Université de Picardie Jules Verne
Ecole d'Ingénieurs Jules Verne
Filière Cybersécurité

Système Unix

C. Drocourt

cyril.drocourt@u-picardie.fr

SOMMAIRE

Chapitre 1 - Prise en main et première exploration

Table des matières

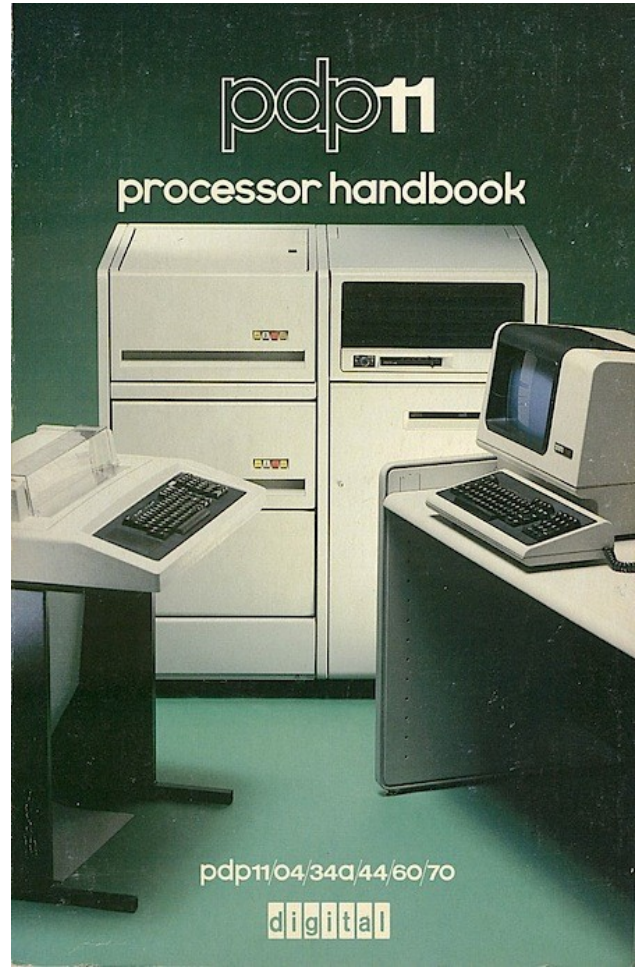
1 - L'histoire d'Unix

1969 : Début du projet Unics (qui devint Unix) dans les laboratoires Bell Labs d'AT&T fondé sur les avantages et les inconvénients de Multics par Ken Thompson et Dennis Ritchie essentiellement en Assembleur, sur PDP-7.

Sources :
http://idgnow.uol.com.br/
http://hawaii.ls.fi.upm.es/
http://fr.wikipedia.org/wiki/UNIX

1970 : Sortie du PDP-11 incompatible avec le PDP-7 d'où l'idée d'un langage de plus haut niveau comme le FORTRAN ou le langage B de Ken Thompson.

Malheureusement ces derniers étaient inadaptés...



Sources :

http://idgnow.uol.com.br/

http://hawaii.ls.fi.upm.es/

http://fr.wikipedia.org/wiki/UNIX

1971 : Définition du langage C par Dennis Ritchie à partir du langage B, lui même issu du langage BCPL (*Basic Combined Programming Language*), et début de réécriture d'Unix. Le problème du langage B est qu'il n'avait qu'un seul type de données.

1973 : - Fin de la réécriture d'Unix en langage C,
- Publication de l'article « The UNIX time-sharing system » (Proceedings of the fourth ACM symposium on Operating system principles, p 27),
- Utilisation d'Unix dans des projets internes,
- Distribution du code source d'Unix dans les universités à des fins éducatives.

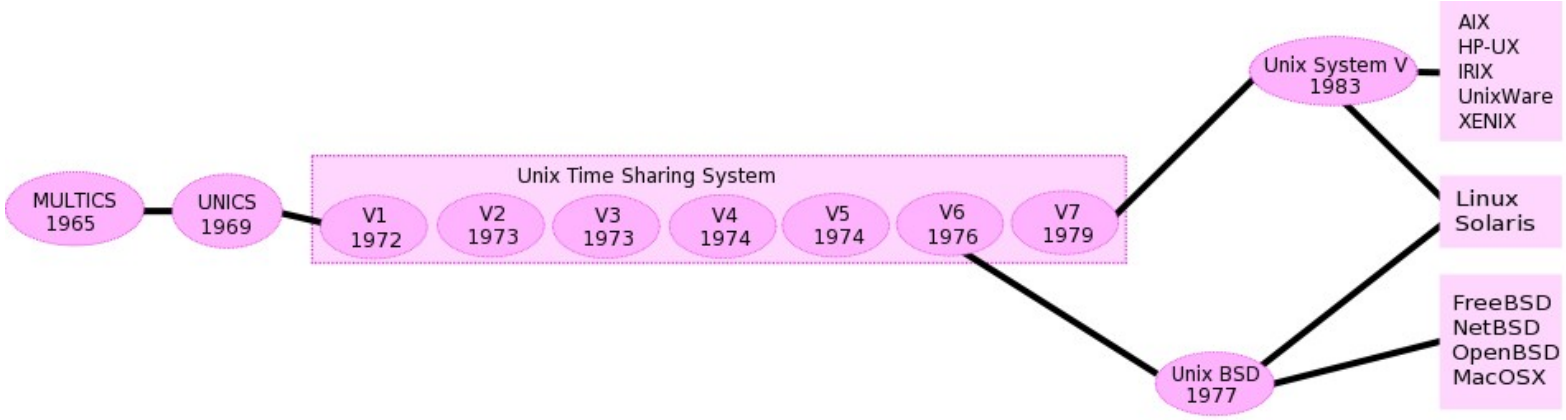
Sources :
http://www.cam.org/~ycd/ch-1.html
http://www.framablog.org/
http://fr.wikipedia.org/wiki/UNIX

1976 : Sortie d'Unix v6 par AT&T.

1977 : Berkeley Software Distribution (BSD) de l'université de Californie.

1979 : Commercialisation d'Unix v7 par AT&T, paiement obligatoire d'une licence pour obtenir le code source.

1983 : Publication d'Unix System V par AT&T.



Sources :

http://projet.unix.free.fr/historique.htm

http://fr.wikipedia.org/wiki/UNIX

2 - Les systèmes Unix

Parmi les Unix propriétaires les plus utilisés aujourd'hui :

- AIX d'IBM (1986),
- HP-UX de Hewlett-Packard (1982),
- Solaris (SunOS) de Sun Microsystems (1981),

Les autres sont :

- XENIX de Microsoft (1980),
- Ultrix de DEC (1991),
- IRIX de SGI (1987),
- UnixWare de Novell,

Les BSD (Berkeley Software Distribution) :

- 3BSD (1979),
- 4BSD (1980),
- NetBSD (1993),
- OpenBSD (1995),
- FreeBSD (1993),
- SunOS (1981),
- NexStep (1989),
- MacOSX (1998),

Les Autres :

- Linux (1991) qui sert de base à Android,
- Minix (1987),
- GNU/Hurd (1990),

3 - Le monde GNU/Linux

3.1 - Le projet GNU

Le **projet GNU** débute en **1983** sous l'initiative de **Richard Stallman** (RMS), qui travaille alors au MIT. GNU est un acronyme qui signifie « **G**NU's **N**ot **U**NIX ».

L'idée du projet est de laisser la possibilité à chaque utilisateur d'accéder aux sources des logiciels du système, d'où la création de la **FSF** (Free Software Fondation) en 1985 pour y adosser le projet, puis de la licence **GPL** en 1989.

En 1990, le système dispose de tous les éléments nécessaires (le célèbre compilateur « **gcc** », des « shells », des éditeurs, les librairies systèmes, ...) hormis l'un des éléments principal : **Le noyau**.

Le projet de développement du noyau « **GNU Hurd** » voit alors le jour, mais son développement basé sur le concept du micro-noyau ne progresse pas suffisamment rapidement.

3.2 - L'origine de Linux

Linux est le noyau d'un système d'exploitation qui débuta en 1991 sous l'impulsion de Linus Torvalds alors étudiant en informatique Finlandais. Son but était d'avoir un système au fonctionnement proche de Minix, autre « clone » d'Unix de l'époque, auquel le concepteur Andrew Tanenbaum, refusait d'y intégrer des modifications.

Tout commence avec un courrier électronique envoyé le 25 Aout 1991 par Linus Torvalds sur la liste newsgroups « comp.os.minix » :

```
From:        torvalds@klaava.Helsinki.FI   (Linus   Benedict
Torvalds)
Newsgroups: comp.os.minix
Subject:  What would you like to see most in minix?
Date:   25 Aug 91 20:57:08 GMT
Hello everybody out there using minix -
...
```

```
Salut à tous les utilisateurs de Minix,

Je suis en train de réaliser un système d'exploitation
(gratuit), (c'est juste un passe-temps, il ne sera pas
important et professionnel comme le Gnu) pour les clones
d'AT 386 (ou 486). Il mijote depuis le mois d'avril, et
commence à être au point. J'aimerais des remarques sur ce
que les gens aiment ou non dans Minix, car mon système lui
ressemble un peu (même organisation du système de fichiers,
pour des raisons pratiques, entre autres).

J'y ai porté Bash (1.08) et Gcc (1.40), et tout semble
fonctionner. Je devrais donc disposer de quelque chose
d'utilisable dans les mois à venir, et j'aimerais connaître
les fonctionnalités que la plupart d'entre vous aimeraient.
Toutes les suggestions sont les bienvenues, mais je ne
promets pas de toutes les implémenter ;-)

Linus (torvalds@kruuna.helsinki.fi)
```

Evolution du noyau :

1991 : 0.0.1

1994 : 1.0, première version stable.

1995 : 1.2, modules chargeables.

1996 : 2.0, ajout d'architectures, multiprocesseur, apparition de Tux, ...

1999 : 2.2, IPv6, ...

2001 : 2.4, USB, ...

2003 : 2.6 *(2.6.0 le 18/12/2003 -> 2.6.39 le 19/05/2011)*

2011 : 3.0 *(3.0 le 05/08/2011 -> 3.19 le 09/02/2015)*

2015 : 4.0 *(4.0 le 12/04/2015 -> 4.20 le 24/12/2018)*

2019 : 5.0 *(5.0 le 04/03/2019 -> 5.4 le 25/11/2019)*

3.3 - Les distributions

Linux n'étant que le **noyau** d'un système d'exploitation, il ne peut être utilisé seul, mais nécessite des programmes annexes, comme le Shell, l'interface graphique, les logiciels, ... L'ensemble étant alors appelé une distribution.

Le noyau Linux s'est alors naturellement associé avec des logiciels issus du système GNU, les distributions sont alors appelées **GNU/Linux**.

Il existe une multitude de distributions Linux, en fonction de l'architecture, de l'orientation d'utilisation (serveur, client, firewall, embarqué, ...), du type de licence utilisée, ...

Cependant, les distributions essentielles en terme d'utilisation et de parts de marché peuvent être énumérées :

- **Slackware** : L'une des plus ancienne, qui date de 1993, est qui est toujours en activité aujourd'hui,
- **Debian** : Dont le projet fut lancé en 1993 pour fournir un système d'exploitation composé uniquement de logiciels libres, la première version stable sortie en 1996.
- **RedHat** : Datant de 1995, et que introduisit les paquets « rpm » permettant de gérer les dépendances,
- **Suse** : Distribution d'origine Allemande datant de 1994, historiquement basée sur Slackware mais reposant maintenant sur des paquets « rpm », connue pour son outil de configuration Yast, racheté par Novell en 2004,
- **Mandriva** : Anciennement Mandrake, distribution d'origine Française fondée en 1998 sur une base de RedHat 5.1,
- **Ubuntu** : Distribution populaire initiée en 2004 et basée sur Debian et financée par la société Canonical.
- **CentOS** : Version libre de RHEL initiée en 2004, soutenue officiellement par RedHat depuis le 7 Janvier 2014,

Et encore Fedora, Mint, Gentoo, IPCop, ...

4 - logiciels libres

Linux est distribué sous licence GPL, qui est une licence de logiciels libres. Ce type de licence ne signifie pas gratuité et il est important de faire cette distinction.

Logiciel Libre signifie que d'après la Free Software Fondation :

- **Liberté 0** : La liberté d'exécuter le programme, pour tous les usages ;
- **Liberté 1** : La liberté d'étudier le fonctionnement du programme, ce qui suppose l'accès au code source,
- **Liberté 2** : La liberté de redistribuer des copies, ce qui comprend la liberté de vendre des copies,
- **Liberté 3** : La liberté d'améliorer le programme et de publier ses améliorations, ce qui suppose, là encore, l'accès au code source,

Il existe par conséquent plusieurs licences de logiciels libres, la GPL étant l'une d'entre elles, on peut citer parmi les plus connues :

- La licence BSD,
- La licence Apache,
- Mozilla Public Licence,
- ...

La grande différence entre ces licences étant la liberté laissé à l'utilisateur de redistribuer le logiciel.

Remarque importante *:*

.....Cette version du support de cours est normalement adaptée aux distributions RedHat/CentOS en version 6.X et 7.X, et Debian en version 7.X et 8.X. Cependant, lorsque d'importantes différences existent avec les versions précédentes, celles-ci seront précisées en remarque.

5 - SUS/POSIX

Le sigle **SUS** (Single UNIX Specification) désigne un ensemble de spécification que se doit de respecter un système Unix. Il est développé et maintenu par l'Austin Group. Le travail a débuté en 1980

La norme **POSIX** (qui signifie "Portable Operating System Interface", le X venant d'Unix) reprend les spécifications de SUS et est définit en 1988. Elle correspond à la norme IEEE 1003. Cette norme définit le standard que doit respecter un système souhaitant être compatible Unix.

Un système d'exploitation peut supporter un niveau de spécification de la norme, ou ne supporter qu'une partie d'un niveau. Par exemple, Microsoft Windows respecte une partie de la norme POSIX.1.

Il existe plusieurs niveaux de spécification :

- **POSIX.1** (1988 - IEEE 1003.1), interfaces de programmation,
- **POSIX.1b** (1993 - IEEE 1003.1b), extensions pour le temps réel,
- **POSIX.1c** (1995 - IEEE 1003.1c), extensions pour les « threads »,
- **POSIX.2** (1992 - IEEE 1003.2) , « Shell » et utilitaires,

En parallèle à la norme POSIX, et compte tenu du fait que les spécifications de la norme POSIX sont payants, une nouvelle version de SUS apparait en 1997 sous le nom SUSv2 ou UNIX98.

Finalement, les spécifications SUS et la norme POSIX fusionnent à partir de 2001 :

- **POSIX:2001** (IEEE 1003.1-2001), Single UNIX Specification version 3, UNIX03,
- **POSIX:2004** (IEEE 1003.1-2004), Mise à jour,
- **POSIX:2008** (IEEE 1003.1-2008), Single UNIX Specification version 4,

6 - Avenir

L'avenir des Unix propriétaires s'avèrent incertain pour plusieurs raisons :

- Selon Gartner en 2019, 1 serveur vendu sur 85 utilise un Unix Solaris, AIX ou HP-UX, soit 1,18 %,
- Oracle à déjà annoncé qu'il n'y aurait pas de version 12 de Solaris,
- IBM vient de racheter RedHat qui vent plus de Linux que IBM de AIX,

Cependant, même si les demandes vont baisser, on considère qu'il faudra plus de 20 ans pour ne plus avoir de bases installés car les derniers secteurs sont soit critiques, soit onéreux à migrer.

