Présentation et cours Korn Shell (compatible avec le Bash)

Date de publication : 14/12/2004 , Date de mise a jour : 14/12/2004

Par marcg (autres articles)

Usage et conpréhension du Korn Shell (Bash)

Introduction
1.1. Le but
1.2. Le rôle
1.3. Les avantages du Korn Shell
1.4. Le choix du shell
1.5. La syntaxe
1.6. Les fichiers exécutables
Les généralités du Shell
2.1. L'éxécution d'une commande
2.2. Les droits des fichiers
2.3. La modification des droits
2.3. Les variables d'environnement
2.4. Autre variable du KSH
2.5. L'environnement
2.6. La personnalisation
2.6.1. Le fichier .profile (.bash_profile)
2.6.2. Les alias
2.6.3. Echantillon de fichier .profile et .kshrc
2.7. L'édition des commandes
Redirection,Substitution,enchaînement Les redirections sont le détournement des 3 descripteurs de fichiers standards à savoir :
3.1. Les redirections en sortie
3.2. Les redirections en entrée
3.3. La redirection des erreurs
3.4. Les enchaînements (pipe et tee)
3.5. la subtitution de commande
3.6. enchaînement de commandes
3.7. Les commandes groupées dans un autre shell
3.8. Les commandes groupées dans le même shell
3.9. Les opérateurs logiques
Les caractères spéciaux
4.1. Les caractères connus
4.2. Les mètacaractères
4.3. Les expressions
Les commandes interne
5.1. Généralités
5.2. Commande cd

5.3. Processus
5.4. Les limites du système
5.6. Droits sur les fichiers à leur création
5.7. Les alias
5.8. La commande whence (type en bash)
Les commandes externes
6.1. Généralités
6.2. Find
6.3. Head, tail
6.4. Grep
6.5. Sed
6.6. Cut
6.7. Awk
Les paramètres et les variables
7.1. Généralité
7.2. Manipulation des variables
7.3. La substitution de variables
7.3.1. Substitution temporaire
7.3.2. Substitution réelle =
7.3.3. Substitution impossible ?
7.4. Les attributs de variables
7.5. Les tableaux
7.6. Les paramètres du shell
7.7. La commande set
Les tests
8.1. La commande test
8.2. Execution conditionnelle
8.3. Tests sur les fichiers
8.4. Tests sur les chaînes de caractères
Les expressions arithmétiques et logiques
9.1. Les opérateurs
9.2. Les priorités
9.3. Conversions
9.4. La commande let ou (( ))
9.5. La commande bc (beautiful calculator)
Les contrôles de boucles
10.1. Les structures courantes
10.2. La structure select
2.1. La modification du déroulement d'une boucle
Les fonctions
Le contrôle des processus
L'éxécution : Appel de shells et de scripts

Introduction

1.1. Le but

Le shell permet de réaliser des petits programmes qui deviennent des outils.
Ces outils faciliteront les tches répétitives de l'administrateur, de l'utilisateur.
Pour le programmeur, un en-capsulage de ces sources et un enchaînement de programme peuvent être utiles.

1.2. Le rôle

Le shell est :
- l'interpréteur de commande,
- l'interface entre l'utilisateur et les commandes,
- un langage de programmation (interpréteur), il permet donc de réaliser de nouvelles commandes,
- un gestionnaire de processus,
- un environnement de travail configurable.

1.3. Les avantages du Korn Shell

Le korn shell regroupe les fonctions du C shell et du bourne shell, tout en apportant de nouvelles propriétés, afin d'obtenir un shell plus convivial, plus puissant et plus rapide.

Le korn shell a les possibilités supplémentaires suivantes :

- un historique des commandes peut-être mis en place et utilisé (vi ou emacs).
- une compatibilité bourne shell.
- des variables d'environnements supplémentaires (par exemple la définition de directories).
- des commandes Bourne avec de nouvelles fonctionnalités (test, expr, echo).
- des sélections par menu possible.
- des messages d'erreur plus significatifs.
- des alias peuvent être créés.

Les différence entre le Korn Shell et le Bash sont suffisement faibles pour envisager des scripts commun.

ATTENTION dans la suite de ce documents la commande print est utilisée pour afficher, il convient de la remplacer par echo en Bash

1.4. Le choix du shell

Les choix possibles :
- Le Bourne shell /bin/sh (sous AIX : /bin/bsh)
- Le C shell /bin/csh
- Le Korn shell /bin/ksh
- Le Bourne shell again (Bash) /bin/bash

Le positionnement d'un shell peut se faire :
- par l'administrateur (fichier /etc/passwd)
- aprés le login, par une demande utilisateur grce aux commandes :
. bsh
. csh
. ksh
. bash
La variable SHELL est affectée lors du login et n'est plus modifiée par la suite (entre autre par les commandes précédentes).
Une fois le sous shell (bsh, csh ou ksh) démarré, il est possible de le quitter à l'aide de la commande exit ou CTRL D

1.5. La syntaxe

La forme générale est la suivante :
commande options arguments

Plusieurs commandes peuvent être passées sur une même ligne, pour ce faire, elles doivent être séparées par " ; ".
Les paramétres d'une commande peuvent être séparés par :
- une tabulation ou un blanc
- le caractère entré ("NL")
Ces séparateurs sont définis par la variable IFS.

1.6. Les fichiers exécutables

Dans le cas de commandes placées dans un fichier ascii, celui-ci doit être rendu exécutable, ce fichier exécutable est également appelé script shell.
Ce fichier est exécutable comme un processus.
Ce fichier peut réaliser l'équivalent d'une suite de commande UNIX ou (et) korn shell.
Ce fichier est exécuté sous le shell standard, sauf spécification contraire sur la premiére ligne du fichier.
exemple:
#!/bin/ksh

