Cours Shell Unix
Commandes
&
Programmation

Révision corrigée du 31/01/2003

Sébastien ROHAUT

2002-2003

Table des matières

1 Présentation

1.1 Définition

Un **système d'exploitation** est un « programme » ou ensemble de programmes assurant la gestion de l'ordinateur et de ses périphériques.

Programme ou ensemble de programmes et d'API servant d'interface entre le matériel (hardware) et les applications (software).

Unix est un système d'exploitation multi-tâches et multi-utilisateurs. Il est disponible du simple micro (PC, Mac, Atari, Amiga) jusqu'au gros système (IBM Z séries) et même dans des PDA.

- **Portable** : Écrit majoritairement en C, seules quelques parties sont en assembleur.
- **Multi-tâches** : Le système peux exécuter plusieurs tâches en même-temps, de manière **préemptive**, sur un ou plusieurs processeurs.
- **Multi-utilisateurs** : Plusieurs utilisateurs peuvent se connecter et travailler en même temps sur une machine, soit directement sur celle-ci (Linux, BSD, Sco) soit depuis un terminal distant.
- **Stable** : protection mémoire, les plantages du système par lui-même sont très rares.
- **Deux standards principaux** : System V et BSD, qui tout en restant compatibles diffèrent au niveau de certains appels systèmes, de la hiérarchie du système de fichier, de la séquence de démarrage...

Les composants de base d'un Unix sont le **noyau** (**kernel**) et les **outils** (**shell** et commandes).

Le système d'exploitation a pour principales tâches les points suivants :

1. Gestion de la mémoire
2. Accès aux périphériques
3. Accès disque / Système de fichiers
4. Gestion des programmes (processus)
5. Sécurité / Accès aux données
6. Collecte d'informations système : Statistiques

1.2 Historique

1.2.1 Les origines

- **1969** : Bell Laboratories, centre de recherches commun à AT&T et Western Electrics, Ken Thompson travaille sur MULTICS (Multiplexed Information and Computing Service). Bell se retire du projet, Multics est abandonné. Ken Thompson décide de développer son propre OS, en s'éloignant volontairement de tout existant et écrit UNICS (Unified Information and Computing System) sur DEC PDP-7. Équipe : Dennis Ritchie, Rudd Canaday, puis Brian Kernighan.
- **1970** : Premier portage sur DEC PDP-11/20, avec le premier compilateur C, conçu spécialement pour rentre cet OS portable.
- **1971** : Version 1 d'Unix sur PDP/11-20 avec un système de fichiers, fork(), roff, ed, suite à la demande de AT&T qui avait besoin d'un système de traitement de textes pour l'aide à l'écriture de ses brevets.
- **1973** : La V2 intègre les tubes (pipes)

- **1974** : AT&T ne voyant pas d'avenir commercial à Unix, décide de distribuer le code source aux universités selon quatre critères de licence. Unix gagne donc la faveur des universitaires. Entre 1974 et 1977 les versions de la V3 à la V6 voient le jour.
- **1978** : La V7 est annoncée, développée afin de pouvoir être portée sur d'autres architectures matérielles. AT&T rattrape le coup et décide de distribuer les sources sous licence. Diffusion d'un manuel système et de développement par Brian Kernighan et Rob Pike. Apparition du Bourne Shell. Taille du noyau : 40Ko ! La V7 est la base commune à tous les Unix.
- **1979** : Le coût des licences Unix incite l'université de Californie à Berkeley à continuer ses travaux sur les sources diffusées avant la licence, et créée sa propre variante : BSD Unix. Le DARPA décide d'utiliser Unix pour ses développements, notamment BSD Unix.
- **1983** : AT&T met en vente la version commerciale de Unix SYSTEM V.
- **1986** : Première ébauche des normes POSIX sur la standardisation des appels systèmes et des fonctions.
- **1987** : Création de X-Window, interface C/S graphique développée au sein du MIT. System V v3, premiers Unix propriétaires de HP et IBM suite à la modification de la licence de SYSTEM V. BSD 4.3, Unification de BSD et SYSTEM V (Sun et AT&T), d'où abandon des particularités de chaque système.
- **1988** : Troisième version de X/Open Portability Guide, servant de référence pour tous les développements d'Unix ultérieurs (commandes, appels système, langages, requêtes, graphique, internationalisation, réseau).
- **1990** : System V v4 de AT&T, nouveaux standards d'unification avec Sun. Les autres constructeurs se sentent menacés et fondent OSF (Open Software Foundation).
- **1991** : OSF/1. Apparition des premiers clones Unix comme Linux et FreeBSD.
- **1992** : Sun sort Solaris (SunOS), dérivé de System V v4, avec la gestion des threads. AT&T créée USL (Unix Software Laboratories) et transfert toutes les licences à cettre société.
- **1993** : **Novell** rachète **USL**, puis transfert les droits de licences à **X/Open**.
- **Depuis 1993** : S'il existe un grand nombre d'Unix propriétaires, la plupart restent conformes aux normes et standards établis (X/Open, Posix). On distingue deux grandes branches SYSTEM V et BSD. Les deux sont compatibles. L'arrivée de Linux (dérivé de System V mais avec pas mal d'améliorations issues de BSD) a changé la donne.
- Les code source d'Unix appartient aujourd'hui à la société **Caldera** issue de Novell, mais les droits et la force de proposition sont transférés à l'**Open Group**.

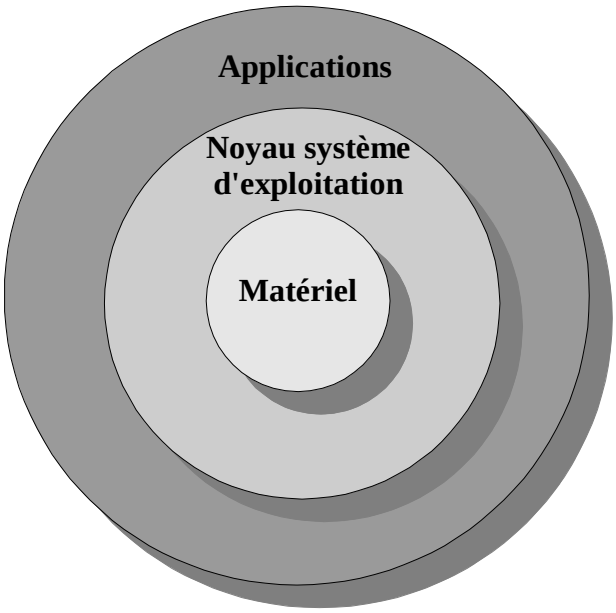
1.2.2 Unix sur Micro

Le premier Unix disponible sur PC a été porté par Microsoft en 1979, **Xenix**, disponible jusqu'en 1984. Il fonctionnait sur 8086 à l'aide de très lourdes modifications. L'essor a eu lieu avec l'apparition du 80286, premier processeur Intel à posséder un mode protégé et des instructions de commutation de tâches. A l'arrivée du 80386 en 1987, la société Santa Cruz Operations qui diffusait dèja Xenix modifia Xenix et l'appela UNIX System V/386.

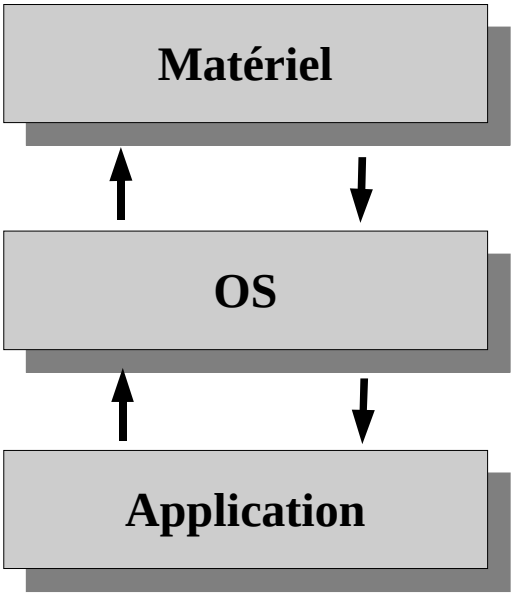
Depuis on trouve plusieurs portages Unix sur PC, dont les principaux sont SCO Openserver / Unixware, Sun Solaris, Linux, FreeBSD, OpenBSD, NetBSD.

Le PC n'est pas le seul micro supportant Unix : on trouve les Mac (anciens et nouveaux modèles), les Atari et Amiga, les machines à base de processeurs Alpha, ... Sur PC et dans les écoles, on utilise généralement Linux.

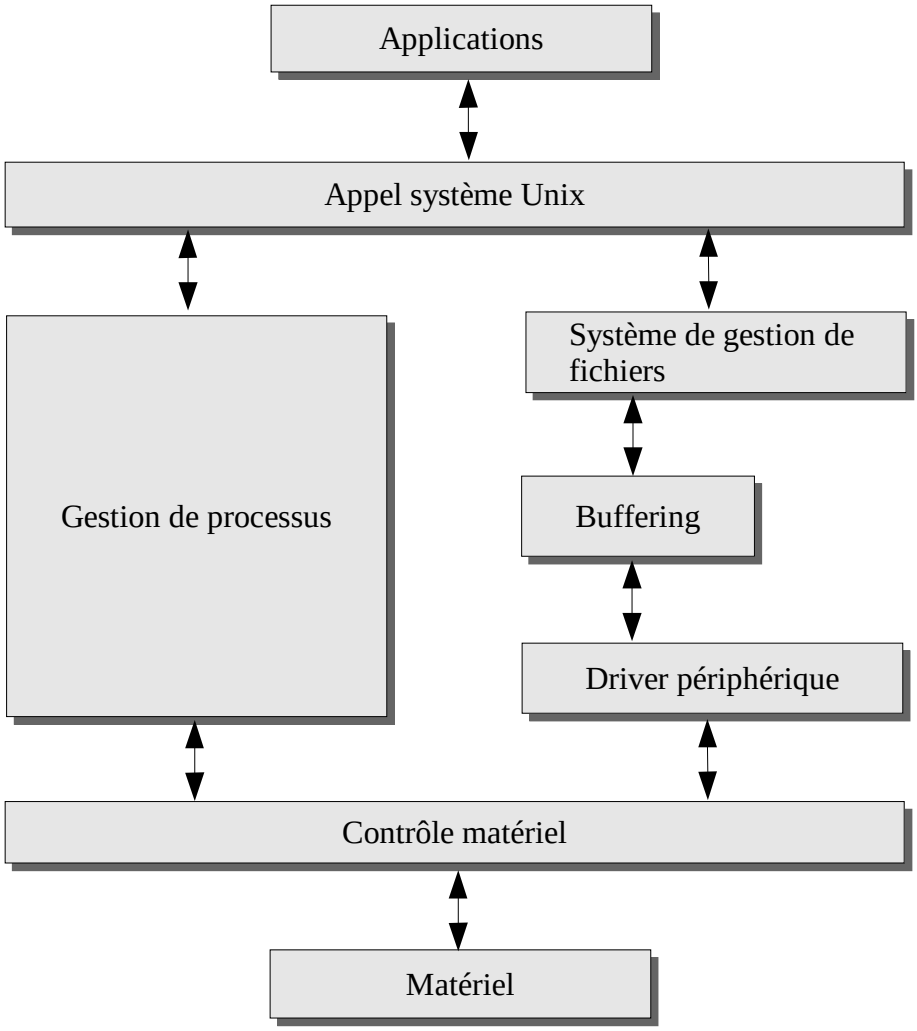
1.3 Architecture



Structure d'un SE UNIX



Rôle d'un OS



2 Connexion et déconnexion

2.1 Première Connexion

Pour pouvoir travailler sous Unix il faut ouvrir une session, à l'aide d'un nom d'utilisateur et d'un mot de passe. On distingue les administrateurs des utilisateurs normaux. L'administrateur est appelé **root** ou utilisateur privilégié et dispose de tous les pouvoirs sur la machine et le système Unix. L'utilisateur normal dispose de droits réduits et définis par l'administrateur. Pour se connecter :

```
Login : <tapez ici votre nom d'utilisateur>
Password : <tapez ici votre mot de passe>
```

Le mot de passe n'apparaît pas en clair et doit être tapé en aveugle. En cas d'erreur, un message indiquera :

```
Login incorrect
```

Suivant la version d'Unix et la configuration, plusieurs lignes de message peuvent apparaître, qu'il est possible d'ignorer. Puis le prompt du shell devrait apparaître, quelque chose du genre

```
user@machine$
```

ou

```
$
```

ou

```
%
```

Pour se familiariser avec la saisie de commandes, nous pouvons tester quelques programmes d'information :

- **date** : affiche la date et l'heure
- **who** : liste des utilisateurs connectés (who am i : qui suis-je)
- **cal** : affiche le calendrier (cal 12 1975)
- **man** : mode d'emploi (man cal)

Pour interrompre une commande on peut utiliser la combinaison **Ctrl+C**. Pour lier les commandes, on sépare les commandes par le caractère « ; »

```
who ; date
```

Pour se déconnecter, il suffit de taper

```
exit
```

On peut aussi utiliser la combinaison de touches **Ctrl+D**.

2.2 Changer son mot de passe

On utilise la commande **passwd** pour modifier son mot de passe.

```
$ passwd
Old password:
New password:
Reenter password:
```

Dans certains cas, il faudra utiliser la commande **yppasswd** (cas de NIS). Sur les Unix récents l'administrateur peut définir des règles de sécurité comme le nombre minimum de caractères, contrôler le mot de passe depuis un dictionnaire s'il est trop simple, lui donner une date de péremption, ...

2.3 Utilisateur, groupes

On distingue sous Unix les utilisateurs et les groupes, notion que nous verrons et détail lors de la gestion des droits. Un groupe définit un ensemble d'utilisateurs, un utilisateur fait obligatoirement partie d'au moins un groupe, ou de plusieurs. Le groupe par défaut d'un utilisateur est « **users** ».

2.4 Prendre la place d’un autre

Le système permet dans certains cas à un utilisateur connecté de changer de nom en cours de travail avec la commande su. Le mot de passe du nouvel utilisateur sera demandé.

```
su [-] utilisateur [-c commande]
```

Si – est précisé, l'environnement du nouvel utilisateur est chargé, et si -c est précisé les commandes qui suivent sont exécutées.

2.5 Obtenir son nom de connexion

La commande **logname** affiche le nom de login de l'utilisateur, en principe toujours le nom utilisé lors de la première connexion.

```
$ logname
oracle
```

3 Une première commande : « echo »

En principe cette commande n'est pas utile tout de suite, mais la première chose que l'on apprend généralement avec un shell ou un langage quelconque est d'afficher un message du genre « Hello, world ! ». la commande **echo** est une commande centrale du shell : elle transmet tous ses paramètres sur écran (ou canal de sortie standard).

echo texte

Le texte est quelconque mais peut aussi admettre quelques caractères de formatage.

Caractère	***Effet***
\n	Saut de ligne (newline)
\b	Retour arrière (backslash)
\t	Tabulation
\c	Pas de retour à la ligne (carriage)
\\	Affiche \
\$	Affiche $
\valeur	Affiche le caractère spécial de code octal valeur

```
$ echo "Bonjour\nComment ça va ?\c"
```

On peut aussi afficher des variables (Partie Programmation).

4 Le système de fichiers

4.1 Définition

Un système de fichiers / FileSystem / FS: comment dont sont gérés et organisés les fichiers par le système d'exploitation. Le FS d'Unix est hiérarchique.

4.2 Les divers types de fichiers

On distingue principalement trois types de fichiers :ordinaires, catalogue, spéciaux.

4.2.1 fichiers ordinaires (ordinary files)

Ce sont soit des fichiers contenant du texte, soit des exécutables (ou binaires), soit des fichiers de données. Par défaut, rien ne permet de différencier les uns des autres, sauf à utiliser quelques options de certaines commandes (ls -F par exemple) ou la commande **file**.

```
$ file nom_fic
nom fic : 32 Bits ELF Executable Binary (stripped)
```

4.2.2 catalogues (les répertoires ou directory)

Les répertoires permettent d'organiser le disque dur en créant une hiérarchie. Un répertoire peut contenir des fichiers normaux, des fichiers spéciaux et d'autres répertoires, de manière récursive.

4.2.3 fichiers spéciaux

Ce sont le bien souvent des fichiers servant d'interface pour les divers périphériques. Ils peuvent s'utiliser, suivant le cas, comme des fichiers normaux. Un accès en lecture ou écriture sur ces fichiers est directement dirigé vers le périphérique (en passant par le pilote Unix associé s'il existe

4.3 Nomenclature des fichiers

On ne peut pas donner n'importe quel nom à un fichier, il faut pour cela suivre quelques règles simples. Ces règles sont valables pour tous les types de fichiers.

Sur les anciens systèmes un nom de fichier ne peut pas dépasser 14 caractères. Sur les systèmes récents, on peut aller jusqu'à 255 caractères. Il est possible d'utiliser des extensions de fichiers mais cela ne modifie en rien le comportement du système (un exécutable n'a pas besoin d'une extension particulière).

Unix fait la distinction entre les minuscules et majuscules. Toto, TOTO, ToTo et toto sont des noms de fichiers différents.

La plupart des caractères (chiffres, lettres, majuscules, minuscules, certains signes, caractères accentués) sont acceptés, y compris l'espace (très déconseillé). Cependant quelques caractères sont à éviter :

- `& ; ( ) ~ <espace> \ | ` ? - (en début de nom)`
-

Quelques noms valides :

```
Fichier1
Paie.txt
123traitement.sh
Paie_juin_2002.xls
8
```

```
...
```

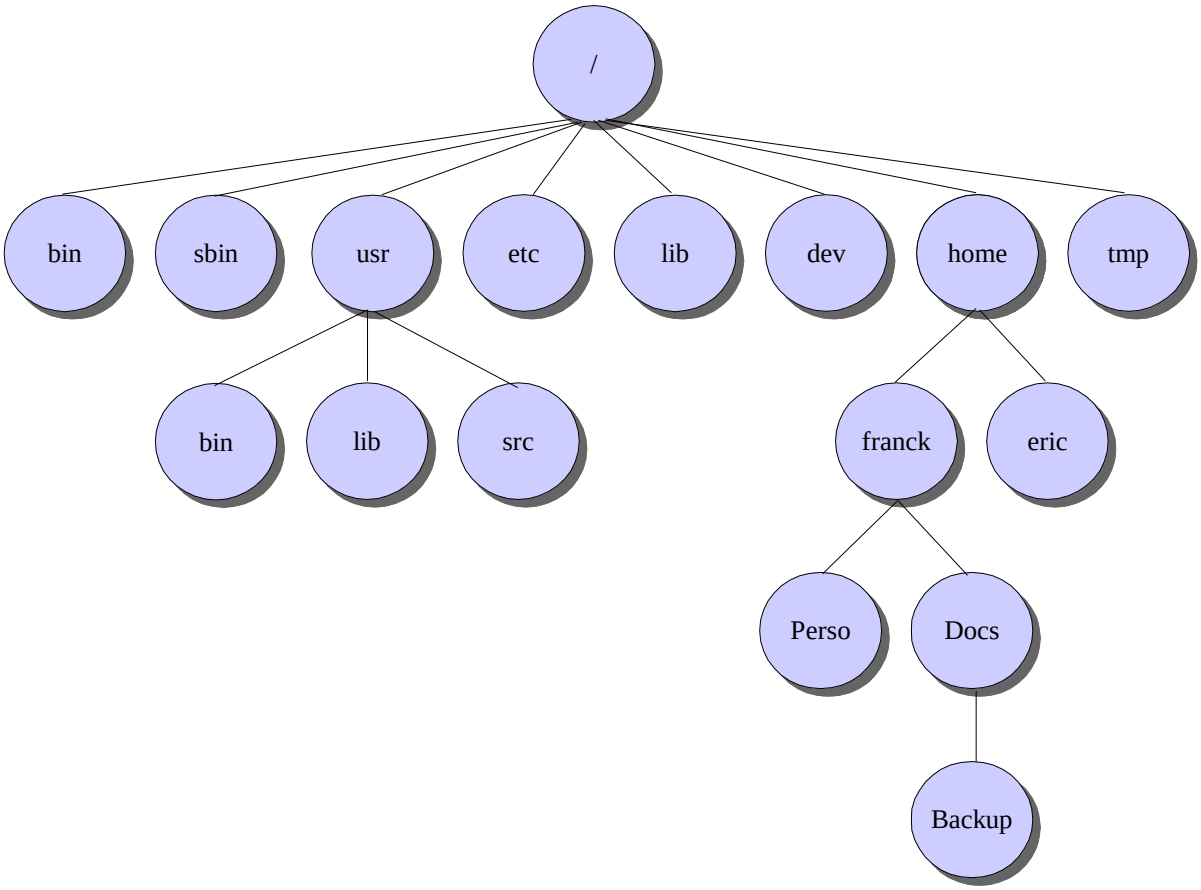
Quelques noms pouvant poser problème :

```
Fichier*
Paie(decembre)
Ben&Nuts
Paie juin 2002.xls
-f
...
```

4.4 Chemins

4.4.1 Structure et nom de chemin

Les chemins permettent de se déplacer dans le FileSystem. Un nom de fichier est ainsi généralement complété de son chemin d'accès. C'est ce qui fait que le fichier « toto » du répertoire « rep1 » est différent du fichier « toto » du répertoire « rep2 ». Le FS d'Unix étant hiérarchique, il décrit une arborescence.



Le schéma précédent représente une arborescence d'un FS Unix. Le « / » situé tout en haut s'appelle la **racine** ou **root directory** (à ne pas confondre avec le répertoire de l'utilisateur root).

Le **nom de chemin** ou **path name** d'un fichier est la concaténation, depuis la racine, de tous les répertoires qu'il est nécessaire de traverser pour y accéder, chacun étant séparé par le caractère « / ». C'est un c**hemin absolu**.

```
/home/toto/Docs/Backup/fic.bak
```

4.4.2 Chemin relatif

Un nom de chemin peut aussi être relatif à sa position courante dans le répertoire.

Le système (ou le shell) mémorise la position actuelle d'un utilisateur dans le système de fichier, le répertoire actif. On peut donc accéder à un autre répertoire de l'arborescence depuis l'emplacement actuel sans taper le chemin complet.

