Outils et Utilitaires pour UNIX

Catherine Lamy
E.P.S.I Formation par alternance, Paris, France

17 décembre 1997

1 Présentation d'UNIX

1.1 Historique

En 1965, les Bell Laboratories (appartenant a AT&T), la General Electric Company et le MIT (Massachusetts Institute for Technology) se regroupèrent en vue de développer de projet MULTICS. Il s'agissait de créer un environnement interactif et adapté au développement sur de grosses machines. Trois ans plus tard, il était devenu clair que ce projet ne pourrait être mené à son terme et les Bell Laboratories se retirèrent donc du projet en 1969. Les idées mûries au cours de cette période ne furent pas pour autant perdues puisque Ken Thompson décida de développer son propre système d'exploitation en prenant cette fois-ci le parti de s'éloigner résolument de tout ce qui existait sur le marché. Les systèmes d'exploitation de l'époque travaillaient selon la technique du "Closed Shop" : les utilisateurs transmettaient leurs paquets de cartes perforées à un opérateur qui, seul, avait accès au calculateur. Il n'était alors pas possible à un utilisateur de communiquer avec l'ordinateur par des commandes interactives. Thompson et ses collborateurs décidèrent de doter leur futur système d'exploitation des caractéristiques suivantes :

– Le système de gestion des fichiers devaient avoir une structure hiérarchique, pour permettre une flexibilité optimale en matière de répartition des données sur le disque dur.

– Les sorties dans des fichiers et celles envoyées aux périphériques devaient être similaires. Les processus devaient également pouvoir communiquer entre eux, comme s'il s'agissait d'entrées ou de sorties d'un fichier.

– Il devait être possible de lancer de nouveaux processus et ceux-ci devaient pouvoir réaliser leur tâche indépendamment des processus déjà en cours. On parle dans ce cas de processus "asynchrones".

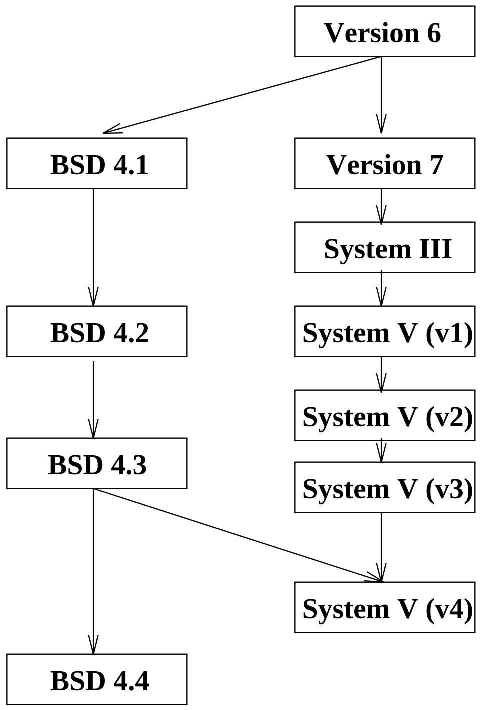
– Chaque utilisateur devait pouvoir choisir son propre interpréteur de commandes, de façon à exploiter au mieux ses compétences et à réagir à ses besoins.

– Le système devait offrir dès le départ une vaste palette d'utilitaires, tout spécialement des compilateurs pour divers langages de programmation.

– Le système d'exploitation devait être aussi portable que possible, pour pouvoir être transféré rapidement et facilement d'un type de machine vers un autre. (Se rappeler qu'à l'époque les disparités entre les machines et les architetures étaient énormes).

La première version du système baptisé UNIX par ironie par rapport à MULTICS fut donc écrite en **assembleur** au cours de l'été 1969. Ce premier système était surtout un système de fichiers exploité dans un environnement mono-utilisateur et multi-tâche. Sa première justification officielle était la réalisation d'un système de traitement de texte dédiés aux secrétariats. Au fil du temps, d'autse fonctionnalités furent implémentées sur UNIX. Jusqu'en 1972, UNIX se composait d'un système de gestion de fichiers et d'un prédécesseur du système de gestion de processus actuel. Y furent intégrés ensuite les pipelines, pour l'échange de données entre les programmes. À cette époque, cette dernière fonctionnalité était celle que le système d'exploitation remplissait le moins. C'est pourquoi Denis Ritchie développa le langage **C** sur la base du langage **B** écrit par Thompson pour le système UNIX en 1970. Il apporte une efficacité plus grande, la possibilité de typer les objets et celle de définir des structures. Une bonne partie du système d'exploitation fut alors réécrite en langage C et seules quelques parties restèrent en assembleur. UNIX mettait ainsi fin à la longue tradition des systèmes d'exploitation écrits en assembleur.

Les principales fonctionnalités du système furent regroupées dans un noyau mis à la disposition des utilisateurs et des programmes utilitaires par l'intermédiaire d'appels système (*Systems Calls*). Ces appels ressemblent à des appels de sous-programmes en langage C. La conversion du noyau en langage C facilita grandement l'implémentation d'UNIX sur d'autres types de machines



En 1974 parut le premier article décrivant UNIX et aussitôt un décret du gouvernement fédéral interdit aux Bell Laboratories de commercialiser UNIX. Le système sera donc seulement distribué (version 3), dans un but éducatif, aux universités (Berkeley et Columbia notamment). Ces utilisateurs ne supportaient pratiquement que les coûts de copie du système, en échange de l'engagement de faire remonter aux Bell Laboratories les messages d'erreur et éventuels problèmes rencontrés dans l'utilisation d'UNIX. Jusqu'en 1975, les développement d'UNIX furent donc réalisés en interne aux Bell Laboratories et l'année 1976 vit la première version à bénéficier d'une diffusion commerciale: Unix système 6. Ce n'est pourtant qu'en 1979 avec la version 7 que le véritable essor d'UNIX commence. Cette version fut en effet retenue par de nombreux constructeurs d'ordinateurs aux États-Unis pour l'architecture de leur matériel (hardware): Hewlett

Pacard, Digital Equipement, PCS .... Certains commencèrent pourtant à se distinguer, comme l'université de Berkeley qui marqua le début d'une carrière parallèle avec le développement d'un dérivé d'UNIX destiné à offrir de nouvelles extensions. Comme le nom d'UNIX était propriété de AT&T, ils l'appelèrent BSD (Berleley Software Distribution) version 4.1. Il en alla de même pour les entreprises citées précédement, ce qui les amena à choisir des noms comportant un 'X' (versions X : HP-UX, DG-UX, Xenix ...). Cette pléthore de dénominations et de numéros de version subsiste encore actuellement.

Les perfectionnements ultérieurs et la diffusion d'UNIX System III ne simplifia pas les choses : malgré les performances et les fonciotnnalités d'UNIX, les constructeurs conituèrent à ne l'utiliser que comme une base et à modifier une foule de détails. On revenait donc au point de départ : la portabilité des applications redevenait problématique au sein même du monde UNIX. Le succès d'UNIX s'en ressenti tout naturellement, ce qui amena AT&T à présenter un standard UNIX : en 1983 apparut Unix System V et la documentation correspondante UNIX System V Interface Definition (SVID, définitivement publié en 1985) où se trouvaient définis les performances et les fonctionnalités qu'un système UNIX "correct" devait offrir à son utilisateur, ainsi que les moyens dont devait disposer cet utilisateur pour accéder à ces éléments. Il s'agissait essentiellement de la description des commandes dont devait impérativement disposer l'utilisateur, mais en aucun cas d'indiquer des normes pour l'implémentation du système : cette définition assure une portabilité d'une machine UNIX à une autre, par simple recompilation de l'application.