Les généralités du Shell

2.1. L'éxécution d'une commande

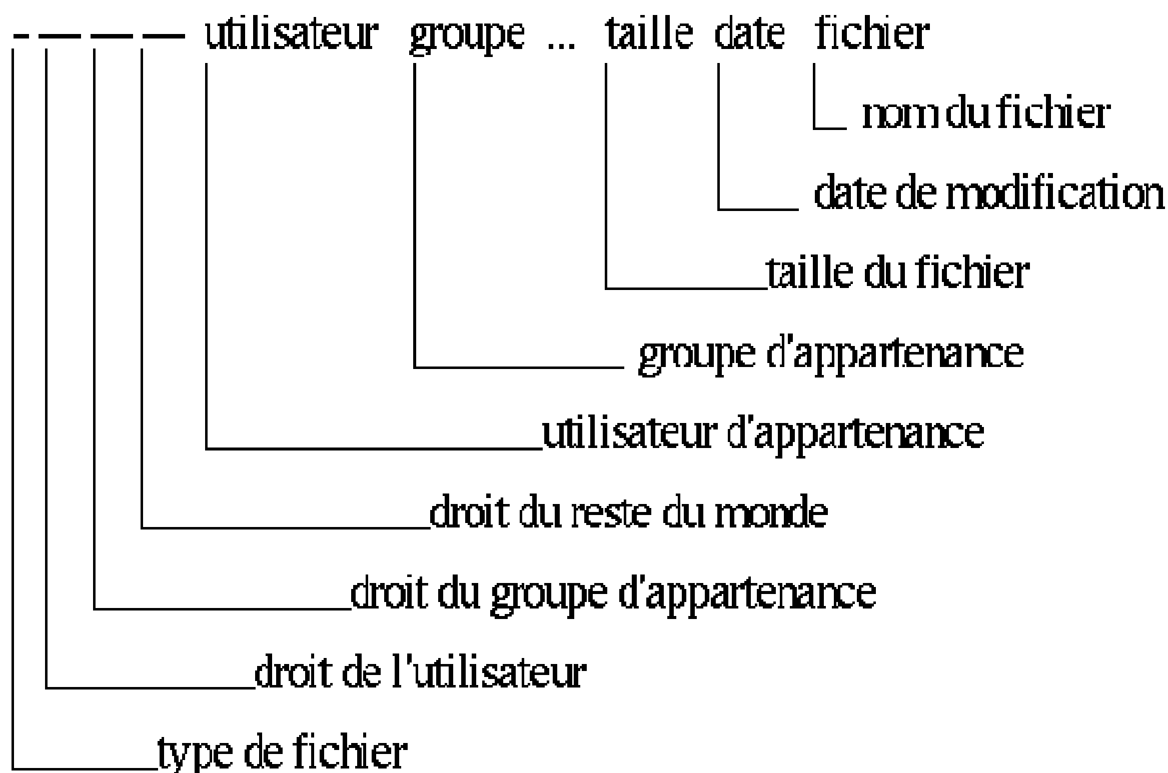
Les commandes peuvent être lancées en avant plan (l'utilisateur doit donc attendre la fin de la commande avant de pouvoir exécuter la suivante).
Les commandes peuvent être lancées en arrière plan (dans ce cas le numéro du process est visualisé et l'utilisateur peut continuer immédiatement à exécuter d'autres commandes).
Un processus lancé en arrière plan peut être interrompu par la commande KILL ou CTRL Z.

2.2. Les droits des fichiers

Un script shell doit avoir des droits d'exécution pour que le shell en cours lance un shell pour l'exécuter.
exemple:
script

Un script shell peut être lancé qu'il est ou non des droits d'exécution en passant le nom du script comme paramètre à un shell.
exemple:
ksh script

Dans les deux cas précédents, un autre shell est exécuté, donc tout environnement modifié dans le script n'existera plus à la fin de l'exécution de ce script.
Pour visualiser les droits d'un fichier la commande ls peut être utilisée avec comme paramètre -al
Les droits sont visualisés ainsi :



Les droits de l'utilisateur, du groupe ou du reste du monde sont représentés par :
- absence d'autorisation
r autorisation en lecture (en octal : 4)
w autorisation en écriture (en octal : 2)
x autorisation en exécution (en octal : 1)
ou en droit de passage pour les directories

Des droits par défaut sont positionnés lors de la création d'un fichier.
La commande umask permet de positionner les droits par défaut.
Le paramètre de la commande umask est une valeur de 3 chiffres exprimés en octal.
exemple:
umask 022
Le masque de l'exemple positionne pour un fichier les droits suivant :
rw- r-- r--
Les droits sont calculés par la méthode suivante (complément à 666) :
666 octal de référence pour les fichiers
022 le umask
644 les droits par défaut soit rw- r-- r--
(110 100 100)
Du fait de l'octal de référence, il est impossible de positionner par défaut le droit d'exécution pour un fichier.

Pour une directorie (répertoire), l'octal de référence est 777 soit :
777 octal de référence pour les fichiers
022 le umask
755 les droits par défaut soit rwx r-x r-x
(111 101 101)
Il est donc possible pour une directorie d'avoir les droits de passage par défaut.

2.3. La modification des droits

La commande chmod permet de modifier les droits d'un fichier ou d'une directorie.
Il est possible de modifier les droits par :
- utilisation de l'octal correspondant au choix.
exemple: chmod 555 toto
r-x r-x r-x
(555 correspondant à 101101101 en binaire)
- utilisation de symbole.
u pour l'utilisateur
g pour le groupe
o pour le reste du monde
a pour tous (u et g et o)
r pour lecture
w pour écriture
x pour exécution (ou passage)
+ pour ajout d'un droit
- pour retrait d'un droit
exemples: chmod gu+r toto
chmod uo-x *
chmod

2.3. Les variables d'environnement

Le shell utilise des variables d'environnement :
(attention aux MAJUSCULES)
PS1 pour le prompt primaire, par défautt $
PS2 pour le prompt secondaire, par défaut >
PATH pour les répertoires de recherche des exécutables.
MANPATH pour les répertoires de recherche des fichiers de man.
LANG pour la langue utilisée.
HOME pour la directorie de base de l'utilisateur (appelé également home directorie).
LOGNAME pour le nom de l'utilisateur
SHELL pour le shell utilisé
Le korn shell utilise d'autres variables d'environnement nécessaires à l'utilisation de ses fonctions :
HISTFILE pour le fichier historique.
HISTSIZE pour la limite de commandes historiques accessibles.
EDITOR pour l'éditeur de ligne de commandes.
VISUAL pour remplacer $EDITOR si préalablement défini.

2.4. Autre variable du KSH

Le korn shell a la possibilité de définir des variables désignées :
RANDOM pour définir un nombre aléatoire compris entre 0 et 2E 15 c'est à dire 32767.
LINENO pour définir le numéro de ligne courante d'un script shell ou d'une fonction.
SECONDS pour le temps écoulé depuis l'appel du shell (en secondes).
PWD pour le répertoire actuel.
OLDPWD pour le répertoire précédemment utilisé, c'est à dire avant le dernier cd.
NB : ces deux dernières variables sont modifiées à chaque utilisation de la commande cd.

2.5. L'environnement

Le korn shell possède des fichiers de configuration spécifiques. Lorsque celui-ci est appelé lors de la connexion, ces fichiers de configurations sont alors exécutés :
/etc/environment
/etc/profile
.profile
ENV=nom_fichier (Par convention .kshrc)
NB : /etc /environment est exécuté par tous les processus de login.
/etc/profile et .profile sont exécutés par le shell de connexion (le fichier .profile est exécuté dans le répertoire de base de l'utilisateur, home directory)
Seul le korn shell exécute le fichier affecté à ENV. Il s'agit d'un script qui sera exécuté à chaque lancement de korn shell et qui sera affecté à la variable d'environnement ENV. C'est pourquoi, il est nécessaire de placer cette variable dans le fichier .profile.
A chaque nouveau lancement d'un korn shell explicite (c'est à dire utilisation de l'appel ksh ou ksh nom_script ou # !/bin/ksh), il y a exécution du fichier affecté à la variable ENV.
On affecte les paramètres de ENV à l'aide de la commande export dans le fichier .profile
Exemple
Export ENV=$HOME/.kshrc

2.6. La personnalisation

La personnalisation est un des atouts majeurs du korn shell.

