I. Introduction aux systèmes d’exploitation

I.1. Qu'est-ce qu'un système d'exploitation?

Répondre complètement à cette question n'est pas simple. De manière pratique, le système d'exploitation est le logiciel le plus important de la machine, puisqu'il fournit :

- une gestion des ressources de celle-ci : processeurs, mémoires, disques, horloges, périphériques, communication inter-processus et inter-machines ;
- une base pour le développement et l'exécution de programmes d'application.

Pourquoi étudier les systèmes d'exploitation?

Tout programme est concerné : il est important d'appréhender la façon dont fonctionne un système d'exploitation pour améliorer l'efficacité de ses propres programmes ;

Tout programmeur est susceptible de rencontrer les mêmes problèmes de mise en oeuvre dans son propre domaine : pas la peine de réinventer la roue.

C'est un sujet intéressant en soi, dont l'objectif est la recherche de l'efficacité, nécessitant une étude théorique approfondie mais dont l'objectif est la fourniture de solutions réalisables en pratique : c'est une excellente approche du métier d'ingénieur !

Problématique

Pour que les programmes puissent s'exécuter de façon portable et efficace, il faut pouvoir gérer simultanément :

- la multiplicité des différentes ressources ;
- la complexité des composants de chacune d'elles, qui requiert la prise en compte de nombreux détails embêtants, sources de bogues.

Ne sont pas des systèmes d'exploitation :

- l'interprète de commandes ;
- le système de fenêtrage ;
- les utilitaires (cp, chmod, uptime, . . .) ;
- le compilateur (ni sa bibliothèque) ;
- l'éditeur. . .

En fait, tous ces programmes s'exécutent dans un mode non privilégié, car ils n'ont pas besoin d'un accès privilégié au matériel. En revanche, le système d'exploitation fonctionne typiquement en mode privilégié, pour pouvoir accéder a toutes les fonctionnalités du processeur. Ainsi, le système d'exploitation est protégé par le matériel contre les erreurs de manipulation (mais il existe des systèmes d'exploitation s'exécutant sur du matériel non protégé, comme par exemple le DOS sur les anciens IBM PC).

Fonctionnalités d'un système d'exploitation

Un système d'exploitation a pour but :

- de décharger le programmeur d'une tâche de programmation énorme et fastidieuse, et de lui permettre de se concentrer sur l'écriture de son application ;
- de protéger le système et ses usagers de fausses manipulations;
- d'offrir une vue simple, uniforme, et cohérente de la machine et de ses ressources.

Principes des systèmes d'exploitation : Appels système

Ils constituent l'interface entre le système d'exploitation et les programmes d'application (ou leurs bibliothèques) qui s'exécutent en mode non privilégié (aussi appelé parfois mode utilisateur. Ils sont realisés au moyen d'instructions spécifiques (les << traps >>, ou interruptions logicielles), qui permettent le passage en mode privilegié (aussi appelle mode noyau, ou << kernel >>), lorsqu'il existe sur le processeur.

Au niveau du processeur, le mode noyau se différencie habituellement du mode utilisateur par les fonctionnalités suivantes :

- le code et les données utilisées par le système d'exploitation ne sont accessibles qu'en mode noyau. Ceci se fait en n'incluant les segments mémoires correspondants que lors du passage en mode noyau ;
- les instructions de modification de la table des segments mémoires ne sont permises qu'en mode noyau. Ainsi, un programme utilisateur ne pourra modifier ses droits d'accès à la mémoire ;
- les instructions de lecture et d'écriture sur les ports d'entrée/sortie du matériel ne sont permises qu'en mode noyau. Un programme d'application ne peut donc accéder directement au matériel sans passer par le système d'exploitation.

La différence entre mode noyau (privilégié) et mode utilisateur (non privilégié) est gérée directement au niveau du processeur. Elle n'a rien à voir avec la notion de super-utilisateur mise en oeuvre sur certains systèmes d'exploitation, qui est gérée au niveau logiciel dans le code du système d'exploitation. En fait, même le super-utilisateur d'un système passe le plus clair de son temps Cours de systèmes d'exploitation

I.2. Historique du système GNU/Linux

UNIX est l'un des systèmes d'exploitation les plus populaires au monde, en raison du grand nombre d'architectures qu'il supporte. Il fut à l'origine développé en tant que système d'exploitation multitâches pour mini-ordinateurs et grands systèmes vers l'année 1970, mais a bien évolué depuis pour devenir l'un des systèmes les plus utilisés, en dépit de son interface parfois déroutante et de son manque de réelle standardisation.

La raison de la popularité d'UNIX ? Beaucoup de programmeurs le ressentent comme La Vérité; Le Vrai Système d'Exploitation, presque de manière religieuse. D'où le développement de Linux, par un groupe évolutif de programmeurs UNIX désirant mettre les mains dans le cambouis et réaliser leur propre système.

Il existe des versions d'UNIX pour beaucoup d'ordinateurs, depuis les ordinateurs personnels jusqu'aux grosses machines comme le Cray Y-MP. La plupart de ses implémentations pour PC sont lourdes et onéreuses. A l'heure ou nous écrivons ces lignes, une version pour une seule machine de l'UNIX System V de AT&T pour i386 coûte environ US$1500.

Linux est une version d'UNIX gratuite et librement diffusable développée à l'origine par Linus Torvalds à l'université de Helsinki, en Finlande(août 1991). Linux a été développé avec l'aide de nombreux programmeurs et spécialistes UNIX, grâce au réseau mondial Internet, autorisant quiconque ayant suffisamment de connaissances à participer activement à l'évolution du système. Le noyau de Linux n'utilise aucun code en provenance de AT&T ou de quelque autre source propriétaire, et la plupart des programmes disponibles pour Linux est développée par le projet GNU à la Free Software Foundation à Cambrige, Massachusetts. Toutefois, des programmeurs du monde entier ont contribué à l'ensemble.

Linux était au départ un projet de loisirs de Linus Torvalds. Il fut inspiré de Minix, un petit système UNIX développé par Andy Tanenbaum, et les premières discussions à propos de Linux se passèrent sur le forum USENET comp.os.minix. Ces discussions portaient principalement sur le développement d'un petit systeme UNIX académique pour les utilisateurs de MINIX qui désiraient mieux que cela.

Les prémices du développement de linux furent la maîtrise de la commutation de tâches du mode protégé du processeur 80386, tout fut écrit en assembleur. Linus écrit:

``Après ça, tout coulait de source: encore de la programmation touffue, mais j'avais quelques périphériques, et le déboguage était plus facile. C'est à ce stade que j'ai commencé à utiliser le langage C, ce qui a certainement accéléré le développement. C'est aussi à ce moment que j'ai commencé à prendre au sérieux mes idées mégalo de faire un "Minix meilleur que Minix". J'espérais un jour pouvoir recompiler gcc sous Linux...

``Deux mois pour le code de base, puis un peu plus jusqu'à ce que j'aie un pilote de disque dur (sérieusement bogué, mais par chance il fonctionnait sur ma machine), et un petit système de fichiers. C'est à cette époque que j'ai diffusé la version 0.01 [fin août 1991]: Ce n'était pas très beau, je n'avais pas de pilote de disquette, et ça ne pouvait pas faire grand chose. Je ne pense pas que quelqu'un ait un jour compilé cette version. Mais j'étais pris au jeu, et je ne voulais plus m'arrêter tant que je ne pouvais pas jeter Minix aux orties.''

Aucune annonce de Linux version 0.01 ne fut jamais faite. Ce n'était même pas exécutable; l'archive ne contenait que les rudiments des sources du noyau, et considérait que vous aviez accès à un système Minix pour compiler Linux et jouer un peu avec.

Le 5 octobre 1991, Linus annonça la toute première version ``officielle'' de Linux, la version 0.02. A ce moment, Linux était capable d'éxécuter bash (le Bourne Again Shell de GNU), mais pas grand chose d'autre. Encore une fois, c'était un système destiné aux hackers, focalisé sur le développement du noyau. Le support utilisateurs, la documentation, la distribution, ou autres, n'avaient jamais été évoqués. Aujourd'hui, la communauté Linux semble continuer à traiter ces choses là comme très secondaires en comparaison de la ``vraie programmation'', le développement du noyau.

Linus écrit dans comp.os.minix,

`` Vous regrettez les beaux jours de Minix-1.1, lorsque les hommes étaient des hommes et écrivaient leurs propres pilotes de périphériques ? Vous manquez d'un superbe projet et vous languissez après un système que vous pourriez modifier à votre convenance ? Vous êtes frustrés que tout fonctionne sous Minix ? Plus de nuits blanches passées à tenter de faire fonctionner un programme récalcitrant ? Alors ce message pourrait bien être pour vous.

`` Comme signalé il y a un mois, je travaille actuellement sur une version libre et gratuite d'un système ressemblant à Minix pour les ordinateurs AT-386. J'ai finalement atteint un stade où il est utilisable (bien qu'il puisse ne pas l'être pour vous, selon ce que vous désirez), et je compte diffuser les sources pour une diffusion plus large. Il s'agit juste de la version 0.02... mais j'ai pu exécuter bash, gcc, gnu-make, gnu-sed, compress, etc. avec succès sous ce système.

Après la version 0.03, Linus passa le numéro de version directement à 0.10, puisque de plus en plus de gens commencèrent à travailler sur le système. Puis, après plusieurs autres révisions, Linus gonfla à nouveau le numéro pour sortir la version 0.95, afin de refléter son impression: Linux était prêt pour une version ``officielle'' très prochainement. (Généralement, un programme ne se voit attribuer le numéro de version 1.0 que lorsqu'il est théoriquement complet, ou sans bogue). Ceci se passait au mois de mars 1992. Presque un an et demi plus tard, fin décembre 1993, le noyau de Linux en était encore à la version 0.99.pl14... Approchant 1.0 de manière asymptotique. Certains pensaient que la version 1.0 ne verrait jamais le jour.

Aujourd'hui, Linux est un système UNIX complet, capable d'exécuter X Window, TCP/IP, Emacs, UUCP, le courrier électronique et les news Usenet, ou tout ce que vous voudrez. Pratiquement tout les programmes freewares importants ont été portés sous Linux, et on commence à voir apparaître des applications commerciales. Linux supporte beaucoup plus de périphériques que dans ses premières versions. Beaucoup de gens ont effectué des tests de

machines 80486 sous Linux et ont trouvé des performances comparables aux stations de travail de milieu de gamme de Sun Microsystems et Digital Equipment Corporation. Qui aurait pu imaginer qu'un jour, ce ``petit'' clone d'UNIX serait devenu si grand?

Les distributions Linux

Une distribution Linux comprend le noyau, les pilotes, les bibliothèques, les utilitaires d'installation et de post-installation, ainsi qu'un grand nombre de logiciels.

Slackware

La première distribution qui a popularisé Linux auprès du grand public. Elle n'a pas connu l'essor auquel elle pouvait prétendre de par sa précocité. La distribution Slackware est une distribution complète de Linux, fournie sur quatre CD-ROM.

Red Hat

L'apport principal de Red hat est sans doute le concept de paquetage. Un paquetage comprend un logiciel, sa documentation ainsi que tous les utilitaires qui en simplifient l'installation, la désinstallation ou la mise à jour, le tout prêt à l'emploi sur un simple clic. Ce concept a, du reste, été repris par d'autres distributions.

Cette distribution comprend des outils d'installation et de configuration simples d'emploi en mode graphique ou en mode caractère.

Red Hat existe pour les nombreuses plate-formes parmi lesquelles on peut citer : Intel, Compaq Alpha, sun SPARC, Power PC et Mac (en cours)

Debian

Debian est une distribution non commerciale de Linux, avec l'objectif d'être de grande qualité. Debian utilise son propre format de paquetage. Le développement de Debian est réalisé, via l'Internet, de la même manière que le noyau Linux. Debian support les paltes-formes intel, Compaq alpha, Sun SPARC.

S.u.S.E

Basée sur la distribution Slackware, cette distribution d'origine allemande entreprend un déploiememnt aux Etats-Unis. Elle possède ses propres outils d'installation et utilise les paquetages Red Hat. L'installation se fait via des menus depuis le CD-ROM.

Mandrake

La distribution Mandrake est basée sur la distribution Red Hat. Elle dispose de son propre outils graphique d'installation.

Caldera

La distribution OpenLinux de Cadera inclut des produits commerciaux :

- Netscape FastTrack Web Server,
- Le système de gestion de bases de données ADABAS,
- Novell Netware

EasyLinux

C'est la distribution de la société EIT.

CorelLinux

C'est la distribution de la société Corel. Elle inclut le logiciel wordPerfect. Elle est basée sur la distribution Debian et utilise l'interface KDE. Elle dispose de son propre outil d'administration.

Trinux

C'est un système Linux qui fonctionne uniquement en mémoire et qui possède des outils d'administration réseaux.

TurboLinux

C’est une version de linux en cluster, distribuée par la société TurboLinux. Cette distribution est payante. Elle est destinée à équiper de gros serveurs.

I.3. Découverte du système GNU/Linux

Présentation d’UNIX

UNIX est un système d’exploitation moderne, complet et efficace, disponible sur la plupart des ordinateurs vendus, du PC au super calculateur Cray. C’est pourtant un système ancien, puisque son histoire remonte à la fin des années 60. Son architecture ouverte et sa grande diffusion dans les centres de recherches et les universités lui ont permis d’évoluer en intégrant de nombreuses améliorations.
Aujourd’hui, UNIX est très utilisé en informatique scientifique, et pour les serveurs réseaux : la grande majorité des serveurs sur Internet fonctionnent sous UNIX. Par contre, sa relative complexité d’utilisation l’écarte des applications grand public.

Le but de ce cours est d’abord de donner un aperçu général du fonctionnement du système UNIX et de se familiariser avec ses commandes de bases : manipulation sous shell des fichiers et processus.

Nous nous sommes efforcé de ne décrire que des commandes standards, qui devraient fonctionner sous toutes les versions d’UNIX et pour tous les utilisateurs.

Architecture générale du système

UNIX est un système d’exploitation multi-tâche multi-utilisateurs. Le fonctionnement multi-tâche est assuré par un mécanisme *préemptif* : le système interrompt autoritairement la tâche en cours d’exécution pour passer la main à la suivante ; ceci évite tout risque de bloquage du système à la suite d’une erreur survenant dans un programme utilisateur. La cohabitation simultanée de plusieurs utilisateurs est rendue possible par un mécanisme de *droits d’accès* s’appliquant à toutes les ressources gérées par le système (processus, fichiers, périphériques, etc.).

Le noyau UNIX

Le *noyau* est le programme qui assure la gestion de la mémoire, le partage du processeur entre les différentes tâches à exécuter et les entrées/sorties de bas niveau.

Il est lancé au démarrage du système (le *boo*t) et s’exécute jusqu’à son arrêt.

C’est un programme relativement petit, qui est chargé en mémoire principale.

Le rôle principal du noyau est d’assurer une bonne répartition des ressources de l’ordinateur (mémoire, processeur(s), espace disque, imprimante(s), accès ré-seaux) sans intervention des utilisateurs. Il s’exécute en mode *superviseu*r, c’est à dire qu’il a accès à toutes les

fonctionnalités de la machine : accès à toute la mémoire, et à tous les disques connectés, manipulations des interruptions, etc.

Tous les autres programmes qui s'exécutent sur la machine fonctionnent en mode *utilisateur* : ils leur est interdit d'accéder directement au matériel et d'utiliser certaines instructions. Chaque programme utilisateur n'a ainsi accès qu'à une certaine partie de la mémoire principale, et il lui est impossible de lire ou écrire les zones mémoires attribuées aux autres programmes.

Lorsque l'un de ces programmes désire accéder à une ressource gérée par le noyau, par exemple pour effectuer une opération d'entrée/sortie, il exécute un *appel systèm*e. Le noyau exécute alors la fonction correspondante, après avoir vérifié que le programme appelant est autorisé à la réaliser.

I.4. Présentation des interprètes de commandes et découverts des commandes de base du système

L'invite du shell

Vous aurez remarqué que l'invite Linux est assez similaire à l'invite MsDos. Sous DOS, l'invite n'est pas très bavarde. Elle n'indique que la lettre du lecteur, suivie d'un double point et éventuellement du nom du répertoire dans lequel l'utilisateur se trouve. Exemple :

C:> D:> E:\dos ...

Sous Linux, elle communique plusieurs choses : Le nom de l'utilisateur connecté, le nom de la machine, ainsi que le répertoire courant. Exemple :

[root@local /root]#

Ici, l'utilisateur est root, la machine se nomme local, et le répertoire courant, est root Le # indique qu'il s'agit de l'administrateur système (on l'appelle également le root ou le super utilisateur.

[max@screamer /etc]$

L'utilisateur se nomme max, la machine screamer, il se trouve dans le répertoire /etc et le signe $ indique qu'il s'agit d'un utilisateur classique.

[root@cassoulet /]#

Ici, l'utilisateur est root la machine s'appelle cassoulet et il se trouve dans le répertoire racine .

L'interpréteur de commandes

L'interpréteur de commandes est un logiciel faisant partie du système d'exploitation Linux dont le but est de faire l'interface entre la personne travaillant sur l'ordinateur et les commandes qui sont exécutées. Que se

passe t - il si l'utilisateur tape un nom de commande incorrect ? Tapez par exemple :

[root@localhost /root]# abcde [Entrée]

bash: abcde: command not found
  Les messages d'erreur ne sont pas à négliger. Sous Linux on utilise un interpréteur de commande appelé le bash.

Commandes de manipulation de fichiers

ls - Lister les noms des fichiers
La commande ls (abréviation de list), lorsqu'on la tape au clavier, demande au système d'afficher la liste des noms des fichiers que l'on possède (équivalent de dir sous DOS).

ls [Entrée]

Le système doit renvoyer la liste des noms de vos fichiers. ls l vous, permet d'afficher toutes les informations sur chaque fichier du répertoire en cours.

more - Afficher le contenu d'un fichier
Pour voir le contenu d'un fichier, le plus efficace est d'utiliser la commande more qui admet en paramètre le nom du fichier concerné - on peut aussi utiliser la commande cat que nous verrons plus loin.

more /home/max/compta.txt [Entrée]

Sur l'écran doit s'afficher le début du fichier compta.txt.
More permet quelques commande pour se déplacer dans le fichier :
[Entrée] - fait apparaître une ligne supplémentaire du fichier
[Espace] - idem mais par bloc de 24
[b] - rermonte dans le texte
[q] - arrete l'exécution de more

cat - Concaténer des fichiers
Théoriquement destinée à concaténer des fichiers, elle est aussi utiliséée pour afficher tout le contenu d'un fichier en une fois - on l'utilise aussi pour créer de nouveaux fichiers ou des fichiers vides. Commande pouvant être comparée à la commande type sous DOS.

cat /home/max/compta.txt [Entrée]

cp - Copier un fichier
La duplication de fichier n'est pas chose anodine pour tout un chacun, cela permet d'avoir une sauvegarde.

cp /home/max/compta.txt /home/max/nouveau_fichier [Entree]

mv - Déplacer et renommer un fichier Cette commande est similaire au Ren du DOS, permettant de renommer un nom de fichier ou de le déplacer, Move sous DOS.

mv /home/max/compta.txt /home/bobn/compta.txt [Entrée]

rm - Détruire un fichier

rm compta.txt [Entrée]

ATTENTION - avec les systèmes Linux il n'y a aucun moyen de récupérer un fichier supprimé.

find - Chercher un fichier

find chemin fichier
find /home compta.txt [Entrée]

grep - Recherche dans un fichier
Cette commande permet de rechercher toutes les lignes du fichier contenant l'expression recherchée

grep client Tux /home/max/compta.txt [Entrée]

Infos et commandes associées aux fichiers et aux répertoires

Un utilisateur gère deux types d'objets informatiques : les fichiers et les répertoires. Dans le monde Unix, un fichier et un répertoire c'est la même chose, si cela vous perturbe, ce n'est pas la peine de bloquer dessus. On détermine chaque objet via la commande ls avec l'option -l, ce qui donne un format long.
Exemple :

[root@localhost /root]# ls -l [Entrée]

```
   rwxr xr x    1   max   root   1639  jan 8 05:46   SwitchConfig
   rw r r       1   max   root   12809 jan 7 10:33   config.in.fra~
   rwxr xr x    1   max   root   13709 jun 6 1989    dos2unix.com
   drwxr xr x   4   max   root   512   jan 8 18:52   kcs
   drwxr xr x   2   max   root   512   jan 8 18:52   kernel
   drwxr xr x   4   max   root   512   jan 9 20:57   usr
```

Examinons le résultat obtenu :

- La colonne 1 indique les droits sur l'élément, de gauche à droite nous avons l'utilisateur, le groupe auquel il appartient et les droits pour les autres.

- La colonne 3 indique ici que le propriétaire est max.

- La colonne 4 indique ici que le groupe auquel appartient ces objets est root.

- La colonne 5 indique la taille de l'objet en octet.

- Les colonnes 6, 7, 8 contiennent la date et l'heure de la dernière modification de l'objet.

cd - Changement de répertoire
Pour changer de répertoire on disposes de la commande cd.

cd max [Entrée]
cd /home/max [Entrée]

mkdir - Création de répertoire
Les répertoire sont un moyen très utiles pour classer vos fichiers, pour en créer on utilise la commande mkdir.

mkdir TP [Entrée]

rmdir - Destruction de répertoire
On peut detruire un répertoire depuis le répertoire père à l'aide la commande rmdir. Un répertoire non vide ne peut être vidé avec cette commande sans options, pour cela on utilise les options -r (récurssif) et -f (force).

rmdir TP [Entrée]
rmdir /home/max/TP [Entrée]
rm -rf /home/max/tmp [Entrée]

pwd - Comment connaître le répertoire courrant
A force de se déplacer dans l'arborescence on fini par ne plus savoir où l'on est, ce n'est pas que valable pour les débutants. Pour cela on utilise la commande pwd. vos fichiers, pour en créer on utilise la commande mkdir.

pwd [Entrée]

Autres commandes de base

clear - Efface l'écran
clear [Entrée]

df -Espace disque df [Entrée]
Sans paramètre optionnel, df affiche l'espace disque disponible sur tous les systèmes de fichiers montés. Pour avoir l'aide, tapez ceci : df -help [Entrée] et/ou man df [Entrée].

gzip - Compression et décompression de fichiers
Différentes distributions incluent des packages portant l'extention : . tar.gz .tgz ou .gz. Ce sont des fichiers compressés à l'aide l'utilitaire gzip. Pour les décompresser on doit utiliser l'option -d :

gzip -d fichier.tgz [Entrée]
Pour compresser un fichier on tape la commande gzip suivit du nom du fichier à compresser :

gzip fichier [Entrée]
On obtient l'aide de gzip comme ceci :
gzip -help [Entrée]

kill - Tuer un processus
kill pid [Entrée] - pid = n° du processus à tuer.

diff - Comparer deux fichiers
Cette commande compare deux fichiers et affiche les modifications à faire pour le rendre identique à l'autre.

diff ancien nouveau [Entrée]
diff compta.txt compta2.txt [Entrée]
On obtient l'aide comme ceci : diff -help [Entrée]

/dev/lpt - Dirige un fichier vers l'imprimante
Cette commande est associée à cat et imprime le fichier vers le port parallèle désigné.

cat compta.txt > /dev/lpt0 [Entrée]

tar - Création d'archive
Les distributions Linux utilisent le plus souvent les archives tar combinées avec gzip. On les reconnait à leurs extentions .tar (archive tar), .tar.gz ou .tgz (archive tar compressée avec gzip. Syntaxe générale :
tar options archives [Entrée]
Lister le contenu de l'archive :

tar tvf archives [Entrée]
tar tvfz archives [Entrée]
Décompresser :

tar xvf archives [Entrée]
tar zxvf archives [Entrée]
Création d'archive :

tar zcvf archives.tgz fichiers_à_archiver [Entrée]
tar cvf archives [Entrée]
Obtenir l'aide de tar :

tar -help [Entrée]
man tar [Entrée]

II. Gestion des utilisateurs sous GNU/Linux

II.1. Notion d'utilisateurs et de groupes sous GNU/Linux

Qui est utilisateur ?

Le système, dès son installation, avant même la première connexion au système a créé des users système.
Un utilisateur n'est donc pas uniquement une personne physique, le système a besoin d'utilisateurs pour sa gestion interne, notamment comme propriétaire des divers processus.
La commande ps aux | less montre qu'avant toute connexion d'utilisateur humain (repérée par les lignes login --user), root a lancé init, et la plupart des services, crond, xinetd, lpd, smbd, ... , avant de lancer les connexions utilisateurs dans les consoles, y compris éventuellement la sienne !

Les principales commandes

useradd, usermod, userdel	gestion des comptes utilisateur
groupadd, groupmod, groupdel	gestion des groupes
pwck, grpck	vérification des fichiers
passwd	changer le mot de passe d'un utilisateur
chfn, id, groups, finger	utilitaires divers

II.2. Gestion des utilisateurs et des groupes sous GNU/Linux

Créer un compte pour un nouvel utilisateur

Cela signifie lui permettre d'être connu du poste local, s'y loguer, avoir un accès complet sur son rép. personnel.
Mais aussi dans une configuration réseau, de pouvoir se connecter à son compte par telnet et ftp, et de pouvoir bénéficier de services réseau de partage distant (sous Linux par NFS et sous Windows 9x par SMB).

- Pour créer l'utilisateur **stagex**, root passe la commande :
  useradd stagex
  Ceci crée :

- le répertoire personnel **/home/stagex, portant par défaut le nom du compte**

- une nouvelle entrée dans les 2 fichiers fondamentaux / etc/passwd et /etc/group.

- Pour connaître les options de useradd : **man useradd**

- Pour lui attribuer le mot de passe :
  passwd stagex
  saisir 2 fois **stgx**

- Supprimer le compte d'un utilisateur (non connecté), au hasard .. totox.
  userdel [-r] totox
  L'option **-r** supprime aussi le rép. personnel et les fichiers de l'utilisateur
  La commande supprime toute trace de l'utilisateur dans le fichier de configuration : /etc/passwd y compris dans les groupes d'utilisateurs.

- Modifier le compte de l'utilisateur toto
  usermod [options] totox
  Les options sont les mêmes que useradd
  usermod -G stagiaire,prof stagex ajoute stagex dans les 2 groupes stagiaire et profs (qui doivent exister)

Remarques

- Attention : si root passe la commande passwd il s'apprête à redéfinir son propre mot de passe !

- Un utilisateur quelconque ne peut pas créer de compte, même s'il a le privilège de faire partie du groupe root.

- Par contre, il peut modifier lui-même son mot de passe.

- Voir les diverses options avec **useradd -h**

- Pour une gestion sous interface graphique, utilisez l’outil **linuxconf**

- Attention ! Le compte créé permet à l'utilisateur d'accéder au système de fichier Linux (avec des droits que nous verrons). Pour pouvoir se connecter au réseau SAMBA, à partir d'une station distante Windows9x, il faut créer un compte Samba avec l'utilitaire smbpasswd (voir le chapitre serveur Samba).
  A noter que linuxconf semble créer automatiquement les comptes Linux et Samba conjointement.

Les groupes

- Un groupe est, aussi pour Linux, un ensemble d'utilisateurs qui partagent les mêmes fichiers et répertoires. Nous verrons que les fichiers accordent des droits d'accès réglables à ces groupes.

- Chaque utilisateur doit faire partie au moins d'un groupe, son ***groupe primaire***. Celui-ci est défini au moment de la création du compte, et *par défaut*, l'utilisateur appartient à un nouveau groupe créé, portant son nom.

- Ainsi, dans /etc/passwd chaque utilisateur possède un groupe par défaut, précisé par son identifiant gid dans ce fichier.

- L'appartenance au groupe primaire n'étant pas exclusive, **tout utilisateur peut faire partie de plusieurs autres groupes**, appelés ses *groupes secondaires*.
  Mais le rôle joué par le groupe primaire demeure prépondérant, comme nous le verrons dans le système des permissions des fichiers.

- Pour lister tous les groupes (primaire et secondaires) d'un utilisateur :
  groups stagex

- Pour créer un nouveau groupe
  groupadd stagiaire

- Supprimer un groupe, au hasard .. encore totox.
  groupdel totox
  Le groupe est supprimé du fichier /etc/group.

- Pour ajouter un utilisateur à un groupe
  Le plus simple est d'éditer le fichier /etc/group et d'ajouter une liste d'utilisateurs (séparés par des virgules) sur la ligne du groupe (ou utiliser Linuxconf).

Visite des coulisses

- Tout ce qui concerne la gestion et l'authentification des utilisateurs est inscrit dans un seul fichier **/etc/passwd**

- La gestion des groupes est assurée par **/etc/group**

- Les mots de passe cryptés sont maintenant placés dans **/etc/shadow**, par sécurité lisible seulement par root.

Structure de /etc/passwd

Ce fichier comprend 7 champs, séparés par le symbole :

1. nom de connexion

2. ancienne place du mot de passe crypté

3. numéro d'utilisateur **uid**, sa valeur est le véritable identifiant pour le système Linux; l'uid de root est 0, le système attribut conventionnellement un uid à partir de 500 aux comptes créés.

4. numéro de groupe **gid**, dans lequel se touve l'utilisateur par défaut; le gid de root est 0, des groupes d'utilisateurs au delà de 500

5. nom complet, il peut être suivi d'une liste de renseignements personnels (cf chfn)

6. rép. personnel (c'est également le rép. de connexion)

7. shell, interprétateur de commandes (par défaut /bin/bash)

Structure de /etc/group

Ce fichier comprend 4 champs, séparés par le symbole :

1. nom du groupe

2. x pour remplacer un mot de passe non attribué maintenant

3. numéro de groupe, c-à-d l'identifiant **gid**

4. la liste des membres du groupe

Outils de gestion des comptes

Linuxconf
Se connecter comme root et lancer linuxconf en ligne de commande (on pourrait utiliser linuxconf sous X-KDE)
Section *Comptes utilisateurs*

- Sélectionner un compte et examiner sa définition actuelle sous l'interface de Linuxconf
  Comparer avec son entrée dans /etc/passwd

- Ajouter un nouveau compte (stagey/stgy) en donnant seulement les nom de login et nom complet
  A la validation, on observe que Linuxconf exécute 2 commandes useradd et chage. Puis on est averti que le mot de passe a bien été enregistré dans passwd et smbpasswd

- Pour connaitre le rôle de chage, consulter le manuel et le fichier / etc/shadow Créer maintenant un autre compte (toto) en précisant le groupe primaire (zig), et des groupes secondaires (stagiaire). Examiner la syntaxe des 2 lignes de commandes exécutées
  (/usr/sbin/useradd et /usr/bin/chage

- Si le rep. de base n'est pas spécifié, par défaut création et attribution de /home/stagex

- On peut tout de suite placer l'utilisateur dans une liste de groupes (sans virgule)

Kuser

- Lancer sous X-KDE, la commande K/Système/Gestionnaire d'utilisateurs

- Utilisation à découvrir

Compléments

- La structure d'une ligne de /etc/passwd et de /etc/group

- login:x:uid:gid:commentaires:home:shell
- groupe:x:gid:liste-groupes-secondaires

- Options de la commande **useradd** (pour détails cf man useradd)
  Nous avons jusqu'ici utilisé cette commande avec ses options par

défaut.
La maitrise de cette commande est indispensable pour écrire des scripts de génération automatique de comptes.
Syntaxe : useradd [options] nom_login
Exemple : useradd toto -u 1200 -p moi -g 520 -G groupes -s /bin/bash
Options :

-u uid	pour fixer l'identifiant uid
-g groupe-primaire	
-G liste	fixe l'appartenance de l'utilisateur à une liste de groupes secondaires (séparateur , sans espace)
-s shell	par défaut, attribution du shell par défaut bash
-c commentaire	
-d rep. personnel	par défaut dans le répertoire /home
-e date-expiration	fixe la date d'expiration du compte (format MM/JJ/AA)
-m	pour créer le répertoire personnel
-k rep-skel	recopie le contenu de rep-skel dans le rép. personnel, par défaut /etc/skel

- La recopie du répertoire /etc/skel est très important pour l'administrateur, car il lui permet de configurer de façon uniforme les sessions de travail des utilisateurs.
  C'est ainsi que tous les utilisateurs qui passe en mode graphique KDE hérite du même bureau.

- Pour examiner les valeurs par défaut appliquées par **useradd** : commande useradd -D ou éditer /etc/default/useradd

- GROUP=100 *identifiant du groupe primaire*
- HOME=/home *racine des rép. personnels*
- INACTIVE=-1 *(nb de jours avant destruction du compte*
- EXPIRE= *nb de jours avant expiration du mot de passe*
- SHELL=/bin/bash *shell de connexion attribué au compte*
- SKEL=/etc/skel *fichiers recopiés par défaut dans chaque rép. personnel*

- La commande **passwd**
  Elle est chargée du cryptage du mot de passe dans /etc/shadow
  Syntaxe : **passwd [option] nom-login**
  Options

  - **--stdin**, la commande abandonne son caractère interactif habituel et examine son entrée standard pour s'en servir comme mot de passe.

Très utile dans un script : echo mot | passwd --stdin
(attention tout caractère est significatif, y compris les " ")

- **-d** , pour supprimer le mot de passe, l'utilisateur pourra se connecter sans !

- **-l** , pour verrouiller le compte et empêcher sa connexion.

- **-u** , pour déverrouiller.

- Connaitre l'uid et le gid de l'utilisateur courant

- Commande **id**
- uid=501(stage1) gid=501(stage1) groups=501(stage1), 504(stagiaire)

- Pour décrire un utilisateur : **chfn**
  Cette commande permet d'indiquer dans le champ numéro 5 du fichier /etc/passwd différentes informations sur un utilisateur, son nom complet, son bureau, ses numeros de téléphone (séparées par des virgules).