7 - Interface Graphique

Sous Unix, il existe plusieurs niveau à l'interface graphique, pour simplifier nous avons :

- **Le serveur X**, qui a en charge l'affichage des objets graphiques simples, et qui gère les entrées sorties comme la carte graphique, la souris, le clavier, ...
- **Le gestionnaire de fenêtres** (Window Manager), qui s'occupe de gérer les différentes fenêtres associées aux applications, il en existe une multitude sous Linux comme twm, fvwm, kwm, metacity, ...
- **L'environnement**, qui va prendre en charge les éléments évolués comme le copier/coller, un gestionnaire de fichiers, ... Sous Linux, les environnements les plus utilisés sont Gnome et KDE (parfois XFCE).

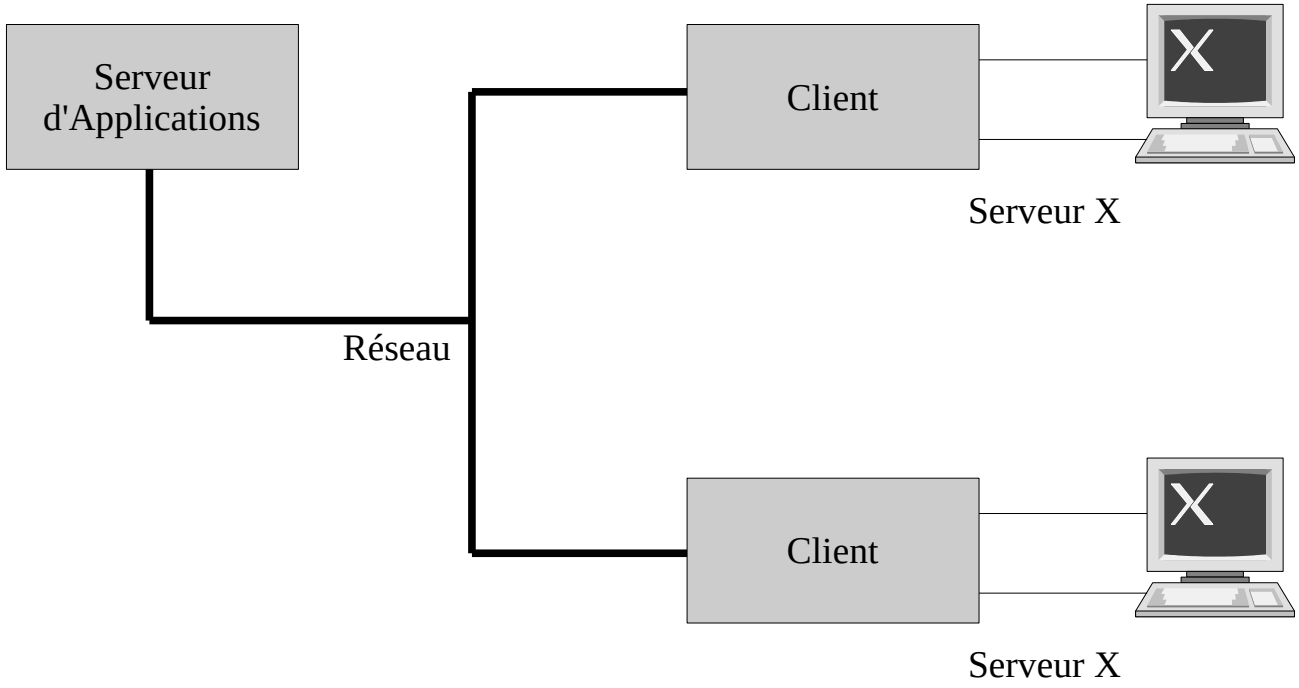
7.1 - X-Window

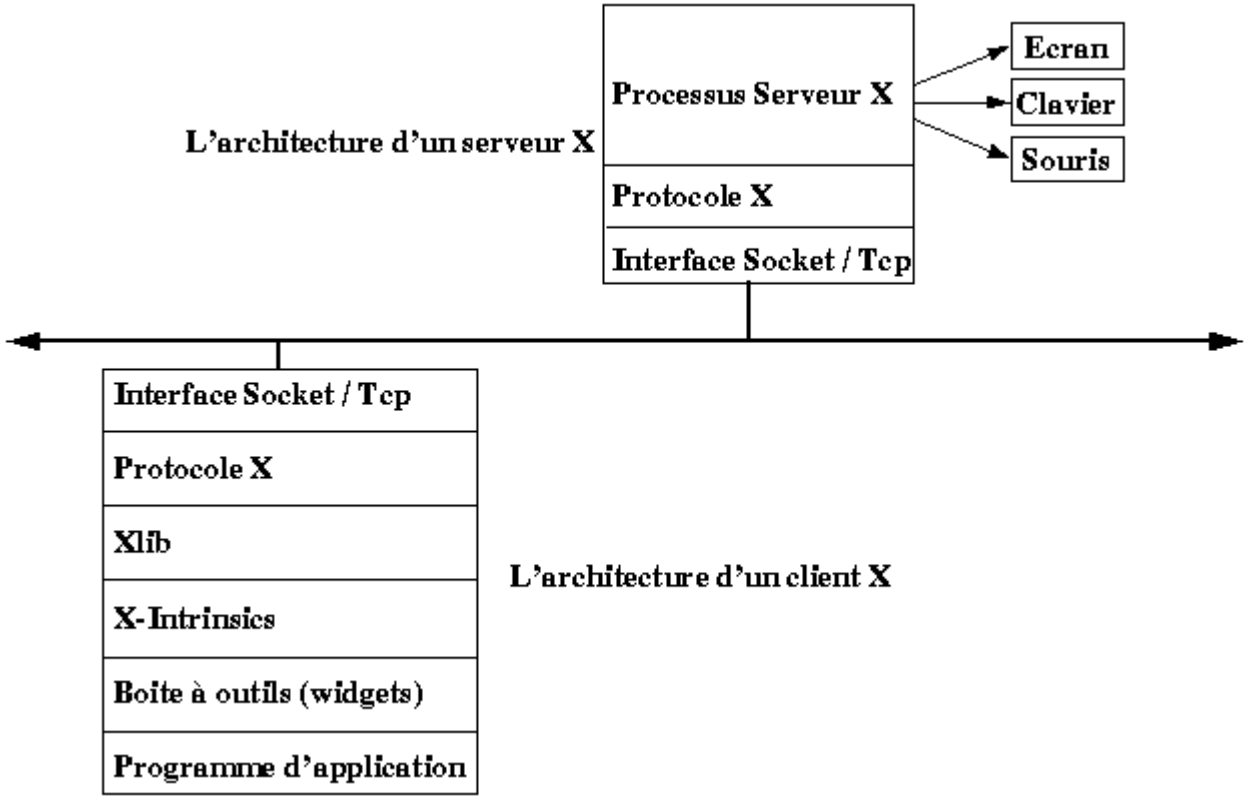
X-Window est une interface graphique multi-fenêtre distribuée, on l'appelle souvent X. X-Window est issu de travaux réalisés au Massachusetts Institute of Technology (MIT) dans le milieu des années 80. L'objectif initial était de créer un système permettant de travailler sur des machines hétérogènes en réseau tout en disposant d'une interface utilisateur graphique commune.

Très rapidement le monde industriel s'est rendu compte de l'intérêt de X et un consortium unissant de gros constructeurs informatiques (DEC, HP, IBM, SUN,...) et le MIT a été créé. Le consortium gère l'évolution de X et garantit sa pérennité

X-Window est construit autour d'un modèle client/serveur. Un client X est un demandeur de ressources graphiques ; c'est un programme d'application qui se déroule quelque part sur une machine du réseau. Un serveur X assure la gestion des ressources

graphiques d'un poste de travail et les interactions avec les clients X (gestion de l'écran, du clavier et de la souris).




L'architecture d'un serveur X
Processus Serveur X
Protocole X
Interface Socket / Tcp
Ecran
Clavier
Souris
Interface Socket / Tcp
Protocole X
Xlib
X-Intrinsics
Boite à outils (widgets)
Programme d'application
L'architecture d'un client X

7.2 - XLib

La Xlib est une interface de programmation du protocole X, elle est écrite en langage C. Les appels à la Xlib génèrent du protocole X. La Xlib fait partie du standard garanti par le consortium, elle est installée sous le nom de libX11.a.

7.3 - Les Toolkit

La manipulation de fenêtres par la Xlib est relativement fastidieuse pour le programmeur. Un niveau supérieur, destiné à faciliter la programmation d'applications a donc été créé : X Toolkit Intrinsics. Cette bibliothèque, connue sous le nom d' Intrinsics ou Xt, introduit la notion de widgets. Un widget est une structure de données accompagnant une fenêtre permettant ainsi de qualifier aisément certains paramètres (taille, fond, contours ...). Une hiérarchie des widgets est introduite par Xt, ainsi, en détruisant une fenêtre, on peut d'un seul coup faire disparaître toute sa descendance. La bibliothèque Xt est installée sous le nom de libXt.a et fait partie du standard garanti par le consortium.

7.4 - Les widgets

Un dernier niveau de bibliothèque, appelé boîte à outils, est disponible pour le programmeur. L'idée est d'aller toujours plus loin dans les facilités offertes pour la programmation, des widgets taillés sur mesure pour tel ou tel type d'applications apparaissent. Les boîtes à outils les plus connues sont les Athena's Widgets (Xaw, incluses dans la livraison de base du MIT) et Motif de l' OSF (Xm, licence payante nécessaire) dans le monde industriel. L'une comme l'autre définissent des widgets de type bouton de commande, boîte de dialogue, barre de défilement ...

7.5 - Toolkits modernes

Aujourd'hui, des librairies plus modernes remplacent les couches Xt/Widgets dont les deux plus connus sont :

- **QT** : TroolTech puis Nokia (1995),
- **GTK** : GNOME Fondation (1998),

7.6 - Gestionnaire de fenêtre

Le serveur X et la Xlib n'intègrent pas des fonctionnalités liées à la gestion des fenêtres, c'est donc le rôle d'une application spécifique :

- `uwm` (Ultrix Window Manager) en 1985 (Propriétaire),
- `twm` (Tab Window Manager) en 1987 (OpenSource),
- `mwm` (Motif Window Manager) en 1991,
- `Metacity` : Pour GNOME,
- `KWM` puis `KWIN` pour KDE,

7.7 - Environnement

Les environnements les plus utilisés sont :

- **CDE** (Common Desktopn Environment) : Lancé en 1993 par le Consortium X/Open, basé sur les widgets « Motif » sous l'égide de Sun, HP et IBM. Licence Propriétaire jusqu'en 2012..
- **KDE** (K Desktop Environment) ; Lancé en 1996 et basé sur le toolkit Qt non libre. Licence Libre GPL.
- **Gnome** (GNU Network Object Model Environment) : Projet GNU initié en 1997 en réponse à KDE et basé sur le toolkit GTK (Libre et créé pour le logiciel Gimp). Licence libre GPL.
- Autres : XFCE, LXDE, ...

7.8 - Compatibilités des environnements

Un projet appelé « freedesktop » tente d'uniformiser un certain nombre d'éléments associés à l'interface graphique, pour n'avoir à réaliser qu'une seule configuration, quelque-soit l'environnement choisi.

Par exemple, le bureau de l'utilisateur est un répertoire nommé « Desktop » ou « Bureau », dans le répertoire personnel de l'utilisateur.

8 - Gestion des utilisateurs

8.1 - Utilisateurs et groupes

Sur les systèmes Unix, chaque utilisateur est identifié par un nom de connexion, appelé « `login` ». Ce « `login` » permet de trouver une correspondance unique dans le système de l'utilisateur, basé sur un numéro appelé « `UID` ».

De la même manière, chaque utilisateur appartient au moins à un groupe appelé « Groupe principal », mais peut également appartenir à d'autres groupes appelés « Groupes secondaires ». Ces derniers sont identifiés de manière unique sur le système par un numéro appelé « `GID` ».

Un utilisateur peut obtenir des informations sur son `UID` ainsi que sur les « `GID` » des groupes auxquels il appartient à l'aide de la commande « `id` » :

```
[rahan@drocourt ~]$ id
uid=500(rahan) gid=500(rahan) groupes=500(rahan),501(vboxusers)
```

Ainsi, chaque utilisateur est identifié sur le système par un UID unique, dont la valeur n'a en théorie pas d'importance au niveau système hormis le nombre « 0 ».

En effet, sous Unix la valeur « **0** » pour le champ « **UID** » désigne l'administrateur du système appelé « **root** » qui possède tous les droits.

De plus, il est courant sur les systèmes Unix de réserver les « UID » inférieures à certaines valeurs pour les comptes systèmes et les services (par exemple 1000 sous Linux).

8.2 - Changer de mot de passe

La commande commune pour à quasiment tous les Unix pour changer un mot de passe est la commande « passwd ». Cette dernière commande invoquée sans argument modifie le mot de passe de l'utilisateur courant :

```
[rahan@drocourt ~]# passwd
Changement de mot de passe pour l'utilisateur rahan.
Nouveau mot de passe :
Retapez le nouveau mot de passe :
passwd : mise à jour réussie de tous les jetons d'authentification.
```

9 - Introduction au système

9.1 - Connexion

Le système Unix permet plusieurs types de connexions :

- Nativement sur la console de la machine en mode texte,
- A distance en mode texte (ssh ou autre),
- Sur la console en mode graphique,
- A distance en mode graphique (X11),

Le plus habituel sous Unix et surtout sur un serveur est l'utilisation d'une connexion textuelle (ou console) qui reste historique, et pour laquelle tous les outils sont disponibles.

Pour se déconnecter d'une connexion textuelle, il suffit de taper la commande « exit » ou « logout », ou simplement d'effectuer le raccourci <ctrl><d>.

9.2 - Rappel sur le manuel

Il existe sous « Unix » un manuel en ligne, qui permet d'obtenir de l'aide sur différents éléments du système, la forme générale est :

```
man <objet>
```

Cependant, un « objet » spécifique peut référencer plusieurs possibilités dans le système, il existe donc des sections numérotées :

- 1 : Commandes utilisateur ,
- 2 : Appels système
- 3 : Fonctions de bibliothèque C
- 4 : Périphériques et fichiers spéciaux
- 5 : Formats de fichier et conventions
- 6 : Jeux
- 7 : Divers
- 8 : Outils d'administration système et démons

Il est donc possible de spécifier la section en premier argument le numéro de la section avant de spécifier l'objet sur lequel il porte, ainsi pour demander de l'aide sur la commande « crontab » :

```
[root@drocourt ~]# man 1 crontab
```

ou simplement (choix de la plus petite section par défaut) :

```
[root@drocourt ~]# man crontab
```

Alors que pour obtenir de l'aide sur le fichier « crontab » :

```
[root@drocourt ~]# man 5 crontab
```

Pour sortir du manuel, il suffit de presser la touche « q ».

