Chapitre 1: Caractéristiques générales du système Unix

Unix est un système d'exploitation multitâche et multi-utilisateurs, et à ce titre, il a introduit des concepts nouveaux, principalement en matière de gestion des processus et des fichiers.

L'objectif de cette partie est de présenter les particularités de ce système. Nous allons dans un premier temps rappeler dans les grandes lignes l'évolution d'Unix afin de mieux comprendre certains aspects des versions actuelles et leurs origines.

1. Evolution d'Unix:

L'histoire du système d'exploitation Unix commence en 1969 aux Bell Labs (laboratoires de recherche en informatique d'A.T.&T.). C'est pour répondre aux besoins des ingénieurs de la société que Ken Thompson écrit un système interactif, qui met l'accent sur les utilitaires de développement de logiciels, le partage de fichiers et les outils de documentation qui allait devenir Unix. Ce nom d'Unix a été attribué par opposition au système Multics (vaste projet du M.I.T.). La première évolution interne marquante d'Unix a été sa ré-écriture (par Ken Thompson et Denis Ritchie) en langage C, lequel a été inventé pour la cause en 1971. En 1975, le système Unix (v6) est distribué aux universités et aux centres de recherches. La principale université qui va travailler sur Unix est l'université de Berkeley, qui va produire ses propres versions appelées BSD pour *Berkeley Software Distribution*. En 1979, les Bell Labs sortent leur version appelée v7, avec en particulier, l'ajout de nouveaux utilitaires et un effort en matière de portabilité. Cette version est la première à être diffusée dans le monde industriel. On peut dire qu'elle est à l'origine du développement du marché Unix.

Au début des années 80, une modification de la législation américaine autorise A.T.&T. à commercialiser lui-même ses propres produits qui sont alors appelés System. C'est à cette époque que Microsoft propose sa propre version d'Unix appelée Xenix et destinée aux micro-ordinateurs. A Berkeley, les efforts portent sur l'intégration des protocoles réseaux TCP/IP, la gestion de la mémoire avec l'introduction de la pagination (alors qu'A.T.&T. reste fidèle quand à lui à la segmentation), la modification de certains paramètres du système (taille des blocs, nombre des signaux...) et l'ajout d'outils (l'éditeur `vi`, un interpréteur de commandes `csh`...). Cette prolifération des systèmes «Unix» a engendré un certain nombre de problèmes de compatibilité car chacun allant dans sa propre direction, il y avait alors plusieurs systèmes Unix. Plusieurs facteurs vont alors jouer pour canaliser et recentrer l'offre Unix: la complexité croissante des systèmes et l'action des utilisateurs.

En 1983, A.T.&T. sort la version System V, qui est issue de la v7 et qui sera enrichie au fur et à mesure par de nouvelles versions (releases). Dans le même temps, Berkeley, propose sa version 4.2 BSD (avec les fonctionnalités réseau) qui va servir de base pour de nombreux constructeurs (Sun Microsystems, Digital...) et lui permettre d'entrer dans le monde industriel. Ces deux produits marquent la fin des systèmes Unix-like en raison de leur importance par

rapport à la version de référence, la v7. La fin des années 80 est marquée par une croissance sans précédent du nombre de systèmes Unix dans le domaine des systèmes d'exploitation. Tous les constructeurs proposent une solution Unix à leur catalogue (on trouve alors trois grandes familles: les versions basées sur System V, celles issues de BSD et les versions Xenix sur micro).

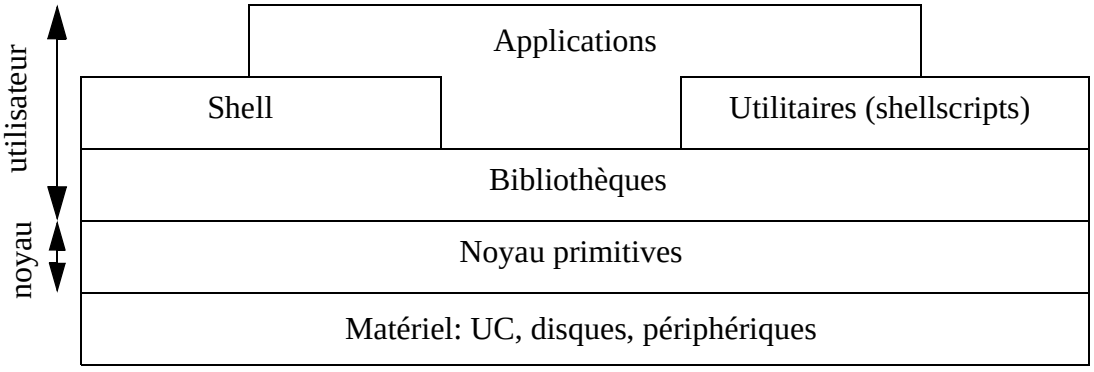
Le début des années 90 est marqué par le regroupement des constructeurs au sein de deux organisations: l'U.I. (Unix International) créée à partir de l'accord entre A.T.&T. et Sun Microsystems d'une part et l'O.S.F. d'autre part. Le premier accord a comme objectif la convergence entre les versions System V et 4.2 BSD. La première réalisation est la version System VR4 (System V release 4) qui réalise la synthèse entre SunOS (version Unix de Sun conçue sur la base BSD), Xenix et System V. L'O.S.F., quand à lui s'est fixé comme objectif de réaliser un système, appelé OSF/1, construit sur un noyau Unix et intégrant les fonctionnalités apportées par ses membres (multifenêtrages, graphismes, bases de données...) de manière à disposer d'un environnement ouvert commun aux différentes architectures des constructeurs. La principale réalisation à l'heure actuelle de l'O.S.F. est Motif qui définit un ensemble de normes au niveau de la présentation sur un environnement multifenêtré.

2. Philosophie d'Unix:

Conçus à l'origine comme un environnement de développement, Unix est aujourd'hui un système complet intégrant un grand nombre de fonctionnalités organisées en couches logicielles. en dehors de cette structure modulaire, Unix se caractérise également par les concepts introduits en matière de gestion de fichiers et de gestion des processus.

2.1 Caractère universel d'Unix:

Une des premières caractéristiques d'Unix est son écriture (à hauteur de 95%) en langage C, permettant ainsi une portabilité sur la plupart des architectures en allant des micro-ordinateurs jusqu'aux supercalculateurs. Mais cet atout ne suffit pas à lui seul à expliquer l'expansion d'Unix. Sa popularité est, en fait, due à sa conception modulaire avec des interfaces bien définies ainsi que la disponibilité d'outils simples qui coexistent entres elles. Alors que les autres systèmes d'exploitation ressemblent à des ensembles monoblocs et relativement fermés, la conception du système repose sur différents niveaux bien distincts: le noyau, un interpréteur de commandes (le shell), des bibliothèques et un nombre important d'utilitaires.



2.1.1 Le noyau:

Le noyau est la partie centrale d'Unix. Il est résident, il se charge en mémoire au démarrage. Sa structure est modulaire, ce qui rend aisées ses manipulations en termes de portabilité et l'utilisation des services qu'il offre via les primitives (ou appels systèmes). Ce fonctionnement par primitives permet de résoudre les problèmes d'accès concurrents aux informations du système. En effet, les appels systèmes font entrer l'exécution en mode noyau. Dans ce mode, le processus est assuré de garder le processeur jusqu'au retour au mode utilisateur lorsque l'appel système est terminé. Les différents noyaux Unix ont été réécrits afin de pouvoir s'adapter aux nouvelles machines multi-processeurs et de supporter le travail en temps réel. Ils sont le plus souvent réécrits en couches: les différentes fonctions du noyau sont implémentées dans des couches logicielles différentes qui communiquent entre elles par messages. La tendance actuelle est également de garder le moins de fonctions possibles dans le noyau afin de constituer un micro-noyau. Les fonctions écartées sont rejetées dans les modules exécutés en mode utilisateur.
L'interface entre le noyau Unix et les applications est définit par une bibliothèque (`libc.a` pour le langage C par exemple). Elle contient les modules permettant d'utiliser les primitives mais aussi des fonctions plus évoluées combinant plusieurs primitives. D'autres bibliothèques sont utilisées pour des services spécialisés (fonctions graphiques,...).

2.1.2 Le Shell:

L'interface utilisateur sous Unix est appelée `shell`. Lorsqu'un utilisateur tape des commandes Unix, ces commandes sont reçues par le shell qui les interprète avant de lancer l'exécution de cette commande. Le shell est une couche logicielle bien séparée du noyau. Il joue un double rôle celui d'interpréteur de commandes et celui de langage de programmation. Ce dernier rôle semblant parfois hermétique à des néophytes. Il existe plusieurs shells dont les plus répandus sont:

- le Bourne Shell (sh): le shell de base sous Unix A.T.&T.,
- le C-shell (csh): le shell Unix BSD,
- le Korn-Shell (ksh) qui est une extension du Bourne shell. Il possède toutes les commandes de son prédécesseur, ainsi que des commandes qui facilitent le travail de l'utilisateur comme des outils de gestion des historiques des commandes tapées...
- le Z-shell (zsh): extension de ksh, qui offre en particulier des modules de complétions des nom de programme, de fichiers, de variables utilisateur et système, l'envoie de message à l'utilisateur de correction en cas d'erreur de frappe.

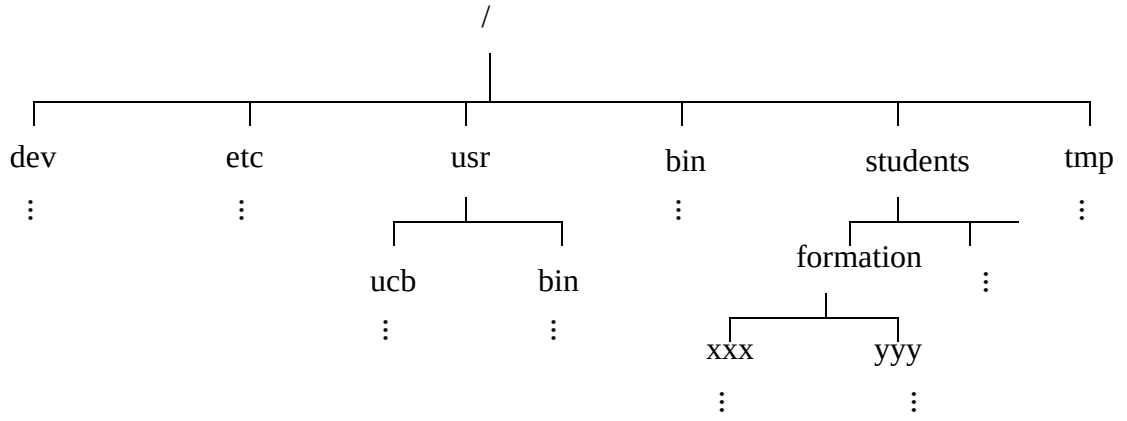
L'utilisateur, à l'aide des commandes qu'il a à sa disposition, peut écrire ses propres fonctions et programmes en langage shell, ce sont alors des shellscripts. Une fois ceux-ci réalisés, ils peuvent être utilisés par l'utilisateur comme n'importe quelle commande du shell lui même.

2.2 Le système de gestion de fichiers:

2.2.1 Vision générale:

Sous UNIX, les fichiers sont enregistrés dans une structure hiérarchisée en arbre. Ce système de fichiers est donc composé d'une racine et de noeuds qui sont des répertoires et des feuilles qui sont des fichiers ordinaires qui contiennent les données et les programmes.
Au niveau de l'utilisateur, les entrées-sorties sont vues de façon uniforme c'est-à-dire avec les mêmes commandes, la même syntaxe et les mêmes attributs qu'il s'agisse d'un fichier ou d'un périphérique. Par exemple, la redirection d'une commande sur un fichier ou sur un périphérique utilise la même syntaxe: `commande > sortie` où sortie est le nom du fichier (ordinaire ou spécial) de redirection. Mais au niveau du noyau, ce dernier effectuera l'opération de redirection soit sur le système de fichiers, soit sur le périphérique selon le type du fichier sortie. Cette vision uniforme des entrées-sorties est caractérisée par un descripteur commun à tous ces éléments, qui est appelé inode. De fait, chaque fichier Unix à un inode comprenant les attributs suivants:

- un propriétaire: celui qui a créé le fichier,
- un groupe: le groupe auquel appartient le créateur au moment où il créé le fichier,
- des droits d'accès: (voir paragraphe s'y rapportant),
- des informations générales sur la taille, la date de la dernière opération effectuée sur le fichier, le nombre de liens sur ce fichier,...



2.2.2 Les droits d'accès:

Les autorisations d'accès sous UNIX sont de trois ordres: accès en lecture (r), en écriture (w) et en exécution (x). A partir de ces trois options, on pourra effectuer toutes les opérations de base sur les fichiers (création, copie, destruction,...). Ces droits vont être donnés pour trois niveaux d'utilisateurs: pour le propriétaire, pour le groupe auquel appartient le propriétaire et pour le reste des utilisateurs. En ce qui concerne les répertoires, l'attribut x est nécessaire pour pouvoir se déplacer dans ce répertoire ou pour y rechercher un fichier.

2.3 Les processus:

2.3.1 Vision générale:

Toute action, tout programme lancé est exécuté par un processus. Le processus est donc l'unité d'exécution pour Unix. Une commande entrée par un utilisateur peut ne pas créer de processus (commandes internes au shell de l'utilisateur), ou en créer un ou plusieurs. Lors de l'exécution de la commande, un processus peut être en mode utilisateur (mode normal) ou en mode noyau, mode dans lequel il entre quand il fait un appel système (par exemple pour demander plus de place en mémoire centrale). L'ensemble de ses processus apparaissent dans une table, consultable par tout utilisateur du système. Cette table recense l'ensemble des processus et donne des renseignements quand à l'utilisateur qui l'a lancé, son taux d'occupation de la mémoire centrale, son taux d'utilisation du temps CPU de la machine, son état à l'instant de l'affichage de l'image de la table...

2.3.2 Terminal de contrôle d'un processus, exécution en arrière plan:

Les processus lancés par un utilisateur sont liés au terminal depuis lequel ils ont été exécutés, et à l'utilisateur qui les a créés. Le système peut ainsi repérer les processus qui recevront un signal d'arrêt (`Ctrl C` par exemple). L'ensemble des processus qui partagent un terminal s'appelle un groupe de processus.
Un utilisateur peut aussi lancer un processus en arrière plan, si par exemple, il veut garder la main sur une fenêtre de commande et pouvoir bénéficier d'un autre programme. Ces processus acquièrent alors une certaine indépendance vis à vis du terminal depuis lequel ils ont été lancés: ils s'exécutent alors sans intervention de l'utilisateur et sans que celui-ci n'ait besoin d'attendre la fin de leur exécution (type de traitement en «batch»). On peut ainsi lancer plusieurs tâches simultanément. Ces processus sont alors protégés contre les annulations par clavier (d'autres mécanismes que nous verrons par la suite permettent d'arrêter ces processus). D'autres processus sont quand à eux lancés par le système ou par un utilisateur en fonction de ses besoins (cas de l'impression ou du mail), ils sont alors gérés par le système et sont appelés des *daemons*.

2.3.3 La propriété effective et réelle d'un processus:

Comme nous venons de le voir un processus à un propriétaire réel, qui est l'utilisateur qui l'a commandé. Il peut avoir aussi un propriétaire effectif, c'est à dire l'utilisateur a qui appartient le fichier exécutable (si le set user bit a été positionné). C'est le cas par exemple de la commande de changement de mot passe qui nécessite l'écriture dans des fichiers protégés et dans lequel seul l'administrateur système doit avoir le droit d'écrire. Dans ce cas là, le propriétaire réel du processus et l'utilisateur et le propriétaire effectif root.

