Cours UNIX

Chapitre 4

Interpréteurs de commandes

Généralités

Les interpréteurs de commandes, appelés *shells*, permettent à un utilisateur de lancer des commandes et d'exécuter des scripts écrits dans un *langage de commandes* spécifique à chaque shell. Pour cela, ils ont des fonctionnalités plus ou moins évoluées comme la combinaison de commandes, le contrôle des processus, les variables, les alias, l'historique des commandes, ...

Nous allons voir dans ce chapitre uniquement les fonctionnalités permettant d'exécuter des commandes, de manipuler l'environnement de travail et de contrôler les processus. La partie concernant les scripts fait l'objet d'un chapitre spécifique (Chapitre 5).

Il existe des dizaines d'interpréteurs de commandes sous UNIX, les plus connus sont:

- La famille Bourne-Shells: `sh` (Bourne-Shell), `ksh` (Korn-Shell), `bash` (Bourne-Again-Shell)
- La famille C-Shells: `csh` (C-Shell), `tcsh` (Turbo-C-Shell)

Tous les systèmes UNIX sont au moins fournis avec le Bourne-Shell `sh`, ou du moins avec un shell compatible. Les versions BSD utilisent, en règle générale, le C-Shell comme shell standard, les versions Système V utilisent plutôt le Korn-Shell, alors que la plupart des distributions basées sur Linux utilisent le Bourne-Again-Shell.

Vous pouvez consulter cette page: http://www.faqs.org/faqs/unix-faq/shell/shell-differences/ qui résume les différences entre certains shells.

Il faut bien garder à l'esprit que ce que nous allons voir ne s'applique pas ou, du moins, pas de la même façon, à tous les shells. Lorsque nous aborderons une fonctionnalité ou une syntaxe spécifique à une **famille** de shell, ce sera précisé comme ceci: [Bourne-Shell] pour `sh`, `ksh` et `bash` et [C-Shell] pour `csh` et `tcsh`.

Caractères spéciaux, expansions

➔ Caractères spéciaux

#	Début d'un commentaire, qui se termine à la fin de la ligne.
\	(*antislash*) Caractère de protection qui enlève à un caractère spécial, lorsqu'il est placé devant, sa fonctionnalité et en fait un caractère normal.
' et "	(*quote* et *double quote*) Délimitation d'une chaîne. En règle générale, dans une chaîne délimités par ', les substitutions de variables ne sont pas faites.
`	(*antiquote* ou *backquote*) Interprétation d'une commande. La résultat de la commande est substituée à la chaîne. Exemple: `` `pwd` `` est substitué par le chemin courant, par exemple `'/home/toto'`.
~	(*tilde*) Répertoire de l'utilisateur, par défaut l'utilisateur courant. Par exemple `~toto` renvoie `'/home/toto'`.
.	[Bourne-Shell] Suivi par un espace, il exécute le script dont le nom suit. Exemple: `. ~/foo` exécute le script `foo` qui se trouve dans le répertoire de l'utilisateur.

➔ Expansions

Les caractères d'expansions sont utilisés par les shells pour construire des listes de noms de fichiers, lors de l'exécution d'une commande:

?	Remplacé par un caractère quelconque
*	Remplacé par plusieurs caractères quelconques (éventuellement vide)
[ ]	Définition d'un ensemble de caractères
!	Négation dans un ensemble de caractères
-	intervalle dans un ensemble de caractères

Quelques exemples:

```
[a-z]?[0-9].c
/etc/rc[!0-9].d
/*bin
```


Substitutions et variables

➔ Alias

La plupart des shells permettent l'utilisation d'alias qui sont des 'raccourcis' de commandes, avec leurs arguments. La commande `alias` permet de créer et de modifier un alias ou de lister tous les alias. La commande `unalias` permet de supprimer un alias.

Exemples:

[Bourne-Shell]

```
# alias rm='rm -i'
# alias cou='echo CouCou !'
# alias
alias rm='rm -i'
alias cou='echo CouCou'
# cou
CouCou
# unalias cou
# cou
bash: cou: command not found
```

[C-Shell]

```
% alias
% alias rm rm -i
% alias cou echo CouCou
% alias
cou     (echo CouCou)
rm      (rm -i)
% cou
CouCou
% unalias cou
% cou
cou: Command not found.
```

➔ Historique des commandes.

L'historique est un mécanisme permettant de rappeler les commandes (avec leurs arguments) déjà tapées par l'utilisateur. La plupart des shells modernes, lorsqu'ils sont correctement configurés, associent les touches de curseurs haut/bas au défilement de l'historique des commandes.

Le caractère ! (en particulier pour le C-Shell et le Bourne-Again-Shell) permet de rappeler une commande antérieure (il peut se trouver n'importe où dans la ligne), et la commande `history` affiche la liste des commandes mémorisées.