2.6.1. Le fichier .profile (.bash_profile)

Ce fichier permet à chaque utilisateur de personnaliser son environnement de travail, en attribuant des tches spécifiques de connexion.
On peut ainsi :
positionner des chemins de recherche
mettre en place les protections de fichiers par défaut (umask)
positionner le type de terminal et l'initialiser
positionner des variables d'environnement
effectuer des tches personnalisées nécessaires suivant le site.
Cf. exemple de fichier .profile page suivante.

2.6.2. Les alias

Les alias permettent de créer de nouvelles commandes à partir de commandes existantes et d'ainsi établir une bibliothèque personnalisable suivant l'environnement de travail.
Dans le cas du ksh, les alias sont à placer dans le fichier d'initialisation défini par la variable d'environnement ENV (en général .kshrc)
Cf. échantillon de .kshrc page suivante.

2.6.3. Echantillon de fichier .profile et .kshrc

```
# échantillon de fichier .profile

ulimit -c 0
umask 022
PATH=\usr\ucb:\bin:/usr/bin :usr/local/bin : .
LPDEST=ps
EDITOR=emacs
MORE=-c
ARCH_DIR=/home/pubg95/archdir/
mesg y
biff y
PS1= « uname -n`-\ !> »
export PATH LPDEST EDITOR MORE ARCH_DIR PS1
export TIMEOUT
echo « Entrez le type du terminal : \c » ; read tt
if [ « $tt » != « »] then
TERM=$tt ; export TERM
tset
endif


# échantillon de fichier .kshrc

alias j=jobs
alias h=history
alias l= « ls -aFx »
alias ll= « ls -aFxl »
alias psa= « ps -ef | head »
export HISTSIZE=60 #le répertoire principal de #l'utilisateur est fixé à  60 #commandes
```

2.7. L'édition des commandes

Le korn shell peut stocker (sur demande), dans un fichier, les commandes passées en interactif.
Pour que le korn shell débute l'historisation, il est nécessaire de configurer la variable VISUAL (dans le fichier .profile ou en interactif par : export VISUAL=vi) ou de passer la commande : set -o vi.
Le fichier de stockage est .sh_history dans la home directorie de l'utilisateur, il est possible de modifier ce fichier de stockage par défaut en modifiant la variable HISTFILE.
Le nombre d'anciennes commandes disponibles dans ce mode est configurable par la variable HISTSIZE.
A partir du moment où ce mode est actif l'utilisateur peut quitter le mode insertion (type vi) pour revenir au mode commande de vi par la touche "Echap".
L'ensemble des possibilités du mode commande de vi sont accessibles.

Par exemple :
k permet d'extraire des commandes du fichier historique en arrière.
j permet d'extraire des commandes du fichier historique en avant.
l permet de déplacer le curseur vers la droite.
h permet de déplacer le curseur vers la gauche.
Cf. petit lexique de commandes vi en annexe
L'édition des commandes peut également ce faire par la commande interactive fc.
$ fc commande
Les premières lettres de la commande suffisent, il est également possible d'appeler la commande par son numéro.
Un éditeur peut être choisi grâce à l'option -e.

Exemple :
fc -e vi commande
La réexécution d'une commande peut être réalisée par la commande r.

Exemple :
r réexecute la dernière commande
r commande réexecute la dernière commande débutant par commande

Exemples:
fc -e - #pour réexecuter la dernière commande.
fc -e - nom_commande #pour réexecuter la dernière commande nom_commande.
fc -e vi 20 30 #vi édite les lignes 20 à 30 du fichier historique
fc -l #pour lister les 16 dernières commandes
(16 par défaut)

Redirection,Substitution,enchaînement

Les redirections sont le détournement des 3 descripteurs de fichiers standards à savoir :

- l'entrée standard (noté 0) : le clavier
- la sortie standard (noté 1) : la console courante
- la sortie des erreurs (noté 2) : la console courante

3.1. Les redirections en sortie

La sortie standard (l'écran de votre console) peut être redirigée (remplacée) par un fichier ou une sortie.

exemple:

ls >toto
#le fichier toto est créé et contient le résultat du ls.
#si toto existait, son contenu serait écrasé par le
#résultat de ls

>vide
#le fichier vide est créé et ne contient rien.

ls >/dev/null
#la commande ls est exécutée, le résultat est poubellisé.

ls >>titi
#le résultat de ls est ajouté à la fin du fichier titi.

3.2. Les redirections en entrée

L'entrée standard (le clavier de votre console) peut être redirigé (simulé).

Exemple:
mail destinataire ‹ clavier
#le fichier clavier contient le texte à destination de bruno

Il est également possible de simuler une saisie dans un shell script par ‹‹.

Exemple:

mail destinataire ‹‹END
tu as brillamment réussi ta formation korn shell
félicitations
END

3.3. La redirection des erreurs

Les erreurs peuvent être redirigées explicitement.

Exemple:

titi 2>fichier_erreur

La commande titi n'existant pas le fichier fichier_erreur contiendra le message d'erreur.

La sortie standard et les erreurs peuvent être redirigées sur le même fichier.

Exemple:

ls -al 1>fichier 2>&1
Les erreurs sont redirigées sur le descripteur 1 soit "fichier".

3.4. Les enchaînements (pipe et tee)

Il est possible d'utiliser le résultat d'une commande comme entrée de la commande suivante, pour ce faire, il existe le pipe | .

Exemple:

ls -al | more # cette commande est pratique pour #lister page par page.

Cette ligne est équivalente à :

ls -al >fichier
more ‹ fichier

Il est possible d'utiliser plusieurs enchaînements successifs :

Exemple:

ps -eaf | grep root | wc -l

La commande tee permet de stocker un résultat intermédiaire.

Exemple:

ps -eaf | tee fichier | grep root | wc -l

Le fichier fichier contient le résultat de ps -eaf.

3.5. La substitution de commande

Une commande ou un enchaînement de commande peut servir de paramètre à une commande.

Exemple:

cat $(cat liste_des_fichiers_a_concatener) >resultat

La syntaxe pour le bourne shell est le " ` " (accent grave).

Exemple bourne shell :

cat `cat liste_des_fichiers_a_concatener` >resultat

3.6. enchaînement de commandes

Une suite de commande, sans interaction entre elles, peut être réalisée en intercalant un " ;
"entre chaque commande.

Exemple:

ls -al ; who am i

Dans le cas de certaines commandes (entre autre les commandes d'impression) il est nécessaire de protéger le caractère ";" par un "\" avant celui-ci pour qu'il soit interprété comme caractère et non comme séparateur de commande.

Exemple:
print bonjour \; vous êtes au cours korn shell
Résultat:
bonjour ; vous êtes au cours korn shell.

3.7. Les commandes groupées dans un autre shell

Il y a donc création d'un nouveau processus fils.

Il est possible de grouper des commandes dans un autre shell en utilisant les parenthèses ().
Il y a donc création d'un nouveau processus fils. L'exécution s'effectue dans un nouvel environnement et sans modification de l'environnement actuel.