N.B.: problèmes de portabilité
au niveau utilisateur :

- ls -lg où le g a le sens inverse
- ps augx VS ps -aef
- df n'a pas la même unité de mesure

au niveau programmation :

- librairie termcap VS librairie terminfo
- signaux : SIGPWR (System V) VS SIGSTOP, SIGCONT (BSD)

Parallèlement se constitua le groupe X/OPEN constitué de cosntructeurs , sociétés de développement indépendantes et de grands utilisateurs d'UNIX. Le but de cette association était de définir des bases générales pour l'ensemble du monde UNIX. Ils produisirent différentes versions d'un X/OPEN portability Guide (XPG) devant servir de référence pour tous les développements ultérieurs d'UNIX. En 1993, X/OPEN lance le COSE (Common Open Software Environment) qui est un ensemble d'accords entre constructeurs permettant le développement d'applications dans un environnement commun. Enfin cette même année, le transfert du nom UNIX à X/OPEN (en 1992 rachat des parts de AT&T dans USL (Unix System Laboratories) par Novell qui appartient au consortium X/OPEN). La mission d'X/OPEN devient donc de certifier la conformité UNIX des systèmes d'exploitation des différents constructeurs.

Lorsque les premiers PC (Personal Computer) virent le jour au début des années 80, UNIX était déjà un produit mûûr et fiable. C'est ce qui fait que l'on retrouve dans MS-DOS un certain nombre de similitudes avec UNIX, tout particulièrement en matière de strcuture de fichiers et d'interpréteur de commandes (ex: jokers utilisés pour les noms de fichiers). Mais cela ne suffisait pas : le système d'exploitation MS-DOS a beaucoup de contraintes liées à son architecture (insuffisance de mémoire adressable, pas de fonction multitâche ou multiutilisateur, pas de mécanisme de protection de la mémoire en cas d'erreur de programmation), et UNIX semblait la voie idéale pour s'en affranchir. Nombre de constructeurs se sont donc penchés sur le problème de savoir si UNIX était portable sur leurs machines. Le premier à avoir essayé fut Microsoft qui proposa le système d'exploitation XENIX en 1984. Dès la génération des 286 ce fut une solution économiquement valable, car rendre une machine déjà peu couteuse accessible à plusieurs utilisateurs ne faisait que renforcer le bon rapport performances/prix. L'intérêt financier du marché d'UNIX sous PC n'échappa pas aux autres constructeurs (plusieurs centaines de millions de PC !) et UNIX fut même concurrencé par d'autres porduits tels que Windows ou OS/2 (IBM). La force de ces autres systèmes était la mise à disposition d'une interface graphique mais le développement du système X-Windows et des terminaux graphiques a nettement réduit l'écart entre ces systèmes. SUN développa même dans sa gamme Solaris une version UNIX (SunOS puis Solaris 3.5 ...)

1.2 La normalisation d'UNIX

Aujourd'hui encore cohabitent différents standards de normalisation UNIX. Rappelons les donc :

- SVID (System V Interface Definition) qui définit l'interface du système V et non son implémentation. Afin de vérifier la conformité des apllications, AT&T développa un ensemble de logiciels appelés SVVS (System V Verification Suite).

- Des utilisateurs voulant exprimer leur volonté de standardisation créèrent aux États-Unis **/usr/group**, organisation qui publia */usr/group standard* décrivant les fonctionalités minimales du système. Cette publication servit de base à IEEE (Institute of Electrical and Electronics Engineers) pour définir une interface dans le même esprit qu'AT&T, laissant l'implémentation tout à fait libre. Cette interface s'appelle POSIX (portable Operating System Interface X) et a reçu l'aval des organismes de standardisation américain ANSI (American National Standar Institute, membre de l'ISO) et international ISO (International Standard Organisation, seul reconnu mondialement par l'ONU).

- X/OPEN est un consortium de constructeurs qui a pour but de définir un environnement commun de développement. Le COSE (Common Open Software Environnement) est suivi par beaucou pde constructeurs, ce qui a pour conséquence d'augmenter le pouvoir d'X/OPEN. C'est à la suite du rachat d"USL par Novell que ce dernier donne le nom d'UNIX à X/OPEN. Unix devient donc un label que seul X/OPEN sera habilité à donner.

- En voulant "développer et proposer un environnement ouvert simplifiant l'utilisation de logiciels et de matériels de marques différentes", IBM et un ensemble de six constructeurs créent OSF (Open Software Foundation).

- En réponse à IBM avec OSF, AT&T et 18 constructeurs et éditeurs, créent ARCHER qui devient UNIX International. S'étant engagé à suivre les spécifications de POSIX, ces deux groupes ne font que mettre en place un ensemble d'extensions.

En juillet 1994, X/OPEN finalise la liste des interfaces de programmation (API). L'organisme apposera son label sur tout système reconnu conforme à ses spécifications. Pour les développeurs, cela rime avec portabilité et facilité de développement. Une seule version source d'un même programme ne nécessite qu'une recompilation pour s'exécuter sur une autres plateforme respectant les spécifications.

1.3 Ce qu'il faut retenir

- Écrit dans un langage de haut niveau, son code est facile à lire et à modifier. De plus ce code source est disponible commercialement.
- Son interface utilisateur est la plus impliste qui soit. elle compense son manque de convivialité par sa puissance.
- Le système est construit sur un petit nombre de primitives de base qui favorisent la modularité. En effet, la puissance d'un système vient surtout des combinaisons possibles de relations entre programmes plutôt que de l'excellence de quelque uns.
- Les fichiers ne sont pas structurés au niveau des données. Il est donc possible de les utiliser très simplement. Pas de notion d'accès séquentiel, indexé ... c'est le travail de l'application.
- Toutes les interfaces avec les périphériques sont unifiées par l'intermédiaire du système de fichiers. C'est à dire que les périphériques se trouvent dans l'arborescence avec les autres fichiers.
- Le programmeur n'a jamais à se soucier de l'architecture de la machine sur laquelle il travaille.
- Il est disponible sur une très grande quantité de machines du micro-ordinateur au supercalculateur
- Les utilitaires disponibles en standard et les programmes proposés au catalogue des diverses sociétés de services sont très nombreux et variés.

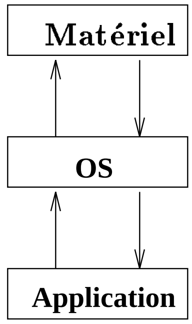
Le système UNIX répartit ses ressources (mémoire, processeur(s), espace disque, imprimante(s), programmes, utilitaires, accès au(x) réseau(x) ...) entre les divers utilisateurs et les diverses tâches. Une sollicitation trop importante du système peut être perçue par les utilisateurs comme un certain ralentissement, mais en principe le système ne se bloque pas.

2 Concepts de base

2.1 Le système d'exploitation

Un système d'exploitation est un ensemble de programmes et de sous-programmes qui ont pour rôle de gérer, de piloter le matériel, pris dans son sens le plus large. C'est l'intermédiaire entre les applications et le matériel, c'est lui qui intercepte les demandes des applications et les transmet au matériel. Un système d'exploitation digne de ce nom n'acceptera jamais qu'une application fasse appel directement au matériel: c'est à lui de trier les commandes et de dépister les erreurs de manipulation ou les commandes absurdes.

UNIX met ses fonctionalités au service des applications sous foirme d'appels sytème. Ces appels système ressemblent à des appels de sous-programmes en langage C. Au sein du système d'exploitation, les gestionnaires de périphériques établissent le contact direct avec le matériel. Dans la situation idéale, lors d'un changement de matériel, il suffit alors de changer les gestionnaires pour que tout fonctionne de nouveau comme il se doit.



De part sa position d'intermédiaire entre les demandes des applications et le matériel, le système d'exploitation doit remplir plusieurs conditions :

– Il doit gérer la mémoire de manière à ce qu'aucun programme n'influe directement sur cette mémoire. Sous UNIX, il n'existe pas de solution simple pour réserver une partie de la mémoire disponible pour une application particulière.