9.3 - Lister les fichiers

Pour afficher la liste des fichiers et répertoires (catalogues ou directory) on utilise la commande **ls**, exemple :

```
[cd@drocourt ~]$ ls
file1  file2  rep1
```

La commande « ls » admet certaines options :

- **-a** : tous les fichiers, y compris les fichiers cachés (fichiers dont le nom commence par un point ".")
- **-F** : identifie les fichiers en ajoutant "/" aux noms de dir., "*" aux exécutables, "@" aux liens symboliques
- **-R** : liste en parcourant récursivement tous les sous-répertoires
- **-i** : indique le i-nombre

```
[cd@drocourt ~]$ ls -aF
./  ../  .cache/  .config  file1.txt
```

- **`-ld`** *répertoire* : affiche informations sur les répertoires spécifiés (et non pas sur leur contenu)
- **`-l`** : listage long, cf ci-dessous :

```
-rw-r--r--  1 root root       512  8 sept. 17:26 ess
1<---2--->  3 <4-> <5->       <6>  <------7----> <-8->
```

  - 1 : type de fichier
    - \- : Fichier ordinaire
    - b : Fichier spécial en mode bloc
    - c : Fichier spécial en mode caractères
    - d : Répertoire
    - s : Socket
    - l : Lien symbolique
    - p : Tube nommé

- **2** : droits d'accès
- **3** : Nombre de lien sur ce fichier
- **4** : Le nom du propriétaire du fichier
- **5** : Le nom du groupe auquel appartient ce fichier
- **6** : La taille du fichier,
- **7** : La date et éventuellement l'heure de dernière modification (quand il s'agit de l'année actuelle),
- **8** : Le nom du fichier

9.4 - Organisation du systèmes

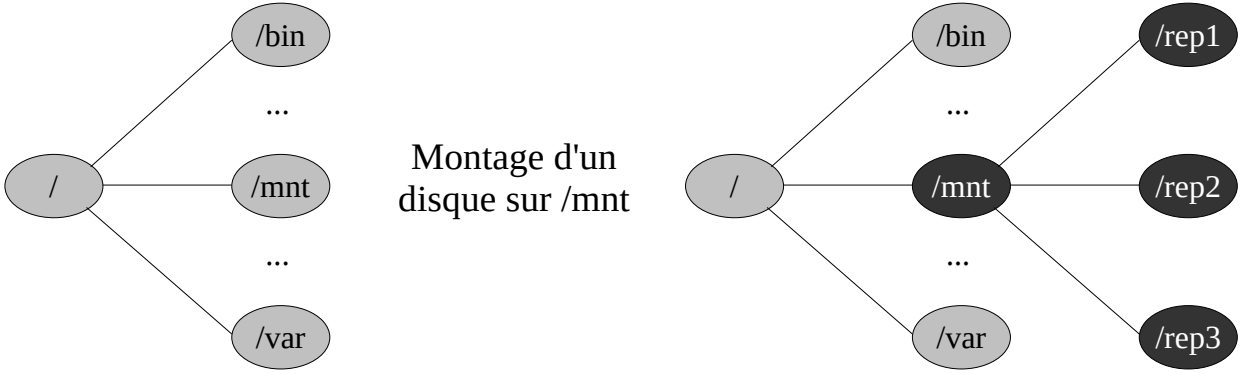
Sous Unix, il faut noter plusieurs spécificités :

- Les répertoires sont notés « / » (et non « \ » comme sous Windows),
- Il n'y a pas de notion de lecteur « C : » ou autre,
- La racine du système est donc directement le « / »,
- Un périphérique de type disque n'est en général pas disponible immédiatement une fois connecté au système,
- le « . » représente le répertoire lui même (adopté aussi par Windows),
- le « .. » représente le répertoire parent (adopté aussi par Windows),

9.5 - Montage

Sous Linux, contrairement à d'autres systèmes d'exploitation, un périphérique de type « unité de stockage » n'est pas disponible automatiquement et immédiatement. En effet, il est nécessaire de « monter » ce dernier dans une partie de l'arborescence.

Ceci signifie que cette partie de l'arborescence, après montage, sera le contenu du support monté. Les éventuels programmes et répertoires présents dans le point de montage avant l'opération ne seront plus accessibles jusqu'à l'opération de « démontage ».



9.6 - Commande « mount »

La commande historique pour connaître les périphériques de type systèmes de fichiers utilisés par le système est « mount », qui utilisée sans arguments, permet également de visualiser les caractéristiques d'utilisation (type, propriétaire, ...) :

```
[root@drocourt ~]# mount
/dev/sda3 on / type ext3 (rw)
none on /proc type proc (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda1 on /boot type ext2 (rw)
/dev/sda6 on /home/users-old type ext3 (rw)
none on /dev/shm type tmpfs (rw)
/dev/sda2 on /var type ext3 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
10.16.7.6:/home/nfs/morglum/http on /home/iut/http type nfs
(rw,addr=10.16.7.6)
10.16.7.6:/home/nfs/morglum/users on /home/users type nfs
(rw,addr=10.16.7.6)
```

9.7 - Utilisation de « mount »

Pour utiliser un périphérique on utilise la commande « mount » :

```
mount [options] device directory
```

Le paramètre « device » précise le nom du périphérique, en général de la forme « /dev/périphérique », et le « directory » spécifie le point de montage, comme « /mnt ».

Les options les plus courantes sont les suivantes :

- **-t <type>** : précise le type du système de fichier (vfat, ext2, iso9660, ...),
- **-r** : monter le système de fichier en lecture seule,
- **-w** : monter le système de fichier en lecture/écriture,
- **-l** : option utilisée seule (sans device, ni répertoire), et permet d'afficher les informations de « label » des différents disques,
- **-a** : monter l'ensemble des partitions systèmes (/etc/fstab),
- **-o <options>** : précise des options supplémentaires,

Par exemple, pour « monter » le périphérique CDROM dans le répertoire « /mnt/cdrom », on utilisera la commande suivante :

```
[root@drocourt ~]# mount -t iso9660 /dev/cdrom /mnt/cdrom
```

L'option « -o » permet de préciser des paramètres supplémentaires.

Démontage d'un système de fichiers

Pour démonter un système de fichiers, on utilisera la commande « umount » :

```
[root@drocourt ~]# umount /mnt/cdrom
```

9.8 - Le fichier /etc/fstab

Sous Linux (et BSD), un fichier est responsable des différents systèmes de fichiers utilisés par le système, et se nomme « /etc/fstab ». Chaque ligne de ce fichier correspond à un périphérique, voici un exemple de ce fichier :

```
/dev/sda3    /        ext3    defaults     0   1
/dev/sda1    /boot    ext3    defaults     0   2
/dev/sda2    none     swap    sw           0   0
```

La valeur des différents champs est la suivante :

- Le premier indique le périphérique (par son chemin, son label ou son UUID),
- Le deuxième indique le point de montage,
- Le troisième indique le type,
- Le quatrième indique les options,
- Le cinquième est un paramètre de la commande « dump »,
- Le sixième est un paramètre de « fsck »,

La même chose peut être retrouvée avec d'autres Unix mais avec des noms de fichiers différents :

- « `/etc/vfstab` » sour Solaris
- « `/etc/filesystems` » sour AIX

9.9 - Devenir administrateur

Il existe essentiellement deux solutions pour gérer les accès administrateurs :

- Utiliser la commande « su »,
- Utiliser la commande « sudo »,

Commande « su »

Elle permet de basculer d'un utilisateur à un autre en connaissant son mot de passe. Par défaut l'environnement de l'utilisateur source est conservé, pour utiliser celui de l'utilisateur destination, on ajoute l'option « - ». Par exemple, si l'utilisateur « sam » veut devenir « luke » :

```
[sam@drocourt  ~]# su luke
Mot de passe :
[luke@drocourt ~]# pwd
/home/sam
[luke@drocourt ~]# exit
[sam@drocourt  ~]# su - luke
Mot de passe :
[luke@drocourt ~]# pwd
/home/luke
```

Pour devenir « root », il suffit donc de réaliser l'opération « su - root », ou plus simplement « su - », **il est donc nécessaire de connaître le mot de passe du « root »**.

Commande « sudo »

La commande « sudo » est configurée via un fichier de configuration « /etc/sudoers », et permet d'indiquer quel utilisateur peut réaliser quelle opération en tant que « root ».

Dans les distributions, cela est géré par l'appartenance à un groupe :

- Groupe « wheel » sous RedHat/CentOS,
- Groupe « sudo » sous Ubuntu.

Le mot de passe demandé sera celui de l'utilisateur, et non celui du « root » !!

Pou lancer une commande unique :

```
[sam@drocourt  ~]$ sudo id
[sudo] Mot de passe de sam :
uid=0(root) gid=0(root) groupes=0(root)
```

Il peut être intéressant de rester « root » lorsque l'on souhaite exécuter plusieurs commandes :

```
[sam@drocourt ~]$ sudo -s
[root@drocourt ~]# pwd
/home/sam
[root@drocourt ~]# exit
[sam@drocourt ~]$
```

De la même manière que pour la commande « su », il est possible de récupérer l'environnement du « root » avec l'option « -i » :

```
[sam@drocourt ~]$ sudo -i
[root@drocourt ~]# pwd
/root
[root@drocourt ~]# exit
[sam@drocourt ~]$
```

Chapitre 2 - Organisation et gestion des fichiers

Table des matières

1 - Le root-filesystem et ses répertoires.

Le système a besoin de certains répertoires ainsi que de certains fichiers essentiels pour pouvoir fonctionner normalement.

La hiérarchie de ces répertoires est normalement régit par une norme appelée **FHS** pour « **Filesystem Hierarchy Standard** ».

Ce standard est normalement suivi pour tous les Unix, en partie ou totalement, hormis quelques exceptions comme MacOS X.

Parmi les membres les plus importants on peut citer Hewlett-Packard, Red Hat, IBM, Dell, ...

- `/dev` : Répertoire contenant les périphériques,
- `/proc` : Pseudo répertoire permettant l'accès aux paramètres du noyau,
- `/etc` : Répertoire contenant les fichiers de configuration du système,
- `/bin` : Répertoire contenant les commandes essentielles au système,
- `/sbin` : Les binaires systèmes spécifiques au « root »,,
- `/lib` : Les librairies systèmes,
- `/mnt` : Répertoire pouvant être utilisé comme point de montage temporaire,
- `/media` : Répertoire des points de montage pour les périphériques amovibles,
- `/usr` : Le répertoire contenant des utilitaires et applications supplémentaires,
- `/home` : Le répertoire utilisé traditionnellement pour les utilisateurs,
- `/tmp` : Répertoire temporaire accessible en écriture pour tous,
- `/var` : Répertoire contenant des données de taille variable,
- `/boot` : Répertoire contenant les données de démarrage comme le noyau,
- `/opt` : Logiciels optionnels, installés manuellement,
- `...`

2 - Les types de fichier

2.1 - Fichiers ordinaires

Sur disque, il peuvent contenir des données, des sources dans un langage donné, des binaires exécutables… (symbole « - »).

2.2 - Fichiers catalogue

Contient une suite de couples (nom, i-nombre). Ils sont également appelés indifféremment répertoires ou « directory » (symbole « d »).

2.3 - Fichiers lien symbolique, fichiers tube nommé, fichiers socket

Qui correspondent à des éléments particuliers qui seront vu plus loin dans ce cours (symbole « l », « p » et « s »).

2.4 - Fichiers spéciaux

Ils correspondent aux dispositifs d'E/S physiques et sont tous référencés dans le catalogue */dev* (symboles « c » et « b »)

Ex: */dev/tty1* est la référence d'un terminal.

/dev/sda3 est la référence d'un disque logique.

```
[cd@drocourt ~]$ ls -al /dev/sda3
brw-rw---- 1 root disk 8, 3 sept. 19 07:01 /dev/sda3
[cd@drocourt ~]$ ls -al /dev/tty1
crw--w---- 1 root tty 4, 1 sept. 19 07:01 /dev/tty1
```

Du point de vue de l'utilisateur, il n'y a aucune différence entre ces fichiers et les fichiers ordinaires.

Ex : *cal > /dev/tty12* affiche le calendrier sur l'écran du terminal tty12.

Ils existent plusieurs périphériques systèmes spéciaux (numéros majeurs 1) dont les plus importants sont :

- `/dev/null` : Périphérique de type entrée vers lequel il est possible de rediriger tout type de flux, qui sera donc supprimé,
- `/dev/zero` : Périphérique de type sortie à partir duquel il est possible de demander de l'information et qui fournira systématiquement des « 0 » binaires,
- `/dev/random` : Périphérique de type sortie à partir duquel il est possible de demander de l'information et qui fournira systématiquement des données aléatoires, pour générer ce type de données le système utilise la charge du système et d'autres indicateurs,
- `/dev/urandom` : Périphérique de type sortie à partir duquel il est possible de demander de l'information et qui fournira systématiquement des données aléatoires à partir d'un algorithme,

3 - Les principales commandes de manipulation de fichiers

3.1 - Référence du catalogue de travail : pwd

Cette commande permet de connaître sa position dans le système de fichier, exemple :

```
[cd@drocourt ~]$ pwd
/home/profs/drocourt
```

3.2 - Changement du catalogue de travail : cd

Pour se déplacer dans l'arborescence du système de fichier, on utilise la commande **cd**. On peut indiquer le chemin dans lequel on veut se déplacer de façon relative (la position courant est donnée par un point et le parent par un ..) ou absolue (dans lequel on précise que l'on fait référence à la racine avec un /). De plus, **cd** sans argument retourne au répertoire de connexion.

Exemples :

```
[cd@drocourt ~]$  pwd
/home/profs/drocourt
[cd@drocourt ~]$ cd ..
[cd@drocourt profs]$ pwd
/home/profs
[cd@drocourt profs]$  cd /tmp
[cd@drocourt tmp]$  pwd
/tmp
[cd@drocourt tmp]$  cd ../home
[cd@drocourt home]$  pwd
/home
[cd@drocourt home]$  cd
[cd@drocourt ~]$  pwd
/home/profs/drocourt
[cd@drocourt ~]$
```

3.3 - Commande « touch »

Cette commande permet de créer un fichier vide :

```
[cd@drocourt ~]$ touch file1
[cd@drocourt ~]$ touch file2
[cd@drocourt ~]$ ls
file1 file2
```

3.4 - Copie physique d'un fichier : cp

La commande **cp** réalise la copie physique d'un fichier, c'est-à-dire une duplication des données sur le disque dur. Exemple :

```
[cd@drocourt ~]$ ls
file1  file2
[cd@drocourt ~]$ cp file1 file3
[cd@drocourt ~]$ ls
file1  file2  file3
```

3.5 - Déplacement ou changement du nom d'un fichier : mv

Cette commande permet en fait uniquement de déplacer un fichier mais permet de donner un nom différents au fichier destination, elle peut donc être utilisée pour renommer un fichier :

```
[cd@drocourt ~]$ ls
file1  file2  file3
[cd@drocourt ~]$ ls /tmp
[cd@drocourt ~]$ mv file2 /tmp
[cd@drocourt ~]$ ls /tmp
file2
[cd@drocourt ~]$ ls
file1  file3
[cd@drocourt ~]$ mv file3 file4
[cd@drocourt ~]$ ls
file1  file4
[cd@drocourt ~]$
```

3.6 - Suppression d'un fichier : rm

Cette commande permet de supprimer un fichier dans le répertoire courant ou ailleurs dans l'arborescence en précisant le chemin d'accès. Exemple :

```
[cd@drocourt ~]$ ls /tmp
file2
[cd@drocourt ~]$ rm /tmp/file2
[cd@drocourt ~]$ ls /tmp
[cd@drocourt ~]$
```

3.7 - Création d'un catalogue : mkdir

Cette commande permet la création d'un répertoire. Exemple :

```
[cd@drocourt ~]$ ls
file1  file4
[cd@drocourt ~]$ mkdir rep1
[cd@drocourt ~]$ mkdir rep2
[cd@drocourt ~]$ ls
file1  file4  rep1  rep2
```

3.8 - Effacement d'un catalogue vide : rmdir

Attention il faut impérativement que le répertoire soit vide sinon le système refusera. Exemple :

```
[cd@drocourt ~]$ ls
file1  file4  rep1  rep2
[cd@drocourt ~]$ rmdir rep2
[cd@drocourt ~]$ ls
file1  file4  rep1
[cd@drocourt ~]$ touch rep1/file10
[cd@drocourt ~]$ rmdir rep1
rmdir: `rep1': Le répertoire n'est pas vide.
[cd@drocourt ~]$ ls
file1  file4  rep1
[cd@drocourt ~]$
```

3.9 - Effacement d'un catalogue non vide : rm –r

Exemple :

```
[cd@drocourt ~]$ ls
file1  file4  rep1
[cd@drocourt ~]$ rm –r rep1
[cd@drocourt ~]$ ls
file1  file4
[cd@drocourt ~]$
```

3.10 - Création de lien sur un fichier : ln

La commande *ln* réalise un lien normal, c'est-à-dire une nouvelle référence à un fichier physique. Il n'y a pas création d'un i-nœud ni duplication des données sur le disque dur. Pour réaliser un lien normal entre deux fichiers, ceux-ci doivent appartenir au même système de fichier.

