INTRODUCTION AU SYSTEME UNIX

1. Historique

De nos jours, le mot Unix désigne une famille de systèmes d'exploitation : System V version 4, OSF/1, BSD Berkeley 4.3 et 4.4, AIX, HP-UX, Solaris, Ultrix, Linux, Posix, etc.

Deux chercheurs des Laboratoires Bell sont à l'origine du système Unix: Ken Thompson ingénieur en électricité et Dennis M.Ritchie docteur en mathématiques appliquées (avec une thèse sur les fonctions récursives). En 1969, Les laboratoires Bell avait jugé le coût de développement du système d'exploitation Multics trop élevé et décidé d'arrêter le projet. A la recherche d'un environnement permettant de construire du logiciel plus facilement, les deux chercheurs des Bell Labs élaborèrent une première version d'un système mono-utilisateur, néanmoins capable de multiprogrammation, sur un PDP-7 de Digital Equipment Corporation (DEC). Ce système baptisé Unix (en 1970), nom antinomique de Multics, ne permettait pas le temps partagé mais possédait déjà un système de fichiers. Saisissant par la suite d'une opportunité d'une demande du département des brevets, ils proposèrent un ensemble bureautique basé sur la version initiale d'Unix sous le nom "système d'édition pour les tâches de bureau". Après le dépôt du brevet, Thompson et Richie obtinrent un PDP-11/20 avec une unité de gestion mémoire PDP-10. Le système Unix fut alors modifié pour permettre la gestion de plusieurs terminaux, le système devint multi-utilisateur. A cette époque, la plupart des concepts du système Unix actuel étaient en place: système de fichiers, gestion des processus, interface avec le système et commandes utilitaires.

Le système Unix et ses utilitaires avaient été écrits en assembleur, un langage proche du langage machine. La nécessité de générer sans arrêt de nouvelles installations de Unix prenant en compte toujours plus de terminaux conduisirent la réécriture de Unix dans un langage de programmation évolué. Thompson avait travaillé en BCPL (Basic Combined Programming Language) qui était une simplification de CPL, lui-même tiré d'Algol60 (langage proche de Pascal). Ayant commencé la réécriture en Fortran, il mit au point le langage B inspiré du langage BCPL. Ritchie retravailla le langage B, ajouta des types et des structures et un générateur de code exécutable : le langage C était né. Le système Unix réécrit en langage C en 1973 fut un succès.

De nos jours, le système d'exploitation Unix est un système multitâche et multi-utilisateur en temps partagé.

