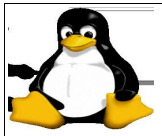
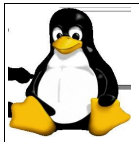


# Formation Linux : rappels historiques

**1969 : Premier compilateur C, sur DEC PDP11**
**1970 : Premier système d'exploitation Unix (Université de Berkeley/Bell labs)**
**1974 : Premier micro-ordinateur (REE-MICRAL)**
**1980 : HP, DEC et Bull sont les 3 premiers vendeurs de stations Unix (HPUX, Ultrix/VMS et Spix)**
**1983 : Apparition de Sun et SunOS, et SCO sur PC AT 286 !**
**1984 : IBM entre sur ce marché (AIX)**
**1985 : Appollo entre sur ce marché, avec une offre fortement typée RESEAU (DNS)**
**1986 : Windows 2 sort des cartons de Microsoft**
**1987 : ATT <achète> Unix, et se crée 2 groupes : OSF et Archer.**
**1989 : Windows 3.11 s'impose tranquillement**
**1991 : Linus Torwalds lance son Unix PC GRATUIT**
**1997 : IBM migre AIX vers Linux**
**1998 : HP migre HPUX vers Linux, IBM porte Linux sur AS400**
**1999 : Linux dépasse NT4 sur le marché des serveurs professionnels**
**2003 : A ce jour, vous disposez d'une foultitude de distributions, de noyaux, de logiciels**



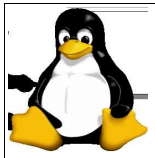
# Formation Linux : rappels historiques

**En fait, ce qui a motivé la construction d'un unix PC (et autres plateformes) LIBRE, c'est :**

- **La privatisation par ATT**
- **La mouvance GNU**
- **la mouvance freeware/shareware (logiciels libres)**

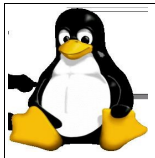
**Ce qui a fait le succès de Linux, c'est :**

- **L'aspect portabilité multi-plateforme**
- **La distribution libre et l'open source**
- **Le manque de fiabilité de Windows dans les années 90**
- **Chacun peut construire son système.**
- **Il reste (en 2003) insensible aux virus.**
- **Le projet Linux, géré par des milliers de personnes, est *très* organisé**



# Formation Linux : Terminologie

**GNU : Gnu is Not Unix : Premier concept d'unix non propriétaire**
**GPL : Gnu Public License : Licence contractuelle de distribution libre de logiciels, gérée par la Free Software Foundation (FSF)**
**Opensource : mot général décrivant tout logiciel livré avec ses sources, gratuitement.**
**JLx : Journées du Libre de x : Congrés généralement annuel des acteurs du logiciel libre dans la ville x**
**LUG : Linux User's group. Le plus près étant le LUG de Lyon.**
**Installparty : Soirée conviviale organisée par un LUG pour installer et configurer vos PC sous linux**
**Noyau : Ensemble cohérent de sources, et de binaires associés, vérifiés et validés par la communauté Linux**
**LDP : Linux Documentation Project : Ensemble du projet de documentation normalisée.**
**Filesystem : système de fichier : organisation logique des données dans la partition du bios.**



# Formation Linux : Généralités

**Linux est un système entièrement basé sur Unix. C'est à dire :**

- **il est multi-postes**

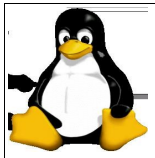
**Historiquement, les postes se connectaient par port série. Maintenant, ils accèdent aux ressources de la machine par le réseau (lan/wan)**

- **il est multi-utilisateurs**

**Naturellement, Unix a été conçu pour gérer plusieurs utilisateurs. En même temps et dans le temps.**

- **il est multi-tâches**

**Dès sa 1ere version, Unix fut un système partageant le temps entre plusieurs processus (ce que d'autres décrivent comme des threads). Depuis, linux est multi-tâche et mutli-processus. De plus, il fournit une interface *normalisée* de communication entre "activités"**



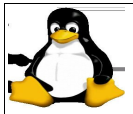
# Formation Linux : Le système installé sur disque

**Un système Unix/linux nécessite :**

**Un support bootable et partitionnable (même disquette), en accès RW**
**Un ensemble de répertoires systèmes sur ce support (filesystem)**
**Un noyau accessible (logiciel à part entière) dans ce filesystem, donc un pilote associé au type de filesystem**
**Une librairie (glibc) accessible sur ce filesystem.**

**Communément, Linux attend :**

**Un support contenant un bootstrap (dit /boot),**
**soit disquette, zip, ethernet, disque dur, cdrom**
**Un disque dur contenant le reste (dit /, ou root)**
**sur les mêmes supports accessibles en RW**



# Formation Linux : Ce que comprend le sysème installé

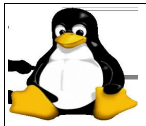
**A ce jour, le système nécessite**

**une partition bootable, /boot avec le noyau vmlinuz (autrefois /vmlinuz)**
**une partition montable en rw, /**

**Dans /, il faut un ensemble de répertoires systèmes**

- **bin, (*)**
- **boot, (*), qui est en fait la partition /boot montée sous /root/boot**
- **etc, (*)**
- **lib, (*)**
- **home,**
- **opt,**
- **usr, (*)**
- **sbin, (*)**
- **tmp,**
- **var,**

**La présence de certains est impérative (*), et pour d'autre, préférable.**



# Formation Linux : Les répertoires systèmes

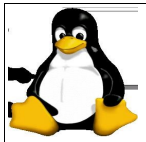
**/dev contient l'ensemble des accès aux pilotes de périphériques. Ce sont, en fait, des fichiers qui décrivent la manière d'accéder aux pilotes. Par exemple :**