Pour se déplacer dans les répertoires, on utilise la commande **cd**. Le « **..** » permet d'accéder au répertoire de niveau supérieur. Le « **.** » définit le répertoire actif (répertoire courant). La commande **ls** permet de lister le contenu du répertoire. La commande **pwd** (print working directory) affiche le chemin complet du répertoire actif.

```
$ cd /
$ ls
bin
sbin
usr
etc
lib
dev
home
tmp
$ cd /usr/lib
$ pwd
/usr/lib
$ cd ../bin
$ pwd
/usr/bin
$ cd ../../etc
$ pwd
/etc
```

4.4.3 Répertoire personnel

Lors de la création d'un utilisateur, l'administrateur lui alloue un répertoire utilisateur. Après une connexion, l'utilisateur arrive directement dans ce répertoire, qui est son **répertoire personnel**. C'est dans ce répertoire que l'utilisateur pourra créer ses propres fichiers et répertoires. La commande **cd** sans argument permet de retourner directement dans son répertoire utilisateur.

```
Login : toto
Password :
$ pwd
/home/toto
```

4.4.4 ls et quelques commandes intéressantes

La commande **ls** permet de lister le contenu d'un répertoire (catalogue) en lignes ou colonnes. Elle supporte plusieurs options.

Option	***Signification***
-l	Sortie de chaque information des fichiers
-F	Ajoute un « / » au nom d'un répertoire, un « * » au nom d'un exécutable, un « | » pour un tube nommé et « @ » pour un lien symbolique.
-a	Affiche toutes les entrées, y compris « . », « .. » et les fichiers cachés (qui commencent par un .)

Option	*Signification*
-d	affiche le nom (et les attributs) des répertoires et pas leur contenu.
-i	Affiche les numéros d'inode.
-R	Mode récursif. Rentre dans les répertoires et affiche leur contenu.
-r	Inverse l'ordre du tri (à l'envers)
-t	Tri par date de modification
-c	Affiche la date de création (si -l) ou tri par date de création (si -t)
-C	Les noms sont affichés sur plusieurs colonnes
-u	Affiche la date d'accès (-l) ou tri par date d'accès (-t)
-1	Liste sur une seule colonne.

Sortie de la commande ls -l :

-rw-r--r--	1	oracle	dba	466	Feb 8 2001	input.log
1	2	3	4	5	6	7

1. Le premier caractère représente le type de fichier, les autres, par blocs de trois, les droits pour l'utilisateur, le groupe et tous (expliqué plus loin).
2. Compteur de liens (expliqué plus loin)
3. Propriétaire du fichier.
4. Groupe auquel appartient :le fichier
5. Taille du fichier en octets
6. Date de dernière modification (parfois avec l'heure)
7. Nom du fichier

Deux autres commandes utiles :

- **cat** : concaténation de fichiers, le résultat étant affiché par défaut sur la sortie standard (écran)
- **touch** : permet de créer un fichier s'il n'existe pas, et s'il existe de modifier sa date d'accès et sa date de modification, touch toto crée le fichier toto s'il n'existe pas

4.5 Gestion des fichiers et répertoires

4.5.1 Création de répertoires

La commande **mkdir** (make directory) permet de créer un ou plusieurs répertoires, ou une arborescence complète.

```
mkdir rep1 [rep2] ... [repn]
```

```
$ mkdir documents
$ mkdir documents/texte documents/calcul documents/images
```

La commande mkdir accepte un paramètre « -p » permettant de créer une arborescence. Dans

l'exemple précédent, si je veux créer documents/texte et que documents n'existe pas, alors :

```
$ mkdir -p documents/texte
```

va créer à la fois documents et texte. C'est valable pour tous les répertoires de niveau supérieur :

```
$ mkdir -p documents/texte/perso
```

va créer les répertoires documents, texte et perso s'ils n'existent pas. S'il existent ils ne sont pas modifiés.

4.5.2 Suppression de répertoires

La commande **rmdir** (remove directory) supprime un ou plusieurs répertoires. Elle ne supprime pas une arborescence. Si des fichiers sont encore présents dans le répertoire, la commande retourne une erreur. Le répertoire ne doit donc contenir ni fichiers ni répertoires.

```
rmdir rep1 [rep2] ... [repn]
```

```
$ cd documents
$ rmdir texte/perso
```

4.5.3 Copie de fichiers

La commande **cp** (copy) copie un ou plusieurs fichiers vers un autre fichier ou vers un répertoire.

```
cp fic1 fic2
cp fic1 [fic2 ... ficn] rep1
```

Dans le premier cas, fic1 est recopié en fic2. Si fic2 existe, il est écrasé sans avertissement (sauf droit particulier). Dans le second cas, fic1, fic2 et ainsi de suite sont recopiés dans le répertoire rep1. Les chemins peuvent êtres absolus ou relatifs. La commande peut prendre les options suivantes :

- **-i** : demande une confirmation pour chaque fichier avant d'écraser un fichier existant.
- **-p** : préservation des permissions, dates d'accès de modification
- **-r** : Récursif. Si la source est un répertoire copie de ce répertoire et de toute son arborescence. Les liens symboliques (voir plus loin) ne sont pas recopiés tels quels, mais seulement les fichiers pointés (avec le nom du lien cependant).
- **-R** : comme -r mais la recopie est identique à l'original (les liens symboliques sont copiés tels quels)

```
$ cd
$ pwd
/home/toto
$ touch cv.txt calc.xls image.jpg
$ ls
cv.txt calc.xls image.jpg documents
$ cp cv.txt documents/texte
$ cp calc.xls documents/calcul/calcul_paie.xls
$ cd documents
$ touch fichier
$ cp fichier ..
$ cp texte/cv.txt .
```

4.5.4 Déplacer et renommer un fichier

La commande **mv** (move) permet de déplacer et/ou de renommer un fichier. Elle a la même syntaxe

que la commande cp. On peut à la fois déplacer et changer de nom.

```
$ cd
$ mv cv.txt cv_toto.txt
$ mv image.jpg documents/images/photo_toto_cv.jpg
```

4.5.5 Supprimer un fichier ou une arborescence

La commande **rm** (remove) supprime un ou plusieurs fichiers, et éventuellement une arborescence complète, suivant les options. La suppression est définitive (à moins d'avoir un utilitaire système propre au filesystem).

```
rm [Options] fic1 [fic2...]
```

Options :

- **-i** : la commande demandera une confirmation pour chacun des fichiers à supprimer. Suivant la version d'Unix, le message change et la réponse aussi : y, Y, O, o, N, n, parfois toutes.
- **-r** : le paramètre suivant attendu est un répertoire. Dans ce cas, la suppression est récursive : tous les niveaux inférieurs sont supprimés, les répertoires comme les fichiers.
- **-f** : force la suppression. Si vous n'êtes pas le propriétaire du fichier à supprimer, rm demande une confirmation, mais pas avec l'option -f. Aucun message n'apparaîtra si la suppression n'a pu avoir lieu.

```
$ cd
$ rm -rf documents
```

4.5.6 Les liens : plusieurs noms pour un fichier

Un lien permet de donner plusieurs noms à un même fichier, ou de faire pointer un fichier sur un autre. Plutôt que de faire plusieurs copies d'un même fichier pour plusieurs utilisateurs, on peut par exemple permettre à ceux-ci d'accéder à une copie unique, mais depuis des endroits et des noms différents. On utilise la commande **ln**.

ln [options] fic1 fic2

```
ln [options] fic1 rep1
ln [options] rep1 fic2
```

Il existe deux types de liens : les liens en dur « **hard links** » et les liens symboliques « **symbolic links** ».

4.5.6.1 Hard link

Un **hard link** permet d'ajouter une référence sur un inode.

Sous Unix chaque fichier est en fait référencé au sein de deux tables : une table d'**inode** (information node, noeud d'information, une par filesystem) qui contient outre un numéro de fichier, des informations comme des droits, le type et des pointeurs sur données, et une table catalogue (une par répertoire) qui est une table de correspondance entre les noms de fichiers et les numéros d'inodes. Le hard link rajoute donc une association dans cette seconde table entre un nom et un inode. Les droits du fichier ne sont pas modifiés.

Un hard link ne permet pas d'affecter plusieurs nom à un même répertoire, et ne permet pas d'effectuer des liens depuis ou vers un autre filesystem. De plus, faites attention au compteur de lien fourni par la commande ls -l : un 1 indique que ce fichier ne possède pas d'autres liens, autrement dit c'est le dernier. Si vous le supprimez, il est définitivement perdu. Par contre, tant que

ce compteur est supérieur à 1, si un lien est supprimé, il reste une copie du fichier quelque part.

```
$ cd
$ touch fic1
$ ln ficr1 fic2
$ ls
fic1 fic2
$ ls -l
-rw-r--r--   2 oracle   system         0 Jul 25 11:59 fic1
-rw-r--r--   2 oracle   system         0 Jul 25 11:59 fic2
$ ls -i
  484 fic1    484 fic2
```

L'exemple précédent montre que les hard links n'ont pas de type particulier et sont considérés comme des fichiers ordinaires. On constate que chacun a 2 liens. Logique puisque deux fichiers pointent sur le même inode. Enfin nous voyons bien en résultat du ls -i que fic1 et fic2 ont le même inode, à savoir 484.

4.5.6.2 Symbolic link

Un lien symbolique ne rajoute pas une entrée dans la table catalogue mais est en fait une sorte d'alias, un fichier spécial contenant une donnée pointant vers un autre chemin (on peut le concevoir comme une sorte de fichier texte spécial contenant un lien vers un autre fichier ou répertoire).

De par cette nature, un lien symbolique ne possède pas les limitations du hard link. Il est donc possible d'effectuer des liens entre plusieurs FileSystems, et vers des répertoires. Le cas échéant le lien se comportera à l'identique du fichier ou du répertoire pointés (un **cd** nom_lien est possible dans le cas d'un répertoire).

La suppression de tous les liens symboliques n'entraîne que la suppression de ces liens, pas du fichier pointé. La suppression du fichier pointé n'entraîne pas la suppression des liens symboliques associés. Dans le cas le lien pointe dans le vide.

```
$ rm fic2
$ ln -s fic1 fic2
$ ls -l
-rw-r--r--   1 oracle   system         0 Jul 25 11:59 fic1
lrwxrwxrwx   1 oracle   system         4 Jul 25 12:03 fic2 -> fic1
$ls -i
  484 fic1    635 fic2
$ ls -F
fic1   fic2@
```

Cet exemple montre bien qu'un lien symbolique est en fait un fichier spécial de type « l » pointant vers un autre fichier. Attention, les droits indiqués sont ceux du fichier spécial. Lors de sont utilisation, ce sont les droits du fichier ou du dossiers pointés qui prennent le dessus. On distingue le caractère « @ » indiquant qu'il s'agit d'un lien symbolique. On remarque aussi que les inodes sont différents et que les compteurs sont tous à 1.

4.5.7 Critères de recherche sur noms de fichier

Lors de l'utilisation de commandes en rapport avec le système de fichier, il peut devenir intéressant de filtrer la sortie de noms de fichiers à l'aide de certains critères, par exemple avec la commande ls. Au lieu d'afficher toute la liste des fichiers, on peut filtrer l'affichage à l'aide de divers critères et caractères spéciaux.

Caractère spécial	*Rôle*
*	Remplace une chaîne de longueur variable, même vide
?	Remplace un caractère unique quelconque
[]	Une série ou une plage de caractères
[!...]	Inversion de la recherche

Ainsi,

- **ls a*** : tous les fichiers commençant par a
- **ls a??** : tous les fichiers de trois caractères commençant par a
- **ls a??*** : tous les fichiers d'au moins trois caractères et commençant par a
- **ls [aA]*** : tous les fichiers començant par a ou A
- **ls [a-m]?*txt** : tous les fichiers commençant par les lettres de a à m, possédant au moins un second caractère avant la terminaison txt.

C'est le shell qui est chargé d'effectuer la substitution de ces caractères avant le passage des paramètres à une commande. Ainsi lors d'un

```
cp * documents
```

cp ne reçoit pas le caractère * mais la liste de tous les fichiers et répertoires du répertoire actif.

4.5.8 Verrouillage de caractères

Certains caractères spéciaux doivent être verrouillés, par exemple en cas de caractères peu courants dans un nom de fichier.

Le backslash \ permet de verrouiller un caractère unique. ls paie\ *.xls va lister tous les fichiers contenant un espace après paie.

Les guillemets "..." les guillemets permettent l'interprétation des caractères spéciaux, variables, au sein d'une chaîne.

Les apostrophes '...' verrouillent tous les caractères spéciaux dans une chaîne ou un fichier.

5 L'éditeur

L'éditeur Unix par défaut se nomme **vi** (visual editor). S'il n'est pas des plus ergonomiques par rapport à des éditeurs en mode graphique, il a l'avantage d'être disponible et d'utiliser la même syntaxe de base sur tous les Unix. Chaque Unix propose généralement une syntaxe étendue au-delà de la syntaxe de base. Pour en connaître les détails : man vi.

```
vi [options] Fichier [Fichier2 ...]
```

Trois modes de fonctionnement :

- mode commande : les saisies représentent des commandes. On y accède en appuyant sur « Echap ».
- mode saisie : saisie de texte classique
- mode ligne de commande « à la ex » :utilisation de commandes spéciales saisies et se terminant par Entrée. Accès pas la touche « : ».

5.1 Commandes de saisie

En mode commande

Commande	***Action***
a	Ajout de texte derrière le caractère actif
A	Ajout de texte en fin de ligne
i	Insertion de texte devant le caractère actif
I	Insertion de texte en début de ligne
o	Insertion d'une nouvelle ligne sous la ligne active
O	Insertion d'une nouvelle ligne au-dessus de la ligne active

5.2 Quitter

- La commande **ZZ** quitte et sauve le fichier
- :**q!** quitte sans sauver
- :**q** quitte si le fichier n'a pas été modifié
- :**w** sauve le fichier
- :**wq** ou **x** sauve et quitte

5.3 Déplacement en mode commande

Commande	***Action***
h	Vers la gauche
l	Vers la droite
k	Vers le haut
j	Vers le bas
0 (zéro)	Début de ligne (:0 première ligne)

Commande	*Action*
$	Fin de ligne (:$ dernière ligne)
w	Mot suivant
b	Mot précédent
fc	Saut sur le caractère 'c'
Ctrl + F	Remonte d'un écran
Ctrl + B	Descend d'un écran
G	Dernière ligne du fichier
NG	Saute à la ligne 'n' (:n identique)

5.4 Correction

Commande	*Action*
x	Efface le caractère sous le curseur
X	Efface le caractère devant le curseur
rc	Remplace le caractère sous le curseur par le caractère 'c'
dw	Efface le mot depuis le curseur jusqu'à la fin du mot
d$ (ou D)	Efface tous les caractères jusqu'à la fin de la ligne
dO	Efface tous les caractères jusqu'au début de la ligne.
dfc	Efface tous les caractères de la ligne jusqu'au caractère 'c'
dG	Efface tous les caractères jusqu'à la dernière ligne, ainsi que la ligne active
D1G	Efface tous les caractères jusqu'à la première ligne, ainsi que la ligne active
dd	Efface la ligne active

Ces commandes peuvent être répétées. 5Dd supprime 5 lignes.

On peut annuler la dernière modification avec la commande « u ».

5.5 Recherche dans le texte

Contrairement à un éditeur de texte classique, **vi** peut rechercher autre chose que des mots simples et fonctionne à l'aide de caractères spéciaux et de critères. La commande de recherche est le caractère « **/** ». La recherche démarre du caractère courant à la fin du fichier. Le caractère « **?** » effectue la recherche en sens inverse. On indique ensuite le critère, puis Entrée.

```
/echo
```

recherche la chaîne 'echo' dans la suite du fichier. Quand la chaîne est trouvée, le curseur s'arrête sur le premier caractère de cette chaîne.

La commande « **n** » permet de continuer la recherche dans le sens indiqué au début. La commande « **N** » effectue la recherche en sens inverse.

5.5.1 Quelques critères :

- **/[FfBb]oule** : Foule, foule, Boule, boule
- **/[A-Z]e** : Tout ce qui commence par une majuscule avec un e en deuxième position.
- **/[A-Za-Z0-9]** : tout ce qui commence par une majuscule, minuscule ou un chiffre
- **/[^a-z]** : plage négative : tout ce qui ne commence pas par une minuscule
- **/vé.o** : le point remplace un caractère, vélo, véto, véro, ...
- **/Au*o** : l'étoile est un caractère de répétition, de 0 à n caractères, Auo, Auto, Automoto, ...
- **/.*** : l'étoile devant le point, une chaîne quelconque de taille variable
- **/[A-Z][A-Z]*** : répétition du motif entre [] de 0 à n fois, recherche d'un mot comportant au moins une majuscule (en début de mot)
- **/^Auto** : le ^ indique que la chaîne recherchée devra être en début de ligne
- **/Auto$** : le $ indique que la chaîne recherchée devra être en fin de ligne

5.5.2 Quelques commandes de remplacement

Pour remplacer du texte, il faut se placer au début de la chaîne à modifier, puis taper l'une des commandes suivantes.

Commande	***Action***
cw	Remplacement du mot courant
c$	Remplacement jusqu'à la fin de la ligne
cO	Remplacement jusqu'au début de la ligne
cfx	Remplacement jusqu'au prochain caractère 'x' dans la ligne courante
c/Auto (Entrée)	Remplacement jusqu'à la prochaîne occurrence de la chaîne 'Auto'

Après cette saisie, le caractère $ apparaît en fin de zone à modifier. Il suffit alors de taper son texte et d'appuyer sur Echap.

5.6 Copier-Coller

On utilise la commande « **y** » (Yank) pour copier du texte, elle-même devant être combinée avec d'autres indications. Pour couper (déplacer), c'est la commande « **d** ». Pour coller le texte à l'endroit choisi, c'est la commande « **p** » (derrière le caractère) ou « **P** » (devant le caractère). Si c'est une ligne complète qui a été copiée, elle sera placée en-dessous de la ligne active.

Pour copier une ligne : **`yy`**

Pour copier cinq lignes : **`5yy`**

Pour placer les lignes copiées à un endroit donné : **`p`**

L'éditeur vi dispose de 26 tampons pour y stocker les données que l'on peut nommer comme on le souhaite. On utilise pour ça le « " ».

Pour copier cinq mots dans la mémoire m1 : **`"m1y5w`**

Pour coller le contenu de la mémoire m1 à un endroit donnée : **`"m1p`**

5.7 Substitution

La substitution permet de remplacer automatiquement plusieurs occurrences par une autre chaîne.

```
:[1ere ligne, dernière ligne]s/Modèle/Remplacement/[gc]
```

Les numéros de lignes sont optionnels. Dans ce cas la substitution ne se fait que sur la ligne courante. En remplacement des numéros de lignes, « **.** » détermine la ligne courante, « **1** » la première ligne, « **$** » la dernière ligne.

Le modèle est l'un des modèles présenté plus tôt. Remplacement est une chaîne quelconque qui remplacera le modèle.

Par défaut seule la première occurrence est remplacée. La lettre « **g** » indique qu'il faut remplacer toutes les occurrences. Avec « **c** », vi demande une confirmation pour chacune des occurrences.

```
:1,$s/[Uu]nix/UNIX/g
```

Cet exemple remplace, dans tout le fichier, Unix ou unix par UNIX.

5.8 Autres en ligne de commande

Commande	*Action*
:w Nom_fic	Sauve le fichier sous Nom_fic, en l'écrasant ou en le créant
:1,10w Nom_fic	Sauve les lignes 1 à 10 dans Nom_fic
:r Nom_fic	Insère le fichier Nom_fic à partir de la ligne courante
:! commande	Exécute la commande puis retourne à l'éditeur
:r! commande	Exécute la commande et insère le résultat à partir de la ligne courante
:f Nom_fic	Affiche en bas d'écran le nom du fichier, le nombre de ligne et la position actuelle
:e Nom_fic	Le fichier est chargé. Un message indique si le précédent a été modifié
:e #	Le dernier fichier chargé est affiché. Permet de commuter entre les fichiers

5.9 Commande set

La commande **set** permet de configurer l'éditeur.

- **set all** : affiche l'ensemble des options possibles
- **set number (ou nu) / nonumber (ou nonu)** : affiche / supprime les numéros de lignes.
- **set autoindent / noautoindent** : l'indentation est concervée lors d'un retour à la ligne.
- **set showmatch / noshowmatch** : lors de la saisie d'une accolade ou d'une parenthèse de fermeture, celle d'ouverture est affichée un très court instant, puis l'éditeur revient au caractère courant.
- **set showmode / noshowmode** : vi affichera une ligne d'état (INPUT MODE).
- **set tabstop=x** : définit le nombre de caractères pour une tabulation.

6 Redirections

Les redirections sont l'une des plus importantes possibilités offerte par le shell.

Par redirection, on entend la possibilité de rediriger l'affichage de l'écran vers un fichier, une imprimante ou tout autre périphérique, les messages d'erreurs vers un autre fichier, remplacer la saisie clavier par le contenu d'un fichier.

Unix utilise des canaux d'entrées/sorties pour lire et écrire ses données. Par défaut le canal d'entrée est le clavier, et le canal de sortie, l'écran. Un troisième canal, le canal d'erreur, est aussi redirigé vers l'écran.

Il est donc possible de rediriger ces canaux vers des fichiers, ou du flux texte de manière transparente pour les commandes Unix.

6.1 En sortie

On se sert du caractère « > » pour rediriger la sortie standard (celle qui va normalement sur écran). On indique ensuite le nom du fichier où seront placés les résultats de sortie.

```
$ ls -l > resultat.txt
$ cat resultat.txt
total 1
-rw-r--r--    1 Administ ssh_user        0 Jul  4 12:04 TOTO
-rw-r--r--    1 Administ ssh_user        0 Jul 25 15:13 resultat.txt
-rw-r--r--    1 Administ ssh_user      171 Jul 25 15:13 test.txt
```

Si le fichier n'existe pas, il sera créé. S'il existe, son contenu sera écrasé, même si la commande tapée est incorrecte. **Le shell commence d'abord par créer le fichier puis exécute ensuite la commande**.

Pour rajouter des données à la suite du fichier, donc sans l'écraser, on utilise la double redirection « >> ». Le résultat est ajouté à la fin du fichier.

```
$ ls -l > resultat.txt
$ date >> resultat.txt
$ cat resultat.txt
total 1
-rw-r--r--    1 Administ ssh_user        0 Jul  4 12:04 TOTO
-rw-r--r--    1 Administ ssh_user        0 Jul 25 15:13 resultat.txt
-rw-r--r--    1 Administ ssh_user      171 Jul 25 15:13 test.txt
Thu Jul 25 15:20:12  2002
```

6.2 En entrée

Les commandes qui attendent des données ou des paramètres depuis le clavier peuvent aussi en recevoir depuis un fichier, à l'aide du caractère « < ». Un exemple avec la commande « **wc** » (word count) qui permet de compter le nombre de lignes, de mots et de caractères d'un fichier.

```
$ wc  < resultat.txt
      4      29     203
```

6.3 Les canaux standards

On peut considérer un canal comme un fichier, qui possède sont propre descripteur par défaut, et dans lequel on peut ou lire ou écrire.

