal passif , d'un émulateur de terminal pour PC (en démarrant avec le run level INIT 3 ou depuis l'interface graphique en pressant la combinaison de touches CTRL + ALT + F1 à F6, et CTRL + ALT + F7 pour revenir à l'interface graphique), d'une fenêtre de terminal (aussi appelée terminal X ou console virtuelle).

- Terminal passif
- Emulateur de terminal sur PC
- Fenêtre de terminal:
  - Xterm (émulation d'un terminal passif DEC VT100)
  - HPterm (émulation du terminal de HP)
  - Xpcterm

Les shells compatible Linux

Il existe différents shells parmi les plus connus:

Les shells compatibles Linux

Nom littéraire du shell	Nom du programme shell
Shell Bourne(l'antique shell de Steve Bourne)	/bin/sh
Korn shell, le shell de David Korn pour UNIX	/bin/ksh et /bin/pdksh (en freeware)

Nom littéraire du shell	Nom du programme shell
C shell	/bin/csh
Tenex shell(le shell C étendu)	/bin/tcsh
Zorn shell	/bin/zsh
Bash(Bourne Again Shell, le shell de Linux)	/bin/bash

il existe de nombreux autres shells compatibles Linux:

```
/bin/ash
/bin/adventure
```

Pour connaître le shell utilisé à l'aide de sa variable d'environnement:

```
$ echo $SHELL
```

Les commandes du shell bash sont des fichiers executables qui se trouvent dans le répertoire “/bin”. Le shell “sait” qu'un nom de fichier saisie à l'invite de commande est une “commande” parce que le fichier est marqué comme étant un executable (le fichier est un programme compilé), c'est à dire qu'un droit d'éxecution est associé à l'un des “sujets” (le propriétaire, un groupe, ou tout le monde) du fichier.

Le shell peut servir d'éditeur de ligne de commande , c'est à dire qu'il est possible d'utiliser les flèches de direction pour déplacer le curseur sur la ligne de commande sans écraser les caractères qui y sont déjà saisis. L'édition de la ligne de commande permettait dans les premiers temps des shells, de corriger ou de modifier une ligne de commande. Il fallait faire basculer le shell, du mode d'interprèteur de commande, vers le mode d'éditeur de ligne de commande.

Pour passer en mode d'éditeur de ligne de commande:

```
$ set -o vi (avec l'éditeur vi)
$ set -o emacs (avec l'éditeur emacs)
```

Le "nesting"

Les shells peuvent imbriqués les uns dans les autres (“nesting ”) ce qui permet de changer d'environnement à l'intérieur d'une même session. Cette technique peut également être utilisée pour des mesures de sécurité. Le shell du premier cercle est bridé (les commande exécutables sont très restricitives) mais il permet avec un mot de passe d'accéder au shell du deuxième cercle dont la marge de manoeuvre est plus importantes, et ainsi de suite jusqu'au dernier cercle où tout est permis pour les rares “privilégiés”...

Le prompt

Les commandes sont saisies depuis le prompt (ou invite de commande) qui peut prendre différentes formes; en général, le prompt affiche le nom de l'utilisateur (user), un arrowbase (@) le nom de la machine (localhost), puis le répertoire courant (pwd), et enfin un dièse (#) pour le super utilisateur

(root), le signe supérieur (>) ou le signe pourcentage (%) pour un utilisateur normal.

Les fichiers de démarage du shell bash

A chaque démarrage du shell, celui-ci exécute un certain nombre de fichiers exécutables avant de présenter l'invite de commande:

Pour l'ouverture de session:

- /etc/profile
- $HOME/.bash_profile
- $HOME/profile
- $HOME/.profile
- $HOME/.bashrc
- $ENV

Pour la fermeture de session:

- $HOME/.bash_logout

Les options du shell

Le shell propose des options permettant de paramétrer un certain nombre de fonctionnalités.

Activation et désactivation

Pour activer ou désactiver les options du shell, il suffit d'utiliser la commande set avec les options -o et +o.

