

Pratique du C interpréteur de commande

Licence Informatique — Université Lille 1
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Semestre 5 — 2010-2011

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

Représentation de l'utilisateur par le système

Tout utilisateur — considéré comme une entité connue par le système d'exploitation — est caractérisé par

- ▶ son *login* i.e. le nom d'utilisateur ;
- ▶ son mot de passe ;
- ▶ un unique numéro d'identification (uid) ;
- ▶ un (ou des) numéro(s) de groupe(s) d'utilisateurs (guid) auquel il appartient ;
- ▶ un répertoire i.e. son espace disque (\$HOME) ;
- ▶ le nom d'un programme d'interface entre l'utilisateur et le système.

Par exemple pour le superutilisateur, on trouve les informations suivantes dans le fichier `/etc/passwd` :

```
root:x:0:0:root:/root:/bin/bash
```

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

Notion de processus

Un processus est l'abstraction d'un programme exécuté par la machine.

programme sur le disque

Magic Number
entête
Code
données initialisées
table des symboles

Séparé en bloc

processus en mémoire

Pile d'exécution
↓
↑
tas (malloc)
données non initialisées
données initialisées
code

Séparé en page

Comme exemple de *magic number*, signalons qu'un fichier commençant par `#!` est sensé être un script pour un interpréteur (ces 2 caractères sont suivis par le chemin d'accès à l'interpréteur).

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

Détails pratiques

Équipe pédagogique :

Francesco	De Comité	G 1
Alexandre	Sedoglavic	G 2
Mikaël	Salson	G 3

Toutes les informations (emploi du temps, semainier, documents, etc.) sont disponibles à l'url :

<http://www.fil.univ-lille1.fr/portail>

Licence → S5 info → PDC.

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdf

Fichier et informations utilisateur relatives

Un fichier est l'abstraction d'un flux linéaire d'octets.

Aucune information sur l'organisation de l'espace du support à ce niveau d'abstraction. Pour manipuler les fichiers, il faut pouvoir les identifier par leurs caractéristiques :

- ▶ nom, type, taille du fichier ;
- ▶ propriétaire du fichier, son groupe ;
- ▶ date de création, date de dernière modification ;
- ▶ protection : qui a droit de le lire et de le manipuler.

Au fichier `foo.bar` sont associées les informations

```
-rw-r--r-- 1 sedoglav calforme 0 Aug 19 05:09 foo.bar
```

Ces informations correspondent dans l'ordre aux droits, nombre de liens, au propriétaire, à son groupe, à la taille, à la date de création et au nom du fichier.

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdf

Très schématiquement (et artificiellement), on peut répartir les *processus* en deux grandes catégories :

- ▶ les processus **d'applications** qui accomplissent des tâches souhaitées par l'utilisateur (calculs scientifique, base de données, bureautique, etc.) ;
- ▶ les processus **systèmes** qui permettent l'exploitation des ressources de l'ordinateur (processeurs, mémoire, terminaux, clavier, disques, coexistence/communication de plusieurs applications, etc).

Utilisateur 1	...	Utilisateur <i>n</i>	
Calculs	Base de données	Bureautique	} Applications
Compilateur	interpréteur	Système d'exploitation	
Langage machine	Dispositif physique		} Matériel

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdf

Le système d'exploitation permet aux applications :

- ▶ d'utiliser les ressources matérielles de la machine ;
 - ▶ d'ordonner leurs exécutions les unes par rapport aux autres (éviter l'occupation du processeur par une application endormie, définir des priorités),
 - ▶ de gérer des droits (exécution, lecture) à des fins de sécurisation ;
- ▶ de communiquer :
 - ▶ par l'intermédiaire de la mémoire vive,
 - ▶ par l'intermédiaire de la mémoire persistente (disque),
 - ▶ par des structures ad hoc (files de messages, sémaphore pour la synchronisation, etc).

Dans cette optique toute tâche complexe impliquant plusieurs applications doit être codée et prévue en bas niveau (langage C par exemple) en utilisant la connaissance du système.

Comment sans cela permettre à l'utilisateur d'utiliser les applications mises à sa disposition en les "combinant" au grès de sa fantaisie et de ses besoins ?