– Il doit intercepter tous les accès à des périphériques. Ainsi, aucun programme ne peut accéder directement à une interface série ou à une unité de disquette. L'accès passe par des fichiers de périphérique qui sont abordés par les applications comme des fichiers tout à fait traditionnels. Il est donc possible de faire abstraction des différences entre les périphériques au niveau des programmes applicatifs.

– Il doit gérer correctement la mémoire disque. Il doit donc fournir un mécanisme sûr et efficace d'enregistrement des données sous forme de fichiers sur le disque dur. Il lui faudra ensuite pouvoir retrouver facilement et rapidement ces fichiers, même si ils sont parmi de nombreux autres. En l'occurence, UNIX offre un système de gestion de fichiers hiérarchique sur lequel nous reviendrons.

– Il doit offrir aux diverses applications et/ou utilisateurs une gestion correcte des programmes. Il s'agit en particulier de permettre aux divers programmes de fonctionner de manière totalement indépendante. En l'occurence, UNIX appelle ses programmes en mémoire des processus comme nous le verrons plus tard.

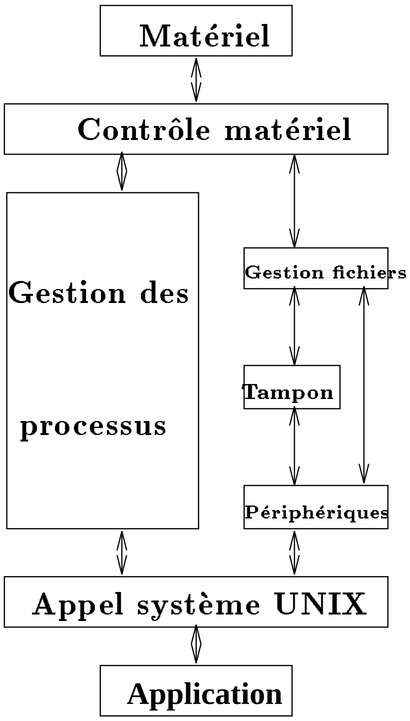
– Il devra protéger les données enregistrées contre tout accès non autorisé. Pour cela, il devra être en mesure d'affecter aux fichiers des autorisations diverses, modifiables par l'utilisateur. UNIX répartit ainsi les droits d'accès sur un fichier en trois catégories : le propriétaire du fichier, les utilisateurs de son groupe principal, tout autre utilisateur.

– Il doit pouvoir relever des informations statistiques sur les programmes utilisés ou en cours d'utilisation et les fournir à la demande, par exemple afin de répartir les coûts du système équitablement entre les divers utilisateurs.

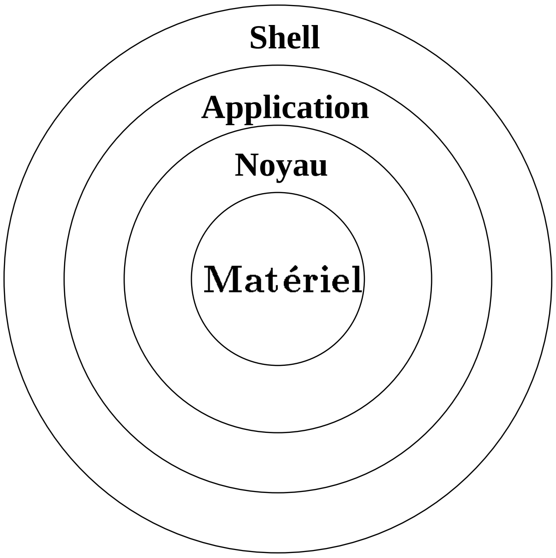
Le noyau du système UNIX est un bloc monolithique apte à réaliser une partie des tâches précédemment mentionnées. Du point de vue de l'utilisateur, il n'existe que des appels système. Ces appels dirigent les demandes (par exmple la lecture d'un fichier ou le lancement d'un programme) vers un sous-système de gestion de fichiers ou de gestion de processus.

Le sous-système de fichiers se charge de la gestion des fichiers en vérifiant notamment les droits d'accès. Le sous-système de processus affecte à chaque programme une zone de mémoire et contrôle les demandes de mémoire supplémetaire émises par les programmes. Il répartit également (par le module Scheduler) la mémoire centrale entre toutes les applications actives.

Tout ceci nous amène donc à avoir une vision plus précise du système d'exploitation UNIX :



Pour travailler avec UNIX, il est important de savoir qu'il est possible de communiquer directement avec le noyau du système d'exploitation. Cette communication passe par un utilitaire qui sert d'intermédiaire entre la saisie des commandes et le noyau du système. On l'appelle **shell** (en français : coquille)



Le noyau seul ne sert pas à grand chose : il lui faut un certain nombre d'utilitaires et de commandes indispensables à son bon fonctionnement. C'est ce que nous regroupons dans la notion d' "UNIX de base".

- Le noyau
  - gestion de la mémoire
  - gestion des entrées-sorties de bas niveau
  - enchaînement des différentes tâches
- Les outils de base
  - langage de commande (**shell**)
  - commandes permettant la manipulation des fichiers
  - commandes permettant la festion des activités du système (procesus)
  - commandes permettant la communication entre les utilisateurs ou les systèmes

À tout cela s'ajoutent un certain nombre d'autres d'utilitaires, faisant d'UNIX un système de développement :

- éditeurs de texte
- outils de traitement de texte
- compilateurs de langage (ex : C)
- outils généraux de développement : gestionnaires de sources, débogueurs ...

2.2 Unix et les utilisateurs : ouvrir un session de travail

Pour pouvoir se connecter sur un système UNIX, il faut y posséder un compte, c'est-à-dire être un utilisateur déclaré du système en question. À chaque utilisateur est associé un numéro d'identification (**uid** user identification), mais également un groupe principal d'utilisateurs (**gid** group identification). Il peut alors être identifié sur le système par une chaîne de caractères (généralement dérivée de son nom) qui constitue son nom d'utilisateur ou **login name** grâce à la saisie d'un mot de passe crypté (**passwd**). C'est ce principe qui permet d'attribuer à chaque objet du système un utilisateur propriétaire et donc des droits d'accès fonction de ce propriétaire, du groupe (des groupes) d'utilisateurs dont il fait partie ...

Lorsque l'on possède un nom d'utilisateur et le mot de passe correspondant, on peut démarrer une session de travail. Celle-ci se déroule en trois étapes :

- se connecter ou se "logguer"
- travailler
- terminer la session : se déconnecter ou se "délogguer"

2.2.1 Se logguer

Considérons un exemple de connexion sur une machine UNIX :

```
login:
```

Il s'agit ici pour l'utilisateur de rentrer le nom par lequel il est identifié sur le système :

```
login:joseph
```

Apparaît alors une seconde ligne : il s'agit de la demande du mot de passe, qui n'apparaîtra pas à l'écran afin de préserver au mieux la confidentialité.

```
Password:
```

Ces deux données ayant été saisies, la procédure de login va vérifier l'identité et le mot de passe de l'utilisateur dans le fichier `/etc/passwd`. Ce dernier contient également les informations nécessaires au système pour la gestion de l'environnement de l'utilisateur :

```
joseph:HJNguiofdnIK:101234:101:Joseph, poste.2222:/users/joseph/:/bin/csh
```

soit :

- login : identificateur de l'utilisateur
- password : mot de passe codé
- uid : valeur numérique correspondant au nom de login
- gid : valeur numérique correspondant au groupe de l'utilisateur
- commentaire : renseignements sur l'utilisateur
- PATH : chemin d'accès au répertoire de l'utilisateur
- shell : nom du shell lancé à la connexion

Si votre nom et votre mot de passe sont corrects, le système lance alors votre interpréteur de commande (shell) qui affiche alors la chaîne d'invite (**prompt**) indiquant que le système a créé un processus shell interactif.

```
<joseph@merteuil: 1>
```

De manière générale, chaque fois qu'il réapparait, ce prompt indique que le processus shell est prêt à analyser une nouvelle commande.