Activation :

```
$ set -o option
```

Désactivation :

```
$ set +o option
```

Visualisation des options

Pour visualiser la liste des options disponibles ainsi que leur état, saisir dans une console :

```
$ set -o
allexport       off
braceexpand     on
emacs           on
errexit         off
errtrace        off
functrace       off
hashall         on
histexpand      on
history         on
ignoreeof       off
interactive-comments    on
keyword         off
monitor         on
noclobber       off
noexec          off
noglob          off
nolog           off
notify          off
nounset         off
onecmd          off
physical        off
pipefail        off
posix           off
privileged      off
verbose         off
vi              off
xtrace          off
```

Détails de certaines options :

ignoreeof

Pour quitter le shell, il existe 3 méthodes :

- La commande exit.
- La commande logout.
- La combinaison des touches ^d (CTRL+d).

Si l'option ignoreeof est activée, il n'est plus possible de quitter le shell en appuyant sur ^d.

noclobber

Quand une redirection est faite vers un fichier déjà existant, celui-çi est automatiquement écrasé sans confirmation. Pour inverser se fonctionnement, il suffit d'activer l'option noclobber.

On vérifie l'état de l'option noclobber

```
$ set -o | grep noclobber
noclobber       off
```

On redirige le résultat de la commande ls vers le fichier liste

```
$ ls -l > liste
```

On redirige le résultat de la commande pwd vers le fichier liste déjà existant

```
$ pwd > liste
```

On active l'option noclobber

```
$ set -o noclobber
```

On vérifie que l'option noclobber est bien activée

```
$ set -o | grep noclobber
noclobber       on
```

On redirige le résultat de la commande pwd vers le fichier liste déjà existant

```
$ pwd > liste
-bash: liste : impossible d'écraser le fichier existant
```

On force la redirection de la commande pwd vers le fichier liste déjà existant

```
$ pwd >| liste
```

emacs & vi

Ces 2 options permettent de paramétrer la rappel des commandes.

- En ksh, ces 2 options sont désactivées.
- En bash, seule l'option emacs est activée.

xtrace

Cette option est utilisée pour déboguer les scripts shell.

Les commandes

La syntaxe des commandes

Chaque commande doit être saisie selon les règles syntaxiques prévues par ses concepteurs afin d'être accepté par le shell.

```
commande options arguments
```

Les commandes internes sont intégrées aux shell (“build in”), tandis que les commandes dîtes externes sont des programmes qui se trouvent généralement dans les répertoires des exécutables ou des binaires (“/bin” ou “/usr/bin” ou “/usr/bin/local” ou “$HOME/bin”).

Une série de commande peuvent être saisient sur la même ligne; c'est la liaison de commandes ; chaque commande est séparée des autres par un point virgule (“;”) et chacune est exécutée séquentielement les une après les autres.

```
commande ; commande
```

La barre oblique inverse (“\” ou l'anti slash), celle qui sépare les dossiers de Windows, permet de continuer l'écriture d'une ligne de commande sur la ligne suivante.

La barre oblique (“/” ou le slash) est le seul caractère qu'il n'est pas permis d'utiliser pour nommer un fichier ou un répertoire, parce qu'il représente la racine du système de fichier et le séparateur de répertoires. Il n'est pas possible de créer dans un même répertoire plusieurs fichiers qui portent le même nom.

La sortie d'une commande peut être redirigée vers l'entrée d'une autre commande à l'aide d'un pipe (“|”, ALTGR + 6 et de valeur ASCII N° 124).

```
commande | commande
```

Les caractères spéciaux du shell

Les caractères spéciaux du shell sont des caractères qui, sauf indications contraires, sont interpréter par le shell pour exécuter certaines opérations.

Les caractères joker (?, * et []) peuvent être employés pour sélectionner plusieurs fichiers.