➔ Complétion des noms de fichiers

La complétion est un mécanisme d'aide à la frappe. Il permet de compléter les noms de fichier à partir d'un préfixe. Suivant la configuration (et le type de shell), la complétion peut se faire en appuyant deux fois sur la touche ESCAPE (Certains C-Shell) ou une fois sur la touche TABULATION (C-Shell récents et Bourne-Shell récents).

➔ Variables

Comme dans les langages de programmation, le *langage de commande* interprété par un shell permet de manipuler des variables identifiées par un nom.

Il existe deux types de variables:

1. Les variables locales, qui ne sont accessibles que pendant la session du shell ou que dans le scripts en cours d'exécution.
2. Les variables d'environnement, qui sont transmises à toutes les applications lancées depuis le shell (qui deviennent toutes des processus fils du processus shell).

En règle générale, la substitution d'une variable par son contenu se fait par:

`$variable` ou `${variable}`

La commande `set` (sans arguments) visualise la liste des variables locales et la commande `env` la liste des variables d'environnement.

Il existe, avec certains shells, des variables spéciales:

$0 Nom du fichier en cours d'exécution.

$n Argument n de la ligne de commande.

$$ PID du shell en cours.

[Bourne-Shell]

L'affection d'une valeur à une variable locale se fait avec l'opérateur = :

```
VAR=valeur
```

La commande `export` affecte une valeur à une variable d'environnement:

```
export VAR=valeur
```

ou

```
VAR=valeur; export VAR
```

Et la commande `unset` supprime une variable (locale et d'environnement) :

```
unset VAR
```

[C-Shell]

L'affection d'une valeur à une variable locale se fait avec la commande `set` :

```
set VAR=valeur
```

La commande `unset` supprime une variable locale:

```
unset VAR
```

L'affection d'une valeur à une variable d'environnement se fait avec la commande `setenv` :

```
setenv VAR valeur
```

La commande `unsetenv` supprime une variable d'environnement :

```
unsetenv VAR
```

De plus, il existe un certains nombre de variables d'environnement prédéfinies. En voici quelques exemples (liste variable suivant les shells et les systèmes):

HOME	Répertoire de l'utilisateur.
PATH	Répertoires de recherche pour le lancement des commandes.
SHELL	Chemin et nom de fichier du shell.
TERM	Type de terminal en cours d'utilisation.
PWD	Chemin courant (répertoire de travail).
DISPLAY	Nom et numéro du terminal graphique X-Window.
EDITOR	Éditeur à lancer si besoin est.
USER	Nom de l'utilisateur courant.
GROUP	Groupe principal de l'utilisateur.
PS1	Invite de commande principal.
UID	UID de l'utilisateur courant.

Exemples:

[Bourne-Shell]

```
# set
PWD=/opt/home/michelon
PS1=#
FTP_PROXY=cache:3128
...
# mavar=CouCou
# echo $mavar
CouCou
# unset mavar
# echo ${TERM}
xterm-color
```

[C-Shell]

```
% env
PWD=/home/michelon
COLORTERM=rxvt-xpm
LANG=fr_FR.ISO-8859-1
...
% set mavar=Hop!
% echo ${mavar}
Hop!
% unset mavar
% echo $DISPLAY
:0.0
```


Lancements de processus et redirections

Les différentes syntaxes de lancement de commandes sont (ces syntaxes peuvent être associées entre elles):

commande

Exécution d'une *commande*. L'exécutable ou le script correspondant est recherché dans les répertoires listés dans la variable d'environnement PATH, et le shell attends la fin de l'exécution pour rendre la main à l'utilisateur.

(commande)

Un nouveau shell est lancé et il exécute la commande.

commande1; commande2; ...; commanden

Les différentes commandes sont exécutées les unes après les autres.

commande &

La commande est exécutée en arrière plan: le shell n'attend pas la fin de l'exécution pour rendre la main à l'utilisateur.

commande < fichier

Lancement de la commande avec redirection du contenu du *fichier* vers l'entrée standard du processus lancé.

commande > fichier

Lancement de la commande avec redirection de la sortie standard du processus lancé vers le *fichier*.

commande >> fichier

Lancement de la commande avec redirection de la sortie standard du processus lancé vers le *fichier*. Le contenu du fichier n'est pas écrasé et les nouvelles données sont rajoutées à la fin.

commande1 | commande2 | ... | commanden

Toutes les commandes sont lancées en même temps, la sortie standard du processus lancé par la *commande1* est redirigé vers l'entrée standard du processus *commande2*, la sortie standard du processus lancé par la *commande2* est redirigé vers l'entrées standard du processus suivant, et ainsi de suite...

commande << IDENT

...

...

IDENT

Les lignes comprises entre << IDENT et IDENT sont redirigées vers l'entrées standard du processus lancé par la *commande*.