Exemple:

(export LANG=france ; echo $LANG) ; echo $LANG

Il est possible de rediriger les sorties de l'ensemble des commandes groupées.

Exemple:
(export LANG=france ; echo $LANG) >fichier

3.8. Les commandes groupées dans le même shell

Il est possible de grouper des commandes dans un même shell en utilisant les accolades {};
ces accolades doivent être précédées et suivi d'un blanc.
L'exécution s'effectue dans le même environnement et donc avec modification de
l'environnement actuel.

Exemple:

{export LANG=france ; echo $LANG; } ; echo $LANG

Il est possible de rediriger les sorties de l'ensemble des commandes groupées.

Exemple:

{export LANG=france ; echo $LANG;} >fichier

Il est à noter qu'un " ; " existe obligatoirement avant l'accolade fermée; effectivement "}" et
vu par le shell comme une instruction .

3.9. Les opérateurs logiques

- L'opérateur d'exécution si échec || (ou logique).

Exemple:

more fichier || >fichier

affichage du fichier ou, si l'affichage est impossible, création du fichier.

- L'opérateur d'exécution si réussite && (et logique).

Exemple:

more fichier && echo "une ligne de plus" >fichier

affichage du fichier et, si l'affichage est possible, ajout d'une ligne dans le fichier.

Les caractères spéciaux

4.1. Les caractères connus

| le pipe
& lancement en arrière plan
|| ou logique
&& et logique
; fin de commande
› redirection d'une sortie
‹ redirection d'une entrée
2$# redirection d'une erreur
() groupage de commande dans un autre shell
{ } groupage de commande dans le même shell
` substitution de commande en bourne shell
$() substitution de commande en korn shell
\ inhibition

4.2. Les mètacaractères

Le symbole * sert à remplacer de 0 à plusieurs caractères.

Il est déconseillé d'utiliser le symbole * dans un nom de fichier.

Prêtez attention tout particulièrement à la commande rm qui est très dangereuse avec le symbole *.En effet, il existe un risque important d'effacer des fichiers importants de façon irrémédiable.Notez bien que cela n'arrive en générale qu'une fois.

Exemple:

*ls fichier**
#liste les fichiers dont le nom débute par fichier.

*ls *c*
#liste les fichiers dont le nom se termine par c.

*ls *t**
#liste les fichiers contenant t dans leur nom.

Pour remplacer un seul caractère par n'importe quel caractère le symbole ? est utilisé.

Il est déconseillé d'utiliser le symbole ? dans un nom de fichier.

Exemple:

ls -al fich?er

Pour remplacer un seul caractère par une liste de caractères les symboles [...] sont utilisés.

[246] choix possible entre le caractère 2, le 4 ou le 6
[1-3] choix parmi les caractères compris entre 1 et 3 dans l'ordre alphabétique.
[!135] tous sauf les caractères 1, 3 et 5.

Il est déconseillé d'utiliser les symboles [] dans un nom de fichier.

Exemple :
*ls [a-c]r?toto[!b]**
 # liste les fichiers commançant par a b ou c suivie de r suivie de n'importe quel caractère suivie de la chaine toto suivie de tous sauf b et se terminant par n'importe quel chaine (y comprie rien).

4.3. Les expressions

Un remplacement de 0 à n occurrences est réalisable par la symbolique :

*( )

Il peut être nécessaire d'avoir plusieurs expressions permettant d'avoir différente solution (logique ou).

symbolique: |

Un remplacement de 1 à n occurrences est réalisable par la symbolique :

+( )

Un remplacement de 0 à 1 occurrence est réalisable par la symbolique :

?( )

Un remplacement de une occurrence exactement est réalisable par la symbolique :

@( )

Un remplacement de toute les chaînes, sauf celles qui correspondent à une expression, est réalisable par la symbolique :

!( )

Exemple:

*ls -al *!(.[cho])*
liste tous les fichiers sauf ceux se terminant par .c, .o ou .h.

Les commandes interne

5.1. Généralités

Il s'agit de commandes contenues dans le programme KSH.Elles ne créent pas de processus et laissent l'environnement inchangé. Leur exécution est rapide.

5.2. Commande cd

CD : Change Directorie
cd : sert à ce déplacer dans l'arborescence.

* Particularités KSH :
Le KSH utilise des variables d'environnements facilitant le déplacement dans l'arborescence.
PWD : directorie courant .
OLDPWD : directorie précédent.

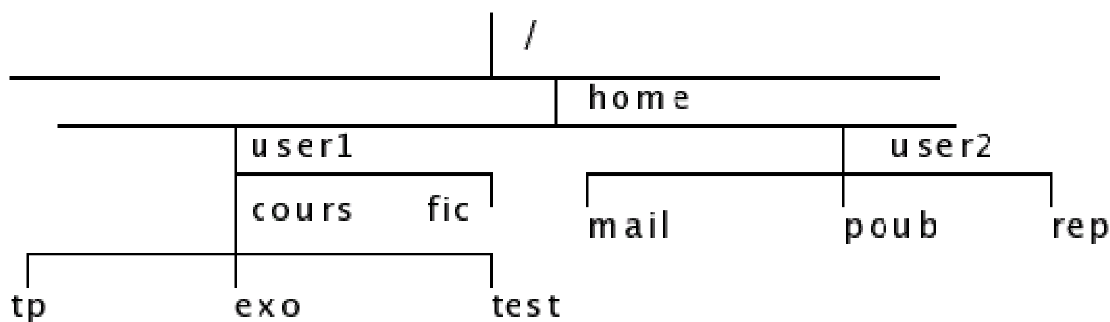
retour au directorie précédent : cd -

Exemple :

$ pwd
/home/masociété
$ cd /etc
/etc
$ cd -
$ pwd
/home/masociété

CDPATH : variable contenant des chemins d'accès aux répertoires et permettant l'accès direct.

Exemple :



$ export CDPATH=/home/user1:/home/user1/tp:/home/rep

$export S1='$PWD $'
/home/user1 $ cd tp
/home/user1/cours/tp $ cd rep
/home/user2/rep $ cd exo
/home/user1/cours/exo $

5.3. Processus

wait (n) permet de conditionner l'exécution du process père à la fin du process fils dont le PID est n.

kill permet d'émettre un signal vers un autre processus.

kill -signal PID

La liste des signaux est dans /usr/include/sys/signal.h
ou accessible via la commande kill -l

Il est possible d'envoyer des signaux par l'intermédiaire du clavier :
‹ CTRL C› signal 2 (INT)
‹ CTRL D› signal 3 (QUIT)
‹ CTRL Z› signal 17 (STOP)

Pour terminer un processus, il suffit de lui envoyer le signal -15 : kill -15 PID

La plupart des utilisateurs utilise le signal -9,
Le signal -9 est un kill sans condition n'y information au process en d'autre termes , il est impossible à un process de "trapper" le signal -9, donc de se terminer proprement (fermeture de fichier,fin de transaction,...), faite l'essai avec une base de donné transactionnel (oracle, postgres, ...) et vous êtes bon pour une restauration

trap associe une exécution de fichier à la réception d'un signal :

trap action signal

trap '$home/.finish' INT # exécution de .finish à la réception du signal INT

trap " " signal # désactive le signal

Conseil : utiliser les symboles des signaux plutôt que leur valeur.