- Le canal d'entrée standard se nomme « **stdin »** et porte le descripteur **0**.
- Le canal de sortie standard se nomme « **stdout »** et porte le descripteur **1**.

➡ La canal d'erreur standard se nomme « **stderr »** et porte le descripteur **2**.

On peut rediriger les canaux de sortie 1 et 2 vers un autre fichier.

```
$ rmdir dossier2
rmdir: ‘dossier2’: No such file or directory
$ rmdir dossier2 2>error.log
$ cat error.log
rmdir: ‘dossier2’: No such file or directory
```

On peut aussi rediriger les deux canaux de sortie dans un seul et même fichier, en les liant. On utilise pour cela le caractère « >**&** ». Il est aussi important de savoir dans quel sens le shell interprète les redirections. Les redirections étant en principe en fin de commande, le shell recherche d'abord les caractères « <, >, >> » en fin de ligne. Ainsi si nous voulons grouper les deux canaux de sortie et d'erreur dans un même fichier, il faut procéder comme suit.

```
$ ls -l > resultat.txt 2>&1
```

La sortie 2 est redirigée vers la sortie 1, donc les messages d'erreurs passeront par la sortie standard. Puis le résultat de la sortie standard de la commande ls est redirigé vers le fichier resultat.txt. Ce fichier contiendra donc à la fois la sortie standard et la sortie d'erreur.

On peut aussi utiliser à la fois les deux types de redirection.

```
$ wc < resultat.txt > compte.txt
$ cat compte.txt
      4      29     203
```

On peut aussi se reporter à la commande « **tee »** page 36.

6.4 Filtre : définition

Un **filtre** (ou une commande filtre) est un programme sachant écrire et lire des données par les canaux standards d'entrée et de sortie. Il en modifie ou traite éventuellement le contenu. wc est un filtre.

Nous nous attarderons sur quelques filtres plus tard, mais en voici quelques uns : **more** (affiche les données page par page), **sort** (tri des données), **grep** (critères de recherche)

6.5 Pipelines / tubes

Les redirections d'entrée/sortie telles que nous venons de les voir permettent de rediriger les résultats vers un fichier. Ce fichier peut ensuite être réinjecté dans un filtre pour en extraire d'autres résultats. Cela oblige à taper deux lignes : une pour la redirection vers un fichier, l'autre pour rediriger ce fichier vers le filtre. Les **tubes** ou **pipes** permet de rediriger directement le canal de sortie d'une commande vers le canal d'entrée d'une autre. Le caractère permettant cela est « | ».

```
$ ls -l > resultat.txt
$ wc < resultat.txt
```

devient

```
$ ls -l | wc
```

Il est possible de placer plusieurs « | » sur une même ligne.

```
$ ls -l | wc | wc
      1       3      24
```

La première commande n'est pas forcément un filtre. L'essentiel est qu'un résultat soit délivré. Idem pour la dernière commande qui peut par exemple être une commande d'édition ou d'impression. Enfin, la dernière commande peut elle-même faire l'objet d'une redirection en sortie.

```
$ ls -l | wc > resultat.txt
```

7 Les droits d'accès

Nous avons indiqué que le rôle d'un système d'exploitation est aussi d'assurer la sécurité et l'accès aux données, ce qui se fait grâce au mécanisme des droits. Chaque fichier se voit attribué des droits qui lui sont propres, des autorisations d'accès individuelles. Lors d'un accès le système vérifie si celui-ci est permis.

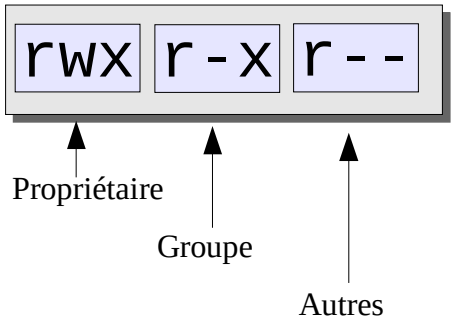
A sa création par l'administrateur, un utilisateur se voit affecté un **UID** (« **User Identifier**») unique. Les utilisateurs sont définis dans le fichier /etc/passwd. De même chaque utilisateur est rattaché à au moins un groupe (groupe principal), chaque groupe possédant un identifiant unique, le **GID** (« **Group Identifier**»). Les groupes sont définis dans /etc/group.

La commande **id** permet d'obtenir ces informations. En interne, le système travaille uniquement avec les UID et GID, et pas avec les noms par eux-mêmes.

```
weas02> id
uid=75(oracle) gid=102(dba)
```

A chaque fichier (inode) sont associés un UID et un GID définissant son propriétaire et son groupe d'appartenance. On peut affecter des droits pour le propriétaire, pour le groupe d'appartenance et pour le reste du monde. On distingue de ce fait trois cas de figure

1) UID de l'utilisateur identique à l'UID défini pour le fichier : cet utilisateur est propriétaire du fichier.
2) Les UID sont différents : le système vérifie si le GID de l'utilisateur est identique au GID du fichier : si oui l'utilisateur appartient au groupe associé au fichier.
3) Dans les autres cas (aucune correspondance) : il s'agit du reste du monde (others), **ni les propriétaires, ni appartenant au groupe**.



Sortie de la commande ls -l :

d	rwxr-xr-x	1	oracle	dba	466	Feb 8 2001	oracle
-	rwxrwxrwx	1	oracle	dba	14536	Feb 8 2001	output.log
-	rwxr-x---	1	oracle	dba	1456	Feb 8 2001	launch.sh

Sur la première ligne du tableau, le répertoire oracle appartient à l'utilisateur oracle et au groupe dba, et possède les droits rwxr-xr-x.

7.1 Signification

Droit	*Signification*
	Général
r	Readable (lecture)
w	Writable (écriture)
x	Executable (exécutable comme programme)
	Fichier normal
r	Le contenu du fichier peut être lu, chargé en mémoire, visualisé, recopié.
w	Le contenu du fichier peut être modifié, on peut écrire dedans. La suppression n'est pas forcément liée à ce droit (voir droits sur répertoire).
x	Le fichier peut être exécuté depuis la ligne de commande, s'il s'agit soit d'un programme binaire (compilé), soit d'un script (shell, perl, ...)
	Répertoire
r	Les éléments du répertoire (catalogue) sont accessibles en lecture. Sans cette autorisation, le ls et les critères de filtres sur le répertoire et son contenu ne sont pas possibles. **Ce droit ne suffit pas pour entrer dans le catalogue**.
w	Les éléments du répertoire (catalogue) sont modifiables et il est possible de créer, renommer et supprimer des fichiers dans ce répertoire. On voit donc que c'est ce droit qui contrôle l'autorisation de suppression d'un fichier **même si on est pas propriétaire des fichiers du répertoire**.
x	Le catalogue peut être accédé par cd et listé. Sans cette autorisation il est impossible d'accéder et d'agir sur son contenu qui devient verrouillé. Avec uniquement ce droit les fichiers et répertoires inclus dans celui-ci peuvent être accédés mais il faut alors obligatoirement connaître leur nom.

Ainsi pour un fichier :

rwx	r-x	r--
Droits de l'utilisateur, en lecture, écriture et exécution	Droits pour les membres du groupe et lecture et exécution	Droits pour le reste du monde en lecture uniquement

7.2 Modification des droits

Lors de sa création, un fichier ou un répertoire dispose de droits par défaut. On utilise la commande **chmod** (change mode) pour modifier les droits sur un fichier ou un répertoire. Il existe de méthodes pour modifier ces droits : par la forme symbolique et par la base 8. Seul le propriétaire d'un fichier peut en modifier les droits (sauf l'administrateur système).

Le chmod sur un lien symbolique est possible comme sur tout autre fichier, mais cela ne modifie pas les droits du lien par lui-même mais les droits du fichier pointé.

7.2.1 Par symboles

La syntaxe est la suivante :

```
chmod modifications Fic1 [Fic2...]
```

S'il faut modifier les droits de l'utilisateur, on utilisera le caractère « **u** ». pour les droits du groupe, le caractère « **g** », pour le reste du monde le caractère « **o** », pour tous le caractère « **a** ».

Pour ajouter des droits, on utilise le caractère « **+** », pour en retirer le caractère « **-** », et pour ne pas tenir compte des paramètres précédents le caractère « **=** ».

Enfin, le droit d'accès par lui-même : « **r** », « **w** » ou « **x** ».

On peut séparer les modifications par un espace, et cumuler plusieurs droits dans une même commande.

```
$ ls -l
total 0
-rw-r--r--   1 oracle   system         0 Aug 12 11:05 fic1
-rw-r--r--   1 oracle   system         0 Aug 12 11:05 fic2
-rw-r--r--   1 oracle   system         0 Aug 12 11:05 fic3
$ chmod g+w fic1
$ ls -l fic1
-rw-rw-r--   1 oracle   system         0 Aug 12 11:05 fic1
$ chmod u=rwx,g=x,o=rw fic2
$ ls -l fic2
-rwx--xrw-   1 oracle   system         0 Aug 12 11:05 fic2
$ chmod o-r fic3
$ ls -l fic3
-rw-r-----   1 oracle   system         0 Aug 12 11:05 fic3
```

7.2.2 Par base 8

La syntaxe est identique à celle des symboles. A chaque droit correspond une valeur **octale c'est à dire de zéro (0) à sept (7)**, positionnelle et cumulable.

Propriétaire			***Groupe***			***Reste du monde***		
r	**w**	**x**	**r**	**w**	**x**	**r**	**w**	**x**
400	200	100	40	20	10	4	2	1

Pour obtenir le droit final il suffit d'additionner les valeurs. Par exemple si on veut rwxrw-rw- alors on fera 400+200+100+40+20+4+2=766, et pour rw-r—r- 400+200+40+4=644.

```
$ chmod 766 fic1
$ ls -l fic1
-rwxr-xr-x   1 oracle   system         0 Aug 12 11:05 fic1
$ chmod 644 fic2
$ ls -l fic2
-rw-r--r--   1 oracle   system         0 Aug 12 11:05 fic2
```

7.3 Masque des droits

Lors de la création d'un fichier ou d'un répertoire et qu'on regarde ensuite leurs droits, on obtient généralement rw-r—r-- (644) pour un fichier et rwxr-xr-x (755) pour un répertoire. Ces valeurs sont contrôlées par un masque, lui-même modifiable par la commande **umask**. la commande prend comme paramètre une valeur octale qui sera soustraite aux droits d'accès maximum. Par défaut,

tous les fichiers sont créés avec les droits 666 (rw-rw-rw) et les répertoires avec les droits 777 (rwxrwxrwx), puis le masque est appliqué.

Sur la plupart des Unix, le masque par défaut est 022, soit ----w--w-. Pour obtenir cette valeur, on tape umask sans paramètre.

Pour un fichier :

```
Maximum rw-rw-rw- (666)
Retirer ----w--w- (022)
Reste   rw-r--r-- (644)
```

Pour un répertoire :

```
Maximum rwxrwxrwx (777)
Retirer ----w--w- (022)
Reste   rwxr-xr-x (755)
```

ATTENTION : la calcul des droits définitifs (effectifs) n'est pas une simple soustraction de valeurs octales ! Le masque retire des droits mais n'en ajoute pas.

Pour un fichier :

```
Maximum rw-rw-rw- (666)
Retirer ---r-xrw- (056)
Reste   rw--w---- (620) et PAS 610 !
```

7.4 Changement de propriétaire et de groupe

Il est possible de changer le propriétaire et le groupe d'un fichier à l'aide des commandes **chown** (change owner) et **chgrp** (change group).

```
chown utilisateur fic1 [Fic2...]
chgrp groupe fic1 [Fic2...]
```

Sur les UNIX récents seul root peut utiliser chown. La commande chgrp peut être utilisée par n'importe qui à condition que cet utilisateur fasse aussi partie du nouveau groupe.

En précisant le nom d'utilisateur (ou de groupe), le système vérifie d'abord leur existence. On peut préciser un UID ou un GID, dans ce cas le système n'effectuera pas de vérification.

Pour les deux commandes on peut préciser l'option **-R**, dans ce cas les droits seront changés de manière récursive. Les droits précédents et l'emplacement du fichier ne sont pas modifiés. Enfin sous certains Unix il est possible de modifier en une seule commande à la fois le propriétaire et le groupe.

```
chown utilisateur[:groupe] fic1 [fic2...]
chown utilisateur[.groupe] fic1 [fic2...]

$ chgrp dba fic1
$ ls -l fic1
-rwxr-xr-x   1 oracle   dba            0 Aug 12 11:05 fic1
```

7.5 Extractions des noms et chemins

La commande **basename** permet d'extraire le nom du fichier dans un chemin.

```
$ basename /tmp/seb/liste
liste
```

La commande **dirname** effectue l'inverse, elle extrait le chemin.

```
$ dirname /tmp/seb/liste
/tmp/seb
```

8 Les filtres et utilitaires

Rappel : un **filtre** (ou une commande filtre) est un programme sachant écrire et lire des données par les canaux standards d'entrée et de sortie. Il en modifie ou traite éventuellement le contenu. **wc** est un filtre. Les utilitaires sans être obligatoirement des filtres permettent un certain nombre d'actions sur des fichiers ou leur contenu comme le formatage ou l'impression.

8.1 Recherche de lignes

Il s'agit d'extraire des lignes d'un fichier selon divers critères. Pour cela on dispose de trois commandes **grep**, **egrep** et **fgrep** qui lisent les données soit depuis un fichier d'entrée, soit depuis le canal d'entrée standard.

8.1.1 grep

La syntaxe de la commande **grep** est

grep [Options] modèle [Fichier1...]

Le modèle se compose de critères de recherche ressemblant beaucoup aux critères déjà exposés pour vi par exemple. Il ne faut pas oublier que ces critères doivent être interprétés par la commande grep et pas par le shell. Il faut donc verrouiller tous les caractères.

```
$ cat fic4
Cochon
Veau
Boeuf
rat
Rat
boeuf
$ grep \^[\b\B\] fic4
Boeuf
boeuf
```

La commande grep peut aussi prendre quelques options intéressantes.

- **-v** effectue la recherche inverse : toutes les lignes ne correspondant pas aux critères sont affichées
- **-c** ne retourne que le nombre de lignes trouvées sans les afficher
- **-i** ne diférencie pas les majuscules et les minuscules
- **-n** indique le numéro de ligne pour chaque ligne trouvée
- **-l** dans le cas de fichiers multiples, indique dans quel fichier la ligne a été trouvée.

```
$ grep -i \^\[\b\] fic4
Boeuf
boeuf
```

Nous voyons que la syntaxe dans le cas de critères de recherches est assez lourde.

8.1.2 egrep

La commande **egrep** étend les critères de recherche et peut accepter un fichier de critères en entrée. Attention cependant egrep est lent et consomme beaucoup de ressources.

```
egrep -f fichier_critère Fichier_recherche
```

Caractère spécial	*Signification*
+	Répétition, le caractère placé devant doit apparaître au moins une fois.
?	Le caractère situé devant doit apparaître une fois ou pas du tout.
|	Ou logique, l'expression située avant ou après doit apparaître
(...)	groupage de caractères

bon(jour|soir) sortira bonjour et bonsoir.

8.1.3 fgrep

La commande **fgrep** est un grep simplifié et rapide (fast grep). Elle accepte aussi un fichier de critères de recherche mais il s'agit là de critères simples, sans caractères spéciaux. On saisira dans le fichier de critère un critère simple (du texte et des chiffres) par ligne).

8.2 Colonnes et champs

La commande **cut** permet de sélectionner des colonnes et des champs (découpage vertical) dans un fichier.

8.2.1 Colonnes

La syntaxe est la suivante :

```
cut -cColonnes [fic1...]
```

Le format de sélection de colonne est le suivant :

- **La numérotation des colonnes démarre à 1**.
- une colonne seule, (ex -c2 pour la colonne 2)
- une plage (ex -c2-4 pour les colonnes 2, 3 et 4)
- une liste de colonnes (ex -c1,3,6 pour les colonnes 1, 3 et 6)
- les trois en même temps (ex -c1-3,5,6,12-)

```
$ cat liste
Produit prix    quantites
souris  30      15
disque  100     30
ecran   300     20
clavier 45      30
$ cut -c1-5 liste
Produ
souri
disqu
ecran
clavi
$ cut -c1-3,10-12,15
Prorx   quantites
sou0    15
dis0    30
ecr0    20
cla530
```

8.2.2 Champs

La commande **cut** permet aussi de sélectionner des champs. Ces champs doivent être par défaut délimités par une tabulation, mais l'option -d permet de sélectionner un autre caractère (espace, ; ...). La sélection des champs est identique à celle des colonnes. **Leur numérotation démarre à 1**.

```
cut -dc -fChamps [fic1...]

$ cat liste
Produit prix    quantites
souris  30      15
dur     100     30
disque  100     30
ecran   300     20
clavier 45      30
carte   45      30
$ cut -f1 liste
Produit
souris
dur
disque
ecran
clavier
carte
$ cut -f1,3 liste
Produit quantites
souris  15
dur     30
disque  30
ecran   20
clavier 30
carte   30

$ cat /etc/group
system:*:0:root,ftp,sa_sgd,arret,ptrladm,ptrlagt
daemon:*:1:daemon
uucp:*:2:uucp
kmem:*:3:root,ingres
mem:*:3:root,ingres
bin:*:4:bin,adm
sec:*:5:
mail:*:6:mail
terminal:*:7:
tty:*:7:root
news:*:8:uucp
$ cut -d: -f1,3
system:0
daemon:1
uucp:2
kmem:3
mem:3
bin:4
sec:5
mail:6
terminal:7
tty:7
news:8
```

8.3 Compter les lignes

La commande **wc** (word count) permet de compter les lignes, mots et caractères.

wc [-l] [-c] [-w] [-w] fic1

- -l : compte le nombre de lignes
- -c : compte le nombre d'octets
- -w : compte le nombre de mots
- [-m : compte le nombre de caractères]. **Cette option n'est pas proposée partout**.

```
$ wc liste
        12        48       234 liste
```

Le fichier liste contient 12 lignes, 48 mots et 234 caractères.

8.4 Tri de lignes

La commande **sort** permet de trier des lignes. Par défaut le tri s'effectue sur tout le tableau et en ordre croissant. Le tri est possible sur un ou plusieurs champs. Le séparateur de champs par défaut est la tabulation ou au moins un espace. S'il y a plusieurs espaces, le premier est le séparateur, les autres des caractères du champ. **La numérotation des champs démarre à 0**.

```
sort [options] [+pos1 [-pos2] ...] [fic1...]
```

+pos1 est le premier champ, -pos2 le dernier.

```
$ cat liste
Produit objet   prix    quantites
souris  optique 30      15
dur     30giga  100     30
dur     70giga  150     30
disque  zip     12      30
disque  souple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
clavier 105     45      30
clavier 115     55      30
carte   son     45      30
carte    video   145     30
$ sort +0 liste
Produit objet   prix    quantites
carte   son     45      30
carte   video   145     30
clavier 105     45      30
clavier 115     55      30
disque  souple  10      30
disque  zip     12      30
dur     30giga  100     30
dur     70giga  150     30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
souris  optique 30      15
```

Quelques options

Option	*Rôle*
-d	Dictionnary sort (tri dictionnaire). Ne prend comme critère de tri que les lettres les chiffres et les espaces.
-n	Tri numérique, idéal pour le colonnes de chiffres
-b	Ignore les espaces en début de champ
-f	Pas de différences entre majuscules et minuscules (conversion en minuscules puis tri)
-r	Reverse, tri en ordre décroissant.
-tc	Nouveau délimiteur de champ c.

Exemple, tri numérique sur le prix par produits et quantités en ordre décroissant

```
$ sort -n -r +2 -3 liste
ecran   19      500     20
ecran   17      300     20
ecran   15      150     20
dur     70giga  150     30
carte   video   145     30
dur     30giga  100     30
clavier 115     55      30
clavier 105     45      30
carte   son     45      30
souris  optique 30      15
disque  zip     12      30
disque  souple  10      30
Produit objet   prix    quantites
```

Il est aussi possible de démarrer le tri à partir d'un certain caractère d'un champ. Pour cela on spécifie le « .pos » : +0.3 commencera le tri à partir du quatrième caractère du champ 0.

8.5 Joindre deux fichiers

La commande **join** permet d'effectuer une jointure de deux fichiers en fonction d'un champ commun. les deux fichiers doivent être triés sur les champs indiqués. **La numérotation des champs démarre à 0**.

```
join [-tc] [-j1 n] [-j2 n2] fic1 fic2
join [-tc] [-1 n] [-2 n2] fic1 fic2
```

L'option -t indique le séparateur, -j1 (-1) le champ du premier fichier et -j2 (-2) le champ du second fichier. Par exemple :

```
$ join -t: -j1 4 -j2 3 /tmp/seb/passwd /tmp/seb/group
```

va joindre passwd et group en fonction de leur champ commun GID.

8.6 remplacement de caractères

La commande **tr** permet de substituer des caractères à d'autres et n'accepte que des données depuis le canal d'entrée standard, ou par des fichiers en redirection d'entrée.

```
tr [options] original destination
```

L'original et la destination représentent un ou plusieurs caractères. Les caractères originaux sont remplacés par les caractères de destination dans l'ordre indiqué. Les crochets permettent de définir

des plages.

Par exemple, remplacer le o par le e et le i par le a.

```
$ cat liste | tr "oi" "ea"
Preduat ebjet   prax    quantates
seuras  eptaque 30      15
dur     30gaga  100     30
dur     70gaga  150     30
dasque  zap     12      30
dasque  seuple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
clavaer 105     45      30
clavaer 115     55      30
carte   sen     45      30
carte   vadee   145     30
```

Avec cette commande on peut convertir une chaîne en majuscules ou en minuscules, passage de toutes les minuscules en majuscules :

```
$ cat liste | tr "[a-z]" "[A-Z]"
PRODUIT OBJET   PRIX    QUANTITES
SOURIS  OPTIQUE 30      15
DUR     30GIGA  100     30
DUR     70GIGA  150     30
DISQUE  ZIP     12      30
DISQUE  SOUPLE  10      30
ECRAN   15      150     20
ECRAN   17      300     20
ECRAN   19      500     20
CLAVIER 105     45      30
CLAVIER 115     55      30
CARTE   SON     45      30
CARTE   VIDEO   145     30
```

8.7 Visualiser du texte

Rien n'empêche de détourner un quelconque flux pour l'afficher sur écran ou imprimante. Voici quelques commandes.