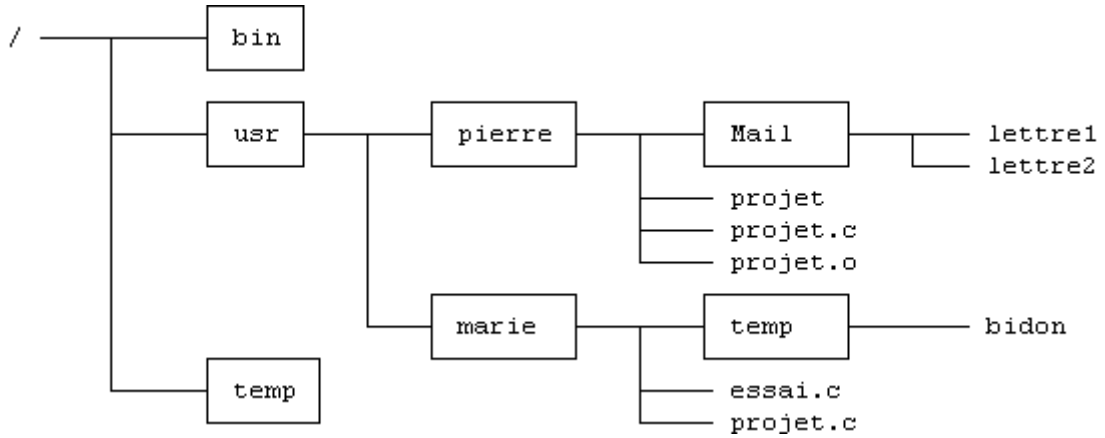
1.1. Organisation du systeme Unix

Le système Unix est constitué de différentes parties. Le core est le noyau central du système, on y trouve les fonctions de base et un automate d'affectation de ressources de type temps partagé. Vient ensuite le kernel (noyau) où se trouvent les pilotes (drivers) d'unités assurant les services d'entrée/sortie pour les divers types de périphériques connectables. Le shell (coquille) permet la communication avec l'utilisateur : il comprend un langage de commande souple (plusieurs versions disponibles) et les mécanismes faisant sa convivialité: redirection des E/S, utilitaires fondamentaux, pipes et filtres, gestion d'exécution en avant-plan et arrière-plan. La dernière couche est l'ensemble des outils et des applications comme la gestion du système de fichiers hiérarchisé, la manipulation de chaînes de caractères, des compilateurs, des traitements de texte, des outils de communication et de connexion réseaux.

Chaque programme en cours d'exécution est appelé processus et reçoit un numéro d'identification : le PID (Processus IDentification). Il est possible de connaître à l'aide d'une commande (*ps*) de connaître les différents processus en cours ainsi que le temps consommé par chacun d'entre eux. Si un programme semble "boucler", il est possible de l'arrêter à l'aide d'une autre commande (*kill*).

2. Système de fichiers hiérarchisés

Tous les fichiers sont considérés comme des ensembles d'octets. L'ensemble des fichiers est hiérarchisé: il est organisé en répertoires (directory) et se présente sous forme d'arbre inversé. Pour atteindre une branche inférieure, on liste les noms successivement en partant de la racine: au sommet on trouve le répertoire root (la racine) dont le nom est le caractère / (slash).



Un répertoire contient des fichiers et/ou des sous-répertoires. Dans le schéma ci-dessus, les noms de répertoires apparaissent dans un rectangle. Le fichier projet.c de Pierre et le fichier projet.c de Marie sont deux fichiers distincts et indépendants. Les noms de chemin absolu (pathname) permettant de désigner ces deux fichiers sont respectivement /usr/pierre/projet.c et /usr/marie/projet.c.

Le nom terminal d'un chemin de fichier est appelé basename ou nom local (par rapport au répertoire le contenant).

Les noms des répertoires importants sont : /temp le répertoire des fichiers temporaires, /dev le répertoire des unités (devices), /usr le répertoire des utilisateurs, /etc le répertoire des programmes et des fichiers spéciaux pour l'administration du système et /bin le répertoire des fichiers exécutables et des utilitaires Unix.

A un instant donné, on ne peut voir que les constituants (appelés encore entrées) du répertoire dans lequel "on se trouve". La commande *ls* permet d'afficher la liste des entrées du répertoire courant. La commande *cd* permet de changer de répertoire courant: le nom du nouveau répertoire courant doit suivre la commande *cd*. Exemple :

```
cd  /usr/marie/temp
```

Il est possible de se déplacer dans l'arborescence des répertoires de manière relative. Pour cela, on utilise la notation "." qui désigne le répertoire courant et la notation ".." qui désigne le répertoire supérieur. Supposons que l'on se trouve dans /usr/pierre, pour aller dans /usr/marie/temp, on pourra faire :

```
cd ..                    on se trouve dans /usr
cd ./marie (ou encore cd marie) on se trouve dans /usr/marie
cd   temp                on se trouve dans /usr/marie/temp
```

A chaque utilisateur est associé un répertoire appelé "home directory". Lorsqu'un utilisateur veut effectuer une session de travail avec l'ordinateur, il se connecte au système avec la commande *login* et son répertoire courant est alors son "home directory". La notation "~" suivi du nom d'un utilisateur désigne le "home directory" de cet utilisateur (en C Shell uniquement). Ainsi pour aller dans le répertoire temp de marie, on aurait pu faire :

```
cd  ~marie
```

2.1. SECURITE DES FICHIERS

Chaque fichier possède 3 groupes d'attributs de protection: les bits de mode dont les valeurs ne peuvent être mises en place que par le propriétaire. Les 3 modes possibles sont permission ou non en lecture, en écriture et en exécution. Les 3 groupes sont celui de l'utilisateur, celui de son équipe de travail (ses collègues) et celui de l'ensemble des autres utilisateurs.