Un *shell* est un processus qui sert d'interface avec le système. Il

- ▶ ne fait pas partie du système d'exploitation (c'est un processus comme les autres qui l'utilise) ;
- ▶ est une interface interactive entre l'utilisateur, les applications disponibles et l'OS. Il permet d'exécuter et de combiner des filtres ;
- ▶ En mode *batch*, il offre un langage de programmation : les instructions sont définies dans un *script* que le shell interprète (pas de compilation).

Les suites d'instructions ne sont pas compilées et sont donc portables sur tout UNIX. Il existe plusieurs interpréteurs de commandes :

- ▶ dérivés du Bourne shell (sh, AT&T, 1977) comme ksh (korn shell), bash (Bourne again shell), zsh (zero shell), etc. ;
- ▶ dérivés du C shell (csh, BSD, 1979) comme tcsh (Tenex C shell), etc.

Commande externe

Un shell permet d'exécuter une commande externe :

```
% <commande externe> [option(s)] [argument(s)]
```

i.e. un programme exécutable ; le shell se *clone* puis se *mute* en un processus associé à l'exécutable.

Par exemple, l'exécutable de la commande `ls` se trouve dans le répertoire `/bin` ; il affiche les informations relatives à un fichier :

```
% /bin/ls -l /usr/bin/man  
-rwxr-xr-x 1 root root 46308 Apr 8 2005 /usr/bin/man
```

- ▶ `/usr/bin/man` est un argument indiquant que l'on désire un affichage concernant ce fichier ;
- ▶ `-l` est une option indiquant que l'on désire un affichage de toutes les informations.

La façon la plus simple (et primitive) d'envisager la communication entre applications est de considérer ces dernières comme des *filtres*. Comme tout processus, un filtre possède (au minimum) 3 fichiers d'entrée-sortie :

- ▶ 0 `stdin` est l'entrée standard (par défaut, le clavier) ;
- ▶ 1 `stdout` est la sortie standard (par défaut, l'écran) ;
- ▶ 2 `stderr` est la sortie des erreurs (par défaut, l'écran).

De plus, chaque processus retourne à son père (son processus créateur) un octet qui est un *code de retour*.

Un filtre est une fonction invoquée par un identifiant (`ls`), des options (`-al`), des arguments (`/bin`) qui prend en paramètre une suite d'octets depuis l'entrée standard, retourne des octets dans ses sorties et produit un code de retour.

Une fonction a une *effet latéral* si elle modifie un état autre que ses valeurs de retour. Pour être utile, les filtres ont des effets latéraux divers (affichage, saisie, connexion à un serveur, création/modification/destruction de fichiers, etc).

Interface interactive du shell

Le shell permet notamment de manipuler les abstractions courantes (fichiers, processus, etc). Pour ce faire, il propose une *invite de commande* que nous désignerons par `%`.

1. Cette invite est associée à un éditeur en ligne et à l'ensemble des possibilités classiques (déplacement, copié collé, etc).
2. Un backslash (code ASCII 92) suivi d'un retour chariot permet d'éditer une commande sur plusieurs lignes.
3. Un caractère dièse (code ASCII 35) débute un commentaire.

```
% # ceci est un commentaire  
% \  
> \  
> # encore un commentaire avec \ au milieu  
%
```

Les commandes shell sont de 2 types : interne et externe.

Quelques exemples de commandes externes

L'outil fondamental est le manuel d'utilisation `man` et la première chose à faire est de lire l'aide sur le manuel en utilisant la commande `% man man` dans votre interpréteur de commandes favori.

- ▶ `% man -a mount` affiche l'ensemble des pages d'aide contenant le mot `mount`. Entre autre :

```
mount          (2)  - mount and unmount filesystems  
mount          (8)  - mount a file system
```
- ▶ `% man -S8 mount` affiche l'aide sur `mount` issue de la section 8 du manuel.

On peut aussi utiliser l'utilitaire `info` mais, bien que plus évolué (liens hypertext), il n'est pas forcément complet.

V45 (10-09-2010)

V45 (10-09-2010)

V45 (10-09-2010)

Ceci fait les exécutable disponibles n'auront plus de secrets pour vous :

chmod	changer les droits d'un fichier
cp	copie de fichier
find	rechercher un fichier
grep	afficher les lignes des fichiers contenant une chaîne de caractères
kill	envoyer un signal à un processus
less	afficher le contenu d'un fichier
ls	affichage des informations relatives au contenu d'un répertoire
mkdir	créer un répertoire
mv	déplacement de fichier
passwd	créer ou changer de mot de passe
ps	afficher la liste des processus
rm	détruire un fichier

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

Pour détruire un processus dont le shell attend la terminaison, on utilise le raccourci clavier CTRL-C.
Pour interrompre sans détruire un processus, on utilise le raccourci clavier CTRL-Z ; pour le relancer :

- ▶ en tâche de fond, on utilise la commande interne bg ;
- ▶ en avant plan, on utilise la commande interne fg.