Chapitre 2: Présentation de l'environnement du MBDS

Vous avez à votre disposition un certain nombre de machines. Parmi celles-ci, certaines sont dans le local technique (les serveurs) et d'autres sont à votre disposition (les stations de travail).

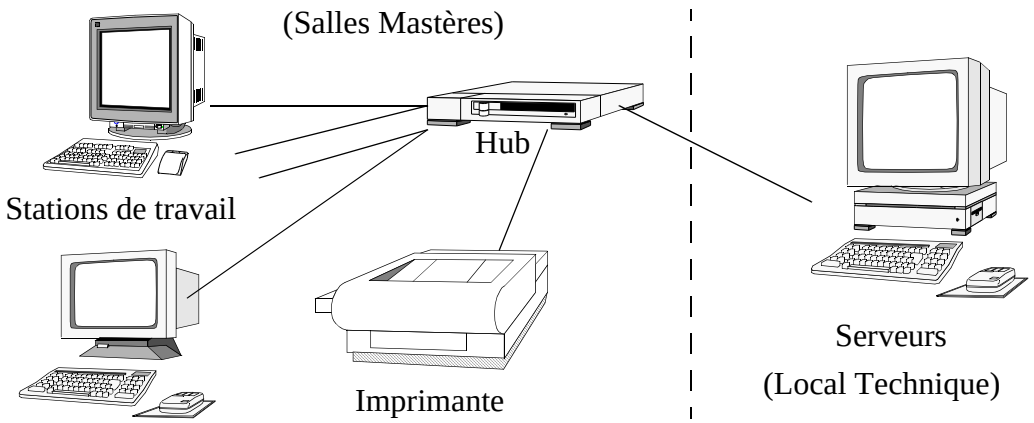
1. Les serveurs:

Ceux-ci ont plusieurs rôles: certains ont de l'espace disque pour les comptes utilisateurs (oslo en l'occurrence pour vous), d'autres jouent le rôle de machine d'«hébergement» pour les répertoires des machines diskless (mbds), d'autres encore sont des serveurs de fichiers et/ou d'exécution (lisboa, madrid ou mbds par exemple). Toutes ces machines ont entre 50 et 100 Mo de mémoire vive et un espace disque de l'ordre de 1 à 4 Go (variable selon les besoins et les périodes).
Ces serveurs sont de plusieurs types et vont vous permettre de travailler sur des environnements hétérogènes. Nous avons donc des serveurs de type Sun (madrid, mbds) avec différents systèmes d'exploitation: le serveur madrid est sous Solaris 2.5 (Unix System V) et mbds sous SunOs 4.1.3 (Unix BSD). Lisboa quand à elle est une RISC6000 et tourne sous AIX 3.5 (System V). Il y a aussi au sein du MBDS un certain nombre de serveurs qui servent à l'administration système, ceux-ci sont des Hewlett Packard (rio, oslo, tunis...) et tournent sous HP-UX 9.05; ils devront être prochainement passés sous HP-UX 10.

2. Les stations de travail:

Vous allez travailler directement sur les stations de travail qui sont à votre disposition. Parmi celles-ci, un certain nombre sont diskless (les Suns noir/blanc), elles doivent récupérer leur noyau Unix ainsi que toutes leurs arborescences sur les serveurs. Vous avez aussi accès à des terminaux X (HP couleur) qui eux vous permettent d'accéder directement à un serveur ou à une machine autorisée. Vous pourrez aussi travailler sur les PCs sous Linux.

Voila, en quelques lignes le descriptif des matériels que vous avez à votre disposition. A ceci se rajoutent des imprimantes (HP4 n/b) et tous les autres PCs sous Windows for Workgroup ou Windows NT qui vous serviront à réaliser vos projets et autres TDs.



Chapitre 3: Utilisation du Shell

1. Introduction:

1.1 Qu'est ce que le Shell:

Comme nous l'avons vu précédemment, le shell est le programme interface entre l'utilisateur et le noyau Unix. Il est indépendant du noyau. A ce sujet, vous pouvez choisir le shell que vous voulez lors de vos sessions de travail (par contre, un shell vous est toujours affecté par défaut pour toutes vos sessions lors de la création de votre compte). Il a plusieurs rôles: il interprète les commandes passées par l'utilisateur afin qu'elles soient traitées par le noyau; il a aussi la fonction de langage de programmation, il est utile pour prototyper des applications, pour effectuer certaines tâches simples et répétitives. Le langage des scripts pouvant paraître compliqué au premier abord pour un non initié, on préférera utiliser le langage Perl (de plus en plus répandu et se rapprochant du langage C, il est aussi très utilisé dans l'écriture de scripts CGI pour tout ce qui concerne les interfaces entre les programmes utilisateurs et le langage HTML).

1.2 Les différents utilisateurs:

Nous allons voir dans cette partie les différents niveaux d'utilisateurs sous Unix. Quel que soit le type d'utilisateur, il y a un certain nombre de paramètres en commun.
Un utilisateur quelconque sous Unix est repéré par son nom (login) qu'il doit taper pour pouvoir entrer dans le système. A chaque nom, correspond un numéro d'utilisateur: l'uid. L'uid est donc le moyen pour le système de reconnaître l'utilisateur (en ce qui concerne l'attribution de la possession d'un fichier par exemple).

Afin d'authentifier l'utilisateur, un mot de passe lui est attribué lors de la création de son compte. Ce mot de passe est «secret», il ne doit pas être divulgué à autrui. De plus, en cas de problèmes lors de l'entrée dans le système sur le mot de passe, vérifiez le type de clavier que vous utilisez, que vous n'êtes pas en mode majuscule, et cas échéant, si vous avez oublié votre mot de passe, allez voir votre administrateur système afin qu'il le change. Il répond à des conditions bien précises de création. Par exemple, au MBDS, il vous faut donner un mot de passe de 8 caractères avec au moins 2 caractères non alpha-numériques.

L'utilisateur appartient aussi à au moins un groupe de travail et par la même a un numéro de groupe par défaut: le gid. En fonction de projets, et au vue de la notion de droits, un utilisateur peut donc appartenir à plusieurs groupes d'utilisateurs.

Lors de la création du compte, l'administrateur système fixe un répertoire de travail par défaut (la HOME dir) dans lequel l'utilisateur sera positionné à l'entrée dans le système. En règle générale, le nom terminal du chemin de ce répertoire représente le nom de login de l'utilisateur. De façon générale, l'utilisateur est propriétaire de son répertoire HOME.

Par défaut aussi, et lors de la phase de création du compte, on donne à l'utilisateur un shell qui sera exécuté lors de l'entrée dans le système.

En résumé, à un utilisateur correspond les renseignements suivants:

- un nom de login,
- un mot de passe,
- un iud,
- un ou plusieurs groupes de travail,
- un répertoire HOME,
- un shell par défaut.

Les informations que nous venons de donner ci-dessus sont enregistrées dans le fichier: `/etc/passwd` (/etc/group pour tout ce qui concerne les groupes. Pour des raisons de sécurité et de commodité, l'administrateur utilisera des outils de gestion du système type NIS ou Kerberos. Dans ce cas là, le fichier /etc/passwd ne contient que certaines informations sur certains utilisateurs, le reste étant placé dans les fichiers relatifs au système de gestion utilisé.

1.2.1 L'utilisateur de base:

Il utilise les applicatifs mis à sa disposition, il «customise» les applications, il possède des droits d'accès traditionnels, il doit veiller à utiliser les ressources (disques, CPU,...) avec modération afin de ne pas surcharger le système.

1.2.2 Le programmeur:

Il possède le même droits que l'utilisateur de base, il a en plus la possibilité de programmer ses propres applications, il utilise donc dans ce cas, les outils de développement installés sur le système, il a accès aux compilateurs et aux interpréteur.

1.2.3 L'administrateur système:

Il possède les mêmes droits que le programmeur. Son rôle est plus grand: il gère le système, il possède des droits d'accès étendus (il peut descendre dans toutes les arborescences de son système), il crée des comptes pour tous les utilisateurs de son serveur, il veille au bon fonctionnement général du système, il installe les outils nécessaires à la communauté, il surveille et régularise la charge du système et des ressources de la machine.

Utilisation:

L'étudiant Scott Tiger arrive au MBDS et est enregistré par l'administrateur système avec les attributs suivants:

- nom de login: tiger
- passwd: pwd4s-t;
- groupe: students-mbds
- uid: 10000
- home-dir: /u/students/mbds##/tiger
- e-mail: Scott.Tiger@ceram.fr

Lorsqu'il a entré tous ces attributs dans les fichiers système, l'administrateur crée son arborescence et lui installe les fichiers de base d'environnement:

- .Xdefaults (fichier de ressources pour les clients X-Window),
- .kshrc (fichier de configuration des paramètres
- .profile (fichier de démarrage lu et exécuté par le shell),
- .mh-profile (fichier de configuration du programme xmh, servant à la lecture des mails),
- Mail (répertoire qui contiendra l'ensemble des messages reçus et la définition des masques de saisie des mails),
- lib/X11/app-defaults (arborescence dans laquelle seront définies les fichiers de configuration concernant l'apparence des fenêtres à l'écran),
- script (répertoire contenant les scripts de base permettant de lancer l'interface X).

Une fois ce travail effectué, Scott Tiger a la possibilité d'ouvrir une session de travail sur toutes les machines qu'il a à sa disposition dans les salles machines du MBDS. Il commencera par être un simple utilisateur qui «customisera» ses fichiers de démarrage, il deviendra très rapidement un programmeur au fur et à mesure de l'avancement des cours (Unix, C, C++,...). Il ne sera jamais administrateur du système.

1.3 Exemple de session utilisateur:

Après avoir été créer votre compte, vous pouvez commencer à entrer dans le système, pour cela, entrez votre nom de login et votre mot de passe (ces derniers sont identiques sur toutes les machines du réseau). Après authentification, votre shell de base se lance et exécute les fichiers /etc/profile, ~/.profile et ~/.kshrc afin de fixer vos variables d'environnement (TERM, PATH, MANPATH...). Le répertoire HOME qui vous a été affecté devient alors le répertoire courant. Pour lancer l'interface graphique, tapez x11.

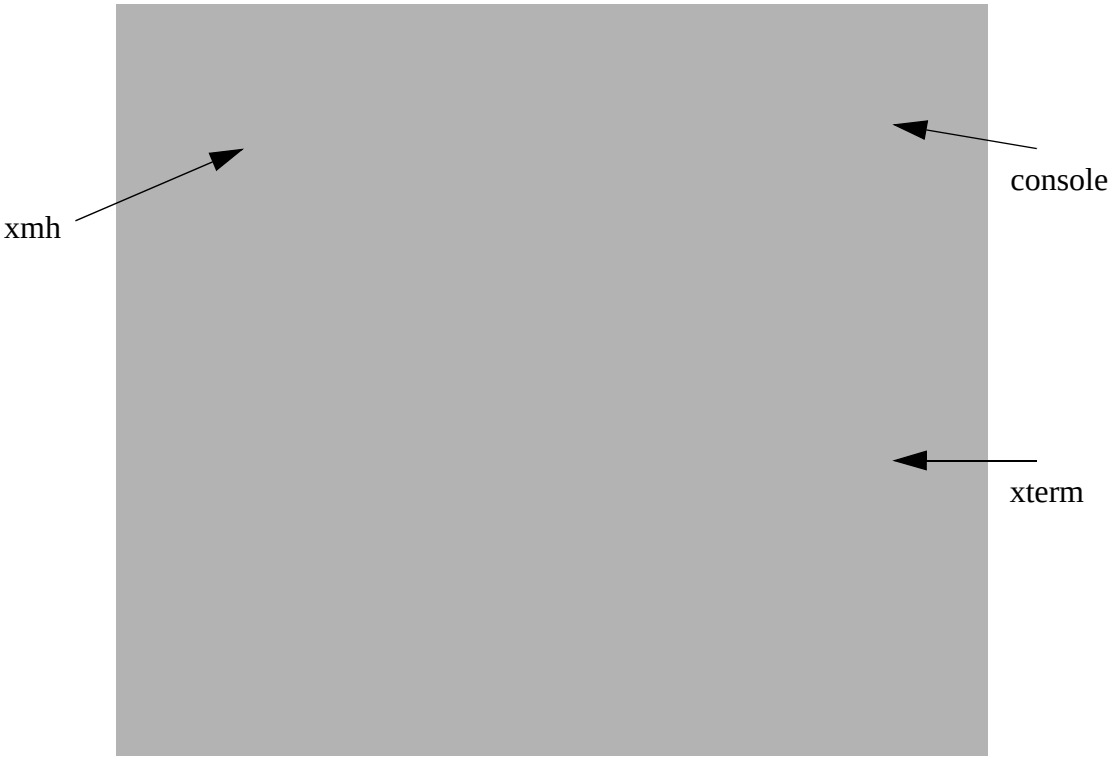
Utilisation:

Par exemple pour notre utilisateur Scott Tiger:

```
machine login: tiger
password:.....

machine$          -- vous êtes entrés sur le système
                  -- et vous êtes en mode caractères
machine$ x11      -- vous lancez l'interface graphique
```

et vous arrivez sur un écran de ce type:



2. Le Shell:

2.1 Entrée dans le système:

Lorsqu'un terminal est allumé, les processus système sont lancés et en particulier celui qui s'occupe de gérer «l'arrivée» de nouveau utilisateur (le daemon du programme login). A ce moment, vous pouvez entrer votre nom d'utilisateur et votre mot de passe afin de démarrer votre session.

2.1.1 Changer son mot de passe:

Vous avez fixé votre mot de passe avec l'administrateur système le jour où vous vous êtes fait recenser sur le réseau. Si vous souhaitez changer de mot de passe, la commande permettant de réaliser cette opération est: `passwd` ou `yppasswd`.

Utilisation:

machine $ yppasswd
Changing NIS password for *USER* on *MACHINE*.
Old password: -- entrez votre mot de passe courant
New password: -- entrez votre nouveau mot de passe
Retype new password: -- rentrez votre mot de passe
NIS entry has changed on *MACHINE*.

Les règles concernant le mot de passe ont été citées plus haut. Attention toutefois, le système Unix les majuscules et les minuscules ne sont pas équivalentes.

2.1.2 Format général des commandes:

La syntaxe générale des commandes Unix est:

```
commande options... arguments...
```

Les options sont le plus souvent précédées par un tiret «-». L'ordre des options est le plus souvent indifférent et peuvent être regroupées derrière un seul tiret. Les arguments quand à eux peuvent être absents et, dans ce cas là, ils prennent des valeurs par défaut.
Par exemple:

- ls
- ls -lia
- ls -l /net
- ls *foo*

2.1.3 Astuces pour se déplacer sur la ligne de commandes:

Il existe sous Unix certains raccourcis clavier utiles pour se déplacer sur la ligne de commandes ou pour rappeler une commande précédemment lancée. Voici une liste succincte de ces raccourcis:

- Ctrl A: pour revenir en début de ligne,
- Ctrl E: pour aller à la fin de la ligne,
- Ctrl F: pour se déplacer vers la droite,
- Ctrl B: pour se déplacer vers la gauche,
- Ctrl D: pour effacer le caractère à droite du curseur,
- Backspace: pour effacer le caractère à gauche du curseur,
- Ctrl K: pour supprimer la fin de la ligne,
- Ctrl P: pour revenir à la commande précédente,
- Ctrl N: pour passer à la commande suivante (dans la liste des commandes),
- Ctrl R + début de la commande: pour retrouver une commande dans la liste,
- Ctrl S: pour interrompre la transmission des caractères entre le programme et l'écran,
- Ctrl Q: pour libérer tous les caractères depuis le Ctrl S précédent,
- Ctrl C: pour interrompre l'exécution d'un programme,
- Ctrl Z: permet de suspendre un processus en avant plan, d'exécuter une commande ou de placer le job en arrière plan (+bg) ou de le ramener en avant plan (+fg).