L'utilisateur normal se connecte en général sur un seul compte : le sien. La personne responsable de la gestion du système se connecte elle aussi sur son compte personnel, mais elle peut également se connecter sur des comptes spéciaux à fin d'exécuter des tâches spécifiques (ex : sauvegarde). Il existe ainsi un compte spécial dont le nom est en général **root** et que l'on appelle super utilisateur (super user) car il permet d'accéder à tous les fichiers et d'exécuter toutes les manipulations systèmes.

2.2.2 Travailler

L'utilisateur est à présent à même de fournir des données au shell qui lit la chaîne d'entrée et l'interpréte comme une commande dès que la touche `ENTER` a été pressée.

2.2.3 Se délogguer

Il faut fermer le fichier d'entrée de l'interpréteur shell afin de ne pas rester connecté en partant (sécurité) et laisser la place libre pour un prochain utilisateur :

```
<joseph@merteuil: 2>exit
```

Réapparait alors le message initial :

```
login:
```

Taper `^D` est également un moyen de terminer une session. En effet, ce caractère simule une fin de fichier, et lorsque le shell rencontre cette fin de fichier, il se termine. En revanche, il faut bien savoir qu'éteindre la machine n'est pas la solution.

2.2.4 Utiliser son terminal et quelques commandes de base pour une session

L'interface entre UNIX et les terminaux **tty** (teletype) est prise en charge par un gestionnaire de ligne série (driver tty) qui est paramétrable par l'utilisateur. Pour voir les paramètres du terminal, il suffit de taper la commande **stty**.

```
<joseph@merteuil: 3>stty -a
speed 9600 baud; rows = 14 columns = 94; ypixels = 186 xpixels = 583
intr = ^c; kill = ^u; eof = ^d;
...
```

Pour modifier ces paramètres, on fera : `stty nom-parametre valeur` ou simplement `stty nom-parametre` si ce paramètre est booléen (ex : stty erase '^H).

- **passwd** permet de modifier le mot de passe de l'utilisateur.
- **uname** affiche les informations relatives à la version du système d'exploitation.
- **date** affiche la date et l'heure connues par le système
- **who** affiche la liste des utilisateurs actuellement connectés sur la machine
- **echo** affiche ses arguments (utile pour afficher du texte par exmple, ou certaines constances du système)

```
<joseph@merteuil: 4>uname
SunOS
<joseph@merteuil: 6>date
Sun Dec 15 10:11:22 MET 1997
<joseph@merteuil: 7>who
joseph console Dec 10 09:29
joseph pts/1 Dec 10 09:45
joseph pts/2 Dec 10 09:33
root pts/3 Dec 15 11:03 (andromede)
```

3 Système de gestion de fichiers

3.1 Types de fichiers

Un fichier UNIX est une chaîne de caractères non structurée d'octets (byte stream). Un certain nombre de caractéristiques sont associées à un fichier :

- sa taille
- l'identification de son propriétaire et les droits d'accès pour les différents utilisateurs
- son type (fichier ordinaire ou spécial, répertoire (**directory**)
- date de création, de la dernière modification
- compteur de ses références dans le système
- ...

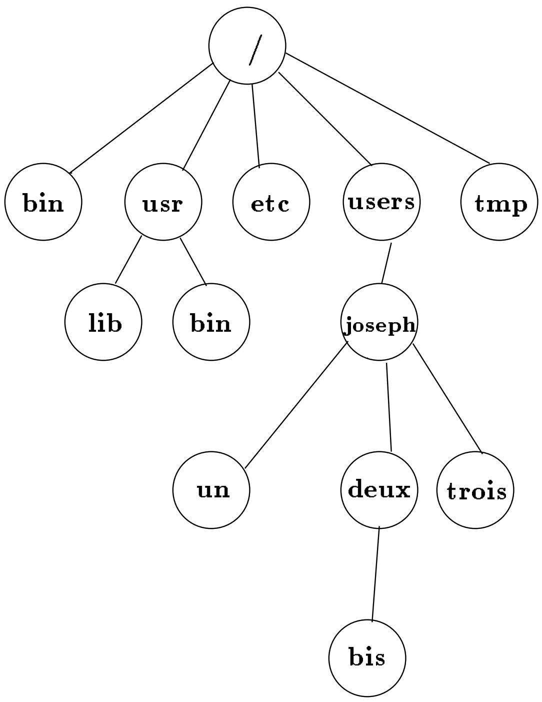
Ces caractéristiques sont regroupées dans un descripteur de fichier appelé *i-noeud* (**i-node**).

Les fichiers normaux contiennent soit des textes en clair (courriers, textes de source des programmes ...), soit un programme exécutable (exécutable provenant d'un compilateur, son, image ...). Dans ce dernier cas, on parle également de fichiers binaires (binary files). Au premier abgord, il n'est pas possible de savoir si un fichier contient du texte ou un programme exécutable. Les répertoires permettent d'organiser l'espace du disque dur. Les fichiers normaux sont ainsi regroupés dans des répertoires qui peuvent eux-mêmes contenir des sous-répertoires. C'est le principe de base d'uns structure hiérarchique. Ces répertoires peuvent également contenir des fichiers spéciaux, soit des interfaces avec les périphériques gérés par le système d'exploitation. Chaque accès en lecture ou écriture sur un périphérique est dirigé directement sur le fichier spécial correspondant. Ces fichiers spéciaux se divisent en deux catégories suivant le mode d'accès : mode bloc pour disques et disquettes, mode caractère pour terminaux et lecteurs de bandes (ce qui est caractérisé par un b ou un c lorsque l'on fait ls -l). Le système de fichiers ressemble à un arbre d'où son nom d'arborescence. Cette arborescence débute systématiquement à la "racine" représentée par / (slash). Les noeuds de l'arbre sont les répertoires, tandis que les feuilles sont les fichiers.

3.2 Noms de chemin et de fichier

La gestion de fichiers sous UNIX repose sur le principe que chaque nom de fichier (référence pour l'utilisateur) est unique : son nom est aussi celui de son chemin d'accès (path name) depuis le répertoire racine. Ce nom est associé à l'i-node défini précédemment. Les noms de fichiers peuvent comporter jusqu'à 255 caractères, et UNIX fait la distinction entre minuscules et majuscules. La plupart des caractères sont autorisés (contre-ex : \ ), cependant il est prudent d'éviter les caractères spéciaux (ex : *).

Pour chaque utilisateur, le système d'exploitation mémorise également le répertoire "actif" (celui dans lequel il se trouve au moment de l'exécution de la commande shell), ce qui lui permettra d'éviter d'avoir à mentionner à chaque opération le chemin d'accès complet au fichier : on parle de nom de chemin relatif. On peut également utiliser le symbole .. qui désigne le répertoire situé juste au dessus du répertoire actif dans l'arborescence UNIX (répertoire père) ou encore le symbole . qui désigne le répertoire courant. Ces deux points font partie intégrante du chemin d'accès et doivent être séparés des autres composantes du nom du chemin par le traditionnel



slash. On notera également que le tilde ˜ remplace le chemn absolu d'accès au répertoire personnel d'un utilisateur

`<joseph@merteuil: 1>pwd`: référence absolue du répertoire actif (courant)
`/users/joseph`
`<joseph@merteuil: 2>cd /usr/bin`: positionnement absolu
`<joseph@merteuil: 3>pwd`
`/usr/bin`
`<joseph@merteuil: 4>who am i`
`joseph pts/2 Dec 15 10:13`
`<joseph@merteuil: 5>cd`: cd sans paramètres: positionnement dans le répertoire personnel
`<joseph@merteuil: 6>pwd`
`/users/joseph`
`<joseph@merteuil: 7>cd deux/bis`: positionnement relatif au catalogue de travail
`<joseph@merteuil: 8>pwd`
`/users/joseph/deux/bis`
`<joseph@merteuil: 9>cd ../../un`: positionnement relatif: .. est le répertoire père
`<joseph@merteuil: 10>pwd`
`/users/joseph/un`
`<joseph@merteuil: 11>cd  /deux`
`<joseph@merteuil: 12>pwd`
`/users/joseph/deux`

3.3 Quelques commandes pour bien utiliser le système de fichiers

La syntaxe générale pour une commade UNIX est :
commande `[option ...] [argument ...]`. Les options précisent la façon dont la commande

doit travailler. Les arguments peuvent être ceux de l'option ou de la commande permettent de passer une argument à l'option.