La commande externe ps retourne dans STDOUT les informations associées aux processus.

```
% ps -l
F S UID      PID PPID C PRI NI ADDR SZ WCHAN TTY          TIME CMD
0 S 613 2434 2426 0 75 0 -   954 rt_sig pts/1    00:00:00 bash
```

La commande externe kill -<Signal> <PID> envoie un signal au processus d'identificateur PID. Les principaux signaux sont :

Signal	Signification
15	terminaison de processus
9	destruction inconditionnelle de processus (CTRL-C)
19	suspension de processus (CTRL-Z)
18	reprise d'exécution d'un processus suspendu

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

Les filtres sont associées à des flux d'octets depuis le fichier standard STDIN vers les fichiers standards STDOUT et STDERR. Ces flux peuvent être redirigés par les opérateurs :

$n > \text{foo}$: fichier standard de descripteur $n (= 1, 2)$ dans le fichier foo (création ou écrasement) ;
 $n >> \text{foo}$: fichier standard de descripteur $n (= 1, 2)$ dans le fichier foo (création ou ajout) ;
 $n < \text{foo}$: le fichier foo est envoyé dans le fichier de descripteur $n (= 0, 1, 2)$;
 $n << \text{EOF}$ (texte EOF) : insertion de texte dans le fichier de descripteur $n (= 0, 1, 2)$;
| : tube de communication entre 2 filtres ;
 $n > \&m$: réoriente le flux de sortie du fichier de descripteur n dans celui de descripteur m ;
 $n < \&m$: réoriente le flux d'entrée du fichier de descripteur n dans celui de descripteur m .

Si l'entier n est omis, la redirection concerne STDOUT pour les sorties et STDIN pour les entrées.

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

www.fil.univ-lille1.fr/~sedoglav/C/Cours01.pdfV45 (10-09-2010)

Interprétation séquentielle vs asynchrone

- ▶ $\text{subexpr1} < \text{optionnel} > ; \text{subexpr2} ; \dots < \text{optionnel} >$
 - ▶ l'opérateur ; permet de séparer l'exécution de commandes %cd / ; ls ;. Par défaut, les shells attendent la fin de l'exécution d'une commande avant de permettre la saisie et l'exécution d'une autre ;
 - ▶ le code de retour de l'expression est celui de la dernière sous-expression dans la liste.
- ▶ $\text{subexpr1} \& < \text{optionnel} > \text{subexpr2} \& \dots < \text{optionnel} >$
 - ▶ les shells permettent aussi de lancer une application en tâche de fond (dans un shell-fils) et ainsi l'exécution d'une autre (même si la première n'est pas terminée, dans le shell d'origine). Pour ce faire, on termine l'expression par & ;
 - ▶ si STDIN n'est pas précisé et que subexpr1 n'est pas interactive, l'entrée standard est /dev/null ;
 - ▶ le code de retour d'une expression asynchrone est 0 dans tous les cas.

Une expression entre parenthèses est interprétée par un shell-fils du shell courant et pas par ce dernier :

```
%( exit ) # est bien différent de
% exit
```

On dispose de 2 opérateurs conditionnels :

$\text{subexpr1} \&\& \text{subexpr2}$: subexpr2 est exécuter si, et seulement si, subexpr1 retourne 0 ;

$\text{subexpr1} || \text{subexpr2}$: subexpr2 est exécuter si, et seulement si, subexpr1 retourne un code non nul.

Ces deux règles sont appliquées par le shell lorsqu'une suite de commandes contient plusieurs opérateurs && et ||. Ces deux opérateurs ont la même priorité et leurs évaluations s'effectue de gauche à droite.

Le code de retour des expressions ainsi construites est le code de retour de la dernière sous-expression exécutée.