Les caractères accéssibles directement au clavier:

- L'esperluette (“&” avec la touche 1)
- Le guillemet (“”“ avec la touche 3)
- L'apostrophe (”'“ avec la touche 4)
- La parenthèse (”(“ et ”)“ avec les touches 5 et ° (degré))
- Le tiret (”-“ avec la touche 6, appelé aussi “moins”)
- Le signe égal (”=“ avec la touche +)
- Le slash (”/“ avec la touche :)
- L'espace (” “ avec la barre d'espace)
- L'étoile (“*” avec la touche *)
- Le dollar (“$” avec la touche $)
- Le signe inférieur (”<“ avec la touche <)
- Le point d'exclamation (”!“ avec la touche !)

Les caractères accéssibles indirectement à l'aide de la touche ALTGR (à droite de la barre d'espace):

- Le tilde (”~“ avec les touches ALTGR + 2)
- Le dièse (”#“ avec les touches ALTGR + 3)
- Les accolades (”{“ et ”}“ avec les touches ALTGR + 4 et ALTGR + +)
- Les crochets (”[“ et ”]“ avec les touches ALTGR + 5 et ALTGR + °)
- Le tube (“|” avec les touches ALTGR + 6)
- Le quote (“`” avec les touches ALTGR 7)
- L'antislash (“\” avec les touches ALTGR + 8)
- Le contrôle (“^” avec les touches ALTGR + 9, c'est le circonflex)
- L'arrowbase (”@“ avec les touches ALTGR + 0)

Les caractères accéssibles indirectement avec la touche MAJ (majuscule):

- La chaine (“§” avec les touches MAJ + !)
- Les signes supérieur (”>“ avec les touches MAJ + <)
- Le point d'intérrogation (”?“ avec les touches MAJ + ,)

Les paramètres de position

Une ligne de commande est interprété par le shell à partir du moment ou l'utilisateur appuie sur la touche RETURN. Le shell vérifie la syntaxe de la commande et la traduit en instruction machine. La ligne de commande est constituée de “mots” délimités par un séparateur (un espace ou une tabulation). Chaque mot est affecté à un paramètre de position numéroté de 0 à 9. La valeur des paramètres de position peut être restituée à l'aide du préfixe dollar (“$”) comme avec une variable d'environnement. Le premier mot est toujours le nom de la commande (“$0”) ou le nom d'un programme, tandis que les suivant (“$1” à “$9”) sont selon, des options ou des arguments.

```
$# restitue le nombre effectif de paramètres de position passé dans une
ligne de commande
$* ou $@ restitue la liste de l'ensemble des paramètres de position.
```

Les alias

Les alias permettent de remplacer une commande par un terme (généralement plus court à saisir et plus facile à retenir). Les alias ne sont pas conservés d'une session à l'autre; pour les initialiser à chaque ouverture du shell, il faut les enregistrer dans un fichier ”.profile“.

```
alias nom='commande'
alias variable="valeur"
```

Visualisation des alias

Pour visualiser la liste des alias déjà existant :

```
$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -lF'
alias ls='ls --color=auto'
alias tarc='tar -cvzf'
alias tarx='tar -xvzf'
```

Pour visualiser un alias en particulier :

```
$ alias la
alias la='ls -A'
```

Création d'un alias

```
$ alias c='clear'
$ alias rm='rm -i'
$ rm liste
rm : supprimer fichier «liste» ? y
```

En bash, pour créer un alias définitivement, il suffit de le renseigner dans le fichier ~/.bashrc

Suppression d'un alias :

```
$ unalias c
$ c
```

```
c : commande introuvable
```

L'historique des commandes

Le shell conserve toutes les commandes qui ont été éxécutées. La commande “history” permet de lister l'historique de toutes les commandes saisies.

Le shell enregistre toutes les commandes saisies dans un fichier texte :

- En bash, il s'agit du fichier `~/.bash_history`
- En ksh, il s'agit du fichier `~/.sh_history`

Pour relancer la commande précédente, il est possible d'utiliser la flêche de direction vers le haut ou la répétition du point d'intérogation (”!!“). Pour exécuter l'une des commandes de l'historique, il suffit d'en connaître le numéro (”!x“ avec x égal au n° de la commande dans la liste de l'historique).