La permission rwxrwxrwx permet à n'importe qui de faire n'importe quoi sur le fichier ayant ces attributs de protection.

La permission r - x- - - - - - permettra au propriétaire uniquement de lire ou d'exécuter, mais pas d'écrire.

La commande *ls* avec l'option *-l* permet d'afficher les attributs des fichiers. La commande *chmod* permet de changer les attributs d'un fichier. Des attributs sont donnés par défaut à un fichier lors de sa création. Ceux-ci sont différents d'un système à l'autre, ils dépendent de l'administrateur Unix. La commande *umask* permet à un utilisateur de définir des attributs qui seront mis par défaut à la création de ses fichiers dans une session de travail. Exemple :

```
chmod 755 projet
chmod 440 projet.c
ls -l
drwxr-xr-x  1   pierre        512   Jan 25  11:15   Mail
-rwxrwx---  1   pierre      79450   Sep 14  20:15   projet
-rw-rw----  1   pierre      12385   Sep 14  20:14   projet.c
```

Le caractère "d" devant les attributs signifie la présence d'un répertoire.

LES CARACTERES JOKERS

Les caractères "*" et "?" peuvent être utilisés (dans certain cas) pour définir un motif de nom de fichiers. Le caractère "?" désigne n'importe quel caractère. Le caractère "*" désigne n'importe quelle suite de caractères. Exemples :

ls p*c affiche le nom de tous les fichiers commençant par la lettre p et finissant par la lettre c

ls a?? affiche les noms de fichiers constitués d'exactement 3 caractères commençant par la lettre a.

2.3. INDEPENDANCE DES ENTREES/SORTIES

Toutes les unités périphériques sont considérés comme des fichiers. Par opposition aux fichiers normaux (regular file) ils sont dits spéciaux. Leurs noms apparaissent dans le répertoire /dev. Par exemple, le fichier standard d'entrée est le clavier, le fichier standard de sortie est l'écran, le fichier de la mémoire est un "device" comme un autre, il en est de même pour les imprimantes et les modems.

3. Commandes et variables d'environnement

Pour dialoguer avec l'ordinateur, on utilise des phrases éventuellement réduites à un seul mot. Le premier mot de la phrase est appelé "verbe" ou "nom de la commande". En général, une phrase tient sur une seule ligne, c'est pourquoi on parle de "ligne commande". Il est possible de mettre une phrase sur plusieurs lignes en utilisant le caractères de continuation "\" (qui doit être impérativement le dernier caractère de la ligne et donc ne doit pas être suivi par des caractères "espace"). A l'opposé, une ligne peut contenir plusieurs commandes (phrases) qui doivent alors être séparées par le caractère ";" .

Les mots qui suivent le nom de la commande sont des paramètres utilisés pour définir le comportement de la commande. Les paramètres d'une commande peuvent être positionnels, introduits par des mots-clés ou encore tout simplement représentés une option de comportement.

Lorsqu'il y des paramètres positionnels, l'ordre positionnel doit être respecté. Exemple: la commande *cp* effectuant la duplication d'un fichier vers un autre fichier (existant ou non) possède 2 paramètres positionnels: le 1er indique le nom (du chemin) du fichier source et le 2nd le nom (du chemin) du fichier cible :

```
cp  ~marie/essai  essai_m
```

Le fichier *essai* du répertoire de marie est copié dans le répertoire courant. La copie a pour *nom essai_m*. Notons que lorsque le nom local du fichier cible est identique au nom local du fichier source, il est possible de seulement préciser le nom du répertoire d'accueil. Exemples:

```
cp  ~marie/essai.c  ~pierre
cp  ~marie/essai.c  .   (avec "." désignant le répertoire courant)
```