La commande *ln –s* permet de créer un lien symbolique : il y a création d'un fichier de type lien symbolique dont le contenu est la référence du fichier à lier, c'est ce type de liens qui est le plus couremment utilisé.

Exemple :

```
[cd@drocourt ~]$ ls –l
total 0
-rw-r--r--  1 drocourt   adm   0 déc  2 16:04 file1
-rw-r--r--  1 drocourt   adm   0 déc  2 16:19 file4
[cd@drocourt ~]$ ln –s file1 file2
[cd@drocourt ~]$
[cd@drocourt ~]$ ls -l
-rw-r--r--  1 drocourt   adm   0 déc  2 16:04 file1
lrwxrwxrwx  1 drocourt   adm   5 déc  2 16:41 file2 -> file1
-rw-r--r--  1 drocourt   adm   0 déc  2 16:19 file4
[cd@drocourt ~]$ rm file1
[cd@drocourt ~]$ ls -l
lrwxrwxrwx  1 drocourt   adm   5 déc  2 16:41 file2 -> file1
-rw-r--r--  1 drocourt   adm   0 déc  2 16:19 file4
[cd@drocourt ~]$ rm file2
```

Les liens « hard » sont différents et correspondent en fait à une autre entrée vers le même fichier :

```
[cd@drocourt ~]$ ls -l
total 0
-rw-r--r--  1 drocourt   adm   0 déc  2 16:19 file4
[cd@drocourt ~]$ ln file4 file3
[cd@drocourt ~]$ ls -l
-rw-r--r--  2 drocourt   adm   0 déc  2 16:19 file3
-rw-r--r--  2 drocourt   adm   0 déc  2 16:19 file4
[cd@drocourt ~]$
```

3.11 - Contenu d'un fichier

Pour avoir le contenu d'un fichier on utilise la commande « cat » :

```
[cd@drocourt ~]$ cat /etc/hosts
127.0.0.1   localhost
127.0.1.1   xps
```

3.12 - Chemin d'une commande

Pour obtenir les répertoire où se trouve le fichier correspondant à une commande on utilise la commande « which » :

```
[cd@drocourt ~]$ which date
/bin/date
```

3.13 - Type d'un fichier

Il est possible de connaître le type d'un fichier à l'aide de la commande « file » :

```
[cd@drocourt ~]$ file /usr/share/pixmaps/filezilla.png
/usr/share/pixmaps/filezilla.png: PNG image data, 48 x 48,
8-bit/color RGBA, non-interlaced
```

4 - Mode d'un fichier

4.1 - Le principe

Chaque utilisateur possède un numéro d'identification (UID) et appartient à un ou plusieurs groupes donnés.

```
[cd@drocourt ~]$ id
uid=500(drocourt) gid=4(adm) groupes=4(adm),501(vboxusers)
```

Pour un fichier donné, les utilisateurs sont classés en trois catégories:

- Le propriétaire
- Le groupe propriétaire
- Les autres

Chacune de ces catégories peut avoir trois droits:

- le droit r (lecture)
- le droit w (écriture)
- le droit x (execution)

Selon le type de fichier, ces droits ont une signification particulière.

Droits	*Fichier*	*Répertoire*
r	voir le contenu	voir le contenu
w	modifier ou effacer le contenu	créer ou détruire des fichiers
x	exécuter	Traverser

4.2 - Modification des droits d'accès: la commande chmod

Change les droits d'accès à un fichier ou à un répertoire.

mode symbolique

```
chmod [-R-f] [ugo|a]{{-|+[rwxst]}|{=[ rwxst]}}{fichier}
```

La notation **symbolique** peut se résumer de la manière suivante :

- qui=permission
- qui+permission : ajouter droit(s)
- qui-permision : enlever droit(s)

avec

- qui = {u}{g}{o}{a}
- permission = {r}{w}{x}

```
[cd@drocourt ~]$ touch ess
[cd@drocourt ~]$ ls -l
total 0
-rw-r--r-- 1 drocourt adm 0 11 sept. 18:42 ess
[cd@drocourt ~]$ chmod g+w ess
[cd@drocourt ~]$ ls -l
total 0
-rw-rw-r-- 1 drocourt adm 0 11 sept. 18:42 ess
[cd@drocourt ~]$ chmod a+x ess
[cd@drocourt ~]$ ls -l
total 0
-rwxrwxr-x 1 drocourt adm 0 11 sept. 18:42 ess
[cd@drocourt ~]$
```

mode numérique ou absolu

```
chmod [-R-f] Code_accés {Fich...|Rép...}
```

La notation **octale** de type : *ugo* où *u, g, o* sont des valeurs octales de 0 à 7 définissant les droits d'accès à l'égard de l'**u**tilisateur, du **g**roupe et des autres (**o**thers) selon la règle :

- 0 : aucun droit
- 1 : exécution (x)
- 2 : écriture (w)
- 4 : lecture (r)
- *addition* = combinaison

Exemple :

Pour instaurer le droit `rwxr-x---` au fichier *ess* : 750 beurk car 7=1(x)+2(w)+4(r), 5=1(x)+4, 0=0(aucun droit).

```
[cd@drocourt ~]$ ls -l
total 0
-rwxrwx--x 1 drocourt adm 0 11 sept. 18:42 ess
[cd@drocourt ~]$ chmod 750 ess
[cd@drocourt ~]$ ls -l
total 0
-rwxr-x--- 1 drocourt adm 0 11 sept. 18:42 ess
[cd@drocourt ~]$
```

4.3 - Changement de propriétaire et de groupe propriétaire

```
chown <proprio> <fic>
chgrp <proprio> <fic>
```

Ces changements ne peuvent être effectués que par le propriétaire initial du fichier ou le root, pour changer le propriétaire et le groupe :

```
chown <proprio>.<groupe> <fic>
```

5 - Les bits spéciaux

5.1 - Le sticky-bit

Ce bit désigné par t apparaît à la place du champ x dans le groupe de bits concernant la classe other. Il ne s'applique qu'aux fichiers exécutables et aux répertoires.

Signification pour un fichier exécutable

Cet attribut indique que le segment texte d'un fichier exécutable n'est swappé out qu'une seule fois et qu'il est conservé dans l'espace disque de swap une fois que la commande est terminée. Ce mécanisme réduit donc le nombre de swap out et permet un chargement en mémoire plus rapide lors d'une exécution ultérieure d'une commande.

Signification pour un répertoire

Pour un répertoire, cette possibilité a été introduite par la version BSD 4.3. Notons qu'elle est actuellement implantée dans la totalité des versions commerciales. Pour un répertoire public dont les permissions sont drwxrwxrwt, le bit t a pour effet de protéger les fichiers qu'il contient contre la suppression. Cela signifie que dans ce cas, seul le propriétaire du fichier et root peuvent supprimer un fichier situé dans un tel répertoire.
Cette notion a été introduite pour protéger contre la suppression les fichiers temporaires qui se trouvent dans les répertoires publics comme /tmp, /usr/tmp, ou /usr/spool/tmp, …

```
[cd@drocourt ~]$ ls -ld /tmp
drwxrwxrwt 12 root root 12288 sept. 20 11:39 /tmp
```

5.2 - Le set-uid bit

On le positionne pour un binaire exécutable pour "prêter" les droits d'accès du propriétaire du fichier aux autres utilisateurs pendant le temps d'exécution du processus exécutant le code contenu dans le fichier.

Cette fonctionnalité est notamment utilisée pour permettre à un utilisateur de changer son mot de passe.

```
[cd@drocourt ~]$ ls -l /etc/passwd /usr/bin/passwd
-rw-r--r-- 1 root system    10437 Jun 11 16:14 /etc/passwd
-r-sr-x--x   1   root   security      48750   Oct   26   1994
/usr/bin/passwd
```

Le fichier */etc/passwd* contient les définitions des utilisateurs, il est propriété de l'utilisateur *root* et du groupe d'utilisateur *system* et il n'est pas accessible en écriture à un autre utilisateur.

Le fichier */usr/bin/passwd* contient le code (binaire) de la commande *passwd*, il est propriété de l'utilisateur *root* et tout utilisateur est autorisé à l'exécuter.

Un utilisateur lance la commande *passwd* : il initialise un processus exécutant le code contenu dans */usr/bin/passwd*. Le set-uid bit ayant été positionné sur celui-ci, les droits de ce processus seront ceux du propriétaire de ce fichier, c'est à dire ceux de *root*. C'est ainsi que l'utilisateur peut modifier un fichier (ici, */etc/passwd*) sur lequel il n'a pas droit d'écriture.

6 - I-nombre et i-nœud

Un i-nœud est référencé par un unique i-nombre qui est associé à un ou plusieurs fichiers. Ce i-nœud contient:

- la taille du fichier en octet
- les adresses des blocs utilisés sur le disque dur
- le numéro d'identification (UID) du propriétaire
- le numéro d'identification (GID) du groupe propriétaire
- le nombre de liens : le nombre de fois où le i-nombre référençant le i-nœud est associé à un nom de fichier
- le type (ordinaire, catalogue, spécial…)
- le droit d'accès au fichier
- la date de dernier accès
- la date de dernière modification du i-nœud.

7 - L'éditeur de texte VI

7.1 - Etude rapide

Même si il existe d'autres éditeurs de texte (anciens comme « ed » ou plus récents comme « emacs » ou après « nano »), l'éditeur VI reste aujourd'hui le plus disponible en terme de plateforme.

7.2 - Entrée et sortie de vi

Il suffit de lancer la commande vi avec comme argument le nom d'un fichier, il affiche alors la première page de copie et des tildes à la place des lignes vides si nécessaire. Le texte est stocké dans un buffer temporaire, les opérations d'édition modifient le buffer, pas le fichier original.

```
[cd@drocourt ~]$ vi toto.txt
Bonjour
~
~
~
~
```

7.3 - Premières commandes

Voici les premières commandes à connaître :

- Pour sauver le fichier à un moment donné on peut taper ": w" suivi de return.
- Pour sauver l'édition et quitter vi, presser la touche "shift" tout en appuyant deux fois sur la touche Z ou alors ": wq" suivi de "return".
- Pour quitter vi sans sauvegarde, taper ": q !" suivi de "return".

7.4 - Le mode commande

VI se met initialement en mode commande et les touches invoquent alors des fonctions d'édition, applicables immédiatement

Recherche

- "?blabla" : recherche de la chaîne blabla du curseur vers le haut
- "/blabla" : recherche de la chaîne blabla du curseur vers le bas
- "n" : passage à l'occurence suivante de la chaîne.

Déplacement du curseur

Les flèches ou :

- « l » : vers la droite,
- « k » : vers le haut,
- « j » : vers le bas,
- « h » : vers la gauche

Appel d'une commande

On peut sans quitter vi lancer une commande UNIX en mode commande en tapant ":!commande".

Refaire/ défaire

- "u" : défait la dernière modification (dest. ou insert).
- "." : refait la dernière modification (dest. ou insert).

Copié/Collé

- "x" : détruit un caractères et le place dans le buffer,
- "dd" : détruit une ligne et la place dans le buffer,
- "5dd" : détruit cinq lignes et les placent dans le buffer,
- "5yy" : Copie les cinq lignes qui suivent le curseur dans le buffer
- "p" : Copie le contenu du buffer en dessous du curseur
- "P" : Copie le contenu du buffer au dessus du curseur.

7.5 - Le mode insertion

Sous VI, pour pouvoir taper du texte il faut se placer en mode insertion. Pour cela on utilise les commandes :

- "i" : pour éditer avant le curseur
- "a" : pour éditer aprés le curseur
- "I" : pour éditer au début de la ligne
- "A" : pour éditer à la fin de la ligne.

Pour quitter le mode insertion et repasser en mode commande il suffit de taper sur la touche "ESC".

7.6 - Configuration de vi

En mode commande

On tape en mode commande :

- ":set" permet de connaître les options positionnées
- ":set all" permet de connaître toutes les options disponibles.

Pour modifier les paramétrages on va utiliser par exemple :

- ": set number" : pour activer la numérotation des lignes,
- ": set nonumber" : pour supprimer cette option,

Quelques options bien pratiques pour rédiger un programme :

- autoindent : indentation automatique ("CTRL D" pour l'éviter)
- showmatch : montre les associations de parenthèses/accolades
- number : numérote les lignes à l'affichage

A la connexion

Au lieu de configurer vi en mode commande à chaque fois qu'il est exécuté, il est plus pratique de le configurer par défaut.

Il suffit d'exporter la variable EXINIT dans le fichier de configuration de l'utilisateur avec les options :

```
export EXINIT = "set redraw showmode"
```

A chaque invocation

On peut configurer VI en fonction du contexte en créant un fichier « .exrc » dans le répertoire concerné qui contiendra les options choisies.

Chapitre 3 - L'exécution des commandes et le shell

Table des matières

1 - Les principales commandes Unix

1.1 - Introduction

Dans les systèmes Unix, on peut différencier deux types de commandes :

- **Les commandes externes** : Qui sont des programmes du système, comme « vi », « awk », …
- **Les commandes internes** : Qui ne sont pas des programmes, et n'existent donc pas dans le système. Ils sont uniquement compris et interprétées par le Shell, c'est la cas des commandes « cd », « eval », … Aucun processus n'est donc créé pour les exécuter.

Il y a par contre des commandes hybrides, c'est à dire que ce sont des commandes externes, mais qui existent aussi en tant que commandes internes, et ce sont ces dernières qui sont utilisées par défaut. C'est le cas par exemple des commandes « echo », « kill », ...