Pour utiliser le rappel des commandes, le shell utilise soit emacs soit vi.

- En bash, c'est l'option emacs qui est activée par défaut mais il est possible d'utiliser vi.
- En ksh, les 2 options sont par défaut désactivées.

Le choix d'activer l'un, désactive l'autre.

```
$ set -o | grep "^emacs\|^vi"
emacs           on
vi              off
```

Pour les utilisateurs non habitués à l'éditeur vi, il est préférable d'utiliser emacs car le rappel des commandes se fait avec les flèches du clavier.

Les données qui ont été modifiées ne sont pas toujours enregistrées sur le disque dur immédiatement, elles transitent dans une mémoire tampon. Le débranchement brutale de l'alimentation de l'ordinateur risque de détruire les données du tampon qui n'ont pas encore été enregistrées!!! Le système de fichier journalisé ” ext3fs“ palie à cet inconvénient. La saturation d'un disque peut entrainer un blocage voir un crash du système.

la plus part des commandes du bash ne tiennent pas compte des fichiers “masqués” ou fichiers cachés , comme par exemple, tous les fichiers commençant par un point. C'est la sécurité par l'obscurité, il faut demander explicitement l'affichage de tous les fichiers pour les voir.

La gestion des processus

Une commande saisie à l'invite puis validée en pressant sur la touche RETURN est exécutée par un

processus fils du shell.

Pendant l'éxécution de la commande le shell garde la main, et la rend une fois que la commande est terminée en présentant à nouveau le prompt; l'éxécution est dit en direct (foreground). Une commande peut être lancée en direct ou en différée (&) afin de récupérer tout de suite la main.

Un programme lancé en direct peut être interrompu avec la combinaison de touches CTRL + Z, et stopper avec CTRL + C.

Il est possible d'éxécuter des commandes en différé (background avec ” & “ en fin de commande), ainsi, l'utilisateur récupère immédiatement la main.

Le shell affiche le numéro de la tâche correspondant à la commande lancée en arrière plan et le numéro du processus qui exécute le travail. La commande “jobs ” permet de connaître les tâches en arrière plan.

La commande “fg” permet de faire revenir une tâche en avant plan, tandis que la commande “bg” lance en tâche de fond un processus suspendu.

Certaines commandes comme “cat” envoient directement leur sortie sur l'écran; il n'est donc pas possible de les lancer en arrière plan, ni de libérer l'invite de commande avant la fin de leur exécution.

La gestion des fichiers

Les types de fichiers

Pour linux tout est fichier ou tout est lien, cependant il existe différents types de fichier.

Les types de fichiers sont symbolisés:

- Les deux points (”..“) représentent le répertoire parent.
- Le point (”.“) représente le répertoire courant.
- Le tilde (”~“) représente le répertoire de l'utilisateur courant.
- La racine (”/“) représente le répertoire racine et le séparateur de répertoire.
- L'étoile (“*”) représente un fichier exécutable, avec le droit “x”.
- Le tube (“|”) représente l'opérateur de redirection vers un autre programme, aussi appelé “pipe” FIFO pour First In First Out.
- L'arrowbase (”@“) représente un lien “hard” ou “symbolique” avec un autre fichier, comme un raccourcis,...
- Le signe égale (”=“) représente les fichiers de communication “sockets”.
- Le point (”.“) est aussi utilisé pour nommer les fichiers cachés.

if [-L nom-du-fichier] - **-L** renvoie vrai si le “fichier” existe et est un lien symbolique (le fichier lié peut exister ou non)
if [-f nom-du-fichier] - **-f** renvoie vrai si le fichier existe et est un fichier normal
if [-e nom-du-fichier] - **-e** renvoie vrai si le fichier existe quel que soit son type.