- Page par page : **pg**, **more**, **less**
- en bloc : **cat**
- création d'une bannière : **banner**
- formatage pour impression : **pr**

On aura plus d'informations avec la commande man.

8.7.1 Début d'un fichier

Pour voir le début d'un fichier on utilise la commande **head**.

```
head [-c nbcar] [-n nblignes] [fic1...]
```

L'option -c n'est pas disponible sous tous les Unix et permet de préciser un nombre d'octets d'en-tête à afficher. Par défaut 10 lignes sont affichées. L'option -n permet d'indiquer le nombre de lignes à afficher. Sur certains Unix c'est la syntaxe suivante qu'il faudra utiliser.

```
head [-nblignes] [fic1...]
```

```
$ head -n 3 liste
Produit objet   prix    quantites
souris  optique 30      15
dur     30giga  100     30
```

8.7.2 Fin et attente de fichier

Pour voir les dernières lignes d'un fichier, on utilise la commande **tail**.

```
tail [+/-valeur[b/c]] [-f] [fic1...]
```

Comme pour head, par défaut les dix dernières lignes sont affichées. La valeur -nblignes permet de modifier cet état. Préciser c indique un nombre de caractères. Un b indique un nombre de blocs (512 octets par bloc).

Un + inverse l'ordre de la commande, et devient un head (tail +10 <=> head -n 10).

Enfin l'option -f laisse le fichier ouvert. Ainsi si le fichier continue d'être rempli (par exemple un fichier trace), son contenu s''affichera en continu sur l'écran jusqu'à interruption volontaire de l'utilisateur (ctrl+C).

```
$ tail -5 liste
ecran   19      500     20
clavier 105     45      30
clavier 115     55      30
carte   son     45      30
carte   video   145     30
$ tail -10c liste
eo      145     30
```

Nous aurons l'occasion de voir l'effet de la commande tail -f plus tard lors de la programmation shell.

8.8 Dupliquer le canal de sortie standard

Dans certains cas, comme par exemple la génération de fichiers traces, il peut être nécessaire de devoir à la fois placer dans un fichier le résultat d'une commande et de filtrer ce même résultat avec une autre commande. On utilise pour cela la commande **tee** qui permet de dupliquer le flux de données. Elle lit le flux de données provenant d'une autre commande par le canal d'entrée, l'écrit dans un fichier et restitue ce flux à l'identique par le canal de sortie. Par défaut le fichier généré écrase l'ancien s'il existe.

```
tee [-a] nom_fic
```

L'option -a signifie append. Dans ce cas le fichier n'est pas écrasé mais complété à la fin. Par exemple, on veut obtenir à la fois dans un fichier la liste des noms d'utilisateurs et afficher leur nombre sur écran.

```
$ cat /etc/passwd | cut -d: -f1 | tee users | wc -l
        65
$ cat users
root
nobody
nobodyV
daemon
bin
uucp
uucpa
auth
cron
```

```
lp
tcb
...
```

8.9 Comparaisons de fichiers

Les deux commandes permettant de comparer le contenu de deux fichiers, ou d'un fichier et un flux sont les commandes **diff** et **cmp**.

8.9.1 diff

La commande **diff** indique les modifications à apporter aux deux fichiers en entrée pour que leur contenu soit identique.

```
diff [-b] [-e] fic1 fic2
```

L'option -b permet d'ignorer les espaces (blank), et l'option -e permet de générer un script ed (nous ne l'utiliserons pas). Cette commande renvoie trois types de messages :

1) **APPEND** : `ligne1 a ligne3,ligne4`, ex `5 a 6,8` veut dire : à la ligne 5 de fic1 il faut raccrocher les lignes 6 à 8 de fic2 pour que leurs contenus soient identiques.

2) **DELETE** : `ligne1,ligne2 d ligne3`, ex `7,9 d 6` veut dire : les lignes 7 à 9 de fic1 doivent être supprimées, elles n'existent pas derrière la ligne 6 de fic2.

3)**CHANGE** : `ligne1,ligne2 c ligne3,ligne4,` ex 8,12 c 9,13 veut dire : les lignes 8 à 12 de fic1 doivent être échangées contre les lignes 9 à 13 de fic2.

Dans tous les cas, le signe "<" indique les lignes de fic1 concernées, et le signe ">" les lignes de fic2 concernées.

```
$ cat liste
Produit objet   prix    quantites
souris  optique 30      15
dur     30giga  100     30
dur     70giga  150     30
disque  zip     12      30
disque  souple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
clavier 105     45      30
clavier 115     55      30
carte   son     45      30
carte   video   145     30
$ cat liste2
Produit objet   prix    quantites
souris  boutons 30      15
dur     30giga  100     30
dur     70giga  150     30
disque  zip     12      30
disque  souple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
ecran   21      500     20
clavier 105     45      30
clavier 115     55      30
```

Le fichier liste est l'original. Dans liste2, la deuxième ligne a été modifiée, une ligne écran à été ajoutée et les deux dernières lignes ont été supprimées.

```
$ diff liste liste2
2c2
< souris        optique 30      15
---
> souris        boutons 30      15
9a10
> ecran 21      500     20
12,13d12
< carte son     45      30
< carte video   145     30
```

- 2c2 : les lignes 2 de liste et liste2 doivent être échangées (elles doivent concorder soit en optique, soit en boutons)
- 9a10 : après la ligne 9 de liste (écran 19) il faut ajouter la ligne 10 (écran 21) de liste2
- 12,13d12 : les lignes 12 et 13 de liste (carte son et vidéo) doivent être supprimés car elles n'existent pas après la ligne 12 de liste2.

8.9.2 cmp

La commande **cmp** compare les fichiers caractère par caractère. Par défaut la commande s'arrête dès la première différence rencontrée et indique la position de l'erreur.

```
cmp [-l] [-s] fic1 fic2
```

L'option -l détaille toutes les différences en trois colonnes. La première colonne représente le numéro de caractère, la deuxième la valeur octale ASCII du caractère concerné de fic1 et et troisième la valeur octale ASCII du caractère concerné de fic2.

L'option -s retourne uniquement le code d'erreur (non visible), nous verrons plus tard comment l'exploiter.

```
$ cmp liste liste2
liste liste2 differ: char 38, line 2
$ cmp -l liste liste2
    38 157 142
    39 160 157
    40 164 165
    41 151 164
    42 161 157
    43 165 156
    44 145 163
   182 143 145
   183 154 143
  ...
```

8.10 Outils divers

Ici sont regroupés quelques outils intéressants non abordés auparavant.

8.10.1 Archivage et compression

- **compress [-c] [-d] [-f]**, **uncompress [-c]** et **zcat** : permettent de compresser, décompresser et afficher un fichier compressé au format d'encodage Lempel-Ziv. Cette méthode est moins performante que la suivante. Avec l'option -c le résultat (compressé ou décompressé) est envoyé par le canal de sortie standard. L'option -d décompresse le fichier, l'option -f force la compression même si la taille du fichier ne varie pas. L'extension par défaut du fichier est « **.Z** ».
- **gzip [-c] [-d] [-f] [-1->9] [-r]** et **gunzip** : version Gnu améliorée de compress et uncompress. La méthode utilisée est la même mais améliorée. Les options -c et -d sont identiques. L'option -f

évite la demande de confirmation avant écrasement des fichiers, les options de -1 (fast) à -9 (best) déterminent le taux de compression. L'option -r indique la récursivité. L'extension par défaut est « .gz ».

- **tar [-c] [-x] [-t] [-v] [-f]** : manipulation d'archives cassette, par extension archivage par concaténation d'une série de fichiers compressés ou non. L'option -c permet la création de l'archive, -x l'extraction, -t liste le contenu de l'archive, -v inhibe le mode silence, -f permet de préciser le nom de l'archive. Par défaut la commande tar est récursive. L'extension par défaut est « .tar ». La version Gnu de tar propose l'option -z qui compresse l'archive tar avec gzip. Ex : tar cvf archive.tar /home/toto ; gzip archve.tar.
- **cpio [-o] [-i] [-p] [-t] [-v] [-d] [-B] [-c]**: copie de fichiers depuis et vers une archive. On a tendance aujourd'hui à utiliser tar car la syntaxe de cpio est un peu plus complexe. Cependant il est utile de la connaître car beaucoup de sauvegardes fonctionnent de cette manière. Les options sont -o (output) -i (input) -p (porting) -B (travailler avec des blocs de 5120 octets au lieu de 512) -c (créé une entête ASCII) -t (créé une table des matières) -v (affiche les noms des fichiers) -d (force la création de l'arborescence lors de la copie). Ex : find . -print | cpio -ocv > archive.cpio.

8.10.2 Espace disque et memoire

- **df [-k] [-F/t type]**: affiche des statistiques sur l'utilisation de l'espace disque. L'option -k permet d'indiquer les valeurs en kilooctets au lieu de blocs de 512 octets, -F (ou -t suivant l'Unix) en fonction du type de filesystem.
- **du [-a] [-k] [chemin]**: informations sur l'utilisation du filesystem, mais par répertoire et fichiers. L'option -k force l'affichage en kilo-octets, et -a pour tous les fichiers et pas seulement pour le chemin.
- **free** : sur certains Unix, donne des statistiques sur l'occupation de la mémoire physique et la mémoire virtuelle.
- **vmstat** : donne des informations sur la mémoire virtuelle et physique. Les informations sont complètes mais difficiles à lire.

8.10.3 Informations diverses

- **uptime** : donne des informations sur l'état actuel du système comme l'heure courante, le temps depuis le dernier démarrage, le nombre d'utilisateurs actuellement connectés, et la charge de la machine.
- **uname [-a] [-m] [-r] [-s] [-v]**: informations sur la version du système, -m le type de matériel -r le numéro de version du système, -s le type de système (os), -v la version de l'os et -a toutes les informations. Sur certains Unix -p donne le type de processeur.
- **w** : statistiques sur le système : affiche une entête comme uptime puis la liste des utilisateurs connectés.

9 L’impression

Il existe principalement deux standards d’impression sous Unix, l’un sous System V et l’autre sous BSD. Un troisième tend à s'installer depuis quelques temps, CUPS (Common Unix Printing System), qui tente de fédérer les divers systèmes d'impression sous Unix.

Quelque soit le standard, le principe est le même. A chaque imprimante déclarée (généralement dans /etc/printcap) correspond une file d'attente (**queue**). L'ensemble de ces files d'attente est géré par un service indépendant (**daemon**). Ces deux principes permettent une impression multi-utilisateur (les jobs d'impression sont en file d'attente, **job queues**), et en réseau (le service peut être accedé depuis une autre machine distante).

En rêgle générale toutes les imprimantes savent directement imprimer du texte brut ASCII en 80 colonnes. Pour imprimer des documents formatés, des images, on peut utiliser un pilote. On parle en fait de filtre d'impression. Le filtre peut être un script ou un binaire qui récupère le flux entrant (texte, image, document, postscript, ...), l'identifie et à l'aide de traitements associés le transforme en langage compréhensible par l'imprimante (Postscript, PCL, Canon, Epson, WPS, ...).

9.1 System V

- **`lp [-dImprimante] [-nChiffre] fic1`** : imprime le contenu du fichier fic1. L'option -d permet de choisir l'imprimante, -n le nombre d'exemplaires.
- **`lpstat [-d] [-s] [-t] [-p]`** : informations sur l'impression. L'option -d affiche le nom de l'imprimante par défaut, -s un état résumé des imprimantes, -t la totalité des informations (statut, jobs, ...), -p [liste] uniquement les informations sur les imprimantes incluses dans la liste.
- **`cancel [ids] [printers] [-a] [-e]`** : supprime les tâches d'impression ids des imprimantes printers. L'option -a supprime tous les jobs de l'utilisateur, -e tous les jobs (seulement l'administrateur).
- Sur certains systèmes comme SCO, on peut trouver les commandes **`enable`** et **`disable`** qui prennent comme paramêtre le nom de l'imprimante, permettant d'en autoriser ou interdire l'accès.
- Le service (ou daemon) s'appelle généralement **lpd** (line printer daemon) ou **lpsched** (line printer scheduler).

9.2 BSD

- **`lpr [-Pimprimante] [-#copies] fic1`** : imprime le contenu du fichier fic1. L'option -P permet de spécifier l'imprimante, -# le nombre de copies.
- **`lpq [-Pimprimante]`** : indique l'état et la liste des jobs pour l'imprimante éventuellement spécifiée par l'option -P.
- **`lprm [-Pimprimante] [-] [ids]`** : permet de supprimer un job de l'imprimante spécifiée par l'option -P, l'option – supprime tous les jobs de l'utilisateur, ids représente une liste de jobs à supprimer.
- Sur certains Unix on peut trouver la commande **lpc**, qui est une sorte de petit shell permettant de contrôler les imprimantes et les jobs.
- Le service s'appelle généralement **lpd.**

9.3 CUPS

CUPS est un système d'impression Unix, orienté réseau, tendant à fédérer les systèmes d'impression System V et BSD, et basé sur le protocole IPP (Internet Printing Protocol). Basé sur une architecture plus complexe, il est pourtant bien plus structuré et simple d'utilisation, notamment une configuration et administration centralisée depuis une interface HTTP, des règles de conversion basées sur les types MIME et des fichiers de description d'imprimante standards (PPD, PostScript Printer Description). Cups reprend les commandes System V et BSD déjà abordées pour plus de simplicité. Enfin CUPS dispose de sa propre API permettant de créer des interfaces utilisateur pouvant s'intégrer dans des environnement graphiques ou des interfaces d'administration.

9.4 Exemples

```
$ lpq
Warning: no daemon present
Rank   Pri Owner      Job  Files                                 Total Size
1st    0   root       0    pirge_325.log                         0 bytes
2nd    0   root       1    test                                  0 bytes
3rd    0   root       2    test                                  0 bytes
4th    0   root       3    (standard input)                      896 bytes
$ lpstat
qms2060:
Warning: no daemon present
Rank   Pri Owner      Job  Files                                 Total Size
1st    0   root       0    pirge_325.log                         0 bytes
2nd    0   root       1    test                                  0 bytes
3rd    0   root       2    test                                  0 bytes
4th    0   root       3    (standard input)                      896 bytes

Lexmark113:
unable to get hostname for remote machine
Warning: no daemon present

Rank   Pri Owner      Job  Files                                 Total Size
1st    0   root       0    load_logi15034.log                    0 bytes
2nd    0   root       1    toutou                                0 bytes
3rd    0   root       2    toutou                                0 bytes
```

10 Les processus

10.1 Définition et environnement

Un **processus** représente à la fois un programme en cours d'exécution et tout son environnement d'exécution (mémoire, état, identification, propriétaire, père ...). Voir les cours de Système d'exploitation.

Voici une liste des données d'identification d'un processus :

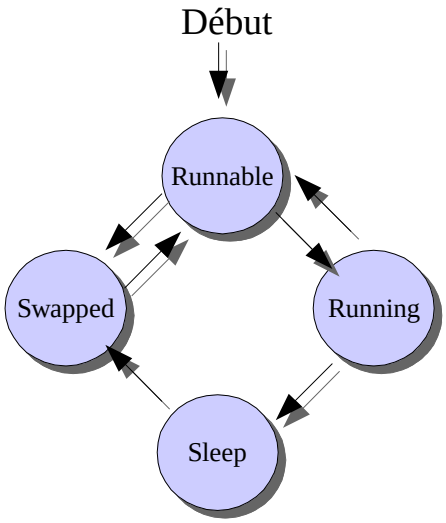
- **Un numéro de processus unique PID** (Process ID) : chaque processus Unix est numéroté afin de pouvoir être différencié des autres. Le premier processus lancé par le système est 1 et il s'agit d'un processus appelé généralement init. On utilise le PID quand on travaille avec un processus. Lancer 10 fois le même programme (même nom) donne 10 PID différents.
- **Un numéro de processus parent PPID** (Parent Process ID) : chaque processus peut lui-même lancer d'autres processus, des processus enfants (child process). Chaque enfant reçoit parmis les informations le PID du processus père qui l'a lancé. Tous les processus ont un PPID sauf le processus 0 qui est un pseudo-processus représentant le démarrage du système (créé le 1init).
- **Un numéro d'utilisateur et un numéro de groupe** : correspond à l'UID et au GID de l'utilisateur qui a lancé le processus. C'est nécessaire pour que le système sache si le processus à le droit d'accéder à certaines ressources ou non. Les processus enfants héritent de ces informations. Dans certains cas (que nous verrons plus tard) on peut modifier cet état.
- **Durée de traitement et priorité** : la durée de traitement correspond au temps d'exécution écoulé depuis le dernier réveil du processus. Dans un environnement multitâches, le temps d'exécution est partagé entre les divers processus, et tous ne possèdent pas la même priorité. Les processus de plus haute priorité sont traités en premier. Lorsqu'il est inactif sa priorité augmente afin d'avoir une chance d'être exécuté. Lorsqu'il est actif, sa priorité baisse afin de laisser sa place à un autre. C'est le scheduler du système qui gère les priorités et les temps d'exécution.
- **Répertoire de travail actif** : A son lancement, le répertoire courant (celui depuis lequel le processus a été lancé) est transmis au processus. C'est ce répertoire qui servira de base pour les chemins relatifs.
- **Fichiers ouverts** : table des descripteurs des fichiers ouverts. Par défaut au début seuls trois sont présents : 0 1 et 2 (les canaux standards). A chaque ouverture de fichier ou de nouveau canal, la table se remplit. A la fermeture du processus, les descripteurs sont fermés (en principe).
- On trouve d'autres informations comme la taille de la mémoire allouée, la date de lancement du processus, le terminal d'attachement, l'état du processus, UID effectif et Reel ainsi que GID effectif et réel.

10.2 Etats d'un processus

Durant sa vie (temps entre le lancement et la sortie) un processus peut passer par divers états ou **process state** :

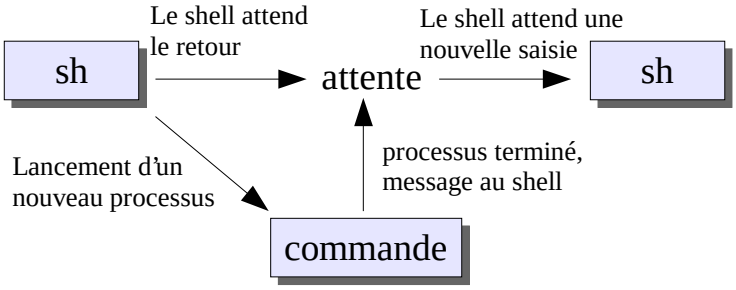
- Exécution en mode utilisateur (**user mode**)
- Exécution en mode noyau (**kernel mode**)
- En attente E/S (**waiting**)
- Endormi (**sleeping**)
- Prêt à l'exécution (**runnable**)

- Endormi dans le swap (**mémoire virtuelle**)
- Nouveau processus
- Fin de processus (**Zombie**)



10.3 Lancement en tâche de fond

En suivant ce que nous avons vu avant, le système étant multi-tâches un certain nombre de processus tournent déjà sur la machine sans que nous le voyons. De même le shell que nous utilisons est lui-même un processus. Quand une commande est saisie, le shell créé un nouveau processus pour l'exécuter, ce processus devient un processus enfant du shell. Jusqu'à présent il fallait attendre la fin de l'exécution d'une commande pour en saisir une autre (sauf à utiliser le « ; » pour chaîner les commandes).



Rien n'empêche le shell d'attendre le message du processus terminé pour rendre la main : de ce fait la commande une fois lancé, le shell peut autoriser la saisie d'une nouvelle commande sans attendre la fin de l'exécution de la commande précédente. Pour cela il suffit de saisir après avoir tapé la commande le **ET Commercial « & »**. Dans ce cas le shell et la commande lancée fonctionneront en parallèle.

```
$ ls -R / > ls.txt 2/dev/null &
[1] 21976
$
[1]    Done                  ls -l -R / > ls.txt 2/dev/null
$ ls
fic1         fic3         liste        ls.txt       rep1         users
fic2         fic4         liste2       mypass       toto.tar.gz
```

Juste après la saisie un chiffre apparaît, il est à retenir car il s'agit du PID du nouveau processus lancé. Après une autre saisie une ligne Done indique que le traitement est terminé. La valeur [1] est propre à un shell particulier (ksh).

Quelques remarques sur l'utilisation du lancement en tâche de fond :

(1)Le processus lancé ne devrait pas attendre de saisie au risque de confusion entre cette commande et le shell lui-même.

(2)Le processus lancé ne devrait pas afficher de résultats sur l'écran au risque d'avoir des affichages en conflit avec celui du shell (par exemple apparition d'une ligne en milieu de saisie).

(3)Enfin, quand on quitte le shell, on quitte aussi tous ses fils : dans ce cas ne pas quitter le shell pendant un traitement important.

10.3.1 wait

Lorsqu'un processus est en tâche de fond, il est possible de récupérer et d'attendre la fin de la dernière commande lancée en arrière-plan avec la commande **wait [pid]**. S'il s'agit d'une autre commande, alors il est important d'avoir noté le PID de celle-ci lorsqu'elle a été lancée.