Pour reconnaître les fichiers des sous-répertoires situés dans la répertoire courant, mais également pour connaître le contenu du répertoire courant, on peut "lister" le contenu dudit répertoire par la commande **ls** (list sorted). Cette commande peut être utilisée seule où avec des options : `ls -F` ajoute par exemple à chaque sous-répertoire un slash qui permet de le différencier des fichiers. On utilisera ls de la manière suivante : `ls [options] [nom de répertoire]`, les options de base étant `-F, -l, -a, -d`. Pour **ls** et de manière générale, les options sont souvent combinables.

```
<joseph@merteuil: 1>pwd
/users/joseph
<joseph@merteuil: 2>ls -F
deux/ trois/ un/
<joseph@merteuil: 3>ls -l
total 54
drwxr-x--- 2 joseph 69 Dec 15 10:14 deux
drwxr-x--- 2 joseph 69 Dec 15 10:14 trois
drwxr-x--- 2 joseph 105 Dec 15 10:14 un
```

`ls -l` affiche de nombreuses informations sur les fichiers. La ligne total donne la taille en blocs (paquet de 512 octets) de toutes les entrées du répertoire, puis on trouve successivement le type du fichier, les droits d'accès, le nombre de liens, le nom de l'utilisateur propriétaire, la taille en octet, la date de dernière modification et le nom du fichier.
Pour éviter d'éventuelles méprises sur le type d'un fichier, vous pouvez de toute façon avoir recours à la commande **file**, qui retourne le contenu probable du ou des fichiers mentionnés.

```
<joseph@merteuil: 4>pwd
/users/joseph
<joseph@merteuil: 5>file deux
deux: directory
<joseph@merteuil: 6>file deux/bis
deux/bis: ascii text
```

Lorsque le fichier est de type texte (texte anglais, texte ascii, texte de commandes) on peut le visualiser par la commande **cat** qui affiche son contenu sur la sortie standard. On peut faire état de plusieurs fichiers, leurs contenus étant alors présentés les uns après les autres, sans aucune séparation. Utilisée sans aucun paramètre, la commande cat affiche à lécran tout ce qui est saisi au clavier, et ceci jusqu'à ce qu'elle soit interrompue par un ^ **D** sur une ligne vide.

```
<joseph@merteuil: 8>pwd
/users/joseph/deux
<joseph@merteuil: 9>cat bis
fichier bis dans le repertoire /users/joseph/deux

coucou
```

D'autres commandes existent qui permettent de visualiser le contenu des fichiers. Par exemple les commandes **more** et **less**.

Il importe de savoir gérer son compte, rangeant les fichiers et les sous-répertoires comme on range des objets dans une maison. Pour organiser les fichiers, le seul moyen valable est de créer des sous-répertoires (fichiers texte, fichiers documentation, programmes, anciens fichiers ...). Pour cela l'utilisateur dipose de la commande **mkdir** (make directory). Dès le départ, un répertoire a déjà une certaine taille car il contient deyx éléments standards (. et ..). Tant qu'un répertoire ne contient pas de fichiers, l'utilisateur peut le supprimer par la commande **rmdir** (remove directory). Il n'est évidement pas possible de supprimer le répertoire courant.

```
<joseph@merteuil: 11>mkdir essai
<joseph@merteuil: 12>ls
deux essai trois un
<joseph@merteuil: 13>rmdir essai
<joseph@merteuil: 14>ls
deux trois un
```

Certains fichiers sont dits "cachés" , c'est-à-dire qu'ils n'apparaissent pas lorsque l'on tape un simple **ls**. Ils ne sont visibles que lorsque l'on utilise l'option **-a**. Ces fichiers sont en général des fichiers de configuration (shell, applications ...), ce qui explique que l'utilisateur ne soit pas gêné par un rappel systématique de leur présence.
Un certain nombre d'autres commandes s'avèrent très utile dans la manipulation des fichiers et des répertoires :

- **touch** qui permet de créer un ou plusieurs fichiers vides (restriction : les noms de doivent pas commencer par un chiffre).
- **cp** qui permet de recopier des fichiers soit dans le meme répertoire sous un autre nom, soit dans un autre répertoire.
- **mv** qui permet de déplacer un fichier et peut également être utilisée pour le renommer. Pour cp et pour mv il convient d'être prudents car par défaut, si le fichier cible existe déjà il sera purement et simplement écrasé sans aucun message d'avertissement.
- **rm** qui permet de supprimer un fichier si on en a le droit. Il est en général prudent d'utiliser `rm -i` qui demande une confirmation pour toute suppression et évite le désagrément de s'apercevoir après coup que l'on a en fait supprimé un fichier que l'on souhaitait conserver. `rm -r répertoire` est une commande très pratique également, car elle supprime un à un tous les fichiers et sous-répertoires du répertoire mentionné dans la commande, et ceci sans aucune demande de confirmation.
- **ln** qui crée une nouvelle référence pour un fichier existant. Physiquement, il n'existe qu'un fichier, mais il a plusieurs noms (même inode cependant comme on peut le voir avec la commande ls -i). Modifier le contenu de l'un revient à modifier celui de l'autre, mais en supprimer un ne supprime pas l'autre puisqu'il existe toujours un nom et que le compteur de liens n'est pas nul. Il est donc possible grâce à cette commande pour l'utilisateur A de créer une référence sur un fichier de l'utilisateur B de manière à pouvoir accéder à ce fichier comme si il se trouvait dans son propre répertoire, même si ce fichier ne lui appartient pas. (ln -s pour deux utilisateurs n'ayant pas leurs répertoires principaux sur le même disque physique sinon message : `ln: essailn: Cross-device link`
- **chmod** permet de changer les droits d'accès à un fichier (lecture, écriture, exécution)

```
<joseph@merteuil: 1>ls -l
-rw-r----- 1 joseph 59 Dec 15 10:33 bis
<joseph@merteuil: 2>ln bis ter
<joseph@merteuil: 3>ls -l
-rw-r----- 2 joseph 59 Dec 15 10:33 bis
-rw-r----- 2 joseph 59 Dec 15 10:33 ter
```

On voit lorsque l'on fait `ls -l` apparaître les droits du(des) fichier(s) considéré(s), et ceci pour les trois groupes de personnes considérées: propriétaire, groupe et autres. Pour chacun d'eux, les droits sont écrit sur trois lettres: rwx correpsondant a read (autorisation de lire le contenu, nécessaire pour copier, mais insuffisant pour faire ls -l qui nécessite aussi x), write (autorisation de modifier le contenu, nécessaire pour détruire des fichiers ou des répertoires. NB : détruire un fichier revient à modifier un répertoire, pas le fichier !). La présence d'un - à la place de r, w ou x signifie que cette action n'est pas autorisée. On peut modifier les droits directement en octal ou en spécifiant pour chaque catégorie (u : propriétaire, g : groupe, o : autres, a : tous) les modifications souhaitées.

```
<joseph@merteuil: 1>ls -l
-rw-r----- 1 joseph 59 Dec 15 10:33 bis
<joseph@merteuil: 1>chmod o+rw bis
-rw-r--rw- 1 joseph 59 Dec 15 10:33 bis
<joseph@merteuil: 1>chmod 644
-rw-r--r-- 1 joseph 59 Dec 15 10:33 bis
```