Les droits des fichiers et des répertoires

Les répertoires doivent avoir le droit exécutable (“x”) pour tous les utilisateurs (“other”) afin qu'ils soient “traversables ”.

Les répertoires doivent avoir le droit de lecture (“r”) pour tous les utilisateurs (“other”) afin qu'ils soient possible d'en lister le contenu.

Les permissions d'un fichier octroyées à tout le monde (“other”) ne sont pas prioritaires et ne sont pas valides quand l'utilisateur appartient au groupe et que le groupe n'a aucune permission sur le fichier. L'ordre de vérification des permissions est “user”, “group” et enfin “other”.

La redirection des Entrées/Sorties

La redirection des Entrées/Sorties s'appèle également “le traitement en pipeline”.

La redirection des Entrées/Sorties (E/S) s'effectue au travers des trois cannaux de communication que possède chaque processus:

- **stdin** (standard input) est par défaut associé au clavier et est identifié par le N°0
- **stdout** (standard output) est par défaut associé à l'écran et est identifié par le N° 1
- **stden** (standard error) est par défaut associé à l'écran et est identifié par le N° 2

La redirection des E/S s'effectue avec l'opérateur supérieur (>) pour la sortie, inférieur (<)pour l'entrée. La redirection simple crée le fichier cible s'il n'existe pas et écrase son contenu s'il existe. La redirection avec ajout (» et «) ne détruit pas le contenu antérieur mais le rajoute à la fin.

Le pipe (|) permet de rediriger la sortie d'un processus vers l'entrée d'un autre. Le pipe est le caractère de la barre verticale (ALTGR + 6).Le pipe est l'opérateur de redirection entre commande. Un pipeline est une succession de commande reliées entre elles par des pipe. La commande “more” permet de ralentir la sortie d'un pipeline et de l'afficher page par page. La commande “tee ” permet de rediriger son entrée à la fois vers l'écran et vers un fichier donné en argument.

Les redirections vers le répertoire ”/dev/null“ sont irrémédiablement perdues puisque ce fichier qui correspond à la corbeille (bit-bucket) est un fichier de périphérique spéciale dont la taille est toujours égale à zéro octets.

La syntaxe des opérateurs de redirection:

```
commande > sortie
commande < entrée
commande < entrée> sortie
commande 2> erreurs.txt (redirige les erreurs de syntaxe, le flux "stden"
vers un fichier)
commande | commande | commande
```

Quelques commandes shell

Pour apprendre l'utilisation du shell bash et de ses variables d'environnement:

```
man bash
```

Pour définir une variable d'environnement:

```
nom=valeur
```

Pour exporter une variable d'environnement:

```
export nom
```

Pour définir et exporter une variable d'environnement (le shell bash permet de faire les deux opération en une seule ligne de commande):

```
export nom=valeur
```

Pour afficher les variables d'environnement actuelles:

```
set
```

Pour exécuter une commande dans un environnement modifié :

```
env
man env
```

Pour obtenir des valeurs numériques “au hasard”:

```
echo $RANDOM
echo $SECONDS
```

Pour concaténer deux fichiers en un seul (le contennu du fichier de destination est écrasé, s'il existait):

```
cat un deux > ensemble
```

Pour conserver le contenu préexistant du fichier de destination, il faut faire une double redirection :

```
cat un deux >> toujours
```

Pour enregistrer les fautes d'orthographes d'un texte. Par défaut, les erreurs de syntaxe sont redirigées vers la sortie standard (l'écran), mais il est possible de les redirigées vers un fichier.

```
cat rapport.txt | spell > faute.txt
```

Les variables d'environnement du shell bash

Le shell utilise des variables d'environnement dont on peut faire apparaître le contenu avec le signe dollars (“$”). Les principales variables d'environnement sont prédéfinies, mais un utilisateur peut en définir aussi. Certaines variables d'environnement sont exportables, c'est à dire que leur valeur est transmise aux processus enfants du processus shell.