10.4 Liste des processus

La commande **ps** (process status) permet d'avoir des informations sur les processus en cours. Lancée seule, elle n'affiche que les processus en cours lancés depuis l'utilisateur et la console actuels. Nous détaillerons les colonnes plus loin.

```
$ ps
   PID TTY      S           TIME CMD
 22608 ttyp0    S        0:00.04 -csh (csh)
```

Pour avoir plus d'informations, on peut utiliser l'option **-f**.

```
$ ps -f
UID          PID   PPID    C STIME    TTY             TIME CMD
oracle     22608  12288  0.0 15:20:09 ttyp0        0:00.05 -csh (csh)
```

L'option -e donnes des informations sur tous les processus en cours.

```
$ ps -ef
UID         PID   PPID    C STIME    TTY            TIME CMD
root          0      0  0.0   Aug 11 ??          2:26.95 [kernel idle]
root          1      0  0.0   Aug 11 ??          0:01.87 /sbin/init -a
root          3      1  0.0   Aug 11 ??          0:00.27 /sbin/kloadsrv
root         51      1  0.0   Aug 11 ??          1:30.22 /sbin/update
root        165      1  0.0   Aug 11 ??          0:00.35 /usr/sbin/syslogd
root        169      1  0.0   Aug 11 ??          0:00.02 /usr/sbin/binlogd
root        229      1  0.0   Aug 11 ??          0:00.00 /usr/sbin/routed
root        260      1  0.0   Aug 11 ??          0:01.65 /usr/sbin/rwhod
root        349      1  0.0   Aug 11 ??          0:00.02 /usr/sbin/portmap
root        351      1  0.0   Aug 11 ??          0:00.00 /usr/sbin/nfsiod 7
root        354      1  0.0   Aug 11 ??          0:00.01 /usr/sbin/rpc.statd
root        356      1  0.0   Aug 11 ??          0:00.01 /usr/sbin/rpc.lockd
root        587      1  0.0   Aug 11 ??          0:00.23 /usr/sbin/xntpd -gx -l
root        626      1  0.0   Aug 11 ??          0:00.06 /usr/sbin/snmpd
root        646      1  0.0   Aug 11 ??          0:00.84 /usr/sbin/inetd
root        654      1  0.0   Aug 11 ??          0:00.01 /usr/sbin/svrMgt_mib
root        655      1  0.0   Aug 11 ??          0:00.03 /usr/sbin/svrSystem_mi
root        657      1  0.0   Aug 11 ??          0:00.03 /usr/sbin/os_mibs
root        659      1  0.0   Aug 11 ??          0:00.08 /usr/sbin/cpq_mibs
root        704      1  0.0   Aug 11 ??          0:01.46 /usr/sbin/cron
root        718      1  0.0   Aug 11 ??          0:00.01 /usr/lbin/lpd
...
```

L'option **-u** permet de préciser une liste d'un ou plusieurs utilisateurs séparés par une virgule. L'option **-g** effectue la même chose mais pour les groupes, **-t** pour les terminaux et **-p** pour des PID précis.

```
$ ps -u oracle
   PID TTY      S           TIME CMD
  2308 ??       S        0:02.27 /mor/app/oracle/product/8.1.7/bin/tnslsnr
LISTENER -inherit
  2311 ??       S        0:00.98 /mor/app/oracle/product/8.1.7/bin/tnslsnr
DIVERS -inherit
  2334 ??       S        0:00.22 ora_pmon_ORAP
  2336 ??       S        0:01.79 ora_dbw0_ORAP
  2338 ??       S        0:02.94 ora_lgwr_ORAP
  2340 ??       S        0:20.23 ora_ckpt_ORAP
  2342 ??       S        0:04.42 ora_smon_ORAP
  2344 ??       S        0:00.09 ora_reco_ORAP
  2346 ??       S        0:03.11 ora_snp0_ORAP
  2348 ??       S        0:02.73 ora_snp1_ORAP
  2350 ??       S        0:02.52 ora_snp2_ORAP
  2352 ??       S        0:02.69 ora_snp3_ORAP
  2354 ??       S        0:00.08 ora_arc0_ORAP
  2369 ??       S        0:00.21 ora_pmon_OLAP
  2371 ??       S        0:00.22 ora_dbw0_OLAP
...
```

Enfin l'option -l propose plus d'informations techniques.

```
$ ps -l
      F S          UID    PID   PPID    C PRI  NI   ADDR   SZ WCHAN    TTY            TIME CMD
80808001 S           75  22608  12288  0.0  44   0      0 424K pause    ttyp0        0:00.06 csh
```

Colonne	*Définition*
UID	User ID, nom de l'utilisateur
PID	Process ID, numéro du processus
PPID	Parent Process ID, numéro du processus père
C	Facteur de priorité, plus la valeur est grande plus la priorité est élevée
STIME	Heure de lancement du processus
TTY	Nom du terminal depuis lequel le processu a été lancé.
TIME	Durée de traitement du processus
CMD	Commande exécutée
F	Drapeaux du processus (sort du cadre du cours)
S	Etat du processus S (sleeping) R (running) Z (zombie)
PRI	Priorité du processus
NI	Nice, incrément pour le scheduler

10.5 Arrêt d'un processus / signaux

Lorsqu'un processus tourne en tâche de fond il ne peux pas être arrêté par une quelconque combinaison de touches. Pour cela il faut employer la commande **kill**. Contrairement à ce que son nom semble indiquer, le rôle de cette commande n'est pas forcément de détruire ou de terminer un processus (récalcitrant ou non), mais d'envoyer des signaux aux processus.

kill [-l] -Num_signal PID [PID2...]

Le **signal** est l'un des moyens de communication entre les processus. Lorsqu'on envoie un signal à un processus, celui-doit doit l'intercepter et réagir en fonction de celui-ci. Certains signaux peuvent être ignorés, d'autres non. Suivant les Unix on dispose d'un nombre plus ou moins important de signaux. Les signaux sont numérotés et nommés, mais attention si les noms sont généralement communs d'un Unix à l'autre, les numéros ne le sont pas forcément. L'option « -l » permet d'obtenir la liste des signaux.

Voici ceux qui nous concernent.

Signal	***Rôle***
1 (SIGHUP)	Hang Up, est envoyé par le père à tous ses enfants lorsqu'il se termine
2 (SIGINT)	Interruption du processus demandé (touche suppr, Ctrl+C)
3 (SIGQUIT)	Idem SIGINT mais génération d'un Core Dump (fichier de débuggage)
9 (SIGKILL)	Signal ne pouvant être ignoré, force le processus à finir 'brutalement'.
15 (SIGTERM)	Signal envoyé par défaut par la commande kill. Demande au processus de se terminer normalement.

```
$ ps
   PID TTY      S           TIME CMD
 22608 ttyp0    S        0:00.04 -csh (csh)
 22610 ttyp0    R        0:00.05 ls
$ kill 22610
22610 Terminated
...
$ ps
   PID TTY      S           TIME CMD
 22608 ttyp0    S        0:00.04 -csh (csh)
 22615 ttyp0    R        0:00.05 ls
$ kill -9 22615
22610 (killed)
```

10.6 nohup

Quand le shell est quitté (exit, ctrl+D, ...) nous avons vu que le signal 1 SIGHUP est envoyé aux enfants pour qu'ils se terminent aussi. Lorsqu'un traitement long est lancé en tâche de fond et que l'utilisateur veut quitter le shell, ce traitement sera alors arrêté et il faudra tout recommencer. Le moyen d'éviter cela est de lancer le traitement (processus) avec la commande **nohup**. Dans ce cas le processus lancé ne réagira plus au signal SIGHUP, et donc le shell peut être quitté, la commande continuera sont exécution.

Par défaut avec nohup (sous sh bash et ksh) les canaux de sortie et d'erreur standards sont redirigés vers un fichier **nohup.out**, sauf si la redirection est explicitement précisée.

```
$ nohup ls -lR / &
10206
$ Sending output to nohup.out
$ ls
fic1         fic3         liste        ls.txt       nohup.out    toto.tar.gz
fic2         fic4         liste2       mypass       rep1         users
```

10.7 nice et renice

La commande **nice** permet de lancer une commande avec une priorité pus faible, afin de permettre éventuellement à d'autres processus de tourner plus rapidement.

```
nice [-valeur] commande [arguments]
nice [-n valeur] commande [arguments]
```

Les deux syntaxes sont généralement admises, à vérifier par man. La valeur représenta la priorité du traitement. Pour la seconde syntaxe une valeur positive causera une baisse de priorité, une négative l'augmentation de la priorité (si autorisé). La première syntaxe n'accepte qu'un abaissement de la priorité. La valeur doit être comprise entre 1 et 20. Plus la valeur est élevée et plus le traitement est ralenti.

```
$ nice -10 ls -lR / >liste 2>/dev/null&
10884
$ ps -l
      F S          UID    PID   PPID    C PRI  NI   ADDR   SZ WCHAN    TTY            TIME CMD
80808001 S           75  10153  10115  0.0  44   0      0 424K pause    ttyp4        0:00.09 csh
80808001 S  +        75  10822  10153  0.0  44   0      0 192K wait     ttyp4        0:00.01 sh
80808001 U N+        75  10884  10822 28.5  54  10      0 848K aa3b3a9c ttyp4        0:03.32 ls
```

La commande **renice** fonctionne un peu comme nice sauf qu'elle permet de modifier la priorité en fonction d'un utilisateur, d'un groupe ou d'un PID. La commande doit donc déjà tourner.

```
renice [-n prio] [-p] [-g] [-u] ID
```

La priorité doit être comprise entre -20 et 20. L'utilisateur standard ne peut utiliser que les valeurs entre 0 et 20 permettant de baisser la priorité. L'option -p précise un PID, -g un GID et -u un UID.

10.8 time

La commande **time** mesure les durées d'exécution d'une commande, idéal pour connaître les temps de traitement, et retourne trois valeurs :

(1)**real** : durée totale d'exécution de la commande

(2)**User** : durée du temps CPU nécessaire pour exécuter le programme

(3)**System** : durée du temps CPU nécessaire pour exécuter les commandes liées à l'OS (appels système au sein d'un programme).

Le résultat est sorti par le canal d'erreur standard 2. On peut avoir une indication de la charge de la machine par le calcul Real / (User+System). Si le résultat est inférieur à 10, la machine dispose de bonnes performances, au-delà de 20 la charge de la machine est trop lourdes (performances basses).

```
$ time ls -lR /home
...
real   4.8
user   0.2
sys    0.5
```

10.9 Droits d'accès étendus

10.9.1 SUID et SGID

Il est possible d'établir des **droits d'accès étendus** à l'exécution d'une commande. Ces droits d'accès étendus appliqués à une commande permettent à cette commande de s'exécuter avec les droits du propriétaire ou du groupe d'appartenance de la commande, et non plus avec les droits de l'utilisateur l'ayant lancé.

L'exemple le plus simple est le programme **passwd** permettant de changer son mot de passe. Si la commande était exécutée avec les droits d'un utilisateur classique, **passwd** ne pourrait pas ouvrir et modifier les fichiers /etc/passwd et /etc/shadow :

```
$ ls -l /etc/passwd
-rw-r--r--   1 root     system      4010 Jun  6 12:26 /etc/passwd
```

Nous voyons que ce fichier appartient à root, et que seul root peut y écrire. Un utilisateur simple ne peut lire que son contenu sans interagir. La commande passwd ne devrait donc pas pourvoir modifier les fichiers. Voyons la commande passwd (/bin/passwd ou /usr/bin/passwd) :

```
$ ls -l /usr/bin/passwd
-rws--x--x   3 root     bin        16384 Apr 12  1999 /usr/bin/passwd
```

Un nouveau droit est apparu : le droit « **s** » pour les droits de le l'utilisateur root. Ce nouvel attribut permet l'exécution de la commande avec des droits d'accès étendus. Le temps du traitement, le programme est exécuté avec les droits du propriétaire du fichier ou de son groupe d'appartenance. Dans le cas de passwd, il est lancé avec les droits de root le temps de son traitement.

Le droit « s » sur l'utilisateur est appelé le **SUID-Bit** (Set User ID Bit), et sur le groupe le **GUID-Bit** (Set Group ID Bit)

La commande **chmod** permet de placer les SUID-Bit et GUID-Bit.

```
chmod u+s commande
chmod g+s commande
```

Les valeurs octales sont 4000 pour le SUID-Bit et 2000 pour le GUID-Bit.

```
chmod 4755 commande
chmod 2755 commande
```

Seul le propriétaire ou l'administrateur peut positionner ce droit. Positionner le SUID-bit ou le SGID-Bit n'a de sens que si les droits d'exécution ont préalablement été établis (attribut x sur le propriétaire ou le groupe). Si ceux-ci ne sont pas présents; le « **s** » est remplacé par un « **S** ».

10.9.2 Real / effectif

Dans les données d'identification du processus nous avons vu la présence **UID et de GID réels et effectifs**. Quand une commande est lancée avec un SUID-Bit ou un SGID-Bit positionné, les droits se trouvent modifiés. Le système conserve les UID et GID d'origine de l'utilisateur ayant lancé la commande (UID et GID réels) transmis par le père, les numéros UID et GID effectifs étant ceux du propriétaire ou du groupe d'appartenance du programme.

Ex : toto (UID=100, GID=100) lance passwd, appartenant à root (UID=1, GID=2) avec le SUID-Bit positionné.

```
UID réel : 100
GID réel : 100
UID effectif : 1
GID effectif : 100
```

Si le SGID-Bit était positionné seul :

```
UID réel : 100
GID réel : 100
UID effectif : 100
GID effectif : 2
```

Il est à noter que les SUID-Bit et SGID-bit ne sont pas transmis aux enfants d'un processus. Dans ce cas les enfants seront exécutés avec les droits de l'utilisateur ayant lancé la commande de base.

10.9.3 Sticky bit

Le **Sticky-bit** (bit collant) permet d'affecter une protection contre l'effacement du contenu d'un répertoire.

Imaginons un répertoire /tmp où tous les utilisateurs ont le droit de lire et d'écrire des fichiers.

```
$ ls -ld /tmp
drwxrwxrwx   6 root     system     16384 Aug 14 13:22 tmp
```

Dans ce répertoire tout le monde peut supprimer des fichiers y compris ceux qui ne lui appartiennent pas (droit w présent partout et pour tous). Si l'utilisateur toto crée un fichier, l'utilisateur titi peut le supprimer même s'il ne lui appartient pas.

Le Sticky-Bit appliqué à un répertoire, ici /tmp, empêche cette manipulation. Si le fichier peut encore être visualisé et modifié, seul son propriétaire (et l'administrateur) pourra le supprimer.

```
$ chmod u+t /tmp
$ ls -ld /tmp
drwxrwxrwt   6 root     system     16384 Aug 14 13:22 tmp
```

En octal, on utilisera la valeur 1000 (chmod 1777 /tmp)

Bien qu'appliqué à l'utilisateur, le Sticky-bit, représenté par un « **t** » apparaît au niveau de « others ».

11 Recherche complexe de fichiers : find

La commande **find** permet de rechercher des fichiers au sein de l'arborescence du système de fichiers, à l'aide de critères et la possibilité d'agir sur les résultats retournés.

```
find repertoires critères options
```

La commande find étant récursive, il suffit d'indiquer un répertoire de base pour que toute l'arborescence depuis ce répertoire soit développée. L'option de base est **-print** (souvent implicite sur la plupart des Unix) qui permet d'afficher sur écran les résultats.

```
$ find . -print
.
./fic1
./fic2
./fic3
./fic4
./rep1
./liste
./mypass
./users
./liste2
./ls.txt
./toto.tar.gz
./nohup.out
./liste_ls
```

Le chemin précisé étant relatif, l'affichage est relatif. Si le chemin précisé était l'absolu, l'affichage aurait été absolu.

11.1 Critères

Les options permettent de définir les critères de recherche. Elles peuvent être combinées et groupées entre elles, sous forme et ET et de OU logiques.

11.1.1 -name, -iname

L'option **-name** permet une sélection par noms de fichiers. Il est possible d'utiliser les critères de recherches (caractères spéciaux) déjà vus. Le critère est idéalement placé en guillemets. Avec **-iname**, on n'effectue pas de différences entre les minuscules et les majuscules.

```
$ find . name "fic*" -print
./fic1
./fic2
./fic3
./fic4
```

11.1.2 -type

L'option **-type** permet une sélection par type de fichier. Nous avons vu au début du cours qu'outre les exécutables, liens, répertoires et fichiers simples étaient présent d'autres types de fichiers.

Code fichier	*Type de fichier*
b	Fichier spécial en mode bloc
c	Fichier spécial en mode caractère
d	Répertoire (directory)
f	fichier normal
l	lien symbolique
p	tube nommé (pipe)
s	Socket (sur certains Unix)

```
$ find . -name "re*" -type d -print
./rep1
./rep2
```

11.1.3 -user et -group, -uid et -gid

Les options **-user** et **-group** permettent une recherche sur le propriétaire et le groupe d'appartenance des fichiers. Il est possible de préciser le nom (utilisateur, groupe) ou l'ID (UID, GID). On peut aussi préciser **-uid** et **-gid** avec les numéros UID et GID.

```
$  find . -type f -user oracle -group dba -print
./fic1
./fic3
```

11.1.4 -size

L'option **-size** permet de préciser la taille des fichiers recherchés.

- **-size 5** : recherche les fichiers d'une taille de 5 blocs (512 octets par bloc, soit ici 2560 octets)
- **-size 152c** : recherche les fichiers d'une taille de 152 caractères (octets)
- **-size 10k** : recherche les fichiers d'une taille de 10ko (10*1024 octets = 10240 octets)
- **-size +/- valeur[c/k]** : si – alors recherche des fichiers ayant une taille inférieure à celle indiquée, si + alors recherche des fichiers ayant une taille supérieure à celle indiquée.

```
$ find /tmp -type f -size +100k -print
/tmp/quota.user
/tmp/seb/ls.txt
/tmp/seb/liste_ls
/tmp/messages
/tmp/oracledml_715.log
/tmp/oracledml_716.log
/tmp/dml2_ae_ajout_RM_FS_AS_BM_MSE_Juillet2002.lst
/tmp/oracledml_714.log
/tmp/dml4_ae_compta_RM_FS_AS_BM_MSE_Juillet2002.lst
/tmp/Corrections_avant_valo_juillet_2002.lst
/tmp/DML_40.lst
```

11.1.5 -empty

L'option **-empty** recherche des fichiers ou répertoires vides, c'est à dire de taille nulle.

11.1.6 -atime, -mtime et -ctime

- **-atime** : recherche sur la date du dernier accès (access time)
- **-mtime** : recherche sur la date de dernière modification (modification time)
- **-ctime** : recherche sur la dare de création (creation time, en fait la date de dernière modification du numéro d'inode)

Ces trois options ne travaillent qu'avec des jours (périodes de 24 heures). 0 est le jour même, 1 24 heures, 2 48 heures, ...

Les signes + ou – permettent de préciser les termes « de plus » et « de moins » :

- -mtime 1 : fichiers modifiés hier (entre 24 et 48 heures)
- -mtime -3 : fichiers modifiés il y a moins de trois jours (72 heures)
- -atimes +4 : fichiers modifiés il y a plus de 4 jours (plus de 96 heures)

```
$  find /tmp -type f -size +100k -ctime +5 -print
/tmp/quota.user
/tmp/oracledml_714.log
/tmp/Corrections_avant_valo_juillet_2002.lst
```

11.1.7 -newer

L'option **-newer** permet de rechercher des fichiers plus récemment modifiés que celui indiqué. Par exemple rechercher un fichier plus récent que toto.txt :

```
$ find . -newer toto.txt -print
```

11.1.8 -perm

L'option **-perm** permet d'effectuer des recherches sur les autorisations d'accès (droits, SUID, SGID, Sticky). Les droits doivent être précisés en base 8 (valeur octale) et complets. Le caractère « - » placé devant la valeur octale signifie que les fichiers recherchés doivent au moins avoir les droits désirés. Ex pour rw-rw-rw- (666).

```
$ find /tmp -perm 666 -print
-rw-rw-rw-   1 root     system      9650 Aug 14 11:15 /tmp/LOG/load_logi.log
```

11.1.9 -links et -inum

L'option **-links** permet une recherche par nombre de hard links. On peut aussi préciser les signes + ou – (plus de n liens et moins de n liens). Un fichier normal seul possède 1 lien. Un répertoire 2 liens (L'entrée dans le catalogue dont il fait partie et dans .). Pour une recherche de liens symboliques il faudra utiliser l'option -type l.

```
$ find . -type f -links +2 -print
./fic2
./hardlink3_fic2
./hardlink_fic2
./hardlink2_fic2
```

L'option **-inum** permet une recherche par numéro d'inode. Elle est utile dans le cas d'une recherche de tous les liens portant un même numéro d'inode. Le numéro d'inode est visible par l'option -i de la commande ls.

```
$ ls -i
   95 fic1                643 hardlink_fic2         36 liste_ls             205 seb1
  643 fic2                219 lien2_symbol_fic1    705 ls.txt              284 seb2
  707 fic3                 95 lien_fic1            237 mypass              110 toto.tar.gz
  546 fic4                 14 lien_symbol_fic1     886 nohup.out           212 users
  643 hardlink2_fic2      266 liste                916 rep1
  643 hardlink3_fic2     1006 liste2               881 rep2
$ find . -inum 95 -print
./fic1
./lien_fic1
```

11.2 commandes

Outre l'option **-print** on trouve deux autres options permettant d'effectuer une action sur les fichiers trouvés. Pour l'option **-printf** permettant un format précis de la sortie, on se reportera au man pages.

11.2.1 -ls

L'option **-ls** affiche des informations détaillées sur les fichiers trouvés correspondant au critère au lieu du simple nom de fichier.

```
$ find . -inum 95 -ls -print
     95    0 -rwxr-xr-x  2 oracle   dba            0 Aug 12 11:05 ./fic1
./fic1
     95    0 -rwxr-xr-x  2 oracle   dba            0 Aug 12 11:05 ./lien_fic1
./lien_fic1
```

11.2.2 -exec

L'option **-exec** va exécuter la commande située juste après pour chaque occurrence trouvée. Quelques remarques :

(1)-exec doit obligatoirement être la dernière option de la commande find.

(2)La commande exécutée par -exec doit se terminer par un « ; ». Ce caractère spécial doit s'écrire \; pour ne pas être interprété par le shell.

(3)Pour passer comme paramètre à la commande le fichier trouvé par find, il faut écrire {} (substitution du fichier).

Exemple pour effacer tous les liens symboliques du répertoire courant (avec le détail des fichiers supprimés) :

```
$ find . -type l -ls -exec rm -f {} \;
     14    0 lrwxrwxrwx  1 oracle   system         4 Aug 14 14:48 ./lien_symbol_fic1 -> fic1
    219    0 lrwxrwxrwx  1 oracle   system         4 Aug 14 14:48 ./lien2_symbol_fic1 ->
fic1
```

11.2.3 -ok

L'option **-ok** est identique à l'option -exec sauf que pour chaque occurrence, une confirmation est demandée à l'utilisateur.

```
$ find . -inum 95 -ok rm -f {} \;
< rm ... ./fic1 > (yes)?   n
< rm ... ./lien_fic1 > (yes)?   y
```

11.3 critères AND / OR / NOT

Il est possible de combiner les options de critère de sélection. Sans aucune précision c'est le ET logique qui est implicite.