- Cryptage des mots de passe
  Pour des questions de sécurité, les mots de passe cryptés ne sont stockés dans /etc/passwd qui doit etre accessible en lecture par tous La commande **/usr/sbin/pwconv** est chargée de transférer les mots de passes cryptés, dans **/etc/shadow**. Pour plus de détails , consulter man pwconv

III.Gestion des fichiers

III.1. Présentation des principaux répertoires du système

Ce chapitre présente l'organisation du système de fichiers et le rôle de chacun des répertoires

Il n'existe pas de norme d'organisation du système de fichiers, mais un standard est à peu près suivi par les différentes distributions de Linux.

L'organisation traditionnelle du répertoire racine est décrite dans le tableau suivante :

Répertoire	**Contient**
/bin	les fichiers exécutables nécessaires à l'initialisation
/boot	le noyau et les fichiers de démarrage
/dev	les fichiers spéciaux
/etc	les fichiers de configuration du système et certains scripts
/home	la base des répertoires utilisateurs
/lib	les librairies système et les modules
/lost+found	le stockage des fichiers retrouvés par fsck
/mnt	les points d'ancrage des systèmes extérieurs
/proc	un système de fichiers virtuels permettant l'accès aux variables du noyau
/root	le répertoire de base du super utilisateur
/sbin	les fichiers exécutables pour l'administration du système
/tmp	les fichiers temporaires
/usr	les programmes, les librairies et les fichiers accessibles pour l'utilisateur
/var	les données variables liées à la machine (spool, traces)

Le répertoire de base / s'appelle : répertoire racine (*root*) par analogie avec la racine d'un arbre représentant le système de fichiers.

Il n'est pas standard d'ajouter des répertoires au niveau de la racine.

Ce système de fichiers peut résider sur différentes partitions, différents supports physiques ou sur d'autres machines sur le réseau. Ce découpage est complètement transparent pour les utilisateurs du système de fichiers. Les différentes parties peuvent être connectées au démarrage du système ou à la demande, en cours d'utilisation.

Le répertoire /etc

Ce répertoire contient les fichiers de configuration du système. On trouve les sous-répertoires suivants :

Répertoire	Contient
./X11	les fichiers de configuration de Xwindow
./rc.d	les scripts de démarrage du système
./logrotate.d	les fichiers de configuration de logrotate pour les paquetages
./cron	les tâches à effectuer à la périodicité donnée (daily, hourly, monthly, weekly)
./skel	les fichiers à recopier dans le répertoire d'un nouvel utilisateur
./sysconfig	les fichiers de configuration des périphériques

Le répertoire /home

Le répertoire /home contient les répertoires des utilisateurs logés sur cette machine. Chaque utilisateur possède son propre répertoire, mais il est possible de créer des structures de groupes de travail en organisant les sous-répertoires.

Le répertoire /usr

Le répertoire /usr contient de nombreux sous-répertoires. On retrouve presque la même organisation que sous la racine, mais le contenu est destiné aux utilisateurs plus qu'au système lui-même.

La structure de /usr est la suivante :

Répertoire	Contient
./X11R6	la hiérarchie des fichiers Xwindow (version 11 révision 6)
./bin	les commandes du système
./doc	les documentations en ligne
./etc	la configuration des commandes utilisateurs
./games	les jeux
./include	les fichiers de définition des librairies pour la programmation en C
./lib	les librairies non système
./local	la hiérarchie des fichiers propres à cette installation
./man	les fichiers des pages du manuel en ligne

./sbin	les commandes d'administration non nécessaires au démarrage
./share	les fichiers de configuration partagés
./src	les sources du système et des applications

Xwindow est l'environnement graphique d'Unix. Il est très flexible et configurable et possède des fonctionnalités rarement vues dans d'autres environnements. On retrouve, dans ce répertoire, une hiérarchie de fichiers ressemblant à celle de la racine, mais dédiée à l'environnement Xwindow.

/usr/local doit contenir les outils installés en dehors du contexte de la distribution. On retrouve une hiérarchie complète semblable à /usr.

Le répertoire /var

Le répertoire /var contient les données variables du système, c'est-à-dire les fichiers propres à l'installation réalisée sur cette machine.

Répertoire	**Contient**
./catman	les fichiers d'aide mis en forme
./lib	quelques fichiers de configuration
./lock	les fichiers de verrous des applications
./log	les fichiers d'enregistrement des traces
./run	les fichiers contenant les "pid" des processus du système
./spool	les fichiers du spooler et de la messagerie

Le répertoire catman contient les pages de manuel mises en forme pour un accès plus rapide lors d'une deuxième utilisation.

Le répertoire log contient les fichiers de trace de fonctionnement du système. Une grande partie du travail d'administration consiste à suivre les enregistrements afin de détecter les mauvais fonctionnements. Le programme logrotate permet de conserver un historique des fichiers. Il existe des outils de gestion des fichiers de trace pour permettre, entre autres, la détection des intrusions sur le système.

Le répertoire spool contient des sous-répertoires de gestion des spoolers d'impression (lpd), de courriers (mail), de forums (news), etc. Ces sous-répertoires peuvent contenir, momentanément, des fichiers de taille importante.

III.2. Types de fichiers et Permissions d'accès aux fichiers

Types de fichiers

Les différents types de fichiers sous Linux sont :

- les fichiers normaux : ce sont des collections d'octets. Il n'existe pas de différence entre les fichiers texte et les fichiers binaires.
- les répertoires : ce sont des fichiers contenant les noms des fichiers et leurs numéros d'inode.
- les liens symboliques : permettent de présenter une image d'un fichier sous un autre nom ou à un autre endroit sans dupliquer les données.
- les fichiers spéciaux en mode bloc : sont les portes sur les périphériques fonctionnant par blocs de données (ex : disques).
- les fichiers spéciaux en mode caractère : sont les portes vers les périphériques fournissant ou consommant les données octet par octet.
- les tubes nommés "fifo" : permettent à deux processus sans relation de parenté de s'échanger des données comme par un tube.

Attributs des fichiers

Pour afficher les attributs principaux des fichiers, il faut utiliser l'option `-l' de la commande **ls** :

```
-rw-r--r--      2       root    root    6656    Apr 15  1998    fichier
prw-r--r--      1       root    root      0     Apr 15  1998    fifo
brw-r--r--      1       root    root      0     Apr 15  1998    bloc
crw-r--r--      1       root    root      0     Apr 15  1998  caracteres
drwxr-xr--      1       root    root    1024    Nov 12 19:42 répertoire
```

Cet affichage présente beaucoup d'informations :

- le premier caractère donne le type du fichier :
    - `-` pour un fichier normal
    - `p' pour un fifo
    - `b' pour un fichier spécial en mode bloc
    - `c' pour un fichier spécial en mode caractère
    - `d' pour un répertoire
- les neuf caractères suivants donnent les droits d'accès (voir plus loin)
- le champ suivant donne le nombre de liens sur le fichier
- on trouve ensuite le nom du propriétaire et du groupe du fichier
- le champ suivant donne la taille en octets du fichier
- la date de la dernière modification est indiquée selon deux formats :
    - avec l'année pour les fichiers vieux de plus de 6 mois ou de plus d'une heure dans le futur
    - avec l'heure pour les autres cas
    - enfin, le nom du fichier.

La commande **stat** permet d'afficher plus d'informations sur un fichier.

```
bash$ stat file1
  File: "file1"
  Size: 3562         Filetype: Regular File
  Mode: (0777/-rwxrwxrwx)    Uid: ( 500/ sandra)  Gid: ( 500/ sandra)
Device:  8,0   Inode: 2043      Links: 1
Access: Wed Nov 18 18:52:42 1997(00000.00:26:18)
Modify: Wed Nov 18 18:52:42 1997(00000.00:26:18)
Change: Wed Nov 18 18:52:59 1997(00000.00:26:01)
```

Permissions d'accès aux fichiers

Tout fichier du système appartient à la fois à un utilisateur (son "propriétaire") et à un groupe.
Ainsi, pour chaque fichier le monde de ses utilisateurs potentiels est scindé en 3 catégories, nommées :

1. **u**, l'**u**tilisateur normal, son propriétaire, bien souvent son créateur, qui n'a pas pour autant tous les droits sur lui !

2. **g**, son **g**roupe, ensemble d'utilisateurs ayant parfois des "permissions" particulières.

3. **o**, tous les (**o**thers) autres.

   Attention, l'utilisateur propriétaire et le groupe propriétaire du fichier peuvent être indépendants :

- le groupe propriétaire n'est pas forcément le groupe primaire de l'utilisateur propriétaire,

- et même, le propriétaire n'est pas forcément membre du groupe !

Mais (heureusement) une règle générale simple s'applique à la création de tout nouveau fichier (ou rép)

- son propriétaire est l'utilisateur (humain ou système) qui l'a créé

- son groupe est le groupe *primaire* de ce même utilisateur

Droits d'accès des utilisateurs aux fichiers

Généralités

Linux permet de spécifier les droits d'action sur un fichier, que peuvent exercer les utilisateurs des 3 catégories précédentes, ou plutôt les ***permissions*** que leurs accordent les fichiers et les répertoires.

Linux a repris les 3 protections d'UNIX sur les fichiers et les répertoires. Leur notation symbolique est :

1. **r,** lecture

2. **w**, écriture

3. **x**, exécution

De façon générale, ces permissions sont consultables complètement par la commande : **ls -l**
Rappel : *ll* est un alias plus court, pour la commande *ls -l*

Par exemple :

```
[stagex@p0x  stagex] ll *.html
-rw-r--r--  1 stagex    stagex   1200  oct 19 12 : 39     amoi.html
```

Description globale
On trouve de gauche à droite

- le 1er caractère indique la nature du fichier
  "**-**" fichier normal, "**d**" un fichier répertoire, "**l**" un lien.

- le système de droits est spécifié symboliquement par les 9 attributs suivants, correspondants aux 3 catégories d'utilisateurs du fichier.
  ...|...|...
  u g o

  La section u fixe les droits accordés au propriétaire du fichier.
  La section g fixe les droits accordés aux utilisateurs faisant partie du groupe auquel appartient le fichier.
  La section o fixe les droits des autres utilisateurs.

- nombre de liens sur le fichier
  1 signifie que le fichier n'a aucun lien qui pointe vers lui, 2 (ou plus) signifiant qu'il existe un lien (ou plus) vers lui.

- le nom du propriétaire du fichier

- le nom du groupe propriétaire

- la date de dernière modification

- le nom complet du fichier

Permissions des fichiers normaux

Pour chaque fichier, les utilisateurs sont ainsi séparés en 3 catégories, le propriétaire, les membres du groupe et tous les autres.
Les permissions accordées par le fichier à ces catégories sont complètement indépendantes mais leur signification est la même.
Vis à vis de chacune de ces 3 catégories, on trouve dans l'ordre :

- le droit de lecture , afficher son contenu --> "r" si permis , "-" si refusé

- le droit d'écriture , modifier son contenu --> "w" si permis , "-" si refusé

- le droit d'exécution , pour un fichier script ou binaire --> "x" si permis , "-" si refusé

Exemples :

- Le fichier de démarrage /etc/rc.d/rc.sysinit possède les droits rwx r-x r-x
  Tous les utilisateurs ont donc le droit de lire et d'exécuter ce fichier (ce qui est à éviter); seul root peut le modifier

- La table de montage /etc/fstab : rw-r--r-- peut être lue par tous, modifiée uniquement par root

Permissions des répertoires

Pour les fichiers de type répertoire, la signification des attributs est différente de celle d'un fichier normal.
Mais elle est toujours identique pour les 3 catégories d'utilisateurs du répertoire.
La présence d'un tiret "-" signifie toujours l'absence complète de droits

- **r** : lire le contenu, la liste des fichiers (avec ls ou dir)

- **w** : modifier le contenu : droits de créer et de supprimer des fichiers dans le répertoire (avec cp, mv, rm)

- **x** : permet d'accéder aux fichiers du répertoire et de s'y déplacer (avec cd).Si on attribue **w**, il faut attribuer aussi **x** sur le répertoire.

Exemples :

Passer les commandes **cd /** puis **ls -l**, pour lister les répertoires situés à la racine.

- A qui appartiennent-ils ? Un user quelconque peut-il y créer des sous-rép. ?

- Commenter les 2 cas particuliers **/root** et **/tmp**

Attention !
on voit que le droit **w** est très étendu, et même dangereux quand il est accordé à un groupe, car un membre du groupe peut supprimer des fichiers dont il n'est pas propriétaire et sur lesquels il n'a même pas de droit d'écriture !

Remarque
Le droit **x** sur un répertoire est un préalable indispensable pour qu'un utilisateur (de la catégorie correspondante au positionnement du **x**), puisse exercer d'éventuels droits sur les fichiers contenus dans le répertoire.

Changements des droits

De façon générale, l'utilisateur qui crée un fichier en devient le propriétaire, et le groupe auquel l'utilisateur appartient (au moment de la création) devient le groupe du fichier.

Remarques préalables

- Mais les droits accordés au propriétaire, au groupe et aux autres dépendent du processus qui a créé le fichier et du masque des droits.

- D'autre part l'administrateur peut être amené à effectuer des changement de propriété (par exemple pour permettre un travail en groupe) et des changements de droits sur des ensembles de fichiers et de répertoires , les étendre ou les restreindre.

- Et root n'est pas soumis à ces restrictions, il a le pouvoir absolu sur ... le système de fichiers. En contre-partie il peut être considéré comme responsable de tout dysfonctionnement !

Changer le propriétaire ou le groupe propriétaire

- Changer le propriétaire
  chown [-R] nv-user fichiers
  Commande réservée au propriétaire actuel des fichiers ou des répertoires (et à root)
  L'option **-R** (récursif) permet d'agir sur l'ensemble des sous-répertoires.
  Exemple : chown -R stage4 /home/stage1

- Changer le groupe propriétaire
  chgrp [-R] nv-groupe fichiers
  Ceci doit être effectué par root ou le propriétaire, à condition que celui-ci soit membre du nouveau groupe.
  Exemple : chgrp -R stage4 /home/stage1

- Changer les 2 en même temps
  chown nv-user.nv-groupe fichiers
  chown new-user.fichiers
  Dans ce cas, en plus, le groupe propriétaire des fichiers est changé pour le groupe primaire du nouveau propriétaire.

Changer les permissions sur les fichiers

- Les droits d'accès peuvent être modifiés par le propriétaire des fichiers ou par root (ou équivalent, d'uid 0).

- La commande **chmod** (*change mode*, change le "mode" des fichiers) peut s'écrire de plusieurs façons équivalentes, sur le modèle :
  chmod *droits* fichiers
  Le paramètre *droits* permet de calculer les nouveaux droits d'accès.

- Ceux-ci peuvent s'obtenir de façon *relative*, par ajout (symbole +) ou retrait (-) par rapport aux droits existants, ou bien de façon *absolue*, en fixant les nouveaux droits qui remplacent les anciens (symbole =).

Ajout, retrait ou fixation des permissions

Pour chaque fichier, on désigne par :

- **u, g et o** les 3 catégories d'utilisateurs (user, group, other) et de plus par **a** (=all) tous les utilisateurs.

- **r,w,x** les 3 attributs de chaque fichier, pour chaque catégorie d'utilisateur.

- **+ - =** l'action d'ajouter, de retirer ou de fixer un droit, qui s'applique à chaque catégorie séparément.

- les changements, sur le modèle "*à quelle(s) catégorie(s), quelle action, quel(s) droit(s)*" sont alors notés symboliquement :
  [u g o a] [+ - =] [r w x]

- par exemple **chmod u+x fichier** signifie "ajouter le droit d'exécution au propriétaire du fichier"

- on peut regrouper les catégories si on veut exercer la même action :
  chmod ug+w fichier "ajouter le droit d'exécution au propriétaire et au groupe"
  chmod go-rwx fichier "enlever tous droits d'accès à tous les utilisateurs, sauf au propriétaire"

Notation relative (aux droits existants)

- **chmod [-R] <action-droits> fichiers**

- L'option -R (récursif) permet de modifier les permissions de tous les sous-répertoires.

- exemple : **chmod [-R] go-rwx /home/toto** enlève tous les permissions d'accès des fichiers du rép. personnel de toto (et des sous-rép.), à tous sauf au propriétaire, c'est-à-dire toto.

Notation absolue

- Pour chaque groupe, elle permet de fixer les nouveaux droits qui remplacent les anciens. Si une catégorie n'est pas présente, ses anciens droits s'appliquent.

- **chmod u=rwx,g=rw,o=r fichiers** remplace les permissions précédentes des fichiers, en les fixant à -rwxrw-r--
  Attention : aucun espace dans la liste des droits, pas même autour des éventuelles virgules

- **chmod u=rwx,g=r fichiers** fixe les permissions à -rwxr--??? en ne changeant pas les permissions précédentes du groupe *other*

- **chmod u=rwx,g=r,o= fichiers** fixe les permissions à -rwxr-----

Remarque importante

Le "super-utilisateur" root n'est pas soumis aux restrictions des permissions. Une petite expérience :

1. Vérifier que /etc/shadow est inaccessible même en lecture aux utilisateurs

2. Vérifier que ses permissions sont --------- ou 400 en octal, seul le propriétaire root peut lire

3. Root supprime ce droit de lecture : chmod u-r /etc/shadow
   Vérifier /etc/shadow

4. Root peut le lire, le copier et le modifier, ce n'est bien sûr pas recommandé, mais root peut tout se permettre (raison de plus pour ne jamais se connecter root, sans nécessité !)

5. Mais bonne nouvelle, root peut donc retrouver de fichiers appartenant à des utilisateurs ayant perdu leurs droits d'accès !

6. [stagex@p00 stagex]$ cp ./bashrc ./bashrc1
7. [stagex@p00 stagex]$ chmod ugo= ./bashrc1 *aucune permission sur le fichier !*
8. [stagex@p00 stagex]$ cat ./bashrc1 *bien sûr il est totalement protégé en lecture*
9. [root@p00 stagex]# cat ./bashrc1 *mais pas pour root !*

Notation octale des permissions

Il existe une autre facon d'indiquer les permissions de chaque catégorie, plus simple en utilisant la numération octale

Voici la table de correspondance entre les 8 chiffres en numérotation octale (base 8) et les 8 valeurs de droits fichiers.
Par convention la présence d'un droit est noté 1, l'absence 0.

```
Binaire ----- Droit ----- Octal
 000 -------- (---) ------- 0
 001 -------- (--x) ------- 1
 010 -------- (-w-) ------- 2
```

```
011 -------- (-wx) ------- 3
100 -------- (r--) ------- 4
101 -------- (r-x) ------- 5
110 -------- (rw-) ------- 6
111 -------- (rwx) ------- 7
```

Synthèse : notation globale pour les 3 catégories

propriétaire			**Groupe**			**autre**		
lecture	écriture	exécution	lecture	écriture	exécution	lecture	écriture	exécution
400	200	100	40	20	10	4	2	1

Pour obtenir les permissions exprimées en octal, il suffit d'ajouter en octal les nombres de la table de correspondance ci-dessus, pour lesquels les droits sont positionnés.

Exemples

chmod 700 /home/rep-a-moi *droits par défaut pour un rép. personnel*.
ls -l /home/rep-a-moi
--> drwx------

Les 2 commandes suivantes sont équivalentes :

```
chmod 764 test
chmod u=rwx,g=rw,o=r test
ls -l test
 -rwxrw-r--
```

Le masque de protection umask

- Rappelons les règles simples de propriété qui s'appliquent à la création d'un fichier ou d'un répertoire :
  - son propriétaire est l'utilisateur qui l'a créé
  - son groupe est le groupe *primaire* de ce même utilisateur
- Mais quelles sont les permissions attribuées *par défaut* à l'utilisateur propriétaire, au groupe propriétaire et à tous les autres ?
  Les permissions maximales accordées par un fichier et un répertoire sont 666 (-rw-rw-rw-) et 777 (-rwxrwxrwx).
  On peut restreindre ces permissions lors de sa création. C'est le rôle de la commande **umask** de fixer les permissions *masquées*, autrement dit **les droits non accordés aux fichiers et répertoires lors de leur création**.

- Exemple de calcul de permissions effectives, affectées lors de la création d'un répertoire, par un utilisateur dont le masque de protection est **027**

- 777 = 111 111 111 permissions maxi = rwx rwx rwx
- \- 027 = 000 010 111 masque de protection
- = 750 = 111 101 000 permissions effectives = rwx r-x ---

- La commande umask

  - **umask** affiche le masque de l'utilisateur actif
    Quelles sont les valeurs des masques par défaut de root et des autres utilisateurs ?

  - **umask -S** affiche les permissions correspondantes au masque, sous forme symbolique.

  - **umask *masque*** fixe les permissions ultérieures de création des fichiers de l'utilisateur actif, conformément à *masque*, en notation octale.
    Attention ! le changement ne s'applique qu'à la présente session.

  - Pour la rendre permanente, on peut intervenir sur un fichier *profile* :

    Dans le fichier profil général **/etc/profile**, on peut modifier la règle habituelle :
    if [$UID == 0] ; then umask 022 ; else umask 077 ; fi

    Pour agir au niveau des utilisateurs, ajouter la ligne umask *masque* dans le fichier de profil personnel $HOME/.bash_profile

Les droits étendus : Le droit SUID

- Sa présence permet à un fichier exécutable de **s'exécuter sous l'identité et donc les droits de son propriétaire**, à la place des droits de l'utilisateur actuel qui l'exécute.

- Il s'agit d'un dispositif de sécurité essentiel qui autorise un utilisateur quelconque (par rapport à la commande) à bénéficier de droits plus étendus que les siens (souvent ceux de root), pour exécuter la commande agir sur d'autres fichiers indispensables, juste le temps et sous le contrôle de l'exécution de la commande, SANS qu'il soit nécessaire d'attribuer ces droits en permanence sur les fichiers.

- Ce droit est noté symboliquement **s** et se positionne à la place du x du propriétaire u (mais sans écraser le droit x)
  Sa valeur octale est 4000

- **Exemple significatif**
  Examiner les droits du fichier exécutable **/usr/bin/passwd**, qui permet de (re)définir un mot de passe et le comparer à ceux du fichier **/etc/shadow** qui contient les mots de passe cryptés.

- Observez :
- ll /etc/shadow
- -r-------- root root shadow
- ll -l /usr/bin/passwd
- -r-**s**r-xr-x root bin passwd

  Comme le droit **x** est accordé à tous, chacun peut donc exécuter la commande passwd, mais personne ne posséde pas lui-même le droit d'écriture dans le fichier /etc/shadow qui doit le stocker.
  Le positionnement du SUID permet d'agir en tant que root lors de la demande d'accès au fichier et comme root a tous les droits, il est alors possible de mettre à jour ce fichier des mots de passe.

- Comment connaitre les commandes comme passwd, qui offre cette permission SUID ? Voici plusieurs façons

- cd /usr/bin
- *# grep filtre les lignes produites par ls en utilisant*
- *# l'expression rationnelle **^...s***
- ls -l | grep "^...s"
- *# pour afficher tous les fichiers possédant le SUID*
- cd /
- ls -lR | grep "^...s"
- *# recherche parmi les fichiers ordinaires ceux qui ont au moins le droit **s***
- find / -type f -perm +4000

Les droits étendus : Le droit SGID

- Pour un fichier exécutable, il fonctionne de la même façon que le **SUID**, mais transposé aux membres du groupe.
  Exemple

  - Examiner les droits symboliques de la commande d'impression **/usr/bin/lpr**

  - Quelle est sa valeur octale ?

  - Si une imprimante a été installée, un répertoire **lp** a été créé dans /var/spool/lpd . Or la commande lpr écrit dans ce

répertoire. Comment un utilisateur quelconque peut-il alors y écrire le fichier d'impression ?

- Positionné sur un répertoire, ce droit modifie le groupe propriétaire d'un fichier créé dans ce répertoire.
  Un fichier créé dans un tel répertoire, verra son groupe propriétaire modifié :
  Ce ne sera plus le groupe primaire du propriétaire qui l'a créé (règle habituelle), mais à la place, le groupe propriétaire du répertoire lui-même.
  Autrement dit, ce droit **s** posé sur un répertoire, met en place un mécanisme d'héritage de groupe, de répertoire conteneur à fichiers contenus.

- Notation symbolique **s**, mis à la place du x du groupe, valeur octale 2000

Le "sticky bit"

- Ce droit spécial, traduit en "bit collant", a surtout un rôle important sur les répertoires.
  Il réglemente le droit **w** sur le répertoire, en **interdisant** *à un utilisateur quelconque de* **supprimer un fichier dont il n'est pas le propriétaire**

- Ce droit noté symboliquement **t** occupe par convention la place du droit x sur la catégorie other de ce répertoire, mais bien entendu il ne supprime pas le droit d'accès x (s'il est accordé).
  Justement, si ce droit **x** n'est pas accordé à la catégorie other, à la place de **t** c'est la lettre **T** qui apparaitra.
  Sa valeur octale associée vaut 1000.

- Pour positionner ce droit :

- **chmod +t rep**
- --> d ... .t rep *si le répertoire a le droit x pour tous*
- --> d ... .T rep *sinon*

- Exemple
  Le sticky bit est présent sur le rép. /tmp
  Pour quelle raison ?

- ls -l /
- ....
- drwxrwxrw**t** root root tmp/

III.3. Outils de gestion de l’arborescence

Commande	Description
pwd	affiche le chemin du répertoire courant
cd	change le répertoire courant (commande interne du shell)
basename	extrait le nom du fichier d'un chemin complet
chmod	modifie les droits d'un fichier
chgrp	change le groupe propriétaire du fichier
chown	change l'utilisateur propriétaire du fichier
cp	copie de fichiers
ls	affiche la liste des fichiers d'un répertoire
mkdir	crée des répertoires
mknod	crée un nom de fichier temporaire unique
rm	détruit des fichiers
rmdir	détruit des répertoires
mv	déplace (ou renomme) des fichiers
touch	met à jour les dates d'accès des fichiers
df	affiche la place disque disponible
du	donne la place disque utilisée par le répertoire courant
file	donne le type de fichier
mtools	ensemble d'outils pour la gestion des disquettes MS-DOS
mdir	affiche la liste des fichiers d'une disquette MS-DOS

III.4. Visualisation et édition de texte

Commande	Description
cat	concatène les fichiers sur la sortie standard
sort	trie les lignes de texte en entrée
sed	effectue des modifications sur les lignes de texte
more	affiche à l'écran l'entrée standard en mode page par page
less	affiche à l'écran l'entrée standard en mode page par page avec des possibilités de retour en arrière
cut	permet d'isoler des colonnes dans un fichier
expand	transforme les tabulations en espaces
head	affiche les n premières lignes

join	permet de joindre les lignes de deux fichiers en fonction d'un champ commun
paste	concatène les lignes des fichiers
tail	affiche les n dernières lignes du fichier
tac	concatène les fichiers en inversant l'ordre des lignes
uniq	élimine les doublons d'un fichier trié
rev	inverse l'ordre des lignes d'un fichier
pr	formate un fichier pour l'impression
tee	écrit l'entrée standard sur la sortie standard et dans un fichier
tr	remplace ou efface des caractères

III.5. Travailler avec l'éditeur de texte : vi

vi est un éditeur de texte en mode écran qu'il faut absolument connaître en tant qu'administrateur car c'est souvent le seul disponible sur les machines. Il est peu convivial mais extrêmement puissant de part la rapidité de ses commandes.

Le lancement de l'éditeur s'effectue par :

```
vi [fichier]
```

L'éditeur fonctionne avec deux modes principaux :

- un mode commande pour se déplacer dans le texte et passer les commandes de type ex.
- un mode saisie de texte dont on sort en appuyant sur <ESC>.

Les commandes de déplacement sur le texte sont nombreuses :

Commande	**Fonction**
h	déplace le curseur d'un caractère vers la gauche
l	déplace le curseur d'un caractère vers la droite
k	déplace le curseur d'une ligne vers le haut
j	déplace le curseur d'une ligne vers le bas
w	déplace le curseur au début du mot suivant
b	déplace le curseur au début du mot précédent
W	déplace le curseur au début du mot suivant séparé par des blancs

B	déplace le curseur au début du mot précédent séparé par des blancs
e	déplace le curseur à la fin du mot
E	déplace le curseur à la fin du mot séparé par des blancs
<CTRL>-f	affiche la page suivante
<CTRL>-b	affiche la page précédente
<CTRL>-d	affiche la demi-page suivante
<CTRL>-u	affiche la demi-page précédente

Voici les quelques commandes nécessaires pour saisir et modifier du texte :

Commande	Fonction
I	passe en mode insertion avant le caractère courant
A	passe en mode ajout après le caractère courant
o	ajoute une ligne après la ligne courante et passe en insertion en début de ligne
O	ajoute une ligne avant la ligne courante et passe en insertion en début de ligne

Les commandes d'effacement de texte placent le texte détruit dans un tampon en mémoire. Ceci permet d'effectuer des annulations de modifications ainsi que des déplacements de texte.

Commande	Fonction
x	détruit le caractère courant
dd	détruit la ligne courante
y	place le caractère courant dans le tampon
yy ou Y	copie la ligne entière dans le tampon
P	copie le contenu du tampon avant le curseur
p	copie le contenu du tampon après le curseur
u	annule la dernière modification apportée au texte

Les commandes de manipulation de fichiers permettent la lecture, l'écriture et la concaténation de fichiers :

Commande	Fonction
:w	écrit dans le fichier en cours
:w nom	écrit dans le fichier `` nom ''
:q	quitte l'éditeur
:wq	sauve et quitte
:e nom	lit le fichier `` nom ''
:e #	lit le fichier précédent
:n	lit le fichier suivant sur la ligne de commande

III.6.Travailler avec l'éditeur de texte : Emacs

Emacs possède différentes utilités suivant les personnes qui l'utilisent. Suivant la personne à qui vous posez la question, vous pourrez obtenir l'une des réponses suivantes:

- Un éditeur de texte
- Un client de Messagerie
- Un lecteur de News
- Un traitement de texte
- Une religion / un sacerdoce
- Un environnement de développement intégré
- Tout ce que vous voulez qu'il soit!

Mais pour notre démonstration, prétendons seulement qu'il s'agit d'un éditeur de texte, un éditeur de texte étonnamment flexible. Nous approfondirons la question un peu plus tard. Emacs a été écrit par Richard Stallman (fondateur de la Free Software Fondation : http://www.fsf.org/ et du projet GNU http://www.gnu.org/) et qui le maintient encore aujourd'hui.

Emacs est l'un des outils d'éditions de textes les plus populaires utilisés sur Linux (et Unix). C'est le second en popularité après **vi**. Il est aussi connu pour ses incommensurables possibilités, sa capacité à être customisé, et son manque de bug (son absence de bug)! Ses larges possibilités et sa capacité à être réellement configuré sont le résultat de la conception et de l'implémentation d'Emacs. Sans aller dans les détails, nous dirons qu'Emacs n'est simplement pas **juste un éditeur**. C'est un éditeur écrit principalement en **Lisp**. Dans le noyau d'Emacs il y a un interpréteur contenant toutes les subtilités de Lisp écris en C. Seulement les parties les plus basiques et les pièces de bas-niveau d'Emacs sont écrites en C. La majorité de l'éditeur est en réalité écrit en Lisp. Donc, dans un sens, Emacs possède un langage de programmation entier incorporé, que vous pouvez utilisez pour customiser, pour étendre, et pour changer son comportement.

Emacs est aussi l'un des plus vieux éditeurs. Le fait est qu'il a été utilisé par des milliers de programmeurs durant les 20 dernières années signifie qu'il existe énormément de modules externes. Ces modules vous permettent de faire avec Emacs ce que Stallman n'a probablement jamais rêvé être possible de faire lorsqu'il a commencé à travailler sur Emacs. Nous parlerons de cela dans une prochaine section.

Il y a beaucoup d'autre sites Web et de documents qui donnent un meilleur aperçu d'Emacs, de son histoire, et des sujets associés. Plutôt que de tenter de reproduire tout cela ici, je suggère que vous voyez par vous même quelques endroits listés dans la section Autres Ressources de ce document.

Exécuter Emacs : Lancer & quitter Emacs

En tant que nouvel utilisateur, vous allez probablement vouloir lancer Emacs pour vous amuser un peu et l'essayer. Une fois dans Emacs, l'une des première choses à savoir, c'est: comment sortir, vous ne serez peut-être pas capable de savoir le faire. Donc, si vous n'avez jamais utilisé Emacs avant, lancer vous maintenant. Sur le prompt du shell, tapez emacs et appuyez sur entrée. Emacs devrais se lancer. Sinon c'est qu'il n'est soit pas installé, soit absent de votre chemin(PATH).

Une fois que vous voyez Emacs, vous devez savoir comment sortir. Les commandes pour quitter Emacs sont C-x C-c. La notation C-x signifie que vous devez appuyez sur la touche Ctrl et sur la touche x, en même temps. Dans ce cas ci, vous devrez ensuite appuyez sur les touches Ctrl et c toujours en même temps, pour quitter Emacs.

Les commandes au clavier utilisées par Emacs semblent être vraiment étranges, bizarre, et peut être même inconfortable pour vous de prime abord, spécialement si vous êtes un utilisateur de vi. A la différence de vi, Emacs n'a pas de mode séparé pour éditer le texte, et exécuter des commandes.

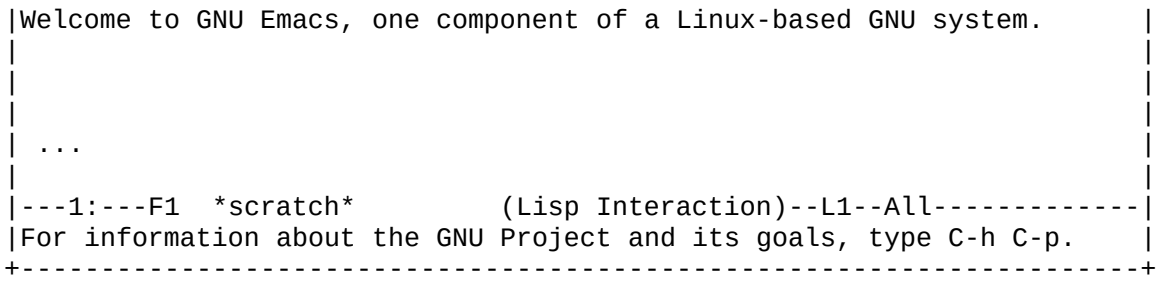
Pour récapituler: emacs lance Emacs. C-x C-c termine Emacs.