Dans certains cas, des mots-clés commençant par le caractère "-" permettent d'introduire les paramètres. Dans la commande *sed*, filtre d'édition, l'option -f permet de préciser le nom du fichier utilisé comme texte de requête en lieu et place de l'entrée standard. Dans la commande *find*, recherchant des fichiers dans une arborescence, l'option -name permet de préciser le nom (ou la forme du nom) des fichiers recherchés, l'option -user permet de limiter la recherche aux fichiers d'un utilisateur donné, l'option -size permet de préciser la taille des fichiers recherchés, etc.

Les options de comportement permettent de demander un mode particulier de fonctionnement de la commande. Ces dernières se distinguent sous Unix des paramètres positionnels car elles commencent avec le caractère "-". Exemple: la commande *ls* avec l'option *-l* permet d'afficher les noms des fichiers dans un format plus "long" donnant plus

de renseignement sur les fichiers: propriétaire, droits, taille, date de dernière modification ou de création.

Les commandes sont interprétées par une composante du système d'exploitation appelée Shell (interpréteur de commandes). Différents Shell existent: le Bourne Shell, le C Shell, le Korn Shell, etc. Dans le vocabulaire d'un Shell, on distingue les commandes internes, les commandes externes et les structures de contrôle.

Les commandes internes sont des verbes connus par l'interpréteur. C'est ce dernier qui lit les paramètres associés et exécute les instructions demandées par l'appel de la commande (exemple: *cd*).

Les commandes externes ne sont pas connues à priori de l'interpréteur. Il s'agit en fait de fichiers exécutables. L'interpréteur recherche le fichier de même nom associé à la commande dans une liste de répertoire connue et fixée (Cf. commande *path*). Les paramètres sont récupérés par le programme exécutable. Certaines commandes externes sont des programmes fournis avec le système d'exploitation Unix, on parle de commandes "système". D'autres ont été créées par les utilisateurs, on parle de commandes "utilisateurs".

Les structures de contrôle permettent de contrôler le flux d'exécution des commandes: choix selon un résultat d'exécuter une suite de commandes ou non, possibilité de répéter un traitement de manière répétitive ou itérative (if, for, while, ...).

3.1. LA REDIRECTION DES ENTREE ET SORTIE

La plupart des commandes Unix rendent l'information demandée vers le fichier standard de sortie, c'est à dire l'écran. Il est possible, grâce au signe > de rediriger la sortie d'une commande vers un fichier. Attention avec la redirection >, si le fichier représentant la sortie existe déjà, le contenu de ce dernier est perdu et remplacé par la sortie de la commande (on dit que le fichier est écrasé). Pour ajouter, cette sortie à l'information déjà existante, il faut utiliser la concaténation >> .

La commande *ls* > *liste* permet d'obtenir la liste des fichiers du répertoire courant dans le fichier *liste*. Ce fichier peut-être imprimé à l'aide de la commande *lpr*.

De même le signe < permet de redéfinir l'entrée standard d'une commande. La commande *spell* < *myfile* permet de corriger les fautes d'orthographe des mots provenant du fichier myfile.

On appelle filtre toute commande Unix, conçue pour exécuter une tâche bien définie, lisant des données sur l'entrée standard et produisant des résultats dans la sortie standard. Il est possible de connecter plusieurs filtres les uns derrière les autres à l'aide du caractère |. Une série de programmes ainsi connectés est appelé pipe (tube). Tous les programmes appelés s'exécutent simultanément!

Notons qu'il n'est pas possible de rediriger l'entrée et la sortie de toutes les commandes système ou utilisateurs.

3.2. LES VARIABLES D'ENVIRONNEMENT

Les variables d'environnement permettent en principe de mémoriser des informations utilisées pour définir l'environnement de l'utilisateur.