Critère	*Action*
-a	AND, ET logique
-o	OR, OU logique
!	NOT, NON logique
(...)	groupement des combinaisons. Les parenthèses doivent être vérouillées \(...\).

Exemple avec tous les fichiers ne contenant pas fic dans leur nom, et tous les fichiers n'étant ni normaux ni des répertoires.

```
$ find . ! -name "*fic*" -print
.
./rep1
./liste
./mypass
./users
./liste2
./ls.txt
./toto.tar.gz
./nohup.out
./liste_ls
./rep2
./seb1
./seb2
$ find . ! \( -type f -o type d \) -ls
    409    0 lrwxrwxrwx  1 oracle   system         4 Aug 14 15:21 ./lien_fic1 -> fic1
    634    0 lrwxrwxrwx  1 oracle   system         4 Aug 14 15:21 ./lien_fic2 -> fic2
```

12 Plus loin avec le Bourne Shell

Toutes les commandes que nous nous avons vu jusqu'à présent sont standards et ne sont pas propres à un shell particulier. Avant d'entamer la programmation shell elle-même, il nous faut voir quelques options détaillées du shell par défaut, le Bourne Shell.

Fichier de configuration

Le Bourne Shell dispose d'un fichier de configuration chargé et exécuté lors du lancement du shell. Il est généralement placé dans le répertoire par défaut de l'utilisateur et se nomme **.profile**.

```
PATH=/home/oracle/bin:$ORACLE_HOME/bin:/usr/sbin:/usr/local/bin:/usr/bin:/sqlb
t/datatools/obacktrack/bin:/usr/opt/networker/bin:.
#export NLS_SORT=BINARY
#. /usr/local/bin/oraenv
umask 022
```

12.1 Commandes internes et externes

Une commande peut être **interne** ou **externe**. Une commande interne est une commande directement incluse et interprétée par le shell, sans passer par un quelconque exécutable. Un exemple est la commande cd.

Une commande externe est un exécutable que le shell recherchera dans une liste de chemins prédéfinis, le PATH, puis exécutera en tant que processus fils.

D'autres mécanismes existent, comme les fonctions et les alias (hors Bourne Shell), que nous aborderons plus tard.

12.2 Herescript

La redirection « << » n'a pas été abordée car elle est particulière. Elle permet l'utilisation des Herescripts (Script ici). Un herescript permet la saisie d'un texte jusqu'à un point donné et l'envoi de son résultat à une commande ou un filtre. Les redirections classiques sont aussi autorisées. Après le « << » on indique une chaîne définissant la fin de saisie, par exemple ici 'end'.

```
$ tr "[a-z]" "[A-Z]" << end
> bonjour les amis
> ceci est un exemple
> de herescript
> end
BONJOUR LES AMIS
CECI EST UN EXEMPLE
DE HERESCRIPT
```

12.3 Ouverture de canaux

Les canaux standards sont au nombre de trois et numérotés de 0 à 2. Ainsi 0< équivaut à < et 1> à >. La commande **exec** permet d'ouvrir sept autres canaux numérotés de 3 à 9. On a donc en tout dix canaux.

On peut envisager, dans le cadre de traitements, de sortir certains résultats par le canal 3, d'autres par le 4, et ainsi de suite. Les canaux ouvert le sont en entrée et en sortie.

```
exec num_canal > fichier_ou_reunion
```

```
$ exec 3>dump.log
$ ls -l >&3
```

```
$ cat dump.log
total 5716
-rw-r--r--   1 oracle   system         0 Aug 14 15:55 dump.log
-rwxr-xr-x   1 oracle   dba            0 Aug 12 11:05 fic1
-rw-r--r--   4 oracle   system         0 Aug 12 11:05 fic2
-rw-r-----   1 oracle   dba            0 Aug 12 11:05 fic3
-rw-r--r--   1 oracle   system        33 Aug 12 12:03 fic4
-rw-r--r--   4 oracle   system         0 Aug 12 11:05 hardlink2_fic2
-rw-r--r--   4 oracle   system         0 Aug 12 11:05 hardlink3_fic2
-rw-r--r--   4 oracle   system         0 Aug 12 11:05 hardlink_fic2
lrwxrwxrwx   1 oracle   system         4 Aug 14 15:21 lien_fic1 -> fic1
lrwxrwxrwx   1 oracle   system         4 Aug 14 15:21 lien_fic2 -> fic2
-rw-r--r--   1 oracle   system       252 Aug 12 14:30 liste
-rw-r--r--   1 oracle   system       234 Aug 13 10:06 liste2
-rw-r--r--   1 oracle   system   4829707 Aug 14 11:23 liste_ls
-rw-r--r--   1 oracle   system    974408 Aug 13 15:20 ls.txt
-rw-r--r--   1 oracle   system      4010 Aug 13 09:10 mypass
-rw-------   1 oracle   system       106 Aug 14 11:09 nohup.out
drwxr-xr-x   2 oracle   dba         8192 Aug 12 11:21 rep1
drwxr-xr-x   2 oracle   system      8192 Aug 14 14:16 rep2
drwxr-xr-x   2 oracle   system      8192 Aug 14 14:17 seb1
drwxr-xr-x   2 oracle   system      8192 Aug 14 14:17 seb2
-rw-r--r--   1 oracle   system      2564 Aug 13 13:45 toto.tar.gz
-rw-r--r--   1 oracle   system       402 Aug 13 09:33 users
```

Tous ce qui sera écrit dans le canal 3 sera écrit dans le fichier dump.log. On peut ensuite fermer le canal en le réunissant avec un pseudo-canal (canal de fermeture -).

```
$ exec 3>&-
```

12.4 Groupement de commandes

Nous avons vu que le chaînage de commande est possible avec le « ; ». Il est aussi possible de grouper les commandes. Quand on exécute les commandes suivantes :

$ uname -a ; pwd ; ls -l >resultat.txt &

seule la dernière commande est exécutée en tâche de fond et seul son résultat est redirigé dans le fichier resultat.txt.

Une solution serait

```
$ uname -a >resultat.txt & ; pwd >>resultat.txt & ; ls -l >>resultat.txt &
[1] 18232
[2] 18238
[3] 18135
```

C'est une solution complexe et qui ne marchera pas toujours. De plus même si les commandes sont lancées séquentiellement, elles tournent toutes en parallèle et c'est la première finie qui écrira en premier dans le fichier. La solution consiste en l'utilisation des parenthèses « () ».

```
$ (uname -a ; pwd ; ls -l) > resultat.txt &
[1] 18239
$
[1]    Done                 ( uname -a; pwd; ls -l ) > resultat.txt
```

Dans ce cas, toutes les commandes placées entre les parenthèses sont lancées par un sous-shell, qui va ensuite exécuter les commandes précisées séquentiellement telles qu'indiquées. Ainsi la redirection concerne l'ensemble des commandes et rien n'empêche de lancer ce sous-shell en arrière-plan. On distingue bien d'ailleurs un seul PID 18239 lors de l'exécution des commandes.

Une seconde possibilité est l'utilisation des accolades {...}. Dans ce cas aucun sous-shell n'est

exécuté, et si une commande interne (cd ou autre) est exécutée, elle concerne le shell actif. L'accolade fermante doit être placée juste après un « ; ».

```
$  { uname -a; pwd; ls -l; } > resultat.txt
```

12.5 Liaison et exécution conditionnelle

En plus du chaînage classique, les commandées peuvent être liées et exécutées de façon conditionnelle. La condition d'exécution d'une commande est la réussite ou non de la précédente. Chaque commande une fois exécutée sort un code retour, généralement 0 si tout s'est bien passé, 1 ou 2 en cas d'erreur. Le shell peut récupérer cette valeur par la variable « **$?** ». (Variables abordées plus loin).

```
$ ls
...
$ echo $?
0
```

Les caractères « **&&** » et « || » permettent d'effectuer une exécution conditionnelle.

```
commande1 && commande2
commande1 || commande2
```

La commande située après « **&&** » sera exécutée uniquement si la commande précédente a retourné 0 (réussite). La commande située après « || » ne sera exécutée que si la commande précédente a retourné autre chose que 0.

```
$ cat liste
Produit objet   prix    quantites
souris  optique 30      15
dur     30giga  100     30
dur     70giga  150     30
disque  zip     12      30
disque  souple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
clavier 105     45      30
clavier 115     55      30
carte   son     45      30
carte   video   145     30
$ (grep "souris" liste && echo "Souris trouvee") || echo "Souris introuvable"
souris  optique 30      15
Souris trouvee
$ (grep "memoire" liste && echo "Memoire trouvee") || echo "Memoire
introuvable"
Memoire introuvable
```

13 Programmation shell

Le shell n'est pas qu'un simple interpréteur de commandes, mais dispose d'un véritable langage de programmation avec notamment une gestion des variables, des tests et des boucles, des opérations sur variables, des fonctions...

13.1 Structure et exécution d'un script

Toutes les instructions et commandes sont regroupées au sein d'un script. Lors de sont exécution, chaque ligne sera lue une à une et exécutée. Une ligne peut se composer de commandes internes ou externes, de commentaires ou être vide. Plusieurs instructions par lignes sont possibles, séparées par le « **;** » ou liées conditionnellement par « && » ou « || ». Le « ; » est l'équivalent d'un **saut de ligne**.

Par convention les shell scripts se terminent généralement (pas obligatoirement) par « .sh » pour le Bourne Shell et le Bourne Again Shell, par « .ksh » pour le Korn Shell et par « .csh » pour le C Shell.

Pour rendre un script exécutable directement :

```
$ chmod u+x monscript
```

Pour l'exécuter :

```
$ ./monscript
```

Pour éviter le ./ :

```
$ PATH=$PATH:.
$ monscript
```

Quand un script est lancé, un nouveau shell « fils » est créé qui va exécuter chacune des commandes. Si c'est une commande interne, elle est directement exécutée par le nouveau shell. Si c'est une commande externe, dans le cas d'un binaire un nouveau fils sera créé pour l'exécuter, dans le cas d'un shell script un nouveau shell fils est lancé pour lire ce nouveau shell ligne à ligne.

Une ligne de commentaire commence toujours par le caractère « # ». Un commentaire peut être placé en fin d'une ligne comportant déjà des commandes.

```
# La ligne suivante effectue un ls
ls # La ligne en question
```

La première ligne a une importance particulière car elle permet de préciser quel shell va exécuter le script

```
#!/usr/bin/sh
#!/usr/bin/ksh
```

Dans le premier cas c'est un script Bourne, dans l'autre un script Korn.

13.2 Les variables

On en distingue trois types : utilisateur, système et spéciales. Le principe est de pourvoir affecter un contenu à un nom de variable, généralement un chaîne de caractère, ou des valeurs numériques.

13.2.1 Nomenclature

Un nom de variable obéit à certaines règles :

(1)Il peut être composé de lettres minuscules, majuscules, de chiffres, de caractères de soulignement

(2)Le premier caractère ne peut pas être un chiffre

(3)Le taille d'un nom est en principe illimité (il ne faut pas abuser non plus)

(4)Les conventions veulent que les variables utilisateur soient en minuscules pour les différencier des variables système. Au choix de l'utilisateur.

13.2.2 Déclaration et affectation

Une variable est déclarée dès qu'une valeur lui est affectée. L'affectation est effectuée avec le signe « = », sans espace avant ou après le signe.

```
var=Bonjour
```

On peut aussi créer des variables vides. Celle-ci existera mais sans contenu.

```
var=
```

13.2.3 Accès et affichage

On accède au contenu d'une variable en plaçant le signe « $ » devant le nom de la variable. Quand le shell rencontre le « $ », il tente d'interpréter le mot suivant comme étant une variable. Si elle existe, alors le $nom_variable est remplacé par son contenu, ou par un texte vide dans le cas contraire.

On parle aussi de référencement de variable.

```
$ chemin=/tmp/seb
$ pwd
/home/toto
$ ls $chemin
dump.log        fic4            lien_fic1       liste_ls        rep1
seb2
fic1            hardlink2_fic2  lien_fic2       ls.txt          rep2
toto.tar.gz
fic2            hardlink3_fic2  liste           mypass          resultat.txt
users
fic3            hardlink_fic2   liste2          nohup.out       seb1
$ cd $chemin
$ pwd
/tmp/seb
$ cd $chemin/rep1
$ pwd
/tmp/seb/rep1
```

Pour afficher la liste des variables on utilise la commande **set**. Elle affiche les variables utilisateur et les variables système, nom et contenu. La commande **echo** permet d'afficher le contenu de variables spécifiques.

```
$ a=Jules
$ b=Cesar
$ set
EDITMODE=emacs
HOME=/home/seb
LD_LIBRARY_PATH=/mor/app/oracle/product/8.1.7/lib
```

```
LOGNAME=seb
MAIL=/usr/spool/mail/seb
MAILCHECK=600
MANPATH=/usr/man:/sqlbt/datatools/obacktrack/man
PATH=/home/oracle/bin:/usr/bin:.:/mor/app/oracle/product/8.1.7/bin:/sqlbt/data
tools/obacktrack/bin:/usr/opt/networker/bin:/sqlbt/datatools/scripts
PS1=$
PS2=>
SHELL=/bin/csh
SHLVL=1
TERM=xterm
USER=seb
a=Jules
b=Cesar
$ echo $a $b a conquis la Gaule
Jules Cesar a conquis la Gaule
```

Une variable peut contenir des caractères spéciaux, le principal étant l'espace. Mais

```
$ c=Salut les copains
les: not found
$ echo $c
```

Ne marche pas. Pour cela il faut soit verrouiller les caractères spéciaux un par un, soit de les mettre entre guillemets ou apostrophes.

```
c=Salut\ les\ Copains # Solution lourde
c="Salut les copains" # Solution correcte
c='Salut les copains' # Solution correcte
```

La principale différence entre les guillemets et les apostrophes est l'interprétation des variables et des substitutions. " et ' se vérouillent mutuellement.

```
$ a=Jules
$ b=Cesar
$ c="$a $b a conquis la Gaule"
$ d='$a $b a conquis la Gaule'
$ echo $c
Jules Cesar a conquis la Gaule
$ echo $d
$a $b a conquis la Gaule
$ echo "Unix c'est top"
Unix c'est top
$ echo 'Unix "trop bien"'
Unix "trop bien"
```

13.2.4 Suppression et protection

On supprime une variable avec la commande **unset**. On peut protéger une variable en écriture et contre sa suppression avec la commande **readonly**. Une variable en lecture seule même vide est figée. Il n'existe aucun moyen de la replacer en écriture et de la supprimer, sauf à quitter le shell.

```
$ a=Jules
$ b=Cesar
$ echo $a $b
Jules Cesar
$ unset b
$ echo $a $b
Jules
$ readonly a
$ a=Neron
a: is read only
$ unset a
```

```
a: is read only
```

13.2.5 Exportation

Par défaut une variable n'est accessible que depuis le shell où elle a été définie.

```
$ a=Jules
$ echo 'echo "a=$a"' > voir_a.sh
$ chmod u+x voir_a.sh
$ ./voir_a.sh
a=
```

La commande **export** permet d'exporter une variable de manière à ce que son contenu soit visible par les scripts et autres sous-shells. Les variables exportées peuvent être modifiées dans le script, mais ces modifications ne s'appliquent qu'au script ou au sous-shell.

```
$ export a
$ ./voir_a.sh
a=Jules
$ echo 'a=Neron ; echo "a=$a"' >> voir_a.sh
$ ./voir_a.sh
a=Jules
a=Neron
$ echo $a
Jules
```

13.2.6 Accolades

Les accolades de base « {} » permettent d'identifier le nom d'une variable. Imaginons la variable fichier contenant le nom de fichier 'liste'. On veut copier liste1 en liste2.

```
$ fichier=liste
$ cp $fichier2 $fichier1
usage: cp [-fhip] [--] source_file destination_file
   or: cp [-fhip] [--] source_file ...  destination_directory
   or: cp [-fhip] [-R | -r] [--]
                  [source_file | source_directory] ... destination_directory
```

Ça ne fonctionne pas car ce n'est pas $fichier qui est interprété mais $fichier1 et $fichier2 qui n'existent pas.

```
$ cp ${fichier}2 ${fichier}1
```

Dans ce cas, cette ligne équivaut à

```
$ cp liste2 liste1
```

Les accolades indiquent que fichier est un nom de variable.

13.2.7 Accolades et remplacement conditionnel

Les accolades disposent d'une syntaxe particulière.

`{variable:Remplacement}`

Selon la valeur ou la présence de la variable, il est possible de remplacer sa valeur par une autre.

Remplacement	***Signification***
{x:-texte}	si la variable x est vide ou inexistante, le texte prendra sa place. Sinon c'est le contenu de la variable qui prévaudra.
{x:=texte}	si la variable x est vide ou inexistante, le texte prendra sa place et deviendra la valeur de la variable.
{x:+texte}	si la variable x est définie et non vide, le texte prendra sa place. Dans le cas contraire une chaîne vide prend sa place.
{x:?texte}	si la variable x est vide ou inexistante, le script est interrompu et le message texte s'affiche. Si texte est absent un message d'erreur standard est affiché.

```
$ echo $nom

$ echo ${nom:-Jean}
Jean
$ echo $nom

$ echo ${nom:=Jean}
Jean
$ echo $nom
Jean
$ echo ${nom:+"Valeur définie"}
Valeur définie
$ unset nom
$ echo ${nom:?Variable absente ou non définie}
nom: Variable absente ou non définie
$ nom=Jean
$ echo ${nom:?Variable absente ou non définie}
Jean
```

13.3 variables système

En plus des variables que l'utilisateur peux définir lui-même, le shell est lancé avec un certain nombre de variables prédéfinies utiles pour un certain nombre de commandes et accessibles par l'utilisateur. Le contenu de ces variables système peut être modifié mais il faut alors faire attention car certaines ont une incidence directe sur le comportement du système.

Variable	*Contenu*
HOME	Chemin d'accès du répertoire utilisateur. Répertoire par défaut en cas de cd.
PATH	Liste de répertoires, séparés par des « : » où le shell va rechercher les commandes externes et autres scripts et binaires. La recherche se fait dans l'ordre des répertoires saisis.
PS1	Prompt String 1, chaîne représentant le prompt standard affiché à l'écran par le shell en attente de saisie de commande.
PS2	Prompt String 2, chaîne représentant un prompt secondaire au cas où la saisie doit être complétée.
IFS	Internal Field Separator, liste des caractères séparant les mots dans une ligne de commande. Par défaut il s'agit de l'espace de la tabulation et du saut de ligne.
MAIL	chemin et fichier contenant les messages de l'utilisateur
MAILCHECK	intervalle en secondes pour la vérification de présence d'un nouveau courrier. Si 0 alors la vérification est effectuée après chaque commande.
SHELL	Chemin complet du shell en cours d'exécution
LANG	Définition de la langue à utiliser ainsi que du jeu de caractères
LC_COLLATE	Permet de définir l'ordre du tri à utiliser (si LANG est absent)
LC_NUMERIC	Format numérique à utiliser
LC_TIME	Format de date et d'heure à utiliser
LC_CTYPE	Détermination des caractères et signes devant ou ne devant pas être pris en compte dans un tri.
USER	Nom de l'utilisateur en cours.
LD_LIBRARY_PATH	Chemin d'accès aux bibliothèques statiques ou partagées du système.
MANPATH	Chemin d'accès aux pages du manuel.
LOGNAME	Nom du login utilisé lors de la connexion.

13.4 Variables spéciales

Il s'agit de variables accessibles uniquement en lecture et dont le contenu est généralement contextuel.

Variable	*Contenu*
$?	Code retour de la dernière commande exécutée
$$	PID du shell actif
$!	PID du dernier processus lancé en arrière-plan
$-	Les options du shell

```
$ echo $$
23496
$ grep memoire liste
$ echo $?
1
$ grep souris liste
souris  optique 30      15
$ echo $?
0
$ ls -lR >toto.txt 2<&1 &
26675
$ echo $!
26675
```

13.5 Paramètres de position

Les paramètres de position sont aussi des variables spéciales utilisées lors d'un passage de paramètres à un script.

13.5.1 Description

Variable	*Contenu*
$0	Nom de la commande (du script)
$1-9	$1,$2,$3... Les neuf premiers paramètres passés au script
$#	Nombre total de paramètres passés au script
$*	Liste de tous les paramètres au format "$1 $2 $3 ..."
$@	Liste des paramètres sous forme d'éléments distincts "$1" "$2" "$3" ...

```
$ cat param.sh
#!/usr/bin/sh

echo "Nom : $0"
echo "Nombre de parametres : $#"
echo "Parametres : 1=$1 2=$2 3=$3"
echo "Liste : $*"
echo "Elements : $@"

$ param.sh riri fifi loulou
Nom : ./param.sh
Nombre de parametres : 3
Parametres : 1=riri 2=fifi 3=loulou
```

```
Liste : riri fifi loulou
Elements : riri fifi loulou
```

La différence entre **$@** et **$*** ne saute pas aux yeux. Reprenons l'exemple précédent avec une petite modification :

```
$ param.sh riri "fifi loulou"
Nom : ./param.sh
Nombre de parametres : 2
Parametres : 1=riri 2=fifi loulou 3=
Liste : riri fifi loulou
Elements : riri fifi loulou
```

Cette fois-ci il n'y a que deux paramètres de passés. Pourtant les listes semblent visuellement identiques. En fait si la première contient bien

```
"riri fifi loulou"
```

La deuxième contient

```
"riri" "fifi loulou"
```

Soit bien deux éléments. Dans les premier exemple nous avions

```
"riri" "fifi" "loulou"
```

13.5.2 redéfinition des paramètres

Outre le fait de lister les variables, la commande **set** permet de redéfinir le contenu des variables de position. Avec

```
set valeur1 valeur2 valeur3 ...
```

$1 prendra comme contenu valeur1, $2 valeur2 et ainsi de suite.

```
$ cat param2.sh
#!/usr/bin/sh
echo "Avant :"
echo "Nombre de parametres : $#"
echo "Parametres : 1=$1 2=$2 3=$3 4=$4"
echo "Liste : $*"
set alfred oscar romeo zoulou
echo "apres set alfred oscar romeo zoulou"
echo "Nombre de parametres : $#"
echo "Parametres : 1=$1 2=$2 3=$3 4=$4"
echo "Liste : $*"

$ ./param2.sh riri fifi loulou donald picsou
Avant :
Nombre de parametres : 5
Parametres : 1=riri 2=fifi 3=loulou 4=donald
Liste : riri fifi loulou donald picsou
apres set alfred oscar romeo zoulou
Nombre de parametres : 4
Parametres : 1=alfred 2=oscar 3=romeo 4=zoulou
Liste : alfred oscar romeo zoulou
```

13.5.3 Réorganisation des paramètres

La commande **shift** est la dernière commande permettant de modifier la structure des paramètres de position. Un appel simple décale tous les paramètres d'une position en supprimant le premier : $2 devient $1, $3 devient $2 et ainsi de suite. Le $1 originel disparaît. $#, $* et $@ sont redéfinis en conséquence.