- **`wc`** : (word count) pour compter, avec comme options principales :

  `-l` : pour compter le nombre de lignes

  `-w` : pour compter le nombre de mots dans un fichier

  `-c` : pour compter le nombre de caractères dans un fichier.

```
[cd@drocourt ~]$ wc -l /etc/passwd
47 /etc/passwd
```

1.2 - Commande « grep »

La commande « grep » permet la recherche d'un motif (chaîne de caractères) dans un fichier ou un ensemble de fichiers :

```
[cd@drocourt ~]$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Quelques options sont intéressantes :

- -i : Pour ignorer la casse,
- -v : Inverse la signification de la recherche,
- -n : Affiche en plus le numéro de la ligne,
- -r : Effectue une recherche récursive dans le ou les répertoires indiqués,

1.3 - La commande « sort »

La commande « sort » permet de trier un fichier par défaut dans l'ordre alphanumérique :

```
[cd@drocourt ~]$ sort /etc/passwd
abrt:x:173:173::/etc/abrt:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
...
```

Quelques options sont intéressantes :

- -n : Effectue un tri numérique,
- -r : Inverse l'ordre de tri,
- -k <n> : Permet d'indiquer sur quel champ effectuer le tri,
- -t <sep> : Permet d'indiquer le séparateur de champ,

1.4 - La commande « cut »

Elle permet de découper des champs sélectionnés dans chacune des lignes d'un fichier. Deux syntaxes sont disponibles:

- **cut -c<liste> <fic>** : Pour couper suivant des caractères,
- **cut -f<liste> -d<car> [-s] <fic>** : Pour couper suivant des colonnes, dans ce cas il est nécessaire de préciser le séparateur de champs à l'aide de l'option « -d »,

Le paramètre « liste » est une liste de numéros de champs séparés par des virgules:

- 1,4,7 : ne seront extraites que les colonnes 1, 4 et 7.
- 1-3,8 : pour 1,2,3,8
- -5,8 : pour 1,2,3,4,5,8
- 3- : pour tous les champs à partir du troisième

Pour extraire la première colonne du fichier « /etc/passwd » :

```
[cd@drocourt ~]$ cut -d':' -f1 /etc/passwd
```

1.5 - Recherche avec "find".

Cette commande permet de rechercher dans toute l'arborescence de chaque répertoire spécifié les fichiers satisfaisant au(x) condition(s) donnée(s).

```
find <chemin> <conditions> <actions>
```

Quelques conditions possibles (plusieurs critères peuvent être combinés) :

- `-name 'fichiers'` recherche fichiers de noms spécifiés,
- `-user username` fichiers appartenant à l'utilisateur username
- `-type type` type : d= directory, f= fichier, l= lien symbolique...
- `-mtime n` fichiers modifiés il y a exactement n jours,
- `-mtime -n` fichiers modifiés depuis moins de n jours,
- `-perm nnn` fichiers dont la protection est nnn
- `-a` pour effectuer un « and » sur deux critères
- `-o` pour effectuer un « or » sur deux critères

Quelques actions possibles :

`-print`	affichage les résultats avec leur chemin complet,
`-ls`	idem en plus détaillé,
`-exec commande {} \;`	applique la commande sur tous les fichiers satisfaisants,
`-OK commande {} \;`	idem mais demande confirmation pour chacun,

Exemple 1 : Recherche des fichiers ayant l'extension « .conf ».

```
[cd@drocourt ~]$ find /etc -name "*.conf" -print
```

Exemple 2 : Recherche des fichiers PDF modifiés depuis moins de 2 jours.

```
[cd@drocourt ~]$ find . -mtime -2 -a -name "*.pdf" -print
```

Exemple 3 : Affiche la taille de chacun des fichiers « .tmp » trouvés.

```
[cd@drocourt ~]$ find . -name "*.tmp" -exec du -sh {} \;
```

1.6 - Les principales commandes standards

- **id** : permet d'obtenir des informations relatives à un utilisateur,
- **logname** : fournit la valeur de la variable d'environnement LOGNAME, qui est initialisée lors de la connexion.
- **su** : permet de prendre l'identité d'un autre utilisateur, par défaut le « root », le signe « - » indique que l'on souhaite récupérer l'environnement de l'utilisateur.
- **cal** : pour afficher un calendrier.
- **date** : pour afficher la date du jour, il est possible de spécifier un format avec l'option + et par exemple les champs %H pour l'heure, %M pour les minutes, ...
- **cmp** : Permet de comparer deux fichiers de nature quelconque (ascii ou binaire) Sa syntaxe est : cmp [-l] [-s] file1 file2
- **diff** : affiche les différences entre deux fichiers texte, l'option « -b » permet d'ignorer les espaces en fin de ligne. De plus, cette commande indique comment passer du fichier source au fichier de destination.

- **split** : permet de scinder un fichier en fichiers plus petits. Le fichier passé en argument est alors découpé par défaut en fichiers de 1000 lignes appelés respectivement par le nom passé en argument suivi des caractères aa, ab, ac, … Quelques options sont intéressantes : -l <lignes> pour modifier le nombre de lignes par fichier, -n <nombre> découpe le fichier source en « n » fichiers et -b <size> pour découper le fichier source en fichiers de taille « size » (K,M,G,...),
- **head [-<x> ou -n <x>]** : Permet d'afficher les « x » premières lignes,
- **tail [-<x> ou -n <x>]** : Permet d'afficher les « x » dernières lignes,
- **tr** : permet d'appliquer un transcodage des caractères entre l'entrée et la sortie standard.
- **paste** : permet de concaténer des fichiers par colonnes,
- **more/less** : pour l'affichage par page d'un fichier,
- **du [répertoire(s)]** : Affiche sur sortie standard l'espace-disque utilisé. Il est possible de demander l'affichage de la somme (-s) et de spécifier une échelle (« -k » Kilooctets, « -m » Mégaoctets, « -g » Gigaoctets, ou « -h » pour automatique).

1.7 - Les filtres

On appelle filtre un programme qui lit des données sur l'entrée standard (stdin) et qui écrit ses résultats sur la sortie standard (stdout) après les avoir filtrées. Les commandes les plus courantes en la matière sont cat, wc, sort, cut, ... qui sont donc utilisables avec les tubes.

2 - Définition du Shell

2.1 - Qu'est-ce qu'un shell ?

Un Shell à deux rôles fondamentaux :

- Exécuter des scripts, appelés Scripts-Shells, écrit dans le langage spécifique du Shell, dans ce cas le Shell est considéré comme un langage interprété,
- Traiter les commandes saisies par l'utilisateur en interactif, c'est donc aussi le shell qui gère l'environnement de travail de l'utilisateur (alias, historique, etc.).

Un Shell interactif est un processus qui fonctionne en boucle perpétuelle:

1. Affiche d'une invite (le contenu de la variable d'environnement PS1),
2. Lecture de la ligne de commande et réalisation de substitutions diverses,
3. Lancement d'un ou plusieurs processus pour exécuter cette ligne de commande,
4. Retour en 1 après la fin de cette exécution (avant-plan) ou immédiatement (arrière-plan),

2.2 - Les différents interpréteurs

Parmi l'ensemble des Shells existants, les plus connus sont :

- **bsh** : Le shell d'origine, aussi appelé Bourne Shell en référence au responsable du développement Steeve Bourne, il est disponible sur tous les systèmes Unix,
- **csh** : Un nouveau type de Shell a été développé par l'Université de Berkeley avec une nouvelle syntaxe et apporte entre autre les fichiers de configuration, l'historique, les alias, … mais incompatible avec le précédent,
- **ksh** : Le Korn Shell a été initié par David Korn des laboratoires Bell pour ajouter les possibilités du csh dans le bsh, il est disponible sur quasiment tous les systèmes Unix,
- **tcsh** : Extension du csh,
- **bash** : Le Shell du projet GNU, écrit par Brian Fox, relativement proche du ksh,
- **zsh** : Plus récent, et augmentant encore les possibilités de bash/ksh,

2.3 - Disponibilité des interpréteurs

Les différents Shells précédents ne sont pas toujours disponibles nativement sur tous les Unix, et le Shell par défaut peut aussi changer :

- Sur les Unix propriétaires, le Shell standard est souvent « ksh », et le « bash » n'est pas toujours disponible,
- Sur les Unix OpenSources (BSD, Linux, et même MacOSX), le Shell par défaut est souvent le « bash »,

Le « bash » est apparu car le « ksh » n'était pas OpenSource, par conséquent, de nombreuses variantes de ce dernier existent aujourd'hui sur les systèmes.

Il est très important de bien distinguer :

- Le Shell qui sera utilisé en interactif, qui est plutôt personnel,
- Le Shell utilisé dans les scripts qui doit être le plus portable possible,

2.4 - Différences Bourne Shell/Korn Shell/Bash.

Le premier Shell de référence est le « bsh », qui, il faut le rappeler, a été conçu **pour traiter des scripts**, son utilisation interactive est donc limitée, mais ce n'était pas son but premier.

Les Shells « ksh » et « bash », sont apparus quelques années après et **sont compatibles « bsh »**. Leurs améliorations se situent autour de deux axes :

- La facilitée d'utilisation en mode interactif (complétion, historique, …), ce qui ne nous intéresse pas ici,
- L'amélioration du langage (tableaux associatifs, boucles, …),

Le problème est que pour ce deuxième point, les deux principaux Shells n'ont globalement :

- soient pas toujours amenés les mêmes améliorations,
- soient pas toujours utilisés les mêmes syntaxes.

2.5 - Shell POSIX

Le Shell POSIX ajoute des fonctionnalités au Shell bsh, qui sont implémentées par défaut dans les deux principaux Shells, à savoir « bash » et « ksh ». Par contre :

- Certaines choses existent dans « ksh » mais pas dans « bash »,
- Certaines choses existent dans « bash » mais pas dans « ksh »,

Par conséquent, afin de rendre un script le plus portable possible et le plus standard, **il ne faut utiliser que les mécanismes de POSIX**. Si le développeur à besoin de caractéristiques plus évoluées, c'est qu'il doit utiliser un autre langage …

Sur les systèmes POSIX, ce Shell est disponible via la commande « sh », sinon :

- Avec « bash » : Il faut positionner l'option « --posix »,
- Avec ksh : Certaines versions supportent la variable d'environnement « POSIXLY_CORRECT=YES »,

3 - Les différents fichiers de démarrage

3.1 - Système

Sur certains systèmes Unix (AIX, Linux, ...), tous les processus exécutés par le système vont être initialisés avec les variables d'environnement définies dans le fichier « /etc/environment ».

3.2 - Le Shell POSIX

Pour le Shell POSIX, aucune amélioration particulière n'a été fait pas rapport au Borne Shell. Les fichiers sont donc :

- `/etc/profile` : Pour les Shells de « login » de tous les utilisateurs,
- `$HOME/.profile` : Pour les Shells de « login » de l'utilisateur,

3.3 - Le Shell « ksh »

Les fichiers de configuration sont :

- `/etc/profile` : Pour les Shells de « login » de tous les utilisateurs,
- `$HOME/.profile` : Pour les Shells de « login » de l'utilisateur,
- `ENV=nom_fichier` : Le contenu de la variable ENV donne le nom du fichier de l'utilisateur qui sera utilisé pour tous les Shells, par convention « `.kshrc` ».

Pour le dernier, il faudra donc préciser dans l'un des fichiers précédents la définition de la variable ENV de la forme :

```
export ENV="$HOME/.kshrc"
```

3.4 - Le Shell « bash »

Les fichiers de configuration sont :

- `/etc/profile` : Pour les Shells de « login » de tous les utilisateurs,
- `$HOME/.bash_profile`,

  à défaut `$HOME/.bash_login`

  à défaut `$HOME/.profile` : Pour les Shells de « login » de l'utilisateur,
- `$HOME/.bashrc` : Pour tous les Shells de l'utilisateur,
- `$HOME/.bash_logout` : A la fin de la session de travail ce fichier est exécuté,

4 - Notions d'environnement

4.1 - Introduction

L'environnement d'un Shell est constitué de :

- Les fichiers/flux ouverts,
- Le répertoire de travail (cd),
- Le masque de création de fichiers (umask),
- La limitation des fichiers (ulimit),
- La gestion des signaux (trap),
- Les variables d'environnements (export),
- Les fonctions,
- Les options (set),
- Le PID de la dernière commande exécutée,
- Les Alias,

4.2 - Les alias

Ce mécanisme permet de donner des surnoms ou des abréviations à des commandes Unix. La syntaxe est la suivante :

alias : sans argument donne la liste des alias.

alias <nom>=<valeur> : initialise un alias

unalias <nom> : supprime un alias.

```
[cd@drocourt ~]$ ll
bash: ll : commande introuvable
[cd@drocourt ~]$ alias ll='ls -l'
[cd@drocourt ~]$ ll
total 144
lrwxrwxrwx   1 root  root     15 févr. 21  2013 bin
lrwxrwxrwx   1 root  root     19 févr. 21  2013 exp
...
```

4.3 - Masque des droits d'accès

Cette fonctionnalité permet de protéger automatiquement les fichiers à leur création en inhibant certains droits. Le masque courant est donné par la commande « umask » ; il est modifié par la commande « umask xyz », où xyz est la nouvelle valeur du masque. On place généralement cette commande dans le fichier de configuration du Shell.

```
[cd@drocourt ~]$ umask
0022
[cd@drocourt ~]$ touch ess1
[cd@drocourt ~]$ ls -l
-rw-r--r-- 1 drocourt users 0 sept. 10 19:55 ess1
[cd@drocourt ~]$ umask 027
[cd@drocourt ~]$ touch ess2
[cd@drocourt ~]$ ls -l
-rw-r--r--   1 drocourt users    0 sept. 10 19:55 ess1
-rw-r-----   1 drocourt users    0 sept. 10 19:55 ess2
```

5 - Les variables

5.1 - Variables locales

Une variable ne peut contenir qu'une chaîne de caractères ou un entier :

```
[cd@drocourt ~]$ val=toto
[cd@drocourt ~]$ echo $val
toto
[cd@drocourt ~]$ unset val
[cd@drocourt ~]$ echo $val

[cd@drocourt ~]$
```

Seul le Shell courant a connaissance de ces variables ; la liste des variables du Shell est donnée par la commande « **set** ».

5.2 - Commande « unset »

Il est possible de supprimer une variable (ou une fonction) à l'aide de la commande « unset ».

5.3 - Variables d'environnement

Certaines variables sont prédéfinies au niveau du Shell et sont lisibles par la descendance de ce processus Shell, par exemple :

- `ENV` : Nom du fichier de configuration,
- `HOME` : Répertoire de connexion,
- `IFS` : Le séparateur de champ (par défaut <space><tab><newline>),
- `LANG` : Le paramètre de langage
- `PS1` : Invite principale du shell en mode interpréteur
- `PS2` : Invite secondaire du shell en mode programmation
- `PS3` : Valeur de l'invite de sélection dans une boucle select
- `PS4` : Invite de trace
- `PATH` : Chemin de recherche pour l'exécution des commandes
- `PPID` : Numéro du processus père du shell.
- `PWD` : Répertoire de travail courant défini par la commande cd

La commande « **export** » permet d'ajouter des variables exportées à l'environnement alors que la commande « **env** » permet de visualiser les variables d'environnement.

```
[cd@drocourt ~]$ TOTO=3
[cd@drocourt ~]$ echo $TOTO
3
[cd@drocourt ~]$ bash
[cd@drocourt ~]$ echo $TOTO

[cd@drocourt ~]$ exit
exit
[cd@drocourt ~]$ export TOTO
[cd@drocourt ~]$ bash
[cd@drocourt ~]$ echo $TOTO
3
[cd@drocourt ~]$ exit
exit
[cd@drocourt ~]$
```

5.4 - Lecture seule

Il est possible de déclarer une variable en lecture seule afin d'interdire toute modification ultérieure :

```
[cd@drocourt ~]$ readonly constante1=34
[cd@drocourt ~]$ constante1=45
bash: constante1 : variable en lecture seule
[cd@drocourt ~]$
```

```
[cd@drocourt ~]$ constante2="toto"
[cd@drocourt ~]$ constante2="titi"
[cd@drocourt ~]$ readonly constante2
[cd@drocourt ~]$ constante2="tutu"
bash: constante2 : variable en lecture seule
```

5.5 - Commande « set »

La commande « **set** » permet :

- Sans argument : de visualiser l'ensemble des variables du Shell en cours,
- En précisant des arguments : elle les place en positionnel ($1, $2, …),
- En précisant des options : elle modifie le comportement su Shell.