Ce que vous allez voir

Quand Emacs se lance il prend la place entière de l'écran(en mode texte), ou bien d'une fenêtre. Vous verrez alors un menu en haut, un peu de texte au milieu, et 2 lignes en bas de l'écran.

Cela ressemblera à cette ébauche en ASCII:

```
+-----------------------------------------------------------------------+
|Buffers Files Tools Edit Search Mule Help                              |
|                                                                       |
```



NOTE: Emacs utilisera normalement la totalité de l'écran/fenêtre. J'ai réduit l'exemple ci-dessus dans un soucis de place. Vous aurez aussi un message de bienvenue dans Emacs lorsque vous le lancerez pour la première fois. Je l'ai aussi omis et l'ai remplacé par ``...`` à la place. Le message de bienvenue donne simplement la version courante d'Emacs que vous utilisez, ainsi que des liens vers la documentation en-ligne et des articles s'y rapportant.

La Barre de menu

La ligne la plus en haut de l'interface d'Emacs est un menu. Si vous êtes sous X, vous la reconnaîtrez comme étant un menu traditionnel que vous pouvez utilisez en vous servant de votre pointeur. Sinon , vous devrez utiliser les raccourcis claviers(non expliqués ici) pour acceder aux menus.

La Barre de status & et le Mini-buffer

Sur les deux dernières lignes de l'interface d'Emacs, la plus haute et principalement une barre de status. Elle contient des informations a propos du buffer sur lequel vous travaillez, dans quel mode Emacs se trouve, et d'autres informations variées sur d'autres choses. Pour l'instant, regardez juste ce qu'elle contient.

La plus basse des lignes est appelé **mini-buffer**. Elle est séparée du buffer principal par la barre de status dont nous venons juste de parler. Vous pouvez penser que le mini-buffer d'Emacs est une ligne de ``commande``. C'est ici que les commandes que vous donnez à Emacs apparaissent, et c'est aussi ici que les messages d'états sont imprimés en réponses aux actions que vous effectuez.

Vous trouverez que ce que je nomme la barre de status est couramment nommé mode line dans la documentation d'Emacs. C'est ici qu'Emacs affiche les informations à propos des modes courants que vous utilisez, ainsi que des choses tels que l'heure et la date courantes, le numéro de ligne, la taille du fichier, ainsi que tous ce que vous voudriez y voir afficher.

Quelques Terminologies

Cette section couvre les terminologies les plus simple d'Emacs que vous pourrez rencontrer quand lors de l'utilisation et des lectures à propos d'Emacs.

Tampons & Fichiers

A la différence d'autres éditeurs, quand vous ouvrez un fichier dans Emacs il ne reste pas ouvert tout le temps que vous travaillez dessus. A la place, Emacs lit celui-ci et le place dans un **tampon** en mémoire. Tant que vous éditerez le tampon et travaillerez avec les données rien ne sera changé sur le disque. Simplement, lorsque vous sauvegarderez le tampon que le fichier sur le disque sera modifié. Cette méthode possède des avantages et des inconvénients mais cela est important que vous compreniez qu'il fonctionne de cette manière.

Comme conséquence, vous verrez le terme ``buffer`` (NdT : Tampon) utilisé dans la documentation Emacs , dans les différents modes, dans les packages, etc... Il faut juste réaliser que buffer signifie ``une copie du fichier qui se situe en mémoire``. Il faut aussi signaler que le buffer ne pointe pas toujours sur un fichier spécifique. Souvent Emacs crée des buffers comme résultat d'une commande que vous avez exécuté, une liste de choix à effectuer, et ainsi de suite.

Points & Régions

Dans la terminologie d'Emacs, vous entendrez ou lirez souvent des références au sujet des **points**. En des termes simple le point signifie le curseur. La différence entre point et curseur n'est pas importante lorsque vous commencer avec Emacs. Mais si vous êtes curieux, pensez-y de cette manière: le curseur est la représentation visuelle du point. Le curseur est toujours sur une position d'un caractère précis du buffer courant. Le point, d'un autre coté, évolue dans l'espace *entre les caractères* du buffer. Donc vous pouvez dire que si le curseur est sur la lettre `e` dans le mot ``les`` alors le point est entre le `l` et le `s`.

Comme la plupart des éditeurs modernes, Emacs permet d'effectuer des opérations sur une section du buffer courant (indentation, correction orthographique, reformatage, couper, coller, copier, ...). Vous pouvez marquer un bloc de texte en utilisant le clavier ou la souris et effectuer alors des opérations uniquement sur la zone sélectionnée. Dans Emacs, ces blocs des textes sont nommés **régions**.

Fenêtres

Cela sera peut-être un peu confus pour n'importe qui ayant utilisé une interface graphique avant. Il faut juste se rappeler qu'Emacs a été développé longtemps avant que les interfaces graphiques et les gestionnaires de fenêtres soit populaires.

Une **fenêtre** dans Emacs est une zone de l'écran dans laquelle un buffer est affiché. Quand Emacs est lancé, vous avez seulement une fenêtre affichée. Quelques fonctions d'Emacs, comme la documentation et l'aide, ouvrent souvent (temporairement) une fenêtre supplémentaire dans votre écran d'Emacs.

Les fenêtres d'Emacs n'ont rien en commun avec celle de X-Window en terme d'interface graphique. Vous pouvez ouvrir d'autres fenêtre X-Window pour

afficher des buffers d'Emacs, par exemple pour comparer deux fichiers côte à côte. Ces nouvelles fenêtres X-Window sont référencées comme des **frames** dans les termes d'Emacs.

Frames

Dans Emacs, une **frame** est une fenêtre X-Window séparée dans laquelle un bufer d'Emacs est affiché. Mais les deux sont une partie de la même session d'Emacs. L'attitude est un peu la même que celle d'ouvrir plusieurs fenêtres d'Emacs lorsque vous tapez Alt+N.

Les commandes de bases au clavier

Cette section couvre les bases de la frappe au clavier pour Emacs. Comme tous les éditeurs puissant, tout ce qui peut être fait avec Emacs n'est qu'a quelques touches plus loin.

Si vous êtes un utilisateur de vi, la notion d'utilisez les touches k,j,l,h pour remonter ou descendre d'une ligne, avancer ou reculer sur celle-ci vous a sûrement pris un peu de temps. En fait cela vous a sûrement pris quelques heures pour y être habitué, à pouvoir vous déplacer normalement dans un fichier en utilisant les différentes combinaisons de touches disponibles dans vi.

Emacs est différent. Il existe différentes combinaisons et commandes à apprendre. Comme vi, vous aurez uniquement à maîtriser les commandes de bases pour pouvoir effectuer quasiment tous les travaux. Alors, au fil du temps, vous pourrez continuer tranquillement l'apprentissage et trouver de meilleures voies pour arriver à vos fins.

Les touches de commandes(Meta, Esc, Control, and Alt)

Comme vous allez bientôt l'apprendre, Emacs fait une énorme utilisation des combinaisons des touches multiples. Etant donné que ce n'est pas un éditeur possédant différents modes comme vi, vous n'aurez pas a penser en terme de mode de commande ou de mode d'édition avant de pouvoir bouger le curseur ou d'exécuter une commande. A la place, vous aurez juste à presser la bonne combinaison de touches et Emacs fera ce que vous lui aura demandé (normalement).

Les touches qu'Emacs utilise les plus sont le plus souvent les touches Control, référencée dans la documentation commme C et la touche Meta M. Comme peu d'ordinateur possède un clavier possédant cette touche, il faudra substituer à cette touche la touche Escape Esc ou la touche Alternate Alt. Dans la plupart des configurations les touches Esc et Alt font la même chose.

Donc lorsque vous verrez une référence dans n'importe quelles documentation sur Emacs à C-x f cela signifie ``appuyez sur control-x et ensuite f``. Et si vous voyez un référence à quelque chose de la forme M-x shell cela signifie ``appuyez sur alt-x et tapez le mot shell``.

Une commande trés utile pour les débutants est la commande M-x a propos ou la commande C-h a. apropos cherchera dans la documentation Emacs en ligne de toutes les fonctions comportant l'expression régulière que vous tapé. C'est la manière la plus simple de découvrir toutes les commandes relatives aux frames. Tapez simplement C-h a et puis frame

Se déplacer dans un Buffer

Maintenant que vous savez ce que signifie toutes ces abréviations fantaisistes signifient, voici la liste des commandes les plus populaires pour se déplacer dans un buffer.

Commandes Clavier	Action
C-p	Monter d'une ligne
C-n	Descendre d'une ligne
C-f	Avancer d'un caractère
C-b	Reculer d'un caractère
C-a	Début de ligne
C-e	Fin de ligne
C-v	Descendre d'une page
M-v	Monter d'une page
M-f	Avancer d'un mot
M-b	Reculer d'un mot
M-<	Début du buffer
M->	Fin du buffer
C-g	Finir l'opération courante

Et, comme vous pouviez vous y attendre, les flèches de direction fonctionnent comme vous vous l'espériez. Votre touche Backspace peut-être pas. Mais c'est une autre histoire.

Commandes Principales

Bon, maintenant que vous savez vous déplacer dans un buffer, qu'en est-il d'ouvrir et de fermer des fichiers, d'effectuer des recherches. Voici les commandes de base.

Avant de passer à ces commandes, je ferais le brievement le point sur son fonctionnement.

Toutes ces ``commandes claviers`` dans Emacs (celles avec M-x qqle chose ou C-qqle chose) ne sont juste que des raccourcis vers des fonctions internes a Emacs. Vous pouvez les appeler aussi en tapant M-x nomdelafonction et en appuyant sur Entrée. Vous pouvez aussi utiliser le raccourci clavier de cette fonction (si elle en a un).

Par exemple, la fonction Emacs qui sauvegarde le buffer vers le disque est appelée save-buffer. Par défaut, elle est aussi reliée à C-x C-s. Vous pouvez donc soit tapez M-x save-buffer et obtenir le même résultat.

Toutes les fonctions le plus communes possèdent un raccourci clavier par défaut. En voici quelques exemples.

Commandes

Clavier	Fonction	Description
C-x C-s	save-buffer	Sauvegarde le buffer courant sur le disque
C-x u	undo	Annule la commande précédente
C-c C-f	find-file	Ouvre un fichier
C-s	isearch-forward	Recherche en avancant une chaine de caractère
C-r	isearch-backward	Recherche en reculant une chaine de caractère
	replace-string	Recherche et remplace une chaine de caractère
	replace-regexp	Recherche et remplace une chaine de caractère en utilisant regexp
C-h t	help-with-tutorial	Utilisation du tutorial
C-h f	describe-function	Affiche l'aide pour une fonction
C-h v	describe-variable	Affiche l'aide pour une variable
C-h x	describe-key	Affiche le comportement d'une combinaison de touches
C-h a	apropos	Recherche l'aide pour string/regexp
C-h F	view-emacs-FAQ	Affiche la FAQ d'Emacs
C-h i	info	Affiche la documentation d'Emacs
C-x r m	bookmark-set	Configure un signet. Utile dans les recherches.
C-x r b	bookmark-jump	Déplacement vers un signet.

Comme vous utiliserez beaucoup de ces fonctions, vous remarquerer que beaucoup vous demanderons quelque chose. Cela aura toujours rapport avec le mini-buffer. C'est similaire d'utiliser la commande : dans vi ou dans la plupart des commandes que vous utilisez avec votre shell favori.

Emacs possède des centaines de fonctions de base. La liste au dessus n'est qu'un trés petit exemple de ce que j'utilise régulièrement. Voyez la documentation en ligne pour une liste plus complète des fonctions disponible et une documentation complète sur ce qui est mentionné au dessus.

La Complétion par la touche Tab

Comme la plupart des shells unix (bash, csh, tsch, ...) Emacs offre la possibilité d'utiliser la complétion grace à la touche Tab. En fait, la commande de complétion du bash a été modifié après celle d'Emacs, donc si vous utilisez cette possibilité dans bash vous serez chez vous.

Pour exemple, essayer M-x search et appuyez sur Tab. Emacs ajoutera un trait d'union pour indiquer qu'il existe plusieurs possibilitées mais qu'elles possèdent toutes un trait d'union comme prochain caractère. Tapez la touche Tab une fois de plus pour qu'Emacs affiche les suites possibles dans lesquelles vous pourrez choisir. Prenez note du fait qu'il fait cela dans une nouvelle fenêtre: une contient le buffer courant que vous éditer en ce moment, l'autre contient la selection des complétions possibles pour

``search-``. Vous pourrez alors appuyer sur C-g pour sortir du processus de selection et fermer cette nouvelle fenêtre.

Tutorials, Aides, & Informations

Emacs est installé avec un tutorial en ligne qui vous aidera à utiliser les fonctiones d'édition de base et les fonctions que tout le monde doit connaître. Il explique aussi comment utiliser les autres aides d'Emacs.

Je vous recommande vraiment de passer un peu de temps a travers ce tutorial si vous pensez à faire un serieux effort pour apprendre Emacs. Comme indiqué dans la table plus haut, vous pouvez entrer dans le tutorial par C-h t. Le tutorial est dirigé est ciblé pour les personnes qui commencent tout juste avec Emacs.

Si vous lancer Emacs sous X, Vous verez que le menu le plus en haut à droite est labelé Help (Aide). Lorsque vous explorerez le menu, vous verrez que certaines rubriques possèdent des raccourcis et ceux ci sont listés dans la partie droite du menu.

Enfin, pour voir le volume de documentation disponible avec Emacs, vous pouvez essayer M-x info ou C-h> i qui lancera Info, le navigateur dans la documentation d'Emacs.

IV. Travail avec les shells

IV.1. approfondissement et familiarisation avec les interprètes de commandes

Le shell

Un interpréteur de commandes (le "shell", la coquille qui entoure le "noyau" du système) est un programme qui sert d'intermédiaire entre l'utilisateur et le système d'exploitation.
Sa tâche essentielle est l'exécution de programmes.
Pour cela, il effectue (en boucle infinie) :

- *la lecture d'une ligne*
- *sa compréhension comme une demande d'exécution d'un programme avec d'éventuels paramètres.*
- *le lancement de ce programme avec passage des paramètres*
- *d'éventuelles redirections d'entrées-sorties*
- *les exécutions de scripts (fichiers de commandes)*

Démarrage du shell

- Lors de la création de son compte, un utilisateur est associé à un type de **shell**

- Lire le fichier **/etc/passwd** : le dernier champ contient le nom du fichier exécutable (shell par défaut) **/bin/bash**

- Le shell associé est ainsi lancé automatiquement dès la saisie du login utilisateur.

- Il poursuit sa configuration en exécutant des scripts globaux à tous les utilisateurs et des scripts liés au compte et qui permettent une personnalisation.

- Enfin, il affiche le prompt et se met en attente de la lecture d'une commande.

- Jusqu'à la commande **exit**, pour quitter le shell (ce qui équivaut à se déconnecter (logout))

Les scripts de connexion

1. d'abord le script **/etc/profile** *communs à tous les users y compris* root. On y trouve notamment la définition de **umask**

2. celui-ci cherche à exécuter tous les scripts /etc/profile.d/*.sh (parcourir alias.sh et numlock.sh)

3. puis il y a exécution de **$HOME/.bash_profile** (la variable $HOME contient le chemin vers le répertoire personnel). Il s'agit ainsi d'un fichier de démarrage personnel et paramétrable.

4. A son tour il exécute $HOME/.bashrc dans lequel il est recommandé de placer toutes les fonctions ou alias personnels (car .bashrc est exécuté dans tout shell)

5. Enfin le précédent script exécute /etc/bashrc, dans lequel on place les alias globaux et la définition symbolique du prompt $PS1

6. Puis le prompt utilisateur s'affiche et le shell attend une commande ...

Personnalisation des commandes bash

- **/etc/bashrc** étant le dernier script d'initialisation du shell bash, **root** peut y définir des alias globaux pour tous les utilisateurs

- Exemple avec **vi** (pour utiliser l'éditeur de Midnigth Commander lancer **mc**)

- ***# vi /etc/bashrc***
- alias lll="ll | less"
- alias x="startx"
- alias m="mc"
- ***:wq*** (pour écrire dans le fichier et quitter vi)

- Puis se reloguer (exit) pour que ces nouvelles commandes soient prises en compte par le nouveau shell.

IV.2. Travail sur les fichiers de configuration utilisateur

Personnalisation du login utilisateur

Chaque utilisateur peut ajouter des commandes shell au fichier de profil personnel, **~/.bash_profile**
Par exemple, voici ce que j'ai mis à la fin de ce fichier :

```
clear
salut="Bonjour $USER !\nJe te souhaite bon courage ...\n\
# le dernier \ pour pouvoir continuer la commande sur la ligne suivante
# $( ..) pour obtenir le résultat de l'exécution de la commande incluse
```

```
Nous sommes le $(date) "
# -e option indispensable pour interpréter les \n
echo -e $salut
```

Manipulation des variables d'environnement

Les systèmes Unix permettent de définir un environnement d'exécution pour chaque programme en cours d'exécution. L'environnement est un ensemble de paramètres, que l'on appelle les *variables d'environnement*, qui permettent de modifier le comportement du programme. Ces variables contiennent une valeur de type chaîne de caractères, dont la signification est propre à chaque variable. Il est d'usage que les noms des variables d'environnement soient écrits complètement en majuscules, mais ce n'est pas une obligation. Chaque programme est susceptible de reconnaître un certain nombre de variables d'environnement qui lui sont propres, mais il existe également des variables standard que tous les programmes utilisent. C'est notamment le cas de la variable d'environnement PATH, qui contient la liste des répertoires dans lesquels le système doit rechercher les programmes à exécuter. Cette variable permet donc de lancer les programmes en tapant simplement leur nom, et de laisser le système rechercher le fichier de ce programme dans chacun des répertoires indiqués par la variable d'environnement PATH.

Par défaut, les programmes sont lancés avec l'environnement du programme qui les lance, c'est à dire dans la plupart des cas l'environnement d'exécution du shell. Les programmes peuvent également définir de nouvelles variables d'environnement, qui seront ainsi accessibles par les programmes qu'ils lanceront eux-mêmes.

Comme tout programme, le shell dispose d'un environnement, qu'il utilise pour stocker ses propres variables. En effet, comme nous l'avons déjà signalé plus haut, le shell est bien plus qu'un interpréteur de commande : il est complètement programmable. Et en tant qu'interpréteur d'un langage de programmation, il fournit la possibilité de définir des variables de ce langage. Les variables du shell sont donc également des variables d'environnement, mais le shell ne les communique pas par défaut aux programmes qu'il lance. Pour être plus précis, le shell utilise deux environnements différents :

- son propre environnement, qui contient les variables d'environnement normales et ses propres variables ;
- l'environnement d'exécution, qui est l'environnement que le shell utilise pour lancer les programmes.

Il est très facile de définir une variable du shell. Pour cela, il suffit de lui affecter une valeur, à l'aide de la syntaxe suivante :

```
variable=valeur
```

où **variable** est le nom de la variable à définir, et **valeur** est la valeur que l'on désire lui affecter. Notez qu'il n'est pas nécessaire de fournir une valeur. Dans ce cas, la variable ainsi définie sera vide.
Par exemple, la ligne suivante :

```
BONJOUR="Bonjour tout le monde \!"
```

permet de définir la variable BONJOUR. Notez que la valeur est encadrée entre guillemets, car elle contient des espaces. Notez également que le caractère point d'exclamation ('!') est précédé d'un caractère antislash

d'échappement ('\'), car il a une signification particulière pour le shell. Ce caractère d'échappement permet simplement de signaler au shell qu'il ne doit pas interpréter le caractère qui le suit, et fait donc en sorte que le point d'exclamation fait partie de la chaînes de caractères à affecter à la variable. Bien entendu, le caractère antislash étant lui-même un caractère spécial pour le shell, et il doit lui-même être préfixé d'un autre antislash si l'on désire l'utiliser dans une chaîne de caractères.
La valeur d'une variable peut être récupérée simplement en préfixant le nom de la variable du symbole dollar ('$'). Ainsi, la commande suivante permet d'afficher le contenu de la variable **BONJOUR** :

```
echo $BONJOUR
```

Les variables ainsi définies ne font partie que de l'environnement du shell, elles ne sont donc pas accessibles aux programmes que le shell lance. Donc, si l'on relance un nouveau shell avec la commande suivante :

```
bash
```

et que l'on essaie de lire le contenu de la variable BONJOUR avec la commande echo, on obtient une chaîne vide. Ceci est normal, puisque le deuxième shell (c'est à dire celui qui est en cours d'exécution) n'utilise pas le même environnement que le premier shell. Vous pouvez quitter le nouveau shell avec la commande suivante :

```
exit
```

Dès lors, vous serez à nouveau dans le shell initial, et la variable **BONJOUR** sera à nouveau accessible.
Pour rendre une variable du shell accessible aux programmes que celui-ci peut lancer, il faut l'exporter dans l'environnement d'exécution. Ceci peut être réalisé avec la commande **export** :

```
export variable
```

où variable est le nom de la variable du shell à exporter dans l'environnement d'exécution.
La syntaxe précédente exporte de manière permanente les variables du shell. Mais il existe également une autre syntaxe, qui permet de ne définir des variables d'environnement que pour l'environnement d'exécution d'une seule commande. Cette syntaxe consiste simplement à préfixer la commande à exécuter par la définition de ladite variable. Par exemple, la commande suivante :

```
BONSOIR="Bonsoir tout le monde \!" bash
```

permet de lancer un shell et de lui communiquer la variable d'environnement BONSOIR. Cette variable ne sera définie que pour ce programme, si l'on quitte ce shell avec un exit, la variable BONSOIR ne sera plus définie.
Une variable peut être détruite à tout instant à l'aide de la commande unset. Cette commande prend en paramètre le nom de la variable à supprimer. Par exemple, la commande suivante supprime notre variable :

```
unset BONJOUR
```

Vous pouvez à tout moment visualiser l'ensemble des variables définies avec la commande **set**. Le tableau donné ci-dessous vous présentera les variables d'environnement les plus utilisées, et que la plupart des programmes utilisent pour permettre à l'utilisateur de modifier leur comportement :

Tableau 1. Variables d'environnements courantes

Nom	**Signification**
HOME	Chemin du répertoire personnel de l'utilisateur.
USER	Nom de login de l'utilisateur. Cette information est également disponible au travers de la variable d'environnement LOGNAME.
TERM	Type de terminal utilisé. La valeur de cette variable sert aux applications pour déterminer les caractérisques du terminal et ses fonctionnalités, afin d'optimiser leur affichage. La valeur de cette variable est souvent linux sur les consoles Linux, et xterm dans les émulateurs de terminal graphiques sous X11.
SHELL	Chemin sur le fichier de programme du shell actuellement utilisé. Sous Linux, il s'agit souvent du shell bash.
PATH	Liste des répertoires dans lesquels les programmes à exécuter seront recherchés. Cette liste ne doit pas contenir le répertoire courant (.), pour des raisons de sécurité de base (il suffit de placer un cheval de troie portant le nom d'une commande classique dans un répertoire pour que l'utilisateur le lance sans s'en rendre compte).
LD_LIBRARY_PATH	Liste des répertoires dans lesquels les librairies dynamiques seront recherchées si elles ne sont pas trouvables dans les répertoires classiques des librairies du système.

Nom	**Signification**
C_INCLUDE_PATH	Liste des répertoires dans lesquels le compilateur C recherchera les fichiers d'en-tête lors de la compilation des fichiers sources C. Cette liste doit contenir les répertoires additionnels, qui ne sont pas déjà pris en compte automatiquement par le compilateur C.
CPLUS_INCLUDE_PATH	Liste des répertoires dans lesquels le compilateur C++ recherchera les fichiers d'en-tête lors de la compilation des fichiers sources C/C++. Cette liste doit contenir les répertoires additionnels, qui ne sont pas déjà pris en compte automatiquement par le compilateur C.
LIBRARY_PATH	Liste des répertoires dans lesquels les librairies à utiliser lors de l'édition de lien des programmes doivent être recherchées. Cette variable n'est utilisée que par les outils de développement lors de la compilation de fichiers sources, et elle ne doit pas être confondue avec la variable d'environnement LD_LIBRARY_PATH, qui indique la liste des répertoires contenant les librairies dynamiques dans lequel l'éditeur de lien dynamique recherchera les librairies utilisées par les programmes lors de leur chargement. Les notions de fichiers sources et de compilation seront détaillées dans le Chapitre 8.
TMPDIR	Répertoire des fichiers temporaires. Par défaut, le répertoire des fichiers temporaires est le répertoire /tmp/, mais il est possible d'en changer grâce à cette variable d'environnement.

Nom	**Signification**
TZ	Définition de la zone horaire de l'utilisateur. Le système travaillant exclusivement en temps universel, chaque utilisateur peut définir sa propre zone horaire pour obtenir l'affichage des dates et des heures dans son temps local. Le format de cette variable d'environnement est assez complexe. Il est constitué de plusieurs champs séparés par des espaces, représentant successivement le nom du fuseau horaire (au moins trois caractères), le décalage à ajouter à l'heure universelle pour obtenir l'heure locale, le nom du fuseau horaire pour l'heure d'été, le décalage pour l'heure d'été, et les dates de début et de fin de l'heure d'été. Les décalages horaires doivent être exprimés avec un '+' pour les fuseaux horaires placés à l'ouest de Greenwich, '-' pour ceux situés à l'est. Les dates de début et de fin de la période d'heure d'été peuvent être exprimés de deux manières différentes. La première méthode est d'indiquer le numéro du jour dans l'année après la lettre 'J'. Ce numéro ne doit pas tenir compte du 29 février, même pour les années bissextiles. La deuxième méthode est d'indiquer le mois de l'année, la semaine du mois et le jour de la semaine, séparés par des '.', et après la lettre 'M'. Les mois sont comptés de 1 à 12, les semaines de 1 à 5 et les jours de 0 à 6, 0 étant le dimanche. Seul le premier champ est obligatoire, il est possible d'utiliser les noms de fuseau horaires définis par la librairie C. En France, on utilise normalement le fuseau CES (temps d'Europe centrale).

Nom	**Signification**
LANG	Valeur par défaut à utiliser pour les paramètres d'internationalisation des applications. Cette valeur sera utilisée pour les paramètres qui n'en définissent pas une explicitement. Elle doit être composée de deux codes à deux caractères, le premier indiquant la langue, et le deuxième le pays (car plusieurs pays peuvent parler la même langue, et un pays peut avoir plusieurs langues nationales). Pour la France, on utilise normalement la valeur « fr_FR ». Cette valeur peut être redéfinie par l'une des variables d'environnement décrites ci-dessous.
LC_MESSAGES	Valeur à utiliser pour déterminer la langue des messages. La valeur par défaut est spécifiée par la variable d'environnement LANG.
LC_TYPE	Valeur à utiliser pour déterminer les règles de classification des caractères. La classification des caractères permet de dire si un caractère est chiffre ou non, s'il est en majuscule ou en minuscule, etc… La valeur par défaut est spécifiée par la variable d'environnement LANG.
LC_COLLATE	Valeur à utiliser pour déterminer les règles de comparaison des caractères. La comparaison des caractères est utilisée pour les tris lexicographiques (tri par ordre alphabétique par exemple). La valeur par défaut est spécifiée par la variable d'environnement LANG.
LC_MONETARY	Valeur à utiliser pour déterminer l'emplacement et le caractère à utiliser pour le symbole monétaire du pays. La valeur par défaut est spécifiée par la variable d'environnement LANG.

Nom	**Signification**
LC_NUMERIC	Valeur à utiliser pour déterminer les conventions locales d'écriture des nombres (séparateurs décimal, format de la virgule, etc…). La valeur par défaut est spécifiée par la variable d'environnement LANG.

En résumé, le shell utilise les variables d'environnement du système pour gérer ses propres variables, et permet de les exporter vers l'environnement d'exécution qu'il communique aux commandes qu'il lance. Un grand nombre de variables d'environnement classiques sont reconnues par les programmes. Elles servent à paramétrer leur comportement. Nous reverrons ultérieurement quelques unes de ces variables lors de la configuration du système de base.

IV.3. Programmation shell de base

Caractère d'échappement et chaînes de caractères

Un certain nombre de caractères sont interprétés par le shell d'une manière spéciale. Nous en avons déjà vu quelques uns pour les redirections et les pipes, mais il en existe d'autres. Par conséquent, il faut utiliser une syntaxe particulière lorsque l'on désire utiliser un de ces caractères dans une commande du shell sans qu'ils soient interprétés par le shell. Pour cela, il suffit de faire précéder ces caractères du caractère d'échappement antislash (caractère de la barre oblique inverse, \). Ce caractère permet d'indiquer au shell que le caractère suivant doit être traité tel quel, et ne doit pas être interprété avec son sens habituel. Par exemple, pour créer un répertoire nommé <, on utilisera la commande suivante :

```
mkdir \<
```

Bien entendu, le caractère antislash peut lui-même être précédé d'un autre antislash, lorsque l'on veut l'utiliser en tant que caractère normal. Le caractère d'échappement antislash permet également, lorsqu'il est placé en fin de ligne, de supprimer le saut de ligne qui le suit. Ceci signifie qu'il permet de répartir une commande trop longue sur plusieurs lignes, à des fins de lisibilité. Vous trouverez quelques exemples de cette notation plus loin dans ce document, pour présenter des commandes trop longues pour tenir sur une page A4.

Il peut être relativement fastidieux de devoir taper des antislashes dans les chaînes de caractères qui contiennent beaucoup de caractères interprétables par le shell. C'est pour cela que le shell permet de définir des chaînes de caractères, dont il ignorera le contenu lors de l'analyse syntaxique. Ces chaînes de caractères sont simplement données entre guillemets simples (caractère '). Par exemple, la commande suivante :

```
MESSAGE='La syntaxe est A | B'
```

permet d'affecter la chaîne de caractère La syntaxe est A | B, contenant des espaces et le caractère | normalement utilisé par le shell pour les pipes, dans la variable d'environnement MESSAGE.

Note: Une chaîne de caractères commence par un guillemet et se termine par un guillemet. Les chaînes de caractères ne peuvent donc pas contenir de guillemet, même précédé d'un caractère d'échappement.
On veillera à ne surtout pas confondre les guillemets simples (caractère ') avec les guillemets inverses (caractère `), qui se ressemblent énormément dans certaines polices de caractères. Ces deux caractères ont une signification complètement différente. Le premier sert à définir des chaînes de caractères, et le deuxième à exécuter une commande et à en inclure le résultat dans une autre commande. Nous verrons plus loin comment utiliser ce type de guillemets.

Les guillemets simples sont donc très pratiques pour écrire simplement une chaîne de caractères, mais ne permettent pas de bénéficier des fonctionnalités de substitutions du shell, comme par exemple le remplacement d'une variable par sa valeur dans la chaîne de caractères. De plus, elles ne peuvent pas contenir de guillemets simples, puisque c'est leur caractère de terminaison. C'est pour ces raisons que le shell donne la possibilité de définir des chaînes de caractères plus souples, à l'aide des guillemets doubles (caractère "). Dans ces chaînes de caractères, la plupart des caractères normalement interprétés par le shell ne le sont plus, comme pour les chaînes de caractères utilisant les guillemets simples. Cependant, les caractères spéciaux $, ` et \ conservent leur signification initiale. Il est donc possible, par exemple, d'utiliser des variables d'environnement dans les chaînes de caractères de ce type :

```
echo "Mon nom est $USER"
```

Le caractère d'échappement antislash peut toujours être utilisé, en particulier pour insérer un caractère de guillemets double dans une chaîne de caractères. En effet, ce caractère marquerait la fin de la chaîne de caractère s'il n'était pas précédé d'un antislash.

Note: Remarquez que les guillemets et les caractères d'échappement ne sont utilisés que pour l'analyse de la ligne de commande. Une fois toutes les chaînes de caractères et toutes les substitutions traitées, les guillemets et les caractères d'échappement inutiles sont supprimés. En pratique, ce sont tous les caractères d'échappement et les guillemets qui restent après traitement de la ligne de commande et qui ne font pas partie du résultat d'une des substitutions. Ainsi, la commande suivante :

```
echo "Bonjour tout le monde"
```

a pour but de passer la chaîne de caractères Bonjour tout le monde en tant que premier (et unique) paramètre de la commande echo, puis de l'exécuter. Les guillemets ne font pas partie de la chaîne de caractères, ils ont été supprimés par le shell et seul le contenu de la chaîne sera effectivement affiché.
Notez que la commande précédente est très différente de celle-ci :

```
echo Bonjour tout le monde
```

même si le résultat est le même. En effet, cette dernière commande passe les chaînes de caractères Bonjour, tout, le et monde en tant que paramètres (4 au total) à la commande echo, alors que l'utilisation des guillemets permet de passer toute la phrase en un seul paramètre. On peut voir la différence en utilisant plus d'un espace entre chaque mot : les espaces superflus ne sont conservés que dans la première commande.

Les substitutions

L'une des fonctionnalités les plus puissante du shell est sans doute sa capacité à effectuer des substitutions d'expressions par leur valeurs. L'une des substitutions les plus courante est sans doute le remplacement d'une variable par sa valeur, mais le shell peut faire beaucoup plus que cela. Les lignes de commandes peuvent être écrites en utilisant différents types d'expressions spéciales, qui seront remplacées par leur valeur par le shell avant l'exécution de la commande. Ces expressions permettent de spécifier des motifs de chaîne de caractères, d'exprimer des chemins partiels sur des fichiers ou des répertoires, de récupérer la valeur des variables du shell, de calculer des expressions mathématiques, voire même d'inclure le résultat d'une autre commande dans la ligne de commande en cours.