2.1.4 Le manuel en ligne:

Une aide en ligne est disponible sous Unix et la plupart des commandes sont présentes dans ce manuel. La commande `man` permet de consulter ce manuel. La syntaxe générale est la suivante:

```
man [section] nom_de_la_commande
```

Si la page existe (si elle est trouvée dans l'une des arborescences définies dans la variable MANPATH), le système la formate et l'affiche à l'écran. En règle générale, la réponse a beaucoup de problèmes se trouve dans ces pages. N'hésitez donc jamais à les consulter.
Attention toutefois, certaines commandes peuvent apparaître plusieurs fois dans la liste des pages de manuel. En effet, certaines fonctions sont utilisées dans plusieurs cas de figures; par exemple, la commande `if..then..else` peut être aussi bien une commande Unix, qu'une commande C, ou C++ ou Perl... dans ce cas, il existe donc plusieurs pages se référant à cette commande (plusieurs sections contiennent des informations sur cette fonction), il faut faire donc attention à appeler la bonne page de manuel. Pour cela, il existe un moyen de connaître toutes les pages de manuel se référant à un mot clé donné:

```
man -k mot-clé
```

l'utilisateur verra une ligne pour chaque entrée du manuel concernant ce mot-clé.

Utilisation:

Dans le cas de la recherche par exemple portant sur la commande `file` (cette commande a plusieurs utilités dans plusieurs cas sur le système) qui sous Unix détermine le type d'un fichier, selon l'ordre des chemins dans la variable MANPATH, on obtient:

```
machine $ man file

file(n)          Tcl Built-In Commands          file(n)
_____________________________________________________

NAME
file - Manipulate file names and attributes

SYNOPSIS
 file option name ?arg arg ...?
_____________________________________________________
```

on voit donc ici que cette page ne correspond pas aux attentes de l'utilisateur. Il doit donc consulter le manuel par mot-clé et trouver le fichier qui correspond à ses besoins:

```
machine $ man -k file
file (n)          - Manipulate file names and attributes
mh-profile (5)    - user profile customization for MH message handler
mh-profile (l)    - user profile customization for MH message handler
rcsfile (5)       - format of RCS file
refile (1)        - file message in other folders
refile (l)        - file message in other folders
shrinkfile (1)    - shrink a file on a line boundary
tiffgt (1)        - display an image stored in a file (Silicon Graphics version)
file (1)          - determine the type of a file by examining its contents
mkfile (8)        - create a file
rasterfile (5)    - Sun's file format for raster images
sccsfile (5)      - format of an SCCS history file
```

La page que souhaitait visualiser l'utilisateur est la page: file (1).

```
machine $ man 1 file
FILE(1)                 1989                 FILE(1)
_____________________________________________________________________
NAME
 file - determine the type of a file by  examining  its  contents

SYNOPSIS
 file [ -f ffile ] [ -cL ] [ -m mfile ] filename...
_____________________________________________________________________
```

2.1.5 Sortie du système:

La commande `exit` ou `Ctrl D` permettent de sortir de la session ou de quitter la fenêtre xterm en cours d'utilisation. Si vous êtes en mode fenêtré, sortez d'abord du window manager utilisé (par défaut mwm) et ensuite quittez votre session.

2.2 Syntaxe de commande

Nous allons voir dans cette partie la plupart des commandes de bases utiles pour manipuler les fichiers et observer le système.

2.2.1 Fonctionnement de l'exécution d'une commande:

Lorsqu'il a la main, un utilisateur peut lancer l'exécution d'une commande en tapant son nom après le prompt. La commande est alors exécutée. Comment fonctionne cette exécution? Il existe une variable d'environnement: **PATH** qui contient l'ensemble des chemins les plus utilisés.

Utilisation:

```
machine $ echo $PATH
/bin:/usr/ucb:/usr/bin:/usr/bin/X11:/net/bin:/net/X/bin:/net/pack/frame/bin:
/usr/openwin/bin:/u/students/mbds##/user/bin
```

Comme sous DOS, cette variable contient tous les chemins où le système peut trouver les commandes les plus utilisées par le demandeur. Le parcours se fait de gauche à droite. Il faut donc faire attention, au cas où un exécutable est présent sur le système sous plusieurs versions différentes à bien positionner la hiérarchie des répertoires en fonction de vos besoins.
Il est aussi conseillé, pour des raisons de sécurité évidente de ne pas mettre dans la liste des chemins sa HOME directory au cas où un fichier «piégé» y ait été placé (exemple: cheval de troie 'ls' ou de démarrage de session...).
Donc, seules les commandes qui peuvent être trouvées via la variable PATH sont exécutées directement. Pour toutes les autres, il est nécessaire d'indiquer au shell le chemin absolu (ou relatif) pour atteindre cette commande.

2.2.2 Commandes liées à l'arborescence de fichiers:

2.2.2.1 Nom d'un fichier (nom absolu / nom relatif):

Pour fixer le nom d'un fichier, il n'y a pas comme sous DOS par exemple de contraintes quand à la syntaxe. Tous les caractères à l'exception de «/» sont autorisés. Le «/» servant de délimiteur de répertoires.
Deux notions sont liées au nom du fichier: son chemin absolu et son chemin relatif.

- Le *chemin absolu* est composé du nom du fichier précédé par le chemin d'accès à ce fichier depuis la racine de l'arborescence (exemple: /u/students/mbds##/user/.profile);
- Le *chemin relatif*, quand à lui est une troncature de cette arborescence par rapport au répertoire courant (exemples: ~/.profile, ../other_user/.profile, ./.oracle). En ce qui concerne les chemins relatifs, un certain nombre de raccourcis sont utilisés:
  - . désigne le répertoire courant,
  - .. désigne le répertoire parent du répertoire courant,
  - ~/ est la contraction de votre HOME directory.

2.2.2.2 Visualisation de l'arborescence:

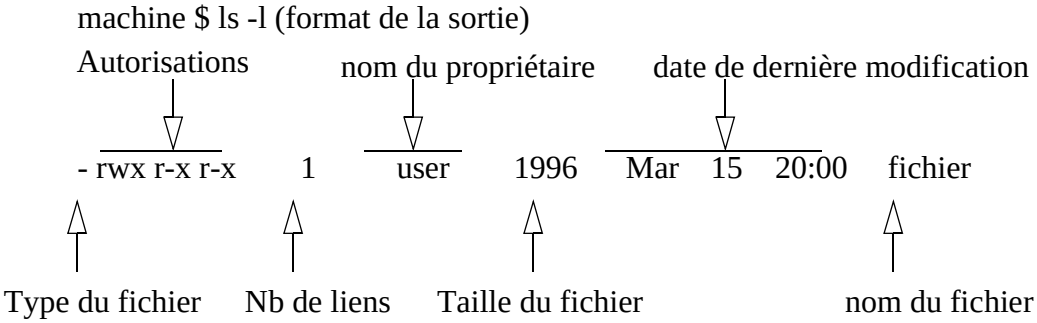
La commande `ls` permet d'obtenir la liste des fichiers d'un répertoire ou d'une arborescence de répertoires. Un certain nombre d'options permettent d'afficher plus ou moins d'informations sur ces fichiers et répertoires.
Les principales options sont les suivantes:

- `-l`: format détaillé (cf. partie utilisation),
- `-a`: liste aussi les fichiers qui commencent par «.» (et qui normalement n'apparaissent pas dans la liste normale),
- `-d`: si l'argument est un répertoire, la commande liste seulement son nom et pas les fichiers qu'il contient,
- `-t`: affiche en triant par date de dernière modification
- `-R`: listage récursif des sous-répertoires,
- `-F`: fait suivre le nom des répertoires par un «/», le nom des exécutables par une «*» et le nom des liens symboliques par un «@»...
- `-g`: pour afficher les informations sur le groupe...

Utilisation:

```
machine $ ls
Mail       script

machine $ ls -lia
total 1256
16384 drwxr-xr-x  31 user       3072 Sep 12 14:08 .
53341 drwxr-xr-x   9 root       1024 Jan  3  1996 ..
16506 -rw-------   1 user        149 Aug 13 16:28 .Xauthority
16484 lrwxrwxrwx  1 user        488 Jan  8  1996 .kshrc
16481 -rwxr-xr-x   1 user        483 Jul 31 16:56 .profile
16505 -rw-r--r--   1 user        258 Aug 13 13:58 .signature
30820 drwxr-xr-x   5 user       1024 Sep  9 11:44 Mail
10240 drwxr-xr-x   2 user       1024 Aug 27 11:21 script
```



La première information concerne le type du fichier. Cet indicateur peut prendre les valeurs suivantes:

- \- pour un fichier ordinaire,
- d pour un répertoire,
- b pour un fichier spécial de type bloc (périphériques...),
- c pour un fichier spécial de type caractère (disque, streamer...),
- l pour un lien symbolique,
- s pour une socket.

La seconde concerne les droits (utilisateur-propriétaire, groupe de travail, autres utilisateurs), vient ensuite le nombre de liens sur le fichier (ou le répertoire), puis le propriétaire effectif, la taille, la date de dernière modification et le nom du fichier.

Tableau 1: Rappels sur les droits

Type de Fichier Autorisation		Fichier Ordinaire	Répertoire
Lecture	r	lire le contenu du fichier	lister les sous-fichiers
Ecriture	w	modifier le contenu du fichier	ajouter, enlever des sous fichiers
Exécution	x	exécuter le fichier	travailler dans le répertoire

2.2.2.3 Se déplacer dans l'arborescence:

La commande permettant de se déplacer dans une arborescence est la commande:

```
cd [répertoire]
```

Cette commande fonctionne avec des chemins absolus et des chemins relatifs (il suffit que les répertoires existent).
Seule, elle sert à se replacer dans le répertoire par défaut (la HOME directory).
Avec certains shell, la commande `cd -` permet de revenir au répertoire où l'utilisateur se trouvait précédemment.

2.2.2.4 Afficher le répertoire courant:

La commande permettant à tout moment de connaître le répertoire dans lequel on se trouve est:

```
pwd
```

Il est souvent intéressant de connaître à un moment donné exactement le lieu où l'on se trouve (avant d'effectuer une suppression par exemple).

2.2.2.5 Type d'un fichier:

Il est possible sous Unix de connaître aussi le type de fichier sur lequel on travaille. En dehors de l'extension qui peut être trompeuse, tous les fichiers ont une en-tête permettant de déterminer leur type (répertoire, exécutable, texte ASCII, programme C, document Postscript...). L'ensemble de ces en-têtes est défini dans le fichier /etc/magic. La commande:

```
file nom_de_fichier
```

permet de visualiser le type du fichier en question.

Utilisation:

```
machine $ file .profile
.profile:       shell script

machine $ file toto.c
toto.c:         c program text

machine $ file fichier.ps
fichier.ps:   PostScript document
```

2.2.2.6 Affichage du contenu d'un fichier:

La commande: `cat [-v] [fichier...]` sert à afficher le contenu d'un fichier si ce dernier est passé en paramètres. Sinon, c'est l'entrée standard qui est prise par défaut (clavier ou résultat d'une autre commande). L'option -v permet d'afficher les caractères non visibles.
Cette commande permet aussi de concaténer des fichiers:

```
cat fichier1 fichier2 > fichier3
```

cette opération permet de créer un fichier (fichier3) en concatenant les fichiers fichier1 et fichier2. Les notions de redirection seront plus amplement abordées dans la partie langage de commande.
Le seul inconvénient de cet outil est qu'il affiche le contenu du fichier dans sa totalité (sans possibilité simple de contrôler le défilement par pages).

2.2.2.7 Affichage page à page:

La commande: `more [fichier...]` permet d'afficher le contenu d'un fichier page à page. Le fichier par défaut est l'entrée standard (en général le résultat de la commande située avant le «|»).

Utilisation:

machine $ ls -lR | more (redirige la sortie de la commande ls -lR vers le more)
machine $ more .profile (affiche page à page le fichier .profile)

En bas de chaque page de la commande more, un certain nombre d'action sont possibles:

- h: permet d'afficher la liste des commandes disponibles à ce niveau,
- [espace]: permet de passer à la page suivante,
- [entrée]: permet de passer à la ligne suivante,
- b: permet de revenir à la page précédente,
- i[espace]: permet d'afficher i lignes en plus,
- =: permet d'afficher le numéro de la dernière ligne affichée,
- /mot: permet de se positionner 2 lignes avant la prochaine occurrence du mot «mot»,
- n: continue la recherche précédente (identique à /),
- :f: permet d'afficher le nom du fichier et le numéro de dernière ligne affichée,
- .: effectue de nouveau la commande précédente,
- q ou Q: permet de sortir de more.

La notion de pipe («|») sera aussi développée plus amplement dans la partie langage de commande.

2.2.2.8 Copier des fichiers:

Lorsque l'on travaille, il est parfois pratique de recopier des fichiers d'un répertoire vers un autre. Dans ce cas, il faut utiliser la commande:

```
cp [-i] fichier_source fichier_destination
```

Pour cela, l'utilisateur doit avoir le droit de lecture sur le fichier source, et si le fichier destination existe, il doit avoir le droit d'écriture sur ce dernier. Dans ce cas, le contenu du fichier destination est écrasé par celui du fichier source (l'option -i demande confirmation de l'opération) et le fichier destination garde ses attributs propriétaire, groupe et droits d'accès.
Si le fichier destination n'existe pas et si l'utilisateur a le droit d'écriture sur le répertoire courant, le fichier destination est créé, son propriétaire et son groupe sont celui de l'utilisateur qui effectue l'opération. Ses autorisations sont celles du fichier source.
La commande: `cp [-i] fichiers... répertoire` copie les fichiers dans le répertoire.

2.2.2.9 Déplacer/renommer des fichiers:

Il est aussi nécessaire parfois de pouvoir renommer ou déplacer des fichiers. La commande a lancer est alors:

```
mv fichier_source fichier_destination
```

ou `mv fichiers... répertoire`

Elle permet de déplacer un ou plusieurs fichiers dans un répertoire avec possibilité de changer leur nom.

Utilisation:

machine $ mv fic fic.old (on renomme le fichier "fic" en "fic.old")
machine $ mv fic* rep (on déplace l'ensemble des fichiers commençant par fic dans le répertoire "rep").
machine $ mv fic rep/fic1 (on déplace le fichier "fic" dans le répertoire "rep" et on le renomme "fic1").