```
[cd@drocourt ~]$ echo $1 $2

[cd@drocourt ~]$ set 23 coco
[cd@drocourt ~]$ echo $1 $2
23 coco
```

La commande « **set** » peut prendre des options sous la forme « -X » pour activer l'option « X », ou « +X » pour désactiver cette option. Les options le plus intéressantes sont :

- `-a` : Toutes les variables déclarées seront exportés,
- `-b` : Active l'avertissement utilisateur de la fin des jobs,
- `-C` : Pour interdire l'écrasement d'un fichier sur une redirection,
- `-e` : Stoppe le Shell en cours sur erreur d'une commande,
- `-n` : Pour tester les commandes sans les exécuter (ScriptShell),
- `-o` : Pour visualiser les options actuelles de « set »,
- `-u` : Afficher une erreur ou stop le script si utilisation d'une variable non initialisée,
- `-v` : Affiche la ligne récupérer sur l'entrée standard avant son exécution,
- `-x` : Affiche la commande à exécuter après les substitutions mais avant l'exécution,

6 - Exécution de commandes

6.1 - Les commandes simples

Plusieurs solutions pour exécuter un programme :

- `/chemin/programme` : Le programme est exécuté par le Shell courant s'il est exécutable,
- `programme` : Le programme est recherché dans les répertoires présents dans la variable d'environnement PATH, dans l'ordre de déclaration,
- `exec programme` : Le processus lié à a l'exécution du programme va remplacer le Shell courant, aucun processus n'est donc créé.

Ainsi, dans l'exemple suivant, le shell se termine donc immédiatement après l'exécution de la commande :

```
[cd@drocourt ~]$ exec sleep 10
(Attente de 10 secondes...)
Déconnexion
```

6.2 - Commandes consécutives

Introduction

L'utilisateur peut entrer sur la ligne de commande :

- Une commande simple,
- Une liste de commandes séparées par les caractère « ; »,
- Une liste de commandes séparées par les caractère « & »,
- Une liste de commandes séparées par les caractère « | »,
- Une liste de commandes séparées par les caractère « && »,
- Une liste de commandes séparées par les caractère « || »,

6.3 - Séparateur « ; »

Si plusieurs commandes se succèdent et sont séparées par des caractères « ; », alors le Shell les exécutent de manière séquentielle et attend la fin de la commande en cours pour passer à la suivante :

```
[cd@drocourt ~]$ date;sleep 1;date
jeudi 26 septembre 2019, 16:50:15 (UTC+0200)
     (attente 1 seconde)
jeudi 26 septembre 2019, 16:50:16 (UTC+0200)
```

6.4 - Séparateur « & »

Si plusieurs commandes se succèdent et sont séparées par des caractères « & », alors le Shell les exécutent de manière asynchrone et n'attend pas la fin de la commande en cours pour passer à la suivante. On dit que les commandes sont exécutées en arrière plan.

Exemple:

```
[cd@drocourt ~]$ sleep 10 &
[1] 4213
[cd@drocourt ~]$
```

Le « & » signifie que l'on a lancé la commande en arrière-plan et « 4125 » indique le PID de cette commande.

6.5 - Séparateur « | »

Si plusieurs commandes se succèdent et sont séparées par des caractères « | », alors le Shell les exécutent de manière séquentielle et attend la fin de la commande en cours pour passer à la suivante. Cependant, pour chaque commande, le Shell connecte la sortie standard à l'entrée standard de la commande suivante. On parle ici de « tube », et les commandes spécialisées dans ce type d'exécution s'appellent des filtres (cut, sed, …).

```
[cd@drocourt ~]$ cat /etc/passwd|wc -l
47
```

6.6 - Séparateur « && »

Si plusieurs commandes sont séparés par « && », le Shell passe à la commande suivante uniquement si celle en cours s'est correctement déroulée, c'est à dire à retourné un code de retour égal à 0 :

```
[cd@drocourt ~]$ rm file.txt && echo "OK"
rm: impossible de supprimer 'file.txt': Aucun fichier ou
dossier de ce type
[cd@drocourt ~]$ touch file.txt
[cd@drocourt ~]$ rm file.txt && echo "OK"
OK
[cd@drocourt ~]$
```

6.7 - Séparateur « ll »

Si plusieurs commandes sont séparés par « || », le Shell passe à la commande suivante uniquement si celle en cours ne s'est pas correctement déroulée, c'est à dire à retourné un code de retour différent de 0,

6.8 - Commandes groupées

Contexte

Il est possible de grouper des commandes afin de les faire exécuter dans un contexte particulier.

Parenthésage de commandes

Il est possible de grouper les commandes entre parenthèses pour les faire exécuter dans un sous-shell dédié.

Accolades de commandes

Il est possible de grouper les commandes entre accolades pour les faire exécuter dans le shell courant.

Exemples :

cd~$	30127 ? S 0:00 sshd: cd@pts/4 30129 pts/4 Ss+ 0:00 _ -bash
cd~$ **sleep 20; sleep 10;**	30127 ? S 0:00 sshd: cd@pts/4 30129 pts/4 Ss 0:00 _ -bash 30335 pts/4 S+ 0:00 _ sleep 20
Cd~$ { **sleep 20; sleep 10; }**	30127 ? S 0:00 sshd: cd@pts/4 30129 pts/4 Ss 0:00 _ -bash 30337 pts/4 S+ 0:00 _ sleep 20
Cd~$ **(sleep 20; sleep 10;)**	30127 ? S 0:00 sshd: cd@pts/4 30129 pts/4 Ss 0:00 _ -bash 30341 pts/4 S+ 0:00 _ -bash 30342 pts/4 S+ 0:00 _ sleep 20

7 - Remplacement de paramètres

7.1 - Introduction

Sur la ligne de commande, le Shell va effectuer des substitutions de certains paramètres pour les remplacer par leurs valeurs.

7.2 - Tilde

Le caractère tile « ~ » suivi du login d'un utilisateur du système sera remplacé par le chemin du « home directory » de ce dernier.

7.3 - Variables

Comme nous l'avons vu, le caractère « $ » suivi du nom d'une variable provoque par défaut son remplacement par sa valeur.

7.4 - Le backslash

Le caractère back slash « \ » permet d'annuler la signification particulière du caractère immédiatement après.

Exemple :

```
[cd@drocourt ~]$ VAR1=10
[cd@drocourt ~]$ echo $VAR1
10
[cd@drocourt ~]$ echo \$VAR1
$VAR1
```

7.5 - Quotage de chaines

Les chaînes de caractères peuvent être délimités par deux caractères :

- Simple quote '...' : les caractères inclus entre 2 simples cotes ne sont pas évalués, ils conservent leur valeur littérale.
- Double quote "..." : les caractères inclus entre 2 doubles quotes ne sont pas évalués et conservent leur valeur littérale à l'exception de $ ` et \.

Exemple :

```
[cd@drocourt ~]$ VAR1=10
[cd@drocourt ~]$ echo '$VAR1'
$VAR1
[cd@drocourt ~]$ echo "$VAR1"
10
```

7.6 - Variables de substitution

- `$!` PID du dernier processus en arrière plan
- `$?` Code retourné par la dernière commande
- `$$` PID du shell courant

7.7 - Métacaractères

Les métacaractères ne sont pas des "wildcards ". Un wildcard est interprété par une commande pour générer certains noms de fichiers. Par contre les métacaractères sont interprétés directement par le Shell avant l'exécution de la commande. Celle-ci reçoit des noms de fichiers, comme si ils avaient été entrés au clavier.

Les métacaractères disponibles sont :

- ? : Masque un caractère, sauf le point en première position.
- [ ] : Définit une classe de caractères :
  - \- : pour définir une suite,
  - ! : pour exprimer une exclusion
- Aucun séparateur n'est utilisé pour exprimer une liste.
- * : Masque toutes chaînes de caractères, sauf le point en première position.

Exemple 1 :

```
[cd@drocourt ~]$ echo /r*
/root /run
[cd@drocourt ~]$ echo "/r*"
/r*
[cd@drocourt ~]$ echo /[a-e]*
/bin /boot /cdrom /dev /etc
```

8 - Terminaison du shell

Le « Shell » interactif se termine lorsque la commande « exit » est entrée sur la ligne de commande.

Comme tout programme Unix, le Shell retourne une valeur de retour :

- Si « exit » est utilisé sans argument, le code de retour correspond à la dernière commande exécutée,
- Si « exit » est utilisé avec un argument de type entier, c'est ce code qui est retourné.

Si la commande n'est pas trouvée, c'est le code 127 qui doit être retournée, par contre si la commande est trouvée mais n'est pas exécutable, c'est le code 126.

Nous nous intéresserons aux codes de retour lors d'un prochain chapitre.

Exemple :

```
[cd@drocourt ~]$ sh
> sleep 1
> exit
[cd@drocourt ~]$ echo $?
0
[cd@drocourt ~]$ sh
> sleeep 1
sh: 1: sleeep: not found
> exit
[cd@drocourt ~]$ echo $?
127
```

9 - Substitution des commandes

La substitution de commandes permet de remplacer une chaîne par le résultat de son exécution. Pour cela deux formalismes sont autorisés :

- `$(command)`
- `` `command` ``

Exemple :

```
[cd@drocourt ~]$ VAR1=10
[cd@drocourt ~]$ echo 'echo $VAR1'
echo $VAR1
[cd@drocourt ~]$ echo "echo $VAR1"
echo 10
[cd@drocourt ~]$ echo `echo $VAR1`
10
```

10 - Evaluation arithmétique

Une évaluation arithmétique est réalisé par des doubles parenthèses précédées d'un symbole dollar : $((expression)).

Il faut noter que « expression » est traitée de la même manière qu'une chaîne de caractère entre des « " », c'est à dire que les substitutions sont réalisées.

```
[cd@drocourt ~]$ var=1
[cd@drocourt ~]$ echo "$var+1"
1+1
[cd@drocourt ~]$ echo $(($var+1))
2
```

Pour exécuter des calculs plus complexes, on pourra utiliser d'autres langages de programmation ou des commandes externes comme « bc ».

Chapitre 4 - La programmation du shell

Table des matières

1 - Introduction

Un « script-shell » est un ensemble de commandes reconnues par le Shell et stockées dans un fichier texte :

- Il permet la réalisation d'un ensemble de commandes sous forme consécutives, plus simples à manipuler et à structurer qu'une commande sur une seule ligne, et offrant plus de possibilités,
- Il est en général exécuté par un Shell dédié, un nouveau processus sera donc créé pour ceci.

Le comportant de l'ensemble des commandes du script est par défaut identique au comportement des commandes utilisées en interactif.

Il est possible, et même recommandé, de placer des commentaires dans les scripts en les faisant précéder du caractère « # ».

2 - Appel d'un shell : les différentes méthodes.

Quatre possibilités pour exécuter un script-shell :

- `./script.sh` : Taper directement le nom du script sous l'interpréteur de commande. L'interprétation s'effectue dans un sous-shell. Il faut néanmoins que l'utilisateur ait impérativement le droit d'exécution sur le fichier et que le script se trouve dans le PATH, sauf à le faire précéder de « ./ ».
- `sh script.sh` : Utiliser la commande « sh » (ou d'un autre shell) suivie du nom du script. L'interprétation s'effectue dans un sous-shell, et seul le droit de lecture est nécessaire.
- `. ./script.sh` : Taper directement le nom du script sous l'interpréteur de commande en le faisant précéder d'un point. L'interprétation s'effectue dans le shell courant
- `exec ./script.sh` : Utiliser la commande exec suivie du nom du script. L'interprétation s'effectue dans un sous-shell non interactif qui remplace le shell courant.

3 - Préambule du shell-script

Un script-shell débute obligatoirement par le chemin de l'interpréteur précédé des 2 caractères « # ! » appelés « Shebang ». Par exemple :

```
#!/bin/sh
```

Ce principe permet d'indiquer au système quel interpréteur il doit utiliser pour exécuter la suite du fichier, ce principe n'est donc pas limité au Shell, mais se retrouve également pour tous les langages interprétés (php, perl, python, …).

4 - Paramètres de position

4.1 - Généralités

Les paramètres positionnels sont des variables Shell qui permettent l'accès aux arguments d'un shell-script :

- $0 nom du script,
- $1,...,$9 arguments (du 1er au 9ème) ,
- $# nombre d'arguments,
- $* liste de tous les arguments sous la forme d'un seul paramètre,
- $@ liste de tous les arguments sous la forme d'une liste,

Exemple 1 :

```
#!/bin/bash
echo Nom du programme : $0
echo $1 $2
echo Fin du programme
```

4.2 - Décalage

Il est possible de réaliser un décalage des paramètres (hormis le 0) à l'aide de la commande « shift [n] », qui décale de une position par défaut, mais qui permet également de décaler de « n » positions.

```
#!/bin/bash
echo $1 $2 $3
shift
echo $1 $2 $3
```

```
[cd@drocourt ~]$ ./ess.sh 1 2 3
1 2 3
2 3
```

4.3 - Modification

Comme nous l'avons déjà vu, il est également possible de positionner les paramètres positionnels à l'aide de la commande « set » :

```
#!/bin/bash
echo $1 $2 $3
set a b c
echo $1 $2 $3
```

```
[cd@drocourt ~]$ ./ess.sh 1 2 3
1 2 3
a b c
```

5 - Sortie du shell-script

Toute commande Unix retourne une valeur sous a forme d'un entier qui renseigne sur la façon dont s'est déroulée l'exécution de ce dernier. Ce code de retour est stocké dans la variable d'environnement « $? ». La convention Unix impose le principe suivant : 0 quand l'exécution s'est déroulée avec succès et autre quand un problème est survenu.