Par défaut, lorsqu'un fichier est créé, il prend les valeurs des droits d'accès par défaut, qui sont modifiables par l'appel de la commande **umask** (qui sans argument retourne la valeur actuelle du masque, umask 077 modifie les droits a 600 pour les fichiers et 700 pour les répertoires).

```
<joseph@merteuil: 1>umask
27
```

Ceci signifie :

```
pour les répertoires
droits demandés:  rwx rwx rwx ou encore 777
masque :          --- -w- rwx ou encore 027
droits accordés:  rwx r-x --- ou encore 750

pour les fichiers
droits demandés:  rw- rw- rw- ou encore 666
masque :          --- -w- rwx ou encore 027
droits accordés:  rw- r-- --- ou encore 640
```

On notera également l'existence des commandes **chgrp** (change group) et **chown** (change owner) qui sont interdites au simple utilisateur sur la plupart des systèmes mais sont en revanche très souvent utilisées par *root*. Leur syntaxe est simple à comprendre : `chown utilisateur[.groupe] fichier [fichier2 ...]` et `chgrp groupe fichier [fichier2 ...]`.
De manière générale, lorsque l'on souhaite savoir comment utiliser une commande, on peut avoir recours au manuel intégré en tapant la commande **man**. Toute la documentation Unix s'accède par les manuels classés en 8 thèmes:

- 1 les commandes utilisateurs
- 2 les appels système
- 3 les appels des librairies
- 4 les périphériques
- 5 les formats de fichiers
- 6 les jeux
- 7 les tables et standard
- 8 les commandes d'administration

`<joseph@merteuil:~>man commande` renverra donc des champs comme ls(1), open(2), passwd(1) et passwd(5) ...

4 Généralités sur le système de gestion de processus

4.1 Notion de processus

Pour pouvoir parler de processus, il faut tout d'abord un programme. Mais un fichier exécutable (binaire) n'est qu'une suite d'octets stockée sur un dique totalement inerte. Son contenu est en fait un ensemble de codes machine qui, lorsqu'ils sont chargés par la mémoire centrale de la machine pourront être lus et exécutés dès lors que le système d'exploitation sera là pour fournir au programme les informations qui lui sont nécessaires à une exécution correcte. L'ensemble des données d'identification pour chaque programme en cours est appelé **environnement de procesus**. Un processus est donc composé de deux éléments : un programme et un environnement de processus. Dès qu'une commande est transmise au shell, un processus est créé, le shell étant lui même un programme géré par le système d'exploitation sous la forme d'un processus.

Pour chaque processus exécuté par le système d'exploitation, est stocké un ensemble d'informations. Un des principaux éléments dans lequel se retrouvent ces données est la **table des processus** qui regroupe les données d'identification nécessaires aux processus tournant au moment présent sur la machine. Elle contient par exemple le numéro du processus **PID** (process identifier), le numéro de l'utilisateur, la durée du temps de traitement (temps CPU) .... On peut visualiser cela en tapant la commande **ps**.

```
<joseph@andromede: 1>ps

  PID TT       S  TIME COMMAND
 1470 pts/0    S  0:02 -tcsh
```

Sans option, ps liste les processus lancés de l'écran de l'utilisateur. UNIX étant un système multi-tâches, plusieurs programmes peuvent être lancés simultanément. C'est afin de les différenceir que le système leur affecte ainsi un numéro. Certains processus peuvent être amenés à lancer eux-mêmes des processus, on les appelle alors processus parents, et les nouveaux processus enfants. Ces derniers se voient communiquer par le système le numéro de leur processus parent **PPID** (parent process identifier). L'exception à cette règle est le pseudo-processus créé au moment du démarrage de la machine : il a normalement le numéro 0 et pas de réel processus parent. Il crée un processus enfant qui porte le numéro 1 et qui est le processus **init**, responsable directement ou indirectement de tous les processus en cours dans le système, il fournit la base permettant d'afficher à l'écran le login et d'utiliser le shell.

NB : quelques exemples d'utilisations de ps multi-options
`<joseph@andromede: 2>/usr/bin/ps -eo pcpu,pmem,nice,user,pid,args | sort -n`: psc
`<joseph@andromede: 3>ps -aux | head`: psh
`<joseph@andromede: 3>/usr/bin/ps -el -o user,pid,pcpu,pmem,pri,nice,stime,time -o args | grep lamyc | more`: psl
`<joseph@andromede: 3>ps -aux | more`: psm
`<joseph@andromede: 3>ps -alx | grep $1`: pst

Un processus ne peut travailler indéfiniment, puisque cela se ferait au détriment de tous les autres processus en cours. Le temps de traitement local est donc divisé en petites plages (time slices) et un processus ne peut rester en mémoire centrale que le temps d'une de ces plages. Lorsque le temps est écoulé, le système vérifie s'il y a d'autres processus à traiter. Le choix du processus suivant est lié à un facteur appelé **priorité**. Chaque processus se voit attribuer, en fonction de son utilisateur, de son temps écoulé de travail ... une priorité, et à la fin de chaque plage de temps, c'est celui qui possède la plus grande priorité qui est exécuté. Les utilisateurs ne peuvent influer sur la priorité initiale avec laquelle le processus est lancé, mais en revanche ils peuvent intervenir personnellment dans une certaine mesure au moyen de la commande **nice**.
L'utilisateur peut également être amené à tuer un processus qui ne veut pas se terminer ou qu'il a lancé par erreur. La commande radicale est alors :
`<joseph@andromede: 3>kill -9 numero-processus`

4.2 Traitement en tâche de fond

Lorsqu'une commande tapée n'est pas une commande interne du shell (ex : cal, time, mail ... ≠ cd, exit ...), un nouveau processus est automatiquement créé qui ne traite que la commande et est un processus enfant du shell. Le prompt du shell peut réapparaître dès que le traitement de cette commande est terminé et que le processus est clos. C'est le shell qui supervise tout cela, se mettant en attente de la fin du processus. Le shell et le processus étant synchronisés par l'attente, on parle d'exécution synchrone du processus. Mais lorsque l'utilisateur souhaite lancer des commandes complexes sans attendre la fin de leur exécution, il lui faut recourir à une technique très simple qu'offre le shell pour lancer des processus enfant dont il n'a pas besoin d'attendre la fin. Il s'agit simplement d'ajouter en fin de la ligne de commande le symbole & (ampersand). On parle alors de pocessus en tâche de fond, le shell et son processus enfant fonctionnant en parallèle, sans plus aucune coordination entre eux : on parle donc d'exécution asynchrone du processus.

Il faut être conscient des contraintes que le lancement d'un processus en tâche de fond impose :

- le processus en tâche de fond ne doit pas attendre de saisie au clavier, le shell ayant retrouvé le monopole de l'accès au clavier. Dans le cas contraire, le système d'exploitation ne saura pas déterminer à quel processus sont destinées les saisies.
- le processus en tâche de fond ne doit pas retourner ses résultats à l'écran ou sur le terminal, car ses sorties risqueraient d'entrer en conflit avec celles du shell ou d'une autre commande.

- Les commandes en tâche de fond dépendent du shell et sauf précautions, terminer le shell amènera la terminaison du processus en tâche de fond.