2.2.2.10 Supprimer un fichier:

Pour pouvoir supprimer un fichier sous Unix, il faut avoir l'autorisation d'écriture sur le répertoire contenant ce fichier, mais pas forcément la lecture ou l'écriture sur le fichier lui-même (si l'utilisateur n'a pas l'autorisation d'écriture sur le fichier, une confirmation lui est demandée). Attention: la suppression d'un fichier est une opération sensible sous Unix. En effet, elle agit physiquement, c'est à dire qu'il n'y a pas de moyen facile de pouvoir récupérer un fichier supprimé par erreur. Cette opération nécessite de faire appel aux sauvegardes (type sauvegardes temporaires ou directement aux bandes).
La commande à lancer est la suivante:

```
rm [options] fichiers...
```

Si le fichier est un lien vers un autre, le lien est supprimé mais pas le fichier destination du lien.
Voici les principales options de la commande rm:

- `-f`: pas de demande de confirmation quand on a pas d'autorisation d'écriture sur le fichier,
- `-r`: détruit tout le contenu du répertoire en paramètre ainsi que le répertoire lui-même,
- `-i`: option par défaut à la création du compte, cette option demande confirmation pour chaque fichier. En cas de suppression en cascade, il peut être intéressant d'enlever cette option, attention toutefois à être sûr de l'opération et à bien remettre cette option pour les suppressions suivantes.

Utilisation:

machine $ rm -i fichier
rm: remove fichier?

machine $ rm fichier (si pas de droit d'écriture sur fichier)
rm: override protection 444 for fichier?

2.2.2.11 Changer les droits:

Nous avons vu précédemment que chaque fichier avait des attributs propres et définis. Il existe un moyen pour l'utilisateur propriétaire d'un fichier de changer les droits sur ce même fichier.
La commande à utiliser est:

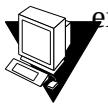
```
chmod mode fichiers...
```

Le paramètre mode indique quelles sont les autorisations données, il peut être «symbolique» ou «absolu»:

- mode symbolique: le paramètre mode prend la forme suivante:

  ```
  [qui]... op permissions..., [op permissions...]
  ```

  * «qui» désigne celui ou ceux qui recevront ces autorisations (u pour le propriétaire, g pour le groupe, o pour les autres et a pour les trois précédents - c'est l'option par défaut),
  * «op» sert à indiquer l'ajout ou la suppression de droits (+ pour ajouter, - pour enlever),
  * les permissions à donner sont r, w et x
- mode absolu: le paramètre mode est représenté par un nombre octal composé de 3 chiffres. Le premier chiffre représente les autorisations du propriétaire, le second ceux du groupe et le troisième correspond à tous les utilisateurs. Pour calculer chacun des chiffres, on ajoute les valeurs des autorisations (4 pour lecture, 2 pour écriture, 1 pour



exécution).

Utilisation:

machine $ chmod u+rwx,go+rx fichier
donne tous les droits au propriétaire, et les droits en lecture et exécution aux autres utilisateurs.
La commande précédente est équivalente à:
machine $ chmod 755 fichier

Un autre moyen de fixer ces autorisations au moment de la création du fichier est d'utiliser le bon masque. La commande:

```
umask [nnn]
```

donne cette valeur par défaut. Chaque fois qu'un nouveau fichier sera créé par l'utilisateur, il aura comme droits le complément à 7 du nombre passé à umask.
Par exemple: umask 026 est équivalent à un chmod 751 sur le nouveau fichier créé.

2.2.2.12 Les liens symboliques:

Afin d'éviter de dupliquer des informations et, aussi pour des raisons d'administrations, il est parfois utile de réaliser un lien symbolique entre fichiers ou entre répertoires. Par exemple, lorsqu'on travaille avec des binaires qui évoluent relativement vite et dont ont dispose de plusieurs versions, il est utile de créer ce type de fichier (xemacs bouge relativement vite, une version tous les trimestres environ, il est utile de réaliser un lien entre la version courante et un fichier xemacs par exemple...)
La commande à utiliser est:

```
ln -s fichier_source fichier_lien
```

A noter, les droits sur le fichier lien n'ont aucun sens et ne peuvent être modifiés, il a en fait les attributs du fichier source.

2.2.2.13 Recherche de fichiers dans l'arborescence:

Il est utile de pouvoir retrouver certains fichiers dans l'arborescence du système et d'effecteur certains traitements avec cette liste de fichiers. La commande utilisée est:

```
find répertoires... [options...]
```

où répertoires... permet d'indiquer le (ou les) répertoire(s) de départ de la recherche et les options permettent de traiter et de récupérer les données.
La commande find de nombreuses options qui permettent de rechercher des fichiers qui ont certaines caractéristiques:

- `-name modèle_nom`: tous les fichiers dont le nom répond au modèle modèle_nom,
- `-size n`: tous les fichiers dont la taille est n (en blocs de 512 octets),
- `-mtime n`: tous les fichiers dont le nombre de jours depuis la dernière modification est n,
- `-atime n`: tous les fichiers dont le nombre de jours depuis le dernier accès est n,
- `-user nom`: tous les fichiers dont le propriétaire est nom,
- `-type t`: tous les fichiers du type t (f pour un fichier ordinaire, et les autres abréviations comme pour la commande ls -l),
- `-perm nombre`: tous les fichiers dont les autorisations sont de la forme nombre,
- `-perm - nombre`: tous les fichiers dont les autorisations sont au moins de la forme nombre...

D'autres options permettent de traiter les informations reçues:

- `-print`: pour afficher à l'écran le résultat de la commande,
- `-exec commande \;`: permet d'exécuter une commande sur les fichiers qui vérifient les critères de la commande find. Ces fichiers sont passés à la commande -exec par les signes {}. Attention à bien terminer la commande par \ ;
- `-ok commande \;`: effectue la même opération que ci-dessus mais demande une confirmation de l'utilisateur avant l'exécution de la commande.

Il est possible de passer plusieurs critères de sélection des fichiers. Pour cela, il faut utiliser les opérateurs logiques ! (négation), -o (ou) et regrouper tous ces critères dans des parenthèses. L'opérateur et étant obtenu par juxtaposition des critères.

Utilisation:

machine $ find / -name "*info*" -print
affiche le nom de tous les fichiers de toute l'arborescence du système qui contiennent le mot info,

machine $ find ~ -mtime -20 -print
affiche les fichiers à partir du répertoire de l'utilisateur qui ont été modifiés dans les vingt derniers jours,

machine $ find /u/students ! -user X -exec ls -lia {} \;
affiche un listing détaillé de tous les fichiers qui n'appartiennent pas à l'utilisateur X dans l'arborescence /u/students,

machine $ find /u/students/X -perm -002 -exec ls -l {} \;
affiche un listing détaillé des fichiers de l'arborescence /u/students/X qui ont au moins l'autorisation d'écriture pour les autres,

machine $ find . \(-name a.out -o -name core -o -name "*.o" \) -ok rm {} \;
supprime après confirmation de l'utilisateur tous les fichiers à partir du répertoire courant (et dans tous les sous-répertoires) du type a.out, core et .o ...

2.2.3 Commandes liées aux répertoires:

Nous venons de voir un certain nombre de commandes liées à l'arborescence de fichiers, nous allons maintenant voir celles qui sont plus spécifiques aux répertoires.

2.2.3.1 Créer un répertoire:

La commande servant à créer des répertoires sous Unix est:

```
mkdir [options] répertoires...
```

Il suffit d'avoir le droit d'écriture dans le répertoire père. Si on veut créer une arborescence directement, il faut utiliser l'option -p. Par exemple pour créer l'arborescence ~/tp_unix/td1, taper mkdir -p tp_unix/td1.

2.2.3.2 Supprimer un répertoire:

Comme pour les fichiers, la commande rm (avec l'option -r) permet de supprimer des répertoires. Il existe une autre commande:

```
rmdir répertoires...
```

qui elle permet de supprimer les répertoires indiqués si ceux-ci sont déjà vides.

2.2.3.3 Déplacer, changer le nom d'un répertoire:

Comme pour les fichiers, il s'agit de la commande mv. Ceci n'est possible sous System V que si les deux répertoires ont le même parent, et sous BSD que si les deux répertoires sont dans le même système de fichiers.

2.2.3.4 Copier tous les fichiers d'un répertoire:

La commande à utiliser est:

```
cp -r répertoire_source répertoire_destination
```

Toute l'arborescence du répertoire source est copié dans le répertoire destination. Les nouveaux fichiers se retrouvent dans le répertoire répertoire_destination/répertoire_source.

2.2.4 Gestion des sorties imprimantes:

Comme nous l'avons vu précédemment, vous avez la possibilité d'utiliser un certain nombre d'imprimantes. Par défaut, vous êtes connectés l'imprimante de la salle machine: **homere**. Pour son utilisation, il y a à votre disposition plusieurs commandes qui vont vous permettre de lancer un travail d'impression, d'autres pour visualiser le contenu de la file d'attente, d'autres encore pour enlever des travaux en attente, et enfin, nous essayerons de voir comment réagir rapidement en cas de problèmes.

2.2.4.1 Imprimer un fichier:

Nous savons maintenant, qu'il existe sous Unix un moyen de connaître le type de chacun des fichiers. Il y a donc des programmes qui permettent d'imprimer des fichiers Postscript, des fichiers ascii, des fichiers .c... Nous allons voir ces commandes. Toutes reposent sur un même concept: l'imprimante que nous avons à notre disposition est une imprimante Postscript, il faut donc lui envoyer du code postscript. Pour cela, il existe des outils qui permettent de traduire le fichier au format postscript.
La commande de base pour imprimer un fichier est:

```
lpr [options] [liste_de_fichiers]
```

Cette commande envoie le travail d'impression dans la file d'attente, le travail sera envoyé à l'imprimante par un programme daemon (lpd). Par défaut, cette commande imprime uniquement des fichiers postscript. Si aucun fichier n'est indiqué c'est la sortie standard qui est imprimée (cf. exemple).
Les options utiles de cette commande sont:

- `-Pprinter`: pour indiquer le nom de l'imprimante sur laquelle l'impression doit être effectuée,
- `-h`: pour supprimer la bannière d'impression (première page contenant le nom de l'utilisateur et d'autres renseignements sur le fichier),
- `#n`: pour indiquer le nombre de copies à imprimer.

Cette commande va servir de base aux autres:

- `a2ps nom_fichier_ascii | lpr -Pprinter` permet de traduire et d'imprimer un fichier ascii en postscript (**a**scii**2p**ost**s**cript),
- `c2ps nom_fichier.c | lpr -Pprinter` permet de traduire et d'imprimer un fichier .c en postscript (**c2p**ost**s**cript),
- `dvi2ps nom_fichier.dvi | lpr -Pprinter` permet de traduire et d'imprimer un fichier .dvi en fichier postscript (**dvi2p**ost**s**cript)...

Et ainsi de suite, vous avez à votre disposition, sur les serveurs ou sur le net, un grand nombre de «traducteurs» au format postscript (man2ps, gif2ps, ttf2ps...) qui fonctionneront sensiblement de la même manière. Ces fichiers vous permettent de formater vos sorties. N'hésitez pas à consulter les pages de man au sujet de ces programmes, il vous indiqueront la marche à suivre pour pouvoir améliorer vos présentations.

2.2.4.2 Visualiser la file d'attente:

Maintenant que votre impression est lancée, elle est placée dans la file d'attente. Vous avez la possibilité de consulter la liste des fichiers en attente grâce à la commande:

```
lpq -Pprinter [utilisateur]
```

`-Pprinter`: permet de sélectionner seulement les informations envoyées à printer,
`utilisateur`: permet de sélectionner uniquement les travaux lancés par l'utilisateur.

Utilisation:

```
machine $ lpq -Phomere
homere is ready and printing
Rank   Owner   Job  Files                 Total Size
active   XX      1    fichier1              99999
1st      YY      2    fichierYY               888
```

2.2.4.3 Enlever un travail en attente:

Lorsque votre impression est dans la file d'attente et si vous souhaitez l'enlever de cette dernière, utilisez la commande:

```
lprm -Pprinter [-] [numero_job]
```

On ne peut supprimer les impressions d'un autre utilisateur (à moins d'être root). Lorsqu'on supprime une requête, celle-ci est interrompue si elle a déjà commencé (une partie des données se trouvant dans la mémoire de l'imprimante).
L'option - seule permet de supprimer toutes les requêtes que l'utilisateur a lancé.
Lorsqu'on donne un numéro de job, seul celui-ci est arrêté.

2.2.4.4 Si il y a un problème?

Si un problème subsiste, c'est à dire si une requête bloque la sortie des autres, supprimer cette requête de la file d'attente. A l'aide de la commande lpc, redémarrez le daemon d'impression de l'imprimante, et laissez sortir les travaux en attente. Reprenez la requête bloquante ultérieurement.

Utilisation:

```
machine $ lpc
lpc> status                    permet de voir le statut des imprimantes connectées
  homere:
  queuing is enabled
  printing is enabled
  no entries
  no daemon present
lpc> restart homere            Permet de relancer le daemon d'impression (lpd)
  homere:
     no daemon to abort
  homere:
    daemon started
lpc> quit                      on sort de l'interface lpc.
```

2.2.5 Commandes d'observation du système:

Nous venons de voir les commandes de base permettant de travailler sur les fichiers et le répertoires. Il existe aussi certaines fonctions de base qui servent à l'observation du système:

2.2.5.1 Date et Heure:

La commande:

```
date [format]
```

permet d'afficher la date système au format précisé. Voici une liste succincte des différents format possibles:

- %m: affichage du mois (01..12),
- %d: affichage du jour dans le mois (01..31),
- %y: affiche des deux derniers chiffres de l'année (00..99),
- %D: affichage de la date au format mm/jj/aa,
- %H: affichage de l'heure (00..23),
- %M: affichage des minutes (00..59),
- %S: affichage des secondes (00..59),
- %T: affichage de l'heure au format HH:MM:SS
- %j: affichage du numéro du jour dans l'année (001..366),
- %w: affichage du jour dans la semaine (dimanche=0),
- %a: affichage de l'abbréviation du jour (Sun..Sat),
- %h: abbréviation du mois (Jan..Dec),
- %r: affichage de l'heure au format 12h (AM/PM) ...

Utilisation:

```
machine $ date '+%a %d %h %y, %T'
Tue 17 Sep 96, 15:13:06
machine $ date
Tue Sep 17 15:13:06 MET DST 1996
```

2.2.5.2 Identifier les utilisateurs:

Il est possible de savoir à tout moment qui est connecté sur quelle machine. Il existe quelques commandes de base qui permettent d'avoir des renseignements sur son propre statut ou sur celui des autres.

La commande:

```
who [am i]
```

affiche le nom de l'utilisateur, le nom du terminal, la date et l'heure de connexion et l'écran de «contrôle» de chaque utilisateur actuellement sur le système.

La commande:

```
w
```

donne les informations de who, des informations sur la charge de la machine et surtout la dernière commande lancée par les utilisateurs.

La commande:

```
id
```

affiche les numéros et les noms de l'utilisateur et des groupes de travail auxquels il appartient.

La commande:

```
groups [utilisateur]
```

affiche la liste des groupes de travail auxquels appartient un utilisateur X. Si aucun utilisateur n'est spécifié ceux sont les groupes auxquels appartient l'utilisateur qui a tapé la commande qui s'affichent.

La commande:

```
finger [utilisateur][@machine]
```

permet d'avoir des informations informelles sur les utilisateurs d'une machine précisée. Si aucun utilisateur n'est spécifié, les informations concernent tous les utilisateurs connectés. Si aucun nom de machine n'est renseigné, les données portent sur la machine sur laquelle l'utilisateur qui tape la commande est connecté.

La commande:

```
chfn
```

permet de modifier les informations informelles passées à la commande finger.

