



Script UNIX

Edition:

2002-2003

Responsable:

Nacéra Bennacer

Table des matières

1. UNIX et Scripts	3
1.1. Initialisation	4
1.2. Les variables d'environnement.....	4
1.3. Les caractères spéciaux du shell	5
1.3.1. La génération des noms de fichiers.....	5
1.3.2. Les autres caractères spéciaux.....	5
1.4. Le passage des paramètres	6
1.5. Les entrées sorties	6
1.6. Les structures de contrôle.....	6
1.6.1. La commande <i>test</i>	6
1.6.2. La commande <i>expr</i>	7
1.6.3. La condition <i>if</i>	7
1.6.4. La boucle <i>until</i>	7
1.6.5. La boucle <i>while</i>	7
1.6.6. La condition <i>for</i>	7
1.6.7. La condition d'aiguillage <i>case</i>	7
1.6.8. Les directives <i>break</i> et <i>continue</i>	8
1.7. Les fonctions.....	8
1.8. Les alias.....	8
1.9. Le job control: <i>jobs</i> , <i>bg</i> , <i>fg</i> , <i>kill</i>	8
2. Les principales commandes UNIX	10
3. Exercices avec corrigé sur le shell (Mir:/corriges/SYSTEME/SCRIPT).....	13
4. Exercices en complément.....	15

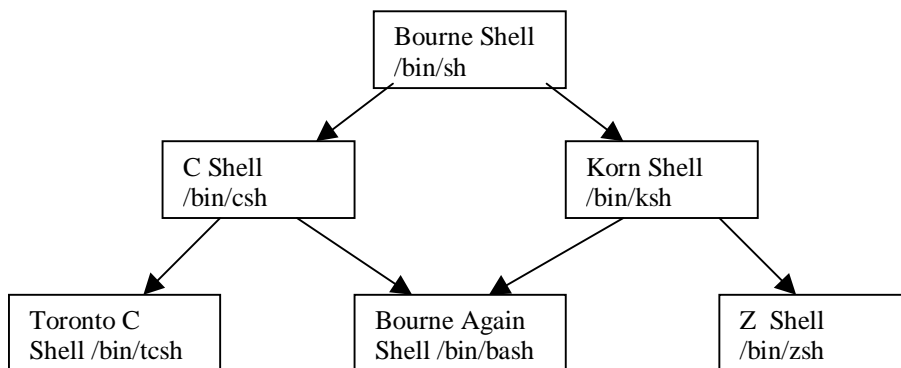
1. UNIX et Scripts

Le shell est un langage de programmation interprété autorisant la récursivité; mais c'est avant tout l'interpréteur standard de UNIX. Lors de l'ouverture d'une session, appelée login sous UNIX, un processus shell est créé qui permet à l'utilisateur de dialoguer avec le système. Trois types de commandes sont exécutées par le shell:

- Les commandes **externes** qui sont contenues dans des répertoires définis dans la variable d'environnement PATH. Dans ce cas l'exécution de la commande est réalisée par un processus dédié créé par le processus shell.
- Les commandes **internes** qui font partie du shell dont l'exécution ne nécessite pas la création de nouveaux processus. Ces commandes peuvent être scindées en deux:
 - Les commandes d'usage fréquent: cd, pwd, umask, set, ...
 - Les commandes composées de directives algorithmiques: if, then, else, while, case, for, ...
- Les commandes définies par une **fonction** ou un **alias**

(*) Priorité des commandes de même nom: d'abord les fonctions ensuite les commandes internes et enfin les commandes externes

Le Bourne Shell est le premier shell créé et il reste celui dont tout système Unix a besoin pour lancer les outils de configuration au démarrage de la machine. ksh, bash et zsh sont plus enrichis mais restent compatibles avec le sh. Le csh et tcsh ont des syntaxes plus différentes de sh et très proches de celle du langage C. L'important est qu'au moins un shell compatible avec sh existe sur une machine UNIX.



1.1. Initialisation

Le shell exécute successivement deux fichiers avant d'afficher son caractère d'invite (par défaut \$): /etc/profile et le fichier .profile (se trouvant dans le répertoire HOME de l'utilisateur). Si la variable ENV est définie (par un login korn shell) et si son contenu est la référence d'un fichier existant; ce fichier est exécuté.

Exemple de .profile:

```
ls()                                #fonction ls() redéfinit ls
{
  /bin/ls -al $*
}
rm()                                 #fonction rm() redéfinit rm
{
  /bin/rm -i $*
}
logout()                             #fonction logout() redéfinit exit
{
  exit                               #commande interne de terminaison d'un shell
}
PS1="Mir[$LOGNAME]"                 #invite du shell
set -o vi                            #édition de l'historique en mode vi
alias p='ps -ef |sort |pg'           #alias redéfinissant ps
Export PS1                            #exportation de la variable d'environnement PS1
```

Il est très utile après une modification de .profile de faire prendre en compte son contenu sans fermer et réouvrir une session. Ce lancement force l'exécution par le shell courant.