**/dev/fd0h1440 est un fichier qui dit au système que le périphérique associé à ce nom là est de type 2, indice 56. L'indice 56 est un numéro d'index dans une table de drivers du noyau installé. Le même /dev/fd0u1440 concerne le même périphérique, matériellement parlant, mais une manière d'y accéder différente.**

**Autre exemple :**

**/dev/hdb pointe sur le début du disque ide0, esclave.**
**/dev/hda1 est la premiere partition du disque ide0, maître.**
**/dev/hda7 est la 7eme partition, ou unité logique, du disque ide0 maître.**
**/dev/hdc, généralement le cdrom sur un PC mono-disque.**
**/dev/lp0, 1er dispositif sur le port //**

**De la même manière, /dev/null indique un driver qui "jette" ce qui lui arrive. Et /dev/mouse pointe sur un pilote de la souris, ou un autre driver /dev, comme /dev/psaux.**



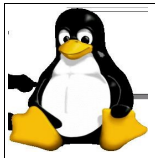
# Formation Linux : Les répertoires systèmes

**/etc contient l'ensemble des fichiers importants pour la gestion de la configuration installée. Par exemple :**

**/etc/xf86config décrit ce qui est nécessaire à la gestion du serveur X,**
**/etc/inetd.conf est le fichier qui indique les réseaux installés,**
**/etc/hosts décrit les noms des machines (en lieu et place des adresses IP),**
**/etc/rc.d est un répertoire qui contient ce qu'il faut faire lors du démarrage,**
**/etc/passwd est un fichier qui contient l'ensemble des noms d'utilisateurs autorisés**
**etc... (ce qui a donné le nom de ce répertoire)**

**Lorsque Unix/Linux démarre, ces fichiers sont exploités pour finir le boot. De plus, durant la vie du système booté, certains de ces fichiers seront lus (sur un accès réseau, le login d'un utilisateur, le changement de répertoire d'un utilisateur, ...).**

**De plus, /etc contient les fichiers de configuration par défaut, lorsque ils sont livrés, ou lorsque Unix n'en trouve pas. Par exemple, /etc/d.profile est le fichier s'exécutant par défaut sur l'arrivée d'un utilisateur (sans compter ceux de ce répertoire principal).**



# Formation Linux : Les répertoires systèmes

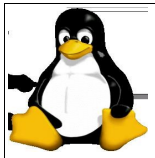
**/lib est le répertoire principal d'accès aux librairies, dont l'indispensable glibc. Bien entendu, des librairies peuvent se trouver ailleurs, mais, par défaut, Unix va chercher dans /lib au boot (pour la glibc), et le compilateur c natif aussi. Par contre, la variable d'environnement LD_LIBRAY_PATH peut pointer sur un ensemble de répertoires contenant d'autres lib.**

**/usr est (historiquement), le répertoire des users, qui devrait contenir ce dont le système n'a pas fondamentalement besoin. Dans le temps, /usr a évolué et est devenu un répertoire système où l'on ajoute les logiciels non indispensables, l'environnement graphique, des librairies, les jeux, etc...**
**/usr contient donc /usr/lib, /usr/bin, /usr/X11, /usr/local, ...**

**/bin contient l'ensemble des commandes accessible à tous.**

**/sbin contient l'ensemble des commandes réservées à l'administrateur.**

**/opt, /var sont des répertoires pour spécialiser les installations. Ils sont destinés à recevoir les logiciels applicatifs (/opt) et/ou livrés (/var)**



# Formation Linux : Les répertoires remarquables

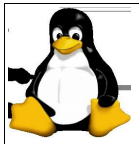
**/home est, dorénavant, destiné à contenir les dossiers et fichiers des utilisateurs. La norme POSIX garantit qu'une (ré)installation ne les touchera pas.**

**/usr/src contient les sources des logiciels installés (cf GPL). Par exemple, ceux de Linux ou des paquetages.**

**/proc, sur des machines bootant un noyau >= 2.2, contient les ressources systèmes utilisées, comme la quantité de mémoire, le nombre de processus, ....**

**/root est le répertoire de l'utilisateur root (administrateur).**

**/mnt/xxx est le répertoire qui va pointer vers les systèmes de fichiers montés. Par exemple, /mnt/windows/C, /mnt/cdrom, ....**



# Formation Linux : Tout est fichier

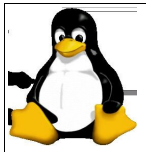
**Sur Unix, tout est fichier. Un fichier est un point d'entrée sur des données, un périphérique, un processus, une file de messages, des ressources systèmes, ...**

## Un fichier est donc :

- **une collection de données (fichier au sens commun, datas, ...)**
- **un répertoire (c'est un fichier, point d'entrée vers la liste des fichiers)**
- **un lien vers un fichier**
- **un périphérique (puisque un point d'entrée dans /dev)**

## Un fichier, quel qu'il soit, est :

- **un nom (de taille maximale non limitée)**
- **un répertoire d'accueil (donc un fichier qui lui même, ...)**
- **un propriétaire (par défaut le créateur)**
- **des droits d'accès**



# Formation Linux : Tout est fichier

**Un fichier est contenu dans un répertoire, lequel est dans un filesystem, lequel est vu par un point de montage.**

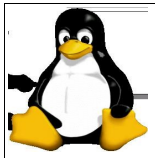
**Linux sait gérer de nombreux filesystem différents (ext2 naturellement), mais aussi est3, minix, sco, hpfs, ntfs, nfs, fat, iso9660.**

**Par exemple, si je dispose d'une machine avec 1 disque, 2 partitions (windows FAT et linux), et un disque sur le réseau, j'aurai :**

**/ (la partition linux), montée par le boot**
**/boot, une autre partition linux montée au boot**
**/swap (une partition montée, mais invisible)**
**/windows ( de type fat)**
**/mnt/reseau_entreprise, de type nfs.**