Quelques illustrations des redirections

```
% ls /bin 1> /tmp/foo ; grep sh 0< /tmp/foo # correct
% ls /ntn /bin 1>/dev/null 2> /tmp/err # correct
% grep sh 0< ls # incorrect car il n'y pas pas de fichier ls
% ls 1> grep sh # incorrect car cr'ee le fichier grep
% (ls /ntn /bin 2>&1) 1>/tmp/foo# manipule 2 filres, ls et sh
```

Un exemple d'insertion de texte où le filtre grep prend son entrée depuis le clavier jusqu'à la saisie de pourfinir :

```
% grep tata << pourfinir
? abcd
? abcdtata
? pourfinir
abcdtata
```

Les commandes suivantes sont équivalentes :

```
% ls /bin>/tmp/foo;grep sh</tmp/foo>&result;
% ls /bin | grep sh >& result # >& redirige stdout et stderr
```

Les pipelines (tubes de communication)

Un tube est une suite d'une ou plusieurs expressions séparée par l'opérateur `|` :

- ▶ `<optionnel> !< \optionnel> subexpr1 <optionnel> | subexpr2 ... < \optionnel>`
La sortie standard de tous — sauf le dernier — les filtres associés aux sous-expressions est redirigée vers l'entrée standard du suivant ;
- ▶ l'opérateur `|` est prioritaire sur les autres redirections ;
- ▶ si le pipeline n'est pas lancé en tâche de fond, le shell attend la fin de la dernière commande du pipe avant de rendre l'invite de commande ;
- ▶ le code de retour de l'expression et celui de la dernière commande du pipe.

Dans ce cas, l'opérateur `!` est une négation du code de retour i.e. `!0 = 1` et si $n \neq 0$ alors $!n = 0$.

Le `!` est aussi utilisé par la commande interne `history`.

La commande interne `alias` établit une correspondance entre 2 chaînes de caractères. Par exemple,

```
%alias ll='echo "Affichage long";ls -l'
```

L'interpréteur substituera le membre de gauche (`ll`) par le membre de droite (`ls -l`) lorsqu'il apparaît comme premier mot d'une commande.

```
% cd /bin/ ; ll ls
Affichage long
-rwxr-xr-x  1 root  root  77964 Feb 13  2003 /bin/ls
```

De plus, il maintient une liste des alias qui peuvent être supprimés par la commande interne `unalias`.

Les alias sont généralement définis dans le fichier de configuration (`.bashrc` ou `.cshrc` suivant le shell utilisé) qui est exécuté par l'interpréteur à son démarrage.

Paramètres du shell

- ▶ un *paramètre* du shell peut être identifié par un nombre, un caractère spécial (cf. section suivantes) ou un nom (une chaîne de caractères alphanumérique qui n'est ni un nombre ni un caractère spécial) ;
- ▶ une *variable* du shell est un paramètre identifié par un nom ;
- ▶ un *paramètre de position* est un paramètre qui n'est ni spécial ni une variable.

Un paramètre est affecté s'il possède une *valeur* (null est une valeur).

Une variable ne peut être désaffectée que par la commande interne `unset`.

Commande interne

Une commande interne est un filtre implanté dans le shell et ne correspond (en théorie) à aucun fichier exécutable.

L'objectif étant :

- ▶ d'augmenter les performances de filtres très fréquemment utilisé ;
- ▶ de permettre des fonctionnalités difficiles à mettre en œuvre avec un code indépendant du shell.

Dans le GNU-bash-3.0, les commandes `false`, `true`, `kill`, `pwd` et `newgrp` sont externes bien que la norme les considère comme internes.

La commande interne `type` retourne dans `STDOUT` des informations sur les commandes (sont elles internes, externes, etc).

La commande externe `which foo` retourne dans `STDOUT` le chemin d'accès à la commande externe `foo` si elle le trouve.

Commandes internes spéciales

Les commandes internes suivantes

`break`, `colon`, `continue`, `dot`, `eval`, `exec`, `exit`, `export`, `readonly`, `return`, `set`, `shift`, `times`, `trap`, `unset`

sont qualifiées de spéciales car :

- ▶ une erreur de syntaxe dans leurs usages peut causer la destruction du shell ;
- ▶ l'affectation des variables (voir plus loin) au cours de l'exécution de ces commandes reste valide après leurs terminaisons.

Ce n'est pas le cas des autres commandes (internes ou externes).