Si on a par mégarde lancé la commande en oubliant le &, il est toujours possible de faire passer le procesus enfant en tâche de fond en réalisant un enchaînement de commandes : **^D** puis **bg** (background). Ceci permet de plus de lancer en tâche de fond des processus qui au début de leur exécution ont pu nécessiter des saisies clavier.

Les processus seront abordés plus à fond dans la suite de la semaine.

5 L'interpréteur de commande : le shell

5.1 Les différents shells sous UNIX

Comme nous l'avons dit précédemment, l'interpréteur de commandes (shell) est lancé lors de l'ouverture d'une session et invite l'utilisateur à lancer ses commandes après l'apparition du prompt. Le premier mot placé dans la ligne de commande est la commande elle-même, même si certains shells (ksh, csh ...) autorisent cette commande à être un alias, qui sera alors remplacé par sa valeur avant tous les autres mécanismes. La commande est séparée du reste des paramètres optionnels ou des arguments par le séparateur de champs qui par défaut est l'espace ou la tabulation. Il est également possible de lancer plusieurs commandes dans une même ligne, à condition de les séparer par des points virgules **;**. On peut éventuellement remplacer les points virgules par des & (Cf. paragraphe précédent).

Il existe deux modes d'utilisation du shell :

- mode commande
  - interactif et conversationnel
  - exécution de commandes au coup par coup
  - possibilité d'enchaînement de plusieurs commandes sur la même ligne
- mode programmation
  - non conversationnelle
  - langage de programmation (scripts)
  - création de nouvelles commandes avec une notation concise, paramétrable et des strcutures de contrôle

On trouve ci-dessous un exemple de script shell : il s'agit de la fonction zmore.

```
#!/bin/sh

PATH="/usr/local/bin:$PATH"; export PATH
if test "`echo -n a`" = "-n a"; then
  # looks like a SysV system:
  n1=''; n2='\c'
else
  n1='-n'; n2=''
fi
oldtty=`stty -g 2>/dev/null`
if stty -cbreak 2>/dev/null; then
  cb='cbreak'; ncb='-cbreak'
```

```
else
  # 'stty min 1' resets eof to ^a on both SunOS and SysV!
  cb='min 1 -icanon'; ncb='icanon eof ^d'
fi
if test $? -eq 0 -a -n "$oldtty"; then
   trap 'stty $oldtty 2>/dev/null; exit' 0 2 3 5 10 13 15
else
   trap 'stty $ncb echo 2>/dev/null; exit' 0 2 3 5 10 13 15
fi

if test $# = 0; then
        gzip -cdfq | eval ${PAGER-more}
else
    FIRST=1
    for FILE
    do
        if test $FIRST -eq 0; then
                echo $n1 "--More--(Next file: $FILE)$n2"
                stty $cb -echo 2>/dev/null
                ANS=`dd bs=1 count=1 2>/dev/null`
                stty $ncb echo 2>/dev/null
                echo " "
                if test "$ANS" = 'e' -o "$ANS" = 'q'; then
                        exit
                fi
        fi
        if test "$ANS" != 's'; then
                echo "------> $FILE <------"
                gzip -cdfq "$FILE" | eval ${PAGER-more}
        fi
        if test -t; then
                FIRST=0
        fi
    done
fi
```

De même qu'il y a deux familles d'UNIX, on trouve deux familles de shell :

- System V
  - Bourne Shell **sh**
  - Korn Shell **ksh**
  - Gnu Shell **bash**
  - Z Shell **zsh** (information à l'adresse :http://www.cco.caltech.edu/cco/texinfo/)
-
  - C-Shell **csh**
  - Toronto C-Shell **tcsh**

Pour reconnaître son shell, il suffit de faire appel à une variable particulière appelée SHELL qui est initialisée dans les fichiers de configuration de l'utilisateur.

```
<joseph@merteuil: 1>echo $SHELL
```

Le Bourne Shell est le shell d'origine d'UNIX (milieu des années 70), ce qui fait qu'il est le moins convival de tous et est délaissé au profit de shells tels que zsh ou tcsh pour l'utilisation en mode commande, ces derniers étant compatibles avec le Bourne Shell. En revanche, en mode programmation, bien que les nouveaux shells apportent des facilités, le Bourne Shell reste bien suffisant d'autant que, étant présent sur tout système UNIX, il garantit la portabilité des scripts quel que soit le système. Il y a donc fréquemment cohabitation de plusieurs shells sur une même machine. On notera également que les systèmes V version 4 supportent le C-Shell et le Toronto C-Shell.

Un shell est paramétrable grâce à ses fichiers d'initialisation, ce qui permet à l'utilisateur de définir son environnement, par exemple en rentrant ses propres alias. Les fichiers de configuration se trouvent dans le répertoire principal de l'utilisateur et varient selon le shell :

```
sh, ksh     .profile
zsh         .profile .zshrc
csh, tcsh   .login .cshrc .tcshrc .logout
```

Dans ces fichiers, on trouvera des variables telles que PATH (définit les différents répertoires dans lesquels seront recherchées les commandes exécutées par l'utilisateur), SHELL (définit le shell courant), TERM (définit le type du terminal courant), DISPLAY (définit l'écran associé à la sortie standard du terminal) ...
En mode interactif, les shells les plus évolués (ksh, bash, zsh, tcsh) offrent des facilités à l'utilisateur :

- l'éditeur de ligne de connande permet l'utilisateur de se déplacer sur la ligne de commande afin de la modifier avec les flèches ← et →
- la touche Tab (tabulation) permet de compléter les noms des fichiers après en avoir entrer les premières lettres
- la commande **history** garde en mémoire les dernières commandes exécutées. On peut se déplacer dans l'historique en mode édition avec les flèches ↑ et ↓ ou utiliser en mode commande !! pour la dernière commande effectuée ou plus généralement !chaine pour rappeler la dernière commande commençant par chaine ou !numero pour rappeler la commande numéro numero dans le fichier history.

Certains caractères ont un sens spécial pour le shell :

```
; & | ( ) { } < > * ? [ ] ' ` " = \ <ENTER> $
```

Lorsqu'un caractère spécial est protégé (précédé de slash ou inclus entre ' (sauf ' lui-même)), il perd son sens spécial. Entre " et ", les subsitutions de paramètres et de commandes sont effectuées, seuls les caractères ", \ et $ devant être protégés.

```
<joseph@merteuil: 4>echo *
deux trois un
<joseph@merteuil: 4> echo \*
*
<joseph@merteuil: 4>echo ' -->* $TERM'
-->* $TERM
<joseph@merteuil: 4>echo " -->* $TERM"
-->* vt102
```

5.1.1 Cas d'erreurs

Il faut savoir réagir aux erreurs qui peuvent se produire. Plusieurs cas peuvent se présenter :

- la commande n'existe pas (se rappeler que le shell différencie les minuscules des majuscules)
- l'utilisateur n'a pas le droit d'exécuter la commande
- les options des commandes sont erronées
- les arguments de la commande sont erronés

```
<joseph@merteuil: 4>dat
dat: Command not found.
<joseph@merteuil: 4>reboot
reboot: can't idle init
<joseph@merteuil: 4>uname -k
uname: illegal option -- k
usage: uname [-snrvmap]
<joseph@merteuil: 4>cal 13 1997
Bad argument.
```

5.2 Redirection des entrées/sorties

La redirection des entrées/sorties est une des principales caractéristiques du shell, et de plus sa puissance n'a d'égale que sa simplicité. C'est l'un des éléments qui font d'UNIX un système d'exploitation souple et performant. En effet, nous savons que nombre de commandes, applications et utilitaires UNIX donnent des informations à l'écran, qui ne sont pas toujours aisémeent utilisable telles qu'elles. Il est alors intéressant de pouvoir envoyer ces données dans un fichier, que l'on pourra ensuite étudier, modifier, conserver à loisir.