Certaines variables d'environnement du shell sont prédéfinies (c'est à dire qu'elle existe dès l'ouverture d'un shell), certaines sont exportables (c'est à dire que les processus qui sont lancés à partir du shell héritent de la valeur de chaque variable exportable), certaines peuvent être définies par l'utilisateur (il suffit simplement de ne pas choisir un nom réservé par le shell ou le système).

Par convention, les variables d'environnement prédéfines sont écrites en majuscule.

Le contenu d'une variable d'environnement peut être affiché avec la commande `echo` et le symbole dollar (“$”) précédent le nom de la variable.

Les variables d'environnement prédéfinies peuvent être affichées avec la commande `set`):

La variable PATH

Le chemin absolu de ces répertoires doit être enregistré dans les chemins de recherche définies dans la variable d'environnement PATH afin que les programmes qu'ils contiennent soient accéssibles par le shell. Pour chaque commande validée, le shell vérifie qu'elle correspond, soit à l'une de ces commandes intégrés, soit à l'un des programmes contenus dans les chemins de recherche définis dans la variables PATH.

Les script shells peuvent être également enregistrés dans les répertoires habituels du PATH (”/bin“ ou ”/usr/bin“ ou ”/usr/bin/local“ ou “$HOME/bin”). Pour désactiver un script shell, il suffit de lui enlever le droit d'exécution (“x”).

Pour rajouter le répertoire personnel de l'utilisateur dans les chemins de recherche (pendant le temps de la session):

```
PATH=$PATH:$HOME
```

La programmation shell

La programmation shell consiste à écrire de “petit programme”, les scripts shells à l'aide des outils du shell (commandes, options, arguments, variables, paramètres, fonctions, structures de contrôle, redirections, filtres,...). Les scripts shells (aussi appelés scripts d'environnement) sont des fichiers textes exécutables dont le shell intreprète et traite chaque ligne séquentiellement (c'est un traitement séquentiel).

Les scripts shells peuvent s'exécuter en tâche de fond

Il faut enregistrer les scripts dans les répertoires prévus pour les programmes (“$HOME/bin”, ”/usr/local/bin“), ou du moins dans un des répertoires de la variable PATH, afin que le shell puisse les retrouver quand l'utilisateur les invoque. Pour exécuter un script qui se trouve dans le répertoire courant depuis ce même répertoire, il faut faire précéder le nom du script d'un point et d'un slash pour indiquer au shell le répertoire de travail (par exemple : ”./script“). Ainsi, le shell est forcé d'exécuter le fichier du répertoire de travail au lieu d'essayer d'aller le rechercher dans les chemins de la variable PATH.

Il y a trois façon d'appeler un script:

- L'appel implicite qui consiste à saisir le nom du script à partir du prompt. L'utilisateur doit avoir les droits de lecture ® et d'exécution (x) pour pouvoir lancer le script. Une fois lancé, le shell crée un processus avec un sous-shell qui aura la responsabilité d'interpréter et d'executer le script (par exemple : “script”).
- L'appel explicite qui consiste à préciser le shell qui exécutera le script. il suffit de saisir le nom de l'interpréteur suivi du nom du script (par exemple : “bash script”).
- L'appel par le point qui consiste à saisir le nom du script depuis le répertoire ou se il trouve, précédé d'un point et d'un espace (par exemple : ”. script“). Ainsi, le script est éxécuté par le shell et dans l'environnement courant; il n'y a pas de lancement de sous-shell.

Les scripts shells

Les scripts shell sont des fichiers textes dans lesquels figurent des commandes telles qu'elles pourraient être écrites à l'invite de commande.

Les scripts shell commencent généralement par indiquer avec lequel des shells ils peuvent être executés:

```
#! /bin/bash
#! /bin/sh
```

Certains caractères sont spécialement utilisés dans les scripts:

#	pour les commentaires
$variable=valeur	pour le passage d'un paramètre au script
read variable	donne la main à l'utilisateur pour saisir une valeur qui sera enregistrée dans variable

for	boucle itérative
if	test de comparaison
case	test multi choix

Les structures de contrôle des scripts shells

Les structures de contrôle peuvent être des branchements ou des boucles.