\$. .profile

Ce tableau récapitule les différentes façons pour lancer un script shell:

Commande	Conditions sur ref	Effet
ksh ref	Fichier lisible	Interprétation par un sous-processus ksh
ref ou ./ref	Fichier lisible et exécutable	Interprétation par un sous-processus ksh
. ref	Fichier lisible	Interpretation dans le ksh courant
exec ref	Fichier lisible et exécutable	Recouvrement du ksh par un ksh non interactif interprétant le fichier

Les versions récentes prennent en compte les premiers caractères de la procédure:

#!/bin/sh ou#!/bin/csh suivant le shell utilisé.

1.2. Les variables d'environnement

Elles sont utilisées par le shell ou le système. Certaines sont initialisées automatiquement et ne peuvent être modifiées: HOME, LOGNAME ou USER. D'autres sont initialisées automatiquement et peuvent être modifiées: PATH, TERM, SHELL. D'autres sont initialisées par l'utilisateur dans le fichier .profile.

L'utilisateur a la possibilité de définir d'autres variables. Ces variables ne se déclarent pas et elles sont locales au shell. Si on lance un autre shell la variable n'existe plus. La commande **set** permet de visualiser toutes les variables d'un shell.

NomVar=valeur affecte valeur à NomVar
 \$NomVar représente le contenu de la variable.
 export NomVar **export** permet de répercuter une variable sur l'arborescence descendante (shell fils).

Exemple:Affectation des variables

```
$ var1=abc
$ var2="abc*toto"
$ var3=abc*toto
```

Affichage de variables

```
$ echo $var1 $var2 $var3
abc abc*toto abc*toto
```

Portée du métacaractère \$: utilisation de {}

```
$ var=toto
$ echo $vartiti
```

rien

```
$ echo ${var}titi
tototiti
```

1.3. Les caractères spéciaux du shell

1.3.1. La génération des noms de fichiers

?	un caractère quelconque
*	une chaîne de caractères (éventuellement vide)
[début de définition d'un ensemble
[!]	début de définition du complément de l'ensemble
]	fin de définition d'un ensemble ou de son complément
-	marque d'intervalle dans un ensemble

Les caractères . en début de mot ou après un / ou / ne sont pas couverts par les abréviations. Les expressions construites à partir des caractères * ? [] sont appelées des expressions régulières.

[af69R] un caractère de la liste énumérée
 [A-T] un caractère de l'intervalle alphabétique

1.3.2. Les autres caractères spéciaux

#	Introduit un commentaire
\$	Introduit un nom de variable
&	Termine une commande lancée en arrière plan
;	Sépare des commandes se trouvant sur une même ligne
< > << >>	Caractères de redirection
` `	Exécute une commande
' '	Délimite une chaîne de caractères où les caractères spéciaux perdent leur signification sauf lui-même
" "	Délimite une chaîne de caractères où les caractères spéciaux perdent leur signification sauf \ \$ ` "
\	Désépécilise le caractère qui le suit
()	Exécuter une commande dans un shell fils
{ }	Regroupe des commandes

1.4. Le passage des paramètres

Le shell reçoit les arguments sur la ligne de commande sous la forme de variables **\$1 \$2 \$3 ... \$9**. **\$0** est le nom de la commande. La commande interne `shift` permet de décaler les arguments d'une position.

p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14
\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9					

⇒ **shift**

p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14
	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9				

Les autres variables du script correspondant à ces paramètres sont :

- \$#** le nombre de paramètres passés au script
- \$*** la liste des paramètres passés au script
- \$\$** le PID du processus exécutant la commande
- \$_** le PID du dernier processus asynchrone
- \$_?** le dernier status

1.5. Les entrées sorties

La commande `read` comme la commande `echo` peut recevoir plusieurs arguments séparés par des espaces ou tabulations.

1.6. Les structures de contrôle

1.6.1. La commande `test`

`test` expression ou [expression]

Cette commande renvoie un code de retour égal à 0 si l'expression est vraie et une valeur différente de 0 dans cas contraire.