2.2.5.3 Dernières connexions au système:

Il est prudent de vérifier (commande aussi très utile à l'administrateur système) que personne ne s'est connecté sur votre login en votre absence. La commande:

```
last [-n] [utilisateur] [terminal]
```

permet d'afficher les dernières connexions au système. On peut limiter le nombre de lignes renvoyées grâce à l'option -n (où n désigne le nombre de lignes affichées). On peut se limiter aussi aux connexions d'un utilisateur ou aux connexions depuis un terminal.

Utilisation:

```
machine $ last -2 X
X ttyp0 machine.ceram.fr  Mon Sep 16 10:18   still logged in
X ftp     machine.ceram.fr  Sat Sep 14 13:48 - 13:48  (00:00)
```

2.2.5.4 Occupation de l'espace disque:

Parfois, il arrive que vous receviez des messages du système ou même de l'administrateur système vous indiquant que vous occupez trop de place sur le disque. Il existe deux commandes qui permettent de savoir la place occupée par une arborescence, ou de connaître la place disponible dans un système de fichiers.

La commande:

```
du [options] [fichiers...]
```

affiche le nombre de kilo-octets occupés par les fichiers (ou répertoires) passés en paramètre. L'option -s permet de réaliser un affichage «silencieux» c'est à dire que seul le résultat final est affiché (et donc pas d'affichage du taux d'occupation des fichiers intermédiaires).

La commande:

```
df [options] [file_system] [fichier]
```

renvoie le nombre de blocs de 512 octets disponibles sur le système de fichiers spécifié. Si aucun système de fichier n'est passé en paramètre, c'est le système de fichier du fichier (ou du répertoire) indiqué qui sera renvoyé. Si aucun renseignement n'est spécifié, la commande df retourne les informations sur tous les systèmes de fichiers.
Attention, les sorties de cette commande diffèrent d'un OS à un autre, validez le format de sortie en fonction des options que vous trouverez dans le manuel.

Utilisation:

```
machine $ du -s .
2591    .

machine $ du .
13      ./skel
7       ./conf_ipc
...

machine $ df .
Filesystem          kbytes    used   avail capacity  Mounted on
/dev/sd0a           47471   12614   30110    30%    /

machine $ df
Filesystem          kbytes    used   avail capacity  Mounted on
/dev/sd0a           47471   12614   30110    30%    /
/dev/sd0g          190911   92221   79599    54%    /usr
/dev/sd0d          190911   70934  100886    41%    /var
/dev/sd3d          480487  310609  121830    72%    /net
...
```

2.2.5.5 Charge de la machine:

Lorsque vous trouvez que le système n'a pas des temps de réponses assez rapides (par rapport à d'habitude), utilisez la commande:

```
uptime
```

qui affiche l'heure et le nombre de jour depuis la mise en route du système, le nombre d'utilisateurs et la charge moyenne de la machine la dernière minute, et dans les 5 et 15 dernières minutes.

Utilisation:

```
machine $ uptime
 5:27pm  up 167 day(s), 23:07,  2 users,  load average: 0.16, 0.10, 0.05
```

2.2.5.6 Les processus:

Un certain nombre de commandes Unix se rapportent à la gestion des processus. Il existe des commandes d'audit de la table (ps, pstat, top), de mise en attente d'un processus (wait), de changement de priorité (nice, renice), d'envoi de signaux aux processus (kill). Nous allons donc voir dans cette partie la syntaxe et l'usage de ces commandes.

a) La commande:

```
ps [options]
```

affiche des renseignements sur les processus en cours d'exécution. Les options diffèrent suivant l'OS utilisé. Nous verrons ici les options pour un système BSD, consultez les pages de manuel pour un système System V.

- `-a`: affiche tous les processus lancés depuis un terminal quelconque,
- `-x`: affiche tous les processus, même ceux qui n'ont pas été lancés depuis un terminal, l'option -l rajoute des renseignements supplémentaires (le numéro du père...),
- `-e`: affiche les valeurs des variables d'environnement,
- `-w` et `-ww`: formatent le nombre de colonnes d'affichage...

Si aucune option n'est spécifiée, seuls les processus lancés par l'utilisateur sont affichés.

En règle générale, un certain nombre de renseignements sont donnés:

- USER: le nom de l'utilisateur propriétaire du processus,
- PPID: le numéro du processus père (processus qui a lancé le processus courant),
- PID: le numéro du processus,
- %CPU: le pourcentage de temps machine consommé par le processus,
- %MEM: le pourcentage de temps mémoire consommé par le processus,
- SZ: la somme de la taille des données et du segment de la pile,
- RSS: la taille en mémoire réelle du processus,
- TT: le terminal depuis lequel le processus a été lancé,
- STAT: l'état du processus au moment du lancement de la commande (voir ci-après),
- WCHAN: l'évènement que le processus attend,
- START: la jour du lancement du processus,
- TIME: le temps d´exécution de la commande,
- COMMAND: le libellé de la commande lancée.

Cette liste représente les principaux renseignements donnés, si vous avez d'autres données affichées à l'écran, n'hésitez pas à consulter les pages de man.

En ce qui concerne les états (STAT) du processus, plusieurs possibilités sont affichées:

- pour la première lettre: Etat courant
  - R: le processus est en cours de traitement,
  - T: le processus est stoppé,
  - P: le processus attend une page (typiquement récupération des données du swap vers la mémoire réelle),
  - D: le processus est en attente d'une E/S disque ou NFS,
  - S: le processus «dort» depuis moins de 20 secondes,
  - I: le processus est «idle», il «dort» depuis plus de 20 secondes,
  - Z: processus en attente d'un message du noyau (processus zombie).
- pour la deuxième lettre: Etat mémoire
  - blanc: le processus est chargé en mémoire,
  - W: le processus est swappé,
  - >: le processus avait spécifié une taille mémoire et a dépassé cette taille.
- pour la troisième lettre: Etat priorités
  - blanc: le processus tourne sans priorité spéciale,
  - N: la priorité a été réduite,
  - <: la priorité a été augmentée.
- pour la quatrième lettre: Etat pour les traitements en mémoire virtuelle
  - blanc: attente d'un signal,
  - A: nettoyage du tas (garbage collection),
  - S: pour un processus traitant un programme utilisant la mémoire virtuelle pour adresser séquentiellement des données volumineuses.

Utilisation:

```
machine $ ps
 PID TT STAT  TIME COMMAND
 7225 p0 IW    0:00 -zsh (zsh)
10522 p1 S     0:00 -zsh (zsh)
10527 p1 R     0:00 ps
```

b) La commande:

```
pstat [options]
```

permet d'avoir des renseignements sur la table des processus. Cette commande est plus utilisée dans le cadre du travail de l'administrateur système.

c) La commande:

```
top [options]
```

affiche et met à jour les informations sur les 15 processus les plus importants au niveau de la CPU. Cette commande est plus une commande d'audit du système.

d) La commande:

```
wait
```

attend jusqu'à ce qu'un processus démarré avec un & ou passé en background se termine et sort un rapport sur les terminaisons anormales. Cette commande étant lancée depuis le terminal de l'utilisateur, le shell exécute la commande wait sans créer de nouveau processus.

e) Le commandes:

`nice`

et `renice`

permettent de modifier les priorités d'un processus. Ces deux commandes ne sont et ne doivent être (pour des raisons évidentes) utilisées que dans des cas limités. Sur certains systèmes, elles seront réservées à l'administrateur système.

f) La commande:

`kill [-signal] pid`

permet d'envoyer un signal à un processus dont pid est le numéro. Le super utilisateur peut tuer tous les processus, un utilisateur ne peut quand à lui tuer que les processus qu'il a lancé. Les signaux les plus utiles sont:

- 1 (ou HUP): hangup envoyé au processus lors d'une déconnexion,
- 2 (ou INT): interromp le processus situé au premier plan,
- 3 (ou QUIT): interromp le processus récepteur du signal et génère un core dump (image de la mémoire au moment de l'arrêt),
- 9 (ou KILL): interromp le processus récepteur du signal,
- 15 (ou TERM): termine «proprement» (effacement des fichiers temporaires créés) le processus récepteur,
- 17 (ou STOP): stoppe le processus récepteur.

En règle générale, il est conseillé d'envoyer un signal TERM pour arrêter un processus, si ce signal n'aboutit pas utiliser alors le signal KILL.

Utilisation:

```
machine $ ps -l
 PID TT STAT  TIME COMMAND
  7225 p0 IW    0:00   -zsh (zsh)
10522 p1 S     0:00   -zsh (zsh)
10523 p1 S     0:00   xemacs              -- nous allons tuer ce processus !
10527 p1 R     0:00    ps

machine $ kill -15 10523
[1]  + 10523 killed     xemacs
```

Chapitre 4: Le Shell - Interpréteur de commandes

Nous allons étudier dans ce chapitre, tout ce qui concerne les variables d'environnement ainsi que les concepts de redirection, puis nous essayerons de mieux comprendre les mécanismes de base de l'interprétation des commandes par le shell.

1. Les variables:

Nous séparerons notre étude en trois parties: la manipulation des variables définies par l'utilisateur, puis nous nous intéresserons aux variables d'environnement, pour finir par les variables spéciales et réservées par le shell lui-même.

1.1 Variables simples:

Nous appellerons variable simple toute variable définie par l'utilisateur pour ses propres besoins. Nous verrons dans cette partie comment fonctionnent tous les mécanismes permettant de manipuler des variables.

1.1.1 Mécanismes d'affectation:

On peut affecter une chaîne de caractères aux variables, et par extension, toute affectation répond à la syntaxe suivante:

```
variable=valeur
```

Attention, il n'y a pas d'espace de part et d'autre du signe «=».
De plus, la commande: variable= affecte la chaîne vide à variable.

1.1.2 Désignation de la variable:

Pour désigner une variable, il suffit d'utiliser la syntaxe suivante:

```
$variable
```

Cette notation servira pour tous les types de variables que nous verrons par la suite.
Si la variable doit être suivie d'une valeur autorisée (exemple de nom de variable auquel on ajoute séquentiellement un nombre), la notation est alors la suivante:

```
${variable}caractère
```

1.1.3 Affichage d'une variable:

La commande:

```
echo [-n] [texte] $variable
```

permet d'afficher la valeur de la variable. On peut aussi l'intégrer dans du texte. L'option -n permet d'éviter le passage à la ligne après l'affichage.

1.1.4 Saisie au clavier:

Il est parfois utile de laisser l'utilisateur entrer la valeur de ses variables. La commande:

```
read variable...
```

lit les valeurs sur l'entrée standard. Le premier mot est affecté la première variable et ainsi de suite. Si il y a plus de variables que de mot saisis, le shell affecte aux dernières variables la chaîne vide.

1.1.5 Exportation des variables:

D'ordinaire, une variable n'est utilisée que dans le shell-script où elle reçoit son affectation. Si on souhaite l'utiliser pour les programmes appelés ultérieurement, on doit l'exporter. La commande:

```
export variable...
```

recopie la valeur des variables en paramètre dans l'environnement qui sera passé aux shell-scripts ultérieurs.

1.1.6 Visualisation des variables disponibles:

La commande:

```
printenv
```

affiche les valeurs des variables de l'environnement du processus en cours, c'est à dire celles qu'il a reçues et celles qu'il a envoyées.

La commande:

```
set
```

affiche les valeurs des variables disponibles, c'est à dire toutes les variables même celles qui ne seront pas transmises.

1.1.7 Supprimer une variable:

La commande:

```
unset variable
```

supprime la variable de la liste des variables disponibles.

1.1.8 Substitutions:

Au niveau de l'affichage, il est parfois utile de pouvoir effectuer un certain nombre de substitutions:

- `$paramètre`: affiche la valeur courante de paramètre,
- `${paramètre}`: affiche la valeur courante de paramètre et autorise la concaténation,
- `${paramètre:-valeur}`: affiche la valeur courante de paramètre si elle est non nulle, sinon affiche valeur,
- `${paramètre:=valeur}`: affiche la valeur courante de paramètre si elle est non nulle, sinon affiche valeur et affecte à paramètre valeur,
- `${paramètre:?valeur}`: affiche la valeur courante de paramètre si elle est non nulle, sinon on écrit valeur sur la sortie standard et on sort. Si valeur est omise, alors, on écrit sur la sortie standard le message: «paramètre : parameter null or not set»,

1.2 Variables du shell - personnalisation de l'environnement:

1.2.1 Les principales variables:

Voici les principales variables d'environnement utilisées par le shell:

- CDPATH: liste des répertoires recherchés quand cd est exécuté avec un nom relatif comme argument,
- DISPLAY: l'écran sur lequel on travaille (par défaut :0.0),
- ENV: nom d'un fichier exécuté à chaque appel de ksh,
- GID: le numéro du groupe actuel de l'utilisateur,
- HOME: répertoire de login,
- HOST: le nom de la machine sur laquelle on se trouve,
- IFS: séparateur de mots dans les commandes (espace, tabulation ou passage à la ligne); IFS est utilisé pour les commandes internes pour découper une liste de paramètres,
- LD_LIBRARY_PATH: la liste des répertoires où les programmes trouveront leurs librairies dynamiques à l'exécution,
- LOGNAME: le nom de l'utilisateur,
- MANPATH: liste des répertoires où se trouvent les pages de manuel,
- PATH: liste des répertoire de recherche des commandes, le séparateur est «:»,
- PS1, PS2: prompts initial et secondaire (habituellement $ et >),
- PWD: le répertoire courant,
- SHELL: le shell par défaut,
- TERM: nom du type de terminal utilisé,
- TERMCAP: caractéristiques du terminal (nombre de lignes, de colonnes, caractères à envoyer pour effacer l'écran, pour positionner le curseur...),
- UID: le numéro de l'utilisateur.

1.2.2 Que se passe-t-il au login?

Nous nous placerons dans le cas où le shell par défaut est un ksh. Après le login, le fichier /etc/profile (commun à tous les utilisateurs) est exécuté. Ce fichier contiendra les variables par défaut ainsi que les principaux paramètres du système. Ensuite, l'utilisateur passe dans son répertoire par défaut, et le fichier .profile est exécuté, afin de fixer les variables d'environnement propres à l'utilisateur. Ces fichiers sont exécutés dans le shell de démarrage de la session, les variables sont donc conservées pour la suite de la session si elles ont été exportées.
La variable ENV est utile pour la personnalisation de l'environnement: elle contient un nom de fichier (par défaut .kshrc) qui est exécuté à chaque fois que ksh est lancé.
Le fichier .kshrc est lié à ksh alors que le fichier .profile est aussi utilisé par sh. On mettra donc dans le fichier .kshrc toutes les définitions de variables propres à ksh.

1.3 Variables spéciales:

Un certain nombre de variables sont utilisées dans les shellscripts afin de déterminer et de séparer les paramètres:

- $#: donne le nombre d'arguments passés au script,
- $*: donne la liste des paramètres passés au script,
- $n: valeur du n-ième argument du script,
- $@: équivalent à $*, mais «$@» signifie: «$1», «$2»,..., «$n»
- $$: le numéro de processus du script,
- $!: le numéro de processus du dernier programme lancé en background,
- $?: le code de retour du dernier programme lancé.

2. Mécanismes d'interprétation du shell:

2.1 Définitions:

Le shell est l'interpréteur de commandes, il n'est pas inclus dans le noyau d'Unix. Quand un utilisateur rentre dans le système, le shell par défaut qui lui a été donné, est automatiquement lancé et toute la session se déroule en fait dans ce shell. Lorsque cet utilisateur tape une commande, le shell la lit, l'interprète et lance son exécution. La ligne tapée peut ne comporter qu'un seule commande mais peut aussi être beaucoup plus complexe. Le shell est comme nous le verrons plus tard un langage de programmation.