La commande shift suivie d'une valeur n effectue un décalage de n éléments. Ainsi avec

```
shift 4
```

$5 devient $1, $6 devient $2, ...

```
$ cat param3.sh
#!/usr/bin/sh
set alfred oscar romeo zoulou
echo "set alfred oscar romeo zoulou"
echo "Nombre de parametres : $#"
echo "Parametres : 1=$1 2=$2 3=$3 4=$4"
echo "Liste : $*"
shift
echo "Après un shift"
echo "Nombre de parametres : $#"
echo "Parametres : 1=$1 2=$2 3=$3 4=$4"
echo "Liste : $*"

$ ./param3.sh
set alfred oscar romeo zoulou
Nombre de parametres : 4
Parametres : 1=alfred 2=oscar 3=romeo 4=zoulou
Liste : alfred oscar romeo zoulou
Après un shift
Nombre de parametres : 3
Parametres : 1=oscar 2=romeo 3=zoulou 4=
Liste : oscar romeo zoulou
```

13.6 Sortie de script

La commande **exit** permet de mettre fin à un script. Par défaut la valeur retournée est 0 (pas d'erreur) mais n'importe quelle autre valeur de 0 à 255 peut être précisée. On récupère la valeur de sortie par la variable $?.

```
$ exit 1
```

13.7 Environnement du processus

En principe seules les variables exportées sont accessibles par un processus fils. Si on souhaite visualiser l'environnement lié à un fils (dans un script par exemple) on utilise la commande **env**.

```
$ env
PATH=/home/oracle/bin:/usr/bin:.:/mor/app/oracle/product/8.1.7/bin:/sqlbt/data
to
ols/obacktrack/bin:/usr/opt/networker/bin:/sqlbt/datatools/scripts
TERM=xterm
LOGNAME=oracle
USER=oracle
SHELL=/bin/csh
HOME=/home/oracle
SHLVL=1
MAIL=/usr/spool/mail/oracle
ORACLE_SID=ORAP
```

```
EPC_DISABLED=TRUE
ORACLE_BASE=/mor/app/oracle
ORACLE_HOME=/mor/app/oracle/product/8.1.7
EDITMODE=emacs
ORATAB=/etc/oratab
```

La commande **env** permet de redéfinir aussi l'environnement du processus à lancer. Cela peut être utile lorsque le script doit accéder à une variable qui n'est pas présente dans l'environnement du père, ou qu'on ne souhaite pas exporter. La syntaxe est

```
env var1=valeur var2=valeur ... commande
```

Si la première option est le signe « - » alors c'est tout l'environnement existant qui est supprimé pour être remplacé par les nouvelles variables et valeurs.

```
$ unset a
$ ./voir_a.sh
a=
$ env a=jojo ./voir_a.sh
a=jojo
$ echo a=$a
a=
```

13.8 Substitution de commande

Le mécanisme de substitution permet de placer le résultat de commandes simples ou complexes dans une variable. On place les commandes à exécuter entre des accents graves « ‘ ». (Alt Gr + 7).

```
$ mon_unix=‘uname‘
$ echo ${mon_unix}
OSF1
$ machine=‘uname -a | cut -d" " -f5‘
$ echo $machine
alpha
```

Attention, seul le canal de sortie standard est affecté à la variable. Le canal d'erreur standard sort toujours vers l'écran.

13.9 Tests de conditions

La commande **test** permet d'effectuer des tests de conditions. Le résultat est récupérable par la variable $? (code retour). Si ce résultat est 0 alors la condition est réalisée.

13.9.1 tests sur chaîne

- **test -z "variable"** : zero, retour OK si la variable est vide (ex test -z "$a")
- **test -n "variable"** : non zero, retour OK si la variable n’est pas vide (texte quelconque)
- **test "variable"** = chaîne : OK si les deux chaînes sont identiques
- **test "variable"** != chaîne : OK si les deux chaînes sont différentes

```
$ a=
$ test -z "$a" ; echo $?
0
$ test -n "$a" ; echo $?
1
$ a=Jules
$ test "$a" = Jules ; echo $?
0
```

13.9.2 tests sur valeurs numériques

Les chaînes à comparer sont converties en valeurs numériques. La syntaxe est

```
test valeur1 option valeur2
```

et les options sont les suivantes :

Option	*Rôle*
-eq	Equal : Egal
-ne	Not Equal : Différent
-lt	Less than : inférieur
-gt	Greater than : supérieur
-le	Less ou equal : inférieur ou égal
-ge	Greater or equal : supérieur ou égal

```
$ a=10
$ b=20
$ test  "$a" -ne "$b" ; echo $?
0
$ test "$a" -ge "$b" ; echo $?
1
$ test "$a" -lt "$b" && echo "$a est inferieur a $b"
10 est inferieur a 20
```

13.9.3 tests sur les fichiers

La syntaxe est

```
test option nom_fichier
```

et les options sont les suivantes :

Option	*Rôle*
-f	Fichier normal
-d	Un répertoire
-c	Fichier en mode caractère
-b	Fichier en mode bloc
-p	Tube nommé (named pipe)
-r	Autorisation en lecture
-w	Autorisation en écriture
-x	Autorisation en exécution
-s	Fichier non vide (au moins un caractère)
-e	Le fichier existe
-L	Le fichier est un lien symbolique
-u	Le fichier existe, SUID-Bit positionné
-g	Le fichier existe SGID-Bit positionné

```
$ ls -l
-rw-r--r--   1 oracle   system      1392 Aug 14 15:55 dump.log
lrwxrwxrwx   1 oracle   system         4 Aug 14 15:21 lien_fic1 -> fic1
lrwxrwxrwx   1 oracle   system         4 Aug 14 15:21 lien_fic2 -> fic2
-rw-r--r--   1 oracle   system       234 Aug 16 12:20 liste1
-rw-r--r--   1 oracle   system       234 Aug 13 10:06 liste2
-rwxr--r--   1 oracle   system       288 Aug 19 09:05 param.sh
-rwxr--r--   1 oracle   system       430 Aug 19 09:09 param2.sh
-rwxr--r--   1 oracle   system       292 Aug 19 10:57 param3.sh
drwxr-xr-x   2 oracle   system      8192 Aug 19 12:09 rep1
-rw-r--r--   1 oracle   system      1496 Aug 14 16:12 resultat.txt
-rw-r--r--   1 oracle   system      1715 Aug 16 14:55 toto.txt
-rwxr--r--   1 oracle   system        12 Aug 16 12:07 voir_a.sh
$ test -f lien_fic1 ; echo $?
1
$ test -x dump.log ; echo $?
1
$ test -d rep1 ; echo $?
0
```

13.9.4 tests combinés par critères ET OU NON

On peut effectuer plusieurs tests avec une seule instruction. Les options de combinaisons sont les mêmes que pour la commande **find**.

Critère	*Action*
-a	AND, ET logique
-o	OR, OU logique
!	NOT, NON logique
(...)	groupement des combinaisons. Les parenthèses doivent être vérouillées \(...\).

```
$ test -d "rep1" -a -w "rep1" && exho "rep1: repertoire, droit en ecriture"
rep1: repertoire, droit en ecriture
```

13.9.5 syntaxe allégée

Le mot test peut être remplacé par les crochets ouverts et fermés « **[...]** ». Il faut respecter un espace après et avant les crochets.

```
$ [ "$a" -lt "$b" ] && echo "$a est inferieur a $b"
10 est inferieur a 20
```

13.10 if ... then ... else

La structure **if then else fi** 'est une structure de contrôle conditionnelle.

```
if <commandes_condition>
then
          <commandes exécutées si condition réalisée>
else
          <commandes exécutées si dernière condition pas réalisée>
fi
```

On peut aussi préciser le **elif**, en fait un else if. Si la dernière condition n'est pas réalisée on en teste une nouvelle.

Exemple : test de la présence de paramètres

```
$ cat param4.sh
#!/usr/bin/sh
if [ $# -ne 0 ]
then
        echo "$# parametres en ligne de commande"
else
        echo "Aucun parametre; set alfred oscar romeo zoulou"
        set alfred oscar romeo zoulou
fi

echo "Nombre de parametres : $#"
echo "Parametres : 1=$1 2=$2 3=$3 4=$4"
echo "Liste : $*"

$ ./param4.sh titi toto
2 parametres en ligne de commande
Nombre de parametres : 2
Parametres : 1=toto 2=titi 3= 4=
Liste : toto titi

$ ./param4.sh
Aucun parametre; set alfred oscar romeo zoulou
Nombre de parametres : 4
Parametres : 1=alfred 2=oscar 3=romeo 4=zoulou
Liste : alfred oscar romeo zoulou
```

13.11 Choix multiples case

La commande **case esac** permet de vérifier le contenu d'une variable ou d'un résultat de manière multiple.

```
case Valeur in
          Modele1) Commandes ;;
          Modele2) Commandes ;;
          *) action_defaut ;;
esac
```

Le modèle est soit un simple texte, soit composé de caractères spéciaux. Chaque bloc de commandes lié au modèle doit se terminer par deux points-virgules. Dès que le modèle est vérifié, le bloc de commandes correspondant est exécuté. L'étoile en dernière position (chaîne variable) est l'action par défaut si aucun critère n'est vérifié.

Caractère	*Rôle*
*	Chaîne variable (même vide)
?	Un seul caractère
[...]	Une plage de caractères
[!...]	Négation de la plage de caractères
|	OU logique

```
$ cat case1.sh
```

```
#!/usr/bin/sh
if [ $# -ne 0 ]
then
        echo "$# parametres en ligne de commande"
else
        echo "Aucun parametre; set alfred oscar romeo zoulou"
        exit 1
fi

case $1 in
        a*)
                echo "Commence par a"
                ;;
        b*)
                echo "Commence par b"
                ;;
        fic[123])
                echo "fic1 fic2 ou fic3"
                ;;
        *)
                echo "Commence par n'importe"
                ;;
esac

exit 0
$ ./case1.sh "au revoir"
Commence par a
$ ./case1.sh bonjour
Commence par b
$ ./case1.sh fic2
fic1 ou fic2 ou fic3
$ ./case1.sh erreur
Commence par n'importe
```

13.12 Saisie de l'utilisateur

La commande **read** permet à l'utilisateur de saisir une chaîne et de la placer dans une ou plusieurs variable. La saisie est validée par entrée.

```
read var1 [var2 ...]
```

Si plusieurs variables sont précisées, le premier mot ira dans var1, le second dans var2, et ainsi de suite. S'il y a moins de variables que de mots, tous les derniers mots vont dans la dernière variable.

```
$ cat read.sh
#!/usr/bin/sh
echo "Continuer (O/N) ? \c"
read reponse
echo "reponse=$reponse"
case $reponse in
        O)
                echo "Oui, on continue"
                ;;
        N)
                echo "Non, on s'arrête"
                exit 0
                ;;
        *)
                echo "Erreur de saisie (O/N)"
                exit 1
                ;;
esac
echo "Vous avez continue. Tapez deux mots ou plus :\c"
```

```
read mot1 mot2
echo "mot1=$mot1\nmot2=$mot2"
exit 0
$ ./read.sh
Continuer (O/N) ? O
reponse=O
Oui, on continue
Vous avez continue. Tapez deux mots ou plus :salut les amis
mot1=salut
mot2=les amis
```

13.13 Les boucles

Elles permettent la répétition d'un bloc de commandes soit un nombre limité de fois, soit conditionnellement. Toutes les commandes à exécuter dans une boucle se placent entre les commandes **do** et **done**.

13.13.1 Boucle for

La boucle **for** ne se base pas sur une quelconque incrémentation de valeur mais sur une liste de valeurs, de fichiers ...

for var in liste

```
do
          commandes à exécuter
done
```

La liste représente un certain nombre d'éléments qui seront successivement attribuées à var1.

13.13.1.1 Avec une variable

```
$ cat for1.sh
#!/usr/bin/sh
for params in $@
do
        echo "$params"
done
$ ./for1.sh test1 test2 test3
test1
test2
test3
```

13.13.1.2 Liste implicite

Si on ne précise aucune liste à for, alors c'est la liste des paramètres qui est implicite. Ainsi le script précédent aurait pu ressembler à :

```
for params
do
        echo "$params"
done
```

13.13.1.3 Avec une liste d'éléments explicite :

```
$ cat for2.sh
#!/usr/bin/sh
for params in liste liste2
do
        ls -l $params
```

```
done
$ ./for2.sh
-rw-r--r--   1 oracle   system       234 Aug 19 14:09 liste
-rw-r--r--   1 oracle   system       234 Aug 13 10:06 liste2
```

13.13.1.4 Avec des critères de recherche sur nom de fichiers :

```
$ cat for3.sh
#!/usr/bin/sh
for params in *
do
        echo "$params \c"
        type_fic=‘ls -ld $params | cut -c1‘
        case $type_fic in
                -)      echo "Fichier normal" ;;
                d)      echo "Repertoire" ;;
                b)      echo "mode bloc" ;;
                l)      echo "lien symbolique" ;;
                c)      echo "mode caractere" ;;
                *)      echo "autre" ;;
        esac

done
$ ./for3.sh
case1.sh Fichier normal
dump.log Fichier normal
for1.sh Fichier normal
for2.sh Fichier normal
for3.sh Fichier normal
lien_fic1 lien symbolique
lien_fic2 lien symbolique
liste Fichier normal
liste1 Fichier normal
liste2 Fichier normal
param.sh Fichier normal
param2.sh Fichier normal
param3.sh Fichier normal
param4.sh Fichier normal
read.sh Fichier normal
rep1 Repertoire
resultat.txt Fichier normal
toto.txt Fichier normal
voir_a.sh Fichier normal
```

13.13.1.5 Avec une substitution de commande

```
$ cat for4.sh
#!/usr/bin/sh
echo "Liste des utilisateurs dans /etc/passwd"
for params in ‘cat /etc/passwd | cut -d: -f1‘
do
        echo "$params "
done
$ ./for4.sh
Liste des utilisateurs dans /etc/passwd
root
nobody
nobodyV
daemon
bin
uucp
uucpa
auth
```

```
cron
lp
tcb
adm
ris
carthic
ftp
stu
...
```

13.13.2 Boucle while

La commande **while** permet une boucle conditionnelle « tant que ». Tant que la condition est réalisée, le bloc de commande est exécuté. On sort si la condition n'est plus valable.

```
while condition
do
          commandes
done
```

ou

```
while
bloc d’instructions formant la condition
do
          commandes
done
```

Par exemple :

```
$ cat while1.sh
#!/usr/bin/sh
while
        echo "Chaine ? \c"
        read nom
        [ -z "$nom" ]
do
        echo "ERREUR : pas de saisie"
done
echo "Vous avez saisi : $nom"
```

Par exemple une lecture d'un fichier ligne à ligne :

```
#!/usr/bin/sh
cat toto.txt | while read line
do
          echo "$line"
done
```

ou

```
#!/usr/bin/sh
while read line
do
          echo "$line"
done < toto.txt
```

13.13.3 Boucle until

La commande **until** permet une boucle conditionnelle « jusqu'à ». Dès que la condition est réalisée,

on sort de la boucle.

```
until condition
do
          commandes
done
```

ou

```
until
bloc d’instructions formant la condition
do
          commandes
done
```

13.13.4 seq

La commande seq permet de sortir une séquence de nombres, avec un intervalle éventuel.

```
seq [debut] [increment] fin
```

ex :

```
$ seq 5
1
2
3
4
5
$ seq -2 3
-2
-1
0
1
2
3
$ seq 0 2 10
0
2
4
6
8
10
```

13.13.5 true et false

La commande **true** ne fait rien d'autre que de renvoyer 0. La commande **false** renvoie toujours 1. De cette manière il est possible de faire des boucles sans fin. La seule manière de sortir de ces boucles est un exit ou un break.

```
while true
do
          commandes
          exit / break
done
```

13.13.6 break et continue

La commande **break** permet d'interrompre une boucle. Dans ce cas le script continue après la commande **done**. Elle peut prendre un argument numérique indiquant le nombre de boucles à sauter, dans le cadre de boucles imbriquées (rapidement illisible).

```
while true
do
          echo "Chaine ? \c"
          read a
          if [ -z "$a" ]
          then
                    break
          fi
done
```

La commande **continue** permet de relancer une boucle et d'effectuer un nouveau passage. Elle peut prendre un argument numérique indiquant le nombre de boucles à relancer (on remonte de n boucles). Le script redémarre à la commande **do**.

13.14 Les fonctions

Les fonctions sont des bouts de scripts nommés, directement appelés par leur nom, pouvant accepter des paramètres et retourner des valeurs. Les noms de fonctions suivent les mêmes règles que les variables sauf qu'elles ne peuvent pas être exportées.

```
nom_fonction ()
{
          commandes
          return
}
```

Les fonctions peuvent être soit tapées dans votre script courant, soit dans un autre fichier pouvant être inclus dans l'environnement. Pour cela :

```
. nomfic
```

Le point suivi d'un nom de fichier charge son contenu (fonctions et variables dans l'environnement courant.

La commande **return** permet d'affecter une valeur de retour à une fonction. Il ne faut surtout par utiliser la commande exit pour sortir d'une fonction, sinon on quitte le script.

```
$ cat fonction
ll ()
{
        ls -l $@
}
 li ()
{
        ls -i $@
}
$ . fonction
$ li
  252 case1.sh        326 for4.sh         187 param.sh        897 resultat.txt
  568 dump.log        409 lien_fic1       272 param2.sh       991 toto.txt
  286 fonction        634 lien_fic2       260 param3.sh       716 voir_a.sh
  235 for1.sh        1020 liste            42 param4.sh      1008 while1.sh
  909 for2.sh         667 liste1          304 read.sh
  789 for3.sh        1006 liste2          481 rep1
```

13.15 expr

La commande **expr** permet d'effectuer des calculs sur des valeurs numériques, des comparaisons, et de la recherche dans des chaînes de texte.

Opérateur	*Rôle*
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
!=	Différent. Affiche 1 si différent, 0 sinon.
=	Egal. Affiche 1 si égal, 0 sinon.
<	inférieur. Affiche 1 si inférieur, 0 sinon.
>	supérieur. Affiche 1 si supérieur, 0 sinon.
<=	inférieur ou égal. Affiche 1 si inférieur ou égal, 0 sinon.
>=	supérieur ou égal. Affiche 1 si supérieur ou égal, 0 sinon.
:	Recherche dans une chaîne. Ex expr Jules : J* retourne 1 car Jules commence par J. Syntaxe particulière : expr "Jules" : ".*" retourne la longueur de la chaîne.

```
$ expr 7 + 3
10
$ expr 7 \* 3
21
$ a=`expr 13 – 10`
$ echo $a
3
$ cat expr1.sh
#!/usr/bin/sh
cumul=0
compteur=0
nb_boucles=10
while [ "$compteur" -le "$nb_boucles" ]
do
        cumul=`expr $cumul + $compteur`
        echo "cumul=$cumul, boucle=$compteur"
        compteur=`expr $compteur + 1`
done
$ ./expr1.sh
cumul=0, boucle=0
cumul=1, boucle=1
cumul=3, boucle=2
cumul=6, boucle=3
cumul=10, boucle=4
cumul=15, boucle=5
cumul=21, boucle=6
cumul=28, boucle=7
cumul=36, boucle=8
cumul=45, boucle=9
cumul=55, boucle=10
$ expr "Jules Cesar" : ".*"
11
```

13.16 Une variable dans une autre variable

Voici un exemple :

```
$ a=Jules
$ b=a
$ echo $b
a
```

Comment afficher le contenu de a et pas simplement a ? En utilisant la commande **eval**. Cette commande située en début de ligne essaie d'interpréter, si possible, la valeur d'une variable précédée par deux « $ », comme étant une autre variable.

```
$ eval echo \$$b
Jules
```

13.17 Traitement des signaux

La commande **trap** permet de modifier le comportement du script à la réception d'un signal.

commande	*Réaction*
trap " signaux	Ignore les signaux. trap " 2 3 ignore les signaux 2 et 3
trap 'commandes' signaux	Pour chaque signal reçu exécution des commandes indiquées
trap signaux	Restaure les actions par défaut pour ces signaux

13.18 Commande « : »

La commande « : » est généralement totalement inconnue des utilisateurs unix. Elle retourne toujours la valeur 0 (réussite). Elle peut donc être utilisée pour remplacer la commande true dans une boucle par exemple :

```
while :
do
          ...
done
```

Cette commande placée en début de ligne, suivie d'une autre commande, traite la commande et ses arguments mais ne fait rien, et retourne toujours 0. Elle peut être utile pour tester les variables.

```
$ : ls
$ : ${X:?"Erreur"}
X : Erreur
```

13.19 Délai d'attente

La commande **sleep** permet d'attendre le nombre de secondes indiqués. Le script est interrompu durant ce temps. Le nombre de secondes et un entier compris entre 0 et 4 milliards (136 ans).

```
$ sleep 10
```

14 Particularités du Korn Shell

Si le Bourne Shell est un shell POSIX présent et standard sur tous les UNIX, le Korn Shell, apparu dans sa version moderne sous System V release 4 devient aujourd'hui un standard de fait. Le Korn Shell ou ksh est entièrement compatible avec le Bourne Shell (scripts exécutables sans modification) et le Bourne Again Shell (bash sous Linux). Ksh est présent sous Linux avec le nom pdksh.

Parmi les nouvelles possibilités :

- Les alias
- Possibilité de typer les variables, gestion des tableaux et des chaînes
- commandes supplémentaires
- gestion des jobs

Le fichier de configuration par défaut est le fichier **.profile**. On peut y placer ses définitions de variables, alias, ...