Une variable d'environnement est une variable informatique de type chaîne de caractères. Elle permet de mémoriser une valeur (chaîne de caractères). Chaque processus possède

un environnement qui lui est propre. Dans cet environnement, une seule valeur est associée à une variable informatique à un instant donné. Il est possible de changer cette valeur à tout moment par programmation.

Le nom d'une variable commence par une lettre. Il est composé de lettres, de chiffres ou de caractères soulignés. Le nom d'une variable ne doit être ni un nom réservé, ni un nom de commande interne ou externe. Pour cette raison et par convention, le nom d'une variable est souvent composé de caractères majuscules. La valeur d'une variable Shell est obtenue en faisant précéder son nom par le caractère "$". Dans l'exemple suivant, la commande *setenv* (C Shell) permet de donner une valeur à la variable d'environnement MAIL et la commande *echo* permet d'afficher la valeur de cette variable :

```
setenv  MAIL  /usr/spool/mail/pierre
echo  $MAIL
```

Lorsque la valeur donnée à une variable n'est pas un mot unique, mais une liste de mots séparés par des caractères blancs (espaces), on utilise les caractères "" (appelés quotes) pour encadrer cette liste de mots :

```
setenv  EXEMPLE  "ceci est un exemple de chaîne avec des espaces"
```

Une variable peut recevoir la sortie d'une commande sous forme de chaîne de caractères, pour cela toutes les lignes de la sortie de la commande sont réunies en une seule ligne. Exemple en Bourne Shell :

```
setenv  LISTE  `ls`
```

Lorsque l'interpréteur de commande rencontrent le caractère "`" (appelés backquote) dans une ligne de commande, il commence par évaluer ce qui se trouve entre les caractères backquote, puis lance l'exécution de la commande appelée. Exemple :

```
echo  `ls`
```

L'interpréteur de commande évalue la commande *ls*, génère la ligne commande suivante puis l'exécute :

```
echo  Mail  projet  projet.c  projet.o
```

Remarque: Lorsqu'un processus génère un processus secondaire appelé processus fils, l'environnement crée pour le fils est une copie de l'environnment du père. On dit que le fils hérite de l'environnement du père. Le processus fils peut créer ou modifier des variables d'environnement. Ceci n'a aucune incidence sur l'environnement du père.

4. Le Bourne Shell.

L'affectation d'une variable s'énonce avec une syntaxe différente du C Shell. On utilise l'opérateur d'affectation "=" :

```
LISTE=`ls`
echo  "liste des fichiers : " $LISTE
```

En Bourne Shell, un certain nombre de variables spéciales sont initialisées par l'interpréteur. La variable HOME permet de récupérer le nom du répertoire de login de l'utilisateur (home directory). La variable PATH indique au Shell la liste des répertoires dans lesquels il doit chercher les commandes externes.

La variable spéciale $? est modifiée après chaque commande et permet de connaître sous forme de caractères décimaux le statut de fonctionnement de la commande qui vient d'être appelée. La plupart des commandes retourne la valeur zéro lorsque qu'elles se terminent normalement et une valeur non nulle permettant d'identifier la nature du problème rencontré dans les autres cas. Le code de retour d'une commande peut être testé avec la structure de contrôle *if*.

4.1. Le passage des parametres dans un script en Bourne Shell

Une suite de commandes peut être enregistrée dans un fichier, appelé script. Celui-ci peut être appelé comme une commande externe par la suite. Dans un script, le caractère "#" en 1ère colonne permet d'insérer des commentaires qui ne seront pas interprétés. La première ligne d'un script est un commentaire spécial, introduit par les 2 caractères "#!", permettant de définir le Shell interpréteur des commandes. Exemple :

```
#!/bin/sh
# script my_ls en Bourne Shell
echo  "liste des entrées du répertoire `pwd`"
ls -l
```