5.4. Les limites du système

ulimit permet de visualiser ou de positionner les limites des ressources du système

Visualisation :
ulimit -a # toutes les limites du système (dépendent de l'*nix)

Positionnement :
ulimit -c ulimited (mise a taille ilimiter du fichier "core" (systeme linux) indispensable pour debugger !)
ou ulimit -c 150000 (mise à 150000 octects)

5.6. Droits sur les fichiers à leur création

Les droits de chaque utilisateur sur ses propres fichiers sont positionnés à la création du compte par la commande umask.

Ces droits peuvent être changés par l'utilisateur.

Visualisation :

$ umask
$ 022 # masque par défaut

Positionnement :

$ umask 004 # pas de droit de lecture pour les autres
$ umask 020 # pas de droit d'écriture pour le groupe

5.7. Les alias

alias permet de donner un synonyme à une commande ou à un groupement de commandes.

Cela permet de gagner du temps et de simplifier les commandes.
On peut mettre un certain nombre d'alias dans le fichier .profile (cf. personnalisation)

La syntaxe est la suivante : alias nom_alias = ' commande '

Exemple :

$ alias l='ls-al|more' # est une commande très #pratique.
$ alias goappli='cd /home/compte_appli/appli'

Pour obtenir la liste des alias définis : alias

5.8. La commande whence (type en bash)

whence (type) permet de connaître le chemin absolu d'une commande, d'un exécutable ou son type (option -v uniquement whence)

Exemple :

$ whence vi
/usr/bin/vi
$ whence -v pwd
pwd is shell built-in
En bash pour connaître le type du fichier c'est la commande : file

Les commandes externes

6.1. Généralités

Ces commandes sont des fichiers exécutables que l'on trouve dans /bin ;/usr/bin ;/etc; voir /sbin et /usr/local/bin

L'ordre de recherche correspond à celui mis dans la variable PATH des fichiers :

/etc/profile (/etc/bash_profile et/ou /etc/profile.d/*)
$HOME/.profile ($HOME/.bash_profile)

6.2. Find

find effectue une recherche récursive dans toute l'arborescence à partir du répertoire spécifié.

Syntaxe :

find répètoire critére_recherche commande

où répèrtoire est le répertoire de départ de la recherche : /toto; /; .
et critère_recherche est le critère de recherche des fichiers.

Cela peut être:

par nom -name toto

par user -user toto

par type -type f fichier, exécutable...

et bien d'autre (voir man find)

commande est la commande appliquée aux fichiers trouvés :

-print pour les imprimer à l'écran

-ls pour les lister

-exec .... pour leur appliquer des commandes plus complexes.

Exemples :

$ find . -user user1 -print

#à partir du répertoire courant afficher les fichiers et répertoire du user user1

*$ find / -type f -name *.core -exec rm {} \;*

*#à partir de la racine rechercher les fichiers *.core et les détruire, {} remplace les fichiers trouvés par le find et le \ protège le ; de l'interprétation du shell, en effet il est utilisé pour terminer le -exec*

*$ find / -name "*toto" -user toto -print;*

#à partir de la racine recherche des fichiers finissant par la chaîne "toto" et appartenant à l'utilisateur toto

$find / \(-type f -name "toto"\) -o \(-type d -name tata \)*
 # à partire de la racine recherche des fichiers de type ordinaire commanssant par la chaîne toto OU de type répèrtoire de nom "tata"

Noter l'usage des opérateurs logiques : -o pour le OU et -a pour le ET que l'on peut ne pas mettre
le \ avant la (ou) et obligatoire, en effet si l'on ne les mets pas c'est le shell qui va interpréter la (et nom pas la commande find !

6.3. Head, tail

head n fic
visualise sur la sortie standard les n premières lignes du fichier fic.

tail -n fic
visualise sur la sortie standard les n dernières lignes du fichier fic.

tail -f
fic permet de visualiser la fin du fichier fic. Si celui-ci croît ,l'affichage est mis à jour.

6.4. Grep

grep permet de rechercher une chaîne de caractères.

$ grep masociété * pour rechercher toutes les chaînes de caractères contenant masociété dans le répertoire courant.

Résultat :

fic1: masociété est aux services de ces clients.
fic2: ils seront référencés par le code : masociété_client

$ grep -l masociété * pour rechercher les fichiers contenant masociété dans le répertoire courant.

Résultat :
fic1:
fic2:

$ grep -w masociété * pour rechercher les fichiers ne contenant pas masociété dans le répertoire courant.

grep peut également être utilisé dans un pipe.

ps -eaf |grep ksh|wc -w pour retourner le nombre de processus Korn Shell tournant sur la machine +1, grep ksh étant lui aussi un processus.
pour eviter cette effet il suffit de rajouter en find de commande : | grep -v grep !
le ET logique et donc obtenue via une succéssion de | par contre grep ne sait pas réaliser le OU logique pour ceci utiliser la commande egrep

6.5. Sed

sed est un éditeur permettant de modifier un ou plusieurs fichiers.

syntaxe :
sed 's/ancienne_chaîne/nouvelle_chaîne/g liste_fichiers

Plusieurs substitutions en une fois :

ls -al |sed -e 's/rwx/7/g' \
-e 's/rw-/6/g' -e 's/r-x/5/g'\
-e 's/r--/4/g' -e 's/-wx/3/g'\
-e 's/-w-/2/g'-e 's/--x/1/g'

Cette commande affiche la liste longue des fichiers en remplaçant les permission rxx par leur valeur hexa.

On peut passer par l'intermédiaire d'un fichier.

$ cat substit
s/rwx/7/g
s/rw-/6/g
s/r-x/5/g
s/r--/4/g
s/-wx/3/g
s/-w-/2/g
s/--x/1/g

$ ls -al |sed -f substit

sed sait aussi ajouter ou suprimer, voir le man

6.6. Cut

cut extrait les colonnes ou les champs précisés.

par colonnes : option -c

ls -al |cut -c16-24,34-40

une colonne : un caractère, liste des propriétaires et la tailles des fichiers.

par champs : option -f, le séparateur est par défaut la tabulation, si l'o désire le changer option -d

exemple :

$ more cours
COURS nb_jour nb_participant
UNIX 3 10
KSH 2 5
ADM 5 10

$ more cours |cut -d=' ' -f1,3
COURS nb_participant
UNIX 10
KSH 5
ADM 10

6.7. Awk

awk est un outil très puissant et très utilisé sous UNIX, il permet de mettre en forme ,de modifier le contenu des fichiers.

awk traite chaque ligne du fichier référence, les champs séparés par un blanc, les nomme $1,$2..., ce qui facilite leur manipulation.

ls -al |awk '{print $3,$5}'
affiche les propriétaires et la taille des fichiers.

awk est un language complet orienté édition, et comporte bien d'autre possibilité, notement les fonctions, biblihotèque mathématique, ...
reportez vous au manuel

Les paramètres et les variables

7.1. Généralité

Il existe 2 types de variables shell :

les variables d'environnements utilisées pour configurer l'environnement de travail : PS1, PWD, HOME...

les variables utilisateurs définies pour ces propres besoins.