Les mécanismes de substitution décrits ci-dessous sont présentés par ordre de priorité décroissante. Ceci signifie que si une expression substituable contient elle-même une autre expression substituable de priorité inférieure, cette expression sera remplacée après la substitution de l'expression contenante.

Substitution du nom d'utilisateur

Le caractère tilde ('~') est remplacé par le nom de l'utilisateur courant, ou, à défaut de nom, par le chemin sur le répertoire personnel de cet utilisateur. Il est possible de spécifier un autre utilisateur en donnant le nom de login de cet autre utilisateur immédiatement après le caractère tilde. Par exemple, la commande suivante :

```
cp *.txt ~jean
```

permet de copier tous les fichiers d'extension **.txt** dans le répertoire personnel de l'utilisateur **jean**.

Remplacements de variables

Comme il l'a déjà été indiqué plus haut, la valeur des variables du shell et des variables d'environnement peut être récupérée en préfixant le nom de la variable par le caractère dollar ('$'). En fait, cette écriture est l'une des formes les plus simples que peuvent prendre les substitutions de paramètres. En effet, il est possible de remplacer l'expression par une partie seulement de la valeur de la variable, ou une par une autre valeur calculée à partir de celle de la variable.

En pratique, les expressions utilisées par les substitutions de variables peuvent être relativement compliquées, et il peut être nécessaire de les isoler du reste de la ligne de commande à l'aide d'accolades. La syntaxe exacte complète de ce type de substitution est donc la suivante :

```
${expression}
```

où **expression** est l'expression qui définit la chaîne de remplacement à utiliser.

Si cette expression est un nom de variable, ce sera la le contenu de cette variable qui sera utilisée pour la substitution. Il est possible de fournir une valeur par défaut pour le cas où cette variable ne contient rien ou n'est pas définie. Pour cela, on utilisera la syntaxe suivante :

```
${variable:-valeur}
```

où **valeur** est la valeur par défaut à utiliser dans ce cas. Notez que dans la variable reste indéfinie après la substitution. Pour fixer la valeur de la variable à cette valeur par défaut en plus d'effectuer la substitution, on utilisera plutôt la syntaxe suivante :

```
${variable:=valeur}
```

valeur a toujours la même signification dans cette syntaxe.
Il est parfois préférable d'afficher un message d'erreur plutôt que de donner une valeur par défaut lorsqu'une variable n'est pas définie. Ceci peut se faire avec la syntaxe suivante :

```
${variable&colon;?message}
```

où **message** est le message à afficher dans le cas où la variable **variable** serait non définie ou de valeur nulle.
Si l'on veut tester si une variable est non définie et renvoyer une valeur spécifique si elle est définie, on utilisera plutôt la syntaxe suivante :

```
${variable&colon;+valeur}
```

où **valeur** est la valeur à renvoyer si la variable est définie. Si la variable n'est pas définie, la substitution sera faite avec la chaîne de caractères vide (l'expression complète sera donc supprimée).
Le shell permet également de faire la substitution avec une sous-chaîne de la valeur de la variable, à partir d'une position donnée et d'une longueur. La syntaxe à utiliser est donnée ci-dessous :

```
${variable&colon;position&colon;longueur}
```

où **position** est la position à laquelle commence la sous-chaîne à extraire, et **longueur** est le nombre de caractères à extraire. Ce dernier champ est facultatif (on ne mettra pas non plus les deux-points précédents si on décide de ne pas spécifier de longueur). Si on ne le précise pas, la sous-chaîne extraite sera constituée du reste de la valeur de la variable à partir de la position indiquée. La position quant à elle doit être positive ou nulle. Une valeur négative indique un point de départ correspondant au nombre de caractères correspondant à partir de la droite de la valeur de la variable. Si l'on veut obtenir la longueur d'une chaîne de caractères contenue dans une variable, on utilisera cette syntaxe :

```
${#variable}
```

où **variable** est toujours le nom de la variable.
Il est également possible de considérer que la valeur d'une variable est une chaîne de caractère préfixée d'une autre chaîne de caractères particulière. Le shell permet d'extraire la chaîne de caractère principale, en supprimant ce préfixe. Pour réaliser cette opération, on utilisera l'une des syntaxes suivantes :

```
${variable#préfixe}
```

ou :

```
${variable##préfixe}
```

où **variable** est la variable contenant la chaîne de caractères à traiter, et **préfixe** est le préfixe à supprimer.

En fait, le préfixe peut être spécifié à l'aide d'un motif de caractères. Ce motif peut correspondre à une partie plus ou moins grande de la valeur de la variable. Dans ce cas, il y a plusieurs manières d'interpréter ce motif, et donc plusieurs choix de préfixes possibles à supprimer. La première syntaxe devra être utilisée lorsque l'on désire supprimer le plus petit préfixe possible correspondant au motif. La deuxième syntaxe, quant à elle, permettra de supprimer le préfixe le plus long. Par exemple, si la variable **VAR** contient la chaîne de caractères **abbbc**, la commande suivante :

```
echo ${VAR#a*b}
```

affichera la chaîne de caractères **bbc**, car le plus petit préfixe correspondant au motif **a*b** est **ab**. Inversement, la commande :

```
echo ${VAR##a*b}
```

utilisera le préfixe le plus long, à savoir **abbb**. Le résultat de cette substitution sera donc la chaîne de caractères C.

Le shell fournit une syntaxe similaire pour extraire des suffixes de la valeur des variables. Cette syntaxe utilise simplement le caractère % au lieu du caractère #. Comme pour les préfixes, le fait de doubler ce caractère implique que le suffixe le plus long correspondant au motif sera utilisé, alors que l'utilisation d'un seul % permet de choisir le suffixe le plus court. Ainsi, la commande :

```
echo ${VAR%b*c}
```

affichera la chaîne de caractères **abb**, alors que la commande :

```
echo ${VAR%%b*c}
```

n'affichera que a.

Pour terminer ce tour d'horizon des remplacements de variables, nous allons voir les possibilités de recherche et de remplacement du shell dans les chaînes de caractères contenues dans des variables. La syntaxe suivante :

```
${variable/motif/remplacement}
```

permet de rechercher la plus grande sous-chaîne de caractères correspondant au motif motif dans la chaîne contenue dans la variable variable, et de remplacer cette sous-chaîne par la chaîne de caractères remplacement. Par exemple, si la variable VAR contient la chaîne de caractères abab, la commande suivante :

```
echo ${VAR/b/d}
```

affichera la chaîne de caractères adbc.

Ce remplacement n'est donc effectué qu'une seule fois. Si l'on veut que toutes les occurences du motif soit remplacées par la chaîne de remplacement, il suffit de doubler les caractères / :

```
${variable//motif//remplacement}
```

Dans les deux syntaxes, la présence du champ remplacement est facultative. Ceci permet de supprimer purement et simplement les sous-chaînes de caractères qui correspondent au motif.

Cependant, une précision s'impose : si le motif commence par le caractère #, ce motif devra obligatoirement être placé au début de la chaîne de caractères contenue dans la variable. De même, si le motif commence par le caractère %, il devra obligatoirement se trouver à la fin de cette chaîne. Ces deux notations permettent d'obtenir le même effet que les suppressions de préfixes et de suffixes présentées plus haut.

Contrôle des processus

Un des avantages des lignes de commandes par rapport aux environnements graphiques est la facilité avec laquelle elles permettent de contrôler les processus. Ce paragraphe décrit les principales méthodes pour lancer et arrêter un processus, ainsi que pour lui fournir les données sur lesquelles il doit travailler et récupérer ses résultats.

Lancement d'un programme en arrière plan

Le lancement normal d'un programme se fait en tapant sa ligne de commande et en appuyant sur la touche de validation. Le shell ne rendra pas la main et ne permettra pas de lancer un autre programme tant que le processus en cours ne sera pas terminé. Cependant, vous pouvez fort bien désirez lancer en arrière plan une commande dont la durée d'exécution peut être très longue et continuer à travailler. Après tout, Linux est multitâche… Eh bien, rien de plus facile !
Pour lancer une commande en arrière plan, il faut :
s'assurer que la commande aura toutes les informations nécessaires pour travailler sans intervention de l'utilisateur (ou, autrement dit, que la commande ne soit pas interactive) ;
ajouter une espèrluette (caractère « & ») à la fin de la ligne de commande commande.
Par exemple, la commande suivante :

```
cp /cdrom/kernel/linux-2.2.10.tar.gz . &
```

copiera l'archive du noyau 2.2.10 du CD-ROM vers le répertoire courant, et s'exécutera en arrière plan.

Lorsqu'une commande est lancée en arrière plan, le shell affiche deux nombres qui permettront de l'identifier par la suite. Le premier nombre, indiqué entre crochets, est le numéro de « job » du shell. Ce numéro sert à identifier les commandes du shell de manière unique. Un job est donc en réalité une commande du shell, simple ou complexe. Le deuxième numéro est le numéro de processus (« PID », pour « Process IDentifier ») dans le système du processus maître du job. Le PID est un numéro unique dans le système, qui permet d'identifier de manière unique les processus en cours. Ces deux nombres permettront de manipuler les processus, avec les commandes que l'on verra plus tard.

Il ne faut pas confondre les numéros de job avec les numéros de processus. Premièrement, un numéro de job n'est unique que dans un shell donné, et n'a aucune signification au niveau du système complet, alors que le numéro de processus est attribué par le système à chaque programme en cours d'exécution. Ensuite, une même commande du shell peut lancer plusieurs processus conjointement. Dans ce cas, il y a bien évidemment plusieurs numéros de processus, mais un seul et unique job. Ce genre de situation se produit par exemple lors de l'utilisation d'une redirection du flux de sortie standard d'un processus vers le flux d'entrée standard d'un autre processus.

Le numéro de processus affiché par le shell lors du lancement d'une ligne de commande complexe représente le PID du processus maître de la commande, c'est à dire, en pratique, le dernier processus d'une série de redirections ou le processus du shell exécutant les commandes complexes. Ceci signifie que dans tous les cas de configuration, ce PID est celui du processus qui contrôle l'ensemble des opérations effectuées par la ligne de commande. C'est donc par ce processus que l'on peut manipuler la commande complète, par exemple pour l'interrompre.

Il est possible de retrouver le PID du processus maître d'une commande à partir du numéro de job correspondant du shell. Ceci se fait simplement, en utilisant l'expressions suivante :

```
%job
```

où job est le numéro du job dont on cherche le PID. Ainsi, dans toutes les commandes décrites ci-dessous, le PID peut être utilisé directement, ou être remplacé par le numéro du job préfixé du caractère de pourcentage.

Listing des processus

Il n'est pas nécessaire de retenir tous les numéros de jobs et tous les PID des processus en cours d'exécution. Il existe en effet des commandes permettant d'obtenir la liste des processus et des jobs. La plus simple à utiliser est bien évidemment la commande du shell pour obtenir la liste des jobs avec leurs lignes de commandes. Pour obtenir cette liste, il suffit de taper la commande suivante :

```
jobs
```

qui affiche, dans l'ordre, le numéro de job, l'état du processus correspondant, et la ligne de commande.

Une autre commande, plus bas niveau, permet d'obtenir des informations plus complète directement à partir du système. Il s'agit de la commande **ps**, dont la syntaxe est donnée ci-dessous :

```
ps [options]
```

Les options les plus utiles sont sans doute x, qui permet de demander l'affichage de toutes les commandes en cours d'exécution et non pas seulement les processus en cours d'exécution dans le shell où la commande **ps** est exécutée, et a, qui permet d'obtenir l'affichage de toutes les commandes, pour tous les utilisateurs connectés. Ces deux options peuvent être cumulées, et la commande suivante :

```
ps ax
```

affiche donc toutes les commandes en cours d'exécution sur le système.

Les informations les plus intéressantes affichées par **ps** sont le PID du processus, qui est donné par le premier nombre affiché, et la ligne de commande, qui est la dernière information affichée. Pour plus de détails sur la commande **ps**, veuillez consulter la page de manuel correspondante.

Gel d'un processus

Il est possible de « geler » un processus en cours d'exécution, c'est à dire de le suspendre, sans pour autant l'arrêter définitivement. Ceci peut être utilisé pour libérer un peu les capacités de calcul, lorsque ce processus consomme trop de ressources par exemple. Pour cela, deux méthodes sont possibles :

- soit on utilise la combinaison de touches CTRL+Z, lorsque le processus est en avant plan ;

- soit on envoie le signal 19 au processus (signal « STOP ») à l'aide de la commande **kill**.

La première méthode est recommandée pour les processus lancés par une ligne de commande complexe, car le signal STOP est envoyé au processus maître de la commande, et est propagé à l'ensemble des processus fils de ce processus. Ceci signifie que tous les processus invoqués dans le cadre de cette commande sont également gelés. La deuxième méthode est plus bas niveau, et permet de geler n'importe quel processus que l'on a lancé.

Relancement d'un processus

Un processus suspendu peut être relancé soit en avant plan, soit en arrière plan. Pour relancer un processus en avant plan, il faut utiliser la commande suivante :

```
fg PID
```

où PID est le PID du processus à relancer en avant plan. **fg** est l'abréviation de l'anglais « foreground », ce qui signifie « avant plan ». Il faut attendre que ce processus se termine pour entrer de nouvelles commandes. Par conséquent, on ne peut lancer en avant plan qu'un seul processus.

De même, pour lancer un processus en arrière plan, il faut utiliser la commande **bg**, qui est l'abréviation de l'anglais « background ». Cette commande s'utilise de la même manière que la commande **fg** :

```
bg PID
```

Le relancement d'un processus suspendu peut également se faire en lui envoyant le signal 18 à l'aide de la commande **kill**.

Redirections

Pour pouvoir lancer un programme en arrière plan, il est nécessaire qu'il n'ait pas besoin de demander des données à l'utilisateur. En effet,

lorsqu'il est en arrière plan, la saisie de ces données ne peut pas se faire, puisque le shell les interpréterait comme une nouvelle commande. De plus, tout affichage en provenance d'une commande en arrière plan apparaît sur la console tel quel, et risque de se mélanger avec l'affichage des autres programmes ou même avec la commande en cours d'édition. C'est pour résoudre ces problèmes que le mécanisme des redirections a été introduit.

Principe de base

Le mécanisme des *redirections* a pour but de transférer les données provenant d'un flux vers les données d'un autre flux. Il se base sur la notion de descripteur de fichier, utilisée par la plupart des systèmes Unix. Un *descripteur de fichier* est un numéro utilisé par les programmes pour identifier les fichiers ouverts. Les descripteurs 0, 1 et 2 sont respectivement affectés d'office au flux d'entrée standard (nommé « stdin »), au flux de sortie standard (« stdout ») et au flux d'erreur standard (« stderr »), qui en général apparaît également sur l'écran.

Les descripteurs de fichiers d'un processus sont généralement hérités par tous ses processus fils. Ceci signifie que lors de leur lancement, ces processus peuvent utiliser tous les descripteurs de fichiers mis à leur disposition par leur père. Dans le cas des lignes de commande, les processus fils sont les processus lancés par le shell, et les descripteurs de fichiers hérités sont donc les descripteurs de fichiers du shell. C'est de cette manière que le shell peut manipuler les descripteurs de fichiers des processus qu'il lance : il effectue d'abord les redirections sur ses propres descripteurs de fichiers, puis il lance le processus fils avec ces redirections actives. Le mécanisme est donc complètement transparent pour les processus fils.

Le mécanisme des redirections permet en fait d'injecter dans un descripteur de fichier des données provenant d'un autre descripteur ou d'un fichier identifié par son nom, et d'envoyer les données provenant d'un descripteur de fichier dans un autre descripteur ou dans un fichier identifié par son nom. Si l'on utilise les descripteurs de fichiers des flux d'entrée / sortie standard, on peut exécuter n'importe quelle commande interactive en arrière plan.

Redirections de données en entrée

Pour injecter des données provenant d'un fichier dans le descripteur de fichier n d'un processus, il suffit d'ajouter la ligne suivante à la fin de la commande permettant de lancer ce processus :

n<fichier

où fichier est le nom du fichier dont les données doivent être injectées dans le descripteur n. Dans cette syntaxe, le descripteur peut ne pas être précisé. Dans ce cas, le shell utilisera le descripteur 0, et les données du fichier seront donc envoyées dans le flux d'entrée standard du processus. Par exemple, supposons que l'on désire utiliser une commande nommée « search », et que cette commande demande un certain nombre d'informations lors de son exécution. Si l'on sait à l'avance les réponses aux questions qui vont être posées, on peut créer un fichier de réponse (nommé par exemple « answer.txt » et alimenter la commande « search » avec ce fichier. Pour cela, on utilisera la ligne de commande suivante :

```
search < answer.txt
```

Il est également possible d'injecter des données provenant d'un autre descripteur de fichiers dans un descripteur de fichiers. On utilisera pour cela la syntaxe suivante :

```
n<&s
```

où n est toujours le descripteur de fichier du processus à exécuter dans lequel les données doivent être injectées, et s est un descripteur de fichiers contenant les données sources à injecter. Par défaut, si n n'est pas précisé, le flux d'entrée standard du processus sera utilisé.

Redirection de données en sortie

Inversement, il est possible d'enregistrer les données écrites par un processus dans un de ses descripteurs de fichier dans un fichier. Pour cela, on utilisera l'opérateur '>' avec la syntaxe suivante :

```
n>fichier
```

où n est le numéro du descripteur de fichier du processus à enregistrer, et fichier est le nom du fichier dans lequel les données doivent être stockées. Par défaut, si n n'est pas spécifié, le descripteur du flux de sortie standard sera utilisé (descripteur 1). Par exemple, si la commande précédente affiche des résultats et que l'on désire les stocker dans le fichier « result.txt », on utilisera la ligne de commande suivante :

```
search < answer.txt >result.txt
```

Notez que cette commande détruira systématiquement le contenu du fichier « result.txt » et le remplacera par les informations provenant du flux de sortie standard du processus « search ». Il est possible de ne pas vider le fichier « result.txt », et d'ajouter les informations en fin de fichier, en utilisant l'opérateur '>>' à la place de l'opérateur '>'. Ainsi, la commande suivante :

```
search < answer.txt >>result.txt
```

aura pour effet d'ajouter à la fin du fichier « result.txt » les informations affichées par le processus « search ».

Le flux d'erreur standard, qui correspond normalement à l'écran et qui permet d'afficher les messages d'erreur, peut être redirigé avec l'opérateur '2>', de la même manière que l'opérateur '>' est utilisé pour le flux de sortie standard (puisque c'est le descripteur de fichier utilisé par défaut par l'opérateur '>'). Par exemple, si l'on veut envoyer les messages d'erreurs éventuels de la commande précédente vers le périphérique nul (c'est à dire le périphérique qui n'en fait rien) pour ignorer ces messages, on utilisera la ligne de commande suivante :

```
search <answer.txt >result.txt 2> /dev/null
```

Une telle ligne de commande est complètement autonome, et peut être lancée en arrière plan, sans aucune intervention de l'utilisateur :

```
search <answer.txt >result.txt 2> /dev/null &
```

Il est également possible d'effectuer une redirection des données provenant d'un descripteur de fichier du processus vers un autre descripteur de fichier de ce processus. On utilisera pour cela la syntaxe suivante :

```
n>&d
```

où n est le descripteur de fichier dont les données doivent être redirigées, et d le descripteur de fichier destination. Cette syntaxe est souvent utilisée pour rediriger le flux d'erreur standard vers le flux d'entrée standard, lorsque l'on veut récupérer les erreurs et les messages d'exécution normale dans un même fichier. Par exemple, si l'on veut rediriger le flux de sortie et le flux d'erreurs de la commande « search » dans un même fichier, on utilisera la ligne de commande suivante :

```
search <answer.txt >result.txt 2>&1
```

Cette ligne de commande utilise deux redirections successives pour les données affichées par la commande « search » : la première redirige le flux de sortie standard vers un fichier, et la deuxième le flux d'erreur standard vers le flux de sortie standard. Notez que l'ordre des redirections est important. Elles sont appliquées de gauche à droite. Ainsi, dans la commande précédente, le flux de sortie standard est redirigé vers le fichier « result.txt », puis le flux d'erreur standard est injecté dans le flux de sortie standard ainsi redirigé.

> **Note:** Il est également possible d'utiliser un autre descripteur de fichier que les descripteurs des flux standards. Cependant, il est nécessaire, dans ce cas, d'ouvrir ce descripteur dans le shell avant de lancer la commande. Ceci peut se faire à l'aide de la syntaxe suivante :
>
> ```
> n<>fichier
> ```
>
> où n est un numéro de descripteur de fichier non encore utilisé, et fichier est un nom de fichier. Ce nouveau descripteur de fichier pourra être utiisé dans les commandes précédentes, afin de faire manipuler le fichier fichier par les processus fils de manière transparente.

Les descripteurs de fichiers ouverts de cette manière le restent d'une commande sur l'autre dans le shell. Ceci implique que toutes les données écrites dans ces descripteurs de fichiers sont ajoutées automatiquement à la fin des fichiers manipulés par ces descripteurs. Ce comportement est différent de la redirection vers un fichier effectuée par l'opérateur '>', qui ouvre à chaque fois le fichier en écriture à son début et qui supprime donc toutes les données déjà esistantes. Il n'y a donc pas d'opérateur '>>&' pour ajouter des données à un descripteur de fichiers, car cela n'a pas de sens.

Les descripteurs de fichiers peuvent également être manipulés directement, par l'intermédiaire de fichiers virtuels du répertoire / dev/fd/. À chaque descripteur de fichier (y compris les descripteurs pour les flux d'entrée / sortie standard !) y correspond un fichier

dont le nom est le numéro du descripteur. Par exemple, le fichier /dev/fd/2 correspond au flux d'erreur standard.

En fait, le répertoire /dev/fd/ est un lien symbolique vers le répertoire /proc/self/fd/ du système de fichiers virtuels /proc/. Ce système de fichiers est géré par le noyau directement, et permet d'accéder aux informations sur le système et les processus. Il contient en particulier un sous-répertoire pourtant le nom du PID de chaque processus existant dans le système, et chacun de ces répertoires contient lui-même un sous-répertoire fd/ où sont représentés les descripteurs de fichiers ouvert par le processus correspondant. Le système de fichiers /proc/ contient également un lien symbolique self/ pointant sur le sous-répertoire du processus qui cherche à l'ouvrir. Ainsi, /proc/self/fd/ est un chemin permettant à chaque processus d'accéder à ses propres descripteurs de fichiers.

En pratique, la manipulation directe des descripteurs de fichiers n'est réellement intéressante que pour les flux standard, dont les numéros de descripteurs sont fixes et connus de tous les programmes. Pour les autres descripteurs, cette technique est souvent inutilisable ou inutile, sauf lorsqu'on utilise des programmes sachant manipuler des descripteurs de numéros bien déterminés.

Insertion de documents

Il existe un dernier opérateur de redirection, qui n'est utilisé en pratique que dans les scripts shell. Cet opérateur permet d'insérer directement un texte complet dans le flux d'entrée standard, sans avoir à placer ce document dans un fichier à part. Cette technique permet donc de stocker des données avec le code des scripts shell, et de n'avoir ainsi qu'un seul fichier contenant à la fois le script et ses données.

Cet opérateur est l'opérateur '<<', il s'utilise selon la syntaxe suivante :

<<EOF texte ⋮ EOF

où texte est le contenu du texte à insérer, et EOF est un marqueur quelconque qui sera utilisé seul sur une ligne afin de signaler la fin du texte.

Par exemple, il est possible de créer un fichier test.txt de la manière suivante :

cat <<fin >test.txt

Ceci est un fichier texte saisi directement dans le shell. On peut écrire tout ce que l'on veut, et utiliser les fonctions d'éditions de ligne du shell si l'on veut. Pour terminer le fichier, il faut taper le mot "fin" tout seul, au début d'une ligne vide.

fin

Les pipes

Les redirections sont très pratiques lorsqu'il s'agit d'injecter un fichier dans le flux d'entrée standard d'un processus, ou inversement de rediriger le flux standard d'une commande vers un fichier, mais elles ont justement le défaut de devoir utiliser des fichiers. Il est des situations où l'on désirerait injecter le résultat d'une commande dans le flux d'entrée standard d'une autre commande, sans passer par un fichier intermédiaire. Ceci est heureusement réalisable, grâce à ce que l'on appelle les « pipes ».

Syntaxe des pipes

Pour rediriger le résultat d'une commande dans le flux d'entrée d'une autre commande, il faut utiliser l'opérateur '|'. Cet opérateur représente un tuyau canalisant les données issues d'une commande vers le flux d'entrée standard de la commande suivante, d'où le nom de « pipe » (« tuyau » en anglais). L'opérateur pipe s'utilise de la manière suivante :

- on écrit la première commande, qui doit fournir les données à la deuxième commande ;
- on écrit l'opérateur pipe ;
- on écrit la deuxième commande, qui doit lire les données provenant de la première.

La commande se trouvant à la gauche de l'opérateur pipe doit être complète, avec ses autres redirections éventuelles. La redirection dans un pipe s'effectue après les autres types de redirections vues précédemment.

Le système contrôle l'exécution des processus qui se trouvent au deux bouts d'un pipe, de telle sorte que le transfert de données puisse toujours se faire. Si le processus source a trop de données, il est figé par le système d'exploitation en attendant que le processus consommateur ait fini de traiter les données déjà présente. Inversement, si le processus source est trop lent, c'est le processus consommateur qui attendra patiemment que les données soient disponibles.

Les pipes sont utilisés très couramment, ne serait-ce que pour afficher page par page le contenu d'un répertoire. La commande suivante effectue un tel travail :

```
ls | less
```

Ici, le résultat de la commande **ls** est redirigé vers la commande **less**, qui permet d'afficher page par page (et de revenir en arrière dans ces pages) la liste des fichiers du répertoire courant.

Prenons un exemple un peu plus complexe. Supposons que l'on veuille archiver et compresser un répertoire. Il est possible d'archiver ce répertoire avec la commande **tar**, puis de comprimer le fichier archive résultant :

```
tar cvf archive.tar * gzip archive.tar
```

Cette méthode est correcte, mais souffre d'un défaut : elle utilise un fichier intermédiaire, qui peut prendre beaucoup de place disque. Une méthode plus économe consiste à lancer **tar** et **gzip** en parallèle, et de rediriger la sortie standard de l'un dans le flux d'entrée de l'autre. Ainsi, il n'y a plus de fichier temporaire, et la place consommée sur le disque est minimale :

```
tar cv * | gzip > archive.tar.gz
```

La première commande demande à **tar** d'archiver tous les fichiers du répertoire et d'envoyer le résultat dans le flux standard de sortie. Le pipe redirige ce flux standard vers le flux d'entrée standard de **gzip**. Celui-ci comprime les données et les émet vers son flux standard de sortie, qui est lui-même redirigé vers le fichier archive.tar.gz. Aucun fichier temporaire n'a été utilisé, et on a ainsi économisé l'espace disque de l'archive complète non compressée, c'est à dire environ la taille complète du répertoire à archiver. Ce genre de considération peut être très important lorsque le disque dur commence à être plein…

> **Note:** En fait, la commande tar de GNU permet de comprimer à la volée les données à archiver, permettant d'éviter de se prendre la tête comme on vient de le faire. Pour cela, il suffit d'utiliser l'option z dans la ligne de commande de **tar**. Ainsi, la ligne de commande suivante fournit le même résultat :
>
> ```
> tar cvfz archive.tar.gz *
> ```
>
> Mais cette solution ne fonctionne pas avec les versions non GNU de tar, qui ne supportent pas cette option.

Un autre exemple pratique est le déplacement de toute une arborescence de fichiers d'un système de fichiers à un autre. Vous ne pourrez pas y parvenir à l'aide de la commande **mv**, car celle-ci ne fait que modifier la structure du système de fichiers pour déplacer les fichiers et les répertoires, elle ne peut donc pas fonctionner avec deux systèmes de fichiers. Vous ne pouvez pas non plus utiliser la commande **cp**, car celle-ci ne prendra pas en compte les dates des fichiers, leurs propriétaires et leur groupes, ainsi que les liens symboliques et physiques. Il faut donc impérativement utiliser un programme d'archivage. La méthode à suivre est donc de créer une archive temporaire, puis de se déplacer dans le répertoire destination, et enfin d'extraire l'arborescence de l'archive :

```
cd source
tar cvf archive.tar *
cd destination
tar xvf source/archive.tar
rm source/archive.tar
```

Malheureusement, cette technique nécessite beaucoup de place disque, puisque l'archive temporaire est stockée directement sur disque. De plus, elle est assez lente, car toutes les données à copier sont recopiées sur le disque dur, et relues ensuite, pour finalement être détruites… La vraie solution est de réaliser un pipe entre les deux processus **tar** invoqués. Dans ce cas, le transfert se fait simplement via la mémoire vive :

```
cd source tar cv * | (cd destination ; tar xvf -)
```

La commande à utiliser est cette fois un peu plus compliquée, car la commande d'extraction des fichiers nécessite un changement de répertoire. Il faut donc utiliser une commande multiple du shell. Ces commandes sont constituées de plusieurs autres commandes séparées par des points virgules. La première commande effectuée ici est le changement de répertoire, et la deuxième est l'extraction par tar de l'archive qui lui est transférée par le flux d'entrée standard (représenté ici par '-'). Ces deux commandes sont mises entre parenthèses, car l'opérateur '|' du pipe est prioritaire sur l'opérateur ';' de concaténation des commandes du shell. Si vous trouvez que ceci est un peu compliqué, je vous l'accorde. Cependant, la commande qui utilise le pipe consomme deux fois moins d'espace disque et est deux fois plus rapide que la commande qui n'en utilise pas. Je vous invite à mesurer le gain de temps sur un répertoire contenant un grand nombre de données (utilisez la commande **time** !).

Substitution du résultat d'une commande

Le shell peut évaluer une commande apparaissant dans une expression afin de la remplacer par son résultat. Il existe deux syntaxes pour réaliser ce type de substitutions. La première, et la plus classique (voire historique), utilise des guillemets inverses :

```
`commande`
```

où commande est la commande devant être remplacée par son résultat (c'est à dire ce qu'elle enverra sur le flux standard de sortie). Pour donner un exemple, la commande suivante :
`kill `cat /var/pid/p.pid``
a pour résultat de lancer un signal SIGTERM au processus dont le PID est stocké dans le fichier /var/pid/p.pid. La commande **cat** est utilisée pour afficher le contenu de ce fichier, et elle est substituée par ce contenu. En fin de compte, la commande **kill** est appliqué au PID affiché par **cat**.

La deuxième syntaxe utilisable est la suivante :

```
$(commande)
```

où commande est toujours la commande à exécuter et à substituer. La différence entre ces deux syntaxes est que dans le premier cas, les caractères $, ` et \ sont toujours interprétés par le shell et doivent être précédés d'un antislash s'ils doivent apparaître tels quels dans la commande à substituer, alors que dans le deuxième cas, on peut utiliser tous les caractères sans protection particulière (sauf, bien entendu, la parenthèse fermante, puisqu'elle marque la fin de la commande).

Substitution de commandes

Nous avons déjà vu qu'il était possible de récupérer le résultat d'une commande pour l'insérer dans une ligne de commande. Cette technique

s'apparente à ce qu'il est possible de faire avec la commande **xargs** et un pipe. De la même manière, le shell fournit une substitution permettant d'obtenir des fonctionnalités similaires à celles fournies par les pipes nommés. Cette substitution est la substitution de commande.

La syntaxe utilisée par les substitutions de commandes est la suivante est similaire à celle des redirections classiques :

```
<(command)
ou :
>(command)
```

où command est la commande à substituer.

La première syntaxe permet de lancer une commande en arrière-plan en redirigeant son flux standard de sortie vers un descripteurs de fichiers du shell. Le résultat de cette substitution est le nom du fichier /dev/fd/n permettant de lire les données écrite par la commande dans ce descripteur de fichier. En pratique, on utilise donc cette substitution en lieu et place d'un fichier d'entrée pour une commande normale. La deuxième commande permet de lancer également une commande en arrière-plan, mais en redirigeant le flux d'entrée standard de cette commande cette fois. Il est alors possible de fournir les données nécessaires à cette commande en écrivant dans le fichier /dev/fd/n dont le nom est fourni par le résultat de la substitution.

Ces deux commandes permettent donc de simplifier l'usage des pipes nommés, en évitant d'avoir à créer un fichier de pipe manuellement et de lancer les deux commandes devant se servir de ce pipe pour communiquer. Ainsi, la commande suivante :

```
cat <(ls)
```

est fonctionnellement équivalente à la série de commande suivante :

```
mkfifo /tmp/lsfifo
ls > /tmp/lsfifo
cat /tmp/lsfifo
 rm /tmp/lsfifo
```

Les substitutions de commandes sont donc nettement plus pratiques et plus sûres, car elles n'imposent pas la création d'un fichier de pipe nommé dont le nom peut être choisi arbitrairement.

Remplacement des caractères génériques

Si, après avoir appliqué toutes les formes de substitutions précédentes, le shell trouve des caractères génériques * et ? dans l'expression en cours de traitement, il interprétera la partie de l'expression contenant ces caractères comme un motif représentant des chemins Unix de fichiers. Ce motif sera donc remplacé par autant de chemins Unix lui correspondant que possible. Rappelons que le caractère générique * représente 0 ou plusieurs caractères quelconques, et que le caractère générique ? représente un caractère et un seul. Les chemins générés sont classés par ordre alphabétique.

Il est possible également de restreindre le jeu de caractère utilisé par le shell pour rechercher les noms de fichiers correspondants au motif. Pour cela, il faut lui indiquer un ensemble de caractères ou de plages de caractères utilisables, séparés par des virgules, et entre crochets. Les plages de caractères sont spécifiées en indiquant le premier et le dernier caractère, séparés par un tiret. Par exemple, la commande suivante :

```
ls [a-c,m-t]*.txt
```

permet d'afficher tous les fichiers dont le nom commence par les lettres a, b, c et les lettres allant de m à t, et dont l'extension est .txt.