Il est possible de passer des paramètres à un script. La valeur des paramètres est récupéré à l'intérieur du script à l'aide des variables $1 $2 $3 $4 $5 $6 $7 $8 $9 (attention, il n'y a pas de variable $10 en Bourne Shell). La variable $1 donne la valeur du premier paramètre positionnel, la variable $2 donne la valeur du 2ème, etc. La variable spéciale $* permet de récupérer le nombre de paramètres passés en argument de la commande et la variable $0 le nom de cette commande. Exemple :

```
#!/bin/sh
# script my_ls amélioré

if [ "$1" != "" ];  then;  cd $1  ;endif
echo  "liste des entrées du répertoire `pwd`"
ls -l
```

Remarque: Dans un Shell donné, on peut appeler des scripts écrits en Bourne Shell, en C Shell ou tout autre Shell. Le lancement d'un script crée un nouveau processus.

4.2. LES STRUCTURES DE CONTROLE DU BOURNE SHELL

La commande test

```
test expression
```

ou bien

```
[ expression ]
```

La condition

```
if command1 ;then
        command2
fi
```

L'alternative

```
if command1 ; then
        command2
elif command3
        command4
else
        commandn
fi
```

L'aiguillage

```
case value in
        pattern1)  command1;;
        pattern2)  command2;;
        pattern3)  command3;;
esac
```

La boucle

```
for variable in  liste
do
        command
done
```

```
while command1
do
        command2
done
```

```
until command1
do
        command2
done
```

La command *test* est en général utilisée avec les autres structures de contrôle. Les conditions *expression* les plus courantes sont :

-d *file*	vrai si *file* est un répertoire (directory)
-f *file*	vrai si *file* est un fichier existant
-r *file*	vrai si *file* est un fichier que l'on peut lire
-s *file*	vrai si *file* est de taille (size) non nulle
"s1" = "s2"	vrai si les 2 chaînes sont identiques
"s1" != "s2"	vrai si les 2 chaînes ne sont pas identiques
n1 -eq n2	vrai si les nombres sont identiques
n1 -ne n2	vrai si les nombres sont différents
n1 -gt n2	vrai si n1 plus grand que n2
n1 -ge n2	vrai si n1 plus grand ou égal à n2
n1 -lt n2	vrai si n1 plus petit que n2
n1 -le n2	vrai si n1 plus petit ou égal à n2
! *expression*	vrai si *expression* est faux

4.3. LES FICHIERS STANDARDS EN BOURNE SHELL

Le fichier /etc/profile est le script du système automatiquement exécuté au démarrage d'une session de travail (juste après le "login").

Le fichier $HOME/.profile est le script utilisateur automatiquement exécuté après le script système au démarrage d'une session de travail.

5. Le C Shell.

Le C Shell est un interpréteur plus interactif que le Bourne Shell. En général, on préfère travailler en C Shell interactivement et écrire ses scripts en Bourne Shell.

Le C Shell possède un mécanisme d'historique des commandes entrées par l'utilisateur. La commande *history* permet d'afficher les dernières commandes appelées. Il est possible grâce au caractère "!" de rappeler, modifier et re-exécuter une commande déjà appelée. Le caractère "!" doit être suivi des premières lettres de la commande à rappeler. La commande *!!* permet de rappeler la toute dernière commande appelée.

5.1. LES STRUCTURES DE CONTROLE DU C SHELL

Le C Shell ne possède pas de commande *test* comme le Bourne Shell.

La condition

```
if (expression) command
```

L'alternative

```
if (expression) then
        command_1
else
        command_2
endif
```

L'aiguillage

```
switch (string)
case pattern1:
        command1
        [breaksw]
case pattern2:
        command2
        [breaksw]
default:
        command2
endsw
```

La boucle

```
foreach variable (wordlist)
        command
end

while (expression)
        command
end
```

En C Shell, les conditions *expression* les plus courantes sont :

-d *file*	vrai si *file* est un répertoire (directory)
-f *file*	vrai si *file* est un fichier existant
-r *file*	vrai si *file* est un fichier que l'on peut lire
-z *file*	vrai si *file* est de taille zéro