Les principales commandes de manipulation des variables sont :

- echo
- print
- export
- read
- set
- unset
- shift

Ces commandes sont internes aux shell et ne créent donc pas de sous shell

7.2. Manipulation des variables

L'affectation se fait par l'intermédiaire du signe =, on accède à la valeur de la variable par le métacaractères $.

```
$ societe=masociété
$ print $societe
masociété

$ a='bonjour à '
$ b='ce soir'
$ c=' 22 heure'
$ print $a $c
bonjour à 22 heures
$ print $a $b $c
bonjour à ce soir 22 heures
```

Attention : la concaténation de variable est possible avec la syntaxe suivantes :

```
print ${a}"à demain"
bonjour à demain

#La commande unset désaffecte une variable.
$ unset a
$ print $a
$
```

Attention le shell n'est pas capable de diferencier 2 variable noté comme suit :
a="var1"
ab="var2"
un print $a donnera :
var1
un print $ab donnera :
var1b

faire print ${ab} donnera le bon résultats

Il est possible d'affecter le résultat d'une commande à une variable.

Exemple :

```
$ nom_machine=$(hostname)
$ print $ nom_machine
masociété
$ print unix>cours #cours : fichier
$ var=$(‹mois) #var : variable
$ print $var
unix
$ ls -l fic*
fic1
fic2
fic3
```

$ liste=$(ls -l fic)*
$ print $liste
fic1 fic2 fic3
Remarque : dans ce cas les espaces, les tabulations et NL sont remplacés par un espace

Exportation de variables:

Les variables ne sont connues que du processus qui les ont créés, il est donc nécessaire de les transmettre aux processus fils de celui-ci.

La commande export permet aux variables d'être visibles par tous les processus fils.

$ variable=12
$export variable
$ export LANG=fr_FR

Mise en évidence des problèmes de visibilités de variables :

$ a=coucou
$ echo $a
$ coucou
$ ksh # lancement d'un shell fils
$ echo $a
$ # a n'est pas connu
$ a=voila
$ echo $a
$ voila
$ exit # on quitte le fils pour retourner au père
$ echo $a $ coucou # le fils n'as pas modifier la viariable du père

read permet de lire des données sur l'entrée standard, de les affecter à une variable ou à la variable d'environnement REPLY. IFS est utilisé comme séparateur lorsque plusieurs variables sont saisies à la fois

$ read jour
jeudi
$ print $jour
jeudi
$ read
nous sommes jeudi
$ print $REPLY
nous sommes jeudi

#Pour afficher un message avant de saisir une variable, on utilise read ?.

$ read a?"valeur de a : "
valeur de a : 1
$ print a=$a
a=1

#La concaténation des variables est possible par l'intermédiaire des accolades {} :

$ a=chaise;as=fauteuils
$ print nous avons 10 $as
nous avons 10 fauteuils
$ print nous avons 10 ${a}s
nous avons 10 chaises

Longueur d'une variable : ${#var}

$ print PATH a ${#PATH} caractères
PATH a 89 caractères

7.3. La substitution de variables

La substitution d'une variable par une autre dépend de son état :
- non définie
- définie mais vide
- définie et non vide.

De plus il existe 3 types de substitution :
- temporaire : signe -
- réelle, jusqu'à la prochaine modification : signe =
- impossible, protection : signe ?

7.3.1. Substitution temporaire

La modification n'est valable que pour la commande exécutée.

si la variable existe

${var-$sub} vaut var si var existe sub sinon
$ var='variable';sub='defaut'
$ print ${var-$sub}
variable
$ var=''
$ print ${var-$sub}
$ unset var
$ print ${var-$sub}
défaut

si la variable est non vide

${var:-$sub} vaut var si var est non vide sub sinon
$ var='variable';sub='defaut'
$ print ${var:-$sub}
variable

$ var=''
$ print ${var:-$sub}
défaut
$ unset var
$ print ${var:-$sub}
défaut

7.3.2. Substitution réelle =

La modification est valable pour toute la suite du processus.

si la variable existe

${var=$sub} #vaut var si var existe sub sinon
$ var='variable';sub='defaut'
$ print ${var=$sub}
variable
$ var=''
$ print ${var=$sub}
$ unset var
$ print ${var=$sub}
défaut

si la variable est non vide

${var:=$sub} #vaut var si var est non vide sub sinon
$ var='variable';sub='defaut'
$ print ${var:=$sub}
variable
$ var=''
$ print ${var:=$sub}
défaut
$ unset var
$ print ${var:=$sub}
défaut

7.3.3. Substitution impossible ?

Si la substitution est impossible, un message est affiché et on sort du fichier.

si la variable n'existe pas

${var?$sub} #affiche le texte puis stop
$more proc1
var='variable';sub='message'
print ${var?$sub}
var=''
print ${var?$sub}

unset var
print ${var?$sub}
$
$proc1
variable
proc1[6]: var:message

si la variable est vide
${var?$sub} #affiche le texte puis stop
$more proc2
var='variable';sub='message'
print ${var:?$sub}
var=''
print ${var:?$sub}
unset var
print ${var:?$sub}
$
$proc2
variable
proc1[3]: var:message

7.4. Les attributs de variables

La commande typeset positionne, réinitialise ou affecte les variables selon différentes options :

-L cadrage à gauche

$ var=MASOCIÉTÉ;print $var
MASOCIÉTÉ
$ typeset -L3 var ;print $var
MAS

-R cadrage à droite

$ var=MASOCIÉTÉ ;print $var
MASOCIÉTÉ
$ typeset -R4 var ;print $var
CIÉTÉ

-Z cadrage à droite et remplissage par des 0 à gauche

$ typeset -Z8 var=$(who | wc -l)
00000005

-conversion en majuscule -u ou en minuscule -l

$ var=masociété ; print $var
masociété

$ typeset -u var ; print $var
MASOCIÉTÉ
$ typeset -l var ; print $var
masociété

-protection d'une variable en lecture uniquement -r

$ typeset -r var
$ var=bonjour
ksh: var is read only
$ # autre méthode
$ readonly a
$ a=bonjour
ksh: a is read only

Si un accès en écriture est tenté dans un shell script, il est interrompu

- supprimer un attribut +

$ typeset -r var
$ var=bonjour
ksh: var is read only
$ typeset +r var
$ var=bonjour;print $var
bonjour

7.5. Les tableaux

Le KSH (Bash) permet de manipuler des tableaux uniquement à une dimension et d'au maximum 1024 éléments (de 0 à 1023).
Il est nécessaire de déclarer un tableau. Les attributs de variables s'appliquent à tous les éléments du tableau.

affectation :

#simple
$ tableau[1]=toto
$ a[4]=10
$
#multiple
$ set -A tab 1 2 3 4
#équivaut à
$ tab[0]=1
$ tab[1]=2
$ tab[2]=3
$ tab[3]=4
$
affichage :
$

$ print ${tableau[1]}
toto
$ print ${a[12]}
10
$ print ${tab[0]}

1 $ print ${a[*]} # affiche tout le tableau en même temps
1 2 3 4
$ print ${#a[*]} # affiche la taille d'un tableau
16
$ print ${#a[12]}
4

7.6. Les paramètres du shell

Voici la syntaxe d'un appel de script en KSH (Bash ou autres) :

commande arg1 arg2 arg3 ..... argn

à chaque champs de cette ligne est affectée un certain nombre de paramètres utilisables dans le fichier script.