Les branchements (IF et CASE) sont des instructions de décision ou de consultation qui sont fonction de certaines conditions.

Le branchement binaire:

```
IF expr THEN
ELSE THEN
FI
```

Le branchement multiple:

```
CASE IN
  1
  2
  3
  DEFAULT
ESAC
```

Le branchement “TEST” (qui correspond à “IF THEN” dans d'autres langages de programmation) procéde à un contrôle d'une expression simple (composée d'un seul élément) ou procède à une comparaison de deux éléments. Le branchement “TEST” renvoi un code de retour qui est égal à 0 quand la comparaison est vraie (true) ou à 1 quand la comparaison est fausse (false).

Les boucles (FOR, WHILE et UNTIL) sont des instructions répétitives ou itératives qui traitent plusieurs fois une même partie de programme en fonction de certaines conditions. Le traitement séquentiel est interrompu en fin de boucle pour revenir en début de boucle (point d'itération) et exécuter de nouveau le corps de la boucle. La condition “true ' (qui a pour valeur 0) est toujours vraie, ce qui permet de créer des boucles sans fin.

La boucle WHILE continue “tant que” la condition est vraie:

```
WHILE expr DO
DONE
```

La boucle UNTIL continue “jusqu'à” ce que la condition soit vraie (c'est la négation de la boucle WHILE):

```
UNTIL expr DO
DONE
```

La boucle FOR est une boucle de comptage dont le nombre de passage est connu dès le départ.

```
FOR expr DO
DONE
```

Les instructions des boucles des scripts shells

- L'instruction **continue** permet de quitter le corps d'une boucle mais l'exécution de la boucle se poursuit en revenant au début.
- L'instruction **break** permet de quitter la boucle, comme si la condition de fin de boucle avait été réalisée.

Les calculs arithmétiques des scripts shells

Le shell BASH ne sait calculer que sur des nombres entiers. Les nombres à virgules flottantes doivent être traité avec la calculatrice “bc”. Les opérateurs arithmétiques sont plus ou moins prioritaires et l'ordre de calcul peut être facilité à l'aide de parenthèses.

Les opérateurs des scripts shells

L'esperluette ("&").

Les opérateurs de comparaison de chaînes de caractères:

```
= pour égal
!= pour différent
```

Les opérateurs de comparaison numérique:

```
-eq pour "equal"
-ge pour "greater or equal"
-gt pour "greater than"
-le pour "less or equal"
-lt pour "less than"
-ne pour "not equal"
```

Les opérateurs logiques:

```
-a pour "and"
-o pour "or"
! pour "not"
```

Les opérateurs arithmétiques:

```
+ pour l'addition
- pour la soustraction
* pour la multiplication
/ pour la division
```

Quelques scripts shells

Pour afficher la version de Linux:

```
echo -n "Votre version de Linux est" cat /etc/redhat -release
```

Pour interrompre l'éxécution d'un programmes, le temps que l'utilisateur saisisse une entrée:

```
echo -n "Saisissez le nom du fichier :"
read fichier
```

Pour éxécuter la commande “uname” et placer le résultat dans la variable “os”:

```
os="uname -s"
if (test $os="linux")
  then echo "Votre os est linux"
else echo "Mais qu'est-ce donc ?"
```

Pour saisir un nom de fichier et connaître son type:

```
echo -n "caractéristiques du nom de fichier :"
read pattern
for filename in $pattern
do echo "$filename"
done
```

Pour réaliser un test multiple:

```
#! /bin/bash
```

```
case $SHELL in
/bin/bash) echo "vous utilisez le shell bash";;
/bin/tcsh) echo "Vous utilisez le shell tcsh";;
*) echo "Vous utilisez le shell $SHELL";;
esac
```