Exemples

```
# (echo Le Texte | tr -d e ) > resultat
# cat resultat ; sleep 2
L Txt
# wc < resultat &
[1] 61716
#        1       2       6
# sort << EOF
> eee
> rrr
> zzz
> aaa
> EOF
aaa
eee
rrr
zzz
```


Contrôle des processus

Certains shells possèdent des mécanismes et des commandes permettant de gérer les processus lancés depuis la ligne de commande. Chaque processus lancé est appelé un *job* dans la terminologie des shells.

Un processus peut être dans un des états suivants:

- *Exécution en premier plan*. C'est le dernier processus que l'utilisateur a lancé, c'est aussi celui dont l'entrée standard est redirigée depuis le clavier. Il peut changer d'état pour devenir un *processus suspendu*, à l'aide des touches CTRL-Z ou être arrêté à l'aide des touche CTRL-C (envoi du signal SIGINT) ou CTRL-\ (envoi du signal SIGKILL).
- *Exécution en arrière plan*. Le processus à été lancé par une commande finissant par la caractère &, ou il à été *suspendu* puis mis en arrière plan manuellement. Il peut devenir un *processus en premier plan* ou un *processus suspendu*.
- *Suspendu*. Le processus a été suspendu à l'aide des touche CTRL-Z ou avec la commande `stop`. Un *processus suspendu* est forcément en arrière plan, et peut devenir un processus en exécution *en arrière plan* ou un processus en *premier plan*.

Lorsqu'un processus passe en arrière plan ou est suspendu, le shell lui alloue un *numéro de job*. Ces numéros sont affichés à l'écran comme ceci: `[n]` et sont manipulés comme ceci: `%n`.

Voici les commandes internes aux shells permettant de manipuler les processus et les faire changer d'état (ici [] signifie que les paramètres ou les options sont facultatifs):

Liste des processus en arrière plan ou suspendus:

```
jobs [-l]
```

L'option -l permet l'affichage des PID en plus.

Le processus dont le nom (ou le PID avec -l) est précédé du signe + est le *processus en cours*: c'est ce processus là qui est traité lorsque l'on ne précise pas de *numéro de job (*avec une des commandes que l'on est en train de décrire), et que l'on peut aussi identifier par `%%` (en plus de son numéro `%n`).

Tuer un processus en arrière plan ou suspendu:

```
kill [%n]
```

Faire d'un processus en arrière plan ou suspendu le processus en premier plan:

`fg [%n]` ou `%n`

Faire d'un processus suspendu un processus en arrière plan:

```
bg [%n]
```

Faire d'un processus en arrière plan un processus suspendu:

```
stop [%n]
```

Exemples avec `tcsh` (^Z et ^C correspondent respectivement à la frappe de CTRL-Z et de CTRL-C):

```
% sleep 5000
^Z
Suspended
% sleep 10000 &
[2] 79893
% jobs
[1]  + Suspended                    sleep 5000
[2]  - Running                      sleep 10000
% bg %1
[1]    sleep 5000 &
% jobs -l
[1]    79892 Running                      sleep 5000
[2]  + 79893 Running                      sleep 10000
% stop %%
[2]  + Suspended (signal)           sleep 10000
% jobs
[1]    Running                      sleep 5000
[2]  + Suspended (signal)           sleep 10000
% kill %1
[1]    Terminated                   sleep 5000
% jobs
[2]  + Suspended (signal)           sleep 10000
% fg
sleep 10000
^C
%
```


Fichiers de configuration

Pour tous les shells, il existe deux fichiers de configuration:

- Le fichier de configuration exécuté lors du login, lorsque le premier shell est lancé, juste après l'ouverture de session.
- Le fichier de configuration exécuté à chaque nouveau "sous-shell" lancé depuis le shell de login.

Chacun de ces fichiers peuvent être:

- Globaux au système.
- Spécifiques à l'utilisateur (ils se trouvent alors dans le répertoire de l'utilisateur).

Les noms de ces fichiers sont différents pour chaque shell:

[Bourne-Shell]

- Fichier de login global: `/etc/profile`
- Fichier de login utilisateur: `~/.profile`
- Fichier de sous-shell utilisateur: dépend de la version. Pour bash, c'est `~/.bashrc`

[C-Shell]

- Fichier de login global: `/etc/csh.login`
- Fichier de login utilisateur: `~/.login`
- Fichier de sous-shell utilisateur: `~/.cshrc` ou `~/.tcshrc`

Ces fichiers sont en fait des scripts, mais il ne contiennent, en général, que des définitions de variables d'environnement et des créations d'alias.

Exemple de fichier `~/.profile` :

```
export PATH=$PATH:.:$HOME/bin:/usr/X11R6/bin;
EDITOR=vim;     export EDITOR
PAGER=less;     export PAGER
alias ls='gnuls --color=auto'
alias ll='ls -l'
alias rm='rm -i'
alias servux='ssh -2 -C -l michelon servux'
export CVS_RSH=ssh
export FTP_PROXY=cache:3128
```