Sauf paramétrage pour indiquer explicitement de faire le contraire, le shell ignore systématiquement les répertoires . et .. dans les substitutions. Ceci est très important. En effet, une commande utilisant le caractère générique * ne s'appliquera pas sur le répertoire courant et le répertoire père par défaut. Paramétrer bash pour qu'il prennent en compte ces répertoires peut être extrêmement dangereux, surtout avec une commande telle que rm -f *, qui dans ce cas effacerait également le répertoire parent en plus du contenu du répertoire courant !

Les expressions régulières

Les substitutions de variables et de noms de fichiers utilisent des motifs pour identifier des chaînes de caractères. Notez que pour spécifier le caractère - lui-même, il suffit de le placer tout seul, à la fin de la liste. De même, pour spécifier le caractère] (normalement utilisé pour marquer la fin du jeu de caractères), il faut le placer au début de la liste, juste après le crochet ouvrant.

Structures de contrôle

Tout langage de programmation digne de ce nom dispose de structures de contrôles évoluées permettant de contrôler l'exécution du programme, de réaliser des boucles et de structurer l'ensemble d'un programme. Le shell n'échappe pas à la règle, et fournit la plupart des constructions classiques. Cette section a pour but d'exposer leurs syntaxes.

Les instructions composées

Dans le langage du shell, une instruction se termine soit par un retour à la ligne (non précédé d'un antislash), soit d'un point-virgule. Les instructions peuvent être pourtant très complexes, car elles peuvent contenir des pipes et des redirections. En fait, une instruction peut à peu près être définie comme étant une ligne de commande normale du shell.

Le shell permet bien entendu de réaliser des instructions composées, afin de regrouper plusieurs traitements dans un même bloc d'instructions. La méthode la plus simple pour réaliser un bloc d'instruction est tout simplement de les regrouper sur plusieurs lignes, ou de les séparer par des points-virgules, entre accolades. Par exemple, les instructions suivantes constituent un bloc d'instruction :

```
{     cd /tmp     rm *.bak }
```

Notez que l'accolade fermante est considérée comme une instruction à part entière. Ceci signifie que si l'on ne met pas l'accolade fermante sur une ligne indépendante, il faut faire précéder l'instruction précédente d'un point-virgule. De même, il faut le faire suivre d'un autre point-virgule s'il ne se trouve pas à la fin d'une ligne.

Les instructions d'instruction composée créée à l'aide des accolades sont exécutée au sein du shell courant. Les variables qu'elles définissent, ainsi que les changement de répertoires, sont donc toujours valides à l'issue de l'exécution de ces instructions. Si cela n'est pas désirable, on pourra créer des instructions composées à l'aide de parenthèses. Les instructions seront alors exécutées dans un autre shell, lancé pour l'occasion, et elles n'auront donc pas d'effets de bords imprévus dans le shell appelant. Par exemple, le répertoire courant à l'issue de l'instruction composée précédente est le répertoire /tmp/, alors que l'instruction composée suivante :

```
(     cd /tmp     rm *.bak )
```

ne change pas le répertoire courant.

> **Note:** On ne confondra pas les instructions composées utilisant des parenthèses et les substitutions de résultat de commande. Les instructions composées renvoient le code d'erreur de la dernière instruction exécutée, alors que le résultat des substitutions est ce que la commande a écrit sur son flux de sortie standard.

Le shell permet également de réaliser des instructions composées conditionnelles, où l'exécution de chaque instruction de l'instruction composée est conditionnée par le résultat de l'instruction précédente. Ces instructions composées sont définies à l'aide des opérateurs || et &&. La syntaxe de ces opérateurs est la même :

```
command1 || command2 command1 && command2
```

où command1 et command2 sont deux commandes du shell (composées ou non). Avec l'opérateur ||, la commande command2 n'est exécutée que si le code d'erreur de la commande command1 est non nul, ou, autrement dit, si cette commande ne s'est pas exécutée correctement. Inversement, avec l'opérateur &&, la commande command2 n'est exécutée que si la première commande s'est exécutée correctement (et renvoie donc un code d'erreur nul). Par exemple, la commande suivante :

```
rm *.txt 2> /dev/null || echo "Aucun fichier à supprimer"
```

permet d'effacer tous les fichiers d'extension .txt, ou d'afficher le message d'erreur Aucun fichier à supprimer s'il n'existe pas de tels fichiers.

Les instructions composées peuvent être utilisées comme n'importe quelle commande normale. En particulier, elles peuvent être utilisées dans des commandes plus complexes, par exemple comme destination d'un pipe.

Les tests

Sous Unix, chaque processus reçoit plusieurs valeurs en paramètres et renvoie un code de retour. La plupart des paramètres sont passés en ligne de commande, et sont récupérés directement par le processus, mais d'autres paramètres peuvent être fournis par le processus appelant par l'intermédiaire de variables d'environnement et de descripteurs de fichiers. Le code de retour, quant à lui, est un entier signalant si l'exécution du processus s'est terminée correctement ou si des erreurs ont eu lieu. Si les codes d'erreurs varient grandement d'un programme à un autre, la valeur 0 signifie toujours, et ce quel que soit le programme, que l'exécution s'est déroulée correctement.

Il est possible de tester le code de retour d'une commande avec l'instruction if. La syntaxe la plus simple pour un test est la suivante :

```
if commande ; then
    action
 fi
```

où commande est la commande dont on désire tester le code de retour, et action est la commande à exécuter si ce code vaut 0 (c'est à dire, si la commande commande s'est exécutée correctement).

Il peut paraître réducteur de ne pouvoir tester que le code de retour d'une commande. Mais en fait, c'est une fonctionnalité très puissante du shell, car elle permet de réaliser tous les types de tests imaginables. En effet, il existe une commande spéciale, **[**, qui permet de réaliser divers types de tests sur les paramètres que l'on lui passer, et d'ajuster son code d'erreur en conséquence. Par exemple, pour tester l'égalité d'une variable d'environnement avec une chaîne de caractère, on utilisera la syntaxe suivante :

```
if [ $variable == valeur ] ; then
     action
fi
```

Notez que dans cette syntaxe, le test effectué est une commande complète. Ceci implique qu'il faut mettre un espace entre chaque paramètre, et en particulier entre le nom de la commande ([), le premier opérande ($variable), l'opérateur utilisé (==), le deuxième opérande (valeur) et le caractère de marque de fin de test (]).

La commande **[** est capable d'effectuer tous les tests standards. Par défaut, elle considère que les deux opérandes du test sont des chaînes de caractères, et elle utilise l'ordre lexicographique pour les comparer. Les tests d'égalité et d'inégalités sont effectués respectivement avec les opérateurs == et !=. Les opérateurs d'antériorité dans l'ordre lexicographique sont < et <=, et les opérateurs de postériorité sont > et >=. Notez que l'utilisation de ces opérateurs peut être relativement pénible, parce que les caractères < et > sont interprétés par le shell en tant que redirections. Par conséquent, il faut souvent les précéder du caractère d'échappement antislash.

L'ordre lexicographique convient dans la plupart des cas, mais il n'est pas très approprié pour la comparaison de chaînes de caractères. Par exemple, le test suivant :

```
if [ -1 \< -2 ] ; then
   echo "-1 est plus petit que -2"
fi
```

est vérifié, car le caractère 1 précède le caractère 2 dans l'ordre lexicographique. La commande **[** fournit donc la possibilité d'utiliser une autre syntaxe pour comparer les entiers. Cette syntaxe utilise les options lt et gt respectivement pour les tests d'infériorité stricte et de supériorité stricte, et les options le et ge respectivement pour les tests d'infériorité et de supériorité ou d'égalité. Ainsi, le test :

```
if [ $i -gt 3 ] ; then
   echo "$i est supérieur à 3"
fi
```

permet de comparer la valeur entière de la variable i avec le nombre 3.

Les opérateurs || et && permettent de tester le code de retour d'une commande, et qu'en fonction de la valeur de ce code de retour, d'exécuter ou non la commande suivante. La syntaxe de ces opérateurs provient en fait de la possibilité de les employer pour effectuer des tests complexes avec l'instruction if. Par exemple, pour effectuer un ET logique entre deux tests, on utilisera la syntaxe suivante :

```
if [ $i == "A" ] && [ $j -lt 3 ] ; then
  echo "i contient la lettre \"A\" et j contient un nombre inférieur à 3"
fi
```

Notez que la deuxième commande **[** n'est exécutée que si le premier test est vérifié. L'utilisation de l'opérateur || se fait selon le même principe. Il est bien entendu possible de regrouper plusieurs commandes de test ensemble, à l'aide de parenthèses.

Comme dans la plupart des langages informatiques, l'instruction if peut prendre des formes plus complexes pour traiter les cas où le test n'est pas vérifié. Ainsi, pour exécuter une action spécifique pour le cas où le test serait faux, on peut utiliser la syntaxe suivante :

```
if commande ; then
   action1
else
   action2
fi
```

où commande est toujours la commande dont le code de retour sera testé, action1 est l'action qui doit être réalisée si cette commande a renvoyé le code de retour 0, et action2 la commande à exécuter dans le cas contraire. De même, si l'on veut enchaîner des tests, on utilisera le mot-clé *elif*. La syntaxe générale du test est donc la suivante :

```
if commande1 ; then
   action1
elif commande2 ; then
   action2
elif commande3 ; then
```

```
    …
else
   actionn
fi
```

> **Note:** Pour des raisons d'optimisations, le shell peut simuler le comportement du programme **[**, et éviter ainsi de le lancer à chaque fois qu'il a à faire un test. Cependant, le principe originel était bien celui décrit ci-dessus, qui, bien que n'étant plus tout à fait exact, permet de mieux comprendre la syntaxe du shell.

> Il est possible de récupérer la valeur du code de retour de la dernière commande exécutée grâce à la variable spéciale $?. Cependant, il est très rare d'avoir à manipuler cette valeur directement, car les structures de contrôle du shell telles que if permettent d'effectuer les actions qui s'imposent sans avoir à la connaître.

> Pour ceux qui savent programmer en C, sachez que le code de retour est la valeur renvoyée par la fonction C exit ou par l'instruction return de la fonction principale main. Les paramètres de la ligne de commande, quant à eux, sont récupérables par l'intermédiaire des paramètres de la fonction principale main.

Il ne faut pas oublier que la fonction première du shell est de permettre les manipulations de fichiers. Il n'est donc pas étonnant que la commande **[** permette également de réaliser tous les tests imaginables sur les fichiers. Ces tests vont de l'existence d'un fichier à sa nature et ses attributs, en passant par son propriétaire et son groupe. La syntaxe générale de ces tests est la suivante :

```
if [ option fichier ] ; then
    &vellip;
fi
```

où option est une option de la commande **[** décrivant la propriété testée, et fichier est le nom du fichier sur lequel le test doit porter.

Les principales options utilisables dans les tests sur les fichiers sont récapitulées dans le tableau ci-dessous :

Tests sur les fichiers

Option	Signification
-e	Test d'existence d'un fichier ou d'un répertoire.
-d	Test d'existence d'un répertoire.
-f	Test d'existence d'un fichier normal.
-s	Test d'existence d'un fichier et vérification que sa taille est non nulle.

Option	Signification
-L	Test d'existence d'un lien symbolique.
-b	Test d'existence d'un fichier spécial de périphérique de type block (disque dur, CD-ROM, lecteur de cassettes, etc…).
-c	Test d'existence d'un fichier spécial de périphérique de type caractère (port série, port parallèle, carte son, etc…).
-p	Test d'existence d'un pipe.
-r	Test d'existence du fichier et d'accessibilité en lecture de ce fichier.
-w	Test d'existence du fichier et d'accessibilité en écriture de ce fichier
-x	Test d'existence du fichier et de possibilité d'exécution de ce fichier.
-g	Test d'existence du fichier et de présence du bit setgid sur ce fichier.
-u	Test d'existence du fichier et de présence du bit setuid sur ce fichier
-k	Test d'existence du fichier et de présence du bit sticky sur ce fichier.
-O	Test d'existence du fichier et d'appartenance de ce fichier à l'utilisateur effectif courant.
-G	Test d'existence du fichier et d'appartenance de ce fichier au groupe effectif courant.
-N	Test d'existence du fichier et de modification de ce fichier depuis la dernière fois qu'il a été lu.

Note: Ce tableau n'est pas exhaustif, mais les options les plus importantes et les plus utilisées s'y trouvent.

La commande **[** accepte également les options -nt et -ot, qui permettent respectivement de tester si un fichier est plus récent ou plus vieux qu'un autre, en se basant sur les dates de dernière modification de ces fichiers. Ces deux opérateurs s'utilisent avec la syntaxe suivante :

```
if [ fichier1 option fichier2 ] ; then
     &vellip;
fi
```

où fichier1 et fichier2 sont les deux fichiers sur lesquels la comparaison doit porter, et option est l'une des options -nt ou -ot.

Le branchement conditionnel

Lorsque l'on veut effectuer différentes opérations selon la valeur d'une variable, l'instruction if peut devenir très lourde à utiliser. En effet, si le nombre de valeurs différentes est grand, elle peut conduire à écrire un grand nombre de tests. Le shell fournit donc une instruction de branchement conditionnel, qui permet de spécifier quelle action doit être prise pour chaque valeur de la variable.

Le branchement conditionnel s'utiliser de la manière suivante :

```
case valeur in
     ( motif1 | motif2 | ... ) commande1 ;;
     ( motifn | motifn+1 | ... ) commande2 ;;
      &vellip;
esac
```

où motif1, motif2, etc… motifn+1 sont des motifs spécifiant les valeurs possibles pour la valeur valeur, et commande1, commande2, etc… sont les commandes à exécuter pour les valeurs de ces motifs.

La commande exécutée est la première commande pour laquelle la variable correspond à l'un de ses motifs correspondants. Une fois exécutée, la recherche se termine, et l'exécution reprend à la suite du branchement conditionnel. Par exemple ce branchement conditionnel :

```
case $i in
     ( *.txt ) echo "$i est un fichier texte" ;;
     ( *.gz ) echo "$i est compressé avec gzip" ;;
     ( *.tar ) echo "$i est une archive" ;;
esac
```

affiche la nature du fichier dont le nom est stocké dans la variable i à partir de son extension.

Le code de retour du branchement conditionnel est 0 si la variable ne correspond à aucun des motifs, ou le code de retour de la commande exécutée sinon.

Les boucles

Il existe deux types de boucles : le while et le until. La syntaxe des boucles while est la suivante :

```
while commande ; do
     action
done
```

où commande est une commande dont le code de retour est utilisé comme critère de la fin de la boucle, et action est l'instruction (composée ou non) exécutée à chaque itération de la boucle. Comme on le voit, le shell utilise le même principe pour les boucles que pour les tests pour évaluer une condition. Tant que la commande commande> renvoie un code de retour égal à 0, l'instruction action est exécutée.

L'instruction while utilise la même syntaxe que l'instruction while :

```
until commande ; do
      action
done
```

à ceci près que l'instruction action est exécutée tant que la commande commande renvoie un code de retour non nul. L'instruction until utilise donc simplement le test inverse de celui de l'instruction while.

Bien entendu, il est possible d'utiliser la commande **[** pour effectuer des tests plus complexes que le simple test du code de retour d'une commande. Par exemple, la boucle suivante calcule la somme des dix premiers entiers :

```
result=0
i=0
while [ $i -le 10 ] ; do
     result=$(($result + $i))
     i=$(($i + 1))
done
echo $result
```

Les itérations

Les itérations sont des boucles qui s'exécutent pour chaque élément d'un ensemble donné. Le shell gère les itérations par l'intermédiaire de l'instruction for. La syntaxe de cette instruction est la suivante :

```
for variable [ in ensemble ] ; do
    action
done
```

où variable est un nom de la variable utilisée pour l'itération, ensemble est l'ensemble des valeurs que peut prendre cette variable, et action est la commande (simple ou composée) à exécuter pour chaque valeur de cette variable.

Le principe des itérations est très simple. Pour chaque valeur indiquée dans l'ensemble des valeurs, la commande est exécutée, avec la valeur en question accessible dans la variable utilisée pour l'itération. Par exemple, la commande suivante :

```
for i in *.txt ; do
    mv $i ${i/%.txt/.doc}
done
```

permet de renommer tous les fichiers portant l'extension .txt en fichier du même nom, mais avec l'extension .doc.

Il n'est pas nécessaire de préciser l'ensemble des valeurs que peut prendre la variable. Dans ce cas, l'ensemble utilisé sera celui de tous les paramètres du script ou de la fonctions. Nous verrons plus loin comment réaliser des fonctions et des scripts, ainsi que la manière de récupérer leurs paramètres.

Les alias

Il est incontestable que certaines commandes peuvent avoir une grande complexité, et il peut être fastidieux de les retaper complètement à chaque fois que l'on en a besoin. D'autre part, la saisie d'une longue ligne de commande multiplie les risques de fautes de frappe et d'avoir à corriger la commande. Ceci peut au mieux faire perdre son temps à l'utilisateur, et au pire l'énerver.

Le shell fournit donc un mécanisme simplifié pour donner un nom simplifié aux commandes complexes : le mécanisme des *alias*. Les alias représentent en

fait des chaînes de caractères complexes, et sont remplacés automatiquement par le shell lorsqu'il analyse les lignes de commandes. C'est un mécanisme plus souple que celui des variables d'environnement, et permet de définir des macro commandes plus facilement qu'avec les fonctions du shell.

Pour créer un alias, vous devrez utiliser la syntaxe suivante :

alias nom=chaîne

où nom est le nom de l'alias, et chaîne est la chaîne de caractère représentée par cet alias. Par exemple, pour faire un alias nommé beep permettant de faire un bip sonore, on pourra utiliser la commande suivante :

alias beep="echo $'\a'"

Cet alias pourra être simplement utilisé simplement en tapant beep en ligne de commande.

Vous pourrez visualiser la liste des alias existant simplement à l'aide de la commande **alias**, appelée sans paramètres. Je vous recommande de consulter cette liste, pour vous donner une idée des alias courant, qui se révèlent généralement très utiles.

La suppression des alias se fait à l'aide de la commande **unalias**. Sa syntaxe est la suivante :

unalias nom

où nom est le nom de l'alias à supprimer.

> **Note:** Les alias ne sont remplacés par la chaîne de caractères qu'ils représentent que lorsque le shell analyse la ligne de commande. Ceci signifie que les définitions d'alias ne sont valides qu'après validation de cette ligne. On évitera donc de définir des alias dans la déclaration d'une instruction composée, car cet alias ne sera pas disponible à l'intérieur de son propre bloc d'instruction.
>
> Par défaut, les alias ne sont disponibles que dans les shells interactifs. Ils ne peuvent donc pas être utilisés dans les scripts shell.

Les scripts shell

Pour l'instant, toutes les fonctionnalités de bash, aussi puissantes soient-elles, ne constituent que l'interface d'un interpréteur de commande puissant. Mais nous avons dit que le shell était véritablement un langage de programmation. Ceci signifie qu'il est possible d'écrire des programmes complexes en langage shell, simplement en stockant plusieurs commandes dans un fichier. On appelle ces fichiers des *script shell*.

L'écriture d'un script shell n'est pas plus compliquée que de taper les commandes du programme les unes à la suite des autres dans un shell interactif. La seule différence est que les scripts shell peuvent être rejoués plusieurs fois, recevoir des paramètres en ligne de commande et renvoyer un code de retour.

Tout script shell est en fait un fichier texte sur lequel on a mis les droits d'exécutions. Il contient les différentes commandes du shell qu'il doit exécuter. Sa première ligne est très importante, elle permet d'indiquer au shell exécutant quel est la nature du fichier. La syntaxe de cette ligne est la suivante :

```
#!shell
```

où shell est le chemin absolu sur le shell ou l'interpréteur de commande capable d'exécuter ce script. En pratique, pour bash, on utilisera toujours la ligne suivante :

```
#!/bin/bash
```

Les paramètres des scripts shells sont accessibles exactement comme des paramètres de fonction. On récupérera donc le premier paramètre avec l'expression $1, le deuxième avec l'expression $2, le troisième avec l'expression $3, etc…

Le code de retour d'un shell pourra être fixé à l'aide de la commande **exit**. Par exemple :

```
exit 0
```

Ce code de retour pourra être récupéré par l'appelant à l'aide de l'expression $?.

V. Fonctionnement du système

V.1. Comment fonctionne le système

UNIX est un système multi-tâches, ce qui signifie que plusieurs programmes peuvent s'exécuter en même temps sur la même machine. Comme on ne dispose en général que d'un processeur, à un instant donné un seul programme peut s'exécuter. Le noyau va donc découper le temps en tranches (quelques millièmes de secondes) et attribuer chaque tranche à un programme. On parle de système en temps partagé. Du point de vue des programmes, tout se passe comme si l'on avait une exécution réellement en parallèle.

L'utilisateur voit s'exécuter ses pro-grammes en même temps, mais d'autant plus lentement qu'ils sont nombreux.
On appelle *processus* un programme en cours d'exécution. A un instant donné, un processus peut être dans l'un des états suivants :

actif : le processus s'exécute sur un processeur (il n'y a donc qu'un seul processus actif en même temps sur une machine mono-processeur);

prêt : le processus peut devenir actif dès que le processeur lui sera attribué par le système;

bloqué : le processus a besoin d'une *ressource* pour continuer (attente d'entrée/-sortie par exemple). Le blocage ne peut avoir lieu qu'à la suite d'un appel système. Un processus bloqué ne consomme pas de temps processeur; il peut y en avoir beaucoup sans pénaliser les performances du système.

Remarque : le passage de l'état actif à l'état prêt (interruption) est déclenché par le noyau lorsque la tranche de temps allouée au processus s'est écoulée. Concrète-ment, le noyau programme dans ce but une *interruption* matérielle. Ceci implique que toute section d'un programme utilisateur peut se voir interrompue n'importe où ; les processus n'ont bien sûr pas accès aux instructions de masquage d'interruptions. Toute opération *critique* (ininterruptible) devra donc être réalisée dans le noyau, un processus y accédant via un appel système.

V.2. Notion de processus, le processus d'initialisation

Initialisation et arrêt

Nous allons décrire la procédure de démarrage et d'arrêt d'un système Linux sur un PC. Cette procédure se décompose en plusieurs parties :

- le BIOS charge le programme de démarrage

- celui-ci lit et démarre le noyau
- le noyau lance le processus init
- ce dernier initialise les autres processus.

Il est possible de démarrer LINUX à partir de différents média et à partir de différents programmes de démarrage. Le chargeur standard de Linux est **LILO** (LInux LOader) et il peut être installé :

- sur le secteur de démarrage d'une disquette
- sur le secteur de démarrage du disque (MBR, Master Boot Record)
- sur le secteur de démarrage de la partition système

Dans le dernier cas, il doit être activé par le chargeur principal (*primary loader*)

Le démarrage du noyau

En fonction de sa configuration, le BIOS va chercher le programme de démarrage (*primary loader*) sur le premier secteur d'une disquette ou d'un disque dur (MBR *Master Boot Record*) et le charge en mémoire. Ensuite le programme de démarrage de second niveau (*secondary loader*) est chargé puis invoqué. Sur l'écran, le programme de démarrage affiche l'invite LILO:. Le premier L est écrit avant de charger le deuxième programme et le I est écrit avant de l'exécuter. Le deuxième L signifie que le chargeur secondaire s'exécute mais qu'il y a une erreur d'accès au disque. Dès que le O s'affiche, le système peut être démarré.

A l'invite de **LILO**, l'utilisateur peut entrer des commandes pour spécifier le noyau à démarrer, des paramètres pour un périphérique ou pour démarrer en mono-utilisateur (single).

Dès que le noyau est chargé et décompressé, il prend la main et s'initialise. Il réserve de la mémoire pour son utilisation, initialise les périphériques dont il a besoin, attache la zone d'échange (*swap*) et monte la racine du système de fichier.

Le compte-rendu de l'initialisation du système est donné dans le fichier /var/log/dmesg.

Le processus init

Ce programme est le premier processus lancé par le noyau. Il est chargé de démarrer les processus systèmes et d'en relancer certains lorsqu'ils se terminent, et ce durant la totalité du fonctionnement du système.

Sa configuration s'effectue dans le fichier /etc/inittab.

Le fichier /etc/inittab

Ce fichier contient des lignes respectant le format suivant :

```
code:niveau:action:commande
```

Le champ code contient une séquence de un à quatre caractères (deux pour compatibilité) unique pour identifier la ligne dans le fichier.

Le champ niveau donne le niveau d'exécution pour lequel cette ligne doit être prise en compte. La notion de niveau d'exécution permet de spécifier des configurations d'exécution différentes. Un standard existe et est résumé dans la table ci-dessous. Il est possible de spécifier plusieurs niveaux lorsque la commande associée doit être lancée à différents niveaux d'exécution.

Niveau	Description
0	Arrêt de la machine
1	mode mono-utilisateur, seul le super-utilisateur peut se connecter
2	mode multi-utilisateurs, avec peu de services réseaux
3	mode multi-utilisateurs, avec tous les services réseaux (mode par défaut)
4	définissable par l'utilisateur
5	Démarrage de X11 au boot
6	redémarrage de la machine

Le champ action définit la manière d'exécuter la commande du champ commande.

Le tableau ci-après présente les actions les plus courantes :

Action	Description
respawn	relance la commande lorsqu'elle se termine
wait	attend la fin de la commande avant de continuer
once	la commande est exécutée une fois
boot	la commande est exécutée au démarrage du système (le champ niveau est ignoré)

bootwait	comme ci-dessus avec attente
off	ne rien faire (permet de conserver la ligne pour une utilisation future)
initdefault	permet de spécifier le niveau d'exécution par défaut
sysinit	la commande est exécutée au démarrage avant celles des directives boot et bootwait
ctrlaltdel	la commande est exécutée lorsque l'utilisateur tape les trois caractères <CTRL>-<ALT>-<SUPPR> sur le clavier
powerfail	la commande est exécutée lorsque le processus init reçoit le signal SIGPWR (défaut d'alimentation)

Le répertoire /etc/rc.d

Ce répertoire contient les scripts utilisés pour l'initialisation du système. Ils sont prévus pour démarrer les différents services et processus et effectuer quelques vérifications de configuration.

La table suivante présente les différents scripts :

Niveau	**Description**
rc.sysinit	exécuté une fois au démarrage pour initialiser le système
rc	script de gestion du niveau d'exécution. Il le reçoit en paramètre.
rc.local	script utilisé pour les initialisations particulières à la machine
init.d	répertoire contenant les scripts d'initialisation des sous-systèmes
rc0.d, rc1.d, rc2.d, rc3.d, rc4.d, rc5.d et rc6.d	répertoires contenant des liens sur les scripts du répertoire init.d devant être lancés à un niveau d'exécution particulier

Le fichier rc.sysinit réalise les opérations suivantes :

- initialise la variable PATH pour les autres scripts
- active la partition de swap
- initialise le nom du système (hostname)
- vérifie l'intégrité du système de fichiers
- démarre la gestion des quotas

- initialise le "Plug and Play"
- prépare la gestion des modules
- initialise l'horloge système
- détruit les fichiers de verrouillage

Le fichier rc exécute les scripts du répertoire rcN.d où N correspond au niveau d'exécution. Ces scripts sont des liens symboliques sur les fichiers de démarrage des sous-systèmes du répertoire init.d. Le lien reprend le nom du fichier d'origine précédé de la lettre S et d'un nombre pour les scripts de démarrage ou de K et d'un nombre pour les scripts d'arrêt du sous-système. La valeur numérique permet de spécifier l'ordre d'exécution des scripts.

Les scripts du répertoire init.d. peuvent être appelés par la suite pour forcer un arrêt ou un démarrage d'un sous-système. La syntaxe utilisée est:

```
/etc/rc.d/init.d/script [start | stop | restart]
```

La plupart des démons du système enregistrent leur numéro de processus dans un fichier du répertoire /var/run.

Après l'exécution des scripts, le processus **init** lance les processus en mode respawn. Ce sont, en général, les programmes d'invitation à l'ouverture de session. Ces programmes doivent être relancés dès qu'une session se termine.

L'arrêt du système

Le système Linux doit être arrêté par une commande système et non en coupant l'alimentation secteur directement. Un arrêt brutal du système peut causer des pertes de données dues aux applications en fonctionnement.

La procédure d'arrêt permet :

- d'avertir les utilisateurs que le système doit être arrêté (commencer quelques jours avant)
- de demander aux applications de s'arrêter et de fermer les connexions et les fichiers ouverts

- de passer le système en mode mono-utilisateur

- de vider les tampons mémoire du cache disque

Le système garde une trace du fait qu'il est démarré pour permettre une vérification d'intégrité dans le cas d'un arrêt brutal.

La commande d'arrêt du système est la commande **shutdown** qui permet, selon les options utilisées :

- de donner l'heure de l'arrêt (*now*, *hh:mm*, *+minutes*)

- de donner le mode arrêt (arrêt ou redémarrage)

V.3. Notion de services et gestion de bas de services

- *# les scripts de /etc/rc.d/init.d/ sont appelés avec un paramètre start, stop, status, restart Par exemple si on a modifié la configuration du serveur Samba dans le fichier* ***smb.conf****, il faut relancer ce service par la commande /etc/rc.d/init.d/smb restart*

- Quand on souhaite démarrer un service, il faut placer dans **/etc/rc.d/init.d** le script de démarrage du service, puis créer un lien symbolique dans chacun des répertoires**/etc/rc.d/rcx.d** (x=0..6), avec comme règle de créer ce lien symbolique avec un nom commençant par S (comme Start) et un K (comme K).

- Les liens symboliques commençant par K sont lus les premiers, et le numéro indique l'ordre dans lequel les fichiers seront lus (normalement lorsque vous faites une installation à partir d'un rpm cela est fait automatiquement).

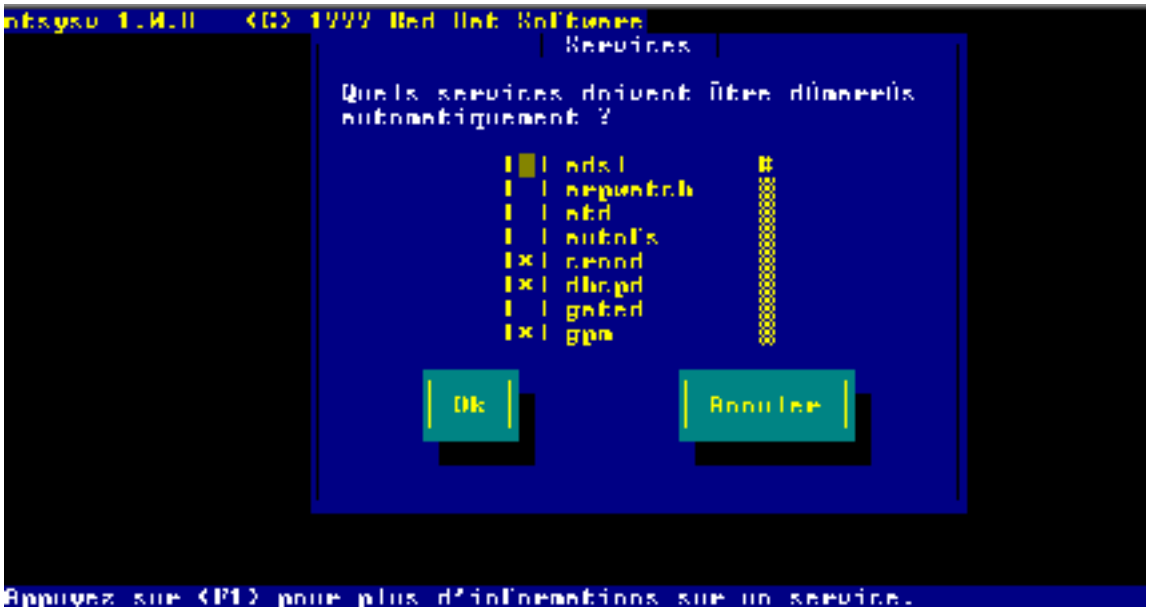
- Ainsi pour gérer le serveur dns on doit créer les liens symboliques vers le script de démarrage :**named**
  ln -s /etc/rc.d/init.d/named /etc/rc.d/rc0.d/K40named *Au niveau d'arrêt 0 on l'arrête*
  ln -s /etc/rc.d/init.d/named /etc/rc.d/rc1.d/K40named *pas de réseau à ce niveau*
  ln -s /etc/rc.d/init.d/named /etc/rc.d/rc2.d/K40named *bind peut etre démarré ici*
  ln -s /etc/rc.d/init.d/named /etc/rc.d/rc3.d/S40named
  ln -s /etc/rc.d/init.d/named /etc/rc.d/rc4.d/K40named *niveau inutilisé sur Red Hat*
  ln -s /etc/rc.d/init.d/named /etc/rc.d/rc5.d/S40named

```
ln -s /etc/rc.d/init.d/named /etc/rc.d/rc6.d/K40named    Le niveau 6 étant le reboot, il faut l'arreter
```

Outils de gestion du démarrage des services

Il est fastidieux de créer les liens précédemment décrits "à la main".
Il existe un outil **ntsysv** qui permet de déterminer les services devant être démarrés à l'initialisation du système, et **ksysv**, frontal graphique très convivial.

En ligne de commande : **chkconfig --level 3** ***smb [on | off]*** pour activer ou non le service samba au démarrage en niveau 3



V.4. Gestion des processus

Un des avantages des lignes de commandes par rapport aux environnements graphiques est la facilité avec laquelle elles permettent de contrôler les processus. Ce paragraphe décrit les principales méthodes pour lancer et arrêter un processus, ainsi que pour lui fournir les données sur lesquelles il doit travailler et récupérer ses résultats.