```
[cd@drocourt ~]$ echo $?
0
```

Pour sortir d'un Script, il faudra donc utiliser la commande « exit », qui comme indiqué au chapitre précédent :

- Si « exit » est utilisé sans argument, le code de retour correspond à la dernière commande exécutée,
- Si « exit » est utilisé avec un argument de type entier, c'est ce code qui est retourné.

Les codes d'erreurs suivants sont normalisés :

- **1** : Erreur général,
- **2** : Utilisation abusive de commande interne du Shell,
- **126** : Impossible d'exécuter la commande,
- **127** : Commande non trouvée,
- **128** : Argument invalide de « exit »,
- **128+n** : Erreur fatal avec signal « n »,
- **255** : Code exit en dehors des valeurs autorisées,

6 - Variables et flux d'entrée

Il est possible de demander à l'utilisateur de saisir la valeur d'une variable à l'aide de la commande « read » :

```
#!/bin/sh
echo "Entrez votre nom :"
read nom
echo "nom : $nom"
```

Il est possible de demander plusieurs variables en une seule commande « read », dans ce cas chaque variable sera initialisée avec la valeur positionnelle correspondante, à l'exception de la dernière qui contiendra le reste.

Exemple de prompt :

```
#!/bin/sh

echo "Entrez deux chiffres :"
read CHIFFRE1 CHIFFRE2 RESTE

echo "Chiffre 1 : $CHIFFRE1, Chiffre 2 : $CHIFFRE2"
```

Exécution :

```
[cd@drocourt ~]$ ./ess.sh
Entrez deux chiffres :
12 23 33
Chiffre 1 : 12, Chiffre 2 : 23
```

Il est intéressant de noter que si un flux est envoyé sur un script, la commande « read » va lire sur ce flux, et non sur l'entrée standard :

```
#!/bin/sh
echo "Entrez votre nom :"
read nom
echo "nom : $nom"
```

```
[cd@drocourt ~]$ ./ess.sh
Entrez votre nom :
Rahan
Nom : Rahan
```

```
[cd@drocourt ~]$ echo "Martin"|./ess.sh
Entrez votre nom :
Nom : Martin
```

7 - Structures conditionnelles

7.1 - Expression conditionnelle

Syntaxe : **`test expr`** ou **`[ expr ]`** ou **`[[ exp ]]`**

Remarque : Attention il faut un espace après [et avant]

Test sur les types de fichier :

`-e fichier` : vrai si le fichier existe

`-d fichier` : vrai si le fichier existe et est un répertoire

`-f fichier` : vrai si le fichier existe et est un fichier régulier

`-s fichier` : vrai si le fichier existe et a une taille non nulle

`-L fichier` : vrai si le fichier existe et est un lien symbolique

`Fichier1 -nt fichier2` : vrai si le fichier1 est plus récent que le fichier2

`Fichier1 -ot fichier2` : vrai si le fichier1 est plus ancien que le fichier2

Test sur l'accès aux fichiers :

`-r fichier` : vrai si le fichier existe et est accessible en lecture (r)

`-w fichier` : vrai si le fichier existe et est accessible en écriture (w)

`-x fichier` : vrai si le fichier existe et est exécutable (x)

`-g fichier` : vrai si le fichier a le set-gid bit positionné

`-k fichier` : vrai si le fichier a le sticky-bit positionné

`-u fichier` : vrai si le fichier a le set-uid bit positionné

Comparaison algébrique

`chaîne1 -eq chaîne2` : égalité

`chaîne1 -ne chaîne2` : non égalité

`chaîne1 -gt chaîne2` : plus grand

`chaîne1 -ge chaîne2` : plus grand ou égal

`chaîne1 -lt chaîne2` : plus petit

`chaîne1 -le chaîne2` : plus petit ou égal

Comparaisons alphanumériques

chaîne1 = chaîne2 : égalité

chaîne1 != chaîne2 : différence

chaîne1 > chaîne2 : supérieure

chaîne1 < chaîne2 : inférieure

chaîne1 >= chaîne2 : supérieure ou égale

chaîne1 <= chaîne2 : inférieure ou égale

-n chaîne : chaîne n'est pas nulle (longueur > 0)

-z chaîne : chaîne est nulle (longueur = 0)

Composition d'expressions

! Expression : Vrai si Expression est fausse

Expression1 && Expression2 : ET logique

Expression1 || Expression2 : OU logique

7.2 - Structure de contrôle IF

```
if <test>
  then commandes2
fi
```

```
if <test>
  then commandes2
  else commandes3
fi
```

```
if <test>
  then commandes2
  elif commandes3
    then commandes4
    else commandes5
fi
```

Ainsi, pour tester si le fichier « file1 » est accessible en lecture par son propriétaire :

```
if test -r file1
   then echo "file1 est lisible"
   else echo "file1 n est pas lisible"
fi
```

Ou :

```
if [ -r file1 ]
   then echo "file1 est lisible"
   else echo "file1 n est pas lisible"
fi
```

7.3 - Structure de choix CASE

```
case chaine in
        motif1[|motif2] ... ) liste_commandes ;;
        …
esac
```

Remarque : Le caractère * est généralement mis en dernier du motif pour exprimer le cas par défaut.

Exemple

```
case $# in
    0) echo $0 n'a pas d'argument ;;
    1) echo $0 à 1 argument ;;
    2) echo $0 à 2 arguments ;;
    3) echo $0 à 3 arguments ;;
    *) echo $0 à plus de 3 arguments ;;
esac
```

8 - Les boucles

8.1 - La boucle for

```
for variable in liste
do
  commandes
done
```

Si la liste est omise, « variable » prend successivement par positionnement les paramètres passées au script (valeur de $@).

Si la liste est remplacée par *, variable prend successivement les références existantes dans le répertoire courant.

```
#!/bin/bash

for i in un deux trois quatre
do
  echo $i
done
```

```
[cd@drocourt ~]$ ./script.sh
un
deux
trois
quatre
```

```
#!/bin/bash

for i
do
  echo "Bonjour $i"
done
```

```
[cd@drocourt ~]$ ./script.sh aa bb
Bonjour aa
Bonjour bb
```

```
#!/bin/bash

for i in *
do
  echo $i
done
```

En supposant que ces trois fichiers sont dans le répertoire courant :

```
[cd@drocourt ~]$ ./script.sh
titi
tata
toto
```

8.2 - Boucle while et until

```
while test
do
  commandes
done
```

```
until test
do
  commandes
done
```

```
#!/bin/bash

# Essai de la boucle while
while [ $1 != fin ]
do
     echo $1
     shift
done
```

```
[cd@drocourt ~]$ ./script.sh 1 titi 5 fin toto
1
titi
5
```

8.3 - Ruptures de boucles

Il est possible de rompre le déroulement normal d'une boucle à l'aide des commandes « continue » et « break » :

- `break` : Va stopper la boucle en cours et se placer juste après le « done »,
- `continue` : Va interrompre la suite d'instructions restantes et se replacer dans le test conditionnel.

9 - Les fonctions.

Une fonction se déclare avec des parenthèses et des accolades comme en langage C, et on récupère les paramètres avec les arguments positionnels. Exemple :

```
#!/bin/bash

fonc1() {
  echo "coucou $1"
}

fonc1
fonc1 para
```

```
[cd@drocourt ~]$ ./script.sh
coucou
coucou para
```

Une fonction peut retourner un code de retour à l'aide de la commande « return », que l'appelant pourra consulter dans la variable « $? ».

10 - Variables locales, Variables globales.

Il n'y a pas de localité dans les variables de script, et toutes les variables sont donc globales. Exemple de script :

```
#!/bin/sh

variable1=3

mafunc() {
  echo "f-Valeur 1 : $variable1"
  variable1=4
  variable2=5
  echo "f-Valeur 1 : $variable1"
}

echo "Valeur 1 : $variable1"
echo "Valeur 2 : $variable2"
mafunc
echo "Valeur 1 : $variable1"
echo "Valeur 2 : $variable2"
```

Exécution :

```
[cd@drocourt ~]$ ./script.sh
Valeur 1 : 3
Valeur 2 :
f-Valeur 1 : 3
f-Valeur 1 : 4
Valeur 1 : 4
Valeur 2 : 5
```

Chapitre 5 - Gestion des processus et communication interprocessus

Table des matières

1 - Les Processus

1.1 - Priorité d'un processus

Calcul de la priorité d'un processus

Règle 1 : Les processus système sont en général plus prioritaires que les processus utilisateur,

Règle 2 : Plus un processus attend dans la table des processus, plus il devient prioritaire.

Remarque : Cette stratégie d'allocation des ressources de l'ordinateur ne permet pas de gérer le temps réel : elle est adaptée à une stratégie « temps partagé ».

La commande nice

Elle permet à tout utilisateur d'essayer de diminuer la priorité de ses processus et seulement à l'utilisateur « root » d'essayer d'augmenter celle de ses processus.

1.2 - Commande « ps »

La commande « ps » (process status) les caractéristiques essentielles des processus de l'utilisateur en cours.

```
[cd@drocourt ~]$ ps
  PID TTY          TIME CMD
 9080 pts/1    00:00:00 bash
10387 pts/1    00:00:00 ps
```

Par défaut, ne sont affichés QUE les processus de l'utilisateur, et qui ne sont descendants QUE du « shell » en cours d'exécution.

La commande « ps » issue de GNU implémente les options issues des deux familles d'Unix :

- BSD : Les options ne doivent pas être précédées du caractère « - » : ps ax,
- System V : Les options sont doivent être précédées du caractère « - » : ps -ef

Les autres options les plus utilisées sont :

- -e ou ax : tous les processus du serveur unix,
- -l, -f ou l : Affichage long,
- -eH ou axf : par arborescence,

```
[cd@drocourt ~]$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   500  9080  8504  0  80   0 - 29019 wait   pts/1    00:00:00 bash
0 R   500 10449  9080  0  80   0 - 28409 -      pts/1    00:00:00 ps
```

La colonne S montre l'état des processus : T pour stoppé, Z pour terminé (zombie), W pour attente, S pour sommeil en attente d'événement et R pour en cours d'exécution.

2 - Création de Processus

2.1 - Processus en avant plan

Un processus s'exécute par défaut en avant-plan et monopolise complètement son terminal d'attachement. Il peut être contrôlé au clavier par la frappe de certains caractères provoquant l'envoi de signaux au processus :

- <intr> : envoi du signal SlGlNT qui cause en principe la mort du processus,
- <quit> : envoi du signal SIGQUIT qui cause en principe la mort du processus avec création d'un fichier « core » contenant l'image mémoire du processus,

Exemple :

```
[cd@drocourt ~]$ sleep 10
(Attente de 10 secondes...)
[cd@drocourt ~]$
```

Le shell lance la commande et attend qu'elle soit achevée pour afficher de nouveau son invite : il s'agit du mode synchrone. Si l'utilisateur envoie le signal <intr> via <CTRL><C>, il tue le processus et l'invite du shell est ré-affichée.

Exemple :

```
[cd@drocourt ~]$ sleep 10
(Attente de 2 secondes...)
<CTRL><C>
[cd@drocourt ~]$
```

Remarques :

Les signaux <intr> et <quit> ne sont pas standardisés. Aussi, on utilise la commande « stty -a » pour retrouver leur valeur ainsi que toutes les caractéristiques du terminal.

Le comportement d'un processus à la réception d'un signal peut être programmé, ainsi certains processus ignorent les signaux SIGINT et SlGQUIT.

Remarques :

La commande « kill -INT <PID> » lancée sur un autre terminal aura le même effet sur le processus <PID> que la frappe du caractère <intr> sur le clavier du terminal de contrôle de ce processus (cf. suite).

Terminal 1	Terminal 2
[cd@drocourt ~]$* sleep 100 Complété *[cd@drocourt ~]$	*[cd@drocourt ~]$* ps a\|grep sleep 3098 pts/0 S+ 0:00 sleep 100 3151 pts/1 S+ 0:00 grep sleep *[cd@drocourt ~]$* kill 3098 *[cd@drocourt ~]$*

2.2 - Processus en arrière plan

Un processus qui s'exécute en arrière-plan ne monopolise plus son terminal d'attachement mais ne peut pas être contrôlé au clavier.

Exemple:

```
[cd@drocourt ~]$ sleep 10 &
[1] 4213
[cd@drocourt ~]$
```

Le « & » signifie que l'on a lancé la commande en arrière-plan et « 4125 » indique le PID de cette commande.

Le shell lance la commande et n'attend pas qu'elle soit achevée pour afficher de nouveau son invite : il s'agit du mode asynchrone.

2.3 - Le contrôle de jobs

Sans le contrôle de jobs, un processus vivant reste soit en avant plan soit en arrière plan. Avec le contrôle de jobs, un processus peut changer au cours de son existence.

Lorsqu'une commande (ou groupe de commandes) est lancée en arrière plan, le Shell lui associe un numéro de job et inscrit ce job dans une table que l'on peut consulter avec la commande « jobs ».

On peut alors faire passer le job en avant plan avec la commande « fg » (foreground) puis le remettre en arrière plan avec la commande « bg » (background) mais il faut auparavant le stopper sans le tuer : cela est réalisé en tapant au clavier le caractère <susp> qui cause l'envoi au job du signal SIGSTOP.

Pour rappel, on peut mettre en pause un processus par l'envoi d'un signal « STOP » ou en utilisant le raccourci clavier associé :

Terminal 1

```
[cd@drocourt ~]$ sleep 1000



[1]+ Arrêté        sleep 1000
[cd@drocourt ~]$
```

Terminal 2

```
[cd@drocourt ~]$ ps axl|grep sleep
0   500  ... S+   pts/1      0:00 sleep 1000
0   500  ... S+   pts/4      0:00 grep sleep
[cd@drocourt ~]$ kill -STOP 3548
[cd@drocourt ~]$ ps axl|grep sleep
0   500  ... T    pts/1      0:00 sleep 1000
0   500  ... S+   pts/4      0:00 grep sleep
[cd@drocourt ~]$ kill -CONT 3548
[cd@drocourt ~]$ ps axl|grep sleep
0   500  ... S    pts/1      0:00 sleep 1000
0   500  ... S+   pts/4      0:00 grep sleep
[cd@drocourt ~]$
```

```
[cd@drocourt ~]$ sleep 1000
^Z
[1]+  Arrêté                 sleep 1000
[cd@drocourt ~]$ ps x|grep sleep
 3329 pts/1    T      0:00 sleep 1000
 3331 pts/1    S+     0:00 grep --color=auto sleep
[cd@drocourt ~]$ jobs
[1]+  Arrêté                 sleep 1000
[cd@drocourt ~]$ fg %1
sleep 1000
^Z
[1]+  Arrêté                 sleep 1000
[cd@drocourt ~]$ ps xl|grep sleep
0   500  3329  2784  ... T    pts/1      0:00 sleep 1000
0   500  3333  2784  ... S+   pts/1      0:00 grep sleep
[cd@drocourt ~]$ bg %1
[1]+ sleep 1000 &
[cd@drocourt ~]$ ps xl|grep sleep
0   500  3329  2784  ... S    pts/1      0:00 sleep 1000
0   500  3335  2784  ... S+   pts/1      0:00 grep  sleep
[cd@drocourt ~]$
```

2.4 - Les threads

Un système d'exploitation moderne ne contient pas seulement des processus mais aussi des Threads. Pour se représenter ce qu'est un Thread, on peut imaginer qu'une même application s'exécute au même moment à plusieurs positions, via plusieurs pointeurs de programmes. Les Threads possèdent donc l'avantages de paralléliser facilement un problème puisqu'ils partagent nativement la même mémoire.

Il n'y a pas beaucoup d'interaction possible avec les Threads sur la ligne de commande si ce n'est de pouvoir les consulter à l'aide de certaines options de la commande « ps » :

- -eT ou axH : Voir tous les threads du système,
- -em ou axm : Voir le détail des threads par processus,

3 - Les signaux

3.1 - Définition

Le signaux sont des systèmes de communication disponibles dans les systèmes Unix. Les principaux sont les suivants :

- SIGHUP : Demande la relecture de la configuration,
- SIGINT : Interruption <intr>,
- SIGQUIT : inter.+ core <quit>,
- SIGKILL : fin non négociable du processus,
- SIGTERM : terminaison, envoyé par exemple lors de l'arrêt du système,

D'autres signaux sont utilisées par le contrôle de jobs :

- SIGSTOP : demande de suspension pour une reprise ultérieure
- SIGCONT : reprise du processus suspendu

Pour consulter la liste des signaux du système on utilise la commande « kill -l » :

```
[cd@drocourt ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS
```

Remarques :

Certains processus exécutant des shells-scripts ou des binaires exécutables sont protégés contre certains signaux ou traitent ces signaux de façon non standard.

Le signal SIGKILL est l'arme absolue contre un processus, car il ne peut être ignoré ni même traité par une fonction de déroutement.

3.2 - La commande kill

La commande « kill » permet de tuer un processus en lui envoyant le signal SIGTERM, signal envoyé par défaut, ou tout autre signal. Sa syntaxe est :

```
kill [-signo] pid …
```

où « signo » désigne le numéro ou le nom du signal. Il peut être exprimé sous la forme d'un numéro ou d'une chaîne de caractères.

Les trois exemples suivants sont car le signal 15 est celui par défaut.:

```
[cd@drocourt ~]$ kill 4013
...
[cd@drocourt ~]$ kill -15 4013
...
[cd@drocourt ~]$ kill -TERM 4013
```

3.3 - Gestion des signaux

La commande interne « trap » permet à un processus de spécifier quel type de comportement il souhaite avoir lorsqu'un signal particulier lui est adressé. « trap » sans argument donne la liste des captages déjà défini.