2.2 Commandes et processus:

La plupart des lignes de commandes entrées par un utilisateur entraînent l'exécution d'un fichier binaire du système de fichier. Le shell lance l'exécution de ces fichiers après avoir interprété le message écrit sur la ligne. Cette exécution engendre un nouveau processus. Certaines lignes de commandes peuvent aussi engendrer plusieurs processus (lignes contenant des pipes par exemple). D'autres comportant des commandes internes au shell, et donc étant exécutables sans faire appel à un fichier externe ne nécessitent pas la création d'un nouveau processus.

2.3 Prompts et tourmenterais de commandes:

Le shell indique qu'il est prêt à recevoir une nouvelle commande en affichant le prompt (par défaut «$»). Toute commande se termine par un passage à la ligne, dans certains cas, le passage à la ligne à une signification particulière pour la commande tapée et n'est donc pas considéré comme un terminateur de commande. Dans ce cas, le shell affiche le deuxième prompt (par défaut «>») pour indiquer qu'il attends la suite de la commande.
Il est aussi possible de taper plusieurs commandes sur la même ligne en les séparant par un «;».

2.4 Environnement d'une commande:

Toutes les commandes lancées s'exécutent dans un environnement de travail. Cet environnement peut guider les variables dans leur fonctionnement. Par exemple, lors du lancement d'un programme «graphique», la requête d'affichage est envoyée sur l'écran défini par la variable DISPLAY, si celle-ci n'est pas renseignée, un message d'erreur apparaît sur la ligne de commandes.
Lorsqu'elle est lancée depuis un shellscript, un commande hérite de l'environnement de ce shellscript, du masque des autorisations et des fichiers qui ont été ouverts.

2.5 Recherche d'une commande:

Si la commande tapée correspond à une commande interne du shell, cette commande est exécutée à l'intérieur de ce shell. Sinon, si la commande correspond à une fonction (cf. partie sur le langage de programmation) elle est elle aussi exécutée à l'intérieur du shell.
Dans tous les autres cas, la commande se réfère à un fichier de l'arborescence. Si le nom ne comporte pas de caractère «/», le shell examine la variable PATH pour rechercher le répertoire où ce trouve cette commande. Un nouveau shell est alors créé pour exécuter cette commande.
Si la commande est un shellscript, le shell lit toutes les lignes du script et les exécutent. Si la commande appelle un fichier binaire exécutable, le code et les données associées du processus du deuxième shell sont remplacés par le code et les données associés à la commande.

2.6 Les alias:

Un alias donne un synonyme pour une chaîne de caractères dans une commande. On peut ainsi redéfinir toutes les commandes en fonction de ses propres besoins. Par exemple, il est courant d'avoir un alias sur la commande `rm` qui transforme celle-ci en `rm -i`.
La commande:

```
alias nom_alias='commande'
```

permet de définir un alias. Si ce dernier est utile pour toutes les sessions, il est utile de les placer dans le fichier .kshrc.
La commande:

```
alias
```

permet d'afficher tous les alias utilisés dans la session.
Pour défaire un alias, utiliser la commande:

```
unalias nom_alias.
```

2.7 Substitution de commande:

Le shell interprète la commande `‘commande‘` en la remplaçant par tous ce que la commande envoie sur la sortie standard.
A noter:

1) La commande est exécutée dans un sous-shell,
2) L'interprétation à l'intérieur des «‘» est complexe: le shell commence par enlever tous les \ (qui inhibent les mots à interpréter) qui précèdent un $, un ‘ ou un \, ensuite, la commande entre «‘» est interprétée comme une commande normale.
3) Lorsque la substitution de commande se trouve dans une redirection, la commande entre «‘» est concernée par cette redirection, il y a donc un ordre dans l'interprétation des commandes.

2.8 Substitution de nom de fichier et autres caractères spéciaux:

Il existe sous Unix des méta caractères qui sont utilisés pour la substitution des noms de fichiers:

- ?: remplace un caractère quelconque dans un nom de fichier,
- *: remplace n'importe quelle chaîne de caractères (y compris la chaîne vide),
- [...]: remplace l'un des caractères entre crochets (on peut donner un intervalle),
- [!...]: remplace l'un des caractères qui n'est pas entre crochets.

D'autres notations sont utiles:

- (...): exécute les commandes de ... dans un sous-shell,
- {...}: exécute les commandes de ... dans le shell courant,
- commande1 && commande2: exécute commande1, si le code de retour est OK alors on exécute commande2,
- commande1 || commande2: exécute commande1, si le code de retour est mauvais alors on exécute commande2.

2.9 Mécanismes d'interprétation:

Nous allons voir dans cette section comment les lignes de commande sont interprétées par le shell avant que ce dernier ne la lance.
Le shell repère dans un premier temps les redirections et les pipes. L'interprétation des parties correspondant aux fichiers dans les redirections est différente du reste de la ligne. Les affectations sont aussi traitées à part. Pour les autres parties, voilà les tâches effectuées par le shell (par ordre d'exécution):

- substitution des alias,
- substitution du caractère «~»,
- substitution des paramètres de position et de variables ($variables),
- substitution des commandes (entre «‘»)
- interprétation des espaces (afin de distinguer les différents mots de la commande),
- génération de noms (pour tout ce qui concerne les méta-caractères *, ?, ou [..]), et si aucun fichier ne correspond au modèle donné, le modèle est laissé de côté.

Toutefois, il existe un moyen de «dépersonnaliser» les substitutions avec les caractères réservés suivants:

- \: inhibe l'interprétation spéciale du caractère suivant. Trois cas de figures sont possibles:
  - si \ est entre guillemets, il est laissé, sauf si il est suivi de $, ‘, \ ou "; dans ce cas, il est enlevé et le caractère suivant perd sa signification spéciale,
  - si \ n'est pas entre guillemets, il est enlevé et le caractère suivant perd sa signification spéciale (s'il en avait une); si le caractère suivant est un passage à la ligne, les deux caractères (\ et passage à la ligne) sont enlevés,
  - si \ est entre ‘, il est laissé sauf si il est suivi de $, ‘ ou "; dans ce cas, il est enlevé et le caractère suivant perd sa signification spéciale.
- ’: inhibe toute interprétation;
- ": inhibe l'interprétation des espaces, la génération de noms, la substitution des alias et de ~, mais pas la substitution des paramètres et des commandes. Une apostrophe n'est pas interprétée entre deux guillemets, ce qui permet de mettre le caractère ’ dans une chaîne.

Utilisation:

```
machine $ var=valeur

machine $ echo $var xyz ‘pwd‘
valeur xyz /u/students/user

machine $ echo \$var xyz
$var xyz

machine $ echo "\$var xyz ‘pwd‘ *"
$var xyz /u/students/user *

machine $ echo ’$var ‘pwd‘ *’
$var `pwd` *

machine $ echo ’$ représente le signe du ’ "’dollar’"
$ represente le signe du  'dollar'
```

2.10 Lancement de l'exécution d'un shellscript:

On appelle shellscript un fichier qui contient des noms de commandes et des instructions internes au shell. L'utilisateur peut lancer un shellscript commande une commande. Les commandes contenues dans le shellscript sont alors lancées comme si l'utilisateur les tapait sur la ligne de commande. On peut lancer l'exécution d'un shellscript de trois manières différentes:

2.10.1 Lancement par le nom du shellscript:

Si l'utilisateur a le droit de lecture et d'exécution sur le shellscript, il peut le lancer en tapant simplement son nom. Il se comporte alors comme n'importe quelle commande. En particulier, il peut être utilisé par un autre shellscript. On peut construire ainsi de nouvelles commandes adaptées à ses propres besoins.
Lors de l'appel, le shell parcours la variable PATH pour trouver le fichier, donc par prudence, il est bon de lancer le script comme suit: `./shellscript`, ceci évitera le temps de recherche du script dans les arborescences de répertoires et surtout l'exécution d'un autre script portant le même nom.

2.10.2 Lancement par appel explicite du shell:

Si le script porte un attribut de lecture, on peut le lancer par appel explicite du shell:

```
/bin/ksh nom_script
```

Dans ce cas, un nouveau shell est lancé, celui-ci lit les commandes du script et les fait exécuter comme si elles avaient été tapées au clavier.

2.10.3 Lancement par appel de la commande interne «.»:

La dernière solution consiste à faire précéder le nom du fichier par un «.»:

```
. shellscript
```

Dans ce cas, il n'y a pas de création d'un nouveau processus, les modifications de l'environnement sont conservées. En fait, «.» est une commande interne au shell qui lit toutes les commandes contenues dans le shellscript et les exécute comme si elles avaient été tapées au clavier. De la même façon, la variable PATH est parcourue pour déterminer l'emplacement dans le système de fichiers du script.

3. Les redirections:

Avant de lancer un processus, le shell lui attribue trois fichiers:

- Une entrée standard (par défaut le clavier),
- Une sortie standard (par défaut l'écran),
- Une sortie d'erreur standard (par défaut l'écran).

Ces trois directions peuvent être liés à d'autres fichiers que ceux indiqués ci-dessus grâce aux commandes de redirection. Le shell associera donc d'autres fichiers que le clavier ou l'écran et il lancera ensuite la commande.

3.1 Redirections des sorties:

3.1.1 Redirection avec écrasement du fichier de redirection:

Cette opération consiste à rediriger la sortie standard d'une commande dans un autre fichier par la commande:

```
commande > fichier
```

Tout ce qui est envoyé vers la sortie standard est donc renvoyé vers ce fichier. La redirection est donc écrite à la suite de la commande sur laquelle elle agit. Si le fichier existe déjà, son contenu est écrasé, sinon il est créé.

3.1.2 Redirection avec ajout à la fin du fichier de redirection:

L'opération fonctionne de la même façon que la précédente, la commande diffère:

```
commande >> fichier
```

3.1.3 Redirection du fichier d'erreur:

Le fichier d'erreur peut être redirigé en ajoutant:

`commande 2>fichier` (ou `commande 2>>fichier`)

Si, par exemple, on ne veut pas que les messages d'erreur s'affichent à l'écran, la commande est: commande 2> /dev/null (/dev/null est un pseudo fichier sorte de vide-poches où aucun message n'est conservé).

3.2 Redirection des entrées:

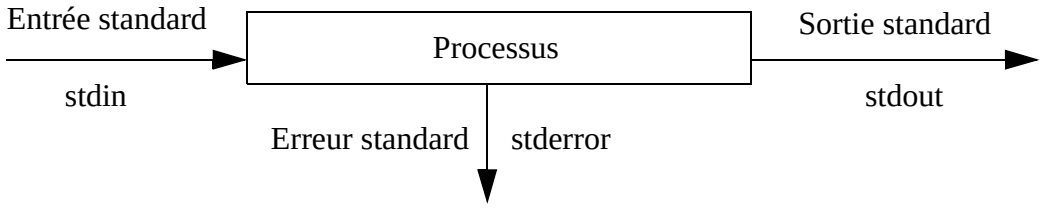
Dans le cas où on veut récupérer les données non plus du clavier mais d'un fichier, on utilise la commande:

```
commande < fichier
```

On peut définir ainsi plusieurs redirections dans une même commande:

tr "[a-z]" "[A-Z]" < fichier1 > fichier2

écrit en majuscules dans fichier2 le contenu de fichier1.



3.3 La notion de pipe:

Les données de sortie d'une commande peuvent servir directement comme données en entrée à une autre commande. Pour effectuer cette opération, il suffit de placer un «|» entre les deux commandes:

```
commande1 | commande2 | ...
```

Chacune des commandes est exécutée dans son propre shell. Le code de retour du pipe est le code de retour de la dernière commande exécutée.

Utilisation:

machine $ ls -lia | more
permet d'afficher en mode page à page

machine $ ls -lia | sort | lpr
permet de trier les fichiers d'un répertoire et de les imprimer par ordre alphabétique.

Il est possible aussi de renvoyer la sortie en même temps sur la sortie standard et dans un fichier. La commande à utiliser est:

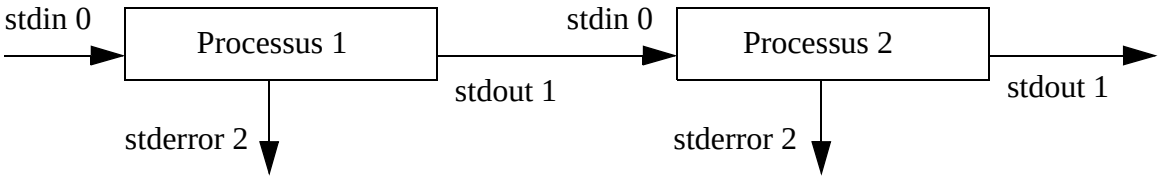
```
tee fichier
```

Utilisation:

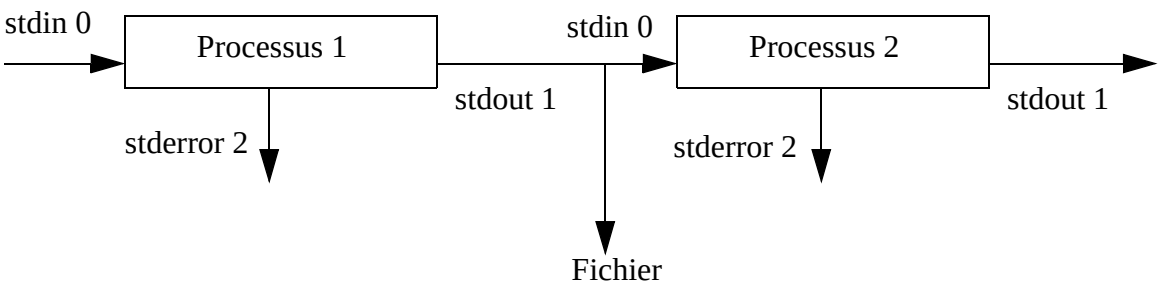
machine $ ls -lia | tee fichier | sort
permet de conserver le résultat de ls -lia dan sun fichier et de le trier par sort.

Ce mécanisme de pipe est donc très intéressant à utiliser dans l'écriture de shellscripts.

Mécanisme de fonctionnement de "|" :



Mécanisme de fonctionnement de tee :



4. Lancement de processus en arrière plan:

4.1 Définition:

Lorsqu'on lance une commande, le shell attend la fin de l'exécution de cette dernière avant de redonner la main à l'utilisateur.
Si on ajoute un «&» à la fin de la ligne de commande, le shell n'attends plus la fin de l'exécution de cette commande, il la lance en arrière plan.
Quand on lance une commande en arrière plan dans un shell interactif (c'est à dire pas dans un shellscript), le shell renvoie à l'écran le numéro du job pour le shell qui a lancé la commande ainsi que le numéro du processus dans la table des processus.

Utilisation:

machine $ xemacs &
[1] 1319
où [1] représente le numéro du job
et 1319 le numéro du processus dans la table.

Le caractère «&» est un terminateur de commande, il est donc possible de taper une commande à la suite sans passage à la ligne.

Attention:
`ls > toto &` lance la commande ls en arrière plan en redirigeant les noms de fichiers sur le fichier toto, alors que:
`ls & > toto` lance la commande ls en arrière plan en affichant à l'écran les noms de fichiers et ensuite effectue une redirection dans le fichier toto (la redirection n'a donc aucun effet sur la commande).