"s1" == "s2"	vrai si les 2 chaînes sont identiques
"s1" !== "s2"	vrai si les 2 chaînes ne sont pas identiques

plus généralement

x1 == x2	vrai si les opérandes sont identiques
x1 !== x2	vrai si les opérandes sont différents
x1 > x2	vrai si x1 plus grand que x2
x1 >= x2	vrai si x1 plus grand ou égal à x2
x1 < x2	vrai si x1 plus petit que x2
x1 <= x2	vrai si x1 plus petit ou égal à x2

! expression	vrai si *expression* est faux

Cette dernière écriture est utilisé dans une structure *while (! expression)* pour programmer l'équivalent de la structure *until* du Bourne Shell non existante en C Shell

5.2. LES FICHIERS STANDARDS EN C SHELL

Le fichier /etc/cshrc est le script du système automatiquement exécuté au démarrage d'une session de travail.

Le fichier ~./.login est le script utilisateur automatiquement exécuté après le script système au démarrage d'une session de travail.

Le fichier ~/.cshrc est le script utilisateur exécuté avant le lancement de tout interpréteur de commande ou de tout processus dans une session de travail.

Le fichier ~./logout est le script utilisateur automatiquement exécuté à la fin d'une session de travail.

6. COMMANDES UNIX usuelles

at	exécution de commandes à une date-heure précise
awk	langage de recherche et de traitement par modèle
basename	extraction du nom local d'un fichier
cat	concaténation de fichiers
cd	changement de répertoire
chmod	modification des permissions d'un fichier
cmp	comparaison de 2 fichiers
cp	duplication de fichier
cpio	archivage de fichiers
cut	suppression de champs d'un fichier texte
dd	conversion et copie d'un fichier
diff	comparaison de 2 fichiers
dirname	extraction d'un nom de chemin de répertoire
df	espace disque libre d'un système de fichier
du	sommaire d'utilisation du disque dur
echo	affichage à l'écran des arguments
expr	évaluation d'une expression
file	détermination du type d'un fichier
find	recherche des fichiers dans une arborescence
getopt	analyse d'options dans une commande
grep	recherche de lignes d'après un modèle donné
head	affichage du début d'un fichier
id	détermination de l'identificateur utilisateur
kill	liquidation d'un process
lint	vérification de la syntaxe d'un programme C
logname	détermination du nom de login
lpq	liste des impressions en attente (dans le spool)
lpr	envoi d'une impression dans la file d'impression
lprm	suppression d'une impression en attente
lpstat	information sur l'état des imprimantes
ls	affichage des entrées d'un répertoire
mail	envoi ou lecture du courrier électronique
make	fabrication d'un exécutable
man	manuel "en ligne" des commandes
mkdir	création d'un répertoire
mv	déplacement d'un fichier (ou renommage)
od	affichage interprété d'un fichier binaire (octal dump)
passwd	changement du mot de passe
pg	pagination d'un fichier sur écran
ps	état des processus actifs

pwd	affichage du répertoire courant
rm	suppression de fichiers
sed	éditeur d'un flot de données
shift	décalage de la numérotation des paramètres
size	taille d'un fichier
sleep	suspension du temps d'exécution
sort	tri des lignes d'un fichier
spell	correcteur de faute d'orthographe
stty	gestion du terminal
tail	affichage de la fin d'un fichier
time	information sur le temps d'exécution d'une commande
touch	mise à jour des temps d'accès d'un fichier
umask	définition du masque des permissions
uucp	copie de fichier Unix vers Unix
vi	éditeur de texte
wc	comptage des lignes d'un fichier texte
where	recherche d'une commande dans le "path"
who	liste des utilisateurs actifs
write	envoi de message à un utilisateur

Pour connaître la syntaxe et le comportement d'un commande, on peut utiliser le manuel en ligne qui se présente sous la forme de la commande *man*. Exemple:

```
man ls
```

permet d'obtenir des renseignements sur la commande *ls*.