SCIENCES SUP

Cours et exercices corrigés

1er et 2e cycles • IUT • Écoles d'ingénieurs

LINUX

Initiation et utilisation

2e édition

Jean-Paul Armspach
Pierre Colin
Frédérique Ostré-Waerzeggers

DUNOD

LINUX

Initiation et utilisation

LINUX

Initiation et utilisation

Jean-Paul Armspach
Ingénieur de recherche
à l'université Louis Pasteur de Strasbourg

Pierre Colin
Professeur à l'École Nationale Supérieure
de Physique de Strasbourg

Frédérique Ostré-Waerzeggers
Ingénieur système et administrateur réseau
à l'université Louis Pasteur de Strasbourg

2e édition

DUNOD

Illustration de couverture réalisée à partir
du logo Linux © 1997 Andreas Dilger

ISBN 2 10 007654 X

Table des matières

Conventions d'écriture

Les conventions d'écriture retenues dans cet ouvrage sont les suivantes :

Télétype italique	Dans le corps du texte, indique un élément d'une commande, ou un nom de fichier : */etc/passwd*
Texte gras	Utilisé dans le corps du texte pour mettre en valeur un nouveau terme technique : **file system**
Télétype	Police non proportionnelle utilisée dans tous les exemples pour montrer ce qui apparaît à l'écran, ou le contenu d'un fichier. Tout ce qui suit un caractère # est un commentaire pouvant apparaître dans une commande Linux.
Télétype gras	Dans un exemple, indique ce qui est saisi par l'utilisateur (les réponses du système apparaissent en Télétype fin): xstra> **ls -l**
§ Télétype italique	Le texte italique qui suit le caractère § dans un exemple est une explication ajoutée par les auteurs et n'apparaît donc pas à l'écran. Il ne doit pas être saisi lors d'une commande.

<ctrl-D>	Indique qu'il faut appuyer simultanément sur les touches "Ctrl" et "d". Bien que D apparaisse en majuscule, il est inutile d'appuyer la touche "Majuscule".
<espace>	Représente le caractère "espace" obtenu par la barre d'espacement.
Esc	Représente le caractère d'échappement ; la touche "Esc" se trouve généralement en haut à gauche sur le clavier.
[*option1,option2*]	Les points cités entre crochets représentent des éléments optionnels.

Avant-propos

Linux est devenu en quelques années une alternative sérieuse aux systèmes d'exploitation Microsoft pour les ordinateurs personnels (PC). Linux est la version PC la plus répandue du système d'exploitation Unix utilisé dans l'informatique professionnelle sur stations de travail et grands ordinateurs.

Le succès actuel de Linux est dû à ses multiples avantages :

- libre et ouvert, diffusé gratuitement ou à faible coût ;
- indépendant de tout constructeur et de tout éditeur de logiciels ;
- évolutif, mais très stable dans son fonctionnement ;
- doté d'une interface graphique conviviale et personnalisable ;
- assurant la portabilité du savoir et des logiciels du monde Unix ;
- disposant d'outils bureautiques et de publication de qualité ;
- supportant de nombreux outils de développements ;
- disposant d'un excellent support des protocoles et applications Internet.

Ajoutons à cette liste un avantage incomparable : les nombreux forums de discussion existant sur Internet (news) assurent à l'utilisateur de Linux une *hot line* gratuite dont la compétence et le temps de réponse sont au niveau des meilleurs supports techniques, ceci quel que soit le niveau de la question posée.

Tous ces avantages ont un prix : pour tirer le meilleur de Linux, il faut comprendre sa philosophie et apprendre à se servir de ses outils de base. C'est le but de cet ouvrage. Issu de cours dispensés en formation permanente à l'Université Louis Pasteur (Strasbourg), au Centre National de la Recherche Scientifique (CNRS) et en entreprises, et de cours dans une école d'ingénieurs, il tient compte de ces diverses expériences d'enseignement. Ce livre s'adresse à toute personne ayant quelques connaissances en informatique et désirant se convertir au monde Linux.

Comme tout support de cours, cet ouvrage présente une progression dans la difficulté, et les notions nouvelles sont introduites par étapes. Le lecteur ne doit donc pas

s'étonner si certaines notions sont introduites simplement au début, puis reprises et approfondies plus loin dans l'ouvrage : nécessaires à la compréhension générale, elles ne justifient pas de prime abord une présentation complète et minutieuse qui risquerait de rompre le fil de la progression. Cet ouvrage n'est pas un manuel de référence : la documentation technique disponible en ligne est exhaustive et irremplaçable. Il s'agit bien d'un manuel de cours et le lecteur pourra avec profit le lire « à côté du clavier », en essayant immédiatement les exemples et exercices présentés.

Dans la majorité des cas, les termes techniques français sont employés tout au long de l'ouvrage. Le terme anglais est cependant présenté systématiquement lors de la première référence pour une raison capitale : la documentation disponible en ligne est très souvent en anglais et le lecteur doit pouvoir s'y référer sans ambiguïté.

L'environnement universitaire particulièrement favorable dans lequel nous travaillons, notre expérience de l'enseignement d'Unix et des réseaux et l'utilisation quotidienne de Linux nous ont permis d'améliorer cet ouvrage. Cette deuxième édition tient compte des dernières évolutions de Linux et des nombreuses remarques des étudiants de l'École Nationale Supérieure de Physique de Strasbourg (ENSPS).

Nos conjoints ont fait preuve de beaucoup de patience pendant la réalisation de cet ouvrage. Merci Martine, Jocelyne et Benoît pour votre soutien.

Les auteurs.

Chapitre **1**

Présentation de Linux

1.1 HISTORIQUE

1.1.1 Au début : Unix

Unix est né, en 1969, dans les **Bell Laboratories (AT&T)** sous l'impulsion de deux hommes, **Ken Thompson** et **Dennis Ritchie**. Il était destiné à fournir aux programmeurs maison un environnement de développement auquel ils avaient goûté avec **Multics** (MULTiplexed Information and Computing Service). Suite au développement par Dennis Ritchie du langage C, adapté à ce type de programmation, une nouvelle version d'Unix est réécrite, en grande partie en C, en 1973. Pour la première fois, un système d'exploitation est écrit en langage non assembleur, et devient donc portable, du moins en principe.

En 1974, la version 4 d'Unix est « donnée » à l'université de Berkeley, Californie, qui commence alors son propre développement du système. C'est le début d'une divergence entre les deux versions d'Unix : AT&T et BSD (Berkeley Software Distribution). Le succès d'Unix devient alors considérable dans les universités américaines : seul système d'exploitation disponible en source, sur mini–ordinateur (PDP11), il est adopté par les départements de « computer science » des universités pour la formation des étudiants en informatique système. Le nombre d'experts Unix croît à une vitesse considérable.

De 1977 à 1979, Ken Thompson et Dennis Ritchie réécrivent Unix pour le rendre réellement portable. Et en 1980 les premières licences de distribution d'Unix System V d'AT&T sont délivrées aux constructeurs.

L'année 1984 voit la création du groupe X/Open, composé de constructeurs informatiques ayant adopté Unix et se donnant pour but de normaliser les différentes versions d'Unix.

Cette même année est créée par le MIT la norme X Window : système de multifenêtrage graphique, indépendant d'Unix, mais qui contribuera au succès de ce système.

En 1987, AT&T, propriétaire d'Unix, et Sun, un des leaders du marché des stations de travail et utilisant la version BSD, concluent une alliance visant à une convergence entre les deux systèmes.

En 1988 sont créés deux consortiums :

- OSF (Open Software Fondation), (DEC, HP, IBM, …) travaillant à la normalisation d'un nouvel Unix baptisé OSF1 ;
- Unix International (AT&T, Sun, …) cherchant à imposer Unix System V.

En 1992, Digital Equipment propose DEC/OSF1, première version commercialement disponible d'OSF1, et Sun propose la première version commerciale résultant de la convergence entre System V et BSD. Il y a encore plusieurs Unix, mais les différences ne représentent plus une difficulté pour l'utilisateur.

Le succès d'Unix tient à plusieurs facteurs :

- Le plus important, à l'origine, est son adoption par les universités américaines pour la formation des étudiants, ce qui a sans aucun doute permis de former plus d'experts sur ce système que sur aucun autre.
- Un autre facteur déterminant est le besoin de standard exprimé par les utilisateurs, qu'ils soient sociétés de développement de logiciel ou utilisateurs finals. Dans un monde où le changement de fournisseur avait toujours été catastrophique pour l'utilisateur, les systèmes ouverts offrent des perspectives d'évolution en douceur tout en maintenant une concurrence entre fournisseurs. Cette pression des utilisateurs est un fait nouveau dans l'histoire de l'informatique. La participation de groupes d'utilisateurs et d'éditeurs de logiciels aux consortiums dirigeant l'évolution d'Unix n'a de réel équivalent pour aucun autre système d'exploitation.
- Unix est le seul système d'exploitation multi–utilisateur disponible à faible coût pour une société développant un système à base de processeur standard.

1.1.2 Linux

Unix est l'un des systèmes d'exploitation le plus populaire au monde, en raison du grand nombre d'architectures qu'il supporte. Il existe des versions d'Unix pour tous les types d'ordinateurs, y compris les ordinateurs personnels.

SCO Unix est le système le plus ancien sur cette plate–forme. Le premier package Unix, de nom SCO Xenix System pour Intel 8086 et 8088, date de 1983. La société SCO est maintenant propriétaire de la marque Unix, qu'elle a achetée à la société Novell, qui l'avait elle–même achetée à une filiale d'ATT (Unix System Labs).

SCO Unix existe toujours. Unix est maintenant une marque déposée de l'Open-Group.

L'idée d'un système d'exploitation libre est né en 1984 avec la Free Software Foundation (FSF). Les bases de l'environnement ont été définies. Puis des outils, tels des éditeurs, des compilateurs, des shells ... ont été développés.

Linux, système Unix libre sur plate–forme PC, était au départ un projet de loisirs de Linus Torvalds, étudiant finlandais. Linux fut inspiré de Minix, un petit système Unix développé par Andrew Tanenbaum. Les premières discussions autour de Linux se passèrent sur le forum comp.os.minix. Ses débuts furent la maîtrise de la commutation de tâches du mode protégé du processeur 80386, tout fut écrit en assembleur. Le 5 octobre 1991, Linus Torvalds annonça la première version « officielle » de Linux, la version 0.02. Après la version 0.03, Linus passa directement en version 0.10.

Linux a continué à évoluer grâce à Linus Torvalds et aussi aux efforts de nombreux volontaires répartis aux 4 coins du monde, reliés entre eux par le réseau Internet (chapitre 13). Sous la pression de ces co–développeurs, Linus Torvalds a accepté que tout le code soit sous licence GPL (General Public Licence), créant ainsi un noyau Unix totalement libre. Grâce à ce réseau, toute personne intéressée par le développement de ce système peut aider : porter des programmes, écrire de la documentation, corriger des bogues… On compte actuellement plus de 18 millions d'utilisateurs de Linux, et nombreux sont ceux qui contribuent d'une façon ou d'une autre au développement de ce système et de son environnement.

À ce jour, Linux est un vrai système 32 bits, multitâches, multi-utilisateurs, réseau et complet. Il s'installe sur la plupart des PC (avec ou sans autre système d'exploitation). Il supporte une large gamme de programmes tels que X Window, TCP/IP, C/C ++GNU et d'autres outils GNU, le courrier électronique, les news, des outils dérivés de LateX (LyX), ou des outils de bureautique. Une machine sous Linux est modulaire et paramétrable à souhait. Elle peut donc servir de station personnelle ou de serveur (Web, ftp…).

Linux est une libre implémentation des spécifications POSIX, avec des extensions System V et Berkeley. Ceci accélère la propagation de Linux au sein de l'administration, qui exige la conformité POSIX de la plupart des systèmes qu'elle utilise. Linux est un phénomène très important et peut devenir une alternative au système Microsoft Windows, grâce notamment aux outils bureautiques…

Linux est le plus souvent diffusé sous forme d'une distribution, un ensemble de programmes (noyau, sources des utilitaires, commandes, applications) formant après installation un système complet. Par conséquent, il est de plus en plus utilisé dans les sociétés commerciales comme station de travail et serveur.

Le succès de Linux tient à plusieurs facteurs :

- Le code source du système, ainsi que le noyau, les programmes utilisateurs, les outils de développement sont librement distribuables (licence GPL, ou GNU),

- Linux est compatible avec un certain nombre de standards Unix au niveau du code source, incluant les spécifications POSIX, system V et BSD,
- Un très grand nombre d'applications Unix gratuites disponibles sur Internet se compilent sous Linux sans aucune modification,
- Le système Linux a été développé pour les processeurs Intel et utilise toutes les fonctionnalités de ce processeur.

1.2 LES ORGANISMES DE NORMALISATION ET LES GROUPES D'UTILISATEURS

L'évolution d'Unix, essentiellement technologique, n'allait pas vers une convergence réelle des différentes branches (OSF, Berkeley, AT&T, IBM, Microsoft). Cela a amené les utilisateurs à se regrouper afin d'exprimer leur indépendance vis–à–vis des constructeurs et surtout d'essayer de définir une norme internationale pour ce système d'exploitation (la première association d'utilisateurs fut /usr/group). Deux organismes de normalisation ont ainsi émergé :

- Le groupe X/Open (1984), association au départ européenne à laquelle se sont joints les plus grands noms américains, s'est donné pour but d'améliorer l'environnement du système, de fournir un guide de portabilité (XPGIII) pour les concepteurs d'applications, de produire des utilitaires nouveaux.
- Le groupe **POSIX** (Portable Open System Interface eXchange) qui fait partie de l'association IEEE (Institute of Electrical and Electronic Engineers), est divisé en sous–groupes de normalisation. Chacun de ces groupes a rédigé des guides, langages et extensions pour Unix.

Ces deux groupes étaient devenus de véritables organismes de standardisation. Mais les travaux du groupe POSIX sont devenus plus déterminants que ceux de l'association X/Open.

D'autres groupes d'utilisateurs existent. Ils essaient d'être de véritables forums d'échanges et de recherche sur Unix, mais contrairement aux précédents ils ne travaillent pas sur des normes.

Il existe en France l'**AFUU** (Association Française des Utilisateurs d'Unix et des systèmes ouverts). Cette association a été créée en 1982 par un petit groupe d'utilisateurs pionniers dans le domaine des systèmes Unix. Ses objectifs sont de promouvoir, de développer et de diffuser la « culture Unix » auprès des utilisateurs et ceci par le biais des adhérents (organisation de manifestations dont la plus importante est la « convention Unix », de séminaires, de conférences professionnelles, publication de revues, de périodiques, comme « Tribunix », et d'ouvrages fondamentaux). L'association est représentée par des chercheurs, des informaticiens, des ingénieurs, … et par ses groupes de travail (groupe Sécurité, groupe Portabilité, groupe Réseaux, …).

Au même titre, l'AFUL (Association Francophone des Utilisateurs de Linux et des Logiciels Libres) est une association de loi 1901 dont l'objectif principal est de

promouvoir, directement ou indirectement les logiciels libres, et en particulier les systèmes d'exploitation libres principalement ceux basés sur les normes POSIX ou dérivés, dont le plus connu est le système Linux muni de l'environnement GNU (statut de l'association).

L'AFUL fonctionne par l'intermédiaire de groupes de travail : personnes souhaitant discuter d'un sujet particulier lié aux logiciels libres, avec comme objectif de produire des documents de synthèse diffusés publiquement.

L'AFUL est aussi représentée par «les groupes d'utilisateurs locaux de Linux/ Logiciels Libres », association d'utilisateurs localisés géographiquement. Ces associations travaillent essentiellement par discussion sur des listes de diffusion et offrent des serveurs Web fournis. En Alsace, le « Linux Users Group » est le représentant régional très actif. Son serveur Web se trouve à l'adresse *http://tux.u–strasbg.fr*.

Les logiciels de **la Free Software Foundation** [FSF] de Richard Stallman (utilitaires de développement [compilateurs C et Ansi C, C++, make, awk, emacs, groff,…) sont distribués en code source. Vous pouvez les dupliquer, les distribuer et les modifier comme bon vous semble. Cette philosophie procure à ces logiciels, par opposition aux logiciels commerciaux, un taux d'utilisation et de distribution extrêmement élevé. De très nombreuses personnes dans le monde entier travaillent sur ces logiciels, les améliorent, corrigent les erreurs rencontrées. C'est en fait un regroupement d'utilisateurs sans existence légale.

Linux est l'un de ces logiciels. Il est protégé par ce qui est connu sous le nom de **la General Public Licence** (GPL). La GPL fut développée pour le projet **GNU** par la FSF ; on parle ainsi de **GNU–Linux** comme de **GNU–C**, **GNU–Emacs**, ... Un programme protégé par la GPL appartient à son ou ses auteurs, mais il peut être distribué librement et gratuitement. La GPL autorise aussi les utilisateurs à modifier les programmes et à en distribuer leur version, tout en gardant cette licence GPL. La distribution de ces programmes peut aussi être payante, à la condition que les sources y soient inclus. Cela peut paraître contradictoire, mais dans ce monde du logiciel libre l'objectif est de développer et diffuser des programmes en permettant à quiconque de les obtenir et de les utiliser.

1.3 LES DISTRIBUTIONS LINUX

GNU–Linux est un système Unix complet, avec un noyau maintenu par Linus Torvalds et diverses applications liées au système. De nombreux outils, tels des compilateurs, des éditeurs, des interfaces graphiques, existent.

Il serait très difficile pour beaucoup d'utilisateurs de construire un système complet en partant du noyau, des sources des utilitaires, commandes, applications.

Linux est le plus souvent diffusé sous forme d'une distribution, un ensemble de programmes (noyau, sources des utilitaires, commandes, applications) formant après installation un système complet. Chacune des distributions a ses avantages et

ses inconvénients. Débuter avec Linux, c'est surtout choisir une distribution qui corresponde avec les usages qu'on attend du système.

RedHat est la première société créée ayant pour objectif de rassembler tout ce qui est nécessaire dans une distribution. Elle a été fondée en 1994 en Caroline du Nord. Ses objectifs furent le développement de solutions logicielles, du support téléphonique, du consulting sur site, des formations.

Pour les développeurs, RedHat met sa distribution à disposition sur Internet. Cette société la vend aussi avec une documentation, c'est la version dite « Server Enterprise ».

Parmi les distributions les plus utilisées, on trouve RedHat, Debian, Slackware (toutes trois gratuites) et SuSE, Mandrake (toutes deux payantes).

Installation

L'installation de Linux sur une machine est une opération assez simple car interactive dans chaque distribution. Toutefois, le meilleur conseil que nous pourrions donner est de choisir une distribution accompagnée d'une documentation papier d'installation complète. Il est également possible (et recommandé pour un débutant) d'acheter une nouvelle machine avec Linux déjà installé. Si vous voulez installer Linux sur une machine existante, avant de commencer, vous devriez savoir répondre aux quelques questions suivantes :

- Quel est le débit de ma connexion à l'Internet ? Si l'accès à l'Internet se fait par un modem, même à 56K, une distribution gratuite téléchargée coûtera beaucoup plus cher qu'un jeu de CD avec la documentation !
- Mes sauvegardes me permettront-elles de remettre mon système dans son état actuel en cas de problème ? Si la réponse est oui, je peux passer à la suite.
- Mon système dispose-t-il d'un espace disque disponible d'au moins 1 Go ? Si ce n'est pas le cas, il est inutile d'aller plus loin. Si cet espace disque disponible se trouve dans des partitions déjà formatées et utilisées par ailleurs, il faudra libérer cet espace en réduisant la taille des partitions (et ceci nous ramène à la question précédente sur les sauvegardes). Linux aura besoin de ses propres partitions. Le paragraphe 10.2.3 donne quelques conseils sur les partitions disque pour Linux, indépendamment de tout autre système d'exploitation.
- Ce système Linux sera-t-il autonome ou intégré à un réseau local ? Dans le premier cas, préférer une installation de type « station de travail » alors que dans le deuxième, une installation de type « serveur » sera peut-être plus adaptée. Dans ce dernier cas, vous devrez toutefois acquérir des connaissances d'administration Unix et réseau.

L'installation des différentes distributions de Linux est de plus en plus facile et conviviale, et propose des configurations par défaut relativement propres. Comme règles de base à l'installation d'un système linux, nous vous conseillons :

- de choisir une installation de type workstation ;
- si votre poste est à votre domicile et que vous utilisez une connexion ADSL, mettre en place une protection dite « firewall moyen ».

Une fois l'installation terminée, il est important de créer un deuxième utilisateur. Ce compte sera en fait votre compte de travail. Vous ne vous connecterez en tant qu'administrateur système (root) que pour lancer des commandes purement système.

Le présent ouvrage ne s'adresse pas à des administrateurs Unix confirmés et n'est pas destiné à l'apprentissage de l'administration système. Il ne présente que les concepts de base permettant d'utiliser Unix/Linux, c'est pourquoi nous n'entrerons pas plus en détail dans l'installation.

1.4 NOTION DE SYSTÈME D'EXPLOITATION

Unix est un système d'exploitation, constitué du **noyau Unix**, **d'un interpréteur de commandes** et d'un grand nombre d'**utilitaires**.

Le **noyau** assure la gestion des ressources physiques (processeur, mémoires, périphériques) et logicielles (processus, fichiers…). L'interface entre les programmes des utilisateurs et le noyau est définie par un ensemble de procédures et de fonctions, soit directement au niveau du noyau, soit par l'intermédiaire de bibliothèques. Pour ce qui concerne l'architecture du noyau proprement dit, les éléments caractéristiques sont les suivants :

- Le noyau est constitué d'un ensemble de procédures et de fonctions écrites pour l'essentiel en langage C
- La structure du noyau est monolithique et la notion de couche, contrairement à d'autres systèmes, n'existe pas.

Comme c'est le cas avec tout système d'exploitation, l'utilisateur d'Unix n'accède pas directement au noyau mais à un **interpréteur de commandes** : le shell (le choix de ce terme indique qu'Unix est « caché » à l'intérieur de cette coquille qui en est la seule partie visible par l'utilisateur).

Une différence importante entre Unix et les autres systèmes d'exploitation est qu'il existe plusieurs shells différents : richesse incomparable, mais source de confusion. Ce point sera abordé plus en détail au paragraphe 2.2 et au chapitre 6.

L'interface utilisateur d'Unix est donc constituée :

- D'un ensemble de programmes exécutables : les commandes.
- Du shell lui–même, interpréteur de commandes mais aussi, plus que dans n'importe quel autre système d'exploitation, langage de commandes permettant d'écrire des programmes d'une grande complexité. Ces programmes de commandes, appelés scripts, seront développés au chapitre 8.

Parmi les **utilitaires**, on trouve :

- différents langages de programmation : C++, Fortran , Java, Perl, TCL/TK, GTK;
- des utilitaires de développement et maintenance de logiciels : make, assembleur, éditeurs de lien ;
- des outils de bureautique : messagerie, traitement de textes ;
- des outils de mise au point de programmes ;
- des éditeurs de textes (sed, vi et vim, emacs, gnotepad) ;
- des formateurs de textes ;
- un système de messagerie complet (courrier, conversation en temps réel…) ;
- un analyseur syntaxique yacc, un générateur d'analyseur lexical lex ;
- un environnement graphique distribué : X11 ;
- les outils pour le Web (Apache, Netscape...)

1.5 VUE GÉNÉRALE D'UNIX

Unix est un système **multi–utilisateur** « temps partagé », c'est–à–dire qu'il est possible de connecter sous Unix plusieurs utilisateurs simultanément. Chacun a à sa disposition l'ensemble des ressources du système, le partage étant effectué par découpage du temps et récupération des temps morts d'entrée–sortie. Comme tout système multi–utilisateur, Unix comporte des mécanismes d'identification et de protection permettant d'éviter toute interférence (accidentelle ou malveillante) entre utilisateurs.

Unix est un **système multitâche**, c'est–à–dire qu'un utilisateur peut lancer plusieurs tâches simultanément. Un processus (ou tâche) correspond à l'exécution d'un programme à un instant donné, le programme étant en lui–même quelque chose d'inerte rangé sur disque sous la forme d'un fichier ordinaire exécutable.

Le **système de fichiers** est un système hiérarchisé arborescent. Il se retrouve sur beaucoup d'autres systèmes d'exploitation (GCOS, DOS, VMS…). Les entrées–sorties sont généralisées. Les périphériques sont considérés, du point de vue de l'utilisateur, comme des fichiers.

Le système est écrit à 99 % en C, ce qui facilite l'appel au noyau par des applications écrites en langage C. Ce système a été écrit de façon à être réellement **portable**.

Chapitre **2**

Connexion d'un utilisateur

2.1 L'UTILISATEUR

Pour permettre à de nombreux utilisateurs de travailler sur la même machine, Linux met en œuvre des mécanismes d'identification des utilisateurs, de protection et de confidentialité de l'information, tout en permettant le partage contrôlé nécessaire au travail en groupe. Tout utilisateur est identifié par un nom (**login name**) et ne peut utiliser le système que si son nom a préalablement été défini par l'administrateur du système (ou super–utilisateur), dont le nom est généralement **root**. Ce dernier a tous les droits et aucune restriction d'accès ne lui est applicable.

2.1.1 Connexion

Lors du démarrage d'une machine, plusieurs étapes se succèdent :

- mise sous tension de la machine et de ses périphériques,
- bootstrap du système (charger le noyau Linux),
- montage des disques,
- vérification des systèmes de fichiers (`fsck`),
- passage en multi–utilisateur,
- lancement des services.

On obtient alors, affichée à l'écran, l'invite **" login : "**

Le système Linux étant un système multi–utilisateur et multitâche, plusieurs personnes sont connectées simultanément et peuvent travailler sans interférer les unes avec les autres. Cela nécessite un système de protection des fichiers propre à chaque utilisateur, système que nous développerons au chapitre 4.

Ainsi pour qu'un utilisateur puisse travailler avec le système Linux, il doit établir une connexion (on dit aussi ouvrir une session).

Lorsque vous avez installé Linux, vous avez eu l'opportunité d'indiquer si vous vouliez utiliser un écran graphique (connexion en mode graphique) plutôt qu'une console (connexion en mode texte) pour ouvrir une session. Bien que le mode texte soit utilisé tout au long de cet ouvrage, excepté au chapitre 16, nous allons présenter les deux méthodes de connexion.

Connexion en mode texte

Pour qu'un utilisateur puisse travailler sur le système, il doit s'identifier en indiquant tout d'abord son nom suivi de la touche <return> (sur le clavier 102 touches, la touche ↵) après l'invite **login :**, puis son mot de passe suivi de la touche <return> à la suite de l'invite **passwd :**.

Lorsque l'utilisateur saisit son mot de passe, les caractères saisis ne sont pas affichés à l'écran (on dit qu'il n'y a pas d'écho des caractères sur le terminal). Ce mécanisme permet de garder la confidentialité du mot de passe.

Apparaît alors à l'écran un certain nombre d'informations (informations générales, date, arrivée de messages, date de dernière connexion). Puis le système lance un programme qui généralement est un interpréteur de commandes (shell). L'interpréteur indique par une chaîne de caractères, appelée **invite** (ou **prompt**), qu'il est prêt à recevoir une commande.

A partir de ce moment, l'utilisateur est connecté (il est entré en session).

Exemple

```
login : xstra
passwd :     § l'utilisateur xstra entre son mot de passe

******************************************
** Le systeme est arrete le Lundi 19/05/03 **
** pour maintenance                       **
******************************************

Lundi 12 Mai 2003
Vous avez dix messages

Derniere connexion : Vendredi 08 Mai 2003 a 18 h 04
xstra>
```

Dans cet exemple, l'utilisateur a pour nom `xstra` et `xstra>` correspond à l'invite (ou prompt) que nous utiliserons dans la suite de cet ouvrage.

Remarques

- En cas d'erreur lors de la saisie du nom ou du mot de passe, le système donne à l'utilisateur la possibilité de recommencer.

- Une erreur lors de la saisie du nom peut être annulée par la combinaison de touches <ctrl–u>.

Connexion en mode graphique

La procédure d'identification est la même en mode graphique et en mode texte. Une invite en mode graphique **login**, parfois nommé **username**, demande votre nom que vous saisissez avec le clavier et validez avec la touche < return >. Puis apparaît une fenêtre demandant votre mot de passe que vous devez saisir toujours suivi par la touche < return >, touche de validation de votre saisie. Comme en mode texte, ne soyez pas surpris, le mot de passe n'apparaît pas lors de sa saisie permettant ainsi la confidentialité. Cette étape est souvent source de difficulté car vous croyez saisir votre mot de passe et en fait les caractères que vous tapez ne correspondent pas à votre souhait. Le système, en cas d'erreur de saisie du mot de passe, refusera votre connexion.

Après avoir réussi votre connexion, c'est-à-dire votre combinaison valide « nom d'utilisateur et mot de passe », l'interface graphique X Window est démarrée (chapitre 16). Vous découvrez alors un bureau semblable à la figure 2.1 dans GNOME.

Dans cet ouvrage, les exemples et exercices sont présentés en mode texte. Pour les réaliser en mode graphique, il faut ouvrir un terminal qui émulera une console en mode texte. Chaque distribution de Linux offre plusieurs émulateurs de terminal dont le plus classique est « xterm ». Un moyen commun à la plupart des distributions est de cliquer sur le bouton droit de la souris et de choisir dans le menu déroulant qui apparaît la ligne nouveau terminal (en anglais New Terminal ou xterm).

Figure 2.1. Bureau Gnome 2.0 de la distribution RedHat 8.0.

2.1.2 Mot de passe

Lors d'une première connexion, il est fortement conseillé à l'utilisateur de s'attribuer un mot de passe. Ce mot de passe sera **chiffré** (le chiffrage est purement logiciel et non inversible). Il sera impossible de le retrouver à partir du mot chiffré, même pour le super–utilisateur (l'administrateur de la machine). Si l'utilisateur oublie son mot de passe, l'administrateur ne peut que le détruire pour lui permettre d'en définir un nouveau.

Un utilisateur peut à tout moment changer son mot de passe, ou s'en attribuer un par la commande *passwd*. Lors du changement, il faut fournir l'ancien mot de passe.

Exemple

```
xstra> passwd
Changing password for xstra
(current) UNIX password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

Par la suite, lors des diverses connexions de l'utilisateur, la lecture du mot de passe se fera sans écho. Souvent seuls les huit premiers caractères du mot de passe sont pris en compte. L'administrateur peut imposer des contraintes sur le mot de passe (six caractères minimum, un caractère non alphabétique,…).

L'administrateur de la machine pourra de plus installer des programmes qui testeront vos mots de passe afin d'interdire par exemple les mots du dictionnaire, les prénoms, les mots identiques à votre login, etc. Ce point de sécurisation de votre mot de passe sera vu au chapitre 15.

Lorsque le nom et le mot de passe sont corrects, **login** récupère dans le fichier */etc/passwd* toutes les informations utiles pour cet utilisateur.

Remarques

Si vous avez réalisé vous-même l'installation, une fois cette dernière terminée, connectez-vous au système en tant qu'administrateur (répondre root au **login :**). Il faut immédiatement définir un mot de passe pour cet utilisateur **root** aussi appelé super-utilisateur. Ce mot de passe doit contenir plus de six caractères. Il vous permettra de vous connecter en tant qu'utilisateur root et ainsi d'avoir tous les droits sur le système. Mais attention il ne faut jamais l'utiliser comme un compte personnel. Il doit être utilisé pour réaliser des modifications dans votre système. C'est pourquoi la première action d'un administrateur est de créer un compte personnel. Pour ceci, il suffit d'utiliser la commande *useradd*.

Exemple

```
root> useradd xstra      § création du compte xstra
root> passwd xstra
```

```
Changing password for xstra
(current) UNIX password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

Remarque

La commande *useradd* permet de créer un utilisateur en précisant des informations associées. Vous trouverez son développement en annexe C.

2.1.3 Fichiers /etc/passwd et /etc/group

La liste des **utilisateurs** du système est généralement dans le fichier */etc/passwd*. Ce fichier est accessible en lecture à tous les utilisateurs et contient, pour chaque utilisateur, les champs suivants :

. nom de connexion (login) de l'utilisateur,

. un caractère x

. le numéro de l'utilisateur (UID = user identifier),

. le numéro de groupe (GID = group identifier),

. [commentaire],

. le répertoire d'accueil,

. [programme à lancer].

Lors de la connexion, le programme désigné est lancé ; généralement il s'agit d'un interpréteur de commandes (shell).

Remarque

Les points cités entre crochets ne sont pas obligatoires.

Exemple

Voici un extrait du fichier */etc/passwd*.

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
ftp:x:14:50:FTP User:/home/ftp:
nobody:x:99:99:Nobody:/:
soline:x:518:504:Soline Waerzeggers:/home/soline:/bin/zsh
florent:x:519:504:Florent COLIN:/home/florent:/bin/bash
```

Un **groupe d'utilisateurs** rassemble un certain nombre d'utilisateurs pouvant facilement partager des fichiers. Ce groupe est répertorié dans un fichier (`/etc/group`) qui est constitué par :

. nom du groupe,
. un champ vide ou contenent un caractère x ou *
. numéro du groupe (GID = group identifier),
. [liste des utilisateurs membres du groupe].

Les notions d'UID et de GID sont importantes pour la protection des fichiers (il ne faut pas les changer inconsidérément). Un utilisateur a un groupe principal de rattachement et peut appartenir à plusieurs autres groupes.

Exemple

Voici un extrait du fichier `/etc/group`.

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
reseau:*:504:soline,florent
```

2.1.4 Déconnexion

En mode texte

Pour sortir de session, vous pouvez utiliser la commande `exit`. La déconnexion est effective lorsqu'apparaît l'invite **login :**.

En mode graphique

Pour sortir de session, il faut sélectionner avec le bouton gauche de la souris le menu démarrer puis choisir la commande *log out*. La déconnexion est effective lorsqu'apparaîtra l'invite graphique **login**.

2.2 LES SHELLS

Après toute entrée en session (**login**), le système positionne l'utilisateur dans son répertoire privé (**home directory** ou **répertoire d'accueil**), qui a été créé par l'administrateur du système au moment de l'ouverture de son compte et qui devient son répertoire de travail courant (current working directory). Puis le système active l'interpréteur de commandes désigné dans le dernier champs du fichier `/etc/passwd`. Il existe plusieurs interpréteurs de commandes : le **Bash**, le **TC–shell**, et d'autres.

Historiquement, le Bourne–shell fut le premier et a donc fait partie de la première version d'Unix. Il a été maintenu dans la version System V d'AT&T ainsi que dans la

version Berkeley (BSD). Il existe d'autres shells tels que le TC–shell (Tenex C–shell) de souche Berkeley, le Z–shell totalement libre, et le Bash, shell standard sous Linux.

Le TC–shell est un descendant très amélioré du C–shell d'origine Berkeley. Le shell standard de Linux est le Bash, descendant très amélioré du Bourne–shell. Cet interpréteur de commandes est présenté plus en détail au chapitre 6.

2.2.1 TC–shell

Le TC–shell doit son nom au fait que sa syntaxe est inspirée du langage C. Il possède des fonctionnalités supérieures à celles du Bourne–shell, entre autres :

- un mécanisme d'historique avec rappel et édition des commandes ;
- la possibilité de création d'alias ;
- des possibilités accrues pour le contrôle de processus.

Il exécute deux fichiers d'initialisation, qui sont les deux fichiers de commandes (scripts) du répertoire privé de l'utilisateur : `.cshrc` et `.login`. Puis l'interpréteur se met en attente d'une commande de l'utilisateur.

Le fichier d'initialisation `.cshrc` est exécuté lors de chaque appel au TC–shell. Il est destiné à positionner des variables du TC–shell, à affecter pour certaines commandes des arguments implicites et à définir d'autres commandes (alias). Il affecte une valeur :

- à la variable `prompt` qui définit l'invite,
- à la variable `path` qui contient les répertoires de recherche des commandes,
- aux variables nécessaires à la gestion de l'historique des commandes, etc.

Le fichier `.login`, qui est exécuté après le fichier `.cshrc`, une seule fois lors de la connexion, est destiné à positionner des variables d'environnement de la session. Il contient les initialisations standard, principalement les caractéristiques du terminal utilisé.

Les principales variables du TC–shell sont :

prompt	valeur du prompt.
path	précise le chemin d'accès aux répertoires contenant les principaux programmes.
home	indique le répertoire d'accueil de l'utilisateur.

En sortie de session, à l'aide de la commande `exit`, le fichier de commandes `.logout` est exécuté

2.2.2 Bash

Après une entrée en session (login) sur un système Linux, vous êtes pris en charge par un interpréteur de commandes (shell) qui est le Bash (`bash`). A la connexion,

avant l'interprétation des commandes, c'est à dire apparition du prompt, le Bash exécute les fichiers d'initialisation : `/etc/profile` et `.bash profile`.

Le fichier `/etc/profile` est généralement géré par l'administrateur système alors que le fichier `.bash profile`, se trouvant dans le répertoire d'accueil (Home Directory), est à la disposition de l'utilisateur. L'existence (ou non), l'accès en lecture d'un de ces fichiers, peut changer l'ordre d'accès à ces fichiers ou à d'autres fichiers, `.bash login` par exemple (cf détails au chapitre 6).

Ces fichiers permettent de modifier ou de créer des variables internes au shell, ou des variables d'environnement, de créer des fonctions, etc.

Les principales variables du Bash sont :

PS1	valeur du prompt.
PATH	précise le chemin d'accès aux répertoires contenant les principaux programmes.
HOME	indique le répertoire d'accueil de l'utilisateur.

En sortie de session, à l'aide de la commande *`exit`*, le fichier de commandes *`.bash logout`* est exécuté s'il existe.

2.3 COMMANDES LINUX

Il existe sous Linux de nombreuses commandes dont on trouvera les plus courantes en annexe A et que nous présenterons tout au long de cet ouvrage.