4.2 Gestion des «jobs»:

Suivant le shell (ksh, zsh mais pas sh), l'utilisateur dispose de plusieurs commandes pour choisir le lieu d'exécution d'une commande:

- pour voir la liste des processus avec leur numéro pour le shell en cours, utiliser la commande: `jobs [-l]`
- pour stopper un processus en cours, taper: `Ctrl z`,
- pour passer un processus en arrière plan, taper: `bg [%n]`,
- pour ramener un processus au premier plan, taper: `fg [%n].`

Ceci permet de stopper provisoirement un processus pour effectuer une tâche annexe et de revenir immédiatement au processus choisi.

Chapitre 5: Le Shell - Langage de programmation

Nous venons de voir les principales fonctionnalités du shell lui-même puis, son aspect interpréteur de commande, nous allons voir dans ce chapitre le rôle langage de programmation du shell.

1. Tests divers:

La commande:

```
test condition && alors || sinon
```

ou

```
[ condition ] && alors || sinon
```

est utilisée dans de nombreux cas par les structures de contrôle du shell. Cette commande renvoie 0 si la condition est vérifiée et une valeur différente de 0 sinon. Dans la deuxième syntaxe, il faut faire attention à mettre des espaces entre les crochets et la condition.
Voici les principales conditions:

- `-d fichier`: vrai si fichier est un répertoire (et si il existe),
- `-e fichier`: vrai si le fichier existe,
- `-f fichier`: vrai si fichier est un fichier ordinaire (et si il existe),
- `-r fichier`: vrai si l'utilisateur a le droit d'accès en lecture sur fichier (s'il existe),
- `-w fichier`: vrai si l'utilisateur a le droit d'accès en écriture sur fichier (s'il existe),
- `-x fichier`: vrai si l'utilisateur a le droit d'accès en exécution sur le fichier (s'il existe),
- `-c fichier`: vrai si le fichier est un fichier caractère spécial (et si il existe),
- `-b fichier`: vrai si le fichier est un fichier bloc (et si il existe),
- `-s fichier`: vrai si fichier n'est pas vide,
- `-L fichier`: vrai si fichier est un lien symbolique,
- `fichier1 -nt fichier2`: si fichier a été modifié plus récemment que fichier2,
- `-n chaîne`: vrai si chaîne n'est pas vide,
- `-z chaîne`: vrai si chaîne est vide,
- `nb1 eq nb2`: vrai si les deux nombres entiers nb1 et nb2 sont égaux, on peut remplacer eq par ne (différent), gt (supérieur), ge (supérieur ou égal), lt (inférieur), le (inférieur ou égal).
- `chaine1 = chaine2`: vrai si les deux chaînes sont identiques,
- `chaine1 != chaine2`: vrai si les deux chaînes ne sont pas identiques,
- `-a`: opérateur logique et,
- `-o`: opérateur logique ou,
- `!`: négation logique

Utilisation:

machine $ test -f fichier && echo fichier || echo pas fichier

machine $ test \(-d rep -o -f fic \) -a $variable=2 && echo ok || echo non ok

2. Décaler la position des paramètres:

Dans un shellscript, nous savons récupérer, à l'aide des variables spéciales, le contenu des paramètres passés. Il est parfois utile de pouvoir mettre ces paramètres dans un ordre voulu. La commande:

```
shift [n]
```

permet de déclarer les paramètres de n positions. Par défaut, la valeur de n est 1, dans ce cas là, $2 devient $1, $3 devient $2 etc...

3. Sortie d'un shellscript:

Pour sortir d'un shellscript et renvoyer un code de retour, utiliser la commande:

```
exit n
```

le code de retour est alors n. Si le shellscript ne se termine pas par un exit, il renvoie le code de retour de la dernière commande exécutée.

4. Les structures de contrôle:

Nous venons de voir que toutes les commandes Unix renvoient un code qui est un nombre entier. En général, si la commande s'est bien déroulée, elle renvoie 0, sinon elle renvoie un nombre négatif ou positif.
Comme dans tout langage de programmation, il y a dans le shell Unix un certain nombre de fonctions qui assurent le contrôle des données et des paramètres:

4.1 if..then..else..fi:

La commande:

```
if test
then liste_de_commandes
else liste_de_commandes
fi
```

teste la valeur renvoyée par le test après le mot if et en fonction de ce code, lance la liste de commande appropriée (si le code=0 alors on exécute les commandes après le mot then sinon les commandes après le mot else).

Utilisation: programme qui supprime interactivement selon la réponse

```
machine $ echo -n "Suppression interactive ?"
read reponse
if test "$reponse" = oui
then rm -i $@
else rm $@
fi
```

Il existe une variante pratique de if:

```
if condition
 then ligne_commande
 elif condition2
  then ligne_commande
  ...
 fi
fi
```

4.2 case..esac:

Nous avons aussi besoin de structures permettant de faire des chois multiples:

```
case mot in
 modèle 1) liste_commande;;
 ..
 modèle n) liste_commande;;
 *) liste_commande;;
esac
```

Le premier modèle rencontré qui répond à la valeur de mot indique la liste de commandes à exécuter. Si mot ne correspond à aucun modèle, l'instruction se poursuit jusqu'à la condition *) et si celle-ci n'existe pas, jusqu'au esac.
Les modèles sont de la forme:

```
valeur1 | valeur2 | ... | valeurn
```

Dans les modèles, on peut utiliser les caractères spéciaux *, ?, [] comme dans le cas de la recherche d'un fichier.

Utilisation: programme qui supprime interactivement selon la réponse

```
machine $ echo -n "Suppression interactive ?"
read reponse
case $reponse in
 o|O|oui|OUI) rm -i $@;;
 n|N|non|NON) rm $@;;
 *) echo "Réponse non valide"
    -- rappel du shellscript;;
esac
```

4.3 for..do..done:

Certains traitement sont répétitifs, une commande doit être lancée pour un grand nombre de paramètres du shellscript. Ceci nécessite l'usage d'une fonction qui boucle sur tous ses paramètres. La commande:

```
for variable [in mots...]
 do
   liste_commandes
 done
```

variable prend tour à tour toutes les valeurs de la liste des mots (qui suivant in). Pour chacune de ces valeurs, la liste de commandes est exécutée. Si «in mots...» n'est pas présent, variable prend toutes les valeurs du script, ce qui est équivalent à «for variable in $@».

Utilisation: programme qui recopie tous les fichiers *.c du répertoire courant dans une autre (afin d'en faire la sauvegarde)

```
machine $ for i in *.c
> do
>   cp $i rep/
> done
```

4.4 while..do..done:

La commande:

```
while liste_de_commandes1
 do
      liste_de_commandes2
 done
```

exécute la liste de commandes 2 tant que le code retour de la liste de commandes 1 est 0.

Utilisation: programme qui affiche tous les paramètres d'un script (un par ligne)

```
machine $ while test $1
> do
>   echo $1
>   shift
> done
```

4.5 until..do..done:

Cette commande est semblable à la précédente à la différence près que la sortie de la boucle s'effectue si la liste de commandes qui suit until renvoie 0.
Syntaxe de la commande:

```
until liste_de_commandes1
  do
      liste_de_commandes2
  done
```

4.6 Instructions liées aux boucles:

A l'intérieur de ces différentes structures, il est important que l'utilisateur puisse fixer lui-même et selon ses propres critères la sortie de boucles.
La commande:

```
continue [n]
```

permet de sauter à la fin de la boucle et de recommencer une nouvelle boucle.
La commande:

```
break [n]
```

provoque, quand à elle la sortie de la boucle en cours et passe à l'instruction qui suit la boucle.
Dans les deux cas, l'entier n permet d'indiquer que l'action se porte sur une boucle externe. Par exemple, break 2 sortira de la boucle qui englobe la boucle dans laquelle cette instruction est écrite.

5. Commandes «built-in» diverses:

Un certain nombre de commandes sont contenues dans le shell lui-même. Ce sont des mots réservés. En voici une liste succincte:

- `cd`: permet de changer de répertoire courant,
- `eval chaînes...`: interprète les chaînes et les exécute comme si elles avaient été tapées au clavier. Par exemple: eval dernier_paramètre='$'{$#} affecte à la variable dernier_paramètre le numéro de la dernière valeur passée au shellscript,
- `exit`: termine le shell courant,
- `exec commande`: la commande est exécutée dans le shell courant; le sortie de cette commande est la même que celle du shell courant, on ne revient donc jamais d'un exec,
- `export`: exporte les variables dans l'environnement,
- `read`: lit la ligne sur l'entrée standard et affecte les variables données en arguments,
- `readonly`: les variables données sont positionnées mais non modifiables,
- `set`: permet de modifier les options du shell courant (voir plus loin),
- `shift`: décale d'un cran vers la gauche la liste des paramètres,
- `trap [liste_de_commandes] n...`: spécifie le comportement du shell en fonction des interruptions, la liste de commandes sera exécutée si le programme reçoit un des signaux dont le numéro est n; si n égal 0 alors la liste de commandes sera exécutée à la sortie du shell en cours,
- `umask`: permet de changer le masque de création par défaut des autorisations sur un fichier,
- `wait [n]`: attends la fin d'une commande lancée de manière asynchrone, l'entier n spécifie le numéro du processus père de celui que wait attends,
- `:`: cette commande ne fait rien mais évalue ses arguments, elle renvoie le code 0 ce qui permet de créer des boucles infinies,
- `.`: exécute le fichier donné en argument dans le shell courant

La commande:

```
set [options]
```

permet de déterminer les caractéristiques des variables d'environnement du shell courant et de ses descendants.
Plusieurs options sont possibles, les plus utilisées sont:

- `-a`: exporte automatiquement toutes les variables qui sont définies ou modifiées ultérieurement,
- `-e`: sort immédiatement si une commande retourne un statut différent de 0,
- `-f`: supprime la génération des noms de fichiers,
- `-h`: localise et mémorise les commandes «fonctions» au moment de leur définition (en général, les fonctions sont localisées au moment de leur exécution),
- `-k`: tous les arguments sont chargés dans l'environnement, et pas seulement celles que précède le nom de la commande,
- `-n`: lit les commandes mais ne les exécute pas, il en vérifie simplement la syntaxe,
- `-t`: sort après exécution de la commande,
- `-u`: traite les variables non fixées comme des erreurs au moment de la substitution,
- `-v`: pour faire afficher les lignes d'un shellscript au moment où elles sont lues par le shell,
- `-x`: affiche chaque commande et ses arguments précédés par le signe +,
- `+option`: pour inhiber la commande set -option précédente.,

6. Calculs, traitements des chaînes de caractères:

Nous allons voir dans cette partie plus en détail la commande:

```
expr arguments...
```

Cette commande évalue les arguments comme une expression. Le résultat est envoyé sur la sortie standard. Ce n'est pas une commande interne au shell.
arguments est une expression comprenant des opérateurs. Les opérateurs suivant sont classés par ordre de priorité croissante et regroupés par groupes d'égale priorité:

- exp1 \| exp2: retourne exp1 si elle est non nulle et non vide, sinon retourne exp2 si exp2 est non nulle et non vide, sinon retourne 0;
- exp1 \& exp2: retourne exp1 si aucune des expression n'est vide ou nulle, 0 sinon,
- exp1 op_comp exp2: où op_comp est un opérateur de comparaison (<, <=, =, !=, >=, >); retourne 0 si vrai, 1 sinon,
- exp1 + exp2: addition,
- exp1 - exp2: soustraction,
- exp1 * exp2 : multiplication,
- exp1 / exp2: division,
- exp1 \% exp2: modulo,
- chaîne: renvoie chaîne sur la sortie standard si la chaîne est non vide, 0 sinon,
- chaîne : expression-régulière: renvoie le nombre de caractères de chaîne qui peuvent être désignés par l'expression régulière.

Utilisation:

programme qui incrémente la valeur d'une variable de 1:

```
machine $ a=‘expr $a +1‘
```

affichage des nombres entiers de 1 à 10:

```
machine $ a=1
machine $ while [ $a -lt 10 ]
> do
>       echo $a
>       a=‘expr $a + 1‘
> done
```

affichage de la valeur du paramètre précédé d'un - dans un script:

```
machine $ expr $fichier : ’-\(.*\)’
```

Chapitre 6: Manipulation des données d'un fichier

1. Trier les lignes d'un fichier:

La commande:

```
sort [options] [fichiers...]
```

trie les lignes des fichiers, la sortie se faisant sur la sortie standard.
Le tri s'effectue sur une ou plusieurs clés extraites de chaque ligne. Voici les principales options qui vont permettre de réaliser ce tri:

- `+pos1 -pos2`: permet d'indiquer le début et la fin d'un clé de tri; celle-ci peut être composée de plusieurs zones de tri concaténées. Le tri commence au champs pos1+1 et se termine au champ pos2+1. S'il n'y a pas de pos2, la zone de tri va jusqu'à la fin de la ligne,
- `-t séparateur`: indique le caractère qui sépare les différents champs. Si un séparateur n'est pas défini, les champs sont séparés par toute suite d'espace ou de tabulations contigus,
- `-b`: ignore les espaces et les tabulations de tête dans chaque champ,
- `-n`: effectue un tri numérique sur les champs,
- `-f`: ignore la différence entre majuscules et minuscules,
- `-c`: affiche un message si les lignes ne sont pas dans le bon ordre,
- `-o fichier`: sort le résultat du tri dans fichier (attention sort fich > fich détruit fich avant de le trier, préférer cette option à toute redirection!),
- `-r`: inverse l'ordre de tri,
- `-u`: ne garde qu'une ligne par clé...

Utilisation:

machine $ sort -t: +2 -3 /etc/passwd
tri le fichier /etc/passwd par numéro d'utilisateur

machine $ ls -l / | sort -b +2 -3 +7
tri de la liste des fichiers sous la racine par propriétaire (+2 -3) et par nom de fichier

2. Recherche d'une chaîne de caractères:

La commande:

```
grep [option] expression-régulière [fichiers...]
```

affiche à l'écran les lignes des fichiers qui contiennent une chaîne de caractères correspondant à l'expression régulière donnée. Cette commande est souvent utilisée après un pipe.
Les options les plus courantes sont:

- `-c`: qui affiche seulement le nombre de lignes contenant la chaîne,
- `-i`: qui ignore la différence entre minuscules et majuscules,
- `-n`: qui affiche les numéros de lignes.

Utilisation:

machine $ ls -l | grep *.c
affiche tous les fichiers .c du répertoire courant

machine $ grep -i mot *
affiche le nom du fichier et la ligne contenant le mot

3. Compter les caractères, les mots, les lignes:

La commande:

```
wc [options] [fichiers...]
```

permet de compter le nombre de lignes, de mots et de caractères de chaque fichier. Si des fichiers ne sont pas spécifiés, l'entrée standard est prise par défaut.
Les options sont:

- `-l`: pour afficher seulement le nombre de lignes,
- `-w`: pour afficher seulement le nombre de mots,
- `-c`: pour afficher seulement le nombre de caractères.

4. Conversion et suppression de caractères:

La commande:

```
tr [-d] chaîne1 chaîne2
```

transforme les caractères provenant de l'entrée standard comme suit: les caractères contenus dans chaîne1 sont remplacés par ceux correspondant dans chaîne2, les autres étant conservés tels quels. Le tout est envoyé ensuite vers la sortie standard.
L'option -d, permet de supprimer les caractères qui ne sont pas compris dans chaîne1

Utilisation:

machine $ tr aeiou AEIOU
bonjour
(Ctrl D)
bOnjOUr

machine $ tr -d ac
abcad
(Ctrl D)
bd

5. Comparaison du contenu de 2 fichiers:

Il existe deux moyens de comparer un fichier, soit la comparaison réelle, soit la différence. Ces commandes peuvent être utilisées dans des shellscripts, pour effectuer une réelle comparaison en vue de modification, préférez les outils livrés avec les éditeurs emacs ou xemacs.