Lancement d'un programme en arrière plan

Le lancement normal d'un programme se fait en tapant sa ligne de commande et en appuyant sur la touche de validation. Le shell ne rendra pas la main et ne permettra pas de lancer un autre programme tant que le processus en cours ne sera pas terminé. Cependant, vous pouvez fort bien désirez lancer en arrière plan une commande dont la durée d'exécution peut être très longue et continuer à travailler. Après tout, Linux est multitâche… Eh bien, rien de plus facile !

Pour lancer une commande en arrière plan, il faut :

- s'assurer que la commande aura toutes les informations nécessaires pour travailler sans intervention de l'utilisateur (ou, autrement dit, que la commande ne soit pas interactive) ;

- ajouter une esperluette (caractère « & ») à la fin de la ligne de commande.

Par exemple, la commande suivante :

```
cp /cdrom/kernel/linux-2.2.10.tar.gz . &
```

copiera l'archive du noyau 2.2.10 du CD-ROM vers le répertoire courant, et s'exécutera en arrière plan.

Lorsqu'une commande est lancée en arrière plan, le shell affiche deux nombres qui permettront de l'identifier par la suite. Le premier nombre, indiqué entre crochets, est le numéro de « job » du shell. Ce numéro sert à identifier les commandes du shell de manière unique. Un job est donc en réalité une commande du shell, simple ou complexe. Le deuxième numéro est le numéro de processus (« PID », pour « Process IDentifier ») dans le système du processus maître du job. Le PID est un numéro unique dans le système, qui permet d'identifier de manière unique les processus en cours. Ces deux nombres permettront de manipuler les processus, avec les commandes que l'on verra plus tard.

Il ne faut pas confondre les numéros de job avec les numéros de processus. Premièrement, un numéro de job n'est unique que dans un shell donné, et n'a aucune signification au niveau du système complet, alors que le numéro de processus est attribué par le système à chaque programme en cours d'exécution. Ensuite, une même commande du shell peut lancer plusieurs processus conjointement. Dans ce cas, il y a bien évidemment plusieurs numéros de processus, mais un seul et unique job. Ce genre de situation se produit par exemple lors de l'utilisation d'une redirection du flux de sortie standard d'un processus vers le flux d'entrée standard d'un autre processus.

Le numéro de processus affiché par le shell lors du lancement d'une ligne de commande complexe représente le PID du processus maître de la commande, c'est à dire, en pratique, le dernier processus d'une série de redirections ou le processus du shell exécutant les commandes complexes. Ceci signifie que dans tous les cas de configuration, ce PID est celui du processus qui contrôle l'ensemble des opérations effectuées par la ligne de commande. C'est donc par ce processus que l'on peut manipuler la commande complète, par exemple pour l'interrompre.

Il est possible de retrouver le PID du processus maître d'une commande à partir du numéro de job correspondant du shell. Ceci se fait simplement, en utilisant l'expressions suivante :

```
%job
```

où job est le numéro du job dont on cherche le PID. Ainsi, dans toutes les commandes décrites ci-dessous, le PID peut être utilisé directement, ou être remplacé par le numéro du job préfixé du caractère de pourcentage.

Listing des processus

Il n'est pas nécessaire de retenir tous les numéros de jobs et tous les PID des processus en cours d'exécution. Il existe en effet des commandes permettant d'obtenir la liste des processus et des jobs. La plus simple à utiliser est bien évidemment la commande du shell pour obtenir la liste des jobs avec leurs lignes de commandes. Pour obtenir cette liste, il suffit de taper la commande suivante :

```
jobs
```

qui affiche, dans l'ordre, le numéro de job, l'état du processus correspondant, et la ligne de commande.

Une autre commande, plus bas niveau, permet d'obtenir des informations plus complète directement à partir du système. Il s'agit de la commande **ps**, dont la syntaxe est donnée ci-dessous :

```
ps [options]
```

Les options les plus utiles sont sans doute x, qui permet de demander l'affichage de toutes les commandes en cours d'exécution et non pas seulement les processus en cours d'exécution dans le shell où la commande **ps** est exécutée, et a, qui permet d'obtenir l'affichage de toutes les commandes, pour tous les utilisateurs connectés. Ces deux options peuvent être cumulées, et la commande suivante :

```
ps ax
```

affiche donc toutes les commandes en cours d'exécution sur le système.

Les informations les plus intéressantes affichées par **ps** sont le PID du processus, qui est donné par le premier nombre affiché, et la ligne de commande, qui est la dernière information affichée. Pour plus de détails sur la commande **ps**, veuillez consulter la page de manuel correspondante.

Notion de signal

Dans un système Unix, tous les processus peuvent recevoir des messages, envoyés soit par l'utilisateur, soit par un autre processus, soit par le système. Ces messages sont appelés *signaux*. La plupart des signaux sont envoyés par le système pour indiquer au processus qu'il a fait une faute et qu'il va être terminé. Cependant, ce n'est pas toujours le cas : certains signaux sont envoyés uniquement dans le cadre de la communication entre les processus, et certains autres ne peuvent même pas être captés par le processus et sont traités directement par le système. Nous n'entrerons pas en détail dans la gestion des signaux ici, car cela nous emmènerait trop loin. Cependant, la manière d'envoyer un signal à un processus à partir du shell sera décrite.

L'envoi d'un signal se fait avec la commande **kill**, avec la syntaxe suivante :

```
kill [-signal] PID
```

où signal est une option qui permet de préciser le signal qui doit être envoyé, et PID est le numéro du processus qui doit le recevoir. Les numéros de signaux les plus importants sont décrits dans le tableau ci-dessous :

Principaux signaux Unix

Numéro de signal	Signification
15	Signal de terminaison de processus.
9	Signal de destruction inconditionnelle de processus.
19	Signal de suspension de processus.
18	Signal de reprise d'exécution d'un processus suspendu.

Lorsqu'aucun signal n'est spécifié, le signal 15 de terminaison est utilisé par défaut. Ce signal demande au processus en cours d'exécution de se terminer immédiatement. Il peut être capté par le processus, pour lui donner une chance d'enregistrer les données sur lesquelles il travaillait et de libérer les ressources qu'il utilisait. Pour certains processus, ceci ne fonctionne pas, et il faut utiliser le signal de destruction du processus à l'aide de la commande suivante :

```
kill -9 PID
```

Attention cependant à cette commande : le processus est immédiatement détruit, sans autre forme de procès. Il peut donc s'ensuivre une perte de données, n'en abusez donc pas trop.

Arrêt d'un processus

Tout processus lancé en ligne de commande peut être arrêté immédiatement sous Linux. Pour cela, deux méthodes sont disponibles. La première consiste à taper la combinaison de touche CTRL+C lorsque le processus est en cours d'exécution interactive (c'est à dire lorsqu'il n'a pas été lancé en arrière plan). S'il a été lancé en arrière plan, on peut soit le ramener en avant plan (avec la commande **fg**, que l'on verra plus loin) avant d'utiliser CTRL+C, soit lui envoyer le signal de terminaison à l'aide de la commande **kill** vue précédemment. Dans le cas d'une ligne de commande, le signal de terminaison est transmis au processus maître de la ligne de commande, et est ensuite propagé à l'ensemble des processus fils de ce processus. Ceci signifie que tous les processus invoqués dans le cadre de cette commande sont également arrêtés.

Gel d'un processus

Il est possible de « geler » un processus en cours d'exécution, c'est à dire de le suspendre, sans pour autant l'arrêter définitivement. Ceci peut être utilisé pour libérer un peu les capacités de calcul, lorsque ce processus consomme trop de ressources par exemple. Pour cela, deux méthodes sont possibles :

- soit on utilise la combinaison de touches CTRL+Z, lorsque le processus est en avant plan ;

- soit on envoie le signal 19 au processus (signal « STOP ») à l'aide de la commande **kill**.

La première méthode est recommandée pour les processus lancés par une ligne de commande complexe, car le signal STOP est envoyé au processus maître de la commande, et est propagé à l'ensemble des processus fils de ce processus. Ceci signifie que tous les processus invoqués dans le cadre de cette commande sont également gelés. La deuxième méthode est plus bas niveau, et permet de geler n'importe quel processus que l'on a lancé.

Relancement d'un processus

Un processus suspendu peut être relancé soit en avant plan, soit en arrière plan. Pour relancer un processus en avant plan, il faut utiliser la commande suivante :

```
fg PID
```

où PID est le PID du processus à relancer en avant plan. **fg** est l'abréviation de l'anglais « foreground », ce qui signifie « avant plan ». Il faut attendre que ce processus se termine pour entrer de nouvelles commandes. Par conséquent, on ne peut lancer en avant plan qu'un seul processus.

De même, pour lancer un processus en arrière plan, il faut utiliser la commande **bg**, qui est l'abréviation de l'anglais « background ». Cette commande s'utilise de la même manière que la commande **fg** :

```
bg PID
```

Le relancement d'un processus suspendu peut également se faire en lui envoyant le signal 18 à l'aide de la commande **kill**.

V.5. Notion de ressources systèmes et travail avec les fichiers de gestion et de configuration des ressources systèmes

L'utilisation du disque : Les commandes de suivi

La commande **df** permet de visualiser la place disponible sur les différents systèmes de fichiers raccordés à la machine. Cette commande affiche :

- le nom du fichier spécial
- la taille totale (en nombre de blocs)
- l'espace utilisé

- l'espace libre
- le pourcentage de place utilisée
- le point de montage

L'option -i permet d'afficher les mêmes valeurs pour les inodes.

La commande **du** permet d'afficher la place disque utilisée par un ensemble de fichiers. Pour chaque fichier, **du** affiche la place occupée en nombre de blocs et parcourt les sous-répertoires de façon récursive. L'option -s permet d'afficher la somme des tailles des fichiers pour un répertoire.

L'utilisation du disque : Gestion des quotas

Linux permet d'affecter des quotas d'utilisation du disque pour un utilisateur ou un groupe d'utilisateurs sur un système de fichiers donné. Cela permet de limiter l'utilisation de la place disque ou des inodes en fonction de deux limites :

- une limite douce qui peut être dépassée pendant un délai donné
- une limite dure qui ne peut pas être dépassée

La gestion des quotas est démarrée dans le fichier rc.sysinit grâce à la commande **quotaon**.

Pour configurer la gestion des quotas, il faut ajouter dans le fichier /etc/fstab les mots usrquota ou grpquota au niveau des options de montage :

```
/dev/hda2   /home   ext2   defaults,usrquota,grpquota  1 1
```

La définition des valeurs des quotas est effectuée à l'aide de la commande **edquota**. Cette commande affiche les informations courantes pour l'utilisateur demandé :

```
Quotas for user sandra:
/dev/sda5: blocks in use: 43552, limits (soft = 50000, hard = 60000)
    inodes in use: 907, limits (soft = 0, hard = 0)
```

Il est possible de modifier les champs soft et hard pour les limiter en nombre de blocs et en inodes. Une valeur à zéro annule la gestion des quotas pour cet utilisateur.

La commande **edquota -t** permet de fixer la période de grâce pour la limite douce.

La commande **repquota -a** permet de visualiser les valeurs courantes pour l'ensemble des utilisateurs.

Les processus et la mémoire

La commande **ps** permet de visualiser la liste des processus en cours d'exécution. Les différentes options sont :

Option	Signification
a	montre les processus de tous les utilisateurs
x	montre les processus non attachés à un terminal
l	format long
m	affiche les informations sur la mémoire

Les principales valeurs affichées par cette commande sont :

Option	Signification
PID	l'identificateur de processus
TTY	le terminal de contrôle
STAT	l'état du processus
SIZE	affiche la taille du processus (texte + données + pile)
RSS	taille du processus en mémoire
TIME	le temps d'exécution cumulé

La commande **top** affiche la liste des processus s'exécutant ainsi que plusieurs variables système. Cet affichage est rafraîchi périodiquement avec un intervalle configurable. Les processus sont triés par ordre décroissant d'utilisation du CPU.

Les valeurs affichées représentent :

- le temps depuis le démarrage de la machine
- la charge moyenne
- le nombre de processus
- le taux de charge du CPU

- l'utilisation de la mémoire
- l'utilisation de la zone d'échange

La commande **free** permet de visualiser l'utilisation de la mémoire. Elle affiche les valeurs pour :

- la mémoire disponible
- la mémoire utilisée
- la mémoire libre
- la mémoire partagée
- le tampon de cache disque
- l'utilisation de la zone d'échange

Le système de fichiers /proc

Le système de fichiers /proc donne l'accès, par l'intermédiaire de pseudos fichiers, aux variables internes du noyau.

A la racine de ce système de fichiers, on trouve un répertoire par processus en exécution ayant comme nom l'identificateur du processus. Dans chacun de ces répertoires, on a :

- cmdline : donne la ligne de commande
- cwd : c'est un lien sur le répertoire courant du processus
- environ : contient l'environnement du processus
- exe : c'est un lien sur le fichier exécutable

- fd : répertoire contenant les liens sur les fichiers ouverts
- maps : liste des zones mémoire du processus
- mem : contenu de l'espace d'adressage du processus
- root : lien sur le répertoire racine du processus
- stat, statm, status : informations sur l'état du processus
- Les principaux fichiers sont :
- devices : liste des pilotes avec les major numbers
- dma : liste des canaux dma utilisés
- interrupts : liste des interruptions utilisées
- ioports : liste des ports utilisés
- loadavg : charge moyenne du système
- meminfo : état de la mémoire
- modules : liste des modules chargés
- stat : informations sur le système
- net : répertoire contenant les informations sur le réseau
- scsi : répertoire contenant les informations sur les périphériques scsi
- sys : répertoire contenant des variables du noyau

- Les informations retournées par le pseudo fichier stat sont :

- cpu : temps passé dans les états user, nice, system et idle en 1/100 de seconde

- disk : pour les quatre premiers disques, donne le résumé des opérations effectuées

- disk_rio : lecture

- disk_wio : écriture

- disk_rblk : lecture blocs

- disk_wblk : écriture blocs

- page : pages lues et écrites

- swap : compte des échanges en lecture et en écriture

- intr : total des interruptions depuis le démarrage

- ctxt : nombre de changements de contexte

- btime : heure du démarrage

- processes : dernier PID utilisé

VI.Eléments de réseau

VI.1. Notion de réseau physique

Un réseau d'ordinateurs est une combinaison de matériels et de logiciels qui permet aux ordinateurs et autres périphériques (imprimantes, modems etc…) de communiquer entre eux à travers diverses formes de média de télécommunication.

Les réseaux peuvent être classifiés selon les distances qu'ils couvrent et selon qu'ils incluent la technologie web.

LAN (Local Area Network) : un réseau confiné à une petite zone géographique.

Intranet: un LAN qui inclut un serveur web

MAN (Metropolitan Area Network): un vieux terme qui décrit un réseau qui couvre une ville entière. Aujourd'hui le terme MAN est remplacé par le WAN

WAN (Wide Area Network) : Un réseau qui couvre une large zone géographique comme par exemple, une ville, un pays etc…

Internet : Le réseau global qui supporte la technologie web

Le modèle OSI

7	Application	***Mail, Web …***
6	Présentation	
5	Session	
4	Transport	***TCP***
3	Réseau	***IP*** *transfert de paquets*
2	Logique	*Acheminement des trames*
1	Physique	*Acheminement de signaux*

TCP/IP

TCP/IP est une suite de protocoles permettant à tous types d 'ordinateurs, utilisant divers systèmes d 'exploitation de communiquer entre eux.

Développé à la fin des années 60 comme une solution de réseau à commutation de paquets, il est devenu depuis les années 90, le protocole le plus utilisé dans les réseaux d 'ordinateurs.

C 'est un système ouvert qui constitue la base de l'Internet.

Il fournit les couches 3 et 4 du modèle ISO
IP ou Internet Protocol est la couche réseau qui contrôle le mouvement des paquets sur le réseau.
IP ne s'assure pas que les paquets sont délivrés
IP fonctionne en mode non connecté.
Deux protocoles assurent la couche transport: TCP et UDP.
TCP fonctionne en mode connecté et s 'assure que les paquets sont délivrés.
UDP transfert seulement des paquets de données appelés DATAGRAMS entre les machines.

VI.4. Adressage IP

Chaque machine sur l'Internet est identifiée par une **adresse IP.**
L'adresse IP doit être **unique**. Si deux machines ont la même adresse IP, le réseau ne saurait à qui délivrer les données.
Pour assurer l 'unicité des adresses IP, elles sont allouées en bloc.

Une adresse IP de IPv4 est un **nombre binaire de 32 bits**, Une chaîne de caractères de 32 zéros et uns.

Une adresse IP de Ipv6 est un **nombre binaire de 128 bits**. Encore expérimentale.

Pour des facilités d'écriture, Les adresses IP sont fractionnées en quatre pièces de 8 bits et converties en décimal avec un point entre les différentes parties (exemple 192.168.0.1)

The Internet Assigned Numbers Authority (IANA), donne autorité sur des blocs d 'adresses aux organes d'enregistrement.
Les organes d'enregistrement (RIPE NCC, ARIN, APNIC et bientôt AFRINIC) assignent de larges blocs d'adresses IP aux ISP qui en retour les fractionnent en des blocs plus petits pour leurs clients .

Pour créer les blocs, les adresses IP sont fractionnées en deux parties : **adresse réseau (préfixe) et l'adresse machine**. Pour spécifier la position du fractionnement, vous pouvez utiliser un slash et un nombre de bits de la partie réseau.
Par exemple, le bloc d'adresse ''199.2.192.64/26 '' a 26 bits pour le réseau ; les six restants pour les machines.
Pour qu'un réseau IP fonctionne, il est vital que toutes les machines sur un réseau ait la même adresse réseau (dans ce exemple, les 26 bits de gauche) et différentes adresses machines.

Masque de sous-réseau

La séparation entre l'adresse réseau et l'adresse machine peut être aussi représentée par un masque de sous-réseau. Ce qui n'est qu'un nombre de '1' a chaque position de l'adresse réseau, et un '0' pour chaque bit de l'adresse machine. Par exemple, le masque pour un réseau /26 est 26 '1' et 6 '0'.

Adressage privé

L'IANA(www.iana.org) a réservé les trois blocs d 'adresses IP suivants pour les Internets privés:

10.0.0.0 - 10.255.255.255 (10/8)
172.16.0.0 - 172.31.255.255 (172.16/12)
192.168.0.0 - 192.168.255.255 (192.168/16)

Routage IP

De base, le routage IP est simple. Si la destination est directement connectée à la source (liaison point à point) ou sur un média partagé (Ethernet par exemple), Le paquet est directement envoyé à la source. Dans le cas contraire, le paquet est envoyé à un routeur par défaut.

V.5. Configuration réseau d’une machine

La configuration du réseau nécessite donc la configuration du protocole IP et des services TCP, UDP et ICMP (entre autres). Cette opération se fait en définissant l'adresse IP le masque de sous-réseau et les routes à utiliser. Vient ensuite la configuration du nom de la machine locale, de son domaine, des noms de machines qu'elle peut résoudre elle-même et des serveurs de DNS qu'elle doit utiliser pour les autres noms.

Il est quasiment certain que votre distribution dispose d'un outil permettant d'effectuer la configuration du réseau simplement. Connaissant à présent la signification des termes utilisés dans les réseaux TCP/IP, vous devriez pouvoir parvenir à une configuration valide relativement simplement. Il est fortement recommandé de consulter la documentation de votre distribution. Les commandes de configuration du réseau sont souvent appelées dans les scripts de démarrage de la machine, ou dans les scripts de changement de niveau d'exécution. Toutefois, il peut être utile de connaître ces commandes, ne serait-ce que pour comprendre comment votre système fonctionne. Cette section a donc pour but de vous présenter ces commandes, ainsi que les principaux fichiers de configuration du réseau utilisés sous Linux.

Configuration statique des interfaces réseau

La principale commande permettant de configurer le réseau est la commande **ifconfig**. Comme son nom l'indique (« InterFace CONFiguration »), elle permet de configurer les interfaces réseaux de la machine. Il faut savoir qu'il existe plusieurs types d'interfaces réseaux. Les plus courants sont les trois types d'interfaces suivants :

- l'interface loopback, qui représente le réseau virtuel de la machine, et qui permet aux applications réseaux d'une même machine de communiquer entre elles même si l'on ne dispose pas de carte réseau ;

- les interfaces des cartes réseaux (que ce soient des cartes Ethernet, TokenRing ou autres) ;

- les interfaces ppp, plip ou slip, qui sont des interfaces permettant d'utiliser les connexions sérielles, parallèles ou téléphoniques comme des réseaux.

La configuration d'une interface comprend l'initialisation des drivers nécessaires à son fonctionnement et l'affectation d'une adresse IP à cette interface. La syntaxe générale que vous devrez utiliser est la suivante :

```
ifconfig interface adresse netmask masque up
```

où interface est le nom de l'interface réseau que vous voulez configurer, adresse est l'adresse IP que cette interface gérera, et masque est le masque de sous-réseau que vous utilisez. Les interfaces que vous aurez à configurer seront certainement des interfaces Ethernet, auquel cas vous devrez utiliser les noms eth0, eth1, etc. dans la commande **ifconfig**. Si vous désirez configurer l'interface loopback, vous devrez utiliser le nom d'interface lo.

Le paramètre up donnée à **ifconfig** lui indique que l'interface doit être activée. Cela signifie que dès que la commande **ifconfig** s'achèvera, votre interface réseau sera active et fonctionnelle. On ne peut faire plus simple... Bien entendu, il existe le paramètre inverse : down. Ce paramètre s'utilise tout simplement dans la commande **ifconfig** avec la syntaxe suivante :

```
ifconfig interface down
```

où interface est toujours le nom de l'interface.

Un exemple de configuration très classique est le suivant :

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
```

> **Note :** Prenez bien garde, lorsque vous écrivez vos adesses IP, à ne pas rajouter de 0 supplémentaire devant les nombres qui la constituent. En effet, il existe une convention en informatique qui dit que les nombres préfixés d'un 0 sont codés en octal, c'est-à-dire en base 8. Il va de soi qu'une adresse IP codée en octal n'utilise pas les mêmes nombres que lorsqu'elle est exprimée en décimal, aussi éprouveriez-vous quelques difficultés pour diagnostiquer le non fonctionnement de votre réseau si vous faisiez cette erreur !

Le noyau utilisera par défaut le nombre 255 pour les adresses de broadcast dans les composantes de l'adresse IP qui ne fait pas partie de l'adresse de sous-réseau. Si vous désirez utiliser une autre adresse (en général, l'alternative est de prendre l'adresse du sous-réseau), vous devrez utiliser l'option broadcast adresse dans la commande **ifconfig**. Cependant, le comportement par défaut convient à la plupart des réseaux. La commande de configuration donnée en exemple ci-dessus sera alors :

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0 broadcast 192.168.0.0 up
```

Enfin, il est possible d'affecter plusieurs adresses IP à certaines interfaces réseau. C'est en particulier le cas pour toutes les interfaces

réseau classiques, mais bien entendu cela n'est pas réalisable avec les interfaces de type point à point comme, par exemple, les interfaces des connexions ppp. Lorsqu'une interface dispose de plusieurs adresses, la première est considérée comme l'adresse principale de l'interface, et les suivantes comme des *alias*. Ces alias utilisent comme nom le nom de l'interface réseau principale et le numéro de l'alias, séparés par deux points (caractère ':'). Par exemple, si l'interface eth0 dispose d'un alias, celui-ci sera nommé eth0:0. Ainsi, pour fixer l'adresse d'un alias d'une interface réseau, on utilisera la syntaxe suivante :

```
ifconfig interface:numéro add adresse netmask masque
```

où interface est toujours le nom de l'interface, numéro est le numéro de l'alias, adresse est l'adresse IP à attribuer à cet alias, et masque est le masque de sous-réseau de cette adresse.

La commande **ifconfig** est en général appelée dans les scripts d'initialisation du système, qui ont été générés par l'utilitaire de configuration réseau de votre distribution. Si vous effectuez un **grep** sur « ifconfig » dans le répertoire /etc/rc.d/ (ou /sbin/init.d/, selon votre distribution), vous trouverez la commande de démarrage du réseau. Il se peut qu'une variable d'environnement soit utilisée à la place de l'adresse IP que vous avez choisie. Quoi qu'il en soit, vous devez sans aucun doute avoir les lignes suivantes, qui permettent l'initialisation de l'interface loopback :

```
ifconfig lo 127.0.0.1 netmask 255.0.0.0 up
```

Définition des règles de routage

La deuxième étape dans la configuration du réseau est la définition des règles de routage. Il est possible de définir plusieurs règles de routage actives simultanément. L'ensemble de ces règles constitue ce qu'on appelle la *table de routage*. La règle utilisée est sélectionnée par le noyau en fonction de l'adresse destination du paquet à router. Chaque règle indique donc un critère de sélection sur les adresses, et l'interface vers laquelle doivent être transférés les paquets dont l'adresse destination vérifie cette règle.

La commande utilisée pour définir une route est, chose surprenante, la commande système **route**. Sa syntaxe est donnée ci-dessous :

```
route opération [-net | -host] adresse netmask masque interface
```

où opération est l'opération à effectuer sur la table de routage. L'opération la plus courante est simplement l'ajout d'une règle de routage, auquel cas add doit être utilisé. L'option suivante permet d'indiquer si le critère de sélection des paquets se fait sur l'adresse du réseau destination ou plus restrictivement sur l'adresse de la machine destination. En général, il est courant d'utiliser la sélection de toutes les adresses d'un même réseau et de les router vers une même interface. Dans tous les cas, adresse est l'adresse IP de la destination, que celle-ci soit un réseau ou une machine. Si la destination est un réseau, il faut indiquer le masque de sous-réseau masque à l'aide de l'option netmask. Enfin, interface est l'interface réseau vers laquelle doivent être envoyés les paquets qui vérifient les critères de sélection de cette règle.

Par exemple, la règle de routage à utiliser pour l'interface loopback est la suivante :

```
route add -net 127.0.0.0 netmask 255.0.0.0 lo
```

Cette règle signifie que tous les paquets dont l'adresse de destination appartient au sous-réseau 127.0.0.0/8 doivent être transférés vers l'interface loopback. Cela implique en particulier que les paquets à destination de la machine d'adresse IP 127.0.0.1 (c'est-à-dire la machine locale) seront envoyés vers l'interface loopback (ils reviendront donc sur la machine locale).

Une autre règle de routage classique est la suivante :

```
route add -net 192.168.0.0 netmask 255.255.255.0 eth0
```

Elle permet d'envoyer tous les paquets à destination du réseau 192.168.0.0/24 vers la première interface Ethernet. C'est typiquement ce genre de règle qu'il faut utiliser pour faire fonctionner un réseau local.

Il n'est normalement pas nécessaire d'ajouter les règles de routage pour les réseaux auxquel la machine est connectée. En effet, la configuration d'une carte réseau à l'aide de la commande **ifconfig** ajoute automatiquement à la table de routage une règle pour envoyer les paquets à destination de ce réseau par l'interface réseau qui y est connectée. Cependant, la commande **route** devient réellement nécessaire lorsqu'il faut définir les passerelles à utiliser pour l'envoi des paquets destinés à une machine à laquelle la machine locale ne peut accéder directement. Les règles de routage faisant intervenir une passerelle sont semblables aux règles de routage vues ci-dessus, à ceci près que l'adresse IP de la passerelle à utiliser doit être fournie. Pour cela, on utilise l'option gw (abréviation de l'anglais « Gateway »). La syntaxe utilisée est donc la suivante :

```
route add [-net | -host] adresse netmask masque gw passerelle interface
```

où passerelle est l'adresse IP de la passerelle à utiliser pour router les paquets qui vérifient les critères de cette règle. Les autres paramètres sont les mêmes que pour les règles de routage classique.

Par exemple, supposons qu'une machine soit connectée à un réseau d'adresse 192.168.0.0, et que sur ce réseau se trouve une passerelle d'adresse 192.168.0.1 permettant d'atteindre un autre réseau, dont l'adresse est 192.168.1.0. Une machine du réseau 192.168.0.0 aura typiquement les règles de routage suivantes :

```
route add -net 192.168.0.0 netmask 255.255.255.0 eth0
route add -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.0.1 eth0
```

La première règle permet, comme on l'a déjà vu, de communiquer avec toutes les machines du réseau local. La deuxième règle permet d'envoyer à la passerelle 192.168.0.1 tous les paquets à destination du réseau 192.168.1.0.

Inversement, si la passerelle utilise l'adresse 192.168.1.15 sur le réseau 192.168.1.0, les machines de ce réseau qui voudront accéder au réseau 192.168.0.0 devront spécifier la règle de routage suivante :

```
route add -net 192.168.0.0 netmask 255.255.255.0 gw 192.168.1.15 eth0
```

Bien entendu, il est nécessaire que toutes les machines des deux réseaux utilisent ces règles de routage pour que la communication entre les deux réseaux se fasse dans les deux sens.

Le problème de ces règles de routage est qu'elles spécifient l'adresse du réseau destination. Il est évidement hors de question d'utiliser une règle de routage différente pour toutes les adresses de réseaux possibles. Il est donc possible de définir ce qu'on appelle une passerelle par défaut, qui n'est rien d'autre que la passerelle vers laquelle doivent être envoyés tous les paquets qui n'ont pas vérifié les critères des autres règle de routage. La syntaxe à utiliser pour définir la passerelle par défaut est plus simple, puisqu'il n'est plus nécessaire de préciser les critères de sélection :

```
route add default gw passerelle interface
```

où la signification des paramètres passerelle et interface est inchangée.

Ainsi, pour reprendre l'exemple précédent, supposons que la machine 192.168.0.47 dispose d'une connexion à Internet. Pour que toutes les machines du réseau local puisse profiter de cette connexion, il suffit de demander à ce que tous les paquets qui ne vérifient aucune autre règle de routage soient envoyés à la passerelle 192.168.0.47. Cela se fait avec la règle de routage suivante :

```
route add default gw 192.168.0.47 eth0
```

Définition du nom de la machine

La configuration du nom d'un ordinateur n'est pas à proprement parler une opération indispensable, mais elle permet de le nommer de manière plus conviviale qu'en utilisant son adresse IP. La commande de base permettant de manipuler le nom de la machine locale est la commande **hostname**. Appelée telle quelle, elle renvoie le nom actuel de la machine :

```
hostname
```

Cette commande permet également de modifier ce nom, simplement avec la syntaxe suivante :

```
hostname nom
```

où nom est le nom à utiliser. Il est d'usage de n'utiliser que le nom de la machine, sans son domaine. Le nom de domaine sera déterminé automatiquement par le système à partir des informations issues de la configuration de la résolution des noms de domaines. Nous verrons cela dans le paragraphe suivant.

Cette commande est donc très simple à utiliser, et elle est en général appelée dans les scripts de démarrage du système. La plupart des

distributions utilisent le fichier /etc/HOSTNAME pour stocker le nom de la machine. Vous êtes bien entendu libre de choisir le nom que vous voulez pour votre ordinateur, mais vous devez vous assurer que ce nom est unique dans votre domaine !

Résolution des noms de domaines

La commande **hostname** ne permet de fixer que le nom de la machine locale. Pour les autres machines du réseau, il faut mettre en place les mécanismes de résolution de noms de domaines. Comme il l'a déjà été indiqué au début de ce chapitre, il existe deux solutions pour trouver l'adresse IP d'une machine à partir de son nom : la consultation d'une liste de noms stockée en local, soit l'interrogation d'un serveur de nom de domaines.

Le fichier /etc/host.conf permet de définir le comportement du système lors de la résolution d'un nom. Sa structure est très simple, puisqu'il y a une option de recherche par ligne. Dans la plupart des cas, les lignes suivantes sont suffisantes :

```
order hosts,bind
multi on
```

Elles permettent d'indiquer que la recherche des noms pour leur résolution doit se faire d'abord localement, puis par appel aux DNS si la recherche précédente a échoué. C'est en général le comportement désiré. La deuxième ligne permet de faire en sorte que toutes les adresses correspondant à une machine soient renvoyées. Si l'on avait utilisé l'option multi off, seule la première adresse IP trouvée aurait été renvoyée.

La liste de noms locale est stockée dans le fichier /etc/hosts (cela explique le nom hosts utilisé pour l'option order dans le fichier /etc/host.conf). Votre ordinateur connaîtra directement l'adresse IP de toutes les machines référencées dans ce fichier. Il est bon de placer ici une entrée pour les ordinateurs les plus couramment utilisés sur le réseau. Chaque entrée commence par une adresse IP, et est suivie de la liste des noms de la machine possédant cette adresse, séparés par des espaces. Pour ceux qui ne disposent pas de réseau local, ce fichier doit être relativement simple : seule la ligne affectant l'adresse 127.0.0.1 à la machine locale (appelée « localhost ») doit s'y trouver.

```
127.0.0.1    localhost
```

De la même manière, le fichier /etc/networks contient les adresses des réseaux. Ce fichier est utilisé par la commande route pour donner un nom aux différents réseaux. Chaque entrée est constituée du nom du réseau, suivi de son adresse IP. Encore une fois, ce fichier se réduit à sa plus simple expression pour ceux qui n'ont pas de réseau local, puisqu'il ne contiendra tout au plus qu'une entrée pour le réseau « loopback », sur lequel se trouve l'adresse de retour 127.0.0.1. Cette entrée aura donc la forme suivante :

```
loopback    127.0.0.0
```

La configuration des serveurs de noms est en revanche une opération nécessaire si l'on désire accéder à des machines dont on ne connaît que le

nom. Le fichier de configuration utilisé est cette fois le fichier /etc/resolv.conf. Sa structure est encore une fois très simple, avec une option par ligne, chaque option étant introduite par un mot-clé.