Une commande est composée en premier d'un **code mnémonique** (son nom proprement dit), suivi parfois d'options et/ou de paramètres. Une option dans une ligne de commandes Unix est une lettre. Elle commence par un tiret "–". Sous Linux, les options peuvent être de la forme :

`-c` c = caractère

`--mot` mot = un mot explicite

l'**espace** est le caractère séparateur des différents éléments d'une commande. Cette dernière est prise en compte et n'est interprétée que lorsque l'utilisateur a tapé la touche <**return**>.

Remarque

Pour obtenir la totalité des options d'une commande, il faut faire appel à l'option *`help`*.

Exemple

```
xstra> ls -i<return>      § option  i, style Unix
66 fich1   69 fich2
```

```
xstra> ls   inode<return> § option   inode, style POSIX
66 fich1   69 fich2
```

Il est possible, à tout moment avant la saisie de <return>, de modifier ou d'effacer une commande saisie au clavier et affichée à l'écran. La touche <backspace> (sur le clavier 102 touches, la touche ← situé au-dessus du <return>) ou la combinaison de touches <ctrl–h> permet d'annuler le dernier caractère saisi et ainsi de modifier la commande. La combinaison <ctrl–u> permet d'effacer tous les caractères situés à gauche du curseur.

Exemple

```
xstra> ls -lz § erreur de frappe : z en trop
               § la touche <backspace> permet
               § d'annuler le dernier caractère
xstra> ls -l
```

L'interpréteur Bash permet de modifier des commandes déjà exécutées et de les relancer (chapitre 6).

Linux autorise la frappe anticipée des caractères, c'est–à–dire qu'il est possible de saisir une réponse à une question sans que cette dernière soit apparue à l'écran (à n'utiliser que lorsque l'on a une parfaite maîtrise du système).

L'exécution d'une commande peut être interrompue à l'aide de la combinaison <ctrl–c>. Si vous avez saisi une commande inconnue du système, ce dernier vous l'indiquera par un message d'erreur.

Exemple

```
xstra> date
Tue May 9 10:50:24 CEST 2000
xstra> dte
bash: dte: Command not found.
xstra>
```

Si la commande existe, elle est exécutée et affiche à l'écran le résultat. En cas d'une erreur d'option, le programme indique l'erreur à l'écran et parfois affiche la liste des options disponibles.

Exemple

```
xstra> who
xstra pts/1 May 9 10:49
xstra pts/0 May 9 09:27
xstra> who   c
who: invalid option  c
Try `who   help' for more information.
xstra>
```

Remarque

Pour les utilisateurs de machines Linux, il est formellement déconseillé de redémarrer la machine (couper le secteur, reset). En effet, un arrêt brutal de Linux a pour conséquence de rendre le système de fichiers incohérent. Ainsi, **si vous êtes « planté »**, c'est–à–dire si l'une de vos applications bloque l'écran, que la combinaison de touches <ctrl–c> ne permet pas de détruire le programme, contrairement à ce que vous feriez avec le DOS, **surtout n'éteignez pas l'ordinateur.** Une solution existe : lancer un écran virtuel par la combinaison de touches <alt–ctrl–F2>. Dans cet écran, vous pourrez lancer des commandes de destruction de programmes et vous pourrez ainsi récupérer la main.

2.4 LE TERMINAL

Il est possible sous Linux d'activer simultanément plusieurs consoles de connexion. En mode texte, le basculement entre les consoles est obtenu par la combinaison de touches <alt–ctrl–Fx>, où Fx représente les touches de fonction F1 à F8 situées sur le haut de votre clavier.

Remarque

Pour les utilisateurs de machines Linux, il est formellement déconseillé de redémarrer la machine (couper le secteur, reset). En effet, un arrêt brutal de Linux a pour conséquence de rendre le système de fichiers incohérent. Ainsi, **si vous êtes « planté »**, c'est-à-dire si l'une de vos applications bloque l'écran, que la combinaison de touches <ctrl–c> ne permet pas de détruire le programme, contrairement à ce que vous feriez avec le DOS, **surtout n'éteignez pas l'ordinateur.** Une solution existe : lancez un écran virtuel par la combinaison de touches <alt–ctrl–F2>. Dans cet écran, vous pourrez lancer des commandes de destruction de programmes et vous pourrez ainsi récupérer la main.

Si vous êtes en mode graphique et que vous ne pouvez plus déplacer la souris, la combinaison de touches <alt–ctrl–backspace> permet de revenir à un mode texte. Le mode graphique est relancé à partir du mode texte par la commande ***startx***.

2.5 EXERCICES

Exercice 2.5.1

Après votre connexion au système, votre nom de login est jerome, modifiez votre mot de passe si vous estimez qu'il est trop facile à deviner.

Exercice 2.5.2

Vous exécutez la commande suivante :

```
| xstra> sleep 60
```

L'exécution de cette commande dure 1 minute. Comment interrompre cette commande et reprendre la main ?

Chapitre **3**

Système de fichiers

3.1 L'ARBORESCENCE

Le système de fichiers constitue un élément clé du système Linux. C'est encore à maints égards une des grandes forces de Linux. Vu par l'utilisateur, le système de fichiers est organisé en une structure arborescente dont les nœuds sont des **répertoires** et les feuilles **des fichiers ordinaires**.

Les fichiers sont des récipients contenant des données. Pour le noyau du système, ils ne contiennent qu'une suite d'octets plus ou moins longue. Linux en lui–même ne connaît ni enregistrement ni structure de fichier. C'est aux programmes d'application de les implanter et de les gérer. Le système Linux ne connaît que trois types de fichiers :

- **Les fichiers ordinaires (regular files).** Ils servent à mémoriser les programmes et les données des utilisateurs et du système.
- **Les fichiers répertoires ou répertoires (directories).** Chaque répertoire contient la liste et la référence des fichiers placés sous son contrôle et la référence du répertoire dont il dépend (répertoire père).
- **Les fichiers spéciaux**. Ils désignent les périphériques, les tubes ou autres supports de communication interprocessus. Les fichiers spéciaux associés aux périphériques peuvent être caractères (terminaux) ou blocs (disque) ; les entrées/sorties (E/S) se font soit caractère par caractère, soit bloc par bloc, un bloc étant composé de n caractères (512, 1024 ou 2048).

L'extension du système de fichiers aux périphériques constitue une des grandes particularités de Linux. Cela permet à un utilisateur de diriger les données produites

par son application vers un fichier ou un périphérique. Autre conséquence, les noms des fichiers et des périphériques obéissent à la même syntaxe et l'accès aux périphériques est contrôlé par le mécanisme de protection des fichiers.

Le système de fichiers est organisé en une seule arborescence logique. Cet arbre est composé de répertoires qui contiennent eux–mêmes d'autres répertoires, ou des fichiers. La figure 3.1 représente un exemple simplifié de cette arborescence.

3.2 LA CLASSIFICATION DES FICHIERS LINUX

Dans la présentation de la structure du système de fichiers au paragraphe précédent, nous avons évoqué des fichiers de type répertoire, des fichiers de type ordinaire et des fichiers spéciaux.

La syntaxe d'un nom de fichier n'est pas très stricte. Il est recommandé de limiter le nom d'un fichier à 14 caractères au plus et de n'utiliser que les lettres majuscules ou minuscules (attention, Linux différencie les majuscules des minuscules), les chiffres et quelques autres caractères (le point `.`, le tiret –, le souligné _). Linux autorise jusqu'à 255 caractères pour le nom du fichier. La longueur minimum est de un caractère.

Les caractères spéciaux suivants sont à proscrire absolument :

`\ > < | $ ? & [ ] * ! " ' ( ) ` @ ~ <espace>`

De plus, les utilisateurs ayant des claviers français doivent éviter les caractères accentués. En annexe B, vous trouverez la liste complète des caractères spéciaux à proscrire, car ils ont une signification particulière pour le système.

Le point (`.`) joue un rôle particulier dans le nom d'un fichier. Les fichiers dont les noms commencent par un point (`.`), comme *`.profile`*, sont des fichiers cachés (c'est–à–dire qu'ils n'apparaissent pas dans la liste des fichiers en tapant la commande *`ls`* sans argument).

Le point sert également à suffixer les noms des fichiers. Cette pratique est très recommandée, car elle facilite la gestion des fichiers. Il est vrai qu'il n'existe pas de syntaxe précise ; il existe toutefois un certain nombre de conventions :

`essai.c`	fichier source C
`include.h`	include de C
`essai.o`	fichier binaire objet
`essai.f`	fichier source fortran
`essai.c.old`	convention autorisée mais personnelle (fichier source C, ancienne version).

Certaines commandes de Linux s'appliquent à plusieurs fichiers. Dans ce cas, plutôt que de les énumérer, il est plus commode de les désigner par un nom générique en utilisant des caractères spéciaux, pour remplacer un ou plusieurs caractères dans le nom du fichier (voir paragraphe 7.7).

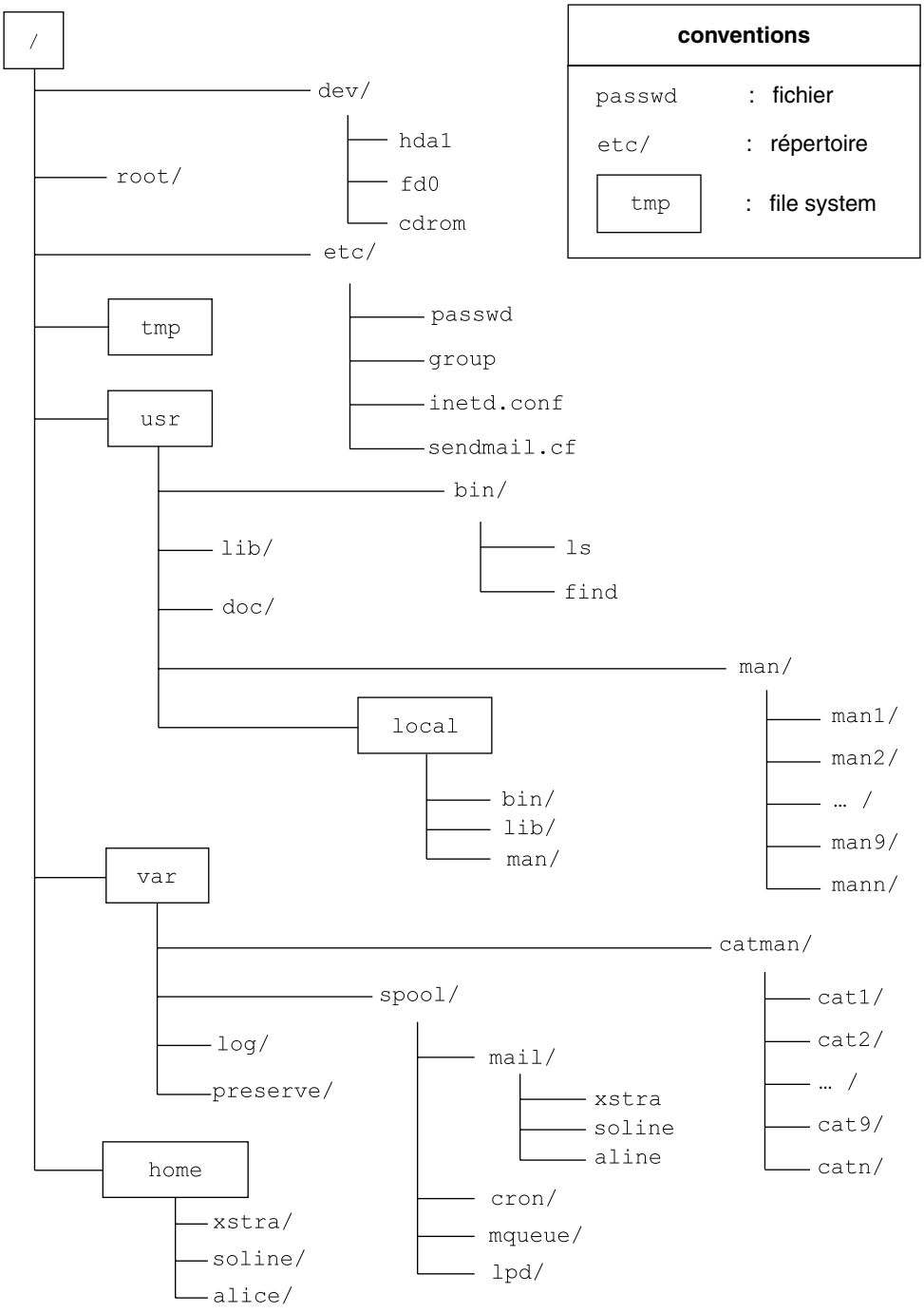


FIGURE 3.1. L'ARBORESCENCE DES FICHIERS LINUX.

Les caractères spéciaux et leur signification

`*`	désigne toute chaîne de 0 à n caractères,
`?`	désigne un caractère quelconque,
`[...]`	désigne un caractère quelconque appartenant à l'ensemble des caractères entre crochets.

Exemples

`fich.*`	désigne tous les fichiers de nom `fich` et ayant un suffixe.
`essai?`	permet d'obtenir tous les fichiers ayant un nom de 6 caractères dont les 5 premiers caractères sont `essai`, le dernier caractère est quelconque.
`[a f]`	désigne n'importe quelle lettre comprise entre `a` et `f`.
`[a z]*`	désigne tous les noms commençant par une lettre minuscule.

3.3 LA DÉSIGNATION DES FICHIERS

Un fichier est repéré par son nom et sa position dans l'arborescence : son chemin d'accès (**pathname**).

La syntaxe de ce chemin d'accès est très précise et peut être décrite des deux manières suivantes :

3.3.1 Le chemin d'accès absolu

Il permet d'accéder à un fichier quelconque dans l'arborescence du système de fichiers. Il est composé d'une suite de noms de répertoires séparés par le caractère `/`. Il commence toujours par le caractère `/` qui désigne le répertoire racine et se termine par le nom du fichier que l'on veut atteindre.

La longueur du chemin d'accès absolu d'un fichier est limitée à 1024 caractères.

Exemples

```
/var/spool/mail/xstra1
/home/xstra/essai
```

3.3.2 Le chemin d'accès relatif

La désignation d'un fichier par son chemin d'accès absolu se révèle rapidement lourde vu le nombre de répertoires intermédiaires à désigner. Tout utilisateur peut se positionner sur n'importe quel répertoire de l'arborescence. Ce répertoire devient courant (**répertoire de travail** ou **current working directory**).

Dès l'entrée en session de l'utilisateur, le système le place dans un répertoire de travail initial (**répertoire d'accueil** ou **home directory**). Ce répertoire a été créé au moment de l'établissement du compte de l'utilisateur. Le système associe alors en

permanence, à tout processus ou tâche, le chemin d'accès complet du répertoire de travail courant de l'utilisateur. Ainsi, l'usager peut désigner un fichier en ne donnant que son chemin d'accès relatif au répertoire de travail courant.

A partir de ce répertoire courant, l'utilisateur construit son propre sous–arbre de répertoires et de fichiers.

Exemple

chemin absolu	`/home/xstra/develop/prog1`
répertoire courant	`/home/xstra`
chemin relatif	`develop/prog1`

Remarque

Tout répertoire contient au moins deux entrées :

`.` § représente le répertoire lui–même.

`..` § représente le répertoire père.

Ces entrées de répertoire ne sont en général pas imprimées par les utilitaires du système. Elles permettent de référencer le répertoire courant sans l'obligation de citer son nom de chemin d'accès absolu ou de référencer avec un chemin d'accès relatif un fichier dans un sous–arbre.

Exemple

répertoire courant	`/home/xstra`
chemin d'accès relatif	`../xstra/develop/prog1`
chemin d'accès absolu équivalent	`/home/xstra/develop/prog1`

Par commodité, nous utiliserons dans la suite de l'ouvrage le terme nom de fichier pour désigner le chemin d'accès à ce fichier.

3.4 LA MANIPULATION DES RÉPERTOIRES

Après l'ouverture de sa session, l'utilisateur se trouve sous le contrôle d'un interpréteur de commandes. Celui–ci est prêt à lire, analyser et éventuellement exécuter les commandes qui lui sont soumises.

Chaque commande se compose d'un ensemble de champs séparés par un ou plusieurs blancs et se termine par une fin de ligne (<return> ou " line feed "). Le premier de ces champs est obligatoirement un nom de commande. Les autres champs définissent des paramètres dont l'interprétation dépend de la commande considérée. Nous reviendrons ultérieurement sur cette syntaxe.

Pour bien organiser son espace de travail, il est souvent utile de grouper ses fichiers par centre d'intérêt en créant des sous–répertoires. Les principales commandes pour gérer les répertoires sont :

pwd Print Working Directory

Affiche le chemin d'accès du répertoire courant. Juste après connexion d'un utilisateur xstra, la commande *pwd* lui précisera son répertoire d'accueil.

Exemple

```
xstra> pwd
/home/xstra
xstra>
```

cd Change Directory

Permet de changer de répertoire de travail.

Exemple

```
xstra> cd .. § Permet de remonter au
            § répertoire père.
xstra> pwd
/home
xstra>
```

Exemple

```
xstra> cd    § Permet de se repositionner
             § sur son répertoire d'accueil.
xstra> pwd
/home/xstra
xstra>
```

mkdir MaKe DIRectory

Crée un nouveau répertoire.

Exemple

xstra vient de se connecter. Il veut se créer un répertoire *perl* à partir du répertoire courant.

```
xstra> mkdir perl
xstra> cd perl
xstra> pwd
/home/xstra/perl
xstra>
```

rmdir ReMove DIRectory

Supprime un répertoire, s'il est vide.

Exemple

L'utilisateur décide de supprimer le répertoire précédemment créé. Ce répertoire est bien vide ; suppression possible.

```
xstra> cd
xstra> rmdir perl
xstra>
```

du Disk Usage

Donne l'occupation disque en bloc [un bloc valant 512 octets ou 1 Kilo–octets (Ko)] des sous–répertoires du répertoire spécifié ou, si aucun répertoire n'est précisé, du répertoire courant (nous reviendrons ultérieurement sur cette commande, en particulier dans le paragraphe 10.2).

L'utilisation de cette commande, et en particulier la capacité d'un bloc, est à vérifier sur votre machine par la commande *man du* ou *info du*.

find FIND

Recherche un fichier à partir du répertoire donné.

Exemples

1) Recherche du fichier *.bash profile* chez l'utilisateur connecté, puis affichage à l'écran de la liste des fichiers.

```
xstra> find . -name .bash_profile -print
```

2) Recherche de tous les fichiers de taille supérieure à 400 000 caractères à partir du répertoire courant et affichage à l'écran de la liste de ces fichiers.

```
xstra> find . -type f -size +400000c -print
```

3) Etant positionné sur le répertoire de l'utilisateur xstra, recherche de tous les fichiers de nom *core*. Puis suppression de ces fichiers.

```
xstra> cd /home/xstra
xstra> find . -name core -exec rm {} \;
```

3.5 LA MANIPULATION DES FICHIERS

Quel que soit le travail que vous allez faire sur la machine, vous aurez à effectuer certaines tâches élémentaires telles que lister le contenu d'un répertoire, copier, effacer, ou afficher des fichiers. Nous présentons ci–dessous brièvement les commandes qui les réalisent :

ls LiSt files

Permet d'obtenir la liste et les caractéristiques des fichiers contenus dans un répertoire. Si aucun argument n'est donné, la commande *ls* affiche la liste des noms des fichiers du répertoire courant par ordre alphabétique.

Exemples

1) Positionnement sur le répertoire de l'utilisateur xstra. *ls* permet d'obtenir la liste des fichiers et répertoires existants à ce niveau.

```
xstra> cd /home/xstra
xstra> ls
bin develop essai projet1
xstra>
```

2) Même démarche mais en voulant obtenir toutes les entrées.

```
xstra> ls -a
.          .bash_history   bin   projet1
..         .bash_logout develop
.bashrc    .bash_profile essai
xstra>
```

3) En étant à la racine, la commande suivante permet de lister le contenu du répertoire */home/xstra* et d'obtenir toutes les informations.

```
xstra> cd /
xstra> ls -l /home/xstra
total 8
drwxr xr x 2 xstra  512  jan 18 10:21 bin
drwxr xr x 2 xstra  512  jan 15 16:05 develop
 rwxr  r   1 xstra   15  jan 16 14:40 essai
drwxr xr x 2 xstra  512  jan 18 10:21 projet1
xstra>
```

cat conCATenate

La commande *cat* est une commande multi–usage qui permet d'afficher, de créer, de copier et de concaténer des fichiers.

Exemples

1) Affichage du contenu du fichier */etc/passwd*.

```
xstra> cat /etc/passwd
§ (cf. résultat dans l'exemple
§ du paragraphe 2.1.3)
```

2) Création d'un fichier

```
xstra> cat >essai
Bonjour
Il fait beau
<ctrl-d>      § caractère de fin de fichier
xstra>
```

locate

Permet d'afficher le nom complet de tout fichier ou répertoire correspondant à un critère de recherche donné.

Exemple

Recherche les fichiers et répertoires contenant la chaîne de caractères *touch*.

```
xstra> locate touch
/usr/share/man/man1/touch.1.gz
/usr/X11R6/man/man4/mutouch.4x.gz
/bin/touch
xstra>
```

more MORE

less LESS (jeu de mots sur « more or less » : *less* est une amélioration de *more*)

Permettent d'afficher page par page à l'écran le contenu d'un fichier texte. La commande *more* est traditionnelle. Un utilisateur de Linux doit lui préférer la commande *less*, équivalente mais plus élaborée. *less* est utilisée par la commande *man* pour l'affichage de la documentation en ligne.

Exemple

```
# afficher page par page le contenu du
# fichier /etc/passwd
xstra> less /etc/passwd
```

Voir l'annexe A pour l'utilisation de *more* et *less*. La commande *less*, qui permet de remonter en marche arrière dans le texte, est beaucoup plus configurable. La variable d'environnement *LESS* permet d'en fixer les options.

cp CoPy

Cette commande permet la copie de fichiers. Elle s'utilise sous quatre formes :

1) La copie d'un fichier source dans un fichier destination.

Exemple

Dans le répertoire *xstra*, copie du fichier *essai* dans *essai1*.

```
xstra> cd /home/xstra
xstra> cp essai essai1
xstra>
```

Remarque

Il n'existe aucun contrôle sur le fichier destination : si le fichier *essai1* existe, son contenu est écrasé par le contenu du fichier *essai*.

2) La copie d'un fichier dans un répertoire.

Exemple 1

Copie du fichier *essai* du répertoire *xstra* dans le répertoire *xstra/projet1*.

```
xstra> cd /home/xstra
xstra> cp essai /home/xstra/projet1
xstra>
```

Exemple 2

Copie du fichier *essai1* du répertoire père */home/xstra* vers le répertoire courant */home/xstra/projet1*.

```
xstra> cd /home/xstra/projet1
xstra> cp ../essai1 .
xstra>
```

3) La copie d'un répertoire dans un autre (seuls les fichiers sont copiés : on obtient un message d'erreur pour la copie des répertoires).

Exemple

Copie du contenu du répertoire *xstra* dans */home/xstra/projet2*.

```
xstra> cd /home/xstra
xstra> mkdir projet2
xstra> cp * /home/xstra/projet2
xstra>
```

4) La copie récursive permet de copier une arborescence.

Exemple

Copie de l'arborescence de *xstra/projet1* sous *xstra/projet2*.

```
xstra> cd /home/xstra/projet1
xstra> cp -r * /home/xstra/projet2
xstra>
```

mv MoVe

Change le nom d'un fichier ou d'un répertoire. En première analyse, cette commande est équivalente à une copie, suivie d'une suppression. Elle s'utilise sous deux formes :

1) Transfert de *fichier1* dans *fichier2* et suppression de *fichier1*. Si *fichier2* existe, il est effacé :
mv fichier1 fichier2

Exemple

Transfert du fichier *essai1* dans *toto*.

```
xstra> cd /home/xstra
xstra> mv essai1 toto
xstra>
```

2) Transfert de(s) fichier(s) cité(s) dans le répertoire avec le(s) même(s) nom(s) : *mv fichier(s) répertoire*

Exemple

Transfert du fichier *toto* dans le répertoire */home/xstra/projet1*.

```
xstra> cd /home/xstra
xstra> mv toto /home/xstra/projet1
xstra> ls toto
ls: toto: No such file or directory
xstra> ls /home/xstra/projet1/toto
toto
xstra>
```

Remarque

La philosophie de cette commande sera détaillée au chapitre 10.

rm ReMove
Supprime un (ou plusieurs) fichier(s) d'un répertoire :
rm fichier(s)

Exemple

Suppression du fichier *toto* du répertoire *projet1*.

```
xstra> cd /home/xstra/projet1
xstra> rm toto
xstra>
```

grep

Recherche, dans un ou plusieurs fichiers, de toutes les lignes contenant une chaîne donnée de caractères (cette commande sera détaillée au chapitre 14).

Exemple

Recherche de la chaîne de caractères *beau* dans le fichier *essai*.

```
xstra> grep beau essai
Il fait beau
xstra>
```

wc Word Count
Cette commande permet le dénombrement des mots, lignes et caractères dans un fichier. Un mot est défini comme une suite de caractères précédée et suivie par des espaces, des tabulations, le début ou la fin de la ligne.

wc lwcL fichier

l	affiche le nombre de lignes
w	affiche le nombre de mots
c	affiche le nombre de caractères
L	affiche la longueur de la ligne la plus longue
fichier	liste de noms de fichiers à parcourir (ou entrée standard si vide)

Exemple

Impression du nombre de mots dans le fichier *essai*.

```
xstra> wc  w essai
4
xstra>
```

ln LiNk
Permet de désigner un fichier par plusieurs noms différents.

Exemple

Le fichier *f1* existe, le fichier *New f2* est créé sans occupation disque et est lié au fichier *f1*.

```
xstra> ln f1 New_f2
xstra> ls
f1 New_f2
xstra>
```

Le fichier (en tant qu'espace disque) porte les deux noms.

Remarque

La philosophie de cette commande sera détaillée au chapitre 10.

touch TOUCH
Cette commande permet (entre autres) de créer un fichier vide.

Exemple

```
xstra> touch f1
xstra>
```

echo ECHO
Affiche à l'écran le texte qui suit la commande *echo*.

Exemple

```
xstra> echo Il y a du soleil
Il y a du soleil
xstra>
```

Nous venons de décrire de façon succincte quelques commandes de base de Linux. Ces commandes sont présentées en annexe A de façon plus détaillée avec d'autres options.

3.6 MANUAL, LE MANUEL LINUX

La commande *man* permet de rechercher des informations sur les commandes. *man* est le manuel Unix en ligne. Cette commande recherche les informations, le cas échéant, dans deux répertoires et leurs sous–répertoires :

/usr/man
/usr/local/man

La liste des répertoires *man* référencés est définie dans le fichier `/etc/man.config`.

L'exécution de la commande permettant d'obtenir des informations sur la commande *ls* est :

```
xstra> man ls § La documentation Linux relative
             § à ls apparaîtra à l'écran page par page.
xstra>
```

Lors de l'appel à cette aide en ligne, vous trouverez des chiffres entre parenthèses situés après les noms de commandes. Ces chiffres précisent à quel chapitre de la documentation Linux sont décrites ces commandes.

La commande *man* fait appel à *less* pour présenter l'aide page par page. La variable *MANPAGER* permet de configurer le comportement de *less* dans *man*. Il est commode de garder dans *man* les options habituelles de *less* :

Exemple

```
xstra> export LESS=' z 2 -rsiaj3$'   § la commande export
xstra> export MANPAGER="less $LESS"  § est décrite au
                                     § paragraphe 6.7
```

Dans le projet Gnu, la documentation n'est pas au format du man, mais dans un format plus récent permettant de produire indifféremment la documentation en ligne et des manuels papiers, et comportant des renvois d'une section vers une autre : le format texinfo. La commande *info* permet de consulter et d'imprimer cette documentation qui se trouve dans le répertoire `/usr/info`. Une grande partie des logiciels disponibles dans une distribution Linux étant d'origine Gnu, la commande *info* sera de fait préférée à la commande *man*, d'autant que la commande *info* renvoie automatiquement sur le *man* au cas où la documentation n'existe qu'au format man. Un excellent tutoriel sur la commande *info* est obtenu par la commande :

```
info info
```

Il est possible de constituer simplement sa propre documentation en créant dans un répertoire précis un fichier dans lequel sont mises les pages de manuel désirées. Pour réaliser cette manipulation, l'administrateur du système doit intervenir.

La figure 3.2 représente la sous–arborescence où se trouvent la documentation standard, et éventuellement la documentation locale.

Les fichiers qui se trouvent dans les répertoires *man1*,..., *man9* sont des fichiers formatés au format man. Dans les répertoires *cat1*,..., *cat9* sont stockés des fichiers en clair (éventuellement compressés par le programme *gzip*). Les fichiers qui ne sont pas dans les répertoires *cat1*,..., *cat9* sont mis en forme à partir de ce qu'il y a dans les répertoires *man1*,..., *man9*. La documentation technique fournie se situe aux niveaux :

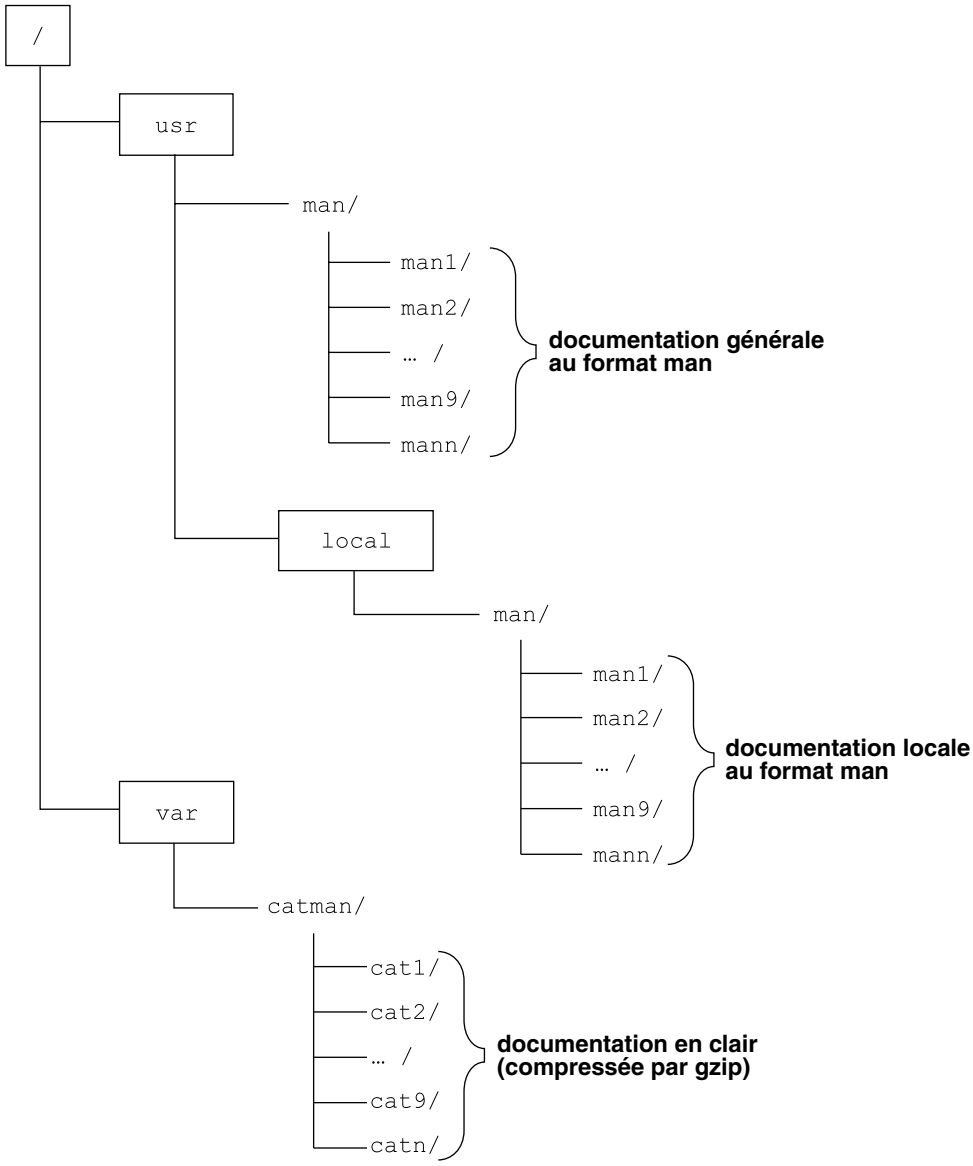


FIGURE 3.2. SOUS-ARBORESCENCE DU MANUEL.

/usr/man/man[1 9] pour le man de base
/usr/local/man/man[1 9] pour le man additionnel

Pour constituer une nouvelle documentation, l'administrateur du système positionnera les nouveaux fichiers dans l'un des deux répertoires suivants :

/usr/man/mann
/usr/local/man/mann

Les répertoires *mann* contiennent les fichiers formatés. Le suffixe de ces fichiers doit être *.n*.

Exemples

```
/usr/man/mann/madoc1.n
/usr/local/man/mann/madoc2.n
```

L'information sur *madoc1* ou *madoc2* sera obtenue à l'aide des commandes *man* ou *info* :

```
xstra> man madoc1
§ La documentation de madoc1 apparaîtra
§ page par page.
xstra> info madoc2 § idem
```

3.7 EXERCICES

Exercice 3.7.1

Dans votre répertoire d'accueil, créez l'arborescence suivante, en n'utilisant que des chemins relatifs :

```
rep1
 |---fich11
 |---fich12
 |---rep2
 |   |---fich21
 |   |---fich22
 |---rep3
 |   |---fich31
 |   |---fich32
```

Exercice 3.7.2

Vérifiez.

Exercice 3.7.3

Comment déplacer toute l'arborescence rep3 sous le répertoire rep2 ? Supprimez tout sauf rep1, fich11 et fich12.

Exercice 3.7.4

À l'aide de la commande `id`, déterminez votre UID et votre groupe (nom de groupe et GID). Combien y a-t-il d'utilisateurs dans votre groupe ?

Chapitre **4**

Protection des fichiers

Comme tout système multi-utilisateur, Linux possède des mécanismes permettant au propriétaire d'un fichier d'en protéger le contenu. Le propriétaire est l'utilisateur ayant créé le fichier. Pour permettre le partage de fichiers et faciliter le travail en équipe, Linux définit la notion de groupe d'utilisateurs, et tout utilisateur appartient à un groupe au moins (cf. § 2.1.3). Les **droits d'accès** (en anglais **permissions**) à un fichier sont définis par son propriétaire.

4.1 DROIT D'ACCÈS AUX FICHIERS

À chaque fichier est associé un ensemble d'indicateurs précisant les droits d'accès au fichier.

Pour chaque fichier il existe **trois types d'utilisateurs** :

- le propriétaire du fichier,
- les membres du groupe propriétaire du fichier,
- les autres utilisateurs du système.

Pour chaque fichier et par type d'utilisateur il existe **trois modes principaux** :

- autorisation d'écriture (w)
- autorisation de lecture (r)
- autorisation d'exécution (x)

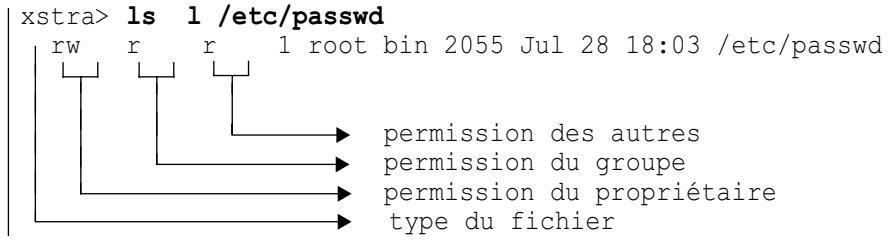
En plus de ces neuf bits (rwx rwx rwx), Unix/Linux définit trois autres bits de permission : **SUID**, **SGID**, **t** que nous présenterons plus loin.

Sous Linux il existe différents types de fichiers :

- fichier ordinaire ( )
- fichier répertoire (d)
- fichier spécial : périphérique accédé en mode caractère (c)
- fichier spécial : périphérique accédé en mode bloc (b)
- tube nommé (named pipe) (p)
- lien symbolique (l)
- socket (s)

Ainsi, à chaque fichier Linux sont associés 10 attributs (1 pour désigner le type, et 9 attributs de protection, 3 pour le propriétaire, 3 pour le groupe et 3 pour les autres utilisateurs). L'ensemble des renseignements sur un fichier est obtenu en utilisant la commande *ls l*.

Exemple



La chaîne rw r r représente les protections du fichier */etc/passwd*. Le premier caractère représente le type de fichier (ordinaire dans ce cas). Les trois caractères suivants indiquent les droits d'accès du propriétaire (rw), les trois suivants sont ceux du groupe (r) et les trois derniers (r) sont les droits d'accès des autres utilisateurs. Le champ suivant représente le nombre de liens sur ce fichier (un dans l'exemple). Les deux champs suivants indiquent l'utilisateur (**root**) et le groupe (**bin**) propriétaires du fichier. Ensuite apparaît la taille du fichier en octets, suivie de la date et de l'heure de la dernière modification du fichier. A la fin apparaît le nom du fichier.

Voici deux exemples de protection des fichiers :

crw rw r — Fichier spécial caractère : lecture et écriture pour le propriétaire et pour le groupe, et lecture pour les autres (par exemple les terminaux).

rwx r x r — Fichier ordinaire : lecture, écriture et exécution permises pour le propriétaire, lecture et exécution pour le groupe et seulement lecture pour les autres. Il est donc impossible aux membres du groupe et aux autres utilisateurs d'écrire dans ce fichier.

4.2 MODIFICATION DES DROITS D'ACCÈS AUX FICHIERS

4.2.1 Modification des droits d'accès

La protection d'un fichier ne peut être modifiée que par le propriétaire à l'aide de la commande *chmod* (CHange MODe). Il existe deux modes d'utilisation de cette commande. La première (la plus ancienne) utilise la **description des protections par un nombre octal**.