Les autres commandes internes sont :

`alias`, `bg`, `cd`, `command`, `false`, `fc`, `fg`, `getopts`, `jobs`, `kill`, `newgrp`, `pwd`, `read`, `true`, `umask`, `unalias`, `wait`

Paramètres spéciaux et de position

- ▶ `0` : le nom de la commande en cours ;
- ▶ `#` : son nombre de paramètres de position ;
- ▶ `*`, `@` : tous ses paramètres de position ;
- ▶ `1` à `9` : ses 9 premiers paramètres de position ;
- ▶ `x` : le paramètre de position $x(> 9)$;
- ▶ `$` : le pid de la commande courante ;
- ▶ `_` : le dernier paramètre manipulé (non normalisé) ;
- ▶ `-` : les drapeaux (options) de la commande courante ;
- ▶ `?` : toutes les commandes ont un code de retour — codé sur un octet — (*exit-status*) i.e. une valeur entière qui fournit une information sur le déroulement de la dernière commande exécutée.
 - ▶ déroulement normal $\Rightarrow ? = 0$,
 - ▶ déroulement anormal $\Rightarrow ? \neq 0$;

Nous verrons en C comment renvoyer le code de retour ;

- ▶ `$!` : le pid du dernier processus lancé en arrière fond.

Pratique du C interpréteur de commande

Quelques abstractions
Utilisateur
Fichier
Processus

Shell
Commande externe
Contrôle des processus

Opérateurs
Les opérateurs de redirection

Commandes internes et variables
Exemples de commande interne
Paramètres du shell
Variables
Typage, évaluation

Compléments
Expressions régulières
Raccourci clavier

V45 (10-09-2010)

Les paramètres spéciaux peuvent s'utiliser tels quels dans un shell :

```
% false ; echo $0 $$ ; ps | grep bash
1 bash 2977
2977 pts/3    00:00:00 bash
```

En mode interactif, on les affecte avec la commande interne set :

```
% echo $# # nous verrons plus tard le sens du $
0
% set foo bar ; echo $# $1 $2
2 foo bar
```

La commande interne shift permet le décalage des paramètres numérotés (1 est perdu et # est mis à jour).

```
% shift ; echo $# $@
1 bar
```

Pratique du C interpréteur de commande

Quelques abstractions
Utilisateur
Fichier
Processus

Shell
Commande externe
Contrôle des processus

Opérateurs
Les opérateurs de redirection

Commandes internes et variables
Exemples de commande interne
Paramètres du shell
Variables
Typage, évaluation

Compléments
Expressions régulières
Raccourci clavier

V45 (10-09-2010)

Les variables

Définition et affectation : une variable est définie dès qu'elle est affectée. En sh, % F00="Bonjour le monde". En csh,

```
% set F00="Bonjour le monde"
```

La commande interne echo permet d'afficher l'argument qui lui est fourni :

```
% echo F00
F00
```

Pour évaluer une variable, il faut préfixer son nom par \$.

```
% echo $F00
Bonjour le monde
```

En sh, la commande interne export étend la portée d'une variable : par défaut, cette dernière n'est connue que par le processus courant ; après coup, cette variable est connue par tous les processus fils de ce dernier. En csh, on utilise :

```
% setenv F00 "Bonjour le monde"
```

Pratique du C interpréteur de commande

Quelques abstractions
Utilisateur
Fichier
Processus

Shell
Commande externe
Contrôle des processus

Opérateurs
Les opérateurs de redirection

Commandes internes et variables
Exemples de commande interne
Paramètres du shell
Variables
Typage, évaluation

Compléments
Expressions régulières
Raccourci clavier

V45 (10-09-2010)

Dans un shell, **tout n'est que chaîne de caractères**.

Chaque commande est une chaîne que le shell évalue. On peut influencer sur cette évaluation grâce aux délimiteurs suivants :

- ▶ les quotes ' ' bloquent l'évaluation ;
- ▶ les guillemets " " forment une chaîne après évaluation des composantes ;
- ▶ les backquotes ` ` forment une chaîne évaluée comme une commande.

```
% echo '$F00'
$F00
% echo "echo '$F00'"
echo 'Bonjour le monde'
% set BAR="n\importe quoi" ; echo $BAR
n\importe quoi
% set BAR='n\importe quoi'
n\importe: Command not found.
```