5.1 Comparaison de 2 fichiers:

La commande:

```
cmp [-s] fichier1 fichier2
```

affiche le premier octet différent dans deux fichiers. L'option -s permet d'avoir un code retour plutôt qu'un affichage. Le code est 0 si les deux fichiers sont identiques, 1 s'ils sont différents et 2 s'il y a eu une erreur.

5.2 Différence entre 2 fichiers:

La commande:

```
diff [-e] fichier1 fichier2
```

indique quelle ligne doivent être changées dans fichier1 pour qu'il soit identique à fichier2. L'option -e affiche les différences sous une forme semblable aux commandes de l'éditeur ed.

6. Extraire le début ou la fin d'un fichier:

6.1 Début du fichier:

La commande:

```
head [-n] [fichiers...]
```

affiche sur la sortie standard les n premières lignes de chacun des fichiers. Par défaut, la valeur de n est fixée à 10.

6.2 Fin du fichier:

La commande:

```
tail [début] [fichier]
```

affiche la fin du fichier sur la sortie standard. La première ligne affichée est indiquée par début qui peut être du type:

- `+n`: nième ligne du fichier,
- `-n`: nième ligne à partir de la fin du fichier.

7. Autres commandes utiles:

Les commandes suivantes peuvent aussi vous être utiles:

- `split`: fractionnement horizontal d'un fichier,
- `cut`: fractionnement vertical d'un fichier,
- `paste`: recollement vertical de plusieurs fichiers,
- `uniq`: sélection/rejet de lignes consécutives en double,
- `comm`: sélection/rejet de lignes communes à deux fichiers,
- `join`: jointure (relationnelle) de 2 fichiers sur une zone commune,
- `od`: visualisation des octets d'un fichier,
- `dd`: copie d'un fichier avec conversion...

Chapitre 7: Le courrier électronique - Le mail

Sous Unix, il existe un outil qui va vous permettre d'échanger du courrier avec les autres utilisateurs. Les messages que vous recevez sont enregistrés dans votre boîte aux lettres (fichier /var/spool/mail/user). Nous allons dans cette partie étudier le fonctionnement du programme xmh utilisé au MBDS comme interface utilisateur au mail.

1. Présentation:

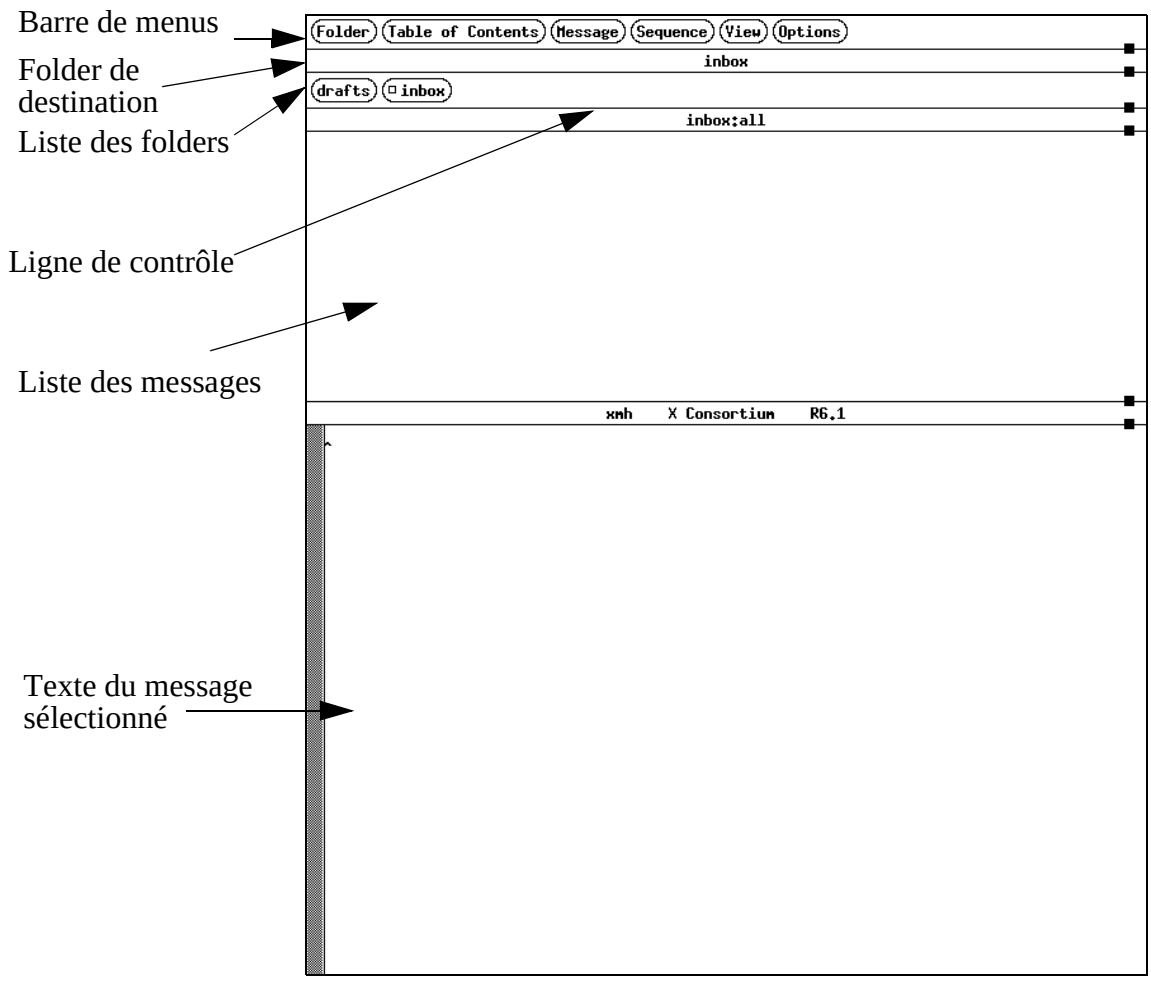
Nous allons voir comment ce présente cet outil, son mode de fonctionnement et les fichiers qui lui appartiennent.

1.1 Lancement:

Le programme xmh ne fonctionne qu'en mode graphique (sous X-Windows). Pour le lancer, placez votre curseur dans une fenêtre de commande et tapez:

```
xmh &
```

La fenêtre suivante apparaît à l'écran:



1.2 Reconnaissance par le serveur de messages:

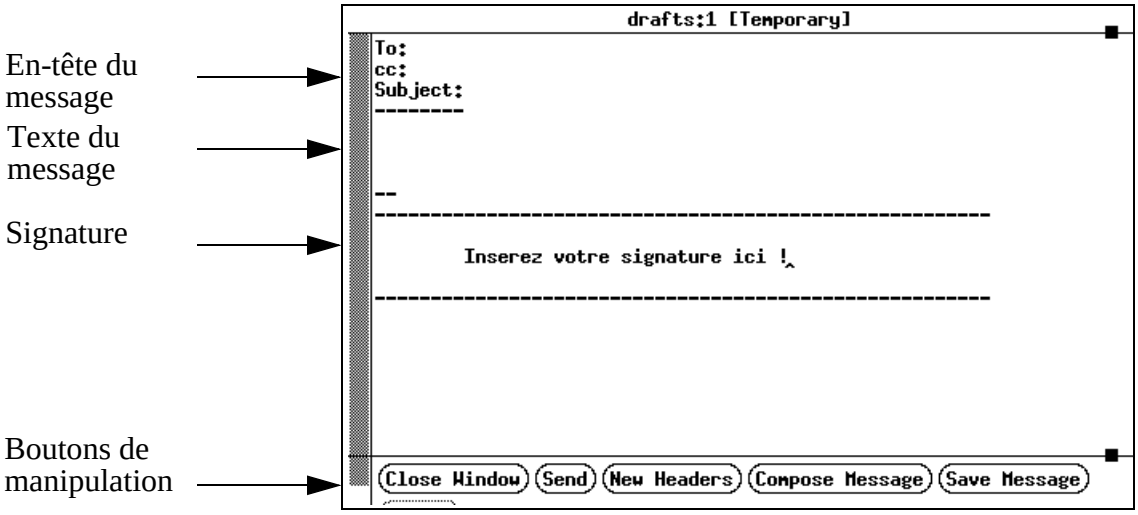
La première chose à faire lors de l'apparition de cette fenêtre est de se faire reconnaître par le serveur de messagerie. Pour cella, aller dans le menu «Options» de la barre de menus, puis sélectionner l'entrée «Set passwd». Une fenêtre s'ouvre où vous devez taper votre mot de passe (le même qui vous a servi à entrer dans le système). Attention, cette opération nécessite que le curseur de la souris soit placé dans la fenêtre elle-même. Lorsque ceci est fait, vous pouvez manipuler vos messages.

1.3 Fonctionnement:

Lors de la première ouverture de xmh, le programme crée dans votre répertoire courant un répertoire Mail ainsi que les sous-répertoires inbox et drafts. C'est dans ces répertoires que seront sauvegardés vos messages.
De même, chaque fois que vous créerez un «folder» sorte de classeur dans lequel vous allez pouvoir archiver vos mails, un nouveau répertoire portant le nom du folder sera créé.
L'ensemble des messages que vous allez recevoir portent chacun un numéro. C'est par ce numéro que sont référencés vos messages dans votre arborescence. Bien entendu, tous les messages se trouvant dans un folder donné se retrouvent dans le répertoire correspondant dans votre arborescence.

2. Envoyer du courrier:

Pour envoyer du courrier, rien de plus simple, choisissez dans le menu «Message», l'option «Compose Message». Une nouvelle fenêtre apparaît:



La ligne To: vous permet d'indiquer le ou les destinataires du message. Chaque nom doit être séparé par une virgule du nom précédent.
La ligne cc: vous permet d'indiquer de la même façon la liste des personnes à mettre en copie.
La ligne Subject: vous permet d'indiquer le sujet de votre message.
Tapez ensuite votre message dans la partie réservée à cet effet.
Insérez votre signature à la fin de votre mail (en général, vos noms, prénoms, fonctions et diverses informations sur le lieu de votre travail).
Choisissez ensuite, le bouton «Send» pour envoyer votre message et «Close Window» pour en terminer.

Il est possible de paramétrer l'affichage de la signature directement sans avoir à la rentrer à chaque fois. Pour cela, éditer le fichier components du répertoire Mail en insérer la signature à l'endroit souhaité.
Si vous souhaitez sauver le message que vous venez d'envoyer, deux solutions:

- se mettre en copie du mail,
- cliquer sur le bouton «Save Message» et celui-ci sera automatiquement sauvé dans le folder drafts.

Petit rappel: pour envoyer un message à un utilisateur, taper son adresse e-mail composée de son nom de login et du nom de la machine ou du domaine auquel il appartient. Par exemple, pour un utilisateur du MBDS, l'adresse e-mail est du type:

```
login@mbds.ceram.fr
```

3. Récupérer son courrier:

Lorsque vous avez lancé le programme, vous avez indiqué votre mot de passe afin d'être authentifié par le serveur de messagerie. Pour consulter vos messages, choisissez dans le menu «Table Of Contents» l'entrée «Incorporate New Mail». Cette opération ramènera votre courrier dans votre répertoire. Vous pourrez ainsi le consulter.
Les nouveaux messages apparaissent à la suite des autres dans la fenêtre. Cliquer avec le bouton du milieu de votre souris sur le message que vous souhaitez visualiser pour en faire apparaître le contenu dans la partie réservée à cet effet. Une barre de défilement sur la gauche de la fenêtre vous permettant de visualiser la totalité du message.

4. Manipulation des messages:

Vous avez maintenant un certain nombre de messages dans votre boite. Plusieurs possibilités s'offrent à vous:

4.1 Réponse à un courrier:

Vous souhaitez répondre au courrier qui vous a été envoyé, en récupérant une partie du message. Choisissez l'option «Reply» du menu «Message». Une fenêtre du même type que celle servant à envoyer un message apparaît. Les champs To:, cc: et Subject: sont déjà remplis et une partie du message est affichée sur l'écran. Formulez votre réponse et de la même manière que précédemment envoyez votre message.

4.2 Mise en copie d'un message reçu:

Vous souhaitez envoyer un message que vous avez reçu à une autre personne. Pour cela, sélectionnez l'entrée «Forward» du menu «Message», une fenêtre apparaît contenant l'en-tête que vous devez remplir (champs To: ,cc: et Subject:) et le message que vous souhaitez faire passer. De la même manière que précédemment, complétez le message et envoyez le.

4.3 Rangement dans des «folders»:

Vous avez dans le folder «inbox», un certain nombre de messages. Parmi ceux-ci, certains doivent être archivés. Pour se faire, vous allez utiliser les folders.
Pour créer un folder, choisissez l'entrée «Create Folder» du menu «Folder», indiquez le nom du folder (attention, ce nom doit être pris comme un nom de fichier puisqu'un répertoire portant ce nom sera justement créé dans le répertoire Mail).
Pour ranger un message dans le dossier de votre choix, sélectionnez le message (en cliquant sur le bouton du milieu de votre souris), puis cliquez avec le troisième bouton sur le folder dans lequel doit aller le message. Un «^» apparaît derrière le numéro du message.
Pour valider votre manipulation, sélectionnez l'entrée «Commit Changes» du menu «Table of Contents». Le message disparaît de la liste des messages du folder «inbox» et est placé dans le folder destination. De même, au niveau de l'arborescence, votre fichier est déplacé du répertoire source vers le répertoire destination.

4.4 Destruction:

De même, pour détruire un message, sélectionnez l'option «Delete» du menu «Message», un «D» apparaît après le numéro du message. De la même façon, validez votre opération.

4.5 Annulation d'une opération:

Au cas où une fausse manipulation a été effectuée, avant de l'avoir validé, vous avez la possibilité de l'annuler. Pour cela, placez vous sur le message, et sélectionnez l'option «Unmark» du menu «Message», ce message sera «démarqué» et ne subira donc aucune modification lors de la validation finale.

4.6 Impression:

Pour imprimer un message, deux possibilités s'offrent à vous:

- sélectionnez l'option «Print» du menu «Message»,
- si cette dernière ne fonctionne pas, imprimez le fichier qui se trouve dans votre arborescence et qui porte le numéro du message via les commandes habituelles d'impression sous Unix.

4.7 Autres fonctionnalités:

D'autres fonctionnalités sont présentes:

- Au niveau du menu «Folder»:
  - vous pouvez sélectionner un folder et consulter les messages qu'il contient (cliquez avec le bouton gauche sur le folder, puis choisissez l'option «Open Folder»),
  - vous pouvez détruire un folder et tous les messages qu'il contient.
- Au niveau du menu «Table Of Contents»:
  - vous pouvez remettre à jour les numéros des messages (option «Pack»),
  - trier vos messages par date d'envoi (option «Sort»)
  - effectuer un nouveau scan de votre folder (option «Rescan Folder»).
- Au niveau du menu «Message»:
  - vous pouvez vous déplacer dans la liste des messages à l'aide des différentes options,
  - déplacer les messages d'un folder vers un autre
  - ...

Table des Matières