Exemple

`rwx rw r x` est représenté par le nombre octal 765. En effet, une lettre équivaut à 1, un tiret à 0.
On a donc `rwx rw r x = 111 110 101 = 765`

Exemple

Cet exemple va modifier la protection du fichier *toto* de la manière suivante :
`r x rw rwx` autorisant un accès lecture et exécution au propriétaire, lecture et écriture au groupe et un accès sans restriction aux autres (cas très rare).

```
xstra> chmod 567 toto
xstra> ls l toto
 r x rw rwx 1 xstra staff 55 Jul 20 17:01 toto
xstra>
```

Le deuxième mode d'utilisation de *chmod*, le mode symbolique, permet une **description absolue ou relative des droits d'accès**, comme suit :

```
chmod [who]op[permission] fichier
```

où

who	est une combinaison de lettre **u** (user=propriétaire), **g** (groupe), **o** (other=autre) ou **a** (all=tous) pour **ugo**,
op	**+** permet d'ajouter un droit d'accès, **–** de supprimer un droit d'accès et **=** d'affecter un droit de manière absolue (tous les autres bits sont remis à zéro),
permission	**r** (read=lecture), **w** (write=écriture), **x** (exécution).

Exemples

`chmod u w file`	supprime le droit d'écriture au propriétaire.
`chmod g+r file`	ajoute le droit de lecture pour le groupe.
`chmod ug=x file`	accès **uniquement** en exécution pour le propriétaire et le groupe, pas de modification pour les autres.

4.2.2 Droit d'accès à la création du fichier

La protection d'un fichier, ainsi que le nom du propriétaire (le vôtre) et le nom du groupe auquel vous appartenez, sont établis à sa création et ne peuvent être modifiés que par son propriétaire.

La commande *umask* permet de définir un masque de protection des fichiers (et répertoires) lors de leur création. Cette commande se trouve en général dans le fichier *.bashrc*, mais elle peut être exécutée à tout moment. Le masque est exprimé en base 8.

Exemples

```
xstra> umask 022
```

La valeur 022 est soustraite de la permission permanente (111 111 111) :

```
111 111 111   <= permission permanente
000 010 010   <= on enlève les bits dont on ne veut pas
111 101 101    => 755
```

umask 022 permet de créer des fichiers répertoires dont la protection est rwx r x r x. C'est souvent l'option par défaut.

Attention

Pour les fichiers ordinaires, *umask 022* donnera une protection de type rw r r , car **la possibilité d'exécution n'est pas autorisée sur les fichiers ordinaires**.

Voici donc ce qui se passe après avoir lancé *umask 022* :

```
xstra> touch f1
xstra> ls  l f1
 rw  r   r    1  xstra staff  40 jan 15 16:04  f1
xstra>
```

La forme symbolique de la commande *umask* est beaucoup plus agréable à l'utilisation que la forme octale. Acceptée par le bash et le Z-shell (mais pas par le TC-shell), elle est totalement cohérente avec la forme symbolique de la commande *chmod*, et devrait être préférée à la forme octale, considérée comme désuète.

Exemple

```
xstra> umask u=rwx,g=rx,o=rx
xstra> umask
0022
xstra> umask -S
u rwx,g rx,o rx
```

4.3 DROIT D'ACCÈS AUX RÉPERTOIRES

Dans le cas des répertoires, l'interprétation des droits est légèrement différente de celle concernant les fichiers. Les informations concernant un répertoire sont obtenues par la commande `ls dl rept`.

Exemple

```
xstra> ls dl bin
drwx r x r x 3 xstra staff 1024 Jul 28 18:04 bin
xstra>
```

L'interprétation des protections pour les répertoires est la suivante :

r autorise la lecture du contenu du répertoire comme dans le cas des fichiers; permet donc de voir la liste des fichiers qui sont dans le répertoire.

x autorise l'accès au répertoire (à l'aide de la commande `cd`).

w autorise la création, la suppression et le changement du nom d'un élément du répertoire. Cette permission est indépendante de l'accès aux fichiers dans le répertoire.

Exemples

Soit un fichier `f1` dans le répertoire `xstra1`.

1) Le répertoire `xstra1` appartenant à l'utilisateur xstra1 a les protections suivantes :

```
drwx
```

Le fichier `f1` de `xstra1` a les protections suivantes :

```
 rwx
```

Seul l'utilisateur xstra1 pourra modifier et supprimer son fichier `f1`.

2) Le répertoire `xstra1` appartenant à l'utilisateur xstra1 a les protections suivantes :

```
dr x
```

Le fichier `f1` de `xstra1` a les protections suivantes :

```
 rwx
```

Seul l'utilisateur xstra1 pourra modifier son fichier `f1` mais il ne pourra pas le supprimer. En effet le propriétaire du répertoire `xstra1` (l'utilisateur xstra1) n'a pas l'autorisation `w` (autorisation de création, suppression, modification du nom d'un élément du répertoire).

La permission de supprimer et le bit `t`

La possibilité de suppression d'un fichier n'est donc pas fixée dans les permissions de ce fichier, mais dans les permissions du répertoire qui le contient. Ce n'est donc pas une permission fixée fichier par fichier. Cela pose problème, en particulier dans le répertoire */tmp* : tout utilisateur doit pouvoir créer des fichiers dans */tmp*, qui a donc les permissions *rwx* pour *other*. Chacun peut donc lire **et écrire** dans */tmp*, et donc y créer des fichiers. Mais chacun pourrait aussi supprimer tout fichier dans */tmp*, y compris des fichiers créés par d'autres utilisateurs ! D'où l'utilité du bit *t* sur un répertoire : si ce bit est positionné, un utilisateur qui peut écrire dans le répertoire peut y créer des fichiers, mais ne peut supprimer un fichier que s'il en est propriétaire. Le bit *t* apparaît à la place du bit *x* de *other* dans la commande *ls -l* :

```
xstra> ls -ld /tmp
drwxrwxrwt 11 root root  3072 May 11 15:09 /tmp
```

4.4 MODIFICATION DU PROPRIÉTAIRE ET DU GROUPE

La commande *chown* permet de changer le propriétaire d'un fichier. Pour des raisons de sécurité, seul l'administrateur peut modifier le propriétaire d'un fichier ou d'un répertoire. Cet utilisateur privilégié, appelé **root**, peut accéder à tous les fichiers et répertoires sans aucune restriction et peut en modifier tous les attributs (nom, propriétaire, groupe propriétaire, permissions, dates). Il existe toujours dans tous système Unix/Linux.

La commande *chgrp* permet le changement de groupe pour les fichiers ou répertoires cités, à condition que l'utilisateur fasse partie du nouveau groupe et soit propriétaire de ces fichiers ou répertoires.

Exemple

```
xstra> chgrp info f1
 § possible si j'appartiens au groupe info
xstra> chown soline f1
chown: f1: Operation not permitted
 § possible seulement pour root
```

Le fichier *f1* appartiendra à *soline* et au groupe *info*.

Une commande équivalente (utilisable seulement par root) serait :

```
root> chown soline.info f1
```

4.5 APPARTENANCE À PLUSIEURS GROUPES

Lors de l'entrée en session, un utilisateur appartient à son groupe de rattachement principal : celui défini dans le fichier */etc/passwd* (voir le paragraphe 2.1.3). Cet utilisateur peut toutefois être membre d'autres groupes, tel que cela est défini dans le

fichier */etc/group* (voir le paragraphe 2.1.3). BSD et System V diffèrent sur les points suivants :

- Dans les versions d'Unix de souche BSD :

  Les permissions de groupe d'un fichier (ou d'un répertoire) sont applicables à tout utilisateur membre du groupe propriétaire du fichier. Tout fichier nouvellement créé a pour groupe propriétaire le groupe auquel appartient le répertoire dans lequel il est créé. (Un utilisateur peut donc créer un fichier appartenant à un groupe dont il n'est pas membre.)

- Dans les versions d'Unix de souche System V :

  Les permissions de groupe d'un fichier (ou d'un répertoire) ne sont applicables à un utilisateur que si son groupe **effectif** est le groupe propriétaire du fichier. Un utilisateur peut changer de groupe effectif à tout moment en effectuant la commande *newgrp nouveau groupe*, à condition d'être membre de *nouveau groupe*. Tout fichier nouvellement créé a pour groupe propriétaire le groupe **effectif** de son créateur.

Linux, comme d'autres versions d'Unix, réalise une synthèse entre ces deux comportements :

- Les permissions de groupe d'un fichier (ou d'un répertoire) sont applicables à tout utilisateur membre du groupe propriétaire du fichier, comme en BSD.
- Tout fichier nouvellement créé a pour groupe propriétaire le groupe **effectif** de l'utlisateur qui le crée si le répertoire qui le contient n'est pas SGID. C'est le comportement System V.
- Tout fichier nouvellement créé a pour groupe propriétaire le groupe auquel appartient le répertoire dans lequel il est créé si ce répertoire est SGID. C'est le comportement BSD.

Le choix entre le comportement System V et BSD peut donc être fixé répertoire par répertoire par positionnement du bit SGID sur le répertoire. Ceci est réalisé par les commandes :

```
chmod g+s repertoire  § met le bit SGID
chmod g s repertoire  § supprime le bit SGID
```

Si un répertoire est SGID, tout sous-répertoire créé ultérieurement héritera du bit SGID, et donc du comportement BSD. Ce comportement est très adapté au travail en groupe :

Exemple

L'utilisateur xstra fait partie du groupe staff et également du groupe reseau. Son groupe principal est staff. Voyons ce qui se passe dans un répertoire non SGID :

```
xstra> umask 002
xstra> ls  ld projet1
drwx rwx r x 2 xstra  staff  4096  jan 22 10:0  projet1
```

```
§ le répertoire projet1 n'est pas SGID
xstra> cd projet1
xstra> ls -l bonjour
 rwx r       1 pierre reseau 25  fev 10 10:25  bonjour
xstra> cat bonjour  § xstra fait partie du groupe reseau
Hello               § il peut donc lire le contenu
Comment vas tu?     § du fichier bonjour
xstra> touch prog2.c
xstra> ls -l prog2.c
 rwx rwx r   1 xstra  staff  0  fev 10 15:02  prog2.c
§ Le fichier prog2.c appartient au groupe staff
§ et non pas au groupe réseau.
```

Les autres membres du groupe réseau n'ont pas la permission de modifier le fichier prog2.c : nous ne sommes pas dans une logique de travail en groupe pour le groupe réseau. Voyons maintenant ce qui se passe dans un répertoire SGID :

```
xstra> umask u rwx,g rwx,o rx
xstra> ls  ld projet2
drwx rws r x 3 xstra  reseau  4096  jan 22 10:0  projet2
§ le répertoire projet2 est SGID
xstra> cd projet2
xstra> touch prog2.c
xstra> ls -l prog2.c
 rwx rwx r   1 xstra  reseau  0  fev 10 15:02  prog2.c
§ Le fichier prog2.c appartient au groupe reseau.
```

Dans ce dernier cas, tous les membres du groupe réseau pourront modifier tout fichier créé dans ce répertoire. Le travail en groupe est très simplifié. Une discussion beaucoup plus détaillée et très intéressante sur cette logique est disponible (en anglais) dans l'aide en ligne au format HTML, par exemple pour la distibution Red Hat dans :

Red Hat Linux Reference Guide

System Related Reference

System Administration

Users, Groups and User-Private Groups

4.6 EXERCICES

Exercice 4.6.1

En utilisant les commandes *mkdir, echo* et *cat*, créez dans un nouveau répertoire de nom "reptest" le fichier "bienvenue" contenant la ligne de commandes :

echo Bienvenue dans le monde Linux

Exécutez ce fichier.

Exercice 4.6.2

En utilisant les commandes `mkdir`, `echo`, `cp`, `chmod`, `cat`, créez un fichier que vous pouvez lire, modifier et supprimer.

Exercice 4.6.3

En utilisant les commandes `mkdir`, `echo`, `cp`, `chmod`, `cat`, créez un fichier que vous pouvez lire et supprimer mais que vous ne pouvez modifier.

Exercice 4.6.4

En utilisant les commandes `mkdir`, `echo`, `cp`, `chmod`, `cat`, créez un fichier que vous pouvez lire mais que vous ne pouvez ni modifier, ni supprimer.

Exercice 4.6.5

Dans quel cas les permissions d'un fichier à sa création sont-elles différentes des permissions fixées par `umask` ?

Exercice 4.6.6

Si vous pouvez travailler avec un collègue appartenant au même groupe que vous, modifiez les permissions du fichier créé à l'exercice "Bienvenue" ci-dessus de telle façon que votre collègue puisse le lire et l'exécuter, mais ne puisse pas le modifier ni le supprimer.

Pouvez-vous modifier les permissions de ce fichier de telle sorte que votre collègue puisse le lire, le modifier et l'exécuter alors que vous-même ne pouvez pas le modifier ?

Exercice 4.6.7

Comment est attribuée la permission d'effacer un fichier ? Créez un fichier que votre collègue peut modifier mais pas supprimer et un autre qu'il peut supprimer mais pas modifier. Est-il logique de pouvoir attribuer de tels droits ? Quelles sont les conséquences pratiques de cette expérience ?

Chapitre **5**

Éditeurs de texte

L'éditeur de texte de base de Linux est un programme appelé `ed`, écrit par Ken Thompson. `ed` a été conçu vers 1970 pour fonctionner dans un environnement de petites machines. Malgré les évolutions technologiques, `ed` n'a pas changé et il est aujourd'hui toujours disponible sur tous les systèmes Linux.

Sur Linux, on dispose bien sûr de l'éditeur `ed`, mais aussi d'un autre **éditeur ligne** `ex` et d'un **éditeur plein écran** `vi`. L'éditeur `ex` possède les mêmes commandes que l'éditeur `ed` à quelques améliorations près ; notamment les messages d'erreur sont donnés explicitement au lieu d'un simple ? sur `ed`.

Il existe d'autres éditeurs, en mode caractères (`emacs`,..) et en mode graphique X11 (`nedit`,...). Nous ne présentons ici que l'éditeur `vi` et un sous-ensemble des commandes de `ex` (recherche, substitution, copie, etc.). En effet, il est intéressant de connaître cet éditeur car il est disponible sur tous les systèmes Linux. Cet éditeur nous permettra, par exemple, l'écriture rapide de fichiers de commandes (voir chapitre 8).

Sous Linux, la commande `vi` lance l'éditeur `vim` correspondant à une version améliorée de `vi`.

L'éditeur `vi`, fonctionnant en pleine page, est essentiellement utilisé pour la création de nouveaux fichiers ou pour des modifications ponctuelles. `ex` est par contre plus adapté pour les recherches de motif ou pour des modifications systématiques. Comme sous `vi`, il est possible d'exécuter des commandes de `ex` en les faisant précéder d'un « : », ce sont les commandes de `ex` que nous utiliserons pour effectuer un traitement global (liste et effacement sélectif, substitution), les mouvements ou copies de groupes quelconques de lignes, la sauvegarde du fichier en cours d'édition ou la sortie de `vi`.

La figure 5.1 indique les cinq modes de fonctionnement de l'éditeur `vi` et les transitions possibles entre ces modes. Après l'appel de l'éditeur, `vi` est en mode

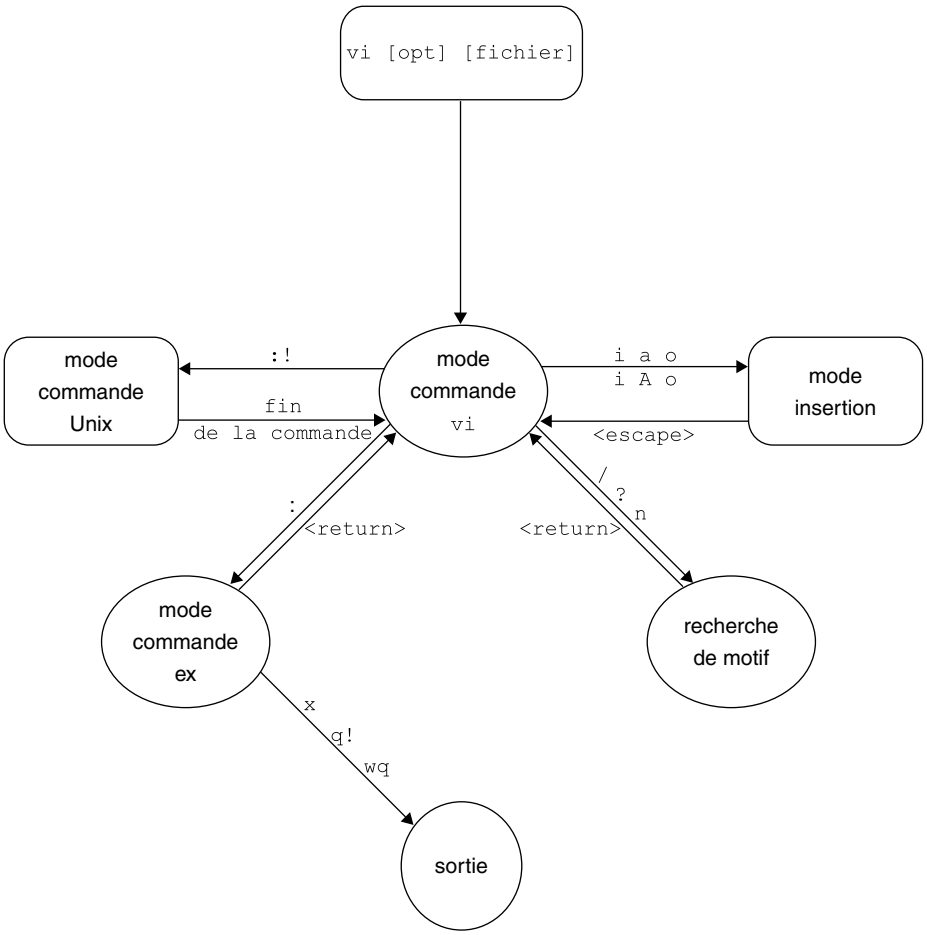


FIGURE 5.1. REPRÉSENTATION SYNOPTIQUE DU FONCTIONNEMENT DE L'ÉDITEUR *VI*.

commande, c'est-à-dire en attente d'une commande. Depuis ce mode, il est possible de passer :

- en mode insertion, le retour au mode commande étant réalisé à l'aide de la touche <Esc>,
- en mode recherche de motif,
- en mode commande de l'éditeur *ex*,
- en mode exécution d'une commande Linux.

Nous verrons que la plupart des commandes de *vi* correspondent à des touches du clavier ou à des combinaisons de touches. Cet éditeur fonctionne sur tout système

Linux du fait que *vi* a connaissance du type de terminal sur lequel vous travaillez grâce à la variable d'environnement TERM (voir chapitre 6). Cette variable indique à *vi* les caractéristiques du terminal.

Remarque

Si la variable d'environnement TERM n'a pas été positionnée, à l'appel de *vi*, un message d'erreur vous indiquera que l'éditeur ne reconnaît pas le type de terminal (voir chapitre 6).

5.1 L'ÉDITEUR PLEINE PAGE VI

5.1.1 Appel de l'éditeur et sorties

`vi nom de fichier`	appel de l'éditeur ; le curseur se positionne au début du fichier,
`vi r nom fichier`	idem mais pour récupérer un fichier après un problème machine,
`vi R nom fichier`	appel de l'éditeur mais avec verrouillage du fichier en lecture seule,
`ZZ (ou :x)`	mise à jour du fichier et sortie de l'éditeur,
`:wq`	idem à ZZ,
`:q!`	sortie de l'éditeur sans modification du fichier.

5.1.2 Renseignements utiles

`:set ts`	affiche le nombre de caractères d'une tabulation,
`:set ts=n`	fixe la tabulation à n caractères (par défaut n=8),
`:set nu`	affiche les numéros de lignes,
`:set nonu`	supprime les numéros de lignes,
`:.=`	affiche le numéro de la ligne courante,
`<ctrl g> (ou :f)`	affiche le nom et l'état du fichier.
`:e!`	Réédite le fichier tel qu'il est sur le disque

5.1.3 Déplacements de la page affichée

`<ctrl b>`	page précédente (back),
`<ctrl f>`	page suivante (forward).

5.1.4 Déplacements du curseur

La saisie d'un des caractères suivant nous amène dans la position indiquée.

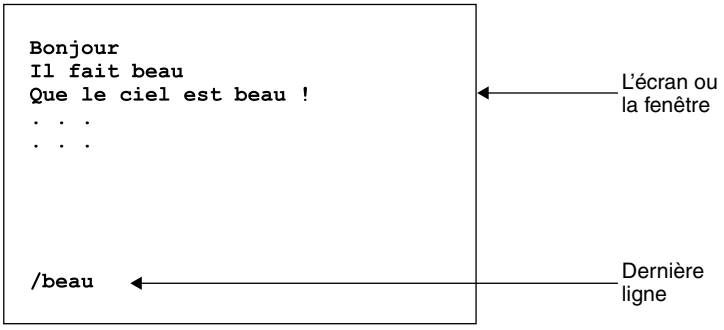
`G`	Dernière ligne,
`nG`	Ligne n,
`H`	Haut de l'écran (high),
`M`	Milieu de l'écran (middle),
`L`	Bas de l'écran (low),
`n<espace>`	n caractères vers la droite,
`0 (zéro)`	Début de ligne,
`^`	Premier caractère non blanc,
`$`	Dernier caractère de la ligne,
`↑ (ou k)`	Ligne précédente,
`← (ou h)`	Vers la gauche,
`→ (ou l)`	Un caractère vers la droite,
`↓ (ou j)`	Ligne suivante,
`n<return>`	n lignes vers le bas,
`+ ou <return>`	Début de la ligne suivante,
	Début de la ligne précédente.

5.1.5 Recherche

`/motif<return>`	recherche motif vers l'avant,
`?motif<return>`	recherche motif vers l'arrière,
`n`	répète la dernière commande de recherche.

Exemple

Voici un texte en vi :



La dernière ligne visible à l'écran en *vi* est une ligne de commandes. Lors de la mise à jour du document, la ligne « /beau » n'est pas sauvegardée.

5.1.6 Insertion

Le mode insertion est obtenue en tapant au choix I, i, a, A, o, O suivi du texte à insérer et terminé par la touche <Esc>. Cette touche remet l'éditeur en mode commande (en cas de doute, taper encore une fois <Esc> ; le signal sonore fera savoir que l'on n'est plus en mode insertion). Les commandes d'insertion sont :

`i… <Esc>`	avant le curseur,
`I… <Esc>`	en début de ligne courante,
`a… <Esc>`	après le curseur,
`A… <Esc>`	en fin de ligne courante,
`o… <Esc>`	au-dessous de la ligne courante,
`O… <Esc>`	au-dessus de la ligne courante.

5.1.7 Caractères spéciaux en mode insertion

`<del> ou <backspace>`	efface le dernier caractère inséré,
`<ctrl u>`	annule le texte inséré de la ligne courante,
`<ctrl v>`	permet d'insérer le caractère spécial qui suit,

5.1.8 Remplacement

`r#`	remplace le caractère courant par le caractère #,
`R… <Esc>`	remplace la chaîne de caractères courante par les caractères compris entre R et <Esc>,
`~`	bascule de MAJUSCULE en minuscule et inversement,
`.`	répète la dernière commande modifiant un texte.

5.1.9 Effacement

`x`	du caractère sous le curseur,
`3x`	de trois caractères à partir de la position du curseur,
`dd`	de la ligne où se trouve le curseur,
`4dd`	de quatre lignes à partir de la ligne où est le curseur.

5.1.10 Restitution

`u`	dernière modification (*undo*),
`U`	la ligne courante avant modification (sauf *dd*),
`p`	dernière commande d'effacement.

5.1.11 Mouvements de lignes

`J`	joint la ligne courante et la suivante,
`r<return>`	brise en deux lignes à partir de la position du curseur,
`yy`	place la ligne courante dans le tampon de travail,
`dd`	idem avec effacement de la ligne,
`10yy`	place dix lignes dans le tampon de travail,
`10dd`	idem avec effacement des dix lignes,
`p`	insère le contenu du tampon sous la ligne courante,
`"ayy`	place la ligne courante dans le tampon a (un tampon nommé se conserve au changement de fichier),
`"anyy`	place n lignes dans le tampon a,
`"ap`	place le contenu du tampon a après la ligne courante.

5.1.12 Décalage

`>> (ou <<)`	décalage vers la droite ou vers la gauche de toute la ligne courante de m positions, m étant défini par la commande *`:set sw=m`* (par défaut m=8),
`n>> (ou n<<)`	décalage des n lignes suivantes,
`:n1,n2 >> (ou <<)`	décalage des lignes n1 à n2.

5.2 LE MODE COMMANDE DE L'ÉDITEUR EX SOUS VI

L'éditeur ligne *`ex`* est capable de réaliser sur un texte les opérations suivantes :

- recherche d'une chaîne de caractère (motif),
- déplacement et duplication de lignes,
- substitution de chaînes de caractères,
- manipulation de fichier.

L'éditeur *`vi`* passe en mode *`ex`* lorsque l'utilisateur frappe le caractère *`:`* en mode commande *`vi`*.

Le passage en mode *`ex`* est matérialisé par l'apparition du caractère *`:`* sur la dernière ligne de l'écran, indiquant l'attente d'une commande *`ex`*. Cette commande est validée par la touche *`<return>`*.

5.2.1 Listage sélectif et recherche de motif

`:g/motif[ /p]`	liste les lignes contenant *`motif`* dans tout le fichier,
`:m,ng/motif[ /p]`	liste les lignes contenant *`motif`* entre les lignes *`m`* et *`n`*.

5.2.2 Déplacement et duplication de lignes

`:i,jmok`	déplace les lignes *i* à *j* en les mettant après la ligne *k*,
`:i,jcok`	copie les lignes *i* à *j* en les mettant après la ligne *k*.

5.2.3 Substitution de chaînes de caractères

`:s/old/new[ /p]`	substitution du premier *old* de la ligne courante par *new*,
`:s/old/(&)[ /p]`	substitution du premier *old* de la ligne courante par *(old)*,
`:s/old/new/gp`	substitution de tous les *old* de la ligne courante par *new*,
`:i,js/old/new/p`	substitution des premiers *old* des lignes *i* à *j* par *new*,
`:i,jg/old/s//new/p`	idem, mais toutes les lignes modifiées sont listées,
`:i,js/old/new/g`	substitution de tous les *old* des lignes *i* à *j* par *new*,
`:g/old/s//new/g`	substitution des *old* de tout le fichier par *new*,
`:g/old/s//new/gp`	idem, et liste toutes les lignes modifiées,
`:[ n] s[ g]`	répète la plus récente substitution sur la ligne courante ou la ligne numéro *n*,
`:u`	*undo*, annule la plus récente substitution réalisée.

Dans les exemples ci-dessus, i et j peuvent prendre les valeurs suivantes :

- `.` la ligne courante
- *$* la dernière ligne
- *%* tout le fichier

Les expressions régulières utilisées en substitution comportent
/ \ & . * [ ] ^ et **$** (voir chapitre 14).

/	séparateur de chaînes (peut être un caractère quelconque, sauf si risque de confusion),
\	caractère de neutralisation (\& désigne le caractère &),
&	chaîne de caractères à remplacer (la dernière chaîne connue),
•	un caractère quelconque (équivalent à ? dans les commandes de l'interpréteur shell),
*	un nombre quelconque de fois le caractère qui le précède,
[ ]	n'importe quel caractère dans l'intervalle [ ] (ex : [1–4]=[1234]),

`^`	début d'une ligne (ou négation si ce caractère se trouve à l'intérieur et au début de [^…]),
`$`	fin d'une ligne (s/$/./ ajoute un point à la fin de la ligne courante).

5.2.4 Insertion et écriture de fichier

`:n rnom de fichier`	insère le fichier nommé après la ligne *n*,
`:r nom de fichier`	insère le fichier nommé après la ligne courante,
`:r!cmd`	idem, mais avec la sortie de la commande *cmd*,
`:w!nom de fichier`	crée un nouveau fichier avec le contenu du fichier courant,
`:i,jw!nom de fichier`	idem, avec les lignes *i* à *j* du fichier courant,
`:w>>nom de fichier`	ajoute le ficher courant à la fin du fichier nommé,
`:w!nom de fichier`	copie le fichier courant sur le fichier nommé.

5.3 PERSONNALISER VI

Le fichier d'environnement *.exrc* permet de personnaliser le fonctionnement de *vi*. Ce fichier se trouve dans le répertoire d'accueil (HOME). Il contient les commandes permettant de positionner les options de *vi*, et de créer de nouvelles commandes.

5.3.1 Les commandes set

La commande *set* permet de positionner les options de *vi*. Sa syntaxe générale est :

```
:set option[ valeur]
```

Dans *vi*, la commande *:set all* affiche les paramètres positionnés à l'aide de la commande *set*.

Exemples

`:set nu`	permet la numérotation des lignes,
`:set nonu`	enlève la numérotation des lignes,
`:set ai`	permet l'indentation automatique
`:set noai`	enlève l'indentation automatique,
`:set sw=3`	décalage de ligne de 3 caractères,
`:set nomesg`	pas d'intrusion de message dans *vi*,
`:set wrapscan`	lors de la recherche d'une chaîne de caractères, recherche circulaire,
`:set showmode`	indique dans quel mode on se trouve (*Input Mode* ou *Replace Mode*),

`:set showmatch`	positionne pendant une seconde le curseur à la parenthèse correspondant à celle saisie.

5.3.2 Les commandes map

La commande *map* permet de créer de nouvelles commandes sous *vi* par la combinaison de commandes existantes. Sa syntaxe générale est :

```
:map nouvelle_commande ensemble_commandes_vi
```

Cette commande permet d'affecter à une touche spéciale du clavier (qui génère une séquence particulière de caractères) une série de commandes de *vi*. C'est ce que l'on appelle le **mapping**.

Si la séquence de caractères *nouvelle commande* ne comporte qu'un seul caractère, il ne doit pas être numérique. Si cette séquence comporte plus qu'un caractère, elle ne doit débuter ni par un caractère alphanumérique, ni par le caractère *:* (deux points).

Exemples

1) Si la touche du clavier **home** génère le caractère *g*, il est possible d'affecter à cette touche le positionnement en début de fichier à l'aide de la commande *map* ci-dessous.

```
:map g 1G § Le caractère g nous positionne
          § en début de fichier.
```

2) Il est parfois intéressant de rajouter une chaîne de caractères à la suite d'un mot sans quitter le mode commande.

```
:map  @  i@machine.u strasbg.fr<ctrl V><Esc>l
# en saisissant @ en mode commande, vi insère la chaîne
# @machine.u strasbg.fr et repasse en mode commande
```

L'exécution de la commande *:map* sous *vi* permet d'obtenir les nouvelles commandes créées par *map*.

5.3.3 Les commandes map! en mode insertion

La commande *map!* permet de créer de nouvelles commandes sous *vi* par la combinaison de commandes existantes en mode insertion. La syntaxe générale est :

```
:map!  nouvelle_commande  ensemble_commandes_vi
```

Cette commande permet d'affecter à une touche spéciale du clavier (qui génère une séquence particulière de caractères) une série de commandes de *vi* (le mapping) en mode insertion.

Exemple

Lors d'une saisie de texte en mode insertion, on souhaite supprimer la fin de la ligne à l'aide de la touche F2 et poursuivre la saisie. Il est possible d'affecter à cette

touche le basculement en mode commande à l'aide de la touche <Esc>, puis supprimer la fin de ligne (d$), et enfin revenir en mode insertion. La commande *map!* ci-dessous le permet :

```
:map!  <F2>  <ctrl v><Esc>ld$a
```

L'exécution de la commande *:map!* sous *vi* permet d'obtenir les nouvelles commandes créées par *map!*.

On obtient l'insertion d'un caractère spécial (en mode insertion) en le faisant précéder de la combinaison *<ctrl v>*. Par exemple, l'insertion du caractère *<Esc>* est obtenue par *<ctrl v><Esc>*.

Chapitre **6**

L'interpréteur de commandes : Bash

Après une entrée en session (login) sur un système Linux vous êtes pris en charge par un **interpréteur de commandes** (shell) : le Bash (`bash`). L'interpréteur de commandes est l'interface entre l'utilisateur et le système d'exploitation. Il a pour rôle de traduire les commandes saisies par l'utilisateur afin que le système puisse les exécuter.

Il existe plusieurs interpréteurs de commandes. Le Bourne-shell (`sh`) est l'ancêtre commun de tous les shells. Il est obsolète en interactif mais encore utilisé en programmation. Le C-shell (`csh`) d'origine BSD, et sa version plus récente le tc-shell (`tcshell`), est utilisable en interactif mais non compatible avec le Bourne-shell en programmation. Le Korn-shell (`ksh`) est l'interpréteur de commandes le plus répandu dans le monde Unix. Il est très agréable en interactif et compatible ascendant en programmation avec le Bourne-shell. Le Bash est l'interpréteur de commandes le plus répandu dans le monde Linux. Le Bash est compatible avec le Bourne-shell. Il inclut des caractéristiques du Korn-shell et du C-shell. Il offre des améliorations aussi bien en interactif qu'en programmation.

Il est également possible de créer des fichiers (avec `vi` ou `emacs` par exemple) contenant une suite de commandes. Ces fichiers sont appelés des **scripts** et peuvent être exécutés, ce qui évite la réécriture de ces commandes. De surcroît les interpréteurs de commandes possèdent un véritable langage de programmation interprété permettant la réalisation de programmes de commandes complexes.

A ce stade il nous est nécessaire d'introduire une notion importante du système d'exploitation Linux : **le processus**. Un processus correspond à l'exécution d'un programme à un instant donné. Le lancement d'un processus correspond donc au

chargement en mémoire centrale d'un fichier exécutable rangé sur le disque en vue de son exécution.

Pour chaque utilisateur, le fait de se connecter provoque le lancement d'un processus. Ce processus, défini dans le fichier */etc/passwd*, est généralement l'interpréteur de commandes Bash (*bash*).

Un processus est reconnu par un numéro appelé le **PID (Process Identifier)**. Il est possible, à partir du processus interpréteur de commandes, de lancer une nouvelle commande. Dans ce cas le processus correspondant à la commande est appelé le **processus fils**, l'interpréteur de commandes étant le **processus père**. Il est également possible, à partir d'un shell, de lancer un autre shell, soit pour exécuter une commande, soit pour être pris en charge par le nouveau shell.

On peut classer les commandes en trois catégories :

- Les commandes définies par une fonction shell ou par un alias. C'est une façon de renommer une commande.
- Les commandes internes qui font partie intégrante du shell ; il n'y a pas création d'un nouveau processus lors de l'exécution de ces commandes.
- Les commandes externes qui sont indépendantes du shell et qui se trouvent dans différents répertoires (*/usr/sbin*, */sbin*,...). À chaque lancement d'une de ces commandes il y a création d'un nouveau processus (processus fils).

L'ordre indiqué ci-dessus est celui dans lequel le shell recherche la commande. Par exemple, en Bash, si un alias porte le même nom qu'une commande, le shell exécutera l'alias.

Exemple d'un alias

```
xstra> alias ls="echo Bonjour"
xstra> ls -l /tmp
Bonjour  l /tmp
xstra> unalias ls
xstra>
```

Les variables shell

Tout interpréteur de commandes permet de définir des variables shell. Une variable est définie par son nom et contient une valeur.

Le nom d'une variable shell est une chaîne de caractères alphanumériques, le premier étant alphabétique. La valeur d'une variable shell est une chaîne de caractères.

Les **variables internes** ne sont connues que par le shell, **les variables d'environnement** sont connues par le shell et les processus fils de ce shell (lancés par le shell).

Remarques

- Le caractère _ (souligné) est le seul caractère non alphanumérique qui peut apparaître dans un nom de variable.

- La syntaxe de définition et d'assignation des variables internes et d'environnement sera explicitée plus loin.

6.1 LES FICHIERS D'INITIALISATION

À la connexion, avant l'interprétation des commandes, c'est-à-dire apparition à l'écran du prompt, le Bash interprète les fichiers d'initialisation : `/etc/profile` et `.bash profile`. Le fichier `/etc/profile` est géré généralement par l'administrateur alors que le fichier `.bash profile`, qui est dans le répertoire d'accueil (HOME), est à la disposition de l'utilisateur. Ces fichiers permettent de modifier ou de créer des variables internes au shell, ou des variables d'environnement, de créer des fonctions, etc. Si le fichier `.bash profile` n'existe pas ou n'est pas en lecture, alors le Bash recherche dans votre répertoire d'accueil (HOME) et dans cet ordre, le fichier `.bash login` puis `.profile` et interprète le premier qui existe et qui est en lecture.

Lorsque le Bash est utilisé en mode interactif et n'est pas un shell obtenu après une connexion, il exécute le fichier `.bashrc`, s'il existe. Souvent l'utilisateur intègre dans le fichier `.bash profile`, l'exécution du fichier `.bashrc` (voir exemple ci-dessous) afin d'obtenir une initialisation commune au shell de login.

Enfin, lorsque le Bash est invoqué par un fichier de commandes (un script, voir chapitre 8), il exécute un fichier dont le nom a été défini dans la variable d'environnement BASH_ENV.