$0 nom de la commande, du script
$1 premier argument
$2 deuxième argument
$n n ième argument
${n} n ième argument (n >9)
$# nombre d'argument excluant $0
$* tous les arguments
$@ tous les paramètres

Exemples:

affich argua argub arguc

$ more affich
print $0
print $1
print $3
print $#

$ affich argua argub arguc
affich
argua
arguc
3

#attention au-delà du 9 ième argument
$ print $1

```
argua
$ print $12
argua2
$ print ${12}
$
```

En shell (sh) au déla du 9eme argument il n'est plus possible de l'utilisé directement il faut passer par la comande shift

Autres paramètres disponibles :

$? valeur du code retour du dernier processus lancé (Rappel : un zéro indique que l'opération a réussi, un 1 ou autre, que l'opération n'a pas abouti)

$$ PID du process lancé par le script s'exécutant

$! PID du dernier process lancé en background

shift permet de décaler le contenu de ces paramètres et de décrémenter $#

```
$more fonc
print ' $# vaut : '$# et '$1 vaut : '$1
shift
print ' $# vaut : '$# et '$1 vaut : '$1
shift
print ' $# vaut : '$# et '$1 vaut : '$1
shift
print ' $# vaut : '$# et '$1 vaut : '$1
shift
print ' $# vaut : '$# et '$1 vaut : '$1
shift
$ fonc aa bb cc dd ee
$# vaut : 5 $1 vaut : aa
$# vaut : 4 $1 vaut : bb
$# vaut : 3 $1 vaut : cc
$# vaut : 2 $1 vaut : dd
$# vaut : 1 $1 vaut : ee
```

7.7. La commande set

La commande set permet d'affecter une variable d'environnement, (cf. set -o vi), mais aussi d'affecter à n'importe quelle variable les paramètres $1, $2,$ #...
La commande set sans argument permet de lister les variables définies, avec leurs valeurs .

Exemple :

```
$ set titi tata toto
$ print $2 $3
tata toto
```

$ print $1
titi
$set coucou
$ print $1 $2 $3
$ coucou #si ré-affectation les paramètres sont remis à vide
$
#affectation d'un résultat d'une commande aux paramètres $1, $2,$ #...
$
$ set $(hostname)
$ print le nom de la machine est $1
$ le nom de la machine est MaMachine

#affectation des tableaux set -A
$ set -A tab il fait beau
$ print ${tab[*]}
il fait beau

Le korn shell offre de nouvelles possibilités à cette commande set :

Set -o permet de lister les options et les paramètres du ksh

set -o option permet d'activer d'une option
ou set -L option

Exemple :

set -a #allexport permet d'exporter automatiquement chaque variable définie

set +o option #permet de désactiver l'option
#ou set +L option

Les tests

8.1. La commande test

Cette commande permet d'évaluer une expression selon la syntaxe suivante :
test expression
ou [expression]

Cette commande renvoie un 0 si l'expression est vraie, une valeur différente de 0 si l'expression est fausse (en général la valeur 1).

Le korn shell (Bash) possède une version améliorée qui accepte des opérateurs de test plus nombreux et des métacaractères non étendus, rendant la commande test moins utilisée

[[expression]] pour les expressions « chaines »
((expression)) pour les expression numériques
[[,]], ((,)) sont des commandes ! **donc l'espace avant n'est pas facultatif !**

8.2. Execution conditionnelle

exp1 && exp2 si exp1 est correcte alors on exécute exp2.

exp1 || exp2 si exp1 est incorrecte alors on exécute exp2.

Cas de la syntaxe [[?.]] :

[[exp1 && exp2]] cette expression est vraie si les deux expressions exp1 et exp2 sont vraies (la deuxième n'est évaluée que si la première est validée).

[[exp1 || exp2]] cette expression est vraie si aucune des deux expression n'est vraie.

[[!exp]] ceci permet l'inversion logique.

Idem pour la syntaxe ((?.))

8.3. Tests sur les fichiers

Il existe de nombreux opérateurs de test qui permettent d'examiner l'état d'un fichier suivant la syntaxe suivante :

test option nom_fichier
ou test nom_fichier1 option nom_fichier2

ou [option nom_fichier]

Les options peuvent être :

-f pour un fichier ordinaire existant
-d pour un repertoire
-s pour un fichier de taille supérieure à 0
-e pour fichier existant (fichier générique)
-x pour fichier existant et exécutable
.... voir le manuel

Le korn shell présente des tests supplémentaires souvent pratiques :

-o pour un fichier existant dont le propriétaire est l'ID utilisateur effectif
-s pour un fichier spécial socket

fichier1 -ef fichier2 pour tester si l'I-node du fichier1 est égale à l'I-node du fichier2

fichier1 -nt fichier2 pour tester si fichier1 est plus récent que fichier2

fichier1 -ot fichier2 pour tester si fichier1 est plus ancien que fichier2

8.4. Tests sur les chaînes de caractères

La syntaxe [[?]] du korn shell offre des expression de tests supplémentaires :

[[chaine1=chaine2]] pour tester si chaine1 est égale à chaine2

[[chaine1 !=chaine2]] pour tester si chaine1 n'est pas égale à chaine2

[[chaine1‹chaine2]] pour tester si la chaine1 est placée avant la chaine2 d'après le classement ASCII (codage des caractères *)

[[chaine1›chaine2]] pour tester si la chaine1 est placée après la chaine2 d'après le classement ASCII (codage des caractères *)

* les variables LC_TYPE LANGUAGE LC_MESSAGES LC_ALL LANG LESSCHARSET modifie l'ordre de comparaison des caractères donc des chaînes.

Les expressions arithmétiques et logiques

9.1. Les opérateurs

Les opérateurs arithmétiques :

+ : addition
- : soustraction
* : multiplication
/ : division
^ : puissance
% : modulo, reste de la division entière
() : priorité des calculs

Les opérateurs de comparaisons :

‹ : inférieur
› : supérieur
‹= : inférieur ou égal
›= : supérieur ou égal
== égal
!= différent

Les opérateurs logiques

() : priorité des tests
! : non logique
&& : et logique
|| : ou logique

9.2. Les priorités

1 : de gauche à droite
2 : les parenthèses
3 : * / %
4 : + - 5 : les comparaisons
6 : les opérateurs logiques

9.3. Conversions

Il est possible d'utiliser les différentes bases (décimale, octale, hexadécimale ou binaire) pour définir une variable arithmétique.
Pour cela on utilise la commande typeset de la façon suivante :
typeset -i8 y=4
#on définit y à 4 en octal.

typeset -i2 x=6
#on définit x à 6 en binaire.