Pratique du C interpréteur de commande

Quelques abstractions
Utilisateur
Fichier
Processus

Shell
Commande externe
Contrôle des processus

Opérateurs
Les opérateurs de redirection

Commandes internes et variables
Exemples de commande interne
Paramètres du shell
Variables
Typage, évaluation

Compléments
Expressions régulières
Raccourci clavier

V45 (10-09-2010)

Pratique du C interpréteur de commande

Quelques abstractions
Utilisateur
Fichier
Processus

Shell
Commande externe
Contrôle des processus

Opérateurs
Les opérateurs de redirection

Commandes internes et variables
Exemples de commande interne
Paramètres du shell
Variables
Typage, évaluation

Compléments
Expressions régulières
Raccourci clavier

V45 (10-09-2010)

Pratique du C interpréteur de commande

Quelques abstractions
Utilisateur
Fichier
Processus

Shell
Commande externe
Contrôle des processus

Opérateurs
Les opérateurs de redirection

Commandes internes et variables
Exemples de commande interne
Paramètres du shell
Variables
Typage, évaluation

Compléments
Expressions régulières
Raccourci clavier

V45 (10-09-2010)

Le shell dispose de variables que la commande interne set permet d'afficher :

```
% set
USER=sedoglav
LOGNAME=sedoglav
HOME=/home/enseign/sedoglav
PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin
MAIL=/var/mail/sedoglav
SHELL=/bin/csh
HOSTTYPE=i586-linux
PWD=/home/enseign/sedoglav
GROUP=enseign
LANG=fr_FR
SYSFONT=lat0-16
TMP=/home/enseign/sedoglav/tmp
HOSTNAME=lx2
```

La commande set permet aussi de manipuler les options du shell. Par exemple, %set -o vi permet de passer en mode d'édition vi.

Quelques variables d'environnement

Les variables définies dans les fichiers /etc/profile et ~/.profile sont créées lors de l'ouverture d'une session.

PATH	les répertoires dans lesquels sont cherchés les exécutables des commandes externes
HOME	votre répertoire de travail
TERM	le type de terminal
PWD	le répertoire courant
DISPLAY	cette variable est utilisée par l'interface graphique pour savoir où se fait l'affichage
PS1	l'invite de commande

Ces variables d'environnement peuvent être utilisées depuis un programme C (fonction getenv) lancé depuis le shell.

Manipulation d'entiers

Pour utiliser l'arithmétique de base, il faut évaluer des chaînes de caractères codant des expressions arithmétiques grâce à la commande externe expr :

```
% set i=12;set i='expr $i + 1'
% echo $i $?
13 0
% expr 2 \* 2
4
```

Le code de retour de la commande expr est :

- 0 si le résultat est différent de 0 ;
- 1 si le résultat est égal à 0 ;
- 2 si un argument est non numérique.

Les emphexpressions régulières décrivent des propriétés de construction de chaînes de caractères. Pour ce faire, on utilise en shell les *métacaractères* :

- ▶ le point d'interrogation `?` correspond à n'importe quel caractère (sauf EOL). L'expression régulière `b?1` représente les chaînes *bal* et *bol* et toutes les autres combinaisons comme *bwl* ;
- ▶ la paire de crochet `[]` permet de spécifier plus restrictivement un ensemble de caractères. L'expression régulière `dupon[dt]` ne représente que les chaînes *dupond* et *dupont*. L'expression `dupon[d-t]` représente les chaînes commençant par *dupon* et se terminant par une lettre comprise entre *d* et *t*. L'expression `dupon[^dt]` représente les chaînes commençant par *dupon* et ne se terminant ni par *d* ni par *t* ;
- ▶ l'étoile `*` désigne 0, 1 ou plusieurs caractères quelconques. Ainsi, `*` représente toutes les chaînes.

Le préfixe `\` (antislash) transforme un métacaractère en caractère.

Raccourci clavier et quelques caractères spéciaux utiles

La liste des raccourcis clavier est affichable par des commandes internes :

- ▶ `bind -p` pour bash ;
- ▶ `bindkey` pour csh.

Retenons pour mémoire :

CTRL-d caractère fin de fichier
CTRL-\ stop la commande en cours

Pour approfondir l'usage d'un interpréteur de commande, la prochaine étape consiste à étudier la syntaxe et la grammaire induite par les opérateurs et les commandes, l'évaluation associée, les expressions (simples et composées), les instructions de contrôle, les fonctions et le passage de paramètres, etc.