Exemple

Extrait d'un fichier `.bash profile`

Exemple

Extrait d'un fichier `.bash profile`

```
xstra> less .bash_profile
# .bash_profile
# Définition des alias et fonctions
if [   f ~/.bashrc ]; then
        . ~/.bashrc
fi
# Environnement utilisateur spécifique
# et programmes de démarrage
PATH $PATH:~/bin:.      # Chemins d'accès des commandes
PS1 '\u@\h:$PWD \$ ''   # Valeur du prompt
#
export PATH PS1         # Variables d'environnement
export BASH_ENV ~/.bashscript
#  Initialisation des protections par défaut
umask u rwx,g rx,o rx
```

```
# Initialisation historique
set  o history
export HISTSIZE 10000
# Sortie de session au bout de 30 minutes
TMOUT 1800
unset USERNAME
xstra>
```

Exemple

Extrait d'un fichier *.bashrc*

```
xstra> less .bashrc
# .bashrc
# Définitions globales
if [   f /etc/bashrc ]; then
        . /etc/bashrc
fi
# Environnement utilisateur spécifique
# et programmes de démarrage
export PS1 '\u@\h:$PWD \$ '
# Alias spécifique à l'utilisateur
alias lsf 'ls  CF'
alias bdf 'df'
alias bye exit
alias H history
alias del ' rm  i'
alias copy ' cp  i'
alias move ' mv  i'
alias md mkdir
alias date "date +%a' ' %d %m %y' ' %T"
alias pvar "declare  p"
# Fonction de l'utilisateur
rgrep () {          # Fonction rgrep
grep  r   include "$2" $1 .
}
f () {              # Fonction f
find .  name "$1"  print 2>/dev/null
}
# Les options Bash
shopt  s extglob
# pas de fichier core
ulimit  c 0
xstra>
```

Exemple

Extrait d'un fichier *.bashscript* dont le nom a été défini dans la variable BASH ENV.

```
xstra> less .bashscript
# Fichier d' initialisation des scripts
```

```
umask u rwx,g rwx,o rx
PATH "/bin:/usr/bin:/usr/local/bin:~/bin"
xstra>
```

La commande *exit* et éventuellement *<ctrl d>* permet de quitter la session. En cas de fin de connexion, le fichier de commandes *.bash logout* est exécuté s'il existe. Il contient d'habitude des commandes de suppression de fichiers temporaires (*core*, *tmp*, etc.), d'effacement d'écran, de message de fin de session, etc.

Exemple

Exemple de fichier *.bash logout*

```
#
#  Fichier de fin de connexion Bash
echo " ** Fin de connexion **"
unalias rm
nohup find $HOME  name core  exec rm {} \; 2>/dev/null&
clear
```

6.2 LES VARIABLES DU BASH

Un certain nombre de variables ont pour le Bash une signification prédéfinie. Nous ne présenterons que les plus importantes. Il est possible de classer les variables du Bash en deux catégories : les variables booléennes et les variables ordinaires.

a) Les variables booléennes

Ces variables permettent, selon qu'elles sont positionnées (*set o option*) ou non (*set +o option*), de modifier le comportement de l'interpréteur de commandes. Chacune de ces options est contenue dans une variable ordinaire SHELLOPTS. Les principales sont les suivantes :

history	Autorise le mécanisme d'historique (voir paragraphe 6.5.1)
noclobber	Cette variable permet d'interdire la redirection d'une commande sur un fichier existant (évite de l'écraser).
notify	Si cette variable est positionnée, le Bash indique immédiatement la fin d'un travail en arrière-plan (background job).
posix	Met le Bash en conformité POSIX1003.2.
xtrace	Affiche, après expansion, chaque ligne de commandes précédée du contenu de PS4 (+ par défaut).

b) Les variables ordinaires

La liste des variables et leur valeur sont obtenues à l'aide de la commande *set* sans argument.

Exemple

```
xstra> set
HOME /home/xstra
IFS
...
LOGNAME xstra
PATH /bin:/usr/bin:/usr/contrib/bin:/home/xstra/bin:.
xstra>
```

Pour affecter une valeur à une variable on utilise simplement le signe égalité (=) sans espace de part et d'autre du signe. La syntaxe de l'affectation est :

nom_variable valeur

La commande *declare* permet également de créer une variable, de lui assigner une valeur et d'en spécifier certains attributs tels que nombre entier, tableau, lecture seule, etc (voir chapitre 8.2.2).

Pour déclarer une variable interne en **variable d'environnement**, il faut utiliser la commande *export nom variable* ou *declare x nom variable*.

Exemple

```
xstra> IMPRIMANTE=laser
xstra> export IMPRIMANTE
ou
xstra> export IMPRIMANTE=laser
```

La valeur (ou référence) d'une variable est obtenue en faisant précéder le nom de la variable du signe *$*. Ainsi, la commande *echo $nom variable* affichera le contenu de la variable.

Exemple

```
xstra> VAR1=florent
xstra> echo $VAR1
florent
xstra>
```

Les commandes *env* et *printenv* permettent de visualiser la liste et le contenu des variables d'environnement (*set* nous donne la liste de toutes les variables et leur valeur).

Exemple

```
xstra> IMPRIMANTE=laser
xstra> export IMPRIMANTE
xstra> set
...
```

```
IMPRIMANTE laser
...
xstra>
```

La commande *unset nom variable* permet de supprimer une variable. Attention : cette commande peut supprimer les variables PATH, PS1, PS2 et modifier fortement le fonctionnement de votre shell.

Certaines variables ont une signification particulière pour l'interpréteur de commandes Bash. Les principales sont indiquées ci-dessous par ordre alphabétique.

CDPATH Liste des chemins d'accès utilisés par la commande *cd* lors d'un changement de répertoire en relatif. Il est ainsi possible, en mode relatif, d'accéder à un répertoire dont la racine est dans la variable CDPATH. Un chemin d'accès défini par la chaîne de caractères vide ou par le caractère **.** (point) correspond au répertoire courant.

Exemple

```
xstra> CDPATH="":/home/xstra
xstra> pwd
/home/xstra/projet1
xstra> cd projet2
xstra> pwd
/home/xstra/projet2
xstra>
```

ENV Le contenu de cette variable est utilisé par le Bash comme fichier d'initialisation lorsqu'il est invoqué par un shell script.

HISTSIZE Nombre de lignes de l'historique (par défaut : 500).

HISTFILE Nom du fichier mémorisant l'historique (par défaut *: $HOME/.bash history*) (voir paragraphe 6.5.1 pour le mécanisme d'historique).

HOME Indique le répertoire d'accueil ; correspond à l'argument par défaut de la commande *cd*.

Exemple

```
xstra> HOME=/home/xstra
xstra> pwd
/home/xstra/projet1/prog
xstra> cd
xstra> pwd
/home/xstra
xstra>
```

HOSTNAME Indique le nom de l'ordinateur.

IFS Définit les caractères reconnus comme délimiteurs de champs internes. Les valeurs par défaut sont : « espace », « tabulation », « nouvelle ligne ».

OLDPWD Nom du répertoire précédent, positionné par la commande *cd*.

PATH Liste des chemins d'accès contenant les commandes. Chaque chemin d'accès est séparé par le caractère *:*. L'ordre dans la liste correspond à l'ordre de recherche des fichiers exécutables. Ainsi une valeur du *PATH=/bin:/etc:.* indique une recherche de la commande dans le répertoire */bin*, puis */etc* et enfin dans le répertoire courant (symbolisé par un point).

Exemple

```
xstra> echo $PATH
/bin:/usr/bin:/usr/local/bin:/usr/gnu/bin:.
xstra>
```

Attention

Quel que soit le shell utilisé, le répertoire courant (**.**) n'est examiné que s'il apparaît dans la variable PATH.

PS1 Valeur du premier prompt (par défaut : bash-2.04**$**). L'affichage du prompt est dynamique, contrairement à d'autres shell, c'est-à-dire qu'il est possible d'y intégrer des variables. Il est aussi possible de le personnaliser en y insérant des caractères spéciaux qui sont interprété comme suit :

- *\h* le nom de l'ordinateur
- *\u* le nom du login
- *\t* l'heure courante
- *\w* le chemin du répertoire de travail
- *\W* le nom du répertoire de travail
- *\!* le numéro de la commande dans l'historique
- *\$* # pour l'administrateur sinon $.

Exemple

```
bash 2.04$ PS1='\u@\h:$PWD \$ '
xstra@courlis:/home/xstra $ pwd
/home/xstra
```

```
xstra@courlis:/home/xstra $ cd projet
xstra@courlis:/home/xstra/projet $
```

PS2 Valeur du deuxième prompt (par défaut >). Il est personnalisable comme le premier prompt.

PWD Nom du répertoire courant, positionné par la commande *cd*.

TERM Définition du type de terminal utilisé (les terminaux connus sur un système sont décrits dans le répertoire */usr/share/terminfo*) :

- *vt100* pour les terminaux compatibles avec le type vt100,
- *xterm* émulateur de terminal X,
- *sun* poste de travail Sun,
- *hp* poste de travail HP.

Exemple

```
xstra> export TERM=xterm
xstra> printenv TERM
xterm
xstra>
```

TMOUT Nombre de secondes d'inactivité provoquant un *logout*.

Les variables PWD, OLDPWD, HOSTNAME, sont affectées par le Bash, les autres sont utilisées par le Bash et affectées par l'administrateur de l'ordinateur et par l'utilisateur.

6.3 LES ALIAS

La commande *alias nom alias='commandes'* permet de donner une équivalence entre une ligne de commandes Linux et une chaîne de caractères. Les alias servent surtout à définir des abréviations ou de nouveaux noms pour des commandes.

Sans argument, la commande *alias* permet d'afficher la liste des alias définis.

Exemple

```
xstra> alias lsf='ls  FCa'
xstra> alias
lsf ls  FCa
...
xstra>
```

Il n'est pas possible de créer des alias avec passage de paramètres. De plus, si le Bash trouve un alias dans l'interprétation du nom de la commande, il remplace cet alias par sa définition et recommence le processus de recherche d'un alias. Il est ainsi possible d'inclure un alias dans la définition d'un autre alias. Nous recommandons, afin d'éviter des problèmes d'interprétations, de définir un alias sur une seule ligne de commandes, et de ne pas utiliser les alias dans une fonction shell.

La commande *unalias nom alias* permet de supprimer l'alias dont le nom est *nom alias*.

6.4 LA FONCTION SHELL

Il est possible de définir des fonctions shell interactivement ou dans un script. Une fonction shell évite la duplication de lignes de programmes identiques dans un script ou l'abréviation d'une commande avec passage de paramètres. Le passage de paramètres sera présenté au chapitre 8. La syntaxe de création d'une fonction (*nom fonction*) est :

```
function nom_fonction { liste de commandes }
```

ou

```
nom_fonction  ()  {liste de commandes;}
```

La fonction est exécutée dans le shell courant, aucun nouveau processus n'est créé pour interpréter les commandes de la fonction. Dans le fichier *.bash profile* (paragraphe 6.1) les fonctions *dir* et *f* ont été créées. On constate que *f file* sera désormais la commande qui permettra de rechercher le fichier *file* à partir du répertoire courant. L'exemple suivant montre l'intérêt d'une fonction : elle est interprétée par le shell en cours.

Exemple

La commande *cdd rep*, en utilisant la fonction *cdd*, cherche dans l'arborescence, à partir du répertoire courant, le répertoire *rep* et positionne l'utilisateur dans ce répertoire. Cette fonctionnalité n'est possible que si *cdd* est une fonction et non un script (voir le chapitre 8).

```
xstra> cdd () {
> tempo=$(find . -name $1 -type d -print 2>/dev/null)
> cd $(echo $tempo | cut -f1 -d' ')
> }
ou
xstra> cdd () {
> tempo=$(find . -name $1 -type d -print 2>/dev/null)
> cd $(echo $tempo | cut -f1 -d' ') ;}
xstra> cdd essai
xstra/bin/essai>
```

La fonction peut être récursive, c'est-à-dire qu'elle peut s'appeler elle-même.

Lorsque l'on quitte la session, la fonction est détruite car elle est liée à un shell. Si l'on souhaite qu'elle soit conservée, il faut l'introduire dans le fichier d'initialisation *.bashrc*. On peut également créer des fonctions dans les scripts. Dans ce cas la fonction est liée au shell qui exécutera le script.

6.5 L'ÉDITION DE COMMANDE EN LIGNE

L'un des grands intérêts du shell Bash réside dans l'interface d'édition de la ligne de commandes en mode interactif. En utilisant certaines touches spécialisées du clavier (flèche de direction, touche de suppression, insertion…) ou les touches de contrôle et échappement, le Bash permet de chercher une ligne de commandes dans l'historique, puis de l'éditer. Cette fonctionnalité d'édition d'une ligne de commande utilise certaines commandes de l'éditeur *emacs*. La combinaison *<ctrl p>* indique qu'il faut appuyer simultanément sur les touches « Ctrl » et « p ». La combinaison *<meta f>* indique qu'il faut utiliser la touche « meta » et « f ». Sur un clavier standard la touche « meta » correspond à la touche « Alt ». Si cette touche n'existe pas sur votre clavier, utilisez la touche « Esc » puis « f ».

6.5.1 Mécanisme d'historique

Lorsque le Bash est utilisé en interactif, un mécanisme d'historique mémorise les dernières commandes lancées par l'utilisateur à partir d'un terminal. Le nombre de lignes gérées par le mécanisme d'historique est défini par la variable HISTSIZE (500 par défaut). Il est ainsi possible de rappeler ces commandes, de les modifier et de les rejouer.

La commande *history n* donne la liste numérotée des *n* dernières commandes. Il faut pour cela avoir positionné la commande *set o history* dans le fichier *. bash profile*. Lorsque l'on quitte la session, l'historique est sauvegardé dans un fichier dont le nom est défini dans la variable HISTFILE *($HOME/.bash history* par défaut).

Exemple

```
xstra> history 8
5  ls
6  ls  l toto
7  ps  ax
8  help cat
9  cc  o essai essai.c
10 cd /home/xstra1
11 ls fich1 fich2
12 man cat
xstra>
```

L'intérêt de ce mécanisme d'historique est de permettre de rechercher puis modifier une ancienne ligne de commandes. Il existe deux modes de recherche : le mode

incrémental et le mode non-incrémental. Dans le mode incrémental, la ligne de commandes recherchée est affichée au fur et à mesure de la saisie du ou des caractères de recherche. Dans le mode non-incrémental, la ligne de commandes recherchée est affichée après la saisie de la chaîne de caractères de recherche.

a) Le mécanisme de recherche de commande dans l'historique

Un certain nombre de combinaisons de touches permettent de rappeler des commandes mémorisées. Ces combinaisons sont les suivantes :

↑ *ou <ctrl P>*	cherche la commande précédente dans l'historique.
↓ *ou <ctrl N>*	cherche la commande suivante dans l'historique.
<ctrl R>motif	recherche incrémentale en arrière, à partir de la ligne courante de l'historique, de la dernière commande contenant le motif. Cette recherche est incrémentale. Il est possible, après la saisie de motif, de refaire un <ctrl-r> qui continuera la recherche en arrière dans l'historique avec le même motif. La ligne de commandes trouvée pourra être modifiée par l'éditeur de lignes.
<meta p>motif<return>	recherche non-incrémentale en arrière, à partir de la ligne courante de l'historique, la dernière commande contenant le motif. La ligne de commandes de l'historique contenant motif sera affichée après le <return>.

Exemple

```
xstra> ls  CF
info.txt  rpm/  tar/
xstra> mkdir logiciel
xstra> ls
info.txt  logiciel  rpm  tar
xstra> <ctrl r>
(reverse i search)`': ls <ctrl r> <return>
  § le premier <ctrl r> suivi de ls recherche la première
  § commande contenant ls et le deuxième <ctrl r>
  § continue la recherche dans l'historique.
  § <return> exécutera cette commande.
info.txt  logiciel/  rpm/  tar/
xstra>
```

b) Le mécanisme de relance d'une commande compatible C-shell

!!	relance la dernière commande exécutée.

`!10`	relance la commande numéro 10 (`cd /home/xstra1`).
`!ls`	relance la dernière commande commençant par `ls` (commande numéro 11 dans l'exemple ci-dessus).
`! n`	relance la n^{ième} commande avant la dernière, sachant que la dernière est celle que l'on saisit.
`!?toto?`	Renouvelle la dernière commande qui contient la chaîne de caractères `toto` (commande numéro 6 dans l'exemple ci-dessus).

Exemples

Cet exemple présente quelques commandes mémorisées

```
xstra> alias p="fc  s | less"
```

`p` permet de relancer la dernière commande suivie de `| less`. La signification de `| less` est détaillée au paragraphe 7.3.

```
xstra> ls  l /etc
    § 350 lignes défilent à toute vitesse.
xstra> p
```

p exécute la commande `ls  l /etc` associé avec `|  less` (affichage page par page)

6.5.2 L'édition de commande en ligne

L'édition d'une ligne de commandes est possible pendant la saisie de la commande ou après une recherche d'une ligne de commandes dans l'historique.

Utilisation de base

Lors de la saisie d'une commande, il est possible :

- d'effacer le dernier caractère tapé par la touche <backspace>,
- d'effacer la ligne entière par < ctrl-u >.

À tout moment, vous pouvez déplacer le curseur d'un caractère vers la gauche avec la flèche gauche (ou `<ctrl b>`) ou vers la droite avec la flèche droite (ou `<ctrl f>`). La position du curseur indique l'emplacement d'insertion du caractère saisi au clavier. Vous pouvez ainsi insérer des caractères à n'importe quelle position dans votre ligne de commandes courante. Les caractères à droite de votre curseur seront déplacés d'un pas pour inclure le caractère saisi lors de chaque insertion. Après chaque caractère saisi, le curseur se déplace d'une position vers la droite. La suppression du caractère saisi est réalisée par la touche `<backspace>`, la touche `<suppr>` ou `<ctrl d>` supprimant le caractère sur la position du curseur. De même, la suppression d'un caractère décalera le texte à droite du curseur d'un pas vers la gauche pour remplacer le caractère supprimé.

Déplacement du curseur

En plus des flèches droite et gauche qui permettent de déplacer le curseur d'une position dans votre ligne de commandes, il existe des commandes qui permettent un déplacement plus rapide :

`<ctrl A>` déplacement du curseur au début de la ligne de commandes.

`<ctrl E>` déplacement du curseur à la fin de la ligne de commandes.

`<meta F>` déplacement du curseur d'un mot vers la droite.

`<meta B>` déplacement du curseur d'un mot vers la gauche.

Fonction couper-coller

Les touches permettant de supprimer une partie de votre ligne de commandes sont :

`<ctrl K>` suppression du texte de la position du curseur jusqu'à la fin de ligne.

`<ctrl U>` suppression du texte de la position du curseur jusqu'au début de la ligne.

`<meta D>` suppression du texte de la position du curseur jusqu'à la fin du mot.

Chaque partie de la ligne de commandes supprimée est stockée dans une mémoire tampon et pourra être réinsérée à l'aide des touches :

`<ctrl Y>` Insertion du dernier texte supprimé.

`<meta Y>` Décalage d'un pas des textes supprimés et insertion du texte courant. Cette commande n'est utilisable qu'après la commande `<ctrl Y>` ou `<meta Y>`.

6.6 UTILITAIRES DU BASH

Il existe dans le Bash quelques fonctionnalités de complètement et substitution utiles décrites ci-dessous.

6.6.1 Recherche et complètement d'une commande

Il est possible de compléter le nom d'une commande en cours de saisie en utilisant la touche `<tab>`. Le contenu de la variable PATH, la liste des alias et fonctions sont utilisés pour la recherche des commandes. Un fichier en mode de protection exécution est considéré comme une commande. En cas d'ambiguïté, c'est-à-dire si plusieurs commandes commencent par le même préfixe, le terminal émettra un bip. Dans le cas contraire le Bash complétera le nom de la commande soit totalement, soit partiellement et émettra un bip indiquant une ambiguïté.

Exemple

```
xstra> his<tab>
```

sera complétée et deviendra

```
xstra> history
```

Il est aussi possible de connaître toutes les commandes commençant par le préfixe en réutilisant la touche *<tab>* après le préfixe.

Exemple

```
xstra> ls<tab><tab>
ls lsacl lsf lsr lssx § Liste des commandes commençant par ls
xstra> ls             § à ce stade, l'utilisateur a la main
                      § pour compléter la commande.
```

6.6.2 Recherche et complètement des noms de fichiers

Cette fonctionnalité est similaire à celle concernant les noms de commandes. Elle considère les noms de fichiers comme arguments des commandes. Il est possible, en saisissant le nom d'un fichier comme argument, de le compléter à l'aide de la touche *<tab>*.

Exemple

```
xstra> cd /home/xstra2/dev<tab>
```

sera complétée et deviendra

```
xstra> cd /home/xstra2/develop
```

Il faut taper *<return>* pour valider cette commande

Il est également possible, en réutilisant la touche *<tab>* , de connaître les noms de fichiers ayant le même préfixe.

Exemple

```
xstra> cd /home/xstra/pro<tab><tab>
projet1 projet2 projet3   § Liste des noms de fichiers
                          § commencant par pro
xstra> cd /home/xstra/pro § L'utilisateur, à ce stade, a la
                          § main pour choisir son répertoire
```

6.6.3 Substitution du caractère *tilde* : ~

- Le caractère ~ seul ou devant le caractère / est remplacé par le contenu de la variable HOME.

- Le caractère ~ suivi d'une chaîne de caractères puis d'un / considérera la chaîne de caractères comme étant le nom d'un utilisateur défini dans le fichier */etc/passwd*. Cet ensemble sera alors remplacé par le chemin d'accès du répertoire d'accueil de l'utilisateur.
- Le caractère ~ suivi du caractère *+* est remplacé par le contenu de la variable PWD.
- Le caractère ~ suivi du caractère - est remplacé par le contenu de la variable OLDPWD.

Sortie de session automatique

La variable *TMOUT* permet d'initialiser une sortie automatique de session si aucun caractère n'est saisi pendant le temps déterminé par la valeur de la variable *TMOUT* en secondes.

6.7 QUELQUES COMMANDES INTERNES AU BASH

Il existe des commandes internes au Bash, qui ne créent donc pas des processus shell fils. Les principales commandes internes sont les suivantes (par ordre alphabétique) :

alias nom alias='commandes' et *unalias nom*

Ces deux commandes ont été présentées au paragraphe 6.3. Sans paramètre, la commande *alias* liste les alias.

cd [rep]

Cette commande, déjà présentée au paragraphe 3.4, permet de changer de répertoire. La valeur par défaut de *[rep]* est le contenu de la variable HOME. Le chemin d'accès de *rep* est défini par la variable CDPATH. Si *rep* commence par le caractère **/**, **.** ou **..** la variable CDPATH n'est pas utilisée. Si *arg* est le caractère alors le changement de répertoire est réalisé avec le contenu de la variable OLDPWD qui correspond au dernier répertoire.

command [v] cmd [args…]

La commande *command* exécute la commande *cmd* avec ses arguments *[args…]* si cette dernière est une commande interne ou une commande dans un des répertoire du PATH. L'option *–v* permet d'obtenir une description de la commande *cmd*.

Exemple

```
xstra> type ls
ls is /usr/bin/ls
xstra> alias type=more
xstra> type ls
ls: No such file or directory
```

```
xstra> command type ls
ls is /usr/bin/ls
xstra>
```

echo [ne] "texte"

Elle permet l'affichage sur l'écran de *texte*. Associé à l'option *–n*, le *texte* est affiché sans retour à la ligne. Associé à l'option *–e*, le *texte* peut contenir des séquences particulières comme suit :

- *\a* alerte (le bip),
- *\b* retour en arrière,
- *\c* suppression nouvelle ligne,
- *\e* le caractère *<esc>*,
- *\n* nouvelle ligne,
- *\r* retour chariot,
- *\t* tabulation horizontale,
- *\v* tabulation verticale,
- ** le caractère \,
- *\nnn* le caractère ASCII dont la valeur octale est *nnn*.

enable [n] cmd

Cette commande permet de déterminer si la commande *cmd* est une commande interne ou un programme stocké dans un répertoire défini dans la variable PATH s'il a le même nom que la commande interne. Si l'option *–n* est utilisée, *cmd* sera un programme, sans cette option, *cmd* sera la commande interne.

Exemple

Par exemple, pour utiliser la commande externe *test*, exécutez *enable –n test*.

```
xstra> type test
test is a shell builtin
xstra> enable -n test
xstra> type test
test is /usr/bin/test
xstra> enable test
xstra> type test
test is a shell builtin
xstra>
```

exec cmd

La commande *cmd* est exécutée à la place du shell sans qu'il y ait création d'un nouveau processus. Il n'est pas possible de retourner au shell.

Exemple

```
xstra> exec ls  l
drw          2 xstra staff 512 Apr 14 1999 projet1
...
...
login :
```

exit [n]

Cette commande provoque l'arrêt du shell avec un code de retour égal à la valeur *n* (0 par défaut).

export [n] [nom]

Cette commande permet de déclarer *nom* comme étant une variable d'environnement. *nom* pourra être de la forme *var=val*, la commande *export var=val* permettra de déclarer *var* comme variable d'environnement et de lui affecter la valeur *val*. Si l'option *–n* est utilisée, la variable d'environnement *nom* deviendra une variable interne. La commande *export* sans argument affiche les variables d'environnement.

hash [r] [nom]

Lors de l'exécution d'une commande externe ou d'un script, le chemin d'accès de la commande est gardé en mémoire dans une table par le shell. Il est possible, à l'aide de la commande *hash,* d'ajouter la commande définie par *nom* dans la table. L'exécution de la commande *nom* sera plus rapide. En effet, cela ne nécessitera pas la recherche du chemin d'accès.

L'option *r* permet de vider la table. La commande *hash* sans argument affiche le contenu de la table.

Exemple

```
xstra> hash
4    /usr/bin/grep
2    /opt/local/bin/nedit
xstra> hash find
xstra> hash
4    /usr/bin/grep
2    /opt/local/bin/nedit
0    /usr/bin/find
xstra>
```

history

Cette commande donne la liste des commandes de l'historique (voir le paragraphe 6.5.1).

pwd

Cette commande permet d'afficher le nom du répertoire de travail.

read [nom]

Cette commande permet de lire le clavier du poste de travail. En paramètre de cette commande, on peut trouver plusieurs variables qui doivent être séparées par un ou plusieurs espaces ou tabulations.

set et unset [option] [/+o options] [arg]

La commande *set* sans option permet l'affichage de toutes les variables avec leurs valeurs. La commande *unset* permet de supprimer une variable.

shopt [su] [opt]

La commande *shopt* permet d'activer et désactiver les variables contrôlant le comportement de l'interpréteur de commande. Sans argument la commande *shopt* permet l'affichage de toutes les options avec leurs valeurs. L'option *[s]* (s pour set) positionne l'option *opt* et l'option *[u]* (u pour unset) désactive l'option *opt*. Une liste non exhaustive des options de *shopt* est :

cdable vars : si l'argument de la commande *cd* n'est pas un répertoire, il sera considéré comme une variable contenant le répertoire.

cmdhist : considère une commande sur plusieurs lignes comme une seule dans l'historique.

extglob : les extensions Bash dans la génération de noms de fichiers, décrites au paragraphe 8.2.4.d, sont activées.

times

Cette commande affiche les temps système et utilisateur cumulés pour les processus lancés par le shell.

trap [arg] [sig]

Cette commande permet d'associer à un signal l'exécution de la commande *arg*. La commande *trap* sans argument permet d'afficher la liste des commandes associées à un signal. L'exemple le plus courant est l'association du signal 0, c'est-à-dire la fin d'une fonction ou d'un shell à l'exécution d'une commande.

type [nom]

Cette commande indique le mode d'interprétation par le shell pour chaque *nom* (commande interne, commande dans la table hash, chemin d'accès de la commande).

Exemples

```
xstra> type test
test is a shell builtin
xstra> type ls
ls is /usr/bin/ls
xstra>
```

ulimit [acftu] [limit]

permet de contrôler certaines ressources disponibles du shell ou celles d'un processus exécuté par ce shell. L'option *-a* permet d'afficher la liste des ressources. Les autres options sont :

- *c* la taille maximum des fichiers *core*,
- *f* la taille maximum des fichiers créés par le shell,
- *t* la quotité CPU maximum en secondes,
- *u* le nombre maximum de processus par utilisateur.

Exemples

Pas de création de fichier *core* lors d'une fin anormale d'un programme.

```
xstra> ulimit -c 0
xstra>
```

6.8 EXÉCUTION D'UN SCRIPT

L'exécution d'un script, ou fichier de commandes (une suite de commandes), est possible de trois manières différentes :

bash option pgme arg1 arg2 arg3

Cette forme permet d'exécuter le fichier ordinaire *pgme* sans autorisation d'exécution. L'option *x* est utilisée pour la mise au point car elle permet de suivre l'exécution des commandes contenues dans le script.

pgme arg1 arg2 arg3

Cette forme est la plus simple et la plus fréquente. Après avoir cherché *pgme* en utilisant les chemins indiqués par la variable PATH, le shell vérifie que *pgme* a l'autorisation d'exécution. Un processus shell fils est créé pour exécuter *pgme*. Ce processus se terminera lorsque le script sera terminé.

Le shell fils créé pour l'exécution du script peut être soit un Bash soit un autre type de shell.

On a un autre type de shell si la première ligne du script débute par *#!* et est suivie par le chemin absolu du shell.

Exemple

```
#!/bin/tcsh # pour le lancement d'un TC shell
# Dans tous les autres cas,
# le shell fils est un Bash.
```

. *pgme* (nom du script précédé du caractère **.** et d'un espace)

Cette forme permet d'exécuter un script sans autorisation d'exécution et sans création d'un processus shell fils, contrairement aux deux formes précédentes. Elle permet ainsi d'initialiser les variables du shell. Le fichier *pgme* doit être obligatoirement dans un répertoire déclaré par la variable PATH.

Exemple

```
xstra> ls  CF
bin/ develop/ essai
xstra> more essai
ls
xstra> . essai
bin develop essai
xstra>
```

6.9 EXERCICES

Exercice 6.4.1

Écrivez votre propre commande "*dir*" affichant page par page les informations données par la commande *ls l*, incluant les fichiers cachés (sauf . et ..).

Exercice 6.4.2

Écrivez un alias imposant la confirmation sur la suppression des fichiers.

Écrivez la commande "*psmoi*" permettant d'obtenir la liste de tous les processus vous appartenant (utilisez la commande *ps*).

Exercice 6.4.3

Vous souhaitez modifier :

1) l'invite afin qu'elle indique le nom de l'ordinateur sur lequel vous travaillez et votre répertoire courant,

2) la valeur par défaut des protections des fichiers et répertoires que vous allez créer.

De plus vous souhaitez retrouver ces modifications lors de chacune de vos entrées en session.

Exercice 6.4.4

Écrivez votre propre fonction "*fd*" permettant de rechercher à partir du répertoire courant l'emplacement d'un répertoire. Testez votre fonction en recherchant tous les répertoires commençant par r.

Chapitre **7**

Commandes Linux

Les commandes Linux en Bourne-shell et Bash ont une syntaxe commune. Cependant, chaque interpréteur de commandes a ses particularités propres. Dans ce chapitre, nous présentons uniquement la commande propre à l'interpréteur de commandes Bash.

7.1 LA COMMANDE LINUX

7.1.1 Syntaxe générale des commandes Linux

```
commande [ ± option… ] [ paramètre… ]
```

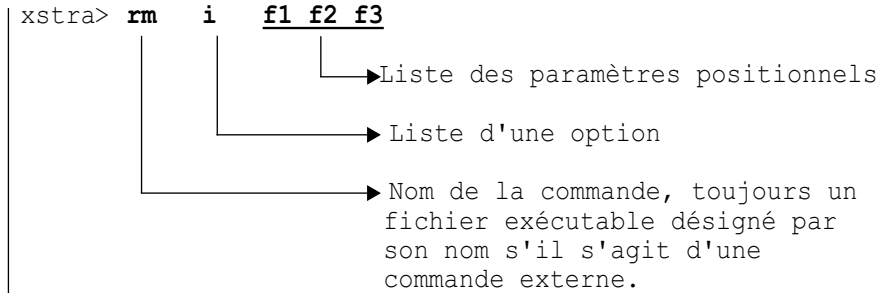
Une commande se compose d'un code mnémonique en minuscules suivi ou non d'arguments séparés par au moins un espace.

On distingue deux types d'arguments :

- les options, presque toujours précédées d'un signe ( ) et quelquefois d'un signe (+). Certaines options ont un paramètre propre,
- les arguments positionnels ou paramètres, qui sont en général des noms de fichiers ou de répertoires.

En général, la liste des options précède celle des paramètres.

Exemple



7.1.2 Conventions utilisées pour la syntaxe des commandes

1) Commande et options	Le texte de la commande ainsi que celui des options est à reprendre tel quel. Dans le manuel Linux, ce texte est représenté en caractères gras ou soulignés.
2) Paramètre	Un nom symbolique décrit l'usage de chaque paramètre. Ce nom symbolique doit être remplacé par la valeur effective que l'usager désire donner au paramètre.
3) […]	Le paramètre ou l'option entre crochets est optionnel.
4) Paramètre…	Les trois points qui suivent un nom symbolique d'un paramètre désignent une liste de paramètres.

Exemple

```
cat [ ] [ benstuv] fichier...
```

7.1.3 La ligne de commandes séquentielles

Il est possible de taper plusieurs commandes sur la même ligne en les séparant par des points-virgules `;`. Les commandes sont exécutées séquentiellement, de façon totalement indépendante, la première n'influençant pas la seconde et ainsi de suite.

Exemple

```
xstra> pwd ; who ; ls
/home/xstra
xstra   ttyp0   Mar    15 10:32
xstra1  co      Mar    15 09:12
bin projet1 projet2 essai
xstra>
```

Cette commande affiche le répertoire courant de l'utilisateur, donne la liste des utilisateurs connectés, puis liste les fichiers du répertoire courant.

7.1.4 La commande sur plus d'une ligne

Il est possible de taper une commande sur plusieurs lignes. Pour cela les lignes de commandes, sauf la dernière, doivent se terminer par la suite de touches *\<return>*.

Exemple

```
xstra> ls  l /home/xstra/develop\<return>
/essai.f <return>
 rwx r       1 xstra staff 258 Jan 15 16:42 essai.f
xstra>
```

Cette commande est équivalente à :

```
xstra> ls  l /home/xstra/develop/essai.f <return>
```

Elle liste les droits d'accès du fichier *essai.f* qui se trouve dans le répertoire */home/xstra/develop*.

7.1.5 Les séparateurs conditionnels de commandes

Il est possible de contrôler la séquence d'exécution de commandes en utilisant des séparateurs conditionnels.

Le séparateur **&&** permet d'exécuter la commande qui le suit si et seulement si la commande qui le précède a été exécutée sans erreur (code retour du processus nul).

Le séparateur **||** permet d'exécuter la commande qui le suit si et seulement si la commande qui le précède a été exécutée avec erreur (code retour du processus différent de 0).

Exemple

Suppression des fichiers si la commande *cd projet1* a été correctement exécutée.

```
xstra> cd projet1 && rm *
```

Exemple

Si le répertoire *projet1* n'existe pas, alors il sera créé par la commande *mkdir*.

```
xstra> cd projet1 || mkdir projet1
bash: cd: projet1: No such file or directory
xstra> ls
projet1
xstra>
```

7.2 LA REDIRECTION DES ENTRÉES-SORTIES

7.2.1 Le principe de redirection

On appelle processus, ou tâche, l'exécution d'un programme exécutable. Au lancement de chaque processus, l'interpréteur de commandes ouvre d'office une entrée standard (par défaut le clavier), une sortie standard (par défaut l'écran) et la sortie d'erreur standard (par défaut l'écran) (Fig. 7.1).

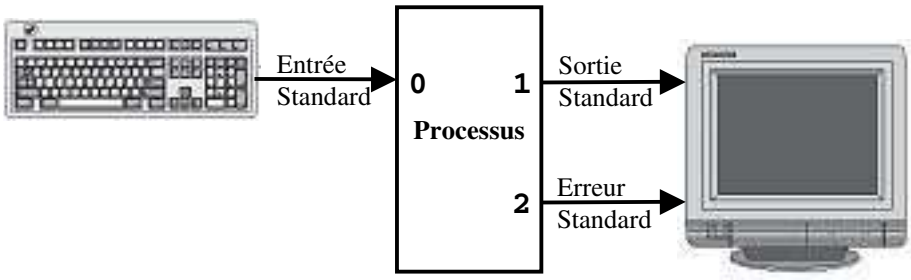


FIGURE 7.1. ENTRÉES/SORTIES STANDARD D'UN PROCESSUS.

Ces entrées-sorties standard peuvent être redirigées vers un fichier, un tube, un périphérique. La redirection de la sortie standard consiste à renvoyer le texte qui apparaît à l'écran vers un fichier (Fig. 7.2). Aucune information n'apparaîtra à l'écran, hormis celles qui transitent par la sortie d'erreur standard.

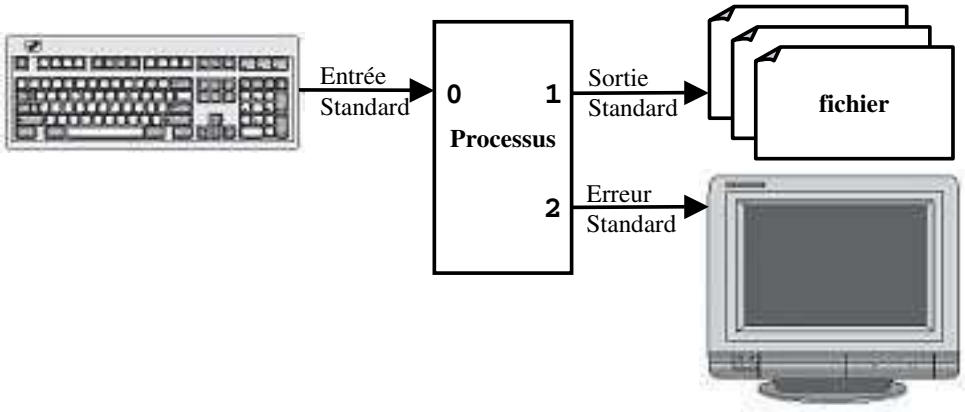


FIGURE 7.2. REDIRECTION VERS LE FICHIER *FS* DE LA SORTIE STANDARD D'UN PROCESSUS.

Il est naturellement possible de rediriger toutes les entrées-sorties standard d'un processus. Par conséquent, le processus recherchera les informations dont il a besoin dans un fichier et non plus au clavier. Il écrira dans des fichiers ce qui devait apparaître à l'écran (Fig. 7.3).