Exemple de script de conversion d'un nombre décimal en héxadécimal :

 # script conversion (du type :convertion nombre)

 # !/bin/ksh
var=$1 #var correspond au premier et donc au seul
print « $var » #paramètre passé dans le script
typeset -i16 hexadecimal=$var
print « $hexadecimal »

9.4. La commande let ou (( ))

Les commandes let et (( )) sont équivalentes, elles servent à :

#affecter des variables numériques.
$ let x=1
$ ((x=4))
#faire des calculs
$ let x=x+1
$ ((i +=1))
#évaluer les tests numériques
$ (((x=x-1) ‹0))

remarques :
Le ksh ne peut gérer que des nombres entiers relatifs compris entre -2^31 et 2^31.
Pour les calculs en décimaux utiliser la calculatrice bc -l.
Les calculs peuvent ce faire dans n'importe quelle base entre 2 et 32

9.5. La commande bc (beautiful calculator)

bc est une calculatrice interactive, elle peut effectuer des calculs réels et des calculs complexes (Cos, Sin, exposant grâce à l'option -l). On sort du mode interactif de la calculatrice par ‹CTRL +D›

$bc -l

(12+21)/7
4.714285714286
‹CTRL D›

exemple d'usage :

$ print "(12+21)/7" | bc
$ 4 Exemples de fonctions :
sqrt (x) pour
s (x) pour sin(x)
c (x) pour cos(x)
e (x) pour exp(x)
l (x) pour ln(x)
a (x) pour arctan(x)
lenght (x) pour définir le nombre de chiffres significatifs
scale (x) pour le nombre de chiffres à droite du séparateur décimal

Les contrôles de boucles

10.1. Les structures courantes

Les structures du type

if - then -else - fi
Until et while, do ? done
For et case, do .. done

sont utilisables par le korn shell de la même manière que par le bourne shell.
Le korn shell offre la possibilité supplémentaire d'utiliser la syntaxe [[?]] dans l'élaboration des tests.

Exemples :

```
if [ -f mon_fichier ]
then
   print mon_fichier existe
else
   print mon_fichier inexistant
fi

while true
do
   ??boucle infinie
done

for i in fic1 fic2 fic3
```

do
ls -l $i
done
#pour lire un fichier texte ligne a ligne while read ligne
do
echo $ligne
done ‹le_fichier_a_lire

#pour effectuer une action sur la liste des arguments il suffit : for i in
do
ls -l $i
done

10.2. La structure select

La structure select permet de créer un menu de la façon suivante :

select x in option1 option2 option3
do
Commandes qui utilise l'identificateur x
(il s 'agit souvent de la structure case)
done

Un message d'invite à faire un choix est contenu dans la variable d'environnement PS3.

PS3= « choisissez une option »

Le menu, c'est à dire les choix possibles sont affichés automatiquement.

exemple:

#script emploi du temps

clear PS3= « choisissez une option numérique du menu »
select jour in lundi mardi mercredi jeudi vendredi fin
do
case $jour in
lundi) print réunion avec Monsieur X ;;
mardi) print déplacement à Paris ;;
mercredi) print repos ;;
Jeudi) print arrivée clients entreprise X ;;
vendredi) print réunion mise au point ;;
fin) exit ;;
**) print choisissez un jour de la liste SVP ; ;*
esac
exec $0 done

la commande 'exec $0' relance le shell permettant ainsi d'avoir un affichage correcte

Il est nécessaire de prévoir tous les cas possibles de choix. C'est pourquoi, on utilise toujours le cas *) qui prend en compte les choix différents de ceux décrit dans le menu. Si l'utilisateur tape une option non prévue dans le menu, on l'invite souvent à retaper une nouvelle option contenue dans le menu.

2.1. La modification du déroulement d'une boucle

Les commande break et continue permettent respectivement d'interrompre une boucle ou de coninuer celle-ci sans executer le bloc de commande suivant.
En Korn Shell (ou bash) il est possible d'indiquer avec ces commandes une sortie de boucle imbriquée :

```
while condition1
do
  while condition2
  do
    if (( condition ))
    then
      break 2
    fi
  done
done
```

Dans cette exemble le break sort de 2 niveaux de boucle soit sort de la 1ere boucle while (condition 1)

Les fonctions

Le korn shell, tout comme le Bourne shell offre la possibilité de définir des fonctions. Celles-ci doivent être déclarées avant leur utilisation.
Comme pour la commande exit, les fonction retourne un code de retour via la commande return.
comme un shell elles acceptent des paramètres qui seront référencés de la même façon ($1, $2 ....) , attention la visibilité de toutes variables déclarées dans la fonction (comme les paramètres de la fonction) et entre les { }.
La syntaxe de ces fonctions diffère suivant le bourne shell ou le korn shell :

Pour le Bourne shell

```
nom_fonc ()
{
commandes
}
```

Pour le korn shell

```
function nom_fonc
{
commandes
}
```

Le contrôle des processus

Un processus est caractérisé par son numéro de tâche. Cette caractérisation permet une grande liberté dans la manipulation et le contrôle des processus :

* jobs permet de dresser la liste des processus courants, ainsi que leur numéro de tâche.
jobs -l liste des numéros de processus et des numéros de tâche.
jobs -n liste des tâches interrompues ou terminées.
jobs -p liste d'un groupe de processus.

* bg exécute une tâche qui avait été suspendue en arrière plan.

* fg exécute en avant plan une tâche suspendue ou une tâche en arrière plan.

* CTRL Z permet d'arrêter un processus lancé en avant plan.

* kill permet d'arrêter n'importe quelle tâche à l'aide de son numéro processus (PID) ou son numéro tâche (job_id) en envoyant un signal spécifique au processus.
kill -l permet de dresser la liste des signaux définis.
kill -l $? permet d'afficher le signal qui a provoqué une erreur de sortie

L'éxécution : Appel de shells et de scripts

Il existe plusieurs moyen de faire appel à un nouveau shell et chacun de ces moyen possède ses propres spécifications.

sh interrompt le shell en cours et commence un nouveau shell.

Cet appel possède différentes options telles que :
-r qui démarre un shell restreint.
-i qui démarre un shell interactif.
-s qui permet de lire des commandes en entrée standard.
-c qui démarre un shell non interactif.

exec sh interrompt le shell courant et commence un nouveau shell.

. sh éxécute le shell dans l'environnement du shell courant (pas de création de fils et partage des variables d'environnement)

Pour lancer un script, aprés avoir changé les droits d'éxécution (chmod +x script), les appels reste les mêmes

Un shell accepte un script en arguments : sh script éxécute le script.