14.1 Historique et répétition

On peut accéder à l'historique des commandes avec la commande **fc**. Cette commande permet aussi de rappeler une ligne précise ou d'en modifier le contenu. La taille de l'historique se contrôle avec la valeur de la variable **HISTSIZE** (HISTSIZE=200 prendra en compte les 200 dernières commandes).

```
$ fc -l
201     ls -ltr
202     pg  transpo19659.lst
203     ls -ltr
204     pg syn19659.lst
205     exit
206     fc -l
207     man fc
208     fc -l
$ fc -s 201
ls -ltr
total 30
lrwxrwxrwx   1 oracle   system         4 Aug 14 15:21 lien_fic1 -> fic1
lrwxrwxrwx   1 oracle   system         4 Aug 14 15:21 lien_fic2 -> fic2
-rw-r--r--   1 oracle   system      1392 Aug 14 15:55 dump.log
-rw-r--r--   1 oracle   system      1496 Aug 14 16:12 resultat.txt
-rwxr--r--   1 oracle   system        12 Aug 16 12:07 voir_a.sh
-rw-r--r--   1 oracle   system       234 Aug 16 12:20 liste1
-rw-r--r--   1 oracle   system      1715 Aug 16 14:55 toto.txt
-rwxr--r--   1 oracle   system       288 Aug 19 09:05 param.sh
-rwxr--r--   1 oracle   system       430 Aug 19 09:09 param2.sh
-rwxr--r--   1 oracle   system       292 Aug 19 10:57 param3.sh
drwxr-xr-x   2 oracle   system      8192 Aug 19 12:09 rep1
-rwxr--r--   1 oracle   system       265 Aug 19 12:38 param4.sh
-rwxr--r--   1 oracle   system       327 Aug 19 13:31 case1.sh
-rwxr--r--   1 oracle   system       338 Aug 19 13:51 read.sh
```

14.2 Modes vi et emacs

Par défaut les touches de la ligne de commande émettent régulièrement des caractères bizarres, ou rien du tout, et ne sont pas interprétées par le shell. C'est souvent le cas des touches de direction, ou backspace.

```
$ toucd^[[D^[[D^[[D^[[A^[[B^[[C
```

La commande

```
$ set -o vi
```

commute la ligne de commande en mode **vi**, permettant l'utilisation des touches de types vi pour la saisie et le rappel de commandes avec l'utilisation de la touche Echap.

Touche	*Action*
k	Vers le haut, rappel d'une ligne précédente de l'historique
j	Vers le bas, ligne suivante de l'historique
h	Vers la gauche, curseur vers la gauche
l	Vers la droite, curseur vers la droite
w	Curseur sur le mot suivant
b	Curseur sur le mot précédent
$	Vers la fin de ligne
0 (zéro)	Vers le début de ligne
x	Suppression d'un caractère
i	Insertion devant le caractère courant
a	Insertion derrière le caractère actif

```
$ set -o emacs
```

passe la ligne de commande en mode **emacs**. Un avantage certain dans ce cas est que les touches directionnelles peuvent être utilisées pour déplacer le curseur et afficher l'historique.

Le contenu de la variable **VISUAL** peut être modifié en conséquence avec la valeur vi ou emacs.

14.3 Les alias

Un **alias** est une substitution d'une commande par un raccourci. L'alias est prioritaire sur les fonctions, commandes internes et commandes externes. Il est possible d'y substituer du texte.

```
alias nom_alias=commande_ou_texte
```

```
$ alias deltree='rm -rf'
$ deltree rep1
```

La substitution ne s'effectue que si l'alias est la première commande ou si le texte de l'alias se termine par un espace.

```
$ alias list=ls -l '
$ alias home='/tmp/seb'
$ list home
total 22
-rwxr--r--   1 oracle   system       327 Aug 19 13:31 case1.sh
-rw-r--r--   1 oracle   system      1392 Aug 14 15:55 dump.log
-rwxr--r--   1 oracle   system       200 Aug 19 15:58 expr1.sh
-rw-r--r--   1 oracle   system        42 Aug 19 15:43 fonction
-rwxr--r--   1 oracle   system        57 Aug 19 14:06 for1.sh
-rwxr--r--   1 oracle   system        66 Aug 19 14:09 for2.sh
```

```
-rwxr--r--   1 oracle   system       285 Aug 19 14:32 for3.sh
-rwxr--r--   1 oracle   system       133 Aug 19 14:37 for4.sh
lrwxrwxrwx   1 oracle   system         4 Aug 14 15:21 lien_fic1 -> fic1
lrwxrwxrwx   1 oracle   system         4 Aug 14 15:21 lien_fic2 -> fic2
...
```

Sans paramètre, c'est la liste des alias qui s'affiche.

```
$ alias
autoload='typeset -fu'
cat=/usr/bin/cat
command='command '
deltree='rm -rf'
functions='typeset -f'
grep=/usr/bin/grep
hash='alias -t -'
history='fc -l'
home=/tmp/seb
integer='typeset -i'
list='ls -l '
local=typeset
ls=/usr/bin/ls
nohup='nohup '
r='fc -e -'
rm=/usr/bin/rm
stop='kill -STOP'
suspend='kill -STOP $$'
type='whence -v'
```

L'option -x permet d'exporter l'alias. Enfin la commande **unalias** supprime l'alias.

14.4 Modifications concernant les variables

14.4.1 Variables système

Voici quelques nouvelles variables système.

Variable	*Contenu*
ENV	Nom du fichier devant être exécuté à chaque chargement du Korn Shell, en plus de /etc/profile et $HOME/.profile. Généralement il s'agit de $HOME/.kshrc
FPATH	Chemin de recherche pour les fonctions inconnues. Si une fonction n'est pas connue, le shell recherche un fichier dans ce chemin portant le nom de cette fonction et intègre son contenu.
HISTFILE	Nom du fichier historique, généralement $HOME/.sh_history
HISTSIZE	Taille en nombre de lignes de l'historique
OLDPWD	Chemin d'accès du répertoire accédé précédemment.
PS3	Définit l'invite de saisie pour un select
PWD	Chemin d'accès courant
RANDOM	Génère et contient un nombre aléatoire entre 0 et 32767
SECONDS	Nombre de secondes depuis le lancement du shell
TMOUT	Délai d'attente maxi en secondes pour une saisie. A la fin de ce temps le shell se quitte
VISUAL	Mode d'édition en ligne, vi ou emacs.

14.4.2 Longueur d'une chaîne

Il est maintenant possible d'obtenir la longueur d'une chaîne autrement qu'avec la commande expr. On utilise le caractère « **#** »

```
$ a=Jules
$ echo "Longueur de $a : ${#a}"
Longueur de Jules : 5
```

14.4.3 Tableaux et champs

Le ksh introduit la gestion des tableaux de variables Deux moyens sont disponibles pour déclarer un tableau, l'un avec l'utilisation des crochets « **[]** », l'autre avec la commande « **set -A** ». Le premier élément est 0 le dernier 1023. Pour accéder au contenu du tableau il faut mettre la variable ET l'élément entre accolades « **{}** ».

```
$ Nom[0]="Jules"
$ Nom[1]="Romain"
$ Nom[2]="Francois"
$ echo ${Nom[1]}
Romain
```

ou

```
$ set -A Nom Jules Romain Francois
$ echo ${nom[2]}
Francois
```

Pour lister tous les éléments :

```
$ echo ${Nom[*]}
Jules Romain Francois
```

Pour connaître le nombre d'éléments

```
$ echo ${#Nom[*]}
3
```

Si l'index est une variable, on ne met pas le $:

```
$ idx=0
$ echo ${Nom[idx]}
Jules
```

14.4.4 Opérations sur chaînes

Les recherches sur chaînes sont maintenant possibles au sein même de la variable. Le texte trouvé en fonction de critères est supprimé de la variable.

```
${variable<Opérateur>Critère}
```

Opérateur	***Rôle***
#	Cherche la plus petite survenance du critère en début de chaîne
##	Cherche la plus grande survenance du critère en début de chaîne
%	Cherche la plus petite survenance du critère en fin de chaîne
%%	Cherche la plus grande survenance du critère en fin de chaîne

```
$ a="Bonjour les amis"
$ echo ${a#Bon*}
jour les amis
$ echo ${a##Bon*}
```

La commande **typeset** propose quelques options intéressantes

Option	*Rôle*
-l	Les majuscules sont converties en majuscules et le contenu sera toujours automatiquement converti en minuscules.
-u	Les minuscules sont converties en majuscules, même remarque.
-r	La variable sera en Read Only
-x	La variable est exportée
-Ln	Justification à gauche avec suppression des espaces à droite et gauche. si n est positionné la chaîne est tronquée à n caractères ou augmentée avec des espaces à droite.
--Rn	Idem mais pour la justification à droite
-Z	Si le premier caractère est un chiffre, justification et remplissage avec des 0 (zéro), n étant la taille du champ.

```
$ typeset -u txt1
$ txt1="abcdefg"
ABCDEFG
$ typeset -u txt1
$ echo $txt1
abcdefg
```

14.4.5 Variables typées

Les variables peuvent être typées en entier (integer) avec la commande « t**ypeset -i** » le permet. L'avantage est qu'il devient possible d'effectuer des calculs et des comparaisons sans passer par expr. La commande **let** ou « **((...))** » permet des calcules sur variables.

Opérateur	*Rôle*
+ - * /	Opérations simples
%	Modulo
< > <= >=	Comparaisons. 1 si vraie, 0 si faux
== !=	Egal ou différent
&& ||	Comparaisons liées par un opérateur logique
& | ^	logique binaire AND OR XOR

```
$ typeset -i resultat
$ resultat=6*7
$ echo $resultat
42
$ resultat=Erreur
ksh: Erreur: bad number
$ resultat=resultat*3
126
```

```
$ typeset -i add
$ add=5
$ let resultat=add+5 resultat=resultat*add
$ echo $resultat
50
```

14.5 Nouvelle substitution de commande

On peut toujours utiliser les accents pour effectuer une substitution de commandes, mais il est maintenant plus simple d'utiliser la syntaxe « **$(...) »**.

```
$ date_courante=$(date)
$ echo $date_courante
Tue Aug 20 16:07:05 MET DST 2002
```

14.6 cd

Nous avons vu deux nouvelles variables nouvelles PWD et OLDPWD. La nouvelle commande **cd** permet de les exploiter. « cd - » on retourne dans le catalogue précédent.

```
$ pwd
/tmp/seb
$ cd ..
$ pwd
/tmp
$ cd -
$ pwd
/tmp/seb
```

Le tilde permet un raccourci pour le répertoire utilisateur.

```
$ cd ~rep1
$ pwd
/tmp/seb/rep1
```

Enfin,

```
$ cd rep1 rep2
$ pwd
/tmp/seb/rep2
```

14.7 Gestion de jobs

Lorsqu'un processus est lancé en tâche de fond, ksh affiche en plus du PID un numéro entre crochets. Il s'agit d'un numéro de job, incrémenté de 1 à chaque nouveau lancement si les commandes précédentes ne sont pas terminées. La commande **jobs** permet d'obtenir des informations.

```
$ ls -lR > /toto.txt &
[1]     25994
$ jobs -l
[1] +  Running                 ls -lR / >toto.txt 2>&1 &
```

Le processus de PID 25994 a été lancé avec le numéro de job 1. Son état actuel est **running** (il peut être **Done**, **Stopped**, **Terminated**).

Pour interrompre un processus en premier plan sans le quitter (le passer en stopped mais pas en terminated) on utilise généralement la séquence **Ctrl+Z**. Dans ce cas le processus est stoppé mais pas terminé, et on a accès à la ligne de commande. Pour le relancer en arrière-plan on utilise la

commande **bg** %n (background, n numéro de job). Pour replacer une commande en arrière-plan au premier plan, on utilise la commande **fg** %n (foreground, n numéro de job).

```
$ ls -lR / >toto.txt 2>&1
[1] + Stopped                  ls -lR / >toto.txt 2>&1
$ bg %1
[1]     ls -lR / >toto.txt 2>&1&
$ jobs -l
[1] + 26329      Running                 ls -lR / >toto.txt 2>&1
$ fg %1
ls -lR / >toto.txt 2>&1
```

La commande **kill** dans sa nouvelle syntaxe peut effectuer la même chose.

```
kill -STOP %1 <-> Ctrl+Z
kill -CONT %1 <-> fg %1
```

14.8 print

La commande **print** est une extension de la commande echo, et accepte de nouveaux paramètres en plus de ceux de la commande echo.

Option	*Rôle*
-	Les mots situés après sont des paramètres, pas des options
-R	Les caractères spéciaux ne sont plus interprétés
-n	N'effectue pas de saut de ligne
-s	Les paramètres sont consignés dans le fichier historique
-nChiffre	Les paramètres sont envoyés dans le canal n

```
$  print  "Bonjour, \n Comment ça va ?\c"
Bonjour,
 Comment ça va ?$
$ print -R "Bonjour, \n Comment ça va ?\c"
Bonjour, \n Comment ça va ?\c
```

14.9 Tests étendus

Une nouvelle commande interne au ksh permet des tests étendus et généralement plus performants, par l'utilisation des doubles-crochets « **[[...]]** ». Il existe quelques différences entre la commande test (ou []) et la nouvelle.

- Les caractères spéciaux (métacaractères) de recherche de fichiers ne sont pas interprétés.
- -a et -o (ET et OU) sont remplacés pas « && » et « || ».
- Si deux chaînes sont comparées, la deuxième peut être un modèle (de type case).
- La recherche des opérateurs est effectuée avant la substitution des variables.

```
$ [[ -d "rep1" && -r "rep1" ]] && echo "rep1 ; repertoire avec droits write"
```

Voici les nouveaux modèles pour la comparaison de texte :

Modèle	*Rôle*
?(Critere1|Critere2...)	Un seul des critères peut survenir
*(Critere1|Critere2...)	Chaque critère peut survenir plusieurs fois
+(Critere1|Critere2...)	Un des critères doit survenir au moins une fois
@(Critere1|Critere2...)	Au moins un critère doit survenir au moins une fois
!(Critere1|Critere2...)	Aucun des critères ne doit survenir

14.10 Options du shell

La commande **set** permet d'autres options que vi et emacs. L'option -o active l'option, +o l'annule. La commande « set -o » sans rien d'autre affiche la liste des options et leur état.

- **allexport** : toutes les variables déclarées seront automatiquement exportées
- **bgnice** : les processus lancés en tâche de fond ont un facteur nice plus important et donc tournent avec une priorité moindre
- **ignoreeof** : La combinaison Ctrl+D n'est plus interprétée.
- **noclobber** : la redirection > n'écrase plus le fichier et produit un message d'erreur s'il existe. Pour l'écraser tout de même : >|

```
$ set -o noclobber
$ wc -l toto.txt
    378264 toto.txt
$ ls > toto.txt
ksh: toto.txt: file already exists
$ ls >| toto.txt
$ wc -l toto.txt
        18 toto.txt
$ set +o noclobber
```

14.11 Commande whence

La commande **whence** indique le type de commande lancée

```
$ whence -v cd
cd is a shell builtin
$ whence -v test
test is a shell builtin
$ whence -v ls
ls is a tracked alias for /usr/bin/ls
$ whence -v rm
rm is a tracked alias for /usr/bin/rm
$ whence -v echo
echo is a shell builtin
$ whence -v touch
touch is /usr/bin/touch
```

Valeur	*Signification*
shell builtin	Commande interne au shell
Alias	Alias de commande
tracked alias	Alias avec suite

Valeur	*Signification*
keyword	Mot-clé
exported alias	alias exporté
function	Une fonction du shell
undefined function	fonction non définie
program (nom du programme)	un programme externe

14.12 Commande select

La commande **select** permet de créer des menus simples, avec sélection par numéro. La saisie s'effectue au clavier avec le prompt de la variable PS3. Si la valeur saisie est incorrecte, une boucle s'effectue et le menu s'affiche à nouveau. Pour sortir d'un select il faut utiliser un **break**.

```
select variable in liste_contenu
do
          traitement
done
```

Si in liste_contenu n'est pas précisé, se sont les paramètres de position qui sont utilisés et affichés.

```
$ cat select.ksh
!#/usr/bin/ksh
PS3="Votre choix :"
echo "Quelle donnee  ?"
select reponse in Jules Romain Francois quitte
do
        if [[ "$reponse" = "quitte" ]]
        then
                break
        fi
        echo "Vous avez choisi $reponse"
done
echo "Au revoir."
exit 0
$ ./select.ksh
./select.ksh: !#/usr/bin/ksh:  not found
Quelle donnee  ?
1) Jules
2) Romain
3) Francois
4) quitte
Votre choix :1
Vous avez choisi Jules
Votre choix :3
Vous avez choisi Francois
Votre choix :4
Au revoir.
```

14.13 read et |&

Le Bourne shell ne proposait pas de mécanisme simple pour lire par exemple un fichier ligne à ligne. Le Korn Shell permet d'envoyer le résultat d'une commande dans un tube avec la syntaxe « **|&** » et de récupérer ce résultat avec la commande **read -p**.

```
commande |&
read -p variable
```

Par exemple, lire un fichier ligne par ligne

```
$ cat read.ksh
#!/usr/bin/ksh
typeset -i compteur
compteur=1
cat liste |&
while read -p ligne
do
        echo "$compteur\t $ligne"
        compteur=compteur+1
done
$ ./read.ksh
1        Produit        objet   prix    quantites
2        souris         boutons 30      15
3        dur    30giga  100     30
4        dur    70giga  150     30
5        disque zip     12      30
6        disque souple  10      30
7        ecran  15      150     20
8        ecran  17      300     20
9        ecran  19      500     20
10       ecran  21      500     20
11       clavier        105     45      30
12       clavier        115     55      30
```

15 Compléments

15.1 La Crontab

Le mécanisme **cron** permet l'exécution régulière et automatique de commandes. A chaque démarrage de la machine (en fait suivant le niveau d'init) un service (daemon) est lancé, généralement appelé cron ou crontab (/usr/sbin/cron, /etc/cron, /sbin/cron, ...) qui lit une table spécifique appelée crontab.

La crontab est généralement placée dans /var/adm/cron ou /etc/cron ou /usr/lib/cron. Tous les utilisateurs n'ont pas forcément le droit d'utiliser la crontab (cron.allow et cron.deny).

La commande **crontab** permet d'accéder et de modifier les données de la table.

Pour lister une crontab, on utilise l'option « -l ». L'option « -r » permet de supprimer tous les ordres de sa crontab. Pour éditer sa crontab personnelle, on utilise l'option « **-e** ». Un éditeur vi est lancé. A la sauvegarde, le contenu qui aura été saisi sera le contenu de la crontab de l'utilisateur. le format est le suivant :

(1)minutes (0 à 59)

(2)heures (0 à 23)

(3)jour dans le mois (1 à 31)

(4)mois (de 1 à 12)

(5)jour de la semaine (de 0 dimanche à 6 samedi)

(6)commande

Une étoile « * » à la place d'un paramètre indique que se paramètre ne doit pas être pris en compte. Le tiret entre deux valeur définit une plage. Par exemple :

```
0 0 * * * rm -rf /tmp/* >/dev/null 2>&1
```

va exécuter la commande rm (ici effacement du contenu de /tmp) tous les jours à minuit pile.

```
0,30 9-18 * * 1-5 who >> /tmp/presence
```

toutes les demi-heures, de 9 heures à 18 heures, du lundi au vendredi, liste de tous les utilisateurs connectés.

15.2 Messages aux utilisateurs

La commande **write** permet d'envoyer un message à un utilisateur. Par défaut l'utilisateur doit être connecté sur la même machine (sur un autre terminal). On peut aussi tenter d'écrire à un utilisateur sur une autre machine. La syntaxe est

```
write user terminal [ligne]
write user@machine [ligne]
write -n machine user [ligne]
```

La commande **write** est interactive, une fois la connexion établie les utilisateurs peuvent communiquer en direct. Pour quitter la saisie : Ctrl+D.

La commande **wall** permet d'envoyer une information à tous les utilisateur connectés.

L'utilisateur qui ne souhaite pas voir les messages, ou l'administrateur qui veut en empêcher l'utilisation abusive, peut utiliser la commande **mesg**.

```
mesg y (autorisation)
```

```
mesg n (interdiction)
```

15.3 ftp

Bien que n'étant pas une commande propre à Unix, il est utile de connaître la commande **ftp** (file transfert protocol). Elle permet le transfert de fichiers entre deux machines. Elle prend comme paramètre le nom de la machine distante. Pour que la commande ftp fonctionne, il faut que le service ftp fonctionne sur la machine distante et sur le port 21.

Voici un exemple (peu pratique) de connexion avec erreur et nouvel essai.

```
ftp> open
(to) machine
Connected to machine.
220 machine FTP server (Digital UNIX Version 5.60) ready.
Name (machine:root): root
331 Password required for root.
Password:
530 Login incorrect.
Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> user
(username) root
331 Password required for root.
Password:
230 User root logged in.
ftp> pwd
257 "/" is current directory.
```

Le plus simple est tout de même :

```
$ ftp machine
Connected to machine.
220 machine FTP server (Digital UNIX Version 5.60) ready.
Name (machine:root): root
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Voici une liste de commandes ftp.

Commande	*Action*
open	Suivi d'un nom de machine, ouvre une connexion sur la machine spécifiée.
user	Saisie de l'utilisateur distant pour une connexion.
quit	Fin de la connexion et fin de la commande ftp.
ascii	Transfert des fichiers en mode ASCII (conversion des caractères spéciaux et fin de ligne en MS et Unix par exemple).
binary	Transfert des fichiers en mode binaire.
glob	Supprime l'interprétation des caractères spéciaux.
help	Affiche l'aide.

Commande	*Action*
prompt	Suivi de on ou off, active ou désactive la confirmation individuelle de transfert pour chaque fichier (mget ou mput).
pwd	Affiche le répertoire distant courant.
cd	Suivi du chemin, déplacement dans l'arborescence distante.
ls	liste les fichiers de la machine distante.
delete	Suivi d'un nom de fichier, supprime le fichier distant.
mdelete	Multiple. Supprime les fichiers distants.
get	Récupère le fichier distants.
mget	Multiple. Récupère les fichiers distants (liste ou modèle).
put	Envoie le fichier local vers la machine distante.
mput	Multiple. Envoie les fichiers locaux sur la machine distante (liste ou modèle).
close / disconnect	ferme la session actuelle.
lcd	Change de répertoire sur la machine locale.
hash	Durant les transferts, écrit un « # » sur écran pour chaque buffer transféré.
system	Informations sur le système distant.
recv	Réception d'un fichier.
send	Envoi d'un fichier.
rename	renomme un fichier distant.
mkdir	Crée un répertoire sur la machine distante.
rmdir	Supprime un répertoire sur la machine distante.
!commande	exécute la commande locale

Index des commandes