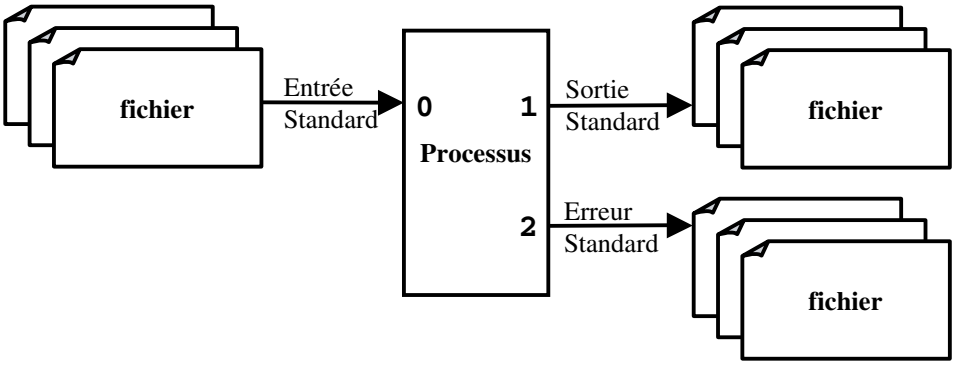


FIGURE 7.3. REDIRECTION VERS DES FICHIERS DE TOUTES LES ENTRÉES/SORTIES STANDARD D'UN PROCESSUS.

La redirection des sorties peut être réalisée par effacement et création du fichier ou par ajout à la fin du fichier si ce dernier existe. Dans le cas contraire, un nouveau fichier sera créé. Dans le cas de la redirection de l'entrée, il est évident que le fichier doit exister. Le tableau 7.1 résume les différentes redirections.

	Le fichier existe-t-il?	
	OUI	NON
Redirection de l'entrée standard	lit le fichier	erreur
Redirection de la sortie standard et de la sortie erreur standard	effacement et création du fichier	création du fichier
Concaténation de redirection de la sortie et de l'erreur standard	ajoute à la fin du fichier	création du fichier

TABLEAU 7.1. RÉSUMÉ DES REDIRECTIONS.

Le caractère **<** suivi du nom d'un fichier indique la redirection de l'entrée standard à partir de ce fichier :

`<fe` Définition de `fe` comme fichier d'entrée standard.

Le caractère **>** suivi du nom d'un fichier indique la redirection de la sortie standard vers ce fichier :

`>fs` — Redirection de la sortie standard vers le fichier `fs` ; le fichier `fs` est créé ou écrasé s'il existe déjà et si la variable booléenne `noclobber` n'est pas initialisée.

Si on double le caractère >, l'information ou la redirection sera ajoutée au fichier `fs`.

`>>fs` — La sortie standard rallonge le fichier `fs`.

Les caractères **2>** suivis du nom d'un fichier indiquent la redirection de la sortie d'erreur standard vers ce fichier :

`2>erfs` — Redirection de la sortie d'erreur standard vers le fichier `erfs`.

Les caractères **2>>** suivis du nom d'un fichier indiquent que la redirection de la sortie d'erreur standard sera ajoutée au fichier :

`2>>erfs` — La sortie d'erreur standard est ajoutée au fichier `erfs`.

Si l'on souhaite rediriger la sortie standard et la sortie d'erreur standard, la syntaxe sera :

`commande >fs 2>erfs` — Dirige la sortie standard vers `fs` et la sortie d'erreur vers `erfs`.

Les caractères **&>** suivis du nom d'un fichier indiquent la redirection de la sortie standard et de la sortie d'erreur standard vers ce fichier :

`&>erfs` — Redirection des sorties standard et de la sortie d'erreur standard vers le fichier `erfs`.

Exemples

```
xstra> ls >temp
    § Range, dans le fichier temp, la liste
    § des fichiers du répertoire courant.
    § Le fichier temp est créé s'il n'existe
    § pas ou écrasé s'il existe.
xstra> ls >>f.rep
    § Ajoute à f.rep la liste des fichiers
    § du répertoire courant.
xstra> ls /toto 2>temp
    § Range, dans le fichier temp,
    § les messages d'erreurs et affiche
    § à l'écran la liste des fichiers.
    § /toto  n'existe pas.
xstra> ls /toto * >temp 2>ertemp
```

```
§ Range dans le fichier temp, la liste
§ des noms de fichiers du répertoire courant
§ et dans le fichier ertemp,
§ le message d'erreur /toto n'existe pas.
```

7.2.2 La commande cat et les redirections

La commande *cat* est une commande multi-usage qui permet d'afficher, de créer, de copier et de concaténer des fichiers. Elle utilise pleinement les mécanismes de redirection. Elle lit l'entrée standard si aucun fichier n'est spécifié. Ce qui est lu est affiché sur la sortie standard.

a) Lecture sur clavier et écriture sur écran

```
xstra> cat
Toute la musique que j'aime,
Toute la musique que j'aime,
<ctrl d>
xstra>
```

Le texte « Toute la musique que j'aime, » est lu du clavier et est affiché à l'écran. La combinaison de touches *<ctrl d>* interrompt la saisie.

b) Copie d'un fichier

```
xstra> cat f1 >f2  § première possibilité
xstra> cat <f1 >f2 § deuxième possibilité
xstra>
```

Le fichier *f1* est copié dans *f2*.

c) Concaténation des fichiers

```
xstra> cat f1 f2 f3 >f123
xstra>
```

Le fichier *f123* contiendra la concaténation des fichiers *f1*, *f2* et *f3*, dans cet ordre.

d) Ajout d'un fichier

```
xstra> cat f1 >>f2
xstra>
```

Le fichier *f1* est concaténé à la suite du fichier *f2*. *f2* est créé s'il n'existe pas.

e) Création d'un fichier par saisie au clavier

```
xstra> cat >f1
bonjour
```

```
.....
<ctrl d>
xstra>
```

Le fichier *f1* est créé. Il contiendra le texte saisi jusqu'à interruption par la combinaison de touches *<ctrl d>*.

f) Création d'un fichier avec condition de saisie

```
xstra> cat <<EOT >f1
Merci de votre visite
A bientot
EOT
xstra>
```

Le fichier *f1* est créé par saisie de texte au clavier, jusqu'à la saisie en début de ligne d'une chaîne de caractères prédéfinie (*EOT* dans ce cas). La chaîne de caractères *<<texte* redirige l'entrée standard jusqu'à apparition du mot *texte*. Cette formulation est très employée pour la création d'un fichier dans un fichier de commandes (script).

7.3 LES TUBES DE COMMUNICATION (PIPE) ET LES FILTRES

7.3.1 Les tubes

Un **tube** (**pipe** en anglais) est un flot de données qui permet de relier la sortie standard d'une commande à l'entrée standard d'une autre commande sans passer par un fichier temporaire (Fig. 7.4).

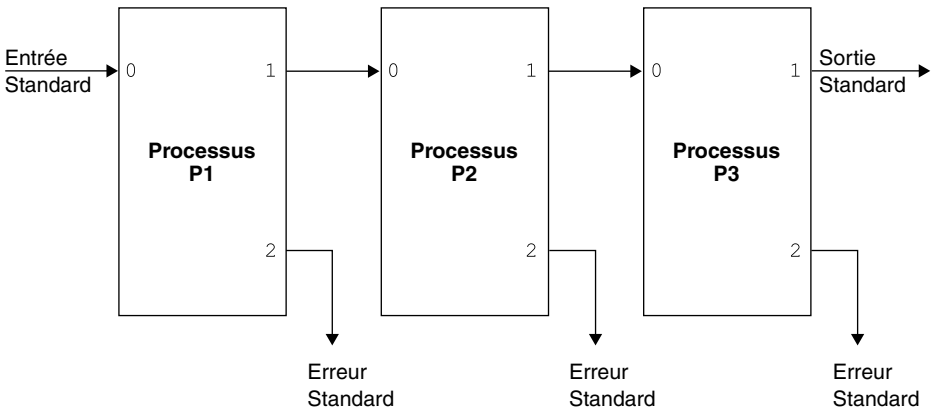


FIGURE 7.4. UN TUBE (PIPE).

Dans une ligne de commandes, le tube est formalisé par la barre verticale |, que l'on place entre deux commandes :

```
P1 | P2 | P3
```

Exemple 1

Affichage page par page du contenu du répertoire courant :

```
xstra> ls  l | less
```

Dans cet exemple, le résultat de la commande `ls  l` n'apparaît pas à l'écran : la sortie standard est redirigée vers l'entrée standard de la commande `less` qui, quant à elle, affichera son entrée standard page par page. La commande `less`, contrairement à la commande `more`, permet à l'aide des touches claviers ↓, ↑, <page-up> et <page-down> de monter et descendre dans le flot de données obtenu dans un tube.

Exemple 2

Affichage page par page du contenu du répertoire courant dont les protections sont `rwxr xr x :`

```
xstra> ls  l | grep "rwxr xr x"| less
```

Pour obtenir le même résultat en utilisant le mécanisme de redirection des entrées-sorties, il faut écrire :

```
xstra> ls  l >temp1; \<return>
grep "rwxr xr x" <temp1 >temp2 \<return>
; less temp2; rm temp1 temp2
```

Cette solution est non seulement plus lourde mais aussi plus lente. Les tubes ont deux qualités supplémentaires :

- Toutes les commandes liées par le tube s'exécutent en parallèle. C'est le système qui réalise la synchronisation entre les processus émetteurs et récepteurs,
- Il n'y a pas de limite de taille pour le flot de données qui transite dans le tube (il n'y a pas création de fichier temporaire).

Exemple 3

```
xstra> who | wc  l
```

Indique le nombre de personnes connectées au système.

```
xstra> ls | wc  w
```

Indique le nombre de fichiers dans le répertoire courant.

7.3.2 Les filtres

Dans les exemples précédents, nous voyons apparaître, à travers les commandes *wc*, *less* et *grep*, une famille particulièrement importante de commandes Linux : les **filtres.** Un filtre est une commande qui lit les données sur l'entrée standard, les traite et les écrit sur la sortie standard.

Le concept de tube, avec sa simplicité, devient un outil très puissant dans Linux qui propose un choix très vaste de filtres. Les filtres les plus utilisés sont les suivants :

grep recherche les occurrences d'une chaîne de caractères.

egrep une extension de *grep* (voir chapitre 14).

wc compte le nombre de caractères (ou octets), mots et lignes.

less affiche son entrée standard page par page.

dd filtre de conversion.

sed éditeur de flot : il applique des commandes de l'éditeur *ed* sur l'entrée standard et envoie le résultat sur la sortie standard (voir chapitre 14).

awk petit langage de manipulation de texte (voir chapitre 14).

sort filtre de tri.

Attention

Le caractère < n'est pas obligatoire pour rediriger l'entrée standard d'un filtre sur un fichier :

```
less /etc/passwd    équivaut à less </etc/passwd
cat  /etc/group     équivaut à cat  </etc/group
```

La puissance des filtres et des tubes est très bien explicitée dans l'exemple suivant :

Compter le nombre d'utilisateurs de la machine dont le login shell est le Bash.

```
xstra> cat /etc/passwd | grep /bin/bash | wc -l
```

La commande *cat* liste le contenu du fichier */etc/passwd* décrivant tous les utilisateurs. Le filtre *grep* ne conserve que les lignes contenant la chaîne */bin/bash*. Le filtre *wc -l* compte le nombre de lignes passées par le filtre *grep*.

7.3.3 La commande xargs

Le mécanisme de tube est très pratique pour assembler entre elles des commandes Linux. Cependant, beaucoup de commandes ne lisent pas dans leur entrée standard : *ls, rm, cp, ln, mv* et bien d'autres ne sont pas des filtres, mais traitent leurs arguments :

rm alpha beta gamma est une commande correcte, alors que

ls | rm n'a AUCUN sens.

Ces commandes ne peuvent donc pas apparaître dans un tube, sauf au début. La commande *xargs* permet de lever cette restriction en construisant une liste de paramètres à partir de l'entrée standard :

cmd1 | xargs cmd2 signifie :

lancer *cmd2* en lui passant en paramètres ce qui arrive dans l'entrée standard de *xargs*, donc la sortie standard de *cmd1*.

Exemple

Le fichier *listf* contient une liste de noms de fichiers, à raison d'un nom par ligne.

```
xstra> cat listf
/etc/passwd
/etc/group
/bin/bash
/usr/bin/id
xstra> cat listf | ls -i  § Erreur grossière
298014 bin      35364 Desktop   723098 Mail
608698 rep1    494443 listf
§ ls  i n'a rien lu dans son entrée standard
xstra> cat listf | xargs ls -i  § Correct
523314 /bin/bash     232382 /etc/group
231586 /etc/passwd   327284 /usr/bin/id
```

7.4 TÂCHES EN ARRIÈRE-PLAN

Linux attend la fin de l'exécution de la commande en cours d'exécution avant de permettre à l'utilisateur de relancer une nouvelle commande. Lorsqu'une commande ne nécessite pas de dialogue avec l'utilisateur et que sa durée d'exécution est importante, il est possible de l'exécuter en arrière-plan en ajoutant le caractère spécial **&** à la fin de la commande. Le système Linux lance alors la commande et redonne immédiatement la main à l'utilisateur pour d'autres travaux.

Le lancement de commandes en arrière-plan permet à un seul utilisateur de lancer plusieurs tâches à partir d'un même terminal. Cela correspond à l'aspect multitâche du système d'exploitation Linux.

Ces processus en arrière-plan ne peuvent plus être interrompus directement à partir de la console à l'aide de la touche *<ctrl c>*. Pour les interrompre il faut :

- soit sortir de sa session, dans ce cas tous les processus attachés à la session seront interrompus,
- soit rechercher le numéro du processus et lui envoyer un signal à l'aide de la commande *kill*, (voir aussi paragraphe 12.5).

Il est possible d'éviter qu'un processus en arrière-plan soit interrompu en quittant sa session. Il faut pour cela utiliser la commande *nohup* :

```
nohup commande <entrée >sortie &
```

Ce mécanisme est intéressant si l'on souhaite exécuter un programme de très longue durée et libérer la console de lancement.

Action	État initial du processus	État final du processus
| xstra> **commande**		processus en avant-plan
| xstra> **commande &**		processus en arrière-plan
| xstra> **fg %numéro_job**	processus en arrière-plan	processus en avant-plan
| **<ctrl z>**	processus en avant-plan	processus stoppé (suspendu)
| xstra> **fg %numéro_job**	processus stoppé	processus en avant-plan
| xstra> **bg %numéro_job**	processus stoppé	processus en arrière-plan
• **kill –STOP %numéro_job** • soit tentative lecture sur le terminal • soit tentative d'écriture sur le terminal (il faut avoir positionné stty tostop)	processus en arrière-plan	processus stoppé

Tableau 7.2. Récapitulation de la gestion des processus en arrière-plan.

Lors du lancement d'une commande en arrière-plan, l'interpréteur de commandes Bash attribue un numéro croissant pour chacune des commandes lancées : le *numéro job*. La liste des travaux en arrière-plan, avec leur numéro, est obtenue à l'aide de la commande *jobs*. Cette commande indique si le processus en arrière-plan est en cours d'exécution (*running*) ou suspendu (*stopped*).

Il est possible de ramener un processus en avant-plan, en tapant la commande *fg %numéro job*. Un processus en avant-plan peut être suspendu par l'envoi de l'ordre de suspension, si le terminal le permet (*stty isig*). Il est possible de déterminer la touche qui envoie l'ordre (*stty susp <ctrl z>*).

Un processus suspendu peut être réactivé à l'aide de la commande *bg %numéro job*. Un processus peut être supprimé à l'aide de la fonction *kill %numéro job*. Le paragraphe 12.5 présente ces mécanismes.

Le tableau 7.2 résume ces différentes possibilités.

7.5 LA SUBSTITUTION DE COMMANDE

La substitution de commande ou backquoting permet d'utiliser le résultat d'une commande comme argument d'une autre commande. Pour utiliser cette fonctionnalité, il faut entourer la commande soit d'accents graves, ou « backquotes » (*`commande`*), soit *$(commande)*. La forme *$(commande)* est plus récente et doit être préférée.

La commande placée entre *$(cmd)* est exécutée en premier, avant l'exécution de la ligne de commandes dont elle fait partie. Son résultat, c'est-à-dire la sortie standard, est entièrement intégré à la ligne de commandes en remplacement de la commande « backquotée ». La ligne de commandes est alors exécutée avec ces nouveaux arguments.

Exemple 1

La commande *echo* affiche à l'écran la chaîne de caractères qui la suit.

```
xstra> echo pwd
pwd
xstra> echo 'pwd'
pwd
xstra> echo `pwd`  § Ancien format
/home/xstra
xstra> echo $(pwd)
/home/xstra
xstra>
```

Dans ce dernier cas, la commande *pwd* qui indique le répertoire courant est exécutée. Son résultat devient l'argument de la commande *echo* (*echo /home/xstra*).

Exemple 2

```
xstra> less $(grep  l 'toto' *)
```

Cette commande permet de visualiser le contenu des fichiers du répertoire courant contenant au moins une fois la chaîne de caractères *toto*.

Exemple 3

```
xstra> echo il y a $(who | wc -l) utilisateurs connectes
```

Cette commande nous indique le nombre d'utilisateurs connectés.

Exemple 4

```
xstra> grep  n "motif" $(find .  type f -print)
```

Cette commande permet de rechercher à partir du répertoire courant et récursivement dans tous les sous-répertoires, les fichiers contenant la chaîne de caractères *motif*.

Exemple 5

```
xstra> rm  i $(find .  mtime +20 -print)
```

Cette commande permet de supprimer les fichiers n'ayant pas été modifiés depuis plus de 20 jours. Elle est équivalente à :

```
xstra> find .  mtime +20  exec rm  i {} \;
```

7.6 LES COMMANDES GROUPÉES

La commande groupée est une succession de commandes séparées par le caractère **;** et considérées comme un ensemble. Cet ensemble, repéré par des parenthèses (…), est exécuté par un nouveau processus shell. Les commandes seront exécutées séquentiellement, sans influence les unes sur les autres.

Le résultat d'une commande groupée est cependant différent de celui qu'auraient les mêmes commandes réalisées séquentiellement.

Exemple

Suppression du fichier *temporaire* dans le répertoire *projet1*.

```
xstra> cd
xstra> (cd projet1; rm temporaire)
xstra> pwd
/home/xstra
 § Le répertoire courant est toujours /home/xstra
```

Même action que la commande groupée mais le répertoire courant sera *projet1*.

```
xstra> cd projet1; rm temporaire
xstra> pwd
/home/xstra/projet1
xstra>
```

À la différence d'une ligne de commandes séquentielles (paragraphe 7.1.3), Linux interprétera ceci comme une seule commande au niveau de la redirection des entrées-sorties.

Exemple

```
xstra> echo "aujourd'hui"; date; \<return>
echo les personnes suivantes; \<return>
 who; echo sont connectees >fs
xstra>
```

Le fichier *fs* contiendra la chaîne de caractères "*sont connectees*", résultat de la dernière commande à laquelle a été appliquée la redirection.

```
xstra> (echo "aujourd'hui"; date; echo les \<return>
> personnes suivantes; who; echo sont connectees) >fs
```

Dans ce cas, le fichier *fs* contiendra le résultat de chacune des commandes :

```
aujourd'hui
14 fevrier 2000
les personnes suivantes
xavier   console   fev 14 08:30
yannick  tty1      fev 14 08:45
soline   tty2      fev 14 09:00
sont connectees
```

Une autre particularité de la commande groupée est de permettre de lancer tout le groupe de commandes en arrière-plan. En effet, le caractère ***&*** à la fin d'une ligne séquentielle ne lance en arrière-plan que la dernière commande.

7.7 LES CARACTÈRES SPÉCIAUX GÉNÉRATEURS DE NOMS DE FICHIER

Les caractères génériques permettent de désigner un ensemble d'objets. Ils peuvent être utilisés pour la génération d'un ensemble de noms de fichiers. Ils s'appliquent donc aux paramètres des commandes qui désignent des **noms** de fichiers.

Ils peuvent aussi désigner un ensemble de chaînes de caractères. On parle alors d'expressions régulières. Ces expressions s'appliquent aux commandes d'édition (*ex*, *vi*, *sed*, …) ou à des filtres (*grep*, *egrep*, *awk*, …) et permettent la recherche d'une chaîne de caractères dans un fichier. Elles s'appliquent donc aux **contenus** des fichiers.

Il y a des différences d'interprétation entre les caractères spéciaux (ou métacaractères) utilisés dans la génération des noms de fichier (présentation ci-dessous) et ceux utilisés dans les expressions régulières (voir le chapitre 14).

L'interpréteur de commandes permet de générer une liste de noms de fichier en utilisant les caractères spéciaux suivants :

`*` désigne toutes chaînes de caractères, y compris la chaîne vide.

Exemple

`a*b` désigne tous les noms de fichier commençant par a et finissant par b.

`?` désigne un caractère quelconque.

Exemple

`a?b` désigne tous les noms de fichier commençant par `a` suivi d'un caractère quelconque et finissant par `b`.

`[...]` désigne un caractère quelconque appartenant à la liste donnée entre crochets. Deux caractères séparés par un tiret ( ) définissent une liste de caractères rangés par ordre alphabétique, dont le premier élément est le premier caractère et le dernier élément le dernier caractère.

Exemple

`a[a z0 9A Z]b` désigne tous les noms de fichier commençant par `a` suivi d'un caractère alphanumérique et finissant par `b`.

`[!...]` désigne une liste de caractères à exclure.

Exemple

`a[!a z]b` désigne tous les noms de fichier commençant par `a` suivi d'un caractère autre qu'un caractère alphabétique en minuscule, et finissant par `b`.

Il est possible d'annuler l'interprétation d'un caractère spécial en le faisant précéder de \. Cela est également possible en délimitant par une apostrophe (') ou une double apostrophe (") une chaîne de caractères contenant un ou plusieurs caractères spéciaux. Il existe une différence entre les deux délimiteurs (voir chapitre 8).

En Bash, le caractère ~ (tilde) désigne le répertoire d'accueil défini dans la variable $HOME.

Le caractère ! (point d'exclamation) est aussi utilisé par l'éditeur de commandes du Bash (voir le chapitre 6.5).

7.8 LES CARACTÈRES DE NEUTRALISATION

Nous avons vu que certains caractères spéciaux ont une signification particulière pour l'interpréteur de commandes, comme par exemple : *<,>,*,?,!,...* Il est possible, en plaçant le caractère **** (backslash) devant un tel caractère spécial, de **neutraliser son interprétation** par l'interpréteur de commandes. On peut également neutraliser l'interprétation d'un alias.

Par exemple, le paragraphe 7.1.4 présente le caractère \ placé en fin de ligne comme technique permettant l'écriture d'une commande sur plusieurs lignes. En fait le caractère \ neutralise le caractère qui suit, c'est-à-dire le caractère <return> désignant le caractère de fin de commande.

Exemple

```
xstra> touch f\*1
xstra> ls
f*1 fichier1 fichier2
xstra> rm f\*1
xstra> ls
fichier1 fichier2
xstra>
```

Attention

*rm f*1* supprime tous les fichiers commençant par *f* et se terminant par *1*.

L'utilisation du caractère **** n'est pas pratique si on souhaite neutraliser une chaîne de caractères. Dans ce cas, il est préférable d'encadrer la chaîne à neutraliser par des apostrophes, appelées aussi **quotes** (’). Le seul caractère ne pouvant être neutralisé est la quote elle-même.

Exemple

```
xstra> touch 'f*?1'
xstra> ls
f*?1 fichier1 fichier2
xstra> rm 'f*?1'
xstra> ls
fichier1 fichier2
xstra>
```

Attention

rm f?1* supprime tous les fichiers commençant par *f*, suivi d'au moins un caractère et se terminant par *1*.

Il existe un troisième mécanisme de neutralisation qui utilise le caractère " (double quotes). Dans une chaîne délimitée par ce caractère ", certains métacaractères sont interprétés, dont le remplacement d'un paramètre (exemple $VAR) et l'évaluation d'une commande. Par contre il n'y a pas d'interprétation des espaces, ni de génération de nom de fichier.

Exemple

```
xstra> echo  e "Les fichiers suivants sont dans \<return>
le repertoire $PWD : \n $(ls)"
Les fichiers suivants sont dans le repertoire /xstra :
listf projet1
xstra>
```

7.9 EXERCICES

Exercice 7.9.1

Quelles commandes Linux devez-vous exécuter pour obtenir à l'écran :

Il y a xxx utilisateurs de ce systeme dont le login shell est bash

Exercice 7.9.2

En utilisant la commande *find*, trouvez et listez les noms de :

1) tous les fichiers sous le répertoire */etc* dont les noms commencent par *rc*,

2) tous les fichiers réguliers vous appartenant ; mettez le résultat dans le fichier */tmp/findmoi* et les erreurs dans */dev/null*,

3) tous les sous–répertoires de */etc*,

4) tous les fichiers réguliers se trouvant sous votre répertoire d'accueil et qui n'ont pas été modifiés dans les 10 derniers jours

Exercice 7.9.3

Trouvez à partir de votre répertoire d'accueil, le nombre de fichiers ayant une taille supérieure à 1 Mega-octets et stockez leurs noms dans un fichier (utilisez la commande *tee*).

Exercice 7.9.4

Créez dans le répertoire *rep1* le fichiers suivants : *fich1*, *fich2*, *fich11*, *fich12*, *fich1a*, *ficha1*, *fich33*, *.fich1*, *.fich2*, *toto*, *afich*.

Listez les fichiers :

1) dont les noms commencent par *fich*,

2) dont les noms commencent par *fich* suivi d'un seul caractère,

3) dont les noms commencent par `fich` suivi d'un chiffre,
4) dont les noms commencent par `.`,
5) dont les noms ne commencent pas par `f`,
6) dont les noms contiennent `fich`.

Exercice 7.9.5

Écrivez un alias en Bash permettant de lister page par page et dans l'ordre alphabétique l'ensemble des variables d'environnement.

Exercice 7.9.6

Créez un fichier de nom -i, puis supprimez-le.

Exercice 7.9.7

Il arrive souvent de lancer une commande Linux produisant plusieurs pages d'écran à toute vitesse (par exemple : `ls  l /etc`). Il faut alors relancer la même commande, en envoyant sa sortie standard dans `less`, ce qui permet d'examiner le résultat page par page. Si votre shell interactif est le Bash, cela ne doit pas être fastidieux : créez l'alias `p` qui relance la dernière commande en envoyant sa sortie standard dans la commande `less`.

Chapitre **8**

La programmation en shell

Le shell est plus qu'un interpréteur de commandes : c'est également un puissant langage de programmation. Cela n'est pas propre à Linux ; tout système d'exploitation offre cette possibilité d'enregistrer dans des fichiers des suites de commandes que l'on peut invoquer par la suite. Mais aucun système d'exploitation n'offre autant de souplesse et de puissance que le shell Linux dans ce type de programmation. Le revers de cette médaille est que la syntaxe de ce langage est assez stricte et rébarbative. De plus, l'existence de plusieurs shells conduit à plusieurs langages différents.

Sous Linux, un fichier contenant des commandes est appelé **script** et nous n'emploierons plus que ce terme dans la suite. De même nous utiliserons le terme **shell** pour désigner à la fois l'interpréteur de commandes et le langage correspondant (tout comme " assembleur " désigne à la fois le langage assembleur et le compilateur de ce langage).

Comme tout langage de programmation conventionnel, le shell comporte des instructions et des variables. Les noms de variables sont des chaînes de caractères ; leurs contenus sont également des chaînes de caractères.

L'assignation (Bourne-shell, POSIX-shell et Bash) d'une valeur à une variable se fait par un nom ; la référence à cette variable se fait par son nom précédé du caractère *$*, comme dans :

```
mavariable bonjour  § assignation
echo $mavariable    § référence
```

Le jeu d'instructions lui-même comporte :

- toutes les commandes Linux,
- l'invocation de programmes exécutables (ou de scripts) avec passage de paramètres,

- des instructions d'assignation de variables,
- des instructions conditionnelles et itératives,
- des instructions d'entrée-sortie.

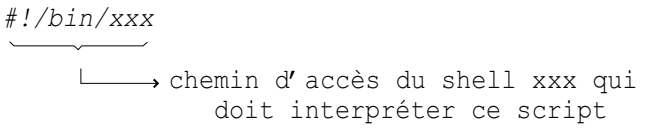
Et bien entendu, les mécanismes de tubes et de redirections sont utilisables dans un script.

Remarques

1) Le shell est un langage interprété ; en conséquence tout changement dans le système sera pris en compte par un script lors de sa prochaine utilisation (il est inutile de " recompiler " les scripts).

2) Il est tout à fait possible d'écrire et d'invoquer des scripts dans un certain shell tout en utilisant un autre shell en interactif.
   En particulier, il est très fréquent (mais non obligatoire) d'utiliser un TC-shell en tant que "login shell" et le Bourne-shell ou un autre shell pour l'écriture des scripts.
   Les scripts les plus simples (listes de commandes) seront identiques quel que soit le shell, mais dès que des instructions de tests ou d'itérations sont nécessaires, les syntaxes du Bourne-shell, du Bash et du C-shell diffèrent.

3) Si un script commence par la ligne



   il est interprété par le shell */bin/xxx*.

Notre objectif ici n'est pas une étude exhaustive de la programmation en shell, mais une introduction à cette technique illustrant les notions et instructions principales.

8.1 LA PROGRAMMATION DE BASE EN SHELL

Dans ce qui suit, on supposera que l'environnement de l'utilisateur est le Bash, et qu'il écrit ses scripts dans le langage de l'interpréteur de commandes Bash. Les bases de programmation exposées dans ce paragraphe peuvent être considérées comme communes à tous les interpréteurs de commandes issus de la famille des Bourne-shell (Bourne-shell, POSIX-shell, Bash).

Attention

Toujours commencer un shell script par la ligne *#!/bin/bash*.

8.1.1 Le premier script

Création avec l'éditeur *vi* du fichier *listf* contenant la ligne *ls aCF*.

Un fichier ordinaire n'a pas le droit *x* (il n'est pas exécutable) à sa création, donc :

```
xstra> chmod a+x listf  § ajoute le droit x
                        § pour tout le monde.
```

Il peut donc être exécuté comme une commande :

```
xstra> listf
./    .kshrc    .securite/    bin/   florent/   xavier/
../   .profile  .sh_history   dpt/   jerome/
xstra>
```

Exécution du script en mode mise au point :

```
xstra> sh  x listf   § mode trace
```

ou

```
xstra> sh  v listf   § mode verbose
```

permettent de demander au shell qui interprète le script de tracer le déroulement du script ou de le commenter (ou les deux). Il est également possible d'inclure **dans** le script les lignes :

```
set  x        (pour le mode trace)
set  v        (pour le mode verbose)
```

Le **mode trace** recopie sur la sortie standard chaque ligne telle qu'elle est interprétée. Le **mode verbose** recopie sur la sortie standard chaque ligne avant interprétation.

8.1.2 Le passage des paramètres

Le script *listf* ne s'applique qu'au répertoire courant. On peut le rendre plus général en lui transmettant le nom d'un répertoire en argument lors de l'invocation. Pour ce faire, les variables 1, 2, …, 9 permettent de désigner respectivement le premier, le deuxième, …, le neuvième paramètre associés à l'invocation du script.

a) Premier script avec passage de paramètres

Avec *vi*, modifier le fichier *listf* de la façon suivante :

```
echo "contenu du repertoire $1 "
ls  aCF $1
```

L'exécution donne :

```
xstra> listf /tmp
contenu du repertoire /tmp
```

```
./  ../  df_file
xstra>
```

b) Généralisation

Le nombre de paramètres passés en argument à un script n'est pas limité à 9 ; toutefois seules les neuf variables 1, …, 9 permettent de désigner ces paramètres dans le script.

La commande *shift* permet de contourner ce problème. Après *shift*, le ième paramètre est désigné par *$i 1*.

Exemple 1

Le script *echopara* contient :

```
echo $1 $2 $3
P1 $1
shift
echo $1 $2 $3
echo $P1
```

L'exécution donne :

```
xstra> echopara un deux trois
un deux trois
deux trois
un
xstra>
```

Cet exemple montre le comportement de *shift*, l'affectation d'une valeur à la variable *P1 (P1=$1)* et la référence à cette variable (*echo $P1*).

Exemple 2

Le script *echopara1* de décalage des paramètres contient :

```
echo $1 $2 $3 $4 $5 $6 $7 $8 $9
shift
echo $1 $2 $3 $4 $5 $6 $7 $8 $9
```

L'exécution donne :

```
xstra> echopara1 1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 10
xstra>
```

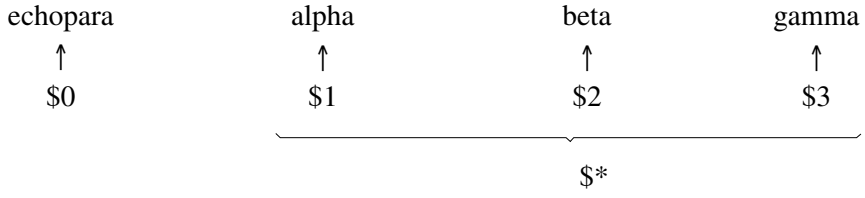
8.1.3 Les variables spéciales

En plus des variables 1, 2, …, 9, le shell prédéfinit des variables facilitant la programmation.

0 contient le nom sous lequel le script est invoqué,

contient le nombre de paramètres passés en argument,
*** contient la liste des paramètres passés en argument,
? contient le code de retour de la dernière commande exécutée,
$ contient le numéro de process (PID) du shell (décimal).

Exemple 1



Exemple 2

Le script *echopara2* contient :

```
echo $0 a ete appele avec $# parametres
echo qui sont : $*
```

L'exécution donne :

```
xstra> echopara2 a b c d
./echopara2 a ete appele avec 4 parametres
qui sont : a b c d
xstra>
```

Attention, la variable *#* n'est pas une variable numérique (qui n'existe pas) mais une variable de type chaîne de caractères (de même pour *$*).

0 permet de savoir sous quel nom ce script a été invoqué. Dans le cas où le script porte plusieurs noms (par des liens), cela permet de prendre telle ou telle décision suivant le nom sous lequel le script a été invoqué.

$ numéro de process (PID) du shell, est unique dans le système : il est fréquemment utilisé pour générer des noms de fichiers temporaires.

Exemple

```
tempfil /tmp/infile_user_$$
$ utilisation du fichier temporaire
rm $tempfil
```

8.1.4 Les caractères spéciaux

Attention

Ces caractères sont générateurs des noms de fichiers et sont utilisés dans le passage des paramètres.

Selon un mécanisme général (voir le paragraphe 3.2 et 7.7), le shell peut générer une liste des noms de fichiers correspondant à un certain modèle (grâce aux caractères * et ?). Cette génération a lieu **avant** l'invocation de la commande (et donc du script) concernée. Par exemple, si le répertoire courant contient uniquement les fichiers *fich1* et *fich2*, lors de la commande `ls fi*`, le shell génère la liste *fich1 fich2* et la passe en argument à la commande `ls`. La commande effectivement lancée par le `shell` est donc : `ls fich1 fich2`.

8.1.5 Les instructions de lecture et d'écriture

Ces instructions permettent de créer des fichiers de commandes interactifs par l'instauration d'un dialogue sous forme de questions/réponses. La question est posée par l'ordre *echo* et la réponse est obtenue par l'ordre *read* à partir du clavier.

read variable1 variable2… variablen

read lit une ligne de texte à partir du clavier, découpe la ligne en mots et attribue aux variables *variable1* à *variablen* ces différents mots. S'il y a plus de mots que de variables, la dernière variable se verra affecter le reste de la ligne.

Exemple

Le script *affiche* contient :

```
echo  n "Nom du fichier a afficher : "
read fichier
more $fichier
```

8.1.6 Les structures de contrôle

Le shell possède des structures de contrôle telles qu'il en existe dans les langages de programmation d'usage général :

- instructions conditionnelles (if.. then.. else, test, case),
- itérations bornées,
- itérations non bornées.

a) Les instructions conditionnelles

Pour la programmation des actions conditionnelles, nous disposons de trois outils :

- l'instruction if,
- la commande test qui la complète,
- l'instruction case.

➤ L'instruction if

Elle présente trois variantes qui correspondent aux structures sélectives à une, deux ou n alternatives.

α) La sélection à une alternative : if… then… fi

```
if commande
then commandes
fi
```

Les *commandes* sont exécutées si la commande condition *commande* renvoie un code retour nul ($? = 0).

Exemple

Le script *echoif1* contient :

```
if grep  i xstra1 /etc/passwd
then echo L\'utilisateur xstra1 est connu du systeme
fi
```

β) La sélection à deux alternatives : if… then… else… fi

```
if commande
then commandes1
else commandes2
fi
```

Les commandes *commandes1* sont exécutées si la commande_condition *commande* renvoie un code retour nul, sinon ce sont les *commandes2* qui sont exécutées.

Exemple

Le script *echoif2* contient :

```
if grep  qi "xstra1" /etc/passwd
then echo L\'utilisateur xstra1 est connu du systeme
else echo L\'utilisateur xstra1 est inconnu du systeme
fi
```

γ) La sélection à n alternatives : if… then…. elif… then… fi

```
if commande1
then commandes1
elif commande2
then commandes2
elif commande3
then commandes3
…
else
commandes0
fi
```

Exemple

```
#!/bin/bash
# ce script a pour nom scriptf
# ce script montre comment utiliser
# les parametres remplacables
# lors de l'invocation d'un script.
# exemples: xstra>scriptf
#           xstra>scriptf 1
#           xstra>scriptf 1 2 3 4
#           xstra>scriptf "1 2 3 4"
#           xstra>scriptf ok un deux trois
#
echo ""     #echo d'une ligne vide   saut de ligne
echo ""
echo "Exemple de passage de parametres a un script."
echo ""
# la ligne suivante montre comment utiliser
# le resultat d'une commande:
echo "Le repertoire courant est: `pwd`"
# la ligne suivante montre la difference entre
# simples et doubles apostrophes:
echo 'Le repertoire courant est: `pwd`'
echo "Ce script a pour nom: $0"
if [  $#  eq 0 ]  # comment tester le nombre de parametres?
  then
    echo "Il a ete appele sans parametre."
  else
    if [  $#  eq 1 ]
      then
        echo "Il a ete appele avec le parametre $1"
      else
        echo "Il a ete appele avec $# parametres, qui sont: $*"
    fi
    if [  $1   'ok' ]  # comment tester le contenu d'un parametre
      then
        echo "Bravo: le premier parametre vaut $1"
      else
        echo "Helas, le premier parametre ne vaut pas ok mais $1"
    fi
fi
echo "Au revoir $LOGNAME, le script $0 est fini."
```

➤ La commande test

Elle constitue l'indispensable complément de l'instruction *if*. Elle permet très simplement :

- de reconnaître les caractéristiques des fichiers et des répertoires,
- de comparer des chaînes de caractères,
- de comparer algébriquement des nombres.