Toutes les commandes utilisent les canaux d'entrées/sorties pour lire des données ou transmettre leurs informations. Le canal d'entrée utilisé en général pour la lecture est lié au clavier, celui de sortie est lié à l'écran, mais ceci est modifiable par le shell très rapidement et très facilement. La commande peut alors lire les données depuis un fichier et non plus à partir du clavier et écrire les résultats dans un fichier et non plus à l'écran. Le fait même que les périphériques aient pour le système un statut de fichier rend tout cela parfaitement transparent pour les commandes. Il existe trois canaux standards sous UNIX : l'entrée **stdin** (standard input), la sortie **stdout** (standard output) et une sortie séparée pour les messages d'erreur **stderr** (standard error).

- redirection de sortie : commande > . On notera que le shell commence par chercher les caractères spéciaux de la ligne de commande. Si il trouve un >, il créé le fichier si il n'existe pas déjà ou écrase le précédent, ceci même si la commande placée devant le > est incorrecte et n'effectue aucune sortie.
- rajouter des éléments à la fin d'un fichier : commande >> . Si le fichier dont le nom suit le >> n'existe pas, il est créé.
- redirection d'une entrée : commande < . Les éditeurs en ligne (vi, pico ...) ne parviennent pas à gérer ce type de redirection.

```
<joseph@merteuil: 1>ls -F
deux/ trois/ un/
<joseph@merteuil: 2>ls -F >fichier
<joseph@merteuil: 3>ls -F
deux/ fichier trois/ un/
<joseph@merteuil: 4>cat fichier
deux/ fichier trois/ un/
<joseph@merteuil: 5>date >> fichier
<joseph@merteuil: 6>cat fichier
deux/ fichier trois/ un/
Sun Dec 15 10:11:22 MET 1997
```

On peut aussi choisir de rediriger pour plus d'une seule opération l'une des trois sorties standard. Pour cela on va utiliser le fait qu'UNIX leur attribue un numéro : 0 pour stdin, 1 pour stdout et 2 pour stderr. On pourra ainsi rediriger la sortie standard d'erreur en utilisant de nouveau le >, comme ci-dessous :

```
<joseph@merteuil: 7>2 >fichier-erreurs
<joseph@merteuil: 8>rm zozo 2 >fichier-erreurs: erreur car zozo n'existe pas
<joseph@merteuil: 8>cat fichier-erreurs
zozo: No such file or directory
```

Le shell travaille de la gauche vers la droite, donc si plusieurs redirections sont présentes dans la même ligne de commande, elles seront traitées de la gauche vers la droite. Cet ordre est important, notamment lorsque l'on souhaite réunir des canaux :

```
<joseph@merteuil: 9>rm zozo >fichier 2>&1
```

Cette commande redirige le canal de sortie standard vers le fichier fichier et réunit en un seul et même canal le canal de sortie standard et le canal d'erreur standard. Si on avait inversé les commandes, seul le canal de sortie standard aurait finalement été redirigé vers le fichier fichier.

On peut ouvrir simultanément et parallèlement aux trois canaux que nous connaissons déjà sept autre canaux, identifiés par leurs numéros de 3 à 9. Ils s'ouvrent à l'aide dela commande **exec** et se ferment en réunissant la sortie avec le pseudo-canal "-". Le shell permet également l'emploi du signe <<com, mais son rôle est totalement différent. Cette commande garde en mémoire ce qui est tapé au clavier jusqu'à ce que soit tapé com <ENTER> en début de ligne. Cette redirection est surtout utilisée dans les scripts shell.

```
$exec 3> xyz : le canal 3 est ouvert en qualité de sortie du fichier xyz.
$who >&3
joseph console Dec 10 09:29
joseph pts/1 Dec 10 09:45
joseph pts/2 Dec 10 09:33
root pts/3 Dec 15 11:03 (andromede)
$exec 3>&-: le canal 3 est fermé
```

```
<joseph@merteuil: 11>wc << zozo
? a b c d
? e f g h
```

```
? zozo
2 8 16
```

5.3 Concept de filtre

Un filtre (ou une commande filtre) est un programme sachant écrire et lire des données par les canaux standard d'entrée et de sortie. Les possibilités de redirection des entrées et sorties permettent alors d'utiliser ces filtres pour résoudre certains problèmes de manière simple et propre. Par exemple, compter le nombre de fichiers ou sous-répertoires présents dans un répertoire?

Compter après un ls peut se révéler fastidieux si les fichiers sont nombreux ou si l'affichage dépasse la taille de l'écran. Rediriger ls -l dans un fichier et utiliser wc pour compter le nombre de lignes est déjà plus astucieux mais créé un fichier inutile qu'il faudra ensuite supprimer. Le plus simple est d'utiliser un — (pipeline). Toutes les données renvoyées par la commande palcée à gauche de ce signe sotn envoyées au canal d'entrée standard de la commande placée à droite. On peut alors faire :
`<joseph@merteuil: 12>ls -l | wc -l`
ce qui nous renverra directement le nombre recherché.

Les principales commandes de filtre sont :

- **sort** lit les données ligne par ligne par le canal de l'entrée standard et les retourne triées par le canal de sortie standard. Par défaut, c'est l'ensemble de la ligne de saisie qui est considérée par le critère de tri (l'ordre de tri est du ressort de l'administrateur système, en général c'est l'ordre alphabétique ou l'ordre ASCII).
- **more**, **pg** et **less** qui lisent les données du canal d'entrée standard et les retournent page par page à l'écran.
- **grep** + **critère** qui recherche et imprime les lignes contenant le critère spécifié

6 Unix et les réseaux

Au milieu des années 70, des essais avaient déjà été faits pour faire tourner UNIX en réseau. Ce sont tout particulièrement les universités américaines qui ont beaucoup travaillé sur ce sujet. Mais là encore il manquait une ligne directrice. Beaucoup d'idées ont ainsi vu le jour, certaines devenant même des produits diffusés sur le marché. Entre autres Hewlett Packard proposa un concept de réseau pour les machines sous UNIX, appelé Remote File Access ou RFA.

Un autre produit rencontra plus de succès : le Network File System (NFS) de Sun. Gràce au NFS, une machine sous UNIX pouvait mettre à disposition des autres machines une partie sans que le concept de gestion de fichiers ne nécessite de modification. Pour l'utilisateur final, cela signifiait qu'il lui était impossible de savoir immédiatement si les fichiers auxquels il accédait étaient placés sur sa propre machine ou sur un poste quelqconque du réseau.

NFS étant d'un prix particulièrement abordable, beaucoup de constructeurs reprirent cette extension réseau pour leur propre UNIX. Malgré ses inconvénients, NFS devint un des standards en matière d'extension réseau, et le produit développé ensuite par AT&T ne sut pas s'imposer sur le marché (Remote File System : RFS). On notera que la grande flexibilité de NFS fait qu'il est facilement portable sur d'autres systèmes d'exploitation : on trouve également NFS pour

MS-DOS, Windows ...

Le réseau c'est à la fois le réseau local des machines d'un même domaine, accessibles en général à tous les utilisateurs de ce domaine. Ils peuvent soit se connecter directement à la machine (se mettre devant la console et effectuer la procédure de login déjà étudiée), soit s'y connecter à distance (rlogin, telnet). Ils se retrouvent alors comme sur la machine, pouvant accéder aux ressources de celle-ci comme l'utilisateur loggué en console. Avec le réseau viennent certaines applications comme le mail (messagerie électronique), ftp (file trandfer protocol) ...

Pour terminer on citera Internet, réseau bien connu de nos jours, dont font parti la pluaprt des machines des écoles d'ingénieurs. Netscape, Internet Explorer ou tout autre navigateur peut alors emmener vers des pages web situées un peu partout dans le monde.