Le mot-clé domain permet d'indiquer le nom du domaine dont fait partie votre machine. Par exemple, si votre nom de domaine est « francophonie.org », vous devrez utiliser la ligne suivante :

```
domain francophonie.org
```

Le mot-clé search permet quant à lui de spécifier une liste de noms de domaines à ajouter par défaut aux noms de machines non complètement qualifiés. Les éléments de cette liste doivent être séparés par des espaces. La recherche d'une machine dont le nom ne comprend pas la partie de nom de domaine s'effectue en ajoutant au nom de la machine les noms des domaines indiqués ici, jusqu'à ce que la résolution du nom en adresse IP soit effectuée. Cette recherche commence bien entendu par le nom de domaine local, s'il a été défini. Il est donc recommandé d'indiquer votre nom de domaine dans cette liste de noms de domaines. Par exemple, si votre machine fait partie du domaine « francophonie.org », vous devrez utiliser la ligne suivante :

```
search francophonie.org
```

Ainsi, si vous recherchez l'adresse de la machine « host1 », la requête au DNS se fera avec le nom complètement qualifié « host1.francophonie.org ». Vous êtes bien entendu libre d'ajouter d'autres noms de domaines pour le cas où la résolution de nom échouerait sur ce domaine.

Enfin, l'option nameserver est essentielle, puisqu'elle permet de donner les adresses IP des serveurs de DNS auxquels doivent être adressées les requêtes de résolution de noms. Par exemple, si vous disposez de deux serveurs DNS, un primaire, d'adresse 192.168.0.10, et un secondaire, d'adresse 192.168.0.15, vous utiliserez la ligne suivante :

```
nameserver 192.168.0.10 192.168.0.15
```

Cette ligne est évidemment obligatoire, faute de quoi la résolution des noms de machines en adresse IP échouera pour toute machine qui ne se trouve pas dans votre fichier /etc/hosts.

VI.6. Utilisation des commandes d’administration de base des services réseaux

Les outils suivants sont indispensables à connaître lorsque l'on utilise un système sous Linux. On ne peut ici donner toutes les options de ces commandes. N'oubliez donc pas que l'on peut avoir plus d'aide en tapant la commande suivie de --help, mais aussi man commande. Exemple **netstat --help** ou **man netstat**. Enfin souvenez vous que sous Linux, on ne peut utiliser indifféremment les majuscules et les minuscules (la commande ping existe, pas la commande Ping).

ping

Cette commande est normalement connue de tous. Elle existe dans tous les systèmes. Elle permet de vérifier si une machine distante répond. Pour envoyer 5 pings à la machine dont l'adresse IP est 192.168.0.1.

ping -c 5 192.168.0.1

On peut aussi utiliser le nom de la machine, si celle-ci est renseignée dans votre fichier Hosts ou dans un serveur DNS.
On peut par exemple utiliser ping pour vérifier si la connexion est toujours active ou pour la monter.
Si vous ne placez pas l'option -c 5 pour n'envoyer que 5 pings, la commande ne s'arrête pas.

Utilisez alors Ctrl + C.

ifconfig

ifconfig permet de connaître la configuration de vos cartes réseau, mais aussi de changer celle-ci.
Pour changer la configuration de votre carte réseau, vous devez taper

ifconfig eth0 192.168.0.2 netmask 255.255.255.0 broadcast 192.168.0.255

Comme les valeurs que je viens de donner sont standards, vous pouviez simplement taper

ifconfig eth0 192.168.0.2

(le netmask et broadcast proposés sont ceux correspondant à une adresse de sous-réseau /24).
Attention au redémarrage de la machine ce changement sera perdu. Il vous faut donc modifier en même temps le fichier

/etc/sysconfig/network-script/ifcfg-eth0.
Vous pouvez utiliser **linuxconf** pour faire plus simplement le même travail.
On peut aussi désactiver une carte réseau

ifconfig eth0 down

et bien sûr la réactiver

ifconfig eth0 up

arp

La commande arp permet de mettre en correspondance des adresses IP et les adresses MAC. Les options possibles importantes sont
arp -a pour avoir toutes les entrées ARP de la table
arp -d nom_de_la_machine pour supprimer une entrée de la table
arp -s nom_de_la_machine adresses_mac pour ajouter une nouvelle entrée dans la table.

route

Cette commande permet de voir, d'ajouter ou d'enlever les routes se trouvant déclarées sur votre machine. Ainsi pour indiquer à votre machine où aller trouver les adresses qui ne sont pas les adresses de votre réseau local, vous devez lui indiquer la passerelle (ou gateway) vers laquelle elle doit envoyer tous les paquets.
Pour voir les routes indiquer

route –n

(on peut aussi utiliser **netstat -nr**)

L'option -n permet de ne pas avoir la résolution des noms.
Pour ajouter une route par defaut :

route add default gateway 192.168.0.1

(La passerelle vers qui j'envoie tous les paquets qui ne sont pas pour le réseau local).

Pour détruire cette route

route del default
Pour ajouter une route vers une machine indiquer

route add -host 195.98.246.28 gateway 192.168.0.1

(Indiquer le netmask si celui-ci n'est pas un mask correspondant à celui de votre sous-réseau).
Pour ajouter une route vers un réseau indiquer

route add -net 195.98.246.0 netmask 255.255.0.0 gateway 192.168.0.1

Enfin pour supprimer une de ces routes remplacer add par del.
La gateway ou passerelle correspond la plupart du temps à votre routeur.
Pour avoir la route que vous venez d'ajouter à chaque démarrage placer la commande dans le fichier */etc/rc.d/rc.local* par exemple.
On peut aussi utiliser **linuxconf** pour faire la même chose.

netstat

Voilà une commande moins connue et pourtant très utile. Je ne peux ici commenter toutes les options, je vous propose de lire le **man netstat**. Elle permet en effet de connaître les ports en écoute sur votre machine, sur quelles interfaces, avec quels protocoles de transport (TCP ou UDP), les connexions actives et de connaître les routes.
Pour voir les connexions actives

netstat –nt

pour les ports ouverts

netstat -ntl.

On peut aussi vérifier s'il existe une route par défaut, par exemple existe-t-il une route par défaut vers la machine 195.98.246.28 utilisez alors

netstat -nr | grep 195.98.246.28.

L'option -a énumère les ports en cours d'utilisation ou ceux qui sont écoutés par le serveur.
L'option -i donne des informations sur les interfaces réseau.

traceroute

Traceroute permet de déterminer la route prise par un paquet pour atteindre la cible sur internet. On peut utiliser soit l'adresse IP, soit le nom d'hôte. Attention certains FireWall ou routeurs ne se laissent pas voir avec la commande traceroute.
La commande traceroute est très utile pour savoir ou peut se trouver un blocage (plutôt ralentissement). Il existe un grand nombre d'options, entre autre il est possible de choisir les gateway (jusqu'à 8) pour atteindre une machine. Je vous conseille donc encore une fois de lire le man traceroute.

VI.7. Utilisation des services réseau de base

telnet

Telnet est l'outil indispensable à connaître. Il existe en tant que client sur tous les systèmes. Par contre Linux dispose en plus d'un serveur telnet permettant d'administrer à distance une machine. On peut ainsi administrer une machine linux depuis un Microsoft quelconque et, mais cela va de soi, depuis une autre machine linux.

En tant que client, telnet vous permet d'envoyer et de lire vos messages. Pour pouvoir administrer à distance, il faut que le serveur telnet soit installé sur la machine que vous souhaitez administrer. Pensez aussi à vérifier que cela est autorisé dans le fichier etc/xinetd.conf et dans / etc/hosts.allow et /etc/hosts.deny.

Si vous devez vous en servir sur un réseau local ou sur internet, préférez lui **SSH**, car alors les mots de passe ne se promènent pas en clair sur le réseau.

Par défaut il n'est pas possible de se connecter en root avec une connexion telnet. Vous devez utiliser un autre compte et utiliser la commande su.

ftp

ftp est un outil qui permet de télécharger des fichiers entre machines. Vous connaissez les clients ftp comme ws_ftp.

Sous Linux il existe un serveur ftp, que vous activez dans / etc/xinetd.conf. Il est installé par défaut dans toutes les distributions. Ce serveur ftp n'est pas lié à l'installation d'apache, comme pour les systèmes Microsoft où vous devez installer IIS pour bénéficier de ce service. Attention toutefois le serveur ftp pose un problème de sécurité important, utilisez plutôt **SFTP**, qui est disponible avec **SSH**.

Vous disposez aussi d'un client ftp en ligne de commande sous Linux comme sous Microsoft. La syntaxe étant pratiquement la même.

```
[philippe@lycee1 /]$ ftp localhost
Connected to localhost.
220 lycee1.ac-creteil.fr FTP server (Version wu-2.6.0(1) Mon Feb 28
10:30:36 EST
2000) ready.
Name (localhost:philippe): philippe
331 Password required for philippe.
Password:
230 User philippe logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> binary
200 Type set to I.
ftp> mget *
```

Voici les commandes que vous allez utiliser le plus :

dir	pour lister un répertoire
cd nom_du_répertoire	pour changer de répertoire
get mon_fichier	pour copier un fichier vers votre client (obtenir). Il se place alors dans le répertoire où vous vous trouviez.
mget *	copier tous les fichiers du répertoire vers votre station
put mon_fichier	pour copier un fichier vers le serveur
mput *	pour copier les fichiers se trouvant dans votre répertoire.
binary	pour copier en mode binaire.

exit	pour quitter

Il existe un grand nombre d'autres commandes. Mais vous avez là les principales, pour copier des fichiers entre machines. La commande ftp vous rendra un grand nombre de services, car elle permet assez simplement d'échanger des fichiers entre linux et windows, sans avoir à installer un client ftp ou à configurer samba.

dig

L'utilitaire dig permet d'interroger un serveur de nom (serveur dns) afin d'avoir des informations sur un domaine ou sur une machine. Par défaut dig utilise le serveur de nom configuré sur votre machine, vous pouvez toutefois interroger un autre serveur de nom.

```
[danny@serveur danny]$ dig intif.francophonie.org a

; <<>> DiG 9.1.0 <<>> intif.francophonie.org a

;; global options:  printcmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55861

;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:

;intif.francophonie.org.                IN      A

;; ANSWER SECTION:

intif.francophonie.org. 259200  IN      A       217.109.182.50

;; AUTHORITY SECTION:

francophonie.org.       259200  IN      NS      agecop.francophonie.org.

francophonie.org.       259200  IN      NS      ns0.unesco.org.
```

```
;; ADDITIONAL SECTION:

ns0.unesco.org.         172790  IN      A       193.242.192.106

agecop.francophonie.org. 259200 IN      A       194.199.57.2

;; Query time: 1587 msec

;; SERVER: 127.0.0.1#53(127.0.0.1)

;; WHEN: Sat Nov 23 16:17:42 2002

;; MSG SIZE  rcvd: 134

[danny@serveur danny]$
```

VII. Gestion des programmes du système

VII.1. Installation et suppression de nouveaux programmes

Installation d'applications avec RPM

- L'utilitaire **RPM** (=RedHat Package Manager) gère une base de données des applications déjà installés. Il permet d'installer (et de désinstaller) facilement les nouvelles applications qui sont disponibles sous forme d'une fichier "paquetage". De plus, pendant une mise à jour RPM conserve les fichiers de configuration déjà présents.

- Syntaxe des paquetages
  La syntaxe générale des paquetages à installer sur les machines à base de processeur Intel est nom.version.i386.rpm
  Pour connaitre toutes les (nombreuses) options, voir **man rpm**

- Pour utiliser la commande **rpm** de façon aisée et transparente, il est recommandé de passer par l'utilitaire graphique **KPackage**
  Mais le recours à la ligne de commande s'avère parfois indispensable, si on ne dispose pas de serveur X, ou si KPackage n'est pas installé !

- Principales options en ligne de commande

  - **rpm -q nomfichier.rpm** , pour obtenir de l'information
    Cette commande suppose la connaissance exacte du nom du paquetage (y compris respect de la casse). Sinon utiliser la commande rpm -qa | grep -i nom

    - rpm -q nomfichier.rpm donne le numéro de version du programme s'il est installé, sinon renvoie le message *"package ... is not installed"*

    - rpm -qa | less donne la liste des programmes rpm installés

    - rpm -qa | grep kernel pour chercher les programmes du noyau

    - rpm -ql kernel | less donne la liste de tous les fichiers inclus dans les paquetages désignés, en particulier les modules installés dans /lib/modules/...

- **rpm -i nomfichier.rpm**, commande générale d'installation
    - rpm -i cette commande, réservée à root, décompresse les programmes en les installant dans les bons répertoires.
    - les options v et h, facultatives, permettent de voir l'état d'avancement de l'installation.
    - rpm -ivh **-- nodeps** nomfichier.rpm, pour contourner le refus d'installer en raison de dépendances non satisfaites.
    - rpm -ivh **-- force** nomfichier.rpm, pour forcer l'installation en cas de conflit avec une version déjà installée
    - Exemple : pour installer les HOWTO en français, monter le cd-rom, aller dans /Mandrake/RPMS/, et passer la commande rpm -ivh howto-french-*
- **rpm -U nomfichier.rpm**, commande de mise à jour d'un paquetage déjà installé.
  L'ancienne version est d'abord retirée, tout en préservant les fichiers de configuration
- **rpm -e nomfichier.rpm**, pour désinstaller (e=extract) un programme.
  Cette commande controle les dépendances, et signale les autres programmes qui en ont besoin. Donc, attention à ne pas désinstaller des fichiers en dépendance.
- **rpm -V nomfichier.rpm**, cette commande compare les fichiers installés avec les fichiers d'origine du paquetage, pour vérifier que tous les fichiers d'un programme sont présents et pour connaitre ceux qui ont été modifiés depuis

VII.2. Gestion des paquetages

Sauvegarde et archivage avec tar

- **Généralités**

  La commande **tar** (=Type ARchive) est une ancienne commande Unix qui permet aisément d'archiver, c'est-à-dire de réaliser la sauvegarde d'un ensemble de fichiers en un seul fichier, que l'on peut également compresser. Certaines applications et des mises à jour (les noyaux Linux notamment) ne sont livrées que sous forme soit binaire, soit de

source à compiler, dans ce format (bien que les applications soient de plus en plus disponibles précompilées, prêtes à l'emploi, sous format **.rpm**)

- **Syntaxe**

tar options fichiers

fichiers :
désigne un ensemble de fichiers ou toute une arborescence précédée d'un chemin absolu (à partir de /) ou relatif. Il est recommandé d'indiquer un chemin absolu qui sera conservé dans l'archive et permettra ensuite un désarchivage correctement positionné (sinon il y a installation conformément au chemin relatif conservé, ce qui nécessiterait un exact positionnement dans le système de fichiers).

options :
Les 3 premières -c -x -t spécifient les 3 types d'actions de la commande

- **-x** e**x**traire le contenu d'une archive
- **-c** créer une nouvelle archive
- **-t** afficher seulement la lis**t**e du contenu de l'archive, sans l'extraire
- **-f fichier** indiquer le nom du **f**ichier archive
- **-v** mode ba**v**ard
- **-z** compresser ou décompresser en faisant appel à l'utilitaire **gzip**
- **-y** compresser ou décompresser avec l'utilitaire **bgzip2**
- **--help** aide
- **-B** pour éviter le **b**locage en utilisant un pipe

- **Utilisation et exemples**

Création
tar -cvf sauve.toto.tar /home/toto effectue la sauvegarde de tous les fichiers du répertoire /home/toto dans le fichier sauve.toto.tar placé dans le rép. courant
tar -cvf /tmp/sauve.toto.tar /home/toto idem, mais le fichier archive est placé dans le rép. /tmp
tar -c /home/toto > sauve.toto.tar variante de la commande précédente
tar -cvf sauve.toto.tar /home/toto
tar -cvzf sauve.toto.tar.gz /home/toto effectue une compression en plus

Listage

tar -tvf sauve.toto.tar pour connaitre l'arborescence regroupée dans le fichier archive, en particulier la place où sera installée son contenu lors du désarchivage.
L'utilitaire **mc**, avec sa fonction d'édition F3, permet d'effectuer le même listage de l'archive

Extraction

tar -xvf sauve.toto.tar exécute le désarchivage dans le répertoire courant.
si l'archive a été créée par tar -cvf sauve.toto.tar /home/toto, il faut se placer à la racine / pour restorer exactement le rép. perso de toto.
décompresse et désarchive
tar -xvfz sauve.tar.gz home/toto/tmp ne désarchive dans l'archive, que le rép. désigné

Compression avec gzip

- **La commande gzip**

Elle est utilisée pour compacter un fichier quelconque, et en particulier une archive tar.
Le décompactage se fait par la commande **gunzip**, ou de manière totalement équivalente par **gzip -d**.
Elle peut décompacter les fichies .gz, mais aussi les fichiers .z, .Z

- **Options**

o **-1 ...-9** fixe le niveau de compression

- **-d** décompresse
- **-c** écrit sur la sortie standard au lieu de remplacer le fichier d'origine (possibilité d'utiliser un tube)
- **-l** affiche des infos sur la dé/compression.
- **-r** dé/compresse tous les fichiers du rép. passé en argument.
- -**h** aide

- **Exemples**
  - **gzip backup.tar /home/toto** compresse *backup.tar* et le remplace par le fichier *backup.tar.gz*, d'une taille beaucoup plus réduite.
    Attention, le fichier d'origine est donc détruit !
  - **gzip -9 *.txt** compresse au maximum chaque fichier **.txt** séparément, et les renomme en ajoutant le suffixe **.gz**

- **Autre utilitaire**

  bzip2 admet la même syntaxe que gzip, mais compresse mieux avec un besoin accru de mémoire

VII.3. Utilisation de l'aide et manuels en ligne

Il existe plusieurs types d'aide en ligne :

- le manuel de référence
- les "HOWTO"
- les documents du "Linux Documentation Project", alias LDP

- les fichiers de documentation des paquetages

Le manuel

Le manuel en ligne est accessible par la commande **man**. Il est organisé en différentes sections :

Section	Intitulé
1	les commandes utilisateur
2	les appels système
3	les bibliothèques de programmation
4	les fichiers spéciaux
5	les formats de fichiers
6	les jeux
7	Divers
8	les commandes d'administration
9	le noyau

La syntaxe de la commande **man** est :

```
man {[}options{]} {[}section{]} commande
```

Par défaut, les sections sont parcourues dans l'ordre des numéros. Il est possible de préciser le numéro de section pour résoudre les problèmes de synonymes :

ex : **man kill** et **man 2 kill**

La variable d'environnement MANPATH donne le chemin de recherche des pages de manuel. Par exemple :

```
    MANPATH=/usr/man:/usr/local/man
```

Il existe une base de mots clés pour la recherche dans les pages de manuel. Cette base est construite par la commande **/usr/sbin/makewhatis**. Les commandes **apropos** et **whatis** permettent d'effectuer des recherches dans cette base de données.

Texinfo

C'est un projet de la FSF (Free Software Foundation) pour remplacer les pages de manuel classiques. La base de données d'informations est répartie et accessible par la commande **info**. Cette commande permet de visualiser en mode interactif des documents hypertexte. Le système est auto renseigné, il suffit de lancer info et de naviguer dans cet environnement pour avoir l'aide en ligne.

Les HOWTO

Les HOWTO sont des documents expliquant comment effectuer une tâche donnée comme : "Installer Linux sur le même disque que Windows NT". Ils sont disponibles dans plusieurs formats (texte, Postscript, HTML) et sont installés dans le répertoire /usr/doc/HOWTO. Ce sont les premiers documents à consulter lorsqu'on recherche une information pouvant s'exprimer par `` Comment faire pour ? ''. Il existe des traductions françaises de ces documents sur *http://www.freenix.fr/linux/HOWTO/*

Les documentations

A partir du répertoire /usr/doc, on trouve les documentations des paquetages installés. Le contenu est assez inégal et dépend du concepteur du paquetage. C'est néanmoins une ressource importante en terme de documentation.

Les guides du Linux Documentation Project (LDP) :

- "Installation and getting started guide"
- "The Linux kernel hackers' guide"
- "The Linux Kernel"
- "The Linux Programmers guide"
- "The Linux Kernel Module Programming Guide"
- "The Linux Network Administrators' Guide"
- "The Linux Users Guide"
- "The Linux System Administrators' Guide"

Comme pour les HOWTO, certains de ces guides sont traduits en français.

VIII. Installation de GNU/Linux

VIII.1. Les pré-requis pour une installation réussie de GNU/Linux.

Installation du système de base

Maintenant que vous connaissez tout des grands principes de Linux (et de la plupart des systèmes Unix), vous devez brûler d'impatience de l'installer et de commencer à exploiter toutes ses possibilités. Cependant, il faut ne pas tomber dans le travers de ceux qui veulent impérativement utiliser une fonctionnalité sous prétexte qu'elle est disponible. Il se peut que vous ayez envie de les essayer, parce que vous en avez été privé depuis longtemps. Ce comportement est tout à fait normal, mais vous allez devoir refréner vos envies, pour deux raisons. La première est qu'il va falloir installer Linux au préalable, et la deuxième est qu'il ne faut pas faire un sac de nœuds avec vos données. Vous pourrez expérimenter à loisir, mais il faudra rester sérieux dès que vous toucherez au système. Comprenez bien que l'installation et la configuration de Linux doivent se faire pas à pas. Griller les étapes peut se révéler être une erreur, car cela rendrait les choses plus compliquées qu'elles ne le sont. La configuration minimale supportée par Linux est un 386 avec au minimum 12 Mo de mémoire vive.

Il va de soi qu'avec un tel foudre de guerre, vous ne pourrez pas aller bien loin, une configuration raisonnable serait plutôt un 486DX2 66MHz avec au moins 32Mo de mémoire. La quantité de mémoire est de loin le facteur le plus important, et si vous voulez utiliser l'environnement graphique X11, il faut au minimum compter sur 64 Mo de mémoire vive. Vous pourrez donc parfaitement transformer un vieux 486 ou pentium en routeur ou en Firewall avec partage de connexion à Internet par exemple.

Vous n'aurez donc certainement aucun problème avec les ordinateurs puissants qui se vendent actuellement.

VIII.2. Informations et paramètres des périphériques à récupérer avant l'installation

Avant de commencer quoi que ce soit, vous devriez récupérer le maximum de données concernant votre matériel. En effet, ces informations peuvent être particulièrement utiles pour diagnostiquer la cause des éventuels problèmes de configuration du matériel que vous pourriez rencontrer. Vous aurez aussi peut être à spécifier certains paramètres matériels lors de l'installation et, dans le pire des cas, lors du premier démarrage de Linux. Les informations dont vous aurez certainement besoin sont les suivantes :

- type de processeur, avec sa marque et éventuellement sa vitesse ;

• type de carte mère, avec sa marque et impérativement son chipset ;
• type de branchement de la souris (série, PS/2, interface bus propriétaire) ;
• nombre de ports série, ainsi que leurs paramètres (ports d'entrée/sortie, lignes d'interruption) ;
• nombre de ports parallèle, ainsi que leurs paramètres (ports d'entrée/sortie, lignes d'interruption) ;
• si vous disposez d'un contrôleur SCSI, marque et modèle de ce contrôleur (modèle du chipset de la carte SCSI) ;
• sinon, nom du contrôleur de disque IDE (regardez sur le chipset de la carte mère !) ;
• types de disques durs (IDE, IDE UltraDMA 33, 66 ou 100, SCSI), ainsi que leurs taille et leur
position dans le système (contrôleurs auxquel ils sont connectés, numéros d'unités logiques SCSI) ;
• type de lecteur de CD-ROM (IDE, ATAPI, SCSI, connecté sur carte son) et le type de leur branchement s'ils sont externes (port parallèle, SCSI ou USB) ;
• nom, modèle et marque de la carte graphique, type de chipset utilisé par cette carte et taille de sa mémoire vidéo ;
• nom, modèle et marque de l'écran, avec ses fréquences de balayage horizontales et verticales, sa bande passante et les fréquences recommandées par le constructeur pour les différentes résolutions ;
• type de carte son (ISA, PCI, PnP) ainsi que le nom du chipset utilisé par cette carte ;
• type de carte réseau (ISA, PCI, PnP) ainsi que le nom de son chipset ;
• type de modem (interne, externe) et, si le modem est interne, sa nature (modem complet ou émulation).

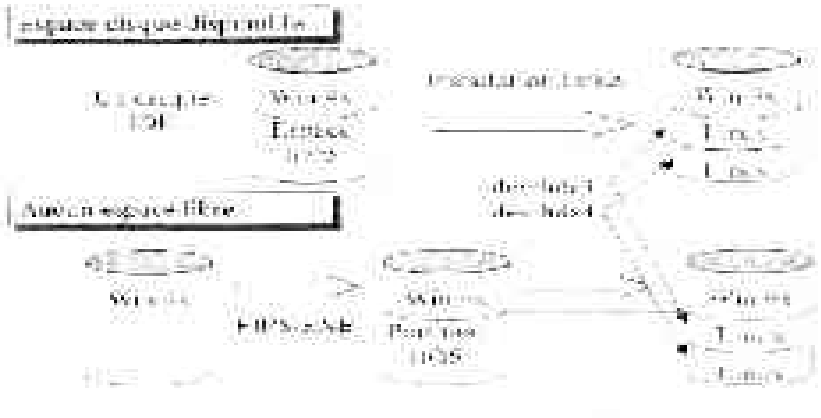
Vous pouvez obtenir ces informations en consultant la documentation fournie avec votre ordinateur, les sites Web des constructeurs des différents composants, les informations fournies par le programme de configuration du BIOS ou affichées directement par le BIOS au démarrage de la machine, ou tout simplement en ouvrant le boîtier de la machine et en regardant ce qu'il y a à l'intérieur. Vous pouvez également récupérer les informations indiquées dans la configuration de MS Windows si celui-ci est installé.
Il se peut que vous n'ayez pas besoin de ces informations et que l'installation se passe sans problème.

Cependant, si d'aventure Linux exige que vous les connaissiez, mieux vaut pouvoir lui répondre.

Note : Bien que cela soit assez rare, certains composants ou périphériques additionnels peuvent ne pas être gérés par Linux, faute d'avoir un gestionnaire de périphérique adéquat (cet état de fait est en général dû à une rétention d'information de la part des fabricants, qui empêchent ainsi les développeurs de Linux d'écrire les gestionnaires de périphériques dans de bonnes conditions). C'est en particulier le cas de nombre de périphériques bas de gamme conçus pour ne fonctionner qu'avec MS Windows, ou de quelques périphériques exotiques. Les Winmodems ont une triste réputation à ce sujet (il s'agit de modems incomplets, dont une partie du traitement de signal est reporté dans un logiciel spécifique à Windows), ainsi que des imprimantes dites « GDI », qui ne comprennent que les commandes graphiques de Windows. Vous pourrez également avoir des problèmes avec des cartes de numérisation vidéo analogiques ainsi que certaines cartes de décompression MPEG. Certaines marques se distinguent par le fait qu'elles refusent obstinément de développer des gestionnaires de périphérique pour leur matériel ou, pire encore, de fournir les informations nécessaires aux développeurs de Linux pour écrire ces gestionnaires sous licence libre. Mais il est toujours recommandé, de manière générale, de bien se renseigner auprès des vendeurs et des groupes de discussion avant tout achat de matériel...
Vous pourrez trouver une liste (non exhaustive) de matériel testé sous Linux dans la liste de matériel compatible Linux (http://www.linuxdoc.org/HOWTO/Hardware-HOWTO/). La Linux Hardware Database (http://lhd.datapower.com/) vous permettra également d'effectuer une recherche sur n'importe quel type de matériel et sur

n’importe quel modèle très facilement. Il existe d’autres sites réalisant périodiquement des tests sur le matériel récent, comme par exemple LinuxHardware (http://www.linuxhardware.org).

VIII.3. Notion sur la gestion des disques pour la partition



Partitionnement du disque

Une « partition » est, comme son nom l’indique, une partie d’un disque dur. Les partitions permettent de diviser l’espace de stockage des disques durs en zones indépendantes de taille restreinte, et dans lesquelles différents systèmes ou différentes données peuvent être stockées de manière totalement indépendante.

Définir les partitions

L’installation du système Linux requiert au minimum deux partitions, une partition de swap et une partition « / » d’une taille dépendante du nombre de paquetages que l’utilisateur souhaite installer. Une taille minimum de 300 Mo paraît raisonnable.

Remarque :
La taille de la partition swap dépend de la nature, gestion ou calcul, de la taille des applications que vous allez exécuter sur le système Linux. Il est souvent conseillé de disposer de deux fois la taille de la RAM. Il est possible de créer plusieurs disques de swap.

Dans le cas d’une installation personnalisée, il est préférable de créer suffisamment de partitions pour y installer des systèmes de fichiers pour Linux et pour vos applications spécifiques.

Remarque : La partition « / » est obligatoire.

Partition	Taille	Description
/boot	De 10 à 20 Mo	Elle contient les fichiers de « boot LILO » et le noyau Linux et doit résider en deça du cylindre 1023 de votre disque

/	De 50 à 100 Mo	Elle contient le système de fichiers principal de Linux
/usr	Au moins 300 Mo. Elle peut atteindre 1Go et peut être plus.	Elle contient les commandes et les services pour les utilisateurs. C'est de loin la plus volumineuse.
/var	Elle est importante pour un serveur. On peut l'estimer à environ 50 Mo	Elle est le pendant de « /usr ». Elle contient les requêtes adressées aux services des utilisateurs et les fichiers historiques (fichiers *log*)
/tmp	Sa taille dépend aussi du nombre de services installés. On peut l'estimer à environ 50 Mo	Elle contient les fichiers temporaires créés par des commandes et des services.
/home	Sa taille dépend du nombre d'utilisateurs et de leur consommation d'espace disque.	Elle contient les répertoires de connexion des utilisateurs du système.
/usr/src	Sa taille est d'au minimum 30 Mo	Elle contient les sources du noyau et des paquetages. Elle est indispensable pour regénérer le noyau personnalisé.

Libérer de l'espace disque

Si la totalité de l'espace disque est occupé par le système Windows, ou si la partition Windows ne laisse pas assez d'espace disque libre, le CD-ROM d'installation de Linux propose une commande \dosutils\fips.exe qui permet de modifier la taille de la partition allouée à Windowsafin de libérer de l'espace pour Linux.

Utilisation de fips

fips (abréviation de l'anglais « First Interactive Partition Splitter ») est un utilitaire similaire à **parte**d, à ceci près qu'il fonctionne sous DOS et qu'il ne permet que de réduire la limite supérieure d'une partition FAT. De plus, cet utilitaire est incapable de réorganiser le système de fichiers à l'issue de la réduction de la taille de la partition. Il est donc nécessaire de défragmenter le système de fichiers avant d'utiliser ce programme, afin de placer toutes ses données au début de la partition.

Note : Vérifiez bien les options du défragmenteur de système de fichiers que vous utilisez : quelques outils consolident bien l'espace libre mais placent certains fichiers à la fin de la partition FAT pour laisser plus de place aux fichiers les plus utilisés au début du disque, afin d'optimiser le débit de données sur ces fichiers (c'est notamment le cas avec le défragmenteur de Norton).

Il faut impérativement désactiver ce type d'option avant de réduire la partition, faute de quoi vous perdriez définitivement les fichiers qui se trouvent à la fin de la partition et votre FAT serait dans un état incohérent.

Une fois la défragmentation réalisée, **fips** peut être utilisé pour réduire la taille de la partition FAT. La plupart des distributions de Linux fournissent des utilitaires DOS sur leur CD-ROM d'installation, généralement dans le répertoire dosutils. C'est là que vous pourrez sans doute trouver **fip**s. Attention,vous devrez impérativement utiliser la version 2.0 de **fips** pour manipuler les FAT32 et FAT32X.

La réduction de la taille d'une partition se fait en modifiant la variable qui contient la taille de la partition dans la table des

partitions. Pour cela, vous devrez simplement lancer **fip**s. Celui-ci vous présentera alors la liste des disques durs installés sur votre système, et vous devrez lui indiquer le disque sur lequel la partition à réduire se trouve. Il vous demandera ensuite la partition que vous désirez réduire, puis le cylindre auquel cette partition devra se terminer. Lorsque vous aurez déterminé la nouvelle taille de cette partition, vous devrez presser la touche 'c' pour poursuivre. **fips** vous demandera alors confirmation avant d'écrire sur disque les nouvelles informations de partition. Si vous êtes sûr de vous, vous pouvez répondre par l'affirmative en pressant la touche 'y'.

Utilisation de fdisk

Le partitionnement en soi peut se faire soit directement à l'aide du **fdisk** de Linux, soit par l'intermédiaire du programme d'installation de la distribution correspondante. Il est recommandé d'utiliser ce programme d'installation, qui vous guidera et vous indiquera comment réaliser cette opération. Si toutefois vous désirez utiliser **fdis**k, il vaut mieux faire attention à ce que vous faites.
Pour lancer **fdis**k, il suffit de taper la commande suivante en ligne de commande :

```
fdisk disque
```

où `disque` est le fichier spécial de périphérique représentant le disque que vous désirez partitionner.
Si vous voulez partitionner le disque maître du premier contrôleur IDE, vous devrez donc taper :

```
fdisk /dev/hda
```

Si vous ne spécifiez aucun disque en paramètre à fdisk, il prendra par défaut le disque `/dev/sda`, ou `/dev/hda` si aucun disque SCSI n'est installé. **fdisk** est un programme très peu interactif. Il attend que vous lui communiquiez les commandes à exécuter en tapant sur une lettre. Les différentes commandes possibles peuvent être affichées avec la commande 'm'.

Lorsque vous créez une partition, vous devez utiliser la commande 'n', puis indiquer son type avec les commandes 'p' (pour « primary ») pour une partition primaire ou 'e' (pour « extended ») pour une partition étendue. Vous devrez ensuite donner son numéro dans la table des partitions, puis indiquer le début et la fin de la partition. Par défaut, l'unité utilisée par **fdisk** est le cylindre. Il est recommandé de conserver cette unité, surtout si l'on utilise un système qui ne sait manipuler que les cylindres.

Toutefois, on peut changer cette unité grâce à la commande 'u' et utiliser le secteur comme unité.