Cette commande existe sous deux syntaxes différentes :

```
test expression
```

ou

```
[ expression ]
```

La commande *test* répond à l'interrogation formulée dans *expression*, par un code de retour nul en cas de réponse positive et différent de zéro sinon.

La deuxième forme est plus fréquemment rencontrée et donne lieu à des programmes du type :

```
if [ expression ]
then commandes
fi
```

Attention

Dans *[expression]*, ne pas oublier le caractère espace entre *[* et *expression* et entre *expression* et *]*. Si *then* est sur la même ligne, il doit être séparé du *]* par un espace et un caractère ; les expressions les plus utilisées sont :

-d nom	vrai si le répertoire nom existe,
-f nom	vrai si le fichier nom existe,
-s nom	vrai si le fichier nom existe et est non vide,
-r nom	vrai si le fichier nom existe et est accessible en lecture,
-w nom	vrai si le fichier nom existe et est accessible en écriture,
-x nom	vrai si le fichier nom existe et est exécutable,
-z chaîne	vrai si la chaîne de caractères chaîne est vide,
-n chaîne	vrai si la chaîne de caractères chaîne est non vide,
c1 = c2	vrai si les chaînes de caractères c1 et c2 sont identiques,
c1 != c2	vrai si les chaînes de caractères c1 et c2 sont différentes,
n1 -eq n2	vrai si les entiers n1 et n2 sont égaux.

(Les autres opérateurs relationnels sont *ne*, *lt*, *le*, *-gt* et *-ge*.)

Remarque

Les expressions peuvent être niées par l'opérateur logique de **négation** *!* et combinées par les opérateurs **ou logique** *o* et **et logique** *a*.

➤ L'instruction case

L'instruction *case* est une instruction très puissante et très commode pour effectuer un choix multiple dans un fichier de commandes.

```
case chaine in
motif1) commandes 1 ;;
motif2) commandes 2 ;;
```

```
….
….
motifn) commandes n ;;
esac
```

Le shell recherche, parmi les différentes chaînes de caractères *motif1*, *motif2*,..., *motifn* proposées, la première qui correspond à *chaine* et il exécute les commandes correspondantes. Un double point virgule (*;;*) termine chaque choix. La *chaine* dans un *case* peut prendre diverses formes :

- un chiffre,
- une lettre ou un mot,
- des caractères spéciaux du shell,
- une combinaisons de ces éléments.

La *chaine* peut être lue, passée en paramètre ou être le résultat d'une commande exécutée avec l'opérateur backquote ` ` ou $( ). Dans les différentes chaînes *motif1* à *n*, on peut utiliser les caractères spéciaux (*****, **?**,…). De plus, pour regrouper plusieurs motifs dans une même alternative, on utilise le caractère **|** (obtenu sur un clavier standard par la combinaison <Alt Gr><6>).

Exemple 1

Le script *comptepara* contient :

```
case $# in
0) echo $0 sans argument ;;
1) echo $0 possede un argument ;;
2) echo $0 a deux arguments ;;
*) echo $0 a plus de deux arguments ;;
esac
```

Exemple 2

Le script *append* contient :

```
case $# in
2) if [ ! f $1 ]
    then
    echo le fichier $1 ne se trouve pas dans le repertoire
    elif [ ! s $1 o ! r $1 ]
      then
        echo le fichier $1 est vide ou protege en lecture
    elif [ ! f $2 ]
      then
        cat $1 >$2
        echo $0 a copie $1 en $2
    elif [ ! w $2 ]
      then
        echo le fichier $2 est protege en ecriture
    else
```

```
        cat $1 >>$2
        echo $0 a rajoute $1 en fin de $2
fi
;;
*) cat <<EOT
     append
     Fonction : le fichier fsource est ajoute au fichier fcible
     Si fcible n'existe pas, fsource est copie en fcible
     Syntaxe : append fsource fcible
EOT
;;
esac
```

Exemple 3

Le script *r4d* contient :

```
# r4d : real*4 dump ( r8d : real*8 dump )
# i2d : integer*2 dump
# ce script permet d'examiner un fichier binaire
# contenant des nombres réels ou entiers codés
# sur 2 ou 4 octets
case $(basename $0) in
r4d) option ' t f4' ;;
r8d) option ' t f8' ;;
i2d) option ' t d2' ;;
i4d) option ' t d4' ;;
esac
od $option $1 | less
```

Dans le répertoire contenant *r4d*, on crée les liens *r8d i2d* et *i4d*

```
xstra> ln r4d r8d; ln r4d i2d; ln r4d i4d
```

On peut maintenant utiliser les commandes *r4d, r8d, i2d, i8d* qui correspondent à un seul et même shell script dont le comportement dépend du nom sous lequel il aura été invoqué.

b) Les itérations

La présence des instructions itératives dans le shell en fait un langage de programmation complet et puissant. Le shell dispose de trois structures itératives : *for*, *while* et *until*.

➤ Itération bornée : La boucle for

Trois formes de syntaxe sont possibles :

1) Forme 1

```
for variable in chaine1 chaine2… chainen
do
commandes
```

```
done
```

2) Forme 2

```
for variable
do
commandes
done
```

3) Forme 3

```
for variable in *
do
commandes
done
```

Pour chacune des trois formes, les commandes placées entre *do* et *done* sont exécutées pour chaque valeur prise par la variable du shell *variable*. Ce qui change c'est l'endroit où *variable* prend ses valeurs. Pour la forme 1, les valeurs de *variable* sont les chaînes de *chaine1* à *chainen*. Pour la forme 2, *variable* prend ses valeurs dans la liste des paramètres du script. Pour la forme 3, la liste des fichiers du répertoire constitue les valeurs prises par variable.

Exemple 1

Le script *echofor1* contient :

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

L'exécution donne :

```
xstra> echofor1
un
deux
trois
xstra>
```

Exemple 2

Le script *echofor2* contient :

```
for i
do
echo $i
done
```

L'exécution donne :

```
xstra> echofor2 le systeme Linux
le
systeme
```

```
Linux
xstra>
```

Exemple 3

Le script *echofor3* contient :

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

L'exécution donne :

```
xstra> echofor3
fich1
fich2
fich3
xstra>
```

Exemple 4

Le script *lsd* contient :

```
echo "liste des repertoires sous `pwd`"
echo "                                "
for i in *
do
  if [   d $i ]
  then
  echo $i "   :repertoire"
  fi
done
echo "               "
```

L'exécution donne :

```
xstra> lsd
liste des repertoires sous /home/xstra/test

filon : repertoire

xstra>
```

➤ Itérations non bornées : while et until

```
while commandealpha
do commandesbeta
done
```

```
until commandealpha
do commandesbeta
done
```

Les commandes *commandesbeta* sont exécutées tant que (*while*) ou jusqu'à ce que (*until*) la commande *commandealpha* retourne un code nul (la condition est vraie).

Exemple 1

Le script suivant liste les paramètres qui lui sont passés en argument jusqu'à ce qu'il rencontre le paramètre *fin*.

```
#!/bin/bash
# ce script a pour nom while_
# ce script montre le fonctionnement
# de la construction while
# exemple: xstra>while_ 1 2 3 fin 4 5 6
#
while [ $1 !  fin ];do
  echo $1
  shift
done
```

L'exécution donne :

```
xstra> while_ 1 2 3 4 fin 5 6 7
1
2
3
4
xstra>
```

Exemple 2

```
#!/bin/bash
# ce script a pour nom until_
# ce script montre l'utilisation
# de la construction until
# exemple: xstra>until_ 1 2 3 fin 4 5 6
#
until [ $1   fin ] ;do
  echo $1
  shift
done
```

L'exécution donne :

```
xstra> until_ 1 2 3 fin 4 5 6
1
2
3
xstra>
```

Exemple 3

```
#!/bin/bash
# ce script a pour nom while_until
# il illustre l'usage combine
# des constructions while et until
# exemple: xstra>while_until 1 2 3 debut 4 5 6 fin 7 8 9
#
while [  $1 !  debut ] ;do
  shift
done
shift
until [  $1   fin ] ;do
  echo $1
  shift
done
```

L'exécution donne :

```
xstra> while_until 1 2 3 debut 4 5 6 fin 7 8 9
4
5
6
xstra>
```

8.1.7 Script récapitulatif

```
#!/bin/bash
# Le nom de ce script est cherche.
# Ce script illustre: le passage d'arguments,
#                le test sur le nombre des arguments,
#                le test sur le nom d'invocation
#                la combinaison logique de conditions
#                les redirections
#                le "single quoting",le "double quoting"
#                et le "back quoting"
#
# Ce script recherche tous les fichiers
# dont le nom est passe en  premier argument.
# La recherche est faite dans toute l'arborescence,
# a partir d'un repertoire passe en deuxieme argument.
# Si ce deuxieme argument est absent,la recherche
# est faite a partir du repertoire courant.
# Les messages d'erreur pouvant resulter de
# l'absence de droits d'acces a des sous repertoires
# sont ignores. La liste des fichiers trouves est placee
# dans un fichier temporaire.
# La recherche pouvant etre longue, ce script est de
# preference lance en arriere plan.
# Lorsque la recherche est terminee, invoquer ce script
# sans argument permet de connaitre le resultat
```

```
# de la recherche. Il est montre comment utiliser
# le nom d'invocation pour prendre une decision :
# un lien a ete cree par la commande ln cherche montre.
# Invoquer le script par montre, sera equivalent
# a l'invoquer par cherche sans arguments.
#
# exemple: xstra> cherche passwd / &
#                ..............
#                un certain temps s'ecoule,
#                mais on peut faire autre chose...
#                ..............
#          xstra> montre           (ou xstra> cherche)
#
#          Resultat de la recherche:
#          /etc/passwd
#          /bin/passwd
#
#          xstra>
# exemple: xstra> cherche 'pass*' / &
#          meme resultat: dans ce cas
#          les simples apostrophes evitent
#          le remplacement de pass*
#          lors de l'invocation du script, et
#          les doubles apostrophes autour de $1
#          dans le find permet la recherche desiree.
#          Sans les simples apostrophes,
#          le shell en cours remplacerait
#          pass* par la liste de tous les noms
#          de fichier commencant par pass
#          dans le repertoire courant,
#          avant d'invoquer le script.
#          Sans les doubles apostrophes,
#          le shell ferait de meme, mais pendant
#          l'interpretation du script:
#          le resultat depend alors de l'existence
#          de fichiers dont le nom commence par pass
#          dans le repertoire courant.
#
TEMPFILE /tmp/cherche_temporary_file_$LOGNAME
PROG $(basename $0)
if [  $#  ne 0  a $PROG !  montre ] ;then
#                 invoque par cherche avec argument(s)
  rm $TEMPFILE 2>/dev/null
  RACINE `pwd`
  if [  $#  eq 2 ]
    then RACINE $2
  fi
  (find $RACINE  name ”$1”  print) 1>$TEMPFILE 2>/dev/null
  if [  !  s $TEMPFILE ]
    then echo "$1 non trouve a partir de $RACINE" >$TEMPFILE
```

```
  fi
else # invoque sans arguments ou par le nom: montre
 if [  !  s $TEMPFILE ]
    then echo "Pas de recherche en cours."
  else
    echo ""
    echo "Resultat de la recherche:"
    more $TEMPFILE
    echo ""
    rm $TEMPFILE
  fi
fi
```

8.1.8 Arithmétique entière sur des variables

Le Bourne-shell ne permet pas de définir des variables numériques, et pourtant les opérateurs -*eq ne lt le gt ge* existent. Ce n'est pas une incohérence. Toutes les variables définies en Bourne-shell sont de type chaîne de caractères, mais si le contenu de cette chaîne représente un nombre entier, les opérateurs précédents peuvent s'appliquer.

Les opérations arithmétiques sur variables sont très faciles en bash grâce à deux mécanismes non disponibles dans le Bourne-shell traditionnel :
L'évaluation arithmétique :

$((variable1 opérateur variable2))

Le test arithmétique :

if ((variable1 opérateur variable2))

Ces deux mécanismes seront présentés plus loin.

8.2 LA PROGRAMMATION AVANCÉE EN BASH

L'étude exhaustive du Bash en tant que langage de commandes nécessiterait un livre entier. Nous présenterons donc simplement les points les plus importants par rapport aux concepts de base exposés lors du paragraphe précédent. De plus il existe des différences entre les versions du Bash. Dans ce paragraphe nous considérerons être dans la version 2 (*bash version*). Certains points développés ci-dessous ne fonctionnent pas en version 1.

8.2.1 Les variables prédéfinies du Bash (non définies en Bourne-shell)

Les variables mises à jour dynamiquement par le Bash sont les suivantes :

PPID	numéro du processus père (Parent Process IDentifier),
PWD	répertoire de travail,
RANDOM	un nombre aléatoire,
SECONDS	temps écoulé depuis le lancement de ce Korn-shell,
!	numéro du dernier processus lancé en arrière-plan,

_ (underscore) dernier mot de la dernière commande exécutée.

8.2.2 Définition de variable : la commande declare

En plus de la forme *variable=valeur*, le Bash possède une commande générale de définition de variable :

declare [+/ option] [p] [variable[valeur]]

option	affecte l'attribut à variable
+option	enlève l'attribut à variable

L'option *–p*, exclusive des autres options, permet d'afficher les variables et leur valeur. L'option *–f* ou *–F* permet d'afficher les fonctions avec ou sans leur définition. Les autres options sont :

-a	*variable* est de type tableau,
-i	*variable* est de type numérique entier,
-r	variable en lecture seule,
-x	variable exportée (variable placée dans l'environnement)

La commande *typeset* est équivalente à *declare*.

Dans le cas de la création d'un tableau la syntaxe la plus simple est :

declare a tab=(element1 element2 element3)

Le tableau *tab* aura 3 éléments qui seront : *tab[0]=element1*, *tab[1]=element2*, *tab[2]=element3*. La création d'un élément du tableau *tab* est réalisée en utilisant la syntaxe simple *tab[numero element]=valeur*.

Attention

Les éléments d'un tableau de variables sont numérotés de `0` à `N 1`.

Exemple

```
xstra> declare  a arbres=(sapin chene acacia orme)
§ définition du tableau arbres
xstra> declare  p arbres
declare  a arbres '([ 0]  "sapin" [ 1]  "chene" [ 2]  "acacia"
[ 3]  "orme")'
xstra> echo $arbres        § désigne le premier élément sapin
sapin
xstra> echo ${arbres[2]}   § désigne l'élément numéro 2 (0, 1, ..)
acacia
xstra> echo ${arbres[*]}   § désigne la suite de tous les éléments
sapin chene acacia orme
xstra> echo ${#arbres[1]} § désigne la taille de l'élément 1
5
xstra> echo ${#arbres[*]} § désigne le nombre d'éléments
```

```
4
xstra>
```

L'option *+a* n'existe pas. Pour détruire un tableau il faut utiliser la commande *unset tab*.

8.2.3 La commande test

La commande *test expression* ou *[expression]* est la même qu'en Bourne-shell, mais le jeu possible pour *expression* y est plus riche.

a) L'opérateur == peut remplacer = pour la comparaison

c1 == c2 vrai si les chaînes de caractères c1 et c2 sont identiques, (notation préférable au simple =)

b) Test sur le type d'un fichier

expression - Le code de retour est vrai si :

-a fich *fich* existe
-b fich *fich* est de type spécial bloc
-c fich *fich* est de type spécial caractère
-d fich *fich* est un répertoire (directory)
-f fich *fich* est un fichier
-L fich *fich* est un lien symbolique
-p fich *fich* est un tube nommé (pipe)
-s fich *fich* existe et est de taille non nulle

c) Test de relations entre fichiers

expression - Le code de retour est vrai si :

fich1 -ef fich2 *fich2* est un lien sur *fich1* (*ln*)
fich1 -nt fich2 *fich1* est plus récent que *fich2* (newer than)
fich1 -ot fich2 *fich1* est plus ancien que *fich2* (older than)

d) Test des droits d'accès d'un fichier

expression - Le code de retour est vrai si :

-r fich le processus en cours a le droit r sur *fich*
-w fich le processus en cours a le droit w sur *fich*
-x fich le processus en cours a le droit x sur *fich*
-g fich *fich* a le bit SGID positionné
-u fich *fich* a le bit SUID positionné

e) Le test [[…]]

Le test entre doubles crochets diffère du test entre simples crochets pour la comparaison entre chaînes de caractères avec les opérateurs et ! . Entre simples crochets, ces opérateurs comparent une chaîne à une autre chaîne ; entre doubles crochets, ils comparent une chaîne à un motif. Un motif peut contenir des caractères générateurs de noms.

L'exemple suivant illustre cette différence : comment tester si la variable TERM contient vt100 ou vt220 ou vt330 ou vtxxx…

Exemple

```
if [   $TERM    vt100 ] # 1 # correct, mais insuffisant
if [   $TERM    vt*   ] # 2 # FAUX
if [[  $TERM    vt*   ]] # 3 # correct : la bonne solution
# entre simples crochets il faudrait ecrire :
if [  $TERM    vt100  o $TERM    vt220  o $TERM    vt330 ]
```

Dans la ligne 2 de cet exemple, le shell remplace vt* par la liste de tous les noms de fichiers qui commencent par vt dans le répertoire courant avant de faire le test, ce qui n'est sans doute pas le but recherché.

8.2.4 L'arithmétique entière

L'utilisation de variables entières n'est pas prévue en Bourne-shell, et les manipulations arithmétiques y sont très incommodes. En Bash, l'arithmétique entière est facile, grâce à une notation adaptée : le double parenthésage **((…))**. Entre des doubles parenthèses, le bash interprète les caractères **< > () *** selon leur signification arithmétique usuelle, et le caractère $ n'est pas nécessaire devant un nom de variable. Le parenthésage y est possible, et sans parenthésage, la priorité des opérateurs arithmétiques est la priorité usuelle. Cette notation offre un cadre cohérent pour l'évaluation arithmétique et le test arithmétique.

a) L'évaluation arithmétique : $((…))

Quelques exemples valent mieux qu'une description formelle.

```
declare  i n1 n2 n3 x
n1 17
n2 3          # jusqu'ici rien de nouveau
n3 $((17/3))  # n3   17/3   5 (en entier)
n3 $((17%3))  # % : le reste de la division : n3 2
# la ligne suivante montre un calcul plus complexe
n1 $((n2*(n1+27) 5)) # n1 127
# l'écriture est très claire entre (( et ))
# c'est l'écriture arithmétique usuelle.
x $((1<<4))   # << : décalage à gauche : x 16
```

b) Le test arithmétique : if ((…))

Le test arithmétique offre les mêmes facilités d'écriture. Ceci est particulièrement pratique pour les caractères **<** **>** qui ne désignent plus des redirections, mais les opérateurs relationnels arithmétiques.

Exemples

```
if ((x>1000)) ; then           # valide et très lisible
# entre [ ] il faudrait écrire
if [  $x  gt 1000 ] ; then     # moins lisible

# encore plus convaincant :
if (( n2*(n1+27) 5) > n3 )) ; then # valide
if ((record_size*nbr_record > bufferlength)) ; then
```

Sans la notation ((…)), les deux dernières lignes de l'exemple précédent exigeraient des contorsions complètement illisibles.

Notons qu'il n'est pas nécessaire de déclarer les variables entières par la commande `declare i` pour utiliser l'évaluation et le test arithmétique. Il suffit que les variables contiennent des chaînes dont l'évaluation soit un entier.

8.2.5 L'écriture de script

Le Bash est un langage très puissant, sa syntaxe est complexe. Nous présenterons brièvement les mécanismes de parenthésage et de substitution, sources d'erreur de syntaxe.

a) Le parenthésage

Une suite de commandes peut être parenthésée par `()` ou par `{}` :

```
(commande1 ; commande2 ; commande3)
```

Les trois commandes sont exécutées dans un sous-shell.

```
{ commande1 ; commande2 ; commande3 ; }
```

Les trois commandes sont exécutées dans le shell en cours.

Attention

Ne pas oublier l'espace après `{` et le `;` avant `}`.

b) La substitution

`$variable` ou `${variable}`

Désigne la valeur de la variable. Si `variable` est un tableau, `$variable` ou `${variable}` désigne son premier élément (voir l'exemple arbres au paragraphe 8.2.2). Dans un shell script, la notation `${variable}` doit être préférée à la notation `$variable`.

Attention

Si *variable* n'est pas définie, le Bash substitue la chaîne vide.

${#variable}

Désigne la longueur de la *variable* (ou de son premier élément si c'est un tableau). Cas particulier : *${*}* ou *$** désigne le nombre de variables de position.

${#variable[*]}

Désigne le nombre d'éléments du tableau *variable* (voir l'exemple arbres au paragraphe 8.2.2).

${variable: chaine}

Désigne la valeur de la *variable* si celle-ci est définie et non vide, sinon désigne la chaîne *chaine*. Très utile pour fixer une valeur par défaut, par exemple :

```
TERM ${ TERM: VT220}
```

${variable:=chaine}

Désigne la valeur de la *variable* si celle-ci est définie et non vide. Sinon, la valeur *chaine* lui est affectée, puis la substitution a lieu.

${variable:?chaine}

Désigne la valeur de la *variable* si celle-ci est définie et non vide. Sinon, *chaine* est envoyée par le shell sur la sortie standard. Si ce shell n'est pas interactif, il se termine.

${variable:+chaine}

Désigne la chaîne *chaine* si la *variable* est définie et non vide. Sinon, désigne la chaîne vide.

c) La substitution de commande $(commande)

Le mécanisme de **substitution de commande** ou **backquoting** a été présenté au paragraphe 7.5. Le Bash reconnaît la notation POSIX *$(commande)*, beaucoup plus claire en cas d'utilisation à plusieurs niveaux d'imbrication. L'exemple suivant illustre cette possibilité. Comment copier dans le répertoire */tmp/backup* tous les fichiers dont le nom est **log** et qui se trouvent dans les répertoires listés par la variable PATH. La commande *tr* qui permet de remplacer le caractère *:* par un espace est expliquée au paragraphe 14.5.

```
xstra> cp $(find $(echo $PATH | tr ':' ' ') \
-type f -name '*log*' -print) /tmp/backup
xstra>
```

d) Les extensions Bash dans la génération de noms

Le Bash possède un mécanisme étendu de génération de noms de fichier permettant de décrire des noms de fichiers respectant certains motifs. Sa syntaxe diffère de la

syntaxe des expressions régulières utilisées par *sed* et *awk*. Le tableau suivant résume cette syntaxe.

**(motif)*	0 ou 1 ou plusieurs occurrences de motif
+(motif)	1 ou plusieurs occurrences de motif
?(motif)	0 ou 1 occurrence de motif
@(motif1|motif2)	motif1 ou motif2
!(motif)	tout sauf motif

Ce mécanisme peut être utilisé dans un test, ce qui le rend très pratique en programmation. Les exemples suivants illustrent cette possibilité :

Exemples

1) Supprimer tous les fichiers sauf les fichiers ayant l'extension *.c* ou *.h* et les fichiers dont les noms commencent par *Makefile* ou *makefile* ou *README* :

```
rm !(*.c|*.h|[Mm]akefile*|README*)
```

2) Dans un script, prendre une décision sur le nom de fichier. Si le nom de fichier comporte une extension *.c* ou *.h*, faire ceci, sinon, faire cela :

```
#!/bin/bash
for filename in * ; do
   if [ $filename   @(*.c|*.h|[Mm]akefile*|README*) ]
   then
   fi
done
```

Attention : Ne pas oubliez d'inclure soit dans le script soit dans le fichier *~/.bashscript* l'option *extglob* à l'aide de la commande *shopt -s extglob*.

3) Si la variable TERM contient vt100 ou vt200 ou vtxxx, ou xterm, faire

```
if [ $TERM   @(vt+([0-9])|xterm) ]; then
```

e) Le quoting (neutralisation)

Nous avons vu au paragraphe 7.8 que les caractères spéciaux du shell peuvent être neutralisés de plusieurs façons différentes. Ce problème est encore plus souvent rencontré dans un shell script qu'en ligne de commande interactive, et la question est : « Quels sont les caractères spéciaux neutralisés dans une chaîne placée entre doubles apostrophes (doubles quotes) ? »

Maintenant que tous les mécanismes d'évaluation ont été présentés, nous pouvons énoncer une règle très simple à mémoriser :

Dans une chaîne entre doubles quotes, tous les caractères spéciaux sont neutralisés **sauf le caractère $ sous toutes ses formes** :

Évaluation de variable	**`$TERM, ${TERM:-vt100}`**
Évaluation de commande	**`$(cmd)`**, et donc aussi **`` `cmd` ``**
Évaluation arithmétique	**`$((LINES*COLUMNS))`**

Nous n'en dirons pas plus sur la programmation en Bash. Toute la description de la programmation décrite précédemment s'applique pour l'écriture de scripts en Bash.

8.3 EXERCICES

Exercice 8.3.1

Écrire un shell script qui écrit sur sa sortie standard les messages suivants :
mon nom est xxx
je suis appele avec yyy arguments
qui sont: 111 222 333 444
(xxx sera remplacé par le nom sous lequel ce shell script aura été invoqué, yyy par le nombre d'arguments et 111, 222, etc. par les arguments en question). Quand ce script fonctionnera correctement, invoquez le avec les cinq arguments :

Bienvenue dans le monde Linux
puis avec un seul argument contenant la chaîne de caractères : Bienvenue dans le monde Linux

Exercice 8.3.2

Écrire un shell script démontrant que le shell fils hérite de son père, mais que le père n'hérite pas de son fils.

Exercice 8.3.3

En utilisant exclusivement les commandes *`cd`* et *`echo`*, écrire le shell script "*`recurls`*" réalisant la même fonction que la commande *`ls -R`*. C'est à dire que la commande *`recurls rep1`* devra lister les noms de tous les fichiers et répertoires situés sous le répertoire rep1, y compris les sous-répertoires et les fichiers qu'ils contiennent. Une solution très simple consiste à rendre le script recurls récursif. (un shell script est récursif s'il s'invoque lui-même).

Exercice 8.3.4

Écrivez le script "*`rename`*" permettant de renommer un ensemble de fichiers. Par exemple *`rename '.c' '.bak'`* aura pour effet de renommer tous les fichiers d'extension *`.c`* en *`.bak`*. les fichiers f1.c et f2.c deviennent f1.bak et f2.bak. Utilisez la commande *`basename`*.

Exercice 8.3.5

Et si `vi` gardait une copie de secours ?

`vi` est un peu délicat à maîtriser au début, et si on sort par `:wq` après avoir fait une gaffe, tout est perdu. Ecrivez donc un petit script en Bash qui crée une copie de secours. Appelons le `sv`i, pour Safe VI. La commande `svi fich1` devra invoquer `vi fich1`, mais laisser derrière elle un fichier fich1.bak contenant la version d'origine de fich1.

Pensez aux deux cas suivants :

Si je dis : `svi tralala` et que tralala n'existe pas?

Si je dis : `svi fich1.bak`, que se passe-t-il?

Exercice 8.3.6

L'espace disque est précieux ! Une idée pour économiser : Ecrire un script qui recherche dans toute mon arborescence tous les fichiers qui n'ont pas été accédés depuis un temps T et dont la taille est supérieure a MIN, et les comprimer par l'utilitaire gzip. T et MIN sont des constantes définies au début du script par des valeurs judicieusement choisies. Au fait, a quoi sert MIN ?

Un tel script pourrait être lancé une fois par semaine.

Exercice 8.3.7

Écrire un script dont le nom est process permettant de copier dans un tableau la liste des processus de l'utilisateur exécutant ce script, puis afficher le nom de chaque processus.

Chapitre **9**

Impression

Sous Linux, l'accès aux imprimantes ne se fait qu'au travers d'un **spooler d'imprimante.** Un spooler est un gestionnaire de file d'attente, ou « queue », dans laquelle sont placés les travaux d'impression. A chaque imprimante est associée une file d'attente. Linux envoie ces travaux à l'imprimante concernée dans l'ordre d'arrivée. Cette façon de faire permet de résoudre les conflits d'accès au cas où plusieurs utilisateurs tentent d'accéder à la même imprimante au même moment. Le processus d'impression proprement dit s'effectue en arrière-plan, ce qui permet à l'utilisateur de reprendre la main immédiatement, sans avoir à attendre la fin de son impression. Nous n'entrerons pas dans les détails de la création d'une file d'attente (ceci est du ressort de l'administrateur), mais nous présenterons les commandes permettant à l'utilisateur de manipuler une file d'attente d'impression : envoyer un ou plusieurs travaux (*lpr*), consulter l'état de la file d'attente (*lpq*), supprimer un ou plusieurs travaux de la file d'attente (*lprm*). Un travail d'impression peut être un fichier ou la sortie standard d'une commande.

9.1 IMPRESSION

9.1.1 Imprimante par défaut

La commande *lpr* permet d'envoyer un travail d'impression dans une file d'attente. Utilisée sans aucune option, *lpr* utilise l'imprimante par défaut :

```
xstra> lpr file1
xstra> ls -l /etc | lpr
xstra> lpr file1 file2 file3
```

9.1.2 Choix de l'imprimante

Il est possible que plusieurs imprimantes soient connectées au système. Dans ce cas, il est nécessaire de désigner l'imprimante de destination par l'option *PNom* où *Nom* spécifie le nom de l'imprimante.

```
xstra> lpr  Pbigbrother file1 file2
```

Cette commande envoie *file1* et *file2* à l'imprimante nommée *bigbrother*. Le nom des imprimantes est déterminé par l'administrateur du système. La variable d'environnement *PRINTER*, si elle existe, est utilisée par *lpr*, *lpq* et *lprm* pour désigner l'imprimante destination.

9.1.3 Options

La commande *lpr* accepte d'autres options dont les plus courantes sont :

- `#num` imprimer en *num* exemplaires.
- `m` Mail : le spooler notifie la fin du travail d'impression en envoyant un mail à l'utilisateur.
- `p` Print : procède à une mise en forme identique à la commande *pr* (voir plus loin).
- `r` Remove : supprime le fichier après l'avoir imprimé (pratique si le fichier à imprimer est temporaire).
- `s` Symbolique : plutôt que de copier le fichier à imprimer dans le répertoire du spooler, *lpr* crée dans ce répertoire un lien symbolique sur le fichier à imprimer. Utile si le fichier à imprimer est très grand. Attention : ne pas modifier ou supprimer le fichier avant la fin de l'impression

9.2 ÉTAT DU SPOOLER D'IMPRIMANTE

La commande *lpq option [travaux] [utilisateur]* indique l'état du spooler d'imprimante. Sans option ni argument, indique l'état de tous les travaux soumis par l'utilisateur. Cette commande permet de retrouver l'identification de chaque travail, identificateur utilisable dans *lprm*. Les options suivantes sont acceptées par *lpq* :

- `l` Long : donne plus d'information sur chaque travail.
- `Pnom` Printer : spécifie une imprimante particulière (`nom`). `lpq` utilise aussi la variable d'environnement `PRINTER`.
- `a` Liste l'ensemble des imprimantes disponibles et leur état.

Exemple

```
xstra> lpq -a | grep Printer
Printer: laser2@vivaldi (dest laser2@ljt.u strasbg.fr)
xstra>
```

9.3 SUPPRESSION D'UN TRAVAIL

Il est parfois nécessaire de supprimer un travail déja placé dans une file d'attente. Cette fonction est réalisée à l'aide de la commande *lprm job* où *job* est l'identification du travail d'impression dans le spooler fournie par *lpq*.

Exemple

```
xstra> lpr f*
xstra> lpq
Rank  Owner  Job  Files         Total Size
1st   colin  26   f1, f2, f3    5228 bytes
xstra> lprm 26
§ Suppression de l'impression de f1, f2, f3
```

Les options suivantes sont acceptées par *lprm* :

lprm	supprime tous les travaux d'impression appartenant à l'utilisateur. Souvent utilisé.
PNom	Printer : spécifie une imprimante particulière (Nom). lprm utilise aussi la variable d'environnement PRINTER.

9.4 MISE EN FORME

La commande *pr* peut être associée à *lpr* pour une mise en page du texte. Par défaut, *pr* produit des pages de 66 lignes précédées d'une en-tête de 5 lignes comportant la date, le nom du fichier imprimé et le numéro de page. *pr* peut aussi paginer le texte en plusieurs colonnes. La commande usuelle est :

```
xstra> pr file1 | lpr  § première solution
xstra> lpr -p file1    § solution équivalente
```

9.5 IMPRESSION POSTSCRIPT

Pour imprimer sur une imprimante PostScript récente, la commande *lpr* convient dans tous les cas : l'imprimante détecte elle-même si le document est PostScript ou ASCII et l'imprime correctement. Si l'imprimante n'accepte que le Postscript, la commande *lpr* ne peut être utilisée que si le document à imprimer est déja au format PostScript. Pour imprimer de l'ASCII, la commande *a2ps* remplacera *lpr*.

```
xstra> ls -l /etc | a2ps
[ stdin (plain): 3 pages on 2 sheets]
[ Total: 3 pages on 2 sheets] sent to the default printer
[ 1 line wrapped]
```

La commande *a2ps* (Ascii to PostScript) de GNU fait beaucoup plus que ne l'indique son nom. C'est un filtre très général de production de document PostScript

à partir de nombreux formats d'entrée, et qui comporte de nombreuses possibilités de mise en page. En pratique, avec une imprimante PostScript, il est préférable de n'utiliser que *a2ps*.

Les options les plus courantes de *a2ps* sont :

`2`	2 pages par feuille en mode paysage (défaut)
`r`	1 page par feuille en mode paysage
`1`	1 page par feuille en mode portrait
`3`	3 pages par feuille en mode portrait
`P Nom`	Printer : spécifie une imprimante particulière (`Nom`).

L'option `P` reconnaît trois noms d'imprimantes spéciales :

`void`	désigne */dev/null* et permet de savoir combien de pages seront imprimées (ou de tester la commande).
`display`	désigne Ghostview, le visualiseur PostScript. Si Ghostview n'est pas installé, a2ps produira une erreur.
`file`	désigne un fichier régulier (*fich.c* sera imprimé dans le fichier *fich.ps*).

9.6 EXERCICES

Exercice 9.5.1

Le répertoire courant contient, entre autres, les fichiers : bibal.h bibal.c et README.
Pour imprimer ces trois fichiers en deux exemplaires, deux possibilités existent : imprimer chaque fichier en deux exemplaires ou imprimer en deux exemplaires l'ensemble des trois fichiers.
Quelles sont les commandes correspondantes ?

Exercice 9.5.2

Comment savoir combien de feuilles serait nécessaire pour imprimer, à raison de deux ou trois pages par feuille, le résultat de la commande « `ls  l /usr/bin` » sur une imprimante PostScript ?

Chapitre **10**

Gestion de l'espace disque

10.1 « FILE SYSTEM »

10.1.1 Organisation des « file systems »

L'**organisation du système de fichiers** est une organisation logique basée sur une organisation physique constituée d'un ou plusieurs disques. Chacun est divisé en une ou plusieurs partitions physiques (P1, P2, etc.). L'espace disque physique disponible est ainsi découpé en disques logiques. Initialiser un disque logique consiste à créer sur ce disque un système de fichiers appelé ***file system***.

Sous Unix/Linux, l'utilisateur ne voit pas les disques physiques, pas plus que les partitions. Il n'y a pas de lettres d'unités. L'arborescence de fichiers est construite au démarrage du système à partir de plusieurs *file systems*. Un *file system* particulier, le **root file system** contient la racine de l'arborescence, le noyau, les fichiers systèmes, et des répertoires vides (*/tmp, /usr,…*) sur lesquels seront greffées des arborescences de fichiers se trouvant dans d'autres *file systems*. (Fig. 10.1).

Lors du démarrage d'une machine, après recherche du **disque système**, chaque ***file system*** est rattaché, à l'aide de la commande `mount`, à un répertoire existant appelé ***point de montage*** (***mount point***). La racine de ce *file system* prend alors pour nom celui du point de montage, et l'arborescence de fichier qu'il contient est alors totalement intégrée à l'arborescence de fichiers du système.

La structure d'un *file system* est globalement la même dans toutes les versions d'Unix/Linux : chaque *file system* est composé d'un grand nombre de blocs contigus. La structure de base d'un *file system* est détaillée à la figure 10.2.

- Le bloc 0 contient le boot et l'identification du disque.
- Le bloc 1, appelé aussi superbloc, contient des informations sur le *file system* :

- la date de dernière mise à jour du *file system*,
- le nombre de fichiers qu'il peut contenir,
- la taille du *file system*,
- un pointeur sur la liste des blocs libres.

• Les blocs 2 à k contiennent les inodes.