- [**-a** fichier] existence du fichier
- [**-s** fichier] existence du fichier + taille non nulle
- [**-d** fichier] existence + état de répertoire
- [**-f** fichier] existence + état de fichier
- [**-r** fichier] existence + autorisation de lecture
- [**-w** fichier] existence + autorisation d'écriture
- [**-x** fichier] existence + autorisation d'exécution
- [**-n** chaîne] chaîne non nulle
- [**-z** chaîne] chaîne nulle

Les chaînes de caractères

- [\$var = chaîne] égalité
- [\$var != chaîne] différence

Les expressions numériques

- [\$var = valeur]

[\$var != valeur]

[\$var -opérateur valeur]

opérateurs possibles :

-lt <

-le ≤

-gt >

-ge ≥

1.6.2. La commande *expr*

Pour évaluer une expression numérique contenant une variable, il faut utiliser la commande **expr**. Les opérateurs autorisés sont + - * / %.

exemple : [`expr \$var + 1` -gt 5]

a= `expr \$var + 1`

remarque : **expr \$var : '*'** renvoie le nombre de caractères de var

1.6.3. La condition *if*

if expression

then

commandes

else

commandes

fi

1.6.4. La boucle *until*

until expression

do

commandes

done

1.6.5. La boucle *while*

while expression

do

commandes

done

1.6.6. La condition *for*

for variable **in** liste de valeurs

do

commandes

done

1.6.7. La condition d'aiguillage *case*

case variable **in**

cas1) commandes;;

cas2 | cas3) commandes;;

...

Casn) commandes;;

esac

Les cas sont conformes à la syntaxe d'expressions régulières

1.6.8. Les directives **break** et **continue**

Ces deux directives peuvent être utilisées à l'intérieur des boucles `for`, `while` et `until`. Elles peuvent être suivies d'un nombre qui correspond au nombre de niveaux d'imbrication pour lequel les directives sont appliqués.

1.7. Les fonctions

Il est possible de déclarer une fonction dans le shell interactif ou à l'intérieur d'un script shell. Elles peuvent renvoyer un code de retour. A l'aide de la commande interne `return`. La syntaxe:

`Nom_fonction() { liste de commandes }`. Exemple:

```
isfile() {
    if [ -f $f ]
    then
        return 0
    else
        return 1
    fi
}
for i in $*
do
    f=$i
    if [ isfile ]
    then
        echo "$i est un fichier"
    else
        echo "$i n'est pas un fichier"
    fi
done
```

1.8. Les alias

En `ksh`, on peut renommer une commande:

```
alias nom=valeur
unalias nom
```

La commande `alias` sans arguments liste les alias déclarés. L'option `-t` liste ceux créés automatiquement lorsqu'on utilise des commandes externes.

1.9. Le job control: **jobs**, **bg**, **fg**, **kill**

Ce mécanisme permet aux utilisateurs de ne voir que les processus lancés sous le shell interactif. Chaque commande analysée par le shell s'appelle tâche ou job. Il peut s'agir d'un groupe de processus. Toute tâche possède un numéro. L'avantage d'un tel mécanisme est que la consultation des tâches peut se faire sans solliciter le noyau (`ps` est gourmand). De plus le numéro d'identification est un petit nombre. Le Korn shell permet le contrôle des processus d'une manière fine.

\$ jobs

Cette commande fournit la liste des tâches créées par le shell, sous la forme :

```
[ 1 ] - état tâche1
```

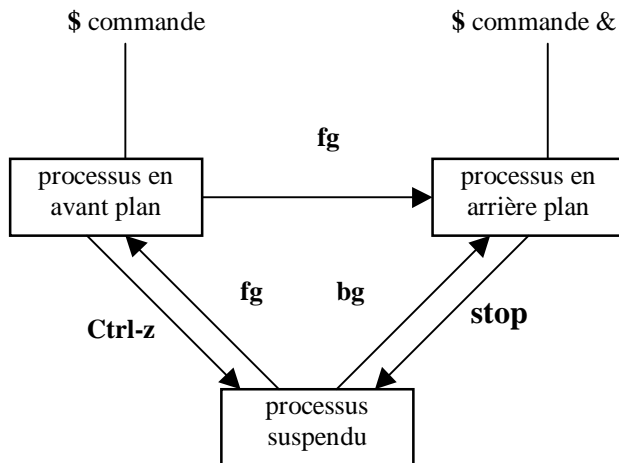
[2] + état tâche2

Le caractère + signale la tâche courante, - la tâche courante précédente.

\$kill tâche

Pour tuer le processus, le job peut être désigné de plusieurs façons

%nombre	Le nombre entre crochets
%chaîne	Le job dont le nom commence par chaîne
%?chaîne	Le job dont le nom contient chaîne
%% ou %+	Le job courant
%-	Le job précédent



2. Les principales commandes UNIX

Gestion de session, de compte, informations générales:

login	création d'une session de travail
passwd	changer le mot de passe
logname	afficher le login de l'utilisateur
who [am i]	liste des utilisateurs connectés à la machine
id [a,u,g]	numéros et les noms d'identification de l'utilisateur et de son groupe
groups	liste des groupes auxquels appartient l'utilisateur
uname[-a]	nom de la machine, nom du système d'exploitation
tty	afficher la référence absolue du terminal associé à l'entrée standard
stty	visualiser et modifier la valeur de paramètres de communication entre le système et le terminal
clear	effacer l'écran du terminal
date	date du jour et l'heure courante
cal i j	calendrier du mois i (de 1 à 12) de l'année j (de 1 à 9999) par défaut le mois de l'année courant
du	fournit l'espace alloué (en nombre de blocs de 512 octets) aux différents fichiers
quota [-v]	indique à l'utilisateur s'il est contrôlé par quotas et dans l'affirmative les valeurs limites

Gestion des processus:

ps [-e, a, d, g, p, t, u]	fournit des informations sur les processus en cours
kill	envoie un signal à un processus

Gestion de fichiers:

pwd	référence du répertoire de travail
cd	changement de répertoire de travail (~, .., ..)
ls [-l,a,i,d]	contenu d'un répertoire ou caractéristiques d'un fichier
cat	contenu d'un fichier
more	contenu par page
touch	modifier les dates de dernier accès en lecture et/ou écriture
cp[-i, r]	copie physique d'un fichier
cp ref_source ref_dest	(les deux fichiers référencent un fichier ordinaire)
cp ref_source1 ... ref_dest	(les sources référencent des fichiers ordinaires et la destination référence un répertoire)
cp -r ref_source1 ... ref_dest	(si l'un des sources et la destination référencent un répertoire)
ln	création de liens
ln ref_source ref_dest	(les deux fichiers référencent un fichier ordinaire)
ln ref_source1 ... ref_dest	(les sources référencent des fichiers ordinaires et la destination référence un répertoire)
mv	changement d'un nom de lien
mv ref_source ref_dest	(les deux fichiers référencent un fichier ordinaire)
mv ref_source1 ... ref_dest	(les sources référencent des fichiers ordinaires et/ou répertoires et la destination référence un répertoire)
rm [-r]	suppression de liens
mkdir	création d'un répertoire
rmdir	suppression d'un répertoire
chmod, chown, chgrp	modification de caractéristiques d'un nœud

D'autres commandes:

sort	trier un fichier
uniq [-c, -u]	comparer des lignes de fichiers et ne conserver qu'un exemplaire de chaque ligne
paste	juxtaposer des lignes correspondantes de deux ou plusieurs fichiers
tr	copie sur la sortie standard les caractères lus sur l'entrée standard après avoir réalisé des substitutions ou des suppressions de caractères définies dans les arguments
compress	compression de fichiers
uncompress	décompression de fichiers

find rechercher dans un répertoire les fichiers d'un nom, d'un type, d'un user, ...
file classer le fichier ASCII, binaire, répertoire, source C, ...

fc -l[r, n] liste le contenu du fichier .history.
fc -s chaîne réexécute la commande la plus récente commençant par chaîne
 par défaut de chaîne, la dernière commande est exécutée.

Sous ksh: l'alias **r** est défini

r p (numéro de la commande ou une chaîne) pour rappeler la commande

Fichiers administration utilisateurs:

Le fichier **/etc/passwd** a la structure suivante:

nom_login:x:UID:GID:commentaire:chemin_repertoire_login:chemin_shell

Le fichier **/etc/group** a la structure suivante:

nom_group::GID:liste_autres_utilisateurs_ayant_accès

Opérations sur les contenus de fichiers:

Sélection "verticale"

head [-n] fichier

Écrit sur la sortie standard les n premières lignes du fichier

tail [-nlcb] fichier

Écrit sur la sortie standard les n dernières lignes ou les n derniers caractères ou les n derniers blocs.

grep [-v] pattern fichier

Récupère les lignes du fichier contenant pattern (une expression régulière). L'option **-v** permet de récupérer les lignes du fichier **ne** contenant **pas** le pattern. **^pattern** permet de spécifier une recherche en début de ligne.

pattern\$ permet de spécifier une recherche en fin de ligne.

Sélection "horizontale"

cut [-ddélimiteur -fi,j] fichier

Récupère les champs i et j séparés par le délimiteur indiqué après **-d**.

cut [-ci-j] fichier

Récupère les caractères i à j de chaque ligne : option **-c**.

Comptage

wc [-lwc] fichier

wc -l compte le nombre de lignes

wc -w compte le nombre de mots

wc -c compte le nombre de caractères

wc équivalent à **wc -l**