Si vous avez créé une partition étendue, celle-ci sera utilisée pour y stocker des partitions logiques.

Pour pouvoir les créer, il faut encore utiliser la commande 'n', et choisir le type de partition logique avec la commande 'l' (pour « logical »). Les partitions logiques sont numérotées avec les nombres 5 et suivants. La création des partitions logiques se fait exactement de la même manière que les partitions primaires, en spécifiant leur début et leur fin, soit en cylindres, soit en secteurs selon l'unité courante.

Une fois les partitions créées, vous pouvez spécifier leur type à l'aide de la commande 't' (pour « type »). Cette commande demande successivement le numéro de la partition à modifier et la valeur de son identificateur en hexadécimal. Rappelons que les identificateurs à utiliser pour Linux sont

83 pour les partitions de systèmes de fichiers Linux, et 82 pour les partitions de swap. La liste des valeurs admissibles peut être obtenue à l'aide de la commande 'l'. Par défaut, le **fdisk** de Linux crée des partitions Linux natives, de code 83.

Lorsque vous aurez complètement défini vos partitions, il ne vous restera plus qu'à activer la partition qui contiendra le gestionnaire d'amorçage. La sélection de la partition active se fait avec la commande 'a' de **fdis**k. C'est donc sur cette partition que le chargeur du MBR ira chercher le gestionnaire d'amorçage du système à lancer.

Note : Théoriquement, il est tout à fait possible d'installer le gestionnaire d'amorçage d'un système directement sur le MBR, mais procéder de cette manière est très déconseillé. En effet, certains systèmes d'exploitation (notamment tous les systèmes de Microsoft) écrasent systématiquement le MBR lorsqu'ils s'installent, détruisant ainsi le chargeur d'un autre système qui y serait éventuellement installé. Cela implique que si l'on désire installer un gestionnaire d'amorçage autre que celui des systèmes Microsoft sur le MBR, il faut le faire après l'installation de ces systèmes.

En pratique, cela veut dire que dans ce cas, on doit installer Linux après Windows ou le DOS.

Notez qu'il n'est toutefois pas toujours faisable d'installer le gestionnaire d'amorçage sur le secteur de boot de la partition de son système, en particulier si cette partition ne se trouve pas sur le premier disque dur de la machine. En effet, la plupart des BIOS sont incapables d'utiliser les MBR des autre disques durs. Dans ce cas, on peut soit créer une partition de démarrage de petite taille (quelques méga-octets, un cylindre au maximum) au début du disque et sur laquelle on installera le gestionnaire d'amorçage et éventuellement quelques outils de réparation en cas de coup dur, soit installer le gestionnaire d'amorçage directement sur le MBR du premier disque dur. Dans ce cas, on devra faire particulièrement attention à l'ordre d'installation des systèmes d'exploitation. De manière générale, il faut toujours installer les systèmes Microsoft en premier (respectivement dans l'ordre suivant si l'on veut éviter les problèmes : DOS, Windows 9x/Millenium et Windows NT4/2000/XP). Nous verrons plus loin comment installer le gestionnaire d'amorçage de Linux et faire une configuration multiboot avec les principaux autres systèmes d'exploitation existants.

Amorçage du système et configuration multiboot

Lorsque vous aurez installé votre système de base, vous devrez faire en sorte qu'il puisse démarrer.

Pour cela, il existe plusieurs possibilités, les principales étant les deux suivantes :

- soit vous démarrez à partir d'une disquette ou d'un CD d'amorçage ;
- soit vous utiliser un gestionnaire d'amorçage.

Il va de soi que c'est la deuxième solution qui est recommandée. Cependant, la première solution pourra être utile si d'aventure votre MBR se trouvait être écrasé ou endommagé. Seule l'utilisation des gestionnaires d'amorçage de Linux et de Windows NT/2000 et XP sera décrite ici.

Le gestionnaire d'amorçage le plus utilisé sous Linux se nomme « LILO » (pour « LInux LOader »).

LILO permet de démarrer un grand nombre de systèmes, dont DOS, Windows 95/98/Millenium, Windows NT/2000/XP et OS/2. Linux dispose également d'un autre gestionnaire d'amorçage également très performant : le « GRUB » (abréviation de l'anglais « GRand Unified Bootloader »). Vous êtes libre de choisir celui que vous voulez. Le GRUB est quelque peu plus moderne que LILO, mais le choix du gestionnaire d'amorçage est ici une question de goût. Windows NT, Windows 2000 et Windows XP disposent également d'un gestionnaire d'amorçage nommé « NTLDR », capable de démarrer d'autres systèmes d'exploitation. Vous aurez donc également la possibilité d'utiliser ce gestionnaire à la place de LILO ou du GRUB si un de ces systèmes est installé sur votre machine.

Quel que soit le gestionnaire d'amorçage que vous désirez utiliser, il vous faudra activer la partition sur laquelle il est installé. Cette opération peut être réalisée à l'aide de l'utilitaire **fdisk**. Cela permettra au bootstrap loader de sélectionner le secteur de boot de cette partition et de lancer le gestionnaire d'amorçage qui y est installé. Bien entendu, cela suppose que ce gestionnaire soit installé sur cette partition, ce qui est normalement le cas (rappelons qu'il n'est en général pas conseillé d'installer le gestionnaire d'amorçage directement sur le MBR du disque dur).

Réalisation d'un multiboot avec LILO

LILO est un gestionnaire d'amorçage extrêmement performant, puisqu'il permet de démarrer Linux comme tout autre système d'exploitation très simplement, en donnant le nom du système à lancer lors de l'amorçage. Il est bien entendu possible de lancer un système par défaut, et de donner un temps d'attente avant de choisir cette option si l'utilisateur n'a rien saisi.

LILO est constitué de deux parties. La première partie peut s'installer sur le secteur d'amorçage principal du disque dur ou sur le secteur de boot de n'importe quelle partition. Comme on l'a déjà indiqué plus haut, il est fortement recommandé d'installer cette partie de LILO sur le secteur de boot de la partition Linux, afin d'éviter qu'elle ne soit écrasée par le DOS ou par une quelconque version de Windows installée ultérieurement. La deuxième partie de LILO est enregistrée directement dans la partition Linux. Elle contient les informations nécessaires pour pouvoir charger les différents systèmes d'exploitation gérés. Bien entendu, la première partie est capable de retrouver directement la deuxième sur le disque dur car, lors de l'amorçage, les systèmes de fichiers de Linux ne sont pas encore chargés.

LILO utilise le fichier de configuration `/etc/lilo.conf` pour y retrouver tous ses paramètres de configuration. Ce fichier contient la description des différents systèmes d'exploitation que LILO doit proposer au démarrage. Vous pourrez consulter ce fichier avec un des nombreux éditeurs de fichiers texte présents sur toute installation de Linux. Toutefois, si vous installez Linux pour la première fois, il est possible que vous n'en connaissiez aucun et que vous soyez un peu perdu. Cela est normal, et dans ce cas je vous recommande de vous familiariser un peu avec le système et l'environnement utilisateur avant de vous lancer dans l'édition de ce fichier. Il existe bon nombre d'éditeurs graphiques ou en mode texte et il est hors de question de tous les décrire ici. Toutefois, toutes les distributions Linux installent un éditeur historique, j'ai nommé l'affreux « vi ». Cet éditeur n'est pas du tout convivial pour les nouveaux utilisateurs, mais il dépannera toujours quand tous les autres seront inutilisables ou inaccessibles. En fait, on finit même par l'apprécier à l'usage...

Quoi qu'il en soit, les options les plus importantes du fichier `/etc/lilo.conf` sont les suivantes :

• l'option `boot`, qui permet d'indiquer sur quel secteur d'amorçage LILO doit s'installer. Cette option suit la syntaxe suivante :

```
boot = destination
```

où `destination` est le nom d'un fichier spécial de périphérique sur lequel LILO va s'installer. Ce nom peut identifier un disque dur (comme par exemple `/dev/hda`), auquel cas LILO va s'installer sur le MBR de ce disque (c'est-à-dire sur le MBR du premier disque du premier contrôleur de disques IDE dans notre exemple), ou une partition (comme `/dev/hda2`). Dans ce cas, LILO s'installe sur le secteur de boot de cette partition (la deuxième

partition du premier disque dur IDE dans notre exemple). Rappelons qu'il est recommandé d'installer LILO sur le secteur de boot de la partition racine de Linux;
• l'option `read-only` permet de demander au noyau de monter la partition root en lecture seule lors du démarrage. Cette option est nécessaire pour que les scripts de démarrage du système puissent effectuer les vérifications du système de fichier de cette partition si nécessaire. La partition sera remontée en lecture et en écriture une fois ces vérifications réalisées ;
• l'option `prompt`, qui permet à LILO de demander le système à lancer à chaque démarrage. Cette option force donc l'apparition du message d'invite de LILO au démarrage : `LILO boot:` auquel on pourra répondre en tapant le nom de la configuration à lancer;
• l'option `timeout`, qui permet de fixer un délai au delà duquel LILO lancera la première configuration définie dans le fichier `lilo.conf`. La syntaxe de cette option est la suivante :

```
timeout = dixièmes
```

où `dixièmes` est le nombre de dixièmes de secondes à attendre avant le lancement du système;
• l'option `keytable`, qui donne la possibilité de spécifier un fichier de traduction des codes de caractères envoyés par le BIOS (qui suppose généralement que le clavier utilise la disposition d'un clavier américain) en les codes de caractères qui seraient envoyés par un BIOS localisé. Cette option permet donc de redéfinir la disposition des touches du clavier pour prendre en compte les claviers non-américains. La syntaxe de cette option est la suivante :

```
keytable = fichier
```

où `fichier` est le chemin sur un fichier de traduction de clavier pour LILO. Un tel fichier peut être généré par le script **keytab-lilo.pl** à l'aide des fichiers de définition des claviers de Linux (généralement installés dans le répertoire `/usr/lib/kbd/keymaps`). La ligne de commande à utiliser pour ce script est la suivante :

```
keytab-lilo.pl us local > fichier
```

où `us` est le nom du fichier de définition de clavier Linux pour la disposition utilisée par le BIOS (donc, effectivement, la disposition d'un calvier américain en général), `local` est le nom du fichier de définition de clavier pour la disposition du clavier réellement utilisé, et `fichier` est le nom du fichier de traduction des codes à générer. Par exemple, la création de ce fichier pour le clavier français se ferait avec la commande suivante:

```
keytab-lil.pl us fr-latin1 > /boot/fr-latin1.klt
```

Remarquez que cette technique de traduction de clavier souffre d'un inconvénient majeur, puisque les combinaisons de touches qui ne sont pas valides dans la disposition américaine des claviers ne peuvent pas être converties. Si une de ces touches doit être utilisée, il faut abandonner l'idée d'utiliser l'option `keytable`.

La suite du fichier `lilo.conf` décrit les différentes configurations que LILO peut lancer. Les sections de configuration permettant de charger Linux ont le format suivant :

```
image = noyau
root = root_device
label = nom
```

où noyau est le chemin complet sur le noyau de Linux à charger, root_device est le nom complet du fichier spécial de périphérique contenant le système de fichier racine et nom est le nom de la configuration tel qu'il devra être saisi à l'invite de LILO. L'exemple donné ci-dessous permet de charger le noyau /boot/vmlinuz en utilisant la partition /dev/hda2 comme partition racine:

```
image = /boot/vmlinuz
root = /dev/hda2
label = linux
```

Si vous désirez créer une section de configuration permettant de lancer un autre système d'exploitation que Linux (DOS ou Windows par exemple), vous pouvez utiliser la possibilité de passer le relai au chargeur de ces systèmes, qu'il s'agisse d'un simple secteur de boot ou d'un gestionnaire d'amorçage complet. Cela se fait avec la syntaxe suivante :

```
other = partition
table = disque
loader = relai
label = nom
```

où partition est la partition sur laquelle le secteur de boot de l'autre système est installé, disque est le disque dur contenant la table des partitions utilisée par ce système, relai est le nom d'un chargeur spécial permettant de simplement passer la main au chargeur du système, et nom est le nom de la configuration. Le chargeur à utiliser pour demander à LILO de passer le relai au chargeur de l'autre système d'exploitation est le chargeur contenu dans le fichier chain.b de LILO. Ce fichier se trouve généralement dans le répertoire /boot/, aussi doit-on spécifier /boot/chain.b pour le champ relai.

Note : Prenez garde au fait que Windows NT/Windows 2000/XP installe NTLDR dans la première partition à laquelle il sait accéder en général. Donc, si un DOS ou Windows 95, Windows 98 ou Millenium est installé en premier, il installera NTLDR dans la partition de ces systèmes. Dans ce cas, la configuration permettant de lancer le DOS ou le Windows 95, Windows 98 ou Millenium qui se trouve sur cette partition risque fort de lancer NTLDR qui proposera, à son tour, de lancer les différents systèmes d'exploitation Microsoft installés.
Cela peut être relativement gênant et peut être corrigé en déplaçant NTLDR sur la partition de Windows NT/Windows 2000/XP et en reconstruisant les secteurs de boot des différentes partition pour que leur chargeurs s'occupent de leurs systèmes respectifs, mais il s'agit là d'une opération extrêmement technique d'une part, et qui ne concerne absolument pas Linux d'autre part. Cela ne sera donc pas décrit dans ce document. Il existe toutefois des documents sur Internet qui décrivent la manière de procéder et je vous invite à vous y référer (avec une prudence extrême cependant).

L'exemple donné ci-dessous permet de donner la possibilité de charger Linux ou Windows NT, en lançant Linux par défaut au bout de 10 secondes. Windows NT est installé sur la troisième partition, et Linux utilise la deuxième et la quatrième partition respectivement pour y stocker sa partition racine et la partition des répertoires personnels des utilisateurs. LILO est ici installé sur la partition racine de Linux :

```
# Exemple de fichier de configuration /etc/lilo.conf :
# Options générales :
boot = /dev/hda2
read-only
prompt
timeout=100
keytable = /boot/fr-latin1.klt
# Première configuration (Linux) :
image = /dev/hda2
```

```
root = /dev/hda2
label = linux
# Deuxième configuration (NT) :
other = /dev/hda3
table = /dev/hda
loader = /boot/chain.b
label = NT
```

L'installation de LILO est très simple une fois que l'on a écrit le fichier `lilo.conf`. En effet, il suffit tout simplement de taper la commande suivante:

```
lilo [-L]
```

L'option `-L` permet de demander à LILO d'utiliser le mode d'adressage LBA pour accéder au disque dur pendant la phase d'amorçage. Cette option est nécessaire si vous disposez d'un grand disque dur et que certaines partitions disposant de systèmes à lancer sont situées au delà du cylindre 1024. Il est recommandé de l'utiliser systématiquement étant donné les tailles des disques durs actuels.

Note : Comprenez bien que si votre BIOS est incapable d'utiliser le mode LBA ou le si bootstrap loader est incapable d'utiliser ce mode, cette option ne vous sera d'aucune utilité. En effet, dans ce cas, le bootstrap loader ne parviendrait même pas à charger le secteur de boot de la partition Linux. C'est pour cette raison qu'il a été recommandé de placer la partition du système principal en deçà de cette limite des 1024 cylindres. Cette limitation est donc bien une limitation du BIOS, mais vous ne devriez plus rencontrer ce genre de problème que sur de vieilles machines sur lesquelles un nouveau disque dur de grande capacité a été installé.

Si lilo signale une erreur, il vaut mieux ne pas insister et corriger le fichier `lilo.conf`.

Lorsque la machine démarre, LILO affiche son invite de démarrage : `LILO boot:`

Il attend ici que vous indiquiez le nom du système que vous désirez démarrer. Vous devez ici taper le nom du système à charger et valider : `LILO`

```
boot:linux
```

Si vous ne tapez rien, et que vous avez donné un délai d'attente dans le fichier de configuration de LILO, la première configuration sera lancée automatiquement après ce délai.

LILO permet de spécifier des paramètres de démarrage complémentaire pour Linux à la suite du nom de la configuration qui permet de le lancer. Ces paramètres servent principalement à renseigner le noyau sur la configuration matérielle (en particulier les ports d'entrée/sortie et les lignes d'interruption des périphériques non Plug and Play), pour le cas où il ne parviendrait pas à les déterminer automatiquement.

L'un des paramètres les plus intéressants est sans doute `mem`, qui permet d'indiquer au noyau la taille de la mémoire vive dont dispose l'ordinateur. Ce paramètre peut être nécessaire si vous disposez de plus de 64 Mo de mémoire, parce que les fonctions du BIOS ne permettent pas d'indiquer les tailles de mémoire plus grandes (la plupart des BIOS récents n'ont toutefois plus ce problème). Par exemple, si votre ordinateur dispose de 256 Mo de mémoire, vous devrez taper la ligne de paramètres suivante au démarrage : `LILO boot:linux mem=256M`

Bien entendu, il est possible d'enregistrer ces paramètres dans le fichier de configuration de LILO afin de ne pas avoir à les saisir à chaque démarrage. Pour cela, il suffit d'indiquer le paramètre de démarrage du noyau dans une ligne `append` de la section de configuration de Linux :

```
append="paramètre"
```

Ainsi, la section de configuration de Linux du fichier `lilo.conf` exemple donné ci-dessus pourrait être remplacée par celle-ci sur une machine disposant de 256 Mo de mémoire :

```
# Première configuration (Linux) :
image = /dev/hda2
root = /dev/hda2
label = linux
append="mem=256M"
```

La liste des paramètres que l'on peut fournir au noyau est relativement grande et ne sera pas décrite ici. Les plus utiles seront présentés en temps et en heure, notamment dans le chapitre décrivant la configuration du système.

Réalisation d'un multiboot avec le GRUB

Le « GRUB » (abréviation de l'anglais « GRand Unified Bootloader ») est le gestionnaire d'amorçage développé par la Free Software Foundation pour amorcer le noyau Hurd du projet GNU. Il est capable de faire démarrer tous les systèmes utilisant un protocole de démarrage standard initialement défini pour le Hurd. Bien entendu, il sait aussi amorcer les systèmes Linux, qui n'utilisent pas ce protocole, ainsi que la plupart des autres systèmes d'exploitation en passant le relai à leurs propres gestionnaires d'amorçage.

En fait, le GRUB fournit la possibilité de contrôler totalement l'amorçage de son système par l'intermédiaire d'un interpréteur de commandes simplifié. Il est possible, par l'intermédiaire de cet interpréteur de commandes, d'effectuer nombre d'opérations dans le but de charger un noyau de système d'exploitation et de l'amorcer. Bien entendu, ces commandes peuvent être écrites dans un fichier de configuration afin d'automatiser le processus de chargement.

Le GRUB est normalement installé dans le répertoire `/boot/grub/`. Ce répertoire contient, outre les fichiers binaires du GRUB lui-même, son fichier de configuration. Ce fichier se nomme normalement `menu.lst`, en raison du fait qu'il permet de définir les différentes configurations correspondantes aux systèmes d'exploitation à charger et qui apparaîtront dans le menu de démarrage lors de l'amorçage de la machine.

Contrairement à LILO, qui enregistre l'emplacement des fichiers des différents noyaux à charger dans une liste de blocs du disque dur, le GRUB sait interpréter les systèmes de fichiers classiques de Linux.

En particulier, il est capable de retrouver son fichier de configuration et les fichiers images des noyaux Linux dans les systèmes de fichiers FAT, EXT2/EXT3 et ReiserFS. Cette particularité fait qu'il n'est pas nécessaire, lorsqu'on modifie le fichier de configuration `menu.lst`, de réinstaller le GRUB.

Tout comme le fichier de configuration de LILO, le fichier `menu.lst` se compose d'une partie contenant les options globales et d'une ou plusieurs parties contenant la description des différents systèmes d'exploitation à proposer au démarrage. Les options générales les plus utiles sont les suivantes :

• l'option `default`, qui permet de spécifier la configuration par défaut à charger. Cette option doit être suivi du numéro de cette configuration. Les configurations sont numérotées à partir de 0, dans leur ordre d'apparition dans le fichier de configuration;

• l'option `timeout`, qui permet de spécifier le délai d'attente avant que la configuration par défaut spécifiée par l'option `default` ne soit lancée.

Les configurations spécifiques aux systèmes d'exploitation suivent la syntaxe suivante :

```
title titre
root partition
kernel noyau options
```

où `titre` est le titre de la configuration tel qu'il doit apparaître dans le menu de démarrage du GRUB, `partition` est la partition dans laquelle se trouve le noyau à charger, et `noyau` est le chemin sur le fichier image de ce noyau dans cette partition. Attention, ce chemin est défini dans la partition elle-même et peut donc être différent du chemin utilisé sous Linux. En effet, il faut définir ce chemin par rapport au point de montage de la partition, faute de quoi le GRUB ne retrouverait pas le fichier image du noyau à charger.

Comme le GRUB n'est pas un chargeur spécifique à Linux mais a été écrit au contraire avec comme principal objectif une généricité absolue, la manière de spécifier la partition dans laquelle le noyau se trouve utilise une syntaxe différente de celle utilisée sous Linux. Cette syntaxe, propre au GRUB donc, est la suivante :

```
(hdn,m)
```

où `n` est le numéro du disque dans l'ordre énuméré par le BIOS de la machine et `m` est le numéro de la partition. Ce dernier numéro est facultatif (ainsi que la virgule qui le précède), ce qui permet de référencer un disque complet et non une partition. La numérotation des disques et des partitions commence toujours à 0 dans le GRUB, ce qui fait que la première partition du premier disque est référencée par `(hd0,0)`, la deuxième partition du premier disque par `(hd0,1)`, la première partition du deuxième disque par `(hd1,0)`, etc.

Tout comme avec LILO, il est possible de spécifier des options de démarrage qui devront êtres fournies au noyau. Ces options devront être spécifiées immédiatement après le nom de l'image du noyau.

Comme vous pouvez le constater, la définition d'une configuration de démarrage pour un système Linux est très simple, puisqu'il suffit quasiment de donner la ligne de commande pour lancer ce noyau ! Par exemple, pour charger le noyau `/boot/vmlinuz` d'un système situé sur la deuxième partition du premier disque, la configuration suivante doit être définie :

```
title Linux
root (hd0,1)
kernel /boot/vmlinuz mem=256M
```

Cet exemple présente également comment spécifier la taille de la mémoire disponible dans la machine (cela n'est normalement pas nécessaire avec les BIOS récents et avec le GRUB).

Bien entendu, le GRUB est capable de charger le secteur de boot d'une partition afin de passer le relai au gestionnaire d'amorçage d'un autre système d'exploitation. Pour cela, il faut utiliser la commande **chainloade**r, plutôt que la commande **kerne**l, dans la description de la configuration de démarrage de ce système. La forme générale d'une configuration de ce type est donc la suivante :

```
title titre
root partition
chainloader +1
```

Le `+1` qui suit la commande **chainloader** indique au GRUB de charger le premier secteur de la partition indiquée par la commande **root** et d'exécuter

le gestionnaire de boot normalement stocké dans ce secteur. Comme pour les configurations Linux, la syntaxe utilisée pour spécifier la partition où ce secteur est situé est la syntaxe du GRUB et non celle utilisée sous Linux.

Le fichier de configuration d'exemple suivant correspond au fichier de configuration de LILO vu dans la section précédente. Il permet de démarrer un Linux installé sur la deuxième partition ou un Windows NT installé sur la troisième partition du premier disque dur de la machine :

```
# Exemple de fichier de configuration /boot/grub/menu.lst :
# Options générales :
default 0
timeout 10
# Première configuration (Linux) :
title Linux
root (hd0,1)
kernel /boot/vmlinuz root=/dev/hda2 mem=256M
# Deuxième configuration (NT) :
title NT
root (hd0,2)
chainloader +1
```

L'installation du GRUB sur une nouvelle machine ne pose quant à elle pas de problème particulier.

Il suffit de s'assurer que le fichier de configuration `menu.lst` se situe bien dans le répertoire `/boot/grub/`, de même que les fichiers binaires du GRUB. Ces fichiers sont respectivement les fichiers `stage1`, `stage2` et tous les fichiers `*_stage1_5`. S'ils ne s'y trouvent pas, vous pourrez les copier à partir du répertoire `/usr/share/grub/i386-pc/`, dans lequel le programme d'installation du GRUB les place par défaut.

Lorsque tous les fichiers sont en place, il n'y a plus qu'à lancer le GRUB en mode interactif avec la commande suivante :

```
grub
```

et à définir le secteur où il doit installer son fichier d'amorçage principal `stage1` (c'est-à-dire dans le secteur de boot d'une partition ou directement sur le MBR du premier disque dur). Pour cela, vous devrez utiliser les deux commandes suivantes :

```
root source
setup destination
```

`source` est ici la partition où est installé le GRUB (il s'agit donc de la partition où se trouvent le répertoire `/boot/grub/`), et `destination` est le disque dur ou la partition dont le premier secteur doit recevoir le code d'amorçage du GRUB. Ces deux informations doivent suivre la syntaxe utilisée par le GRUB pour spécifier les disques durs et les partitions. Par exemple, pour installer le GRUB sur le secteur de boot de la deuxième partition du premier disque dur, on utilisera les deux commandes suivantes:

```
root (hd0,1)
setup (hd0,1)
```

Cet exemple suppose que le GRUB est également installé dans cette partition. Si ce n'est pas le cas pour vous, vous devrez modifier la partition spécifiée dans la commande **roo**t. Vous pourrez quitter le **grub** avec la commande **quit** une fois l'installation terminée.

Réalisation d'un multiboot avec NTLDR

Le gestionnaire d'amorçage de Windows NT, Windows 2000 et XP se nomme « NTLDR ». Ce gestionnaire d'amorçage permet de démarrer ces systèmes, bien entendu, mais également les autres systèmes d'exploitation les plus courants, dont Linux. Cette section ne traitera bien entendu que de la manière d'utiliser NTLDR pour démarrer Linux, pour de plus amples informations sur la manière d'ajouter les autres systèmes d'exploitation, veuillez consulter la documentation de Microsoft.
Tout comme LILO, NTLDR dispose d'un fichier de configuration qui permet de décrire les différentes options de son menu de démarrage. Il s'agit du fichier `boot.ini`, qui est normalement placé à la racine du disque où NTLDR est installé. Il s'agit donc généralement du répertoire racine du disque `C:\`.

Le fichier `boot.ini` contient la définition du système d'exploitation à lancer par défaut et du délai d'attente à attendre avant de le sélectionner. Ces deux informations sont écrites dans la section `[boot loader]` du fichier et sont introduites respectivement par les options `default` et `timeout`. La valeur de l'option `default` doit être l'une des entrées de la section `[operating systems]`, qui contient la définition de tous les systèmes d'exploitation que NTLDR devra proposer au démarrage.

La syntaxe des entrées pour les systèmes d'exploitation dans la section `[operating systems]` est la suivante :

```
emplacement = "Nom"
```

où `emplacement` est la description de l'emplacement où se trouve le système d'exploitation, et `Nom` est le nom avec lequel ce système devra apparaître dans le menu de démarrage de NTLDR. Cette syntaxe est très simple, mais pour les systèmes NT4, Windows 2000 et XP, la définition de l'emplacement du système est assez compliquée. En effet, elle nécessite de définir complètement le disque physique et sa partition, ainsi que le répertoire du système. Pour les autres systèmes d'exploitation, le plus simple est de spécifier un fichier contenant l'image du secteur de boot de leur partition, et de laisser leur chargeur prendre en charge leur amorçage.

Pour cela, il faut bien entendu disposer d'un tel fichier. Il faut donc copier le premier secteur de la partition du système dans un fichier accessible à NTLDR, donc, en pratique, situé dans le répertoire `C:\`. Vous pourrez extraire le contenu du secteur de boot de votre partition Linux avec la commande suivante sous Linux :

```
dd if=/dev/hda3 of=linux.bootsect bs=512 count=1
```

Cette commande suppose que LILO ait été installé sur la partition `/dev/hda3`. Elle permet de lire un bloc de 512 octets de la troisième partition du premier disque dur et de l'enregistrer dans le fichier `linux.bootsect`. Vous pourrez ensuite transférer ce fichier dans le répertoire `C:\` de Windows, soit en passant par une partition FAT, soit tout simplement à l'aide d'une disquette (rappelons que le système de fichiers NTFS n'est utilisable qu'en lecture seule sous Linux).

Une fois ce fichier obtenu, vous pourrez simplement ajouter la ligne suivante dans votre fichier `boot.ini` :

```
C:\linux.bootsect="Linux"
```

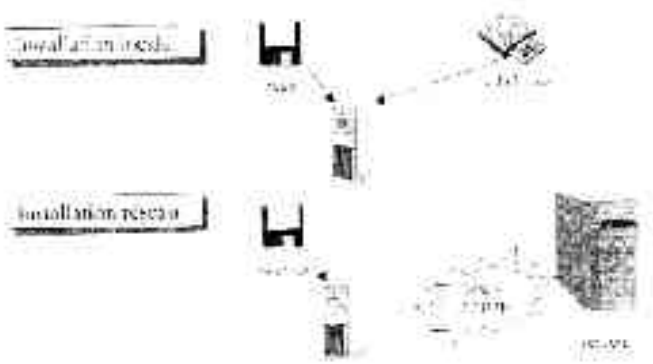
Le contenu de votre fichier `boot.ini` devrait alors ressembler à ceci :

```
[boot loader]
timeout=10
default=multi(0)disk(0)rdisk(0)partition(2)\WINNT
[operating systems]
multi(0)disk(0)rdisk(0)partition(2)\WINNT="Microsoft Windows 2000 Profession-nel"
/fastdetect
```

```
C:\linux.bootsect="Linux"
```

Vous devriez alors pouvoir démarrer Linux directement à partir du menu de démarrage de NTLDR.

VIII.4. Les Types d'installation



Installation depuis le CD-ROM local

- Démarrer l'ordinateur avec le système Linux
- Choisir la langue
- Choisir le clavier
- Choisir le mode d'installation
- Choisir le type de disque dur
- Créer les partitions Linux
- Activer les partitions
- Sélectionner les paquetages à installer
- Installer la souris
- Configurer X-window
- Configuration du réseau
- Configuration de la zone géographique
- Sélection des services Linux
- Installation d'une imprimante
- Choix du mot de passe de root
- Configuration du démarrage de votre système Linux

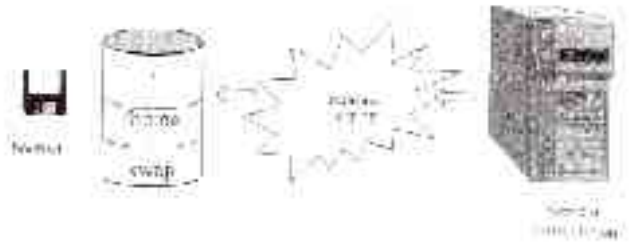
Installation depuis une image sur le disque dur local

Cette méthode nécessite que vous indiquiez le chemin d'accès du répertoire du disque dur contenant l'image du CD-ROM Linux. Elle est pour le reste, identique à l'installation depuis le CD-ROM local.

Remarque : Seule l'arborescence « Red Hat » est nécessaire pour cette installation.

Si l'image du CD-ROM Linux est sur une partition Windows 9x, vous devez vous assurer de l'orthographe du répertoire, qui doit être tres exactement « RedHat ».

Installation depuis le réseau



L’installation réseau peut être réalisé à partir d’un serveur NFS (Unix, Linux) ou d’un serveur FTP.

Ce type d’installation nécessite que vous ayez une carte réseau d’un type assez répandu car le programme d’installation ne connaît qu’un nombre restreint de cartes réseaux.

Installation à partir du serveur d’un réseau

- Démarrer l’installation à partir de la disquette de « botnet », comme pour une installation locale, et sélectionner « NFS image » comme type d’installation dans le menu qui propose les différents types d’instalation réseau.
- Sélectionner la carte réseau parmi les cartes disponibles
- Renseigner les paramètres réseaux (Adresses IP et masque de sous réseau, adresses DNS, nom de domaine, nom du serveur …)

Les classes d’installation

Linux Red Hat offe depuis la version 5.0, trois classes d’installation qui correspondent à trois type de configuration.

1. **La classe station**
   Cette classe est destinée aux utilisateurs non-initiés aux processus d’installation. Elle ne nécessite de fournir quelques informations essentielles. Elle supprime toutes les partions Linux existantes et utilisent l’espace disque disponible pour créer :
   - Une partition *« /boot » de 16 Mo*
   - Une partition *« / » de 600 Mo* environ

   Dans le cas où elle détecte la présence d’un autre système d’exploitation, le plus souvent Windows 9x, elle configure automatiquement le chargeur de Linux « LILO » en « Dual boot » pour qu’il prenne en compte le démarrage de l’autre système d’exploitation.

2. **La classe serveur**
   En choisissant cette classe, où l’installation est également simplifiée, l’utilisateur configure son ordinateur en serveur. Cette installation requiert un disque d’un minimum de 1.6 Go. Elle supprime toutes les partions existantes pour créer :
   - Une partition *«/boot » de 16 Mo*
   - Une partition *« / » de 256 Mo*
   - Une partition *« /usr » de 512 Mo*
   - Une partition *« /home » de 512 Mo*
   - Une partition *« /var » de 256 Mo*

3. La classe paramétrable

L'utilisateur peut, avec la classe paramétrable, choisir librement le nombre et la taille des partitions. Il choisit également les paquetages à installer et il décide de la localisation du chargeur LILO.

- Le chargeur LILO peut être installé sur un disquette. Le système Linux est ensuite démarré à partir de la disquette de boot que l'on vient de fabriquer.
- Le chargeur LILO peut être installé comme chargeur principal dans le MBR, « Master Boot Record ». Il faut, s'il existe plusieurs système d'exploitation sur votre disque, les ajouter au fichier de configuration de LILO, pour qu'il les propose au démarrage de l'ordinateur.
- Le chargeur LILO peut être installé dans la partition Linux. On doit alors configurer le chargeur de l'ordinateur pour qu'il puisse aussi démarrer Linux.