```
trap [argument] <liste de signaux>
```

Pour ignorer les signaux INT et QUIT :

```
[cd@drocourt ~]$ (trap '' INT QUIT;sleep 10)
^C^C^C
(attente 10 secondes ...)
[cd@drocourt ~]$
```

Si une commande est donnée, elle sera exécutée à la réception du signal :

```
[cd@drocourt ~]$ (trap 'echo "coucou"' INT QUIT;sleep 10)
^Ccoucou
[cd@drocourt ~]$
```

4 - Les flux

4.1 - Définition

Dans le système Unix, tout programme exécuté possède sa propre table de descripteurs de flux ouverts. Dans cette table trois flux standards existent par défaut :

- **stdin** : l'entrée standard de descripteur **0**, qui est par défaut le clavier,
- **stdout** : la sortie standard de descripteur **1** qui est par défaut l'écran,
- **stderr** : la sortie erreur standard de descripteur **2**, qui est par défaut également l'écran.

Ce comportement est modifiable et chacun de ses flux peut donc être redéfinit/redirigé :

- Par le programme lui même à tout moment de son exécution,
- Par le programme « père », c'est à dire le programme qui va exécuter la commande demandée.

4.2 - Flux ouverts

Les flux/fichiers ouverts par la suite possèdent donc des numéros de descripteurs supérieurs (3, puis 4, …).

Commande « lsof »

Il est possible de consulter les flux ouverts à l'aide de la commande « lsof » :

```
[cd@drocourt ~]$ lsof -a -p 9985
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF     NODE NAME
...
bash    9985 rahan    0u   CHR  136,1      0t0        4 /dev/pts/1
bash    9985 rahan    1u   CHR  136,1      0t0        4 /dev/pts/1
bash    9985 rahan    2u   CHR  136,1      0t0        4 /dev/pts/1
bash    9985 rahan  255u   CHR  136,1      0t0        4 /dev/pts/1
```

Répertoire « /proc »

Le répertoire « proc » offre un accès aux informations du noyau et du système, en particulier, il contient un répertoire par processus, dans lequel le répertoire « fd » renseigne sur les flux ouverts :

```
[cd@drocourt ~]$ ls -l /proc/$$/fd
total 0
lrwx------ 1 rahan rahan 64 nov.  28 18:03 0 -> /dev/pts/1
lrwx------ 1 rahan rahan 64 nov.  28 18:03 1 -> /dev/pts/1
lrwx------ 1 rahan rahan 64 nov.  28 18:03 2 -> /dev/pts/1
lrwx------ 1 rahan rahan 64 nov.  28 20:23 255 -> /dev/pts/1
```

4.3 - Les périphériques spéciaux

Ils existent plusieurs périphériques systèmes spéciaux (numéros majeurs 1) dont les plus importants sont :

- `/dev/null` : Périphérique de type entrée vers lequel il est possible de rediriger tout type de flux, qui sera donc supprimé,
- `/dev/zero` : Périphérique de type sortie à partir duquel il est possible de demander de l'information et qui fournira systématiquement des « 0 » binaires,
- `/dev/random` : Périphérique de type sortie à partir duquel il est possible de demander de l'information et qui fournira systématiquement des données aléatoires, pour générer ce type de données le système utilise la charge du système et d'autres indicateurs,
- `/dev/urandom` : Périphérique de type sortie à partir duquel il est possible de demander de l'information et qui fournira systématiquement des données aléatoires à partir d'un algorithme,

4.4 - Redirections

La redirection au niveau du code du programme ne nous intéresse pas ici puisqu'il s'agit de programmation, par contre les redirections en amont de l'exécution vont par exemple pouvoir être utilisées dans le « shell », ainsi le processus exécuté n'aura aucune idée de ces redirections.

Afin de réaliser les redirections sur la ligne de commande, les symboles « > » et « < » sont utilisés, dont voici les exemples les plus probants :

- **1> file1** : le flux de sortie standard est redirigé vers le fichier portant le nom « file1 », ainsi l'affichage qui devait être réalisé sur l'écran ne le sera plus, mais placé dans ce fichier,
- **> file2** : Idem car lorsqu'aucun flux n'est précisé, c'est la sortie standard qui est redirigée,

```
[cd@drocourt ~]$ cat /etc/passwd > /tmp/sortie.txt
```

- **2> file3** : idem mais pour la sortie d'erreur. En effet, les deux flux étant séparés et indépendants la redirection de l'un n'aura aucune influence sur l'autre,

```
[cd@drocourt ~]$ ls /mlkmlk 2> /tmp/erreur.txt
```

- **>> file4** : Le double « > » signifie qu'il faut ajouter au contenu existant du fichier, dans le cas contraire, le fichier est écrasé,
- **< file5** : Cette fois c'est l'entrée standard qui est redirigée et prise dans le contenu du fichier nommé « file5 »,
- **2>&1** : Dans le cas ou l'on souhaite rediriger à la fois la sortie standard (1) et la sortie d'erreur (2) dans le même flux, il est préférable de l'indiquer d'une manière spécifique, comme ici ou il est indiqué que pour le flux de descripteur 2 il faut faire de même que pour le flux de descripteur 1,

```
[cd@drocourt ~]$ ls -R /etc > out.txt 2>&1
[cd@drocourt ~]$
```

- **<< word** : associe le descripteur d'entrée avec le fichier du programme en cours, à partir de la ligne qui suit cette déclaration jusqu'à la ligne précédant celle débutant par le terme « word ». Cela permet d'inclure dans des programmes des zones de texte à utiliser ou à recopier dans d'autres fichiers au moment de l'exécution. La commande redirigée de cette manière utilise comme entrée standard les lignes qui suivent immédiatement dans le programme, jusqu'au terme « word ».

```
[cd@drocourt ~]$ cat << EOF > fichier.txt
> bonjour
> ceci est le contenu du fichier
> EOF
[cd@drocourt ~]$ cat fichier.txt
bonjour
ceci est le contenu du fichier
[cd@drocourt ~]$
```

4.5 - Redirection permanente

La commande « exec » permet également de manipuler les descripteurs de manière permanente.

```
[cd@drocourt ~]$ exec 1>sortie.txt
[cd@drocourt ~]$ ls
[cd@drocourt ~]$ lslslsls
lslslsls : commande introuvable
[cd@drocourt ~]$ cat sortie.txt
cat: sortie.txt : le fichier d'entrée est aussi celui de
sortie
[cd@drocourt ~]$ cat sortie.txt 1>&2
bin
Desktop
Documents
Downloads
```

Il est possible de sauvegarder le flux avant la redirection pour le restaurer :

```
[cd@drocourt ~]$ exec 6>&1
[cd@drocourt ~]$ exec > file.log
[cd@drocourt ~]$ echo coucou
[cd@drocourt ~]$ ls
[cd@drocourt ~]$ ps
[cd@drocourt ~]$ exec 1>&6
[cd@drocourt ~]$ ls
 bin       Documents     lost+found    Photos       Desktop
Downloads   Musique Public    Videos
```

Pour fermer le flux qui n'est plus utilisé, il faut utiliser le descripteur « &- » :

```
[cd@drocourt ~]$ exec 6>&-
```

4.6 - Les tubes nommés

Un tube nommé fonctionne comme un tube, à savoir qu'il y a un écrivain et un lecteur, mais il utilise le système de fichier. On peut le créer avec la commande « mkfifo » :

```
[cd@drocourt ~]$ mkfifo tube
[cd@drocourt ~]$ ls -l tube
prw-rw-r-- 1 cd cd 0 déc.   2 09:03 tube
[cd@drocourt ~]$ cat < tube
```

Dans un autre terminal :

```
[cd@drocourt ~]$ echo "coucou" > tube
```

5 - IPC « System V »

Les IPC sont des mécanismes de communication inter-processus signifiants « Inter Processus Communication System V », ils sont composés de :

- Files de messages,
- Segments de mémoire partagée,
- Sémaphores,

5.1 - Caractéristiques communes

Le système gère une table spécifique pour chaque type d'I.P.C. System V. Chaque objet possède une **identification interne** (nombre entier naturel) dont la connaissance est indispensable pour accéder à l'objet ; cette identification est obtenue soit par héritage soit par interrogation du système à l'aide de différentes primitives.

5.2 - Commandes "ipcs"

La commande « `ipcs` » permet la consultation des trois tables d'I.P.C. du système à l'aide des options suivantes :

- « -s » : pour les sémaphores,
- « -q » : pour les files de messages,
- « -m » : pour les segments de mémoire partagé,

De plus, les options suivantes peuvent être intéressantes :

- « `-l` » : donne les limites imposées par le système,
- « `-u` » donne la consommation actuelle.

5.3 - Commande "ipcrm"

Logiquement, les IPS sont créés par les processus via des langages de programmation évolués (et non en Shell). Néanmoins, il peut être pratique de pouvoir supprimer un IPC qui n'aurait pas été correctement supprimé par un programme.

Pour cela on va utiliser la commande « ipcrm », en précisant le type d'IPC ainsi que le numéro de l'objet concerné :

- -s <semid> : pour supprimer un sémaphore,
- -q <msqid> : pour supprimer une file de message,
- -m <shmid> :pour supprimer un segment de mémoire partagé,

6 - Les sockets

6.1 - Introduction

Les sockets sont un autre mode de communication inter-processus sous Unix, on peut les séparer en deux catégories :

- **Socket Unix** : Qui permettent une communication entre des processus d'un même système,
- **Socket Réseaux** : Qui permettent une communication entre processus/application de machines distantes,

Pour consulter les « sockets » actives, on utilisera dans les deux cas la même commande « **netstat** » mais avec des options différentes. Il existe deux types de Sockets :

- **Mode paquet** (DGRAM) : La communication se passe par l'envoi et le réception de paquets,
- **Mode connecté** (STREAM) : Les applications sont connectés et échangent au travers cette connexion.

6.2 - Les Sockets Unix

Elles sont énormément utilisés par les applications sous Unix, pour s'en rendre compte, on peut utiliser l'option « --unix » de la commande « netstat » pour visualiser toutes les Sockets de ce type. Attention, les informations détaillées ne s'affichent que pour les l'utilisateur propriétaire, ou dans le cas la commande est exécutée avec « root » :

```
[cd@drocourt ~]$ netstat --unix
Sockets du domaine UNIX actives (sans serveurs)
Proto RefCnt Flags Type    State     I-Node   Chemin
...
unix  2      [ ]   DGRAM             41127    /run/user/500/systemd/notify
unix  2      [ ]   DGRAM             28144    /run/user/120/systemd/notify
unix  2      [ ]   DGRAM             1012005  /run/wpa_supplicant/p2p-dev-wlp2s0
unix  3      [ ]   DGRAM             19855    /run/systemd/notify
unix  22     [ ]   DGRAM             19864    /run/systemd/journal/dev-log
unix  2      [ ]   DGRAM             19866    /run/systemd/journal/syslog
unix  9      [ ]   DGRAM             19878    /run/systemd/journal/socket
unix  2      [ ]   DGRAM             1014859  /run/wpa_supplicant/wlp2s0
unix  2      [ ]   DGRAM             22367
...
unix  3      [ ]   STREAM  CONNECTE  1217016
unix  3      [ ]   STREAM  CONNECTE  440701
...
```

On peut distinguer un Socket dans le système de fichier à l'aide de la commande « ls » par le type positionné sur « s » :

```
[cd@drocourt ~]$ ls -l /run/user/500/systemd/notify
srwxrwxr-x 1 cd cd 0 déc. 1 12:34 /run/user/500/.../notify
```

6.3 - Les Sockets réseaux

Ce sont les méthodes de communication utilisées pour dialoguer sur le réseau, et plus spécifiquement sur Internet. Elles ne sont pas spécifiques à Unix car tous les systèmes d'exploitation ont implémenté ce type d'objet. Pour les visualiser on utilisera l'option « --inet » de la commande « netstat », à laquelle on ajoute l'option « -a » (All) pour visualiser également celles pour lesquelles il n'y a pas encore de connexion :

```
[cd@drocourt ~]$ netstat -a --inet
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale          Adresse distante        Etat
tcp        0      0 127.0.0.53:domain       0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 xps:59550               imap.serv.u13.org:imap2 ESTABLISHED
tcp        0      0 xps:56488               secure-sv.u13.org:https TIME_WAIT
...
udp        0      0 224.0.0.251:mdns        0.0.0.0:*
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp        0      0 127.0.0.53:domain       0.0.0.0:*
udp        0      0 0.0.0.0:ipp             0.0.0.0:*
...
```

7 - Les RPC

Les RPC ou « Remote Procedure Call », appelés maintenant « Open Network Computing Remote Procedure Call » (ONC-RPC) sont des mécanismes de communication entre processus de différentes machines, et utilisent les protocoles de transport réseau standards, à savoir UDP ou TCP.

Le principe général est :

- L'utilisation d'une méthode d'échange de donnée standardisée appelée XDR pour s'affranchir de la problématique de la représentation des données,
- L'utilisation d'un service d'annuaire pour s'affranchir de fixer des ports par service de manière stricte,

Les premières versions de NFS utilisent les RPC ainsi que le système d'authentification centralisé NIS.

Annexes

Table des matières

1 - Les ACLs

1. Les ACLs doivent être activées sur le « filesystem », une vérification est possible avec la commande « tune2fs -l »,
2. Si ce n'est pas le cas, l'option « acl » doit être ajoutée dans le fichier « /etc/fstab »,
3. Pour positioner les droits des ACLs, on utilise la commande « setfacl ». Par exemple, pour positionner les droits « rw- » pour l'utilisateur « yumssh » :

```
[~]# setfacl -m u:yumssh:rw- file
```

4. L'ajout des ACLs est visible sur le fichier par un signe « + » :

```
[~]# ls -l file
-rw-r--r--+  1 root root       5640 sept. 10 10:48 file
```

5. Pour consulter les ACLs on utilise la commande « getfacl » :

```
[~]# getfacl file
# file: file
# owner: root
# group: root
user::rw-
user:yumssh:rw-
group::r--
mask::r--
other::r--
```

2 - Les commandes POSIX

Voici la liste des commandes standardisées POSIX et préconisées dans un système respectant cette norme :

admin	cd	ctags	ex
alias	cflow	cut	expand
ar	chgrp	cxref	expr
asa	chmod	date	false
at	chown	dd	fc
awk	cksum	delta	fg
basename	cmp	df	file
batch	comm	diff	find
bc	command	dirname	fold
bg	compress	du	fort77
c99	cp	echo	fuser
cal	crontab	ed	gencat
cat	csplit	env	get

getconf	locale	newgrp	qalter
getopts	localedef	nice	qdel
grep	logger	nl	qhold
hash	logname	nm	qmove
head	lp	nohup	qmsg
iconv	ls	od	qrerun
id	m4	paste	qrls
ipcrm	mailx	patch	qselect
ipcs	make	pathchk	qsig
jobs	man	pax	qstat
join	mesg	pr	qsub
kill	mkdir	printf	read
lex	mkfifo	prs	renice
link	more	ps	rm
ln	mv	pwd	rmdel

rmdir	tail	ulimit	uustat
sact	talk	umask	uux
sccs	tee	unalias	val
sed	test	uname	vi
sh	time	uncompress	wait
sleep	touch	unexpand	wc
sort	tput	unget	what
split	tr	uniq	who
strings	true	unlink	write
strip	tsort	uucp	xargs
stty	tty	uudecode	yacc
tabs	type	uuencode	zcat