**Monter un filesystem dans un répertoire ne perd pas les données déjà présentes sur /. Un**

# redémontage le prouve



# Formation Linux : Les utilisateurs

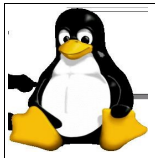
**Un utilisateur, sur une machine linux, est décrit par une ligne dans le fichier /etc/passwd. Il peut se loger plusieurs fois sur la même machine, depuis n'importe quel port (série, //, réseau, usb, ...)**

**Un fichier de démarrage indique à Unix quels sont les ports susceptibles de recevoir des connexions, et de les vérifier (/etc/inittab).**

**L'utilisateur passe obligatoirement par le login, qui demande le nom, le mot de passe, et vérifie la cohérence de l'ensemble.**

**Un utilisateur d'une machine dispose d'un répertoire d'accueil et d'un programme à lancer au login. Il appartient à un groupe d'utilisateurs, ce qui définit ses droits d'accès au reste de la machine.**

**Lorsque l'utilisateur sera logé, Unix va lancer des fichiers de configuration (profile, login, ...) qu'il trouve dans /etc et dans le répertoire d'accueil (home directory).**
**Aucun programme ne peut se lancer si il n'y a pas de 'user' attaché.**



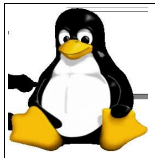
# Formation Linux : Les utilisateurs

**Lorsque un utilisateur est logé, il dispose d'un environnement :**

- **ce sont des variables que chaque programme lancé peut explorer**
  **elles sont héritables d'un processus à l'autre**
- **ce sont des périphériques attachés (mais pas exclusivement)**
- **ce sont des droits et privilèges décrits dans /etc/passwd, et hérités par le groupe**
- **qui n'occulte en rien l'environnement des autres utilisateurs**

- **Il ne consomme des ressources que lorsque il en a besoin**
- **Il n'est pas prioritaire sur le système (Unix, dans tous les cas, garde la main)**

**A chaque action qu'un programme réalise vers l'extérieur de son espace adressable (accès fichier, donc périphérique, data, ...) Unix vérifie la conformité des droits avec ceux attendus sur la ressource désirée.**



# Formation Linux : Les groupes

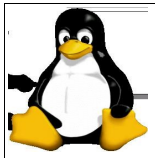
**Un groupe est une entité de regroupement des utilisateurs. Un groupe peut être vide. Un utilisateur doit appartenir à, au moins, un groupe (son groupe par défaut).**

**Lorsque un utilisateur accède à une ressource, Unix détermine la catégorie (propriétaire, même groupe ou reste du monde), et vérifie que cette dernière à les droits sur la ressource.**

**Donc, un fichier est protégé pour les trois catégories d'individus :**

**Le propriétaire U (créateur au départ) : RWX**
**Le groupe du propriétaire G : RWX**
**Le reste du monde O : RWX.**

**Par exemple, le fichier /home/albert/mesdatas/compta.sdc est protégé en rw-r--r-- albert g_compta ce qui veut dire que tout le monde peut lire, mais seul albert peut écrire. Et si le fichier /home/albert/pub/creebilan.exe est protégé comme rwxrwxr-x albert g_compta, tout le monde peut l'exécuter (donc le lire), mais seul albert et les membres de son groupe peuvent le recompiler.**



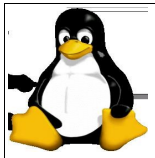
# Formation Linux : Les processus

**Le noyau Unix a pour fonction principale de gérer les processus. Ce sont des programmes compilés/linkés lancé par les utilisateurs (ou le système). Un processus dispose de ressources (mémoire pour stocker le code, les datas, les fichiers ouverts, l'environnement). Il a des droits (ceux de l'utilisateur du lancement).**

**Il peut créer des tâches (internes au processus) ou lancer d'autres processus (ses fils). Ces derniers récupèrent alors tout l'environnement du père (droits, mémoire, ...)**

**Il peut (et toute la gestion d'Unix se trouve là) recevoir des signaux.**

**Unix se réserve toujours le droit de détruire le processus (sur signal), de le swapper, ...Le noyau reste toujours maître du jeu.**



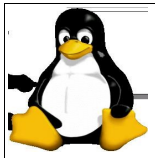
# Formation Linux : Les processus

**Pour un utilisateur donné, le premier processus (généralement, mais non obligatoire), c'est le shell (langage de commande). Celui-ci est un programme qui lit le clavier et exécute les ordres de l'utilisateur.**

**Lorsque le shell interprête une commande, il crée un fils, identique à lui (fork()). Il y a recopie des espaces de datas (donc héritage). Puis, indique à ce fils de changer de programme (execl()). Et enfin, attend la fin du fils (wait()).**

**Cette démarche est immuable depuis 1970, et a été reprise par POSIX tellement elle est simple et fiable.**

**En fait, le shell est un fils de login, qui, lui même est un fils d'init. Init est le premier processus lancé au démarrage de la machine, par le noyau.**



# Formation Linux : Les fichiers

**Un fichier est caractérisé par :**

**un type (d, -, b, c, p, s, l...)**
**un propriétaire**
**un groupe**
**des droits**
**un chemin**
**un nom (tous les caractères sont permis, l'espace, l'étoile, ...), *Unix est case_sensitive !***
**des dates**

```
-rwxr-xr-x    1 michel   users       14503 aoû  9 05:50 cesar
-rw-r--r--    1 michel   users         854 aoû  9 05:50 cesar.c
drwxr-xr-x    5 michel   users        4096 jun  1 08:51 cle
-rw-r--r--    1 michel   users         100 aoû 22 11:40 cookies
-rw-r--r--    1 michel   users     4504631 aoû  9 10:42 descent3-1.4.0b-x86.run
```

**Il n'y a pas d'extension par défaut. Ce n'est pas parce-qu'un fichier possède le droit x que c'est un programme.**

**Par exemple, .sh est consensuellement reconnu comme un fichier de commandes du shell. Mais .exe peut n'être qu'un fichier excel ou un répertoire.**

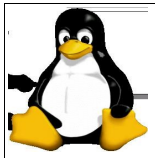
**.tar, .gz, .tgz sont, généralement, des fichiers archives ou compressés.**

**.c, .o, .a sont des sources, objets et librairies (.a est un archive de .o), mais cette convention est plus liée au compilateur C (gcc ou autre) qu'à Unix.**

**.rpm est une extension associée à la distribution linux RedHat, qui fournit un gestionnaire de paquetage.**

**Un fichier peut avoir n'importe quelle taille (le filesystem peut même être rempli à 105%).**

**Unix gère les répertoires comme des fichiers, les devices comme des fichiers (certes spéciaux). Il suffit d'aller voir le contenu de /dev.**



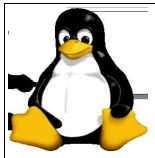
# Formation Linux : Les répertoires

**Un répertoire peut contenir autant de fichiers que l'on veut (et donc de répertoires). L'arborescence ainsi créée n'est pas limitée.**

**Les droits d'accès en r sur un répertoire indique que l'on peut lire le contenu. Le droit en w autorise à créer, supprimer des fichiers (donc modifier la liste), et en x pour se déplacer.**

**Les répertoires '.' et '..' sont des noms réservés. En fait, ils représentent le répertoire (lui-même) et son père (celui dans lequel il est contenu). Ce qui permet de faire des accès relatifs sans devoir systématiquement nommer depuis /.**

**Lorsque l'on accède à un fichier d'un répertoire, Unix vérifie que *TOUT* le chemin autorise l'accès. Et pas seulement le répertoire contenant.**



# Formation Linux : Les fichiers

**Un fichier l est un lien vers un autre fichier. Il peut ainsi se construire tout un tas de ponts entre des fichiers ou des répertoires. Jouer sur les liens permet souvent de modifier les droits d'accès et de simplifier les chemins. C'est Unix qui traverse les liens lorsque l'on accède au fichier lien.**

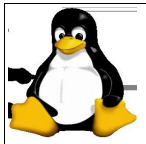
**Par exemple :**

```
lrwxrwxr--   compta_gene.annee_courante -> datas/compta03.sdc
```

**dans datas :**

```
-r--r--r--   compta01.sdc
-r--r--r--   compta02.sdc
-r--r--r--   compta03.sdc
```

**Nous voyons que le lien est modifiable, mais le fichier pointé ne l'est pas. Si je détruis `compta_gene.anne_courante`, c'est le lien qui est cassé, pas le fichier pointé (et, de toutes façon, je ne peux le faire ici).**



# Formation Linux : Le shell

**Son nom vient du fait qu'Unix, au départ, fut conçu comme un kernel (noyau) plus une coquille (langage de commande au dessus).**

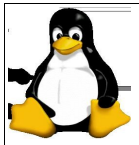
**Il existe plein de shells sous plein d'Unix(es). Le plus naturel sous Linux est le bash. C'est un programme comme les autres. Sa seule fonction est de comprendre une ligne de commande, et de lancer les commandes (qui sont des programmes exécutables aussi bien que des fichiers dans lequel se trouvent des commandes shell (script)).**

**Le shell analyse toute la ligne avant d'exécuter. Il la modifie selon des règles précises, (qui font toute sa puissance), puis recherche le premier mot comme commande et la lance, en lui passant en argument (argc, argv[] pour les C_istes, Command() pour les VB_istes) le reste de la ligne qu'il a modifié.**

**Bien évidemment, pour lui, un espace ou une <tab> sépare 2 mots (comme d'autres caractères spéciaux pour le shell).**

**Le shell est un langage de commande encore inégalé en terme de puissance. Certes, sa prise en**

**main n'est pas aisée.**



# Formation Linux : Le Shell

**Tout d'abord, le principe de fonctionnement. Le shell lit une ligne, analyse et remplace, et cherche à exécuter la commande dont le nom est le premier mot.**

**Il utilise pour cela des variables d'environnement et des méta-caractères.**

**Les métas sont (entre autres) :**

***, qu'il remplace par les noms de fichiers possibles.**
**$, qui indique que le mot suivant doit être une variable**
**;, pour signaler qu'il faut comprendre la commande comme 2 sous commandes**
**&, pour indiquer (à la fin d'une commande) qu'elle s'exécute en tâche de fond.**
**", pour forcer le shell à considérer ce qui est entre 2 " comme un seul mot.**

**Et bien d'autres encore, que l'on verra peu à peu.**

**Le shell affiche un prompt, généralement $ ou #. Tous les exemples qui suivent commenceront par $>, pour montrer le début de la commande.**

**$> ls**

**Liste le contenu du répertoire courant, trié par ordre lexical.**

**$> cd /home/michel**

**Se place dans le répertoire `michel` qui devient courant**

**$> pwd**

**Affiche sur l'écran le chemin complet du répertoire courant**

**$> more mon_fichier**

**Affiche à l'écran le fichier `mon_fichier` du répertoire courant, page à page.**

**$> echo "coucou, c'est moi"**

# Affiche ce qui est entre les " sur l'écran.

```
$> ls *.p
```

**Liste tous les fichiers dont le nom finissent par "`.p`" du répertoire courant**

```
$> ls */[AaEeIiOoUuYy]*
```

**Liste tous les fichiers des sous-répertoires dont le nom commence par une voyelle**

**Dans nos exemples, `ls` est une commande livrée dans Unix. Mais cela pourrait être n'importe quel programme.**

**Le shell, par défaut, utilise 3 fichiers : stdin, stdout, stderr. Stdout et stderr pointent sur le même périphérique, mais, par convention, tous les programmes qui veulent écrire un message d'erreur le font sur stderr. Cela permet de distinguer, lors des redirections.**

```
$> ls *.pas >liste_pas
```

Le shell crée (ou écrase) le fichier liste_pas dans le répertoire courant, puis calcule *.pas, et lance ls, en détournant la sortie standard stdout de ce dernier sur le fichier. Sans que ls ne modifie sont comportement (puisque il n'écrit plus à l'écran mais sur disque).

```
$> ls *.simian >liste_simian 2>erreur_liste
```

Là encore, le shell va créer les 2 fichiers, lancer ls en lui passant le résultat du remplacement de *.simian, et détournant stdout et stderr. Si il n'y a pas de fichier .simian, ls écrira un message d'erreur sur le 2eme fichier. Cela permet de filtrer les erreurs d'un côté et les messages applicatifs de l'autre.

Il existe aussi les commandes pipées, enchaînées par un |. Le shell crée les processus associés à chaque commande, et détourne la sortie du premier sur l'entrée du second.

Par exemple, wc est une commande qui calcule le nombre d'octets, de mots, de lignes de son

**entrée stdin, et cat est une commande qui affiche le contenu des fichiers.**

```
$> cat *.pas | wc -l
```

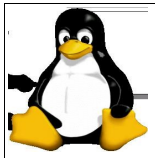
**Le shell détermine *.pas (remplace par tous les noms de fichiers pascals du répertoire, détourne son stdout, lance cat et lui passe cette liste, remet son stdout, détourne son stdin, lance wc, remet son stdin, et attend la fin des 2 processus.**

**Sans le savoir, cat affiche le contenu sur stdout, mais pas à l'écran, sur le stdin de wc. Sans le savoir, wc lit son stdin et cumule les \n. Il affiche alors sur stdout le résultat, qui est le nombre de ligne de tous les fichiers sources pascal d'un répertoire. C'est simple, et pas facile à faire dans d'autres environnements.**

**Supposons que l'on dispose d'un makefile qui compile pendant plus d'une heure. Des commentaires sont placés dans make_erreur (notamment une ligne par fichier). On peut alors :**

```
$> cat make_erreur | grep COMPILATION | wc -l
```

**Ici, grep lit sur le fichier stdin, et cherche et affiche les lignes contenant le mot COMPILATION. Sans le savoir, cat écrit sur l'entrée de grep qui affiche sur l'entrée de wc.**



# Formation Linux : Le Shell

**Le shell sait manipuler des variables. Elles existent jusqu'à la fin du processus. Généralement, ce sont des variables non numériques (mais bash sait faire).**

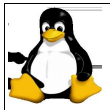
```
$> pwd
/home/chez_moi
$> REP=/home/chez_toi
$> echo $REP
/home/chez_toi
$> cd $REP
$> pwd
/home/chez_toi
```

```
$> PWD=`pwd`
$> echo $PWD
/home/chez_toi
$> cd ../chez_moi
$> echo $PWD
```

```
$> PWD=pwd
$> $PWD
/home/chez_toi
$> cd ../chez_moi
$> $PWD
```

**/home/chez_toi**

**/home/chez_moi**



# Formation Linux : Quelques commandes de base

**ls** : liste les répertoires
**cat** : affiche le contenu d'un fichier
**od** : dump un fichier
**find** : recherche des fichiers, et peut appliquer des commandes
**tar** : fabrique des archives
**cd** : change de répertoire
**grep** : recherche des chaînes dans des fichiers
**ps** : liste les process
**cut** : tronque des lignes (typiquement une commande pipée)
**sed** : modifie le contenu de fichiers, sans édition interactive
**cp** : copie un fichier
**mv** : déplace des fichiers
**rm** : détruit des fichiers
**pwd** : affiche le répertoire courant
**df** : affiche les filesystems montés, et la place libre
**du** : affiche la taille occupée par le répertoire
**lp** : lance l'impression d'un fichier
**at** : configure une exécution à un instant donné
**mkdir** : créer un répertoire
**head** : filtre les 1eres lignes d'un fichier
**tail** : filtre les dernières lignes d'un fichier
**export** : place une variable dans l'environnement
**tr** : transcription de caractères

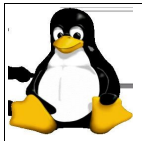
**top : liste des processus**

**Le shell, en plus, autorise de faire des fichiers de commandes, qui peuvent contenir des clauses complexes, avec des for, if, ... par exemple :**

```
for I in `ls *.pas`
do
    NBR=`cat $I | grep "procedure " | wc -l`
    NBR_TOT=`expr $NBR_TOT + $NBR`
done
echo "Nombre de procedures pascal = $NBR_TOT"

for I in $LISTE
do
    NBR=`cat expr | grep "(*" | wc -l`
    if [ $NBR -le 20 ]
    then
        echo "Fichier $I n'a pas assez de commentaires"
```

```
    fi
done
```



# Formation Linux : Démarrage d'une machine

**Le BIOS donne la main au programme qu'il trouve sur la MBR. Généralement, il s'agit d'un amorceur (loader), tel que celui d'OS2 ou LILO.**

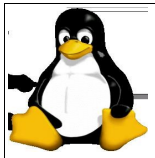
**Lilo (LInux LOader) va proposer des options de démarrage (si plusieurs), et lancer le noyau choisi (ou celui par défaut).**

**Pour configurer Lilo, il faut éditer le fichier /etc/lilo.conf, puis lancer /sbin/lilo pour que le lilo installé sur la MBR (ou sur la partition) puisse connaître son contenu. En effet, au boot, lilo ne sait pas lire / (ni un autre filesystem)**

**Ensuite, lilo va chercher un programme vmlinuz, le fameux noyau, selon la partition /boot indiquée.**

**Vmlinuz initialise les drivers qu'il contient, monte le filesystem nommé / dans lilo.conf, puis crée le processus init.**

**Attention, init a impérativement besoin d'une bibliothèque de routines la glibc.**

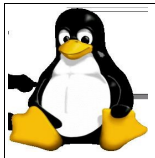


# Formation Linux : Démarrage d'une machine

**Init va lire le fichier /etc/inittab, et lancer les shells-scripts contenues dans le répertoire /etc/rc.d/x, X étant le runlevel. Par exemple, les runlevels 2, 3, 4, 5 permettent d'utiliser la machine avec plusieurs utilisateurs, mais configurée différemment.**
**Les shells commençant par S sont pour le démarrage (start) et par K l'arrêt (halt).**

**C'est en utilisant (et modifiant) ces scripts que l'on peut modifier le comportement de la machine (activer des services, désactiver des démons, ...).**
**Par exemple, le login graphique est activable par défaut uniquement sur le runlevel 5.**

**Les messages du boot du noyau sont importants, puisqu'ils déterminent l'initialisation des drivers. Si l'on souhaite les revoir, il suffit, en mode X11, de taper dmesg.**

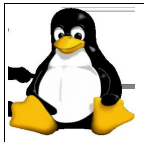


# Formation Linux : Démarrage d'une machine

**Une machine Unix qui a démarré doit être stoppée par une commande particulière. En effet, comme beaucoup de systèmes d'exploitation, Unix gère de la mémoire virtuelle et conserve le plus possible des données (issues du disque) en mémoire RAM. Ce qui fait qu'un arrêt inopiné peut s'avérer catastrophique.**

**Par contre, Unix détecte ces arrêts et lance alors un outil de réparation (fcsk, pour file system check) qui, très performant, corrige le plus souvent possible les problèmes.**

**Arrêter une machine se fait par shutdown, ou halt. Ce sont des shells-scripts que seul root peut exécuter, et qui, en final, lance un init runlevel 0.**



# Formation Linux : Gérer la machine bootée

**Lorsque vous travaillez, des processus se créent, s'arrêtent, restent actifs sur la durée de vie. Il peut-être intéressant de savoir arrêter un processus qui ne marche pas.**

**Un processus va, durant sa vie, recevoir des signaux du système. Il réagit à ces signaux en activant des fonctions de traitement (handler). Par contre, lorsque un processus reçoit un signal et qu'il n' a pas de handler associé, le kernel détruit le processus.**

**Chaque signal (il y en a une trentaine) a un rôle précis, comme par exemple :**

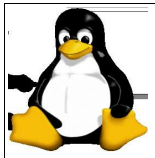
**SIGIO : un périphérique indique la présence d'un événement**
**SIGINT : interruption demandée**
**SIGUSR1 : réservé utilisateurs**
**SIGALRM : alarme (timer) tombée**
**SIGCHLD : fin d'un processus fils**

**Supposons qu'un programme soit lancé. Il crée un processus fils. La fonction système de la librairie pour créer un fils arme le handler pour SIGCHLD.**

**Lorsque le fils se termine, notre programme reçoit SIGCHLD (le kernel lui envoie ce signal), et donc exécute le handler (celui par défaut fourni).**

**Par contre, supposons que le même programme ne lance pas de processus fils, il n'a aucune raison d'armer SIGCHLD, (et comme il n'appelle pas fork, cela n'est pas fait).**
**Si jamais (par une commande utilisateur) il reçoit SIGCHLD, il ne peut exécuter de handler associé, et est donc tué par le kernel.**

**La règle de base en Unix est : un programme qui reçoit un signal non prévu est tué.**



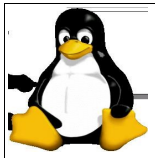
# Formation Linux : Environnement graphique

**Une machine Unix dispose, depuis les années 1980, d'un environnement graphique dit X11, à ce jour en release 6 (X11R6). Cet environnement est un serveur graphique (X), puis un ensemble de logiciels qui se greffent dessus (gestinnaires de fenêtres, gestionnaires de bureaux, utilitaires).**

**L'ensemble de cette mécanique, contrairement à d'autres OS PC, est indépendante du système booté. Ce ne sont que des processus supplémentaires, à lancer.**

**A noter que X fonctionne, d'emblée, en réseau. C'est à dire que l'on peut lancer une application graphique sur un écran qui est sur le réseau, pas forcément sur la machine locale de lancement.**

**L'environnement graphique se lance par startx (ou xdm au login). Startx va rechercher le serveur graphique à utiliser, puis lancer les shells-scripts de configuration (.xinitrc, ...) du répertoire de l'utilisateur. Puis, le serveur X va lancer un gestionnaire de fenêtre et de bureau (généralement KDE ou Gnome, mais il en existe plusieurs dizaines de différents).**

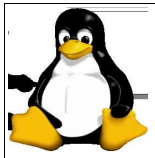


# Formation Linux : Environnement graphique

**Il va sans dire que X est conçu pour que, quelque soit le gestionnaire de fenêtre utilisé, une application graphique fonctionne. Par exemple, si vous utilisez fvwm2, Konqueror (projet KDE) fonctionne quand même.**

**Les gestionnaires de fenêtres les plus courants sont :**

**enlightment**
**fvwm2**
**fvwm95 (au look windows 95)**
**kde**
**gnome**
**windowmaker**



# Formation Linux : Environnement graphique

**KDE est fourni avec tout une palanquée de logiciels/utilitaires qu'il faut trouver dans les menus :**

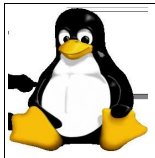
**ksnapshot, kview : aspects graphiques**
**kppp, kmail, krn, konqueror : mail, news et web**
**kmix, kOnCD : lecteurs multimédia**
**knote, kedit, kwrite, ... : edition**
**kGhostview : afficheur pdf**

**Il y a, en plus, tous les utilitaires systèmes non KDE :**

**xOsview, Xload, X11perf, XUSBView, ...**

**Et, bien sûr, les outils bureautiques :**

**Applixware, OpenOffice, Koffice, ...**

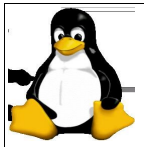


# Formation Linux : Linux et les réseaux

**Les réseaux sous Unix/Linux s'articulent autour de 4 thèmes :**

**les machines**
**les protocoles**
**les services**
**les applications/utilitaires**

**Nativement, Unix est tcp/ip. Tous les réseaux sont tcp/ip. Si il faut s'interconnecter sur des machines autres, il faut utiliser des logiciels (libres) qui ne font pas partie de la distribution. Les machines sont donc repérées sur le réseau par l'association adresse MAC et internet. Cette association se fait soit statiquement, soit dynamiquement.**
**Ensuite, le fichier de configuration des protocoles (/etc/protocols) donnent le lien entre le nom du protocole DARPA utilisé et le numéro à envoyer sur la couche réseau. Utile pour les développeurs de drivers ou d'applications communiquantes. Enfin, les services sont décrits dans /etc/services, qui donne le numéro du port socket, le nom du protocole et le nom du service.**



# Formation Linux : Environnement graphique

**Les logiciels ou utilitaires permettent d'échanger ou de partager des ressources. Ce sont, au minimum, des connections distantes, des commandes déportées ou shells distantes :**

**telnet, rlogin**
**rcp, rlp, rxxx**
**rsh, ssh**

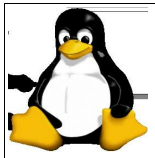
**Il y a aussi des fonctions de transferts de fichiers**

**ftp,  xftp,**

**Et enfin des systèmes de fichiers complètement partagés**

**nfs, samba**

**Et, pour finir, les applications de haut-niveau comme samba.**



# Formation Linux : Développement sous Linux

**Sans parler des EDI, développer sous linux, c'est, généralement, avec un éditeur (vi/emacs ou autre) et un compilateur (gcc), puis une foultitude d'outils autour :**

**joe, jed, tEdit, kEdit, Konqueror pour éditer du texte. Peuvent-être syntaxiques**
**gcc, gpc, gf77, Gnujava, ... pour les compilateurs**
**gdb, xgdb, ddd comme débogueur et frontaux graphiques**
**make, automake, autoconf pour la gestion technique du projet**
**lint, cflow, ctags, global, codeview, cscope pour la maîtrise de la difficulté**
**cvs est le système de gestion de versions naturel (consensuel), mais il y en a d'autres.**
**PostgresSQL, MySQL sont les sgbdr naturels (mais, là encore, ne sont pas les seuls)**

**Il y a ENORMEMENT de logiciels, libres ou non, autour de linux. Il faut simplement aller les chercher, et les installer. Donc, pour l'aspect développement, les SuSEs sont très complètes. Cela donne des idées des trucs dont on a besoin.**

**Un logiciel est livré soit en format tar (éventuellement compressé), soit en format paquetage (rpm et dérivé).**

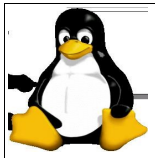
**Si il s'agit d'un format tar :**

**– décompresser si il y a lieu (gunzip)**
**– désarchiver (tar x[z])**
**– chercher un readme, install.sh, setup, ...**
**– suivre les instructions**

**Si il s'agit d'un rpm, il faut le copier dans une arborescence, puis lancer**

**$> rpm -i  (si nouveau paquetage)**
**ou**
**$>rpm -U (si mise à niveau)**

**Voir toutes les options possibles.**



# Formation Linux : Autour de linux

**Linux est un système auto-documenté. Il est livré avec de nombreuses sources d'information.**

**Il y a quelques commandes à connaître, et quelques documents à trouver pour obtenir toute l'information autour de Linux. Sans parler, ensuite, d'internet et de usenet.**

**whereis est une commande permettant de trouver l'emplacement d'une commande, comme par exemple :**

```
$> whereis ls
ls:/bin/ls /usr/share/man/man1/ls.1.tgz
$> whereis halt
halt : /sbin/halt /usr/share/man/man8/halt.8.gz
```

**Lorsque l'on est sûr de l'existence de la commande, whereis permet de s'assurer de sa présence.**

**apropos recherche dans le système d'aide ce que l'on sait sur un mot. Pratique si l'on ne sait pas ce que l'on cherche.**

```
$> apropos shutdown
shutdown (2)         - Terminer une commnication en full-duplex.
shutdown (8)         - Arrêter le système.
$> apropos gloubiboulga
gloubiboulga: rien d'adéquat
```

**Le numéro placé entre parenthèse sur le résultat de la recherche est le numéro de section de la page de manuel.**

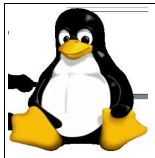
**man lance le manuel en ligne. Il est structuré en section, est normalisé, et se trouve dans /usr/share/man/man<section>/<commande>.<section>.gz . Pour obtenir de l'aide, il suffit de taper man commande.**

**Les sections sont :**

**1 pour les commandes**
**2 pour les appels systèmes**
**3 pour les appels de librairies**
**4 pour les fichiers spéciaux (/dev)**
**5 pour les formats des fichiers de configuration (/etc)**
**6 pour les jeux**
**7 pour les conventions**
**8 pour les commandes d'administrateur**
**9 pour les routines du kernel.**

**La première commande à man_er est man. A l'intérieur d'une page de man, vous disposez du comportement de la commande more.**

**info est une version des man avec des liens de type "hyper-texte" en mode console, à lancer dans un xterm. Il faut tout de même consulter la documentation de info pour comprendre comment se promener entre les liens.**



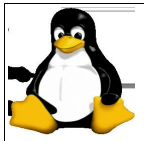
# Formation Linux : Autour de linux

**Les howto sont des fichiers décrivant un concept, un logiciel, une commande. Ils sont stockés dans /usr/share/doc/howto/<langue>/. Soit au format html, soit au format pdf. Le format HTML est donc utilisable par Konqueror.**

**Ils sont généralement fournis en anglais et en français. Mais ils ne sont pas tous traduits.**

**Il y a des Howto sur comment faire une disquette de sauvegarde, sur comment regégénrer un noyau, installer un graveur, la base de données postgres, utiliser ppp, etc, etc.**

**Le LDP est le Linux Documentation Project est un des projets linux concernant la mise à disposition de livres généraux, comme le User's guide, le Programmer's guide, l'Administrator's guide. Lorsque il n'est pas installé, il suffit de le chercher sur le web.**



# Formation Linux : Autour de linux

**Les sites web importants**
**http://linux-france.org et suivre les liens**
**§http://www.linux-doc.org**
**http://www.freashmeat.net**
**http://www.sourceforge.net**

**Il faut aller sur les sites des distributions (debian, redhat, ...) et penser aux forums de USENET. Les news, notamment, offrent de nombreuses possibilités d'échange :**

**fr.comp.os.linux.debats (fcold)**
**fr.comp.os.linux.configuration (fcolc)**
**fr.comp.applications.x11 (fcax)**
**fr.misc.bavardages.linux (fmbl, loisir)**

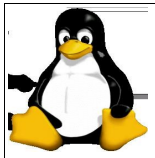
**Attention. Chaque forum dispose d'une charte qu'il faut respecter. Lire d'abord les messages, sur plusieurs jours, avant de poster. Il y a des habitués qui répondent et qui maintiennent les us. De plus, il y a des FAQs (avant de poser une question).**
**Sur http://tnemeth.free.fr/fmbl, vous trouverez des liens utiles.**

**La création de processus se fait en utilisant la fonction `fork()`. Celle-ci duplique le process appelant en mémoire, et le lance. Il y a donc 2 fois le même code qui s'exécute. Le processus appelant renvoie, en retour du fork, le PID du fils, ou, dans l'espace du fils, la valeur 0.**

**Généralement, ensuite, lorsque l'on se trouve dans le code du fils, on exécute `exec()`.**

**Pour utiliser un timer, il faut armer le signal SIGALRM et utiliser la fonction `setitimer()`.**

**Pour utiliser des tâches (internes à un process), ou des thread, il faut utiliser le type `pthread_t` et les fonctions `pthread_create()` et `pthread_exit()`. Les thread sont des fonctions du programme qui s'exécutent en asynchronisme avec le `main()`. Il est possible de réaliser des exclusions mutuelles de thread avec le mécanisme de mutex.**



# Formation Linux : Les librairies et le C

**Les communications inter-process (IPC) sont standard, et issues du Système V. Ce sont les**

- **tubes**
- **sémaphores**
- **segments de mémoire partagés**
- **files de messages**

**Les tubes sont créés par `pipe()`, et sont des descripteurs de fichiers comme ceux obtenus avec `open()`. Il existe les tubes normaux et les tubes nommés.**

**Les 3 autres IPC sont obtenus par des appels systèmes auquel on fournit un clé (`key_t`). Ensuite, on les manipule avec des fonctions `msgget()`, `msgsnd()`, `msgrcv()`, `msgctl()`, `semget()`, `semwait()`, `semctl()`, `shmget()`, `shmat()`, `shmctl()`.**

**Les sémaphores sont souvent utilisés avec les shm.**

**La gestion technique des IO consiste en l'utilisation de l'appel `select()`. Il s'agit d'indiquer au système quels sont les descripteurs à observer en entrée, en sortie, sur des exceptions, et ce pendant un temps précis.**

**On fournit à `select()` un `fd_set *`, sorte de table de descripteurs à remplir par les macros `FD_SET`, `FD_CLR`, ...**
**En retour de select, si il n'y a pas eu EINTR, on vérifie par FD_ISSET si le descripteur est présent dans les tableaux des descripteurs. Auquel cas, il reste à réaliser l'action sur ce filedes.**

**Le même comportement peut être obtenu par `fcntl()`, mais sur un seul descripteur.**

**Il est possible de programmer des E/S asynchrones, conformes à la norme POSIX1.B. Avec les appels `aio_read()` et `aio_write()`. Il faut configurer une structure aio_cb, qui contient des informations concernant le fildes, le buffer, ... et le comportement à avoir lorsque l'E/S est terminée.**

**La gestion des E/S par socket se met en oeuvre en :**

**choisissant un protocole (tcp, udp de /etc/protocols)**
**choisissant un service (ou port de /etc/services)**
**choisissant un domaine (AF_UNIX, AF_INET, AF_IPX, ...)**
**choisissant un type de socket (SOCK_STREAM, SOCK_DGRAM, SOCK_RAW)**

**Il faut ensuite créer la socket avec `socket()`.**

**Puis, la relier au réseau avec `bind()` du côté du serveur.**
**Puis, écouter le réseau, côté serveur, avec `listen()`.**
**Le client, lui, lance une connexion avec `connect()`.**
**Le serveur l'accepte avec `accept()`. Généralement, il crée un process pour traiter la socket, et la requête client, puis retourne sur le `listen()`.**

**En ce qui concerne la gestion des vecteurs d'interruption, cela est possible (mais déconseillé) en mode root. Sinon, il est possible de demander à être réveillé sur IT, et de gérer celle-ci avec `select()`.**

**Le module est installé en mode superuser, et « trappe » l'IT. Il autorise uniquement la fonction `select()`.**