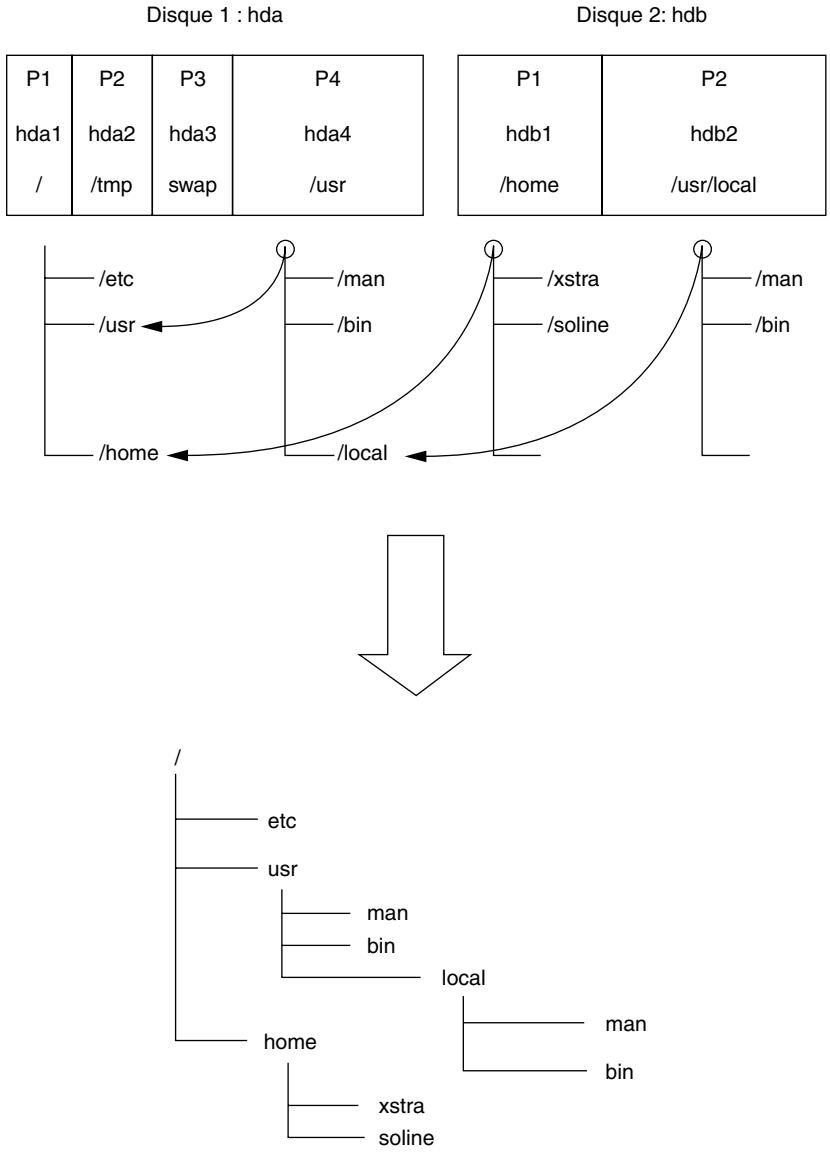


FIGURE 10.1. LES PARTITIONS ET L'ARBORESCENCE DES FICHIERS.

Bloc	*Contenu*
0	Boot bloc
1	Super bloc
2 … … k	Table des inodes
k + 1 … … …	Blocs de données

FIGURE 10.2. STRUCTURES DE BASE D'UN *FILE SYSTEM*.

Sous Unix/Linux, un fichier est une suite d'octets (8 bits) non structurée (byte stream). Toutes les informations concernant un fichier sont regroupées dans une structure de données, appelée **inode**, décrivant la taille du fichier, son emplacement sur disque, son propriétaire, etc. (voir le paragraphe 10.1.2). Un fichier n'est pas repéré par son nom mais par le numéro (index) de l'inode qui le caractérise. Cet index permet de retrouver l'inode dans la table des inodes. C'est donc **l'inode qui caractérise le fichier**.

Le nombre d'inodes, et donc le nombre de blocs qui est réservé à l'enregistrement des inodes, est déterminé à la création du *file system*. Le nombre maximum de fichiers par *file system* est égal au nombre d'inodes alloués à l'initialisation.

- À partir du bloc k+1 sont stockées les données : fichiers et répertoires.

10.1.2 Les inodes

Chaque inode est repéré par un numéro (index) variant de 1 à n et permettant de retrouver son emplacement dans la table des inodes. Aucun fichier n'utilise le numéro 1. Le numéro 2 représente la racine du *file system*. Les informations contenues dans un inode sont :

- le type de fichier (répertoire, fichier ordinaire,…),
- les 12 bits de permissions : suid, sgid, t, rwx rwx rwx
- le nombre de liens,
- le numéro de propriétaire (UID),

- le numéro de groupe (GID),
- la taille du fichier en octets,
- des pointeurs vers les blocs de données correspondants,
- les date et heure du dernier accès,
- les date et heure de la dernière modification,
- les date et heure de la dernière modification de l'inode.

10.1.3 Le répertoire

Le répertoire est un fichier contenant des couples (index, nom_de_fichier) où nom_de_fichier est un nom de fichier ou de répertoire. A sa création, le répertoire contient deux couples dont les noms sont « `.` », le répertoire lui-même, et « `..` », le répertoire père.

Les noms peuvent comporter jusqu'à 255 caractères.

10.1.4 Accès au disque logique

À chaque disque logique est associé un fichier spécial dans le répertoire `/dev` et qui permet d'accéder au disque. Les noms des fichiers spéciaux associés aux disques et partitions sont les suivants :

```
/dev/hda : Hard Disque A : le premier disque IDE
/dev/hdb : Hard Disque B : le deuxième disque IDE
/dev/hda1 : la première partition du disque hda
/dev/hdb3 : la troisième partition du disque hdb
/dev/cdrom : un lien symbolique sur /dev/hdc si votre troi
sième disque IDE est un lecteur de CD ROM
```

10.1.5 La zone swap

Tout système Linux comporte au moins une partition spéciale appelée **partition swap**. Pour des raisons de performances, cette partition n'a pas de structure de *file system*. Elle est utilisée par Linux pour le gestion de la mémoire virtuelle : à un instant donné, la mémoire centrale (RAM) ne contient pas la totalité de la mémoire nécessaire aux processus en cours. Le mécanisme de mémoire virtuelle ne place en RAM que ce qui est indispensable à cet instant, et effectue un va-et-vient entre la RAM et la partition swap.

10.1.6 Commandes ln et mv

Nous l'avons dit précédemment : un fichier, en tant qu'espace disque, est repéré par son index et non par son nom. L'index est le numéro de l'inode décrivant entièrement le fichier sur disque. Un répertoire est un fichier de type particulier contenant des couples (index, nom_de_fichier).

Il est donc possible d'associer deux noms différents au même index. Cette opération est effectuée par la commande `ln` (LiNk) :

ln fichier_existant nouveau_nom

Le même fichier, au sens espace disque, peut alors être accédé par les deux noms : *fichier_existant* et *nouveau_nom*.

Exemples

Dans les exemples suivants, l'option *i* de *ls* permet de visualiser l'index en première colonne :

```
xstra> ls  ilg
6629  rw  r       1 xstra staff 97 Apr 16 17:36 ficha
xstra> cp ficha essai    § copie de ficha sur essai
xstra> ln ficha fichb    § lien entre ficha et fichb
xstra> ls  ilg
6630  rw  r       1 xstra  staff 97 Apr 16 17:42 essai
6629  rw  r       2 xstra  staff 97 Apr 16 17:36 ficha
6629  rw  r       2 xstra  staff 97 Apr 16 17:36 fichb
```

ficha et *fichb* portent le même index alors que *essai* porte un nouvel index. Le fichier *fichb* n'occupe pas de place supplémentaire sur le disque, contrairement au fichier *essai*. *ficha* et *fichb* sont deux noms différents du même fichier disque. Attention : les permissions sont décrites dans l'inode et ne sont donc pas associées à un nom de fichier :

```
xstra> chmod 700 ficha     § change aussi les
                           § permissions de fichb
xstra> ls  ilg
6630  rw  r       1 xstra staff 97 Apr 16 17:42 essai
6629  rwx         2 xstra staff 97 Apr 16 17:36 ficha
6629  rwx         2 xstra staff 97 Apr 16 17:36 fichb
xstra>
```

La commande *rm* (remove) supprime le nom et décrémente le nombre de liens contenus dans l'inode. Le fichier est physiquement supprimé si et seulement si le nombre de liens passe à zéro.

Exemple

```
xstra> rm ficha            § ne supprimera pas fichb
xstra> ls  ilg
6630  rw  r       1 xstra staff 97 Apr 16 17:42 essai
6629  rwx         1 xstra staff 97 Apr 16 17:36 fichb
```

Dans la commande *ln fichier existant nouveau nom*, *nouveau nom* peut se trouver dans un autre répertoire du même *file system*, par exemple :

```
xstra> mkdir test
xstra> ln ficha test/fichierb
```

La commande *mv* (move) remplace le nom du fichier par un nouveau nom :

```
mv fichier_existant nouveau_nom
```

L'inode précédemment repéré par le *nom fichier existant* sera désormais repéré par le nom *nouveau nom*.

```
xstra> mv ficha fichun
xstra> ls  ilg
6630  rw  r       1 xstra staff 97 Apr 16 17:42 essai
6629  rwx         2 xstra staff 97 Apr 16 17:36 fichb
6629  rwx         2 xstra staff 97 Apr 16 17:36 fichun
```

Dans la commande *mv fichier existant nouveau nom*, *nouveau nom* peut se trouver dans un autre répertoire du même *file system*.

Nous avons décrit les commandes *mv* et *ln* telles qu'elles fonctionnent à l'intérieur d'un *file system*. La numérotation des inodes est interne à chaque *file system*. De ce fait, ces deux commandes ne franchissent pas les limites d'un *file system* : deux fichiers différents appartenant à deux *file system* différents peuvent porter le même index (le même numéro d'inode). La logique décrite précédemment est alors inapplicable.

Exemple

```
xstra> mv fichb /tmp/fichierb §/tmp est un autre file system
xstra> ls  ilg
6630  rw r      1 xstra staff  97 Apr 16 17:42 essai
6629  rwx       1 xstra staff  97 Apr 16 17:36 fichun
xstra> ls  ilg /tmp
16    rwx       1 xstra staff   97 Apr 16 17:36 fichierb
```

Nous voyons que :

- il n'y a plus qu'un seul lien sur *fichun* (donc */tmp/fichierb* n'est plus un lien sur *fichun*),
- *fichun* et */tmp/fichierb* ne portent pas le même index.

En fait la commande :

```
mv fichb /tmp/fichierb
```

est exécutée par Linux sous la forme :

```
cp fichb /tmp/fichierb && rm fichb
```

Si la commande *ln* est appliquée à deux fichiers n'étant pas dans le même *file system* on obtiendrait :

```
xstra> ln ficha /tmp/fichb
ln: cannot create hard link '/tmp/fichb' to 'ficha' :
Invalid cross device link
```

La commande `ln` ne franchit pas la limite du *file system*. Il faut dans ce cas utiliser un lien symbolique (option `s` de `ln`) :

```
ln  s fichier_existant nouveau_nom
```

Le fichier `nouveau nom` est créé (nouvel inode), mais n'occupe que peu de place sur disque : il ne contient que la référence à `fichier existant`.

10.2 GESTION DE L'OCCUPATION DISQUE

Le répertoire dans lequel vous vous trouvez après connexion s'appelle le répertoire d'accueil ($HOME). Ce répertoire est dans un *file system* dont la capacité en octets est fixée. La création de fichiers va au fur et à mesure remplir le *file system* jusqu'à ce que le message **file system full** apparaisse. Un *file system* peut être plein pour deux raisons (voir figure 10.2) :

- soit il n'y a plus de place pour un nouvel inode (table des inodes)
- soit il n'y a plus de bloc de données disponible (blocs de données)

Dans le premier cas, le *file system* peut être rempli avec un petit nombre de grands fichiers. Dans le deuxième cas, il peut être rempli avec un très grand nombre de fichiers de taille réduite (ou même nulle). Dans un cas comme dans l'autre, il est impossible de continuer à travailler, et il faudra faire de la place. Il est donc nécessaire de gérer l'espace disque, en veillant à éliminer les fichiers devenus inutiles (*core*, fichiers temporaires,….) et en archivant les fichiers qui ne sont plus utilisés afin de pouvoir les supprimer, ou tout au moins de les conserver sous forme d'une archive compressée. En effet, une archive est **un seul** fichier contenant toute une arborescence, ce qui économise de l'espace dans la table des inodes. Si de plus elle est compressée, il y a aussi économie de blocs de données.

10.2.1 Occupation disque par file system : commande df

La commande `df` indique le nombre de blocs de 1 Ko disponibles par *file system*.

Exemple

```
xstra> df  k
Filesystem  1k blocks    Used  Available  Use% Mounted on
/dev/hda1     8064272 5155608    2499012   68% /
/dev/hda3    68065912  693752   63914572    2% /home
/dev/hdb1    41484300   32828   39344148    1% /data1
xstra>
```

Dans cet exemple, trois *file system* ont été créés par l'administrateur (`/home`, `/data1`). Afin de connaître le *file system* dans lequel l'utilisateur se trouve, il suffit d'utiliser la commande `pwd` qui indiquera le chemin d'accès du répertoire de travail et dont le début correspond au point de montage d'un *file system*.

10.2.2 Occupation disque : commande du

La commande *du noms* affiche le nombre de blocs (1 Ko) occupés par les fichiers et répertoires spécifiés par *noms*. Cette commande est récursive : un nom de répertoire désigne toute l'arborescence sous ce répertoire. Si *noms* n'est pas indiqué, le répertoire de travail (.) est pris par défaut.

L'option *s* n'indique que le nombre total de blocs pour chaque *noms*.

L'option *a* (all) liste l'occupation disque de chaque fichier ou sous-répertoire.

Exemples

```
xstra> du  s toto
8  toto
xstra> du  s *
8  toto
4  repertoire1
20  essai
xstra> du  a *
8  toto
4  repertoire1
2  repertoire1/toto
20 essai
xstra>
```

10.2.3 Comment partitionner le disque ?

Il y a beaucoup de réponses différentes à cette simple question, selon l'utilisation du système. La logique est cependant très simple : l'espace disque attribué à un *file system* ne peut pas être utilisé pour une autre *file system*.

- Avantage : si un *file system* est plein suite à une erreur de manipulation ou d'administration, les autres *file system* ne sont pas affectés et le système peut continuer à fonctionner normalement.
- Inconvénient : si un *file system* est plein sans que cela soit le résultat d'une erreur quelconque, il n'est pas possible de l'agrandir en prélevant de l'espace ailleurs.

Quelques règles simples peuvent aider un utilisateur débutant amené à administrer son ordinateur (cet ouvrage ne s'adresse pas aux administrateurs chevronnés) :

- Toujours créer une partition de swap dont la taille est égale à deux fois la taille de la mémoire RAM présente ou prévue dans un avenir proche.
- Pour un système Linux personnel (un utilisateur habituel qui passe root de temps à autre pour administrer le système), la solution la plus simple est d'attribuer tout ce qui reste à une seule partition qui contiendra le root *file system* et donc l'ensemble de l'arborescence en un seul *file system*.
- Pour un système Linux utilisé par un petit groupe, il serait bon, pour simplifier l'administration, de partitionner un peu plus. La quantité de disque utilisée par

chaque utilisateur dépend beaucoup du type de travail effectué par chacun, et il n'est pas possible de donner une règle générale pour */home*. Un partitionnement possible serait par exemple :

une partition swap égale à 2 fois la taille de la RAM
un *file system* pour `/tmp` d'au moins 200 Mo
un *file system* pour `/var` d'au moins 400 Mo
un *file system* pour `/home` (au moins 200 Mo par utilisateur)
tout l'espace restant pour `/` (au moins 3 Go)

10.3 EXERCICES

Exercice 10.3.1

Créez un fichier texte et un lien dur sur ce fichier dans le même répertoire.
Vérifiez que les deux noms correspondent au même inode.
Changez les permissions de l'un et vérifiez que les permissions de l'autre ont suivi.
Modifiez le contenu de l'un et relisez le contenu de l'autre.
Supprimez l'un, que devient l'autre ? Essayez de créer un nouveau lien entre un de ces noms et un nouveau nom dans */tmp*. Expliquez.

Exercice 10.3.2

L'option *-l* de la commande *ls* permet de connaître le nombre de liens sur un fichier régulier (deuxième champ). Curieusement, ce champ n'est pas égal à 1 pour les répertoires, alors qu'il n'est pas possible de créer de liens durs sur des répertoires. Expliquez la valeur de ce nombre sur votre répertoire d'accueil.

Exercice 10.3.3

Créez un fichier texte et un lien symbolique sur ce fichier dans le même répertoire. Quelles sont les permissions du lien symbolique ? Combien de liens possède le fichier texte ? Avec la commande *more*, affichez le contenu du lien symbolique. Effacez le fichier texte. Que devient le lien symbolique ?

Exercice 10.3.4

Le disque est plein de vide ! Écrire un petit script affichant sur la sortie standard la taille cumulée de tous les fichiers réguliers situés sous le répertoire courant, et l'espace disque total occupé par le répertoire courant et tout ce qu'il contient.

Chapitre **11**

Sauvegarde

Afin de gérer l'espace disque, de préserver les programmes et les données, il est nécessaire d'archiver les informations sur un autre support magnétique que le disque. La perte de données est toujours possible : un ordre d'effacement trop rapide, un problème physique sur le disque, une panne machine font que la perte de temps liée à la perte de données est très supérieure au temps passé à faire des sauvegardes. Comme toujours dans le monde Unix, il y a plusieurs façons de faire et Linux propose différents utilitaires de sauvegarde : `tar`, `cpio`, `dd`, `dump/restore`. Les commandes `dump` et `restore` sont très adaptées aux sauvegardes faites par l'administrateur. La commande *tar*, la plus utile à l'utilisateur, sera présentée dans ce chapitre. Par contre avec les capacités des disques en forte croissance (la capacité d'un disque standard en 2004 est de 120 Giga), il devient de plus en plus difficile de sauvegarder correctement les données sur des bandes magnétiques (capacité bande limitée, temps de sauvegarde important, coût élevé). Une solution est de réaliser cette sauvegarde sur un autre disque du même ordinateur ou sur un disque d'un ordinateur distant connecté au même réseau. Le coût d'un disque supplémentaire de sauvegarde n'est plus prohibitif et souvent moindre à une solution de sauvegarde basée sur les bandes magnétiques. La commande `rsync`, qui permet de copier de façon intelligente (synchroniser) les données d'un disque sur un autre, encourage ce type de sauvegarde.

11.1 LA COMMANDE DE SAUVEGARDE TAR (TAPE ARCHIVE)

Pour un utilisateur, la commande la plus répandue et la plus utilisée est `tar`. Elle permet de sauvegarder toute une arborescence de fichiers dans une **archive :** cette

archive contient tous les fichiers et répertoires, mais aussi les uid, gid, permissions et dates de chaque fichier et répertoire archivé. Ceci permettra de restaurer tout ou partie de cette archive en cas de besoin, avec les propriétés et permissions correctes. Dans ce qui suit, le terme **archive** désigne un support physique tel que bande magnétique, cartouche, disquette **ou fichier régulier**. Il est possible d'enregistrer plusieurs archives sur un même support magnétique.

Les trois opérations de base sont les suivantes :

- Créer une archive : `tar c`
- Extraire d'une archive : `tar x`
- Lister le contenu d'une archive : `tar t`

L'utilitaire `tar` de GNU disponible sous Linux accepte trois syntaxes différentes :

`tar c`	ancien style Unix
`tar -c`	options courtes
`tar --create`	options longues

Dans ce chapitre, nous ne présenterons que les options courtes, appelées short options dans la documentation. La documentation du GNU `tar` obtenue par la commande `info tar` est très complète et très didactique (en anglais).

La syntaxe de la commande `tar` est la suivante :

```
tar options noms
```

Une `option` est une lettre précédée d'un signe – et suivie d'éventuels paramètres. Comme toujours, on peut combiner plusieurs options (`-x -v -z` peut s'écrire `-xvz`). Les autres arguments de la commande sont des noms de fichiers ou de répertoires. Dans tous les cas, un nom de répertoire désigne ce répertoire et tous les fichiers et les sous–répertoires qu'il contient récursivement.

–c ***create*** : créer une nouvelle archive. L'écriture des fichiers commence au début de l'archive et non pas après le dernier fichier archivé,

–r ***append*** : les `noms` sont stockés à la fin de l'archive même s'ils existent déjà sur cette archive,

–u ***update*** : additionne `noms` à l'archive s'ils n'y sont pas déjà ou si la date de la dernière modification est plus récente que celle de la version contenue sur l'archive,

–t ***list*** : les `noms` sont affichés autant de fois qu'ils apparaissent sur l'archive. Si aucun `noms` n'est indiqué c'est l'ensemble du contenu de l'archive qui est affiché,

–x ***extract*** : permet d'extraire les fichiers *noms* de l'archive. Si les *noms* sont des répertoires, tous les fichiers et sous–répertoires sont restaurés. Si aucun *noms* n'est indiqué, l'ensemble de l'archive est restauré. Si plusieurs fichiers sauvegardés sur l'archive portent un même nom, c'est le dernier qui sera restauré.

Les options suivantes peuvent être utilisées en complément des options précédentes :

–v ***verbose*** : la commande *tar* exécute normalement son travail silencieusement. L'option *–v* affiche le nom de tous les fichiers et répertoires traités,

–f ***file*** : permet de préciser le nom du fichier archive. Ce nom peut désigner un fichier spécial, par exemple */dev/fdo* pour le floppy disque, ou un fichier régulier, par exemple *~/backups/monarchive.tar*. Si le nom de fichier est le caractère –, *tar* écrit sur la sortie standard (options *–r*, *–c*) ou lit l'entrée standard (option *–x*, *–t*). Ceci permet d'utiliser la commande *tar* dans un tube.

–p ***preserve*** : associée à l'option *–x*, permet de restaurer les permissions de fichiers et répertoires à la valeur qu'elles avaient au moment de la sauvegarde. Sans cette option, *tar* soustrait la valeur de *umask* aux permissions inscrites dans l'archive pour obtenir les permissions du fichier restauré.

–z ***zip*** : passe l'archive par l'utilitaire de compression *gzip*. Cette option s'applique aussi bien à l'archivage (*–c*) qu'à l'extraction (*–x*, *–t*)

Exemples

1) Sauvegarde

L'exemple suivant permet de sauvegarder tout le répertoire *alice* sur l'archive de nom */dev/fd0* (option *–c*) et affiche les fichiers sauvegardés (option *–v*).

```
xstra> cd /home
xstra> tar -cv -f /dev/fd0 alice
alice/
alice/.bash_profile
alice/.bashrc
alice/bidon
alice/bin/
§ et bien d'autres noms seront encore listés ici.
```

2) Extraction

Dans cet exemple la commande *tar* permet d'extraire (option *–x*) à partir de l'archive le fichier *bidon*. Ce fichier sera remis sous le répertoire *alice* contenu dans */home*. Affichage des fichiers restaurés.

```
xstra> cd /home
xstra> tar -xv -f /dev/fd0 alice/bidon
alice/bidon
xstra>
```

3) Création d'une archive sur disque

Le fichier */tmp/xstra.tgz* contiendra la sauvegarde compressée du répertoire */home/xstra*.

```
xstra> pwd
/home/xstra
xstra> tar -cvz -f /tmp/xstra.tgz .
§ le point en fin de la ligne précédente désigne le
§ répertoire courant.
```

11.2 LA COMMANDE RSYNC (REMOTE SYNCHRO)

La commande *rsync* permet de synchroniser le contenu de deux fichiers ou de deux répertoires. *rsync* est un programme très similaire à *rcp*, mais possède plus d'options et surtout ne transfert que les différences pour les fichiers modifiés. Ceci permet de limiter l'utilisation de la bande passante réseau entre la machine à sauvegarder et l'hôte de la sauvegarde distante. En fait *rsync* peut être assimilée à une commande permettant de réaliser un miroir entre deux disques.

La syntaxe de la commande *rsync* est la suivante :

rsync [options] source destination

Une *option* est une lettre précédée d'un signe et suivie d'éventuels paramètres. Comme toujours, on peut combiner plusieurs options *(a v z* peut s'écrire *avz*). *L'option* peut aussi être un mot précédé par deux tirets « », suivi éventuellement par un paramètre. Les autres arguments de la commande sont la source et la destination des fichiers et répertoires à copier. La syntaxe de *source* et *destination* qui permet de définir, le nom de l'utilisateur (*utilisateur*), le nom de l'ordinateur (*nom ordinateur*) et du fichier ou répertoire sur l'ordinateur (*nom de fichier*) est : *utilisateur@nom ordinateur:nom de fichier*. Le terme *utili sateur* peut être omis s'il est le même sur les deux machines. De plus il n'est pas utile d'indiquer le nom de l'ordinateur sur lequel on exécute la commande. Enfin dans le cas de la *source*, le *nom de fichier* s'il désigne un répertoire et est terminé par le caractère / indique le contenu du répertoire et sans le caractère / comme dernier caractère indique le répertoire lui-même.

Les principales options de *rsync* sont :

a, archive

En fait cette option, la plus utilisée, est l'équivalent de *-rlptgoD*. Elle permet de copier récursivement *source* vers *destination* en préservant au mieux la configuration de *source*.

v (verbose)

La commande *rsync* exécute normalement son travail silencieusement. L'option *v* affiche le nom de tous les fichiers et répertoires traités.

stats

Établit des statistiques à la fin du traitement permettant ainsi d'apprécier le transfert pour vos données.

e, rsh=commande

Permet de choisir un mode de communication autre que celui utilisé par défaut (*rsh*).

n, dry-run

Aucun traitement n'est réalisé par *rsync* mis à part l'affichage de l'action qui aurait dû être fait. Cette option permet de vérifier que la syntaxe de la commande répond à votre attente.

delete

Supprime de la destination les fichiers et répertoires n'existant plus dans la source. Cette option peut être dangereuse. Il est conseillé d'utiliser l'option *n* au préalable, permettant ainsi de contrôler l'action souhaitée.

exclude=MOTIF

Cette option permet d'exclure du transfert les fichiers ou répertoires répondant à la définition du MOTIF. Le MOTIF peut être le nom d'un fichier ou répertoire. Il peut également être plus complexe et désigner un ensemble de fichiers ou répertoires.

z (zip)

Copie en compressant les données. Surtout utile lors d'une connexion faible débit.

Exemples

1) Sauvegarde sur un autre disque d'un autre ordinateur

L'exemple suivant permet de copier les fichiers du répertoire */home/armspach* dans le répertoire */sauvegarde/home* de l'ordinateur dont le nom est *machine backup*. Seuls les fichiers modifiés entre deux commandes successives de *rsync* sont copiés. L'option *-a* indique un archivage récursif, l'option *v* affiche le nom de tous les fichiers et répertoires traités, l'otion *e ssh* permet de choisir un mode sécurisé de transfert des données.

```
xstra> rsync av e ssh /home/armspach \
armspach@machine_backup:/sauvegarde/home
passwd: § l'utilisateur entre son mot de passe
receiving file list ... done
xstra>
```

2) Copie avec exclusion

L'exemple suivant permet de copier les fichiers du répertoire courant en excluant les fichiers avec l'extension *.o*.

```
| xstra> rsync  av   exclude="*.o" . ../save
```

11.3 EXERCICE

Exercice 11.3.1

Vous récupérez par le réseau un fichier dont le nom est *prog.tar.gz*. Que faites-vous ?

Chapitre **12**

Gestion des processus

12.1 NOTIONS THÉORIQUES SUR LES PROCESSUS

Nous avons, tout au long de cet ouvrage parlé des processus sans jamais vraiment les définir.

Nous allons tout d'abord présenter quelques notions théoriques concernant les processus, puis les cinq modes d'exécution d'une commande sous Linux, ainsi que les commandes `ps` et `kill` qui permettent de gérer les processus. Enfin nous présentons le job control.

12.1.1 Processus

Un processus est un programme en cours d'exécution. Les attributs d'un processus appartiennent à ce que l'on appelle son environnement : parmi eux le code, les données temporaires, les données permanentes, les fichiers associés, les variables. L'environnement d'un processus contient aussi les entités que le système lui attribue : les descripteurs, la mémoire allouée, la pile d'exécution du noyau.

12.1.2 Processus père et processus fils

Le processus fils est un processus qui a été créé par un autre processus qui prend le nom de processus père.

12.1.3 Identification d'un processus

Un processus sous Linux est identifié par un numéro unique qui s'appelle le numéro d'identification du processus PID (Process IDentifier) et qui lui est attribué par le système à sa création.

12.1.4 Temps partagé

On appelle **temps partagé** une exploitation dans laquelle plusieurs processus sont simultanément en cours d'exécution. Cette simultanéité n'est qu'apparente. Le noyau Linux va affecter à l'unité centrale le processus en cours dont la priorité est la plus importante. Lorsque ce processus est en attente ou si un processus de priorité plus importante apparaît, le noyau libère l'unité centrale du processus en cours et charge le nouveau processus de haute priorité. De surcroît, afin qu'un processus de haute priorité ne monopolise pas l'unité centrale, le noyau Linux va modifier dans le temps les priorités des processus en attente et en cours afin de permettre à l'ensemble des processus d'être exécuté.

12.1.5 Swapping (va-et-vient)

Le **swapping** consiste en la recopie sur disque d'un processus complet ou d'une partie d'un processus ayant perdu le contrôle de l'unité centrale et ne pouvant plus rester en mémoire centrale. La mémoire centrale ainsi libérée est affectée à un processus plus prioritaire. Il est aisé de comprendre que la performance d'un système dépend beaucoup de la qualité du swapping.

12.1.6 Classification des processus

Il est possible de distinguer deux types de processus : les processus système et les processus utilisateurs.

a) Les processus système (daemons)

Ces processus ne sont sous le contrôle d'aucun terminal et ont comme propriétaire l'administrateur du système ou un uid d'administration. Ils assurent des tâches d'ordre général, parfois disponibles à tous les utilisateurs du système. Ils ne sont d'habitude stoppés qu'à l'arrêt du système d'exploitation. Les plus courants sont :

- `init` : initialise un processus par terminal connecté sur la machine, permettant la connexion des utilisateurs. Ce processus a le numéro 1. C'est le processus parent de tous les interpréteurs de commandes créés par la connexion d'un utilisateur.
- `crond` : permet l'exécution d'un programme en mode cyclique.
- `xinetd` : super démon internet chargé de créer les processus serveurs réseau sur requêtes des clients.

b) Les processus utilisateurs

Ils correspondent à chaque exécution d'un programme par l'utilisateur, le premier d'entre eux étant l'interpréteur de commandes à la connexion. Ces processus appartiennent à l'utilisateur et sont généralement attachés à un terminal.

12.2 EXÉCUTION D'UNE COMMANDE

Les chapitres précédents ont présenté plusieurs modes d'exécution d'une commande. En fait il existe cinq modes d'exécution d'une commande sous Linux :

- Le mode interactif (foreground),
- Le mode en arrière-plan (background), appelé aussi mode asynchrone,
- Le mode différé,
- Le mode batch,
- Le mode cyclique.

12.2.1 Le mode interactif

En mode interactif, mode le plus fréquemment utilisé, la commande est lancée à partir d'un interpréteur de commandes. Pendant l'exécution de la commande, l'utilisateur ne peut pas utiliser le terminal pour lancer une autre commande. Le contrôle n'est restitué à l'utilisateur qu'à la fin de l'exécution de la commande.

À tout moment, il est possible d'interrompre cette commande en utilisant la combinaison de touches *<ctrl c>*, ou de la suspendre à l'aide de *<ctrl z>* (attention, la fonction de ces touches peut être modifiée par la commande *stty*).

12.2.2 Le mode en arrière-plan

Le lancement de commandes en arrière-plan (paragraphe 7.4) permet de rendre immédiatement le contrôle à l'utilisateur. Cette fonctionnalité est intéressante pour des tâches ne nécessitant pas d'interaction entre l'utilisateur et la tâche, comme par exemple la compilation d'un programme.

La commande est lancée suivie du caractère **&**. Son exécution peut être surveillée par les commandes *ps* ou *jobs*.

Exemple

Le fichier *essai* est un exécutable. Son exécution n'interagit pas avec l'utilisateur.

```
xstra> essai &
xstra>
```

L'utilisateur a la main. Il peut continuer à travailler tout en surveillant régulièrement l'exécution de la commande *essai*.

Si l'utilisateur est en Bash et qu'il quitte le shell, la commande en arrière-plan est interrompue automatiquement. Pour éviter ce problème, il faut lancer la commande sous le contrôle de la commande *nohup* ou utiliser la commande interne *disown* que nous décrirons plus tard.

Exemple

```
xstra> nohup essai &
xstra>
```

12.2.3 Le mode différé

L'exécution différée d'une commande est réalisée à l'aide de la commande *at* qui permet de déclencher l'exécution d'une commande à une date fixée.

Exemple

```
xstra> at 20:05 20/09/00  <commande
xstra>
```

demandera le lancement du contenu de *commande* le 20/09/00 à 20 h 05.

L'autorisation d'utilisation de *at* pour un utilisateur est indiquée dans le fichier */etc/at.allow*. Si ce fichier n'existe pas, le fichier */etc/at.deny* est vérifié pour déterminer si l'accès à la commande *at* doit être interdite à l'utilisateur. Si *at.deny* est vide, l'utilisation de *at* est permise pour tous les utilisateurs. Si aucun fichier n'existe, seul l'administrateur (root) a la permission d'utilisation de *at*.

Par défaut *at* lance l'interpréteur de commandes *sh*. De plus le processus ainsi créé ne sera associé à aucun terminal ; les sorties sont soit redirigées soit envoyées à l'utilisateur par *mail*. Il est possible de gérer à l'aide de *atq* (ou *at l*) (liste des processus en cours avec indication du numéro d'identification) et de *atrm* (ou *at d numero at*) (suppression d'un processus) les processus lancés par un utilisateur.

12.2.4 Le mode batch

Le batch permet de placer une commande dans une file d'attente. Le système exécutera toujours la commande placée en tête dans la file d'attente. Ainsi toutes les commandes lancées par *batch* seront exécutées séquentiellement quel que soit l'utilisateur qui a mis la commande dans la file d'attente. La gestion des sorties standard est similaire à celle de la commande *at*. Ce mode est le plus altruiste sur un système Linux très chargé.

12.2.5 Le mode cyclique

L'exécution cyclique d'une tâche est réalisée à l'aide de la commande *crontab*. Le processus *cron* (daemon) scrute un fichier dans lequel sont définies les commandes à exécuter à date fixe. L'autorisation d'utilisation de *crontab* pour un utilisateur est identique à la méthode définie pour l'autorisation d'utilisation de la commande *at*. Les fichiers vérifiés sont */etc/cron.allow* puis */etc/cron.deny*.

L'utilisateur va créer un fichier comprenant les indications de répétition d'exécution de la tâche ainsi que la tâche elle-même. La commande `crontab file` va copier le fichier `file` dans la zone spool du crontab et ainsi le soumettre à une exécution cyclique.

On trouvera dans le répertoire `/var/spool/cron` un fichier au nom de l'utilisateur. Ce fichier contiendra les commandes à exécuter cycliquement.

La commande `crontab e` permet d'éditer directement votre crontab contenue dans `/var/spool/cron`.

Chaque ligne de `file` a six champs. Chaque champ est séparé par un espace ou une tabulation. La signification des champs est la suivante :

Champ 1 : la minute (0-59),

Champ 2 : l'heure (0-23),

Champ 3 : le jour du mois (1-31),

Champ 4 : le mois de l'année (1-12),

Champ 5 : le jour de la semaine (0-6 avec 0 = dimanche),

Champ 6 : la tâche à exécuter.

Chacun des champs peut être soit un astérisque (`*`) signifiant toutes les valeurs, soit une liste d'éléments séparés par des virgules, soit deux nombres séparés par un tiret ( ) indiquant une fourchette inclusive de valeurs.

L'exemple suivant :

```
59 0 * * 1 6    sauvegarde_journaliere
```

permet d'exécuter la commande `sauvegarde journaliere` tous les jours du lundi au samedi à 0 h 59. Attention, l'utilisation de `crontab` n'est possible que si l'administrateur vous le permet. Les sorties standard sont envoyées à l'utilisateur par la messagerie, sauf en cas de redirection.

12.3 LA COMMANDE PS

La commande `ps options` permet d'obtenir des renseignements sur l'état des processus en cours. Par défaut, seuls les processus de la connexion en cours pour l'utilisateur ayant lancé la commande `ps` sont affichés. Une option `u nom utilisateur` permet de sélectionner les processus de cet utilisateur pour l'ensemble de ses connexions. Parmi les nombreuses options de cette commande, les options suivantes sont particulièrement utiles :

-e affiche des renseignements sur tous les processus en cours,

-C affiche des renseignements sur tous les processus d'un certain nom :
`ps C bash`

-f génère, pour chaque processus, le nom de l'utilisateur (UID), le numéro du processus (PID), le numéro du processus père (PPID), l'heure de lancement

du processus (STIME), le nom du terminal (TTY) et le temps d'exécution du processus (TIME).

Exemples

```
xstra> ps
PID         TTY    TIME   CMD
3408        pts/1  0:00   ps
10804       pts/1  0:20    ksh
xstra> ps  ef
UID         PID  PPID  C STIME TTY          TIME CMD
root          1     0  0 May18 ?        00:00:03 init
root        300     1  0 May18 ?        00:00:08 syslogd  m 0
daemon      327     1  0 May18 ?        00:00:00 /usr/sbin/atd
root        343     1  0 May18 ?        00:00:00 crond
root        359     1  0 May18 ?        00:00:03 inetd
root        411     1  0 May18 ?        00:00:00 sendmail
root        444     1  0 May18 ?        00:00:00 httpd
xstra     11736 11731  0 13:45 pts/0    00:00:00 /bin/bash
xstra     11737 11732  0 13:45 pts/1    00:00:00 /bin/bash
xstra     11979  3955  0 16:29 ?        00:00:00 kwm
xstra     11995 11979  0 16:29 ?        00:00:00 kbgndwm
xstra     12021 11979  0 16:30 ?        00:00:02 kfm
xstra     12022 11979  0 16:30 ?        00:00:00 krootwm
xstra     12023 11979  0 16:30 ?        00:00:01 kpanel
xstra     12098 11737  0 16:41 pts/1    00:00:00 man at
xstra     12160 11736  0 17:17 pts/0    00:00:00 ps  ef
```

12.4 LA COMMANDE KILL

Cette commande *kill signal PID* sert essentiellement à arrêter des processus en arrière-plan (background). En effet, pour arrêter le ou les processus liés à la commande en cours d'exécution, il est beaucoup plus simple de taper <ctrl-c>.

L'option *signal* correspond au signal envoyé au processus. Un certain nombre d'événements (erreur, *<ctrl c>*,…) peut être signalé à un processus à l'aide d'un signal. Ce mécanisme permet la communication interprocessus, notamment entre le système d'exploitation et le processus. Les principaux signaux sont :

SIGHUP	(SIGnal Hang UP : fin du shell)
SIGINT	(SIGnal INTerrupt : interruption du programme)
SIGKILL	(SIGnal KILL : tuer le processus)
SIGTERM	(SIGnal TERMinate : terminaison douce)
SIGQUIT	(SIGnal QUIT : terminaison brutale)
SIGSTOP	(SIGnal STOP : stopper le processus)

D'autres signaux existent et sont décrits dans *signal* (7). Par défaut la fonction *kill* envoie au processus le signal SIGTERM lui demandant de s'arrêter. Un processus peut ignorer le signal SIGTERM. Le signal SIGQUIT est plus brutal. Le

signal SIGKILL ne peut être ignoré et permet dans tous les cas d'arrêter le processus (l'arrêt peut alors avoir des conséquences graves). S'il faut tuer un processus, il est donc prudent d'utiliser les commandes suivantes, dans cet ordre :

```
kill pid
kill  QUIT
kill  KILL pid.
```

12.5 LE JOB CONTROL

Le **job** est une ligne de commandes shell. Il est composé d'un ou de plusieurs processus dans le cas d'utilisation du tube (voir le paragraphe 7.3 et 7.4). Chaque job est numéroté de 1 à N par le shell. Ce numéro est interne au shell. Il est plus maniable que le PID. Un job peut se trouver dans trois états :

- **avant-plan** ("foreground"). Le job s'exécute et vous n'avez pas la main en shell.
- **arrière-plan** ("background"). Le job s'exécute et vous avez la main en shell.
- **suspendu** ("suspended"). Le job est en attente, il ne s'exécute pas.

Le passage d'un état du job à un autre est présenté sur la figure 12.1 dans le cas du Bash. Certaines fonctions de création et d'action des jobs ont déjà été présentées dans le chapitre 7. Chaque job peut être référencé en utilisant le préfix % suivi du numéro de job. Nous allons dans ce paragraphe récapituler les différentes fonctions du "job control" et les compléter.

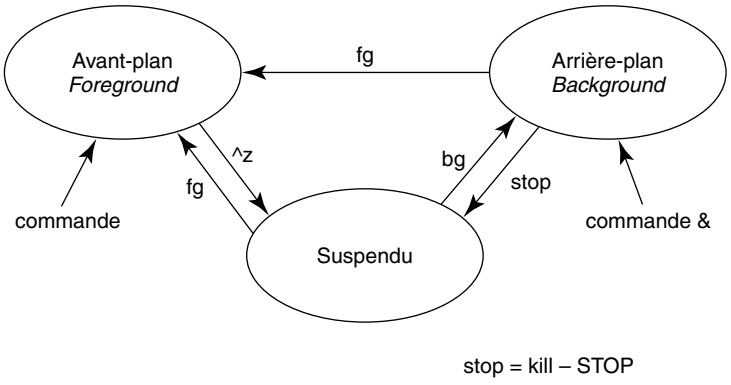


FIGURE 12.1. DIFFÉRENTS ÉTATS D'UN JOB.

12.5.1 Job en arrière-plan

La création d'un job en arrière-plan est réalisée par l'utilisation du lancement d'une commande en "background" (paragraphe 7.4).

Exemple

```
xstra> (find /usr  type d  print | sort >/tmp/toto) &
[1] 2349
xstra> (find /usr  type f  print | sort >/tmp/tutu) &
[2] 3786
xstra>
```

Les deux jobs créés dans cet exemple tournent simultanément. Chacun comporte deux processus. Le shell après chaque commande rend la main et permet l'utilisation du shell en interactif.

La commande *jobs l* permet de lister les jobs en cours.

Exemple

```
xstra> sleep 60&
[1] 2736
xstra> sleep 50&
[2] 2957
xstra> sleep 40&
[3] 3100
xstra> jobs  l
[1]    2736   Running  sleep 60&
[2]    2957   Running  sleep 50&
[3]  + 3100   Running  sleep 40&
xstra>
```

Dans l'exemple précédent, le caractère + indique le job le plus récent, le caractère – indique le deuxième plus récent.

12.5.2 Job suspendu

Une commande peut être suspendue lorsqu'elle est en avant-plan par la touche *<ctrl Z>*.

Lorsqu'elle est en arrière-plan la commande :

kill SIGSTOP %numero job ou

kill STOP %numero job

permet de suspendre la commande associée au numéro de job *numero job*.

La commande *bg %numero job* permet de basculer un job suspendu en job en arrière-plan.

Exemple

```
xstra> stty susp <ctrl Z>
xstra> sleep 100
...
```

```
<ctrl Z>
[1]  2655   Stopped
xstra> jobs  l
[1]   2655   Stopped  sleep 100
[2]   2957   Running  beta&
xstra> kill  STOP %2
[2]  Stopped beta
xstra> bg %2
xstra>
```

12.5.3 Job en avant-plan

Toute commande lancée interactivement en shell est en avant-plan. Un job en arrière-plan ou suspendu peut être mis en avant-plan par la commande *fg %numero job*.

Exemple

```
xstra> jobs  l
[1]   2655   Stopped  sleep 100
[2]   2957   Running  beta&
xstra> fg %1
  § Attente de quelques secondes,
  § la commande sleep 100 est en avant plan
xstra>
```

12.5.4 La commande kill et le job control

L'un des grands avantages du "job control" est de faciliter grandement les possibilités pour tuer un processus. En effet tuer un job est plus pratique que tuer les processus qui le constituent : il est inutile de manipuler des numéros de processus, surtout dans le cas où le job est constitué de plusieurs processus (tube, commandes groupées).

Exemple

```
xstra> (sleep 55; sleep 66)&
[1] 11877
xstra> jobs  l
[1]+ 11877 Running ( sleep 55; sleep 66 ) &
xstra> kill %1
xstra> jobs  l
[1]+ 11877 Terminated ( sleep 55; sleep 66 )
xstra>
```

12.5.5 Sortie de session et job control

Lorsqu'on quitte une session tous les processus attachés à la session seront interrompus. En effet, le Bash envoie à chaque job le signal SIGHUP indiquant la fin de

session et par conséquent l'arrêt du job. Pour éviter qu'un processus en arrière-plan soit ainsi interrompu en quittant sa session il faut :

- soit utiliser la commande *nohup* lors du lancement de la commande (voir paragraphe 7.4),
- soit désactiver l'envoi du signal SIGHUP par le Bash en utilisant la commande *disown -h %numero_job*. Cette commande n'est disponible que pour la version 2 du Bash.

Exemple

```
xstra> jobs -l
[1]   2655   Running  sleep 100
[2]   2957   Running  beta&
xstra> disown -h %2
xstra> # En quittant la session, la commande beta n'est
       # pas interrompue.
```

12.6 EXERCICES

Exercice 12.6.1

Écrivez un shell script qui cherche dans votre arborescence personnelle tous les fichiers de noms *core*, **.tmp*, *a.out* qui n'ont pas été accédés depuis plus de 3 jours, les supprime et vous envoie la liste par mail. Ce script sera exécuté tous les jours à trois heures du matin, sauf les samedi et dimanche.

Exercice 12.6.2

Écrivez le shell script `killprog` qui permet d'envoyer le signal SIGKILL à un processus désigné non pas par son PID mais par son nom : `killprog bash`. Il faudra prendre garde au fait que plusieurs processus différents peuvent porter le même nom, en présenter clairement la liste, et toujours demander confirmation avant d'envoyer le signal.

Chapitre **13**

Réseaux

13.1 INTRODUCTION

Après l'informatique centralisée, nous avons connu une période de développement intense de l'informatique personnelle : stations de travail et surtout micro-ordinateurs. Cette nouvelle informatique a bouleversé les façons de faire et les organisations informatiques traditionnelles. Mais l'évolution a été trop radicale et cette informatique personnelle s'est révélée souvent trop individualiste pour s'intégrer harmonieusement à des organisations importantes.

Si les inconvénients de la centralisation de l'informatique sont connus, les avantages de la centralisation de l'information sont évidents : comment garantir en pratique l'intégrité de l'information si elle est dupliquée sur de nombreux systèmes incompatibles ou communicant mal entre eux ?

Les réseaux locaux se sont alors développés pour résoudre ces problèmes. L'informatique est aujourd'hui très souvent distribuée, ce qui permet de bénéficier simultanément des avantages de la centralisation de l'information et de la décentralisation des moyens, et des atouts que procure une circulation rapide de l'information entre tous ses utilisateurs.

S'il n'est donc pas indispensable d'utiliser Unix/Linux pour construire un réseau (surtout homogène), il est par contre très avantageux de disposer de machines Unix/Linux pour fédérer un réseau de machines hétérogènes. Dans ce domaine aussi, Unix a été un puissant fédérateur et les protocoles développés initialement sous ce système sont supportés par toutes les plates-formes existantes. Ce fait a largement contribué au succès des systèmes ouverts.

13.1.1 Les avantages du réseau

a) Communication facile et rapide de l'information

Particulièrement importante dans le domaine de la recherche qui a vu naître les grands réseaux, la communication rapide et à grande échelle de l'information est indispensable à toute organisation dont la taille dépasse le groupe d'individus. Une organisation répartie sur plusieurs sites distants ne peut plus se passer d'un réseau d'interconnexion, quelle que soit son activité.

b) Partage de ressources : matérielles, logicielles, données

La mise en commun de ressources matérielles : imprimantes, espace disque, périphériques coûteux ou calculateurs puissants utilisés de façon épisodique, est une puissante motivation à la mise en réseau. La mise en commun de ressources logicielles procède de la même logique, et une licence logicielle, comme une imprimante, peut être partagée.

La mise en commun de données est un point essentiel au bon fonctionnement d'une organisation. Sans aller jusqu'à parler des bases de données, la centralisation et le partage de l'information permettent d'éviter les incohérences de duplication. La mise à jour non simultanée des différentes copies conduit en effet à des erreurs d'exactitude de l'information pour certains utilisateurs.

Outre ces deux points (économie de moyens techniques et amélioration de l'intégrité de l'information), cette mise en commun de ressources procure d'importantes économies de moyens humains : la mise à jour d'un jeu de données ou d'un logiciel, effectuée une fois, est immédiatement prise en compte par tous les utilisateurs de toutes les machines du réseau.

c) Accès immédiat et transparent à l'outil le plus adapté

Il s'agit simplement d'optimiser l'investissement en moyens informatiques. Par exemple, un utilisateur effectuant d'importants calculs statistiques sur de grands jeux de données peut avoir besoin d'un tableur pour visualiser ses résultats. Pourquoi utiliser la même machine pour ces deux tâches si différentes ? Si l'accès transparent à l'information est assuré, la mise en réseau de machines de puissances très différentes permet d'utiliser chacune de façon optimale, sans inconvénient pour l'utilisateur.

13.1.2 Les applications réseau

Les applications réseau représentent les fonctionnalités offertes par un réseau de machines. Elles peuvent être classées en cinq catégories présentées ci-dessous. Ces applications, dont les noms sont indiqués entre parenthèses, seront détaillées dans la suite de ce chapitre.

a) Transfert de fichiers (`ftp, tftp, rcp, scp`)

Un transfert tout électronique de fichiers entre machines distantes évite tout problème d'incompatibilité et de manipulation de média, et améliore les délais d'échanges de fichiers.

b) Connexion sur un ordinateur distant (`telnet, rlogin, rsh, ssh`)

L'utilisateur d'une machine A peut soit transformer son poste de travail en terminal d'une machine B, soit lancer des commandes sur cette machine B. Travaillant en environnement multi-fenêtre, un utilisateur pourra ouvrir des sessions simultanées sur des machines différentes sans quitter son poste de travail.

c) Courrier électronique (`mail, talk`)

De type courrier (`mail`) ou conversation interactive (`talk`), le courrier permet de gérer facilement des listes de diffusion. Si de plus il permet d'attacher à un message des documents non textuels (données binaires, exécutables,etc.), c'est un outil de communication irremplaçable.

d) Accès transparent à des fichiers distants (`NFS`)

Fonctionnalité très supérieure au simple transfert de fichiers cité plus haut : un sous-ensemble de l'arborescence de fichiers d'une machine est physiquement localisé sur le disque d'une autre machine. Du point de vue de l'utilisateur (et des logiciels utilisant ces fichiers), rien ne distingue ces fichiers distants de fichiers locaux. Cette fonctionnalité permet la mise en commun d'espace disque, mais surtout de logiciels ou de données, et s'étend hors du monde Unix, en particulier vers les systèmes Apple et Microsoft. En particulier, dans le monde Microsoft, les protocoles SMB sont très utilisés pour cette fonctionnalité de partage de fichier. Le logiciel SAMBA met en œuvre ces protocoles sur une machine Linux, et permet de partager des fichiers entre Linux et des postes de travail fonctionnant sous Microsoft Windows.

e) World Wide Web (`navigateur internet`)

Cette application est de loin la plus connue du grand public, au point de symboliser à elle seule "l'accès à Internet". Elle résulte de la fusion en une seule application de deux technologies :

- **l'hypertexte** permet d'inclure dans un document de grande taille des marques de renvoi (les liens hypertexte) vers d'autres parties du document, ou vers un autre document hypertexte.
- le **protocole http** permet d'accéder par le réseau à une page d'un document hypertexte stocké sur une machine quelconque du réseau. Ce protocole peut être considéré comme une simplification à l'extrême d'un ftp anonyme.

Dans un hypertexte au format HTML (format normalisé de description de documents sur le World Wide Web), les liens hypertextes permettent de renvoyer à un document situé sur une machine quelconque du réseau.

13.1.3 Les différentes échelles de réseaux

Il n'est pas inutile de définir quelques termes fréquemment rencontrés dans le vocabulaire des réseaux, et qui sont parfois sources d'ambiguïtés.

Le réseau local, **LAN** pour **Local Area Network**, désigne un réseau de faible extension, interconnectant sur un même médium quelques dizaines de machines. Un tel réseau est limité, dans le cas général, à un bâtiment ou un faible nombre de bâtiments.

Le réseau métropolitain, **MAN** pour **Metropolitan Area Network**, est un réseau plus étendu, constitué de plusieurs réseaux locaux interconnectés par des liaisons point à point dont les débits sont très souvent identiques à ceux des réseaux locaux.

Le réseau à grande échelle, **WAN** pour **Wide Area Network**, désigne un réseau constitué de plusieurs réseaux des types précédents, interconnectés par des réseaux de télécommunications publics. C'est le cas, par exemple, du réseau **Internet**, d'extension planétaire, et sur lequel sont connectées des millions de machines.

Les frontières entre ces différentes échelles ne sont pas toujours très claires, mais les définitions précédentes s'appliquent à la majeure partie des cas. Encore faut-il **distinguer le réseau physique du réseau logique**, ce qui n'est pas toujours facile pour l'utilisateur : l'appartenance au réseau Internet peut être pour lui une réalité plus immédiate que la limite de son réseau local. Les applications réseau, en tout état de cause, ne seront pas les mêmes suivant l'échelle : si le transfert de fichier, l'émulation de terminal et le courrier s'utilisent sur des réseaux à large échelle, le partage de fichiers ne se pratique habituellement que sur des réseaux locaux ou métropolitains.

13.1.4 Le concept client-serveur

Les applications réseau reposent sur le concept de **client-serveur**, terme qui prête parfois à confusion. Des processus "clients" utilisent sur le réseau des services assurés par des processus "serveurs". Les requêtes du client sont normalisées, de même que les réponses du serveur. En d'autres termes, client et serveur sont les deux moitiés d'une application réseau, et sont conçus pour travailler ensemble via le réseau. Le client prend l'initiative de la communication, le plus souvent lancé par un utilisateur. Le serveur, programme démarré automatiquement à la mise en route de l'ordinateur, répond aux requêtes du client. Dans la terminologie Unix/Linux, on parle de *démon (daemon)* pour désigner ces processus qui tournent constamment en attendant une requête provenant d'un client. La terminologie Microsoft Windows utilise le mot : *service*.

Il serait erroné de penser qu'un serveur est une machine : c'est un processus. Sur UNE machine Linux s'exécutent simultanément DES serveurs (et peut-être aussi DES clients). Il serait tout aussi erroné de penser que le serveur fournit des données et que le client les reçoit.

Il serait encore plus erroné d'associer serveur à "grosse machine" et client à "petite machine". Nous verrons plus loin que lorsqu'un utilisateur se connecte sur un ordinateur à partir d'un terminal X, le serveur X, qui exécute les opérations graphiques, est dans le terminal X. Les clients X, applications demandant un affichage graphique, s'exécutent sur l'ordinateur hôte. Idéalement, il faudrait éviter d'utiliser le terme serveur pour désigner une machine. Malheureusement, cet usage se répand de plus en plus, y compris dans le monde Linux.

13.1.5 Modèle Internet

L'interconnexion de machines, voire de réseaux, a été entreprise dès la fin des années 60 sous l'égide du ministère de la Défense, aux Etats-Unis. Ces travaux ont conduit à Internet, lancé en 1973 par la DARPA (Defense Advanced Research Project Agency), et finalisé en 1981.

Le **modèle Internet** est une norme d'interconnexion de réseaux. Ce réseau à grande échelle s'est mis en place à partir du début des années 1970 sous l'égide de l'Advanced Research Program Agency (précédant la DARPA). C'est dans ce cadre qu'a été introduite la notion de couches de communication.

13.2 LES PROTOCOLES INTERNET

Historiquement, les protocoles Internet ont été définis à une époque où l'université de Berkeley développait conjointement son réseau informatique et sa propre version d'Unix. La création de Berkeley Software Distribution (et son financement par la DARPA) visait à mettre à la disposition de tous les acteurs l'ensemble des développements concernant les protocoles TCP/IP. Cette intégration d'Internet à Unix BSD a largement contribué au succès de ces protocoles.

Les **protocoles Internet** (**IP, TCP** et **UDP**) sont universellement utilisés, non seulement par toutes les versions d'Unix et Linux, mais par tous les constructeurs et éditeurs de systèmes d'exploitation.

Le lecteur intéressé par ce sujet consultera avec profit l'ouvrage de Douglas Comer, TCP/IP : Architectures, protocoles, applications, (Inter-Éditions, 1992)

13.2.1 Le modèle Internet (modes connecté-non connecté)

La communication entre un client et un serveur peut se faire sous deux modes.

Dans le **mode connecté**, une connexion est établie entre la source et la destination pour que l'application puisse se dérouler. C'est le modèle de la conversation téléphonique. Le client et le serveur échangent des données et des accusés de réception concernant ces données.

Dans le **mode non connecté**, client et serveur s'échangent des messages de données sur le réseau sans vérification d'acheminement correct ni de réception par le destinataire.

Au niveau le plus bas, **Internet** fonctionne en **mode non connecté** : la source émet des paquets d'information sans établissement d'une connexion préalable, et ces paquets seront acheminés indépendamment les uns des autres jusqu'à leur destination. C'est au niveau de la couche de transport qu'un "circuit virtuel" peut être établi entre source et destination pour assurer une **communication en mode connecté**.

Dans le modèle Internet, il est possible d'assurer toute la communication en mode non connecté, ce qui présente des avantages de vitesse et de simplicité, au prix d'une

moindre sécurité dans la transmission. Le mode non connecté est encore appelé **mode datagramme**.

L'existence possible de ces deux modes de communication se traduit par deux protocoles de transport différents dans le modèle Internet : TCP et UDP.

Le modèle Internet définit les quatre couches suivantes (la couche 1 est la plus basse) :

#	Couche	Protocole	Rôle
4	Application	FTP TELNET etc.	Service universel vu par l'utilisateur, par exemple : transfert de fichiers, courrier électronique, etc.
3	Transport	TCP ou UDP	Contrôle de l'information de bout en bout, communication de processus à processus
2	Routage	IP	Acheminement des paquets de passerelle en passerelle, de machine source à machine destination.
1	Réseau physique	Ethernet,...	Transport physique de l'information

La figure suivante montre l'organisation générale du modèle Internet et l'empilement des différentes couches. Les applications peuvent être elles-même organisées en plusieurs couches, comme tout logiciel complexe. Un navigateur, par exemple, est une application cliente faisant appel à X11 et mettant en œuvre les protocoles http, ftp, et d'autres.

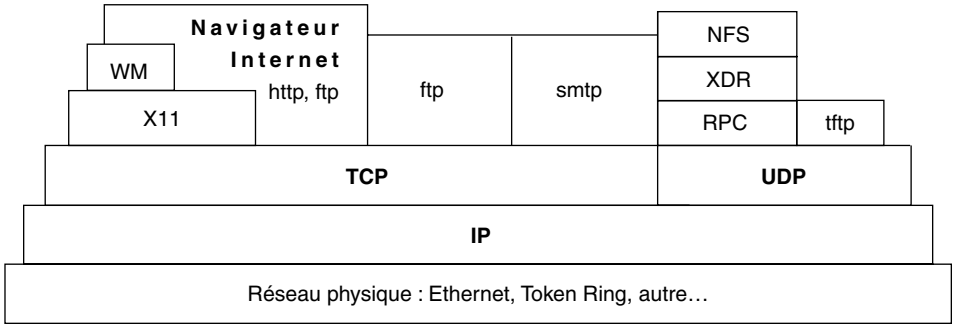


Figure 13.1. Le modèle Internet.

13.2.2 La couche réseau physique

En pratique, et dans la très grande majorité des cas, la couche réseau physique du modèle Internet est la **norme Ethernet/IEEE 802.3**. Ce n'est nullement la seule

possibilité. Du fait que l'interface entre la couche IP et la couche réseau physique est normalisée, il est possible d'utiliser comme réseau une des normes Token Ring IEEE 802.5, Token Passing Bus IEEE 802.4, X25, liaison série V24 (SLIP : Serial Line IP), ou autre.

La description d'Ethernet dépasse le cadre de ce livre, et le lecteur intéressé par le sujet peut se reporter à de nombreux ouvrages spécialisés, par exemple *Les Réseaux Ethernet d'Alexis Ferrero (Addison-Wesley)*. Pour la suite de notre exposé, il suffira de savoir qu'**Ethernet est un réseau local** ayant les caractéristiques suivantes :

- transmission par **trames** de longueur variable mais de structure normalisée
- détection des collisions avec réémission des trames perdues
- débit de 10/100 Mbit par seconde (dans une trame)
- adressage physique sur 6 octets

Sans entrer dans les détails, une **trame Ethernet** contient les champs suivants :

- préambule
- adresse Ethernet de destination
- adresse Ethernet source
- longueur de la trame
- données
- conclusion

Aucune structure n'est imposée au champ de données (sinon sa longueur maximum). Les protocoles de niveaux supérieurs, par exemple **IP**, seront donc "empaquetés" dans le champ de données des trames Ethernet. Ce mécanisme d'empaquetage est utilisé récursivement à chaque couche. Par exemple :

- les données sont transmises dans le champ de données d'un message TCP,
- ce message TCP est empaqueté dans le champ de données d'un message IP,
- ce message IP est empaqueté dans le champ de données d'une trame Ethernet.

Ce mécanisme d'empaquetage permet également de véhiculer sur un même réseau Ethernet des protocoles totalement différents, par exemple TCP/IP, Decnet et Apple, implémentant sur un même réseau physique des réseaux logiques distincts (et s'ignorant mutuellement, mais ceci est une autre histoire…).

13.2.3 La couche routage - le protocole IP Inter-network Protocol

Ce protocole résout le problème de l'acheminement de l'information dans un réseau à grande échelle. Son rôle principal est donc le routage de réseau en réseau, et il garantit l'universalité de l'adressage des machines sur le réseau Internet.

Les adresses IP

Une adresse IP (version 4) est constituée de quatre octets (seulement) ; elle est découpée en deux champs : adresse du réseau - adresse de la machine sur ce réseau.

IP définit trois classes de réseaux correspondant à des adresses ayant la forme suivante :

- réseau de classe A : 0xxxxxxx yyyyyyyy yyyyyyyy yyyyyyyy
- réseau de classe B : 10xxxxxx xxxxxxxx yyyyyyyy yyyyyyyy
- réseau de classe C : 110xxxxx xxxxxxxx xxxxxxxx yyyyyyyy

Dans chaque cas xxxx représente l'adresse du réseau et yyyy l'adresse de la machine sur ce réseau.

Les réseaux de classe A, peu nombreux, peuvent compter 16 millions de machines. Les réseaux de classe B peuvent compter 65 000 machines et les petits réseaux de classe C peuvent en compter 256.

Les adresses de réseau sont délivrées par le NIC (Network Information Center), organisme veillant à l'unicité de l'adresse et centralisant certaines informations sur les réseaux connectés à l'Internet.

Les réseaux peuvent en outre être découpés en sous-réseaux (subnet), dans ce cas l'adresse comporte trois champs :

adresse du réseau - adresse du sous-réseau - adresse de la machine

(Le champ sous-réseau ne fait pas obligatoirement 8 bits.)

Les adresses Internet sont exprimées sous forme des quatre nombres codés chacun sur un octet et séparés par un point. Par exemple :

130.79.200.1 : machine sous-réseau 200.1 réseau 130.79 (classe B)
192.93.30.5 : machine 5 réseau 192.93.30 (classe C).

Il faut noter que le terme "machine" utilisé ici désigne tout équipement directement raccordé au réseau : ordinateur, mais aussi, imprimante, matériel réseau ou terminal X (terminal graphique directement raccordé au réseau).

Il serait donc possible théoriquement de connecter 256 millions de machines sur l'Internet. En fait, le découpage en réseaux conduit à attribuer des numéros IP qui ne seront jamais utilisés, et le taux de remplissage est faible : un réseau comptant 300 machines (classe B) réserve à lui seul 65 536 adresses IP.

L'adressage sur quatre octets n'est donc pas aussi confortable qu'il pourrait sembler. Le développement exponentiel d'Internet, qui compte des millions de machines, a dépassé les espoirs de ses concepteurs.

Le routage

Une fois définie l'adresse réseau de classe A, B ou C du réseau local, une adresse IP individuelle sera attribuée à chaque machine sur ce réseau. Ces machines pourront donc échanger de l'information sur ce réseau, appelé domaine local. IP permet en outre d'atteindre des machines connectées à d'autres réseaux que le domaine local.

Pour interconnecter deux réseaux locaux (appelons-les Osiris et Cendrillon), deux solutions sont envisageables :

1) Une machine A est connectée aux deux réseaux. Cette machine a un numéro IP sur le réseau Osiris et un autre numéro IP sur le réseau Cendrillon. Cette machine est passerelle (gateway) sur chaque réseau.
2) Une machine A du réseau Osiris est reliée à une machine B du réseau Cendrillon. Cela peut être réalisé par une liaison point à point ou un réseau

public de télécommunications. A est la passerelle du réseau Osiris vers Cendrillon, B celle du réseau Cendrillon vers Osiris.

IP est responsable du routage de passerelle en passerelle jusqu'à la destination. En pratique, des **tables de routage** indiquent quelle est la passerelle à utiliser pour atteindre la destination (ou la passerelle suivante vers la destination). Il n'est heureusement pas nécessaire de disposer de tables de routage complètes sur chaque machine : une passerelle par défaut (default router) est définie dans cette table, permettant de sortir de son réseau local (son domaine) par une passerelle dont la table de routage est plus détaillée. Nous verrons dans la description des fichiers de configurations qu'il est possible de donner des noms symboliques à des adresses IP de machines et de domaines. Une adresse IP symbolique est exprimée sous la forme :

machine.domaine1.domaine2.domaine3

Les domaines IP étant hiérarchisés, le domaine Internet est organisé en sous-domaines, eux-mêmes organisés en sous-domaines, etc. Le premier niveau de sous-domaine représente le pays (fr désigne la France, uk la Grande-Bretagne) ou des grandes organisations aux États-Unis (mil pour militaire, edu pour éducation, com pour sociétés commerciales). Le niveau suivant représente souvent des réseaux métropolitains, et ainsi de suite jusqu'à la machine.

L'exemple donné précédemment en adresse IP absolue

```
130.79.200.1
 § machine sous réseau 200.1   réseau 130.79 (classe B)
```

devient en adresse IP symbolique :

```
ns1.u strasbg.fr
```

machine `ns1` dans le domaine `u strasbg`, lui-même dans le domaine `fr`.

Les passerelles peuvent être des machines Unix/Linux ; ce sont souvent des équipements spécialisés appelés routeurs. Ces derniers peuvent effectuer, à haute vitesse, le routage de protocoles différents entre des réseaux de technologies différentes, ainsi que des opérations réseau telles que le filtrage sur protocoles, sur adresses ou sur applications. Comme nous l'avons déjà signalé, la transmission au niveau IP se fait :

- de machine à machine, et non de processus à processus,
- en mode datagram (non connecté).

IP ne garantit donc pas :

- que les trames émises arriveront à destination,
- que les trames émises n'arriveront à destination qu'en un exemplaire,
- que les trames arriveront dans l'ordre dans lequel elles ont été émises.

Ces vérifications sont laissées à la charge d'un protocole de niveau supérieur. Par exemple, dans une connexion TCP, l'émetteur chargera IP de transmettre des trames

de données, le récepteur chargera IP de transmettre des trames d'accusés de réception.

13.2.4 La couche transport

Le niveau IP établit la communication entre machines. La couche transport établit la communication entre processus. Les messages de cette couche seront empaquetés dans des messages IP. L'identification des processus communicant entre eux au niveau de cette couche se fait par des numéros appelés **ports**.

Par exemple, l'application `ftp` de transfert de fichier utilise le port 21. Cette information permet le multiplexage au départ et le démultiplexage à l'arrivée des messages concernant des applications différentes, véhiculés par IP.

Le protocole UDP : User Datagram Protocol

Ce protocole permet l'échange d'information en mode non connecté. Son avantage est sa simplicité (et donc sa vitesse d'exécution). La fiabilité de transmission offerte par UDP n'est pas meilleure que celle offerte par IP. Son usage sera donc réservé à des communications locales pour lesquelles la fiabilité du réseau sous-jacent est très bonne, au transfert de messages indépendants les uns des autres (pas d'ordre de transmission), aux cas où la simplicité est le critère principal.

Par exemple, UDP est utilisé par l'application **tftp : Trivial File Transfer Protocol**, elle-même utilisée par les terminaux X (ou les stations sans disque) lors de leur mise en route (boot sur le réseau). Dans cette phase de démarrage, la simplicité est un atout important. UDP est également utilisé par NFS (Network File System de Sun) : le montage de fichiers partagés sur un réseau ne se fait pas à grande distance, et la vitesse est un critère essentiel.

Le protocole TCP : Transmission Control Protocol

Comme son nom l'indique, ce protocole est destiné au contrôle de la transmission. Établissant un circuit virtuel entre émetteur et récepteur, il instaure un dialogue en mode connecté permettant, par des mécanismes d'accusé de réception et de réémission après temporisation, la vérification de la transmission effective, unique et ordonnée des messages échangés par les applications. La communication est donc fiable, mais au prix d'une plus grande complexité et d'une charge plus importante, aussi bien sur les machines que sur le réseau.

13.2.5 Les applications réseau

Les applications réseau disponibles sur TCP/IP peuvent être classées en trois groupes :

- applications de base,
- applications étendues,
- applications Unix.

a) Les applications de base

Il s'agit d'applications existant sur tout système supportant Internet, quels que soient le système d'exploitation et le réseau utilisés. Ce sont les applications de base des réseaux hétérogènes. Il faut toutefois noter que si le système d'exploitation n'est pas multi-tâche, seuls les clients seront installés, les serveurs correspondant ne pouvant l'être. Il s'agit de :

- tftp (Trivial File Transfer Protocol)

Transfert de fichiers en mode datagram (UDP) : utilisée par les terminaux X et les stations de travail sans disque lors de la phase de démarrage (boot sur le réseau), l'application *tftp* pourrait être utilisée pour des transferts de fichiers dans le cas général.

Cette commande peut poser des problèmes de sécurité : aucun mécanisme d'identification n'est mis en œuvre. Il est possible de sécuriser le serveur tftp d'une machine Linux, en restreignant ses possibilités d'accès à une partie de l'arborescence de fichiers. Dans la plupart des cas, si la fonctionnalité serveur de boot pour des terminaux X ou des stations sans disque n'est pas nécessaire sur une machine, il est préférable de ne pas installer le serveur tftp sur cette machine (pour raisons de sécurité).

- ftp (File Transfer Protocol)

Transfert de fichiers en mode connecté (TCP) : cette commande négocie une connexion avec identification de l'utilisateur sur la machine distante : si un utilisateur A sur la machine alpha échange par *ftp* des fichiers avec un utilisateur B sur la machine beta, cet utilisateur devra connaître le login et le mot de passe de B sur beta. Pour des raisons de sécurité évidentes, ce ne peut être que la même personne (pouvant avoir des noms de login différents sur alpha et beta).

Il existe une exception très intéressante à cette règle : le **ftp anonyme** (**anonymous ftp**), permettant à l'utilisateur A sur alpha de lire (mais pas d'écrire, sauf cas particuliers) des fichiers se trouvant sur la machine beta sans identification préalable. Cela suppose, de la part de l'administrateur de beta, un travail de configuration supplémentaire du serveur ftp pour que cette connexion anonyme ne pose pas de problèmes de sécurité.

Cette possibilité est très répandue sur le réseau Internet : certains centres informatiques mettent ainsi à la disposition de la communauté Internet des informations sur l'état de leurs programmes de recherche, etc. C'est également par ce moyen qu'est diffusé le logiciel libre, dont Linux.

Le client *ftp* est lancé par la commande :

```
ftp [option] [nom_de_machine]
```

Les options ne seront pas explicitées ici (utilisez *man ftp* ou *info ftp* pour en savoir plus). Le champ *nom de machine* est une adresse Internet absolue ou symbolique, par exemple *130.79.68.78* ou *happy.u strasbg.fr*. Si ce

champ est omis, le client *ftp* passe immédiatement en mode commande. Dans le cas où le champ *nom de machine* est spécifié, la connexion est établie, l'utilisateur s'identifie sur la machine distante, puis *ftp* passe en mode commande (le prompt est *ftp>*).

Exemple

Ceci illustre un transfert par *ftp* : l'utilisateur xstra sur la machine courlis va lire un fichier sur la machine beta.

```
xstra> ftp beta.u strasbg.fr
Connected to beta.u strasbg.fr.
220 beta FTP server (SunOS 4.1) ready
Name (beta.u strasbg.fr:xstra):
331 Password required for xstra
Password:
230 User xstra logged in
ftp> ls
200 PORT command successful
150 ASCII data connection for /bin/ls
Xncd14c
Xncd15b
26 ASCII Transfer complete
ftp> binary                    § préparation pour un
200 Type set to I              § transfer de type binaire
ftp> get Xncd14c tempo         § transfert du fichier
                               § Xncd14c de beta
200 PORT command successful.   § dans le fichier
                               § tempo de courlis
150 Binary data connection for Xncd14c (130.79.130.116,1479)
226 Binary Transfer complete.
1317068 bytes received in 2.543 seconds (505.8 Kbytes/s)
ftp> quit
221 Goodbye
/xstra> ls                 § vérification sur courlis
adm/ bin/ tempo tools/     § le fichier tempo existe
xstra>
```

Attention

ftp établit une connexion avec identification sur le site distant, mais il ne s'agit pas d'un login (une session de travail). Pour ce faire, il faut utiliser *telnet*, *rsh* ou *ssh*.

Si votre machine Linux est connectée à l'Internet, il est très imprudent d'utiliser *ftp*, sauf pour des sessions ftp anonymes : le mot de passe est transféré en clair sur le réseau. Dans ce cas, *sftp* doit être préféré à *ftp*.

➤ telnet (TErminaL NETwork protocol)

Émulation de terminal sur le réseau : cette application permet la connexion à distance. Tout ce qui a été dit sur la connexion et l'identification sur le site distant à propos de *ftp* est applicable à *telnet* (sauf, bien sûr, la connexion anonyme !). Le client *telnet* est lancé par la commande :

```
telnet [nom_de_machine]
```

Le champ *nom de machine* est une adresse Internet absolue ou symbolique, par exemple *130.79.68.78* ou *happy.u strasbg.fr*. Si ce champ est omis, le client *telnet* passe immédiatement en mode commande (le prompt est telnet>). Dans le cas où le champ *nom de machine* est spécifié, la connexion est établie, l'utilisateur s'identifie sur la machine distante et entre en session sur cette machine.

Exemple

Ceci illustre une connexion à distance par *telnet*.

```
xstra> telnet beta.u strasbg.fr
Trying...
Connected to beta.u strasbg.fr
Escape character is '^T'
SunOS UNIX (beta)
login:xstra
Password:
Last login: Wed Mar 3 16 :48 :57 from alpha.u strasbg.fr
SunOS Release 4.1.2 (BETA) #1 : Mon Nov 30 10:21:04 GMT 1992
<beta>/home/xstra>ls
Xncd14c Xncd15b
<beta>/home/xstra>logout
Goodbye...
Connection closed.
xstra>
```

Il est possible de passer en mode commande par un mécanisme d'échappement (<ctrl-t> dans l'exemple ci-dessus), le prompt de *telnet* étant telnet>.

Dans le cadre de cet ouvrage, nous n'irons pas plus loin dans la description de telnet. La commande Linux *man telnet* (ou la commande help de *telnet*) permet d'en savoir plus sur les possibilités de votre implémentation de cette application très simple d'utilisation.

Si votre machine Linux est connectée à l'Internet, il est très imprudent d'utiliser *telnet* : le mot de passe est transféré en clair sur le réseau. Dans ce cas, *ssh* doit être préféré à *telnet*. Dans toutes les distributions récentes de Linux, le serveur *telnet* n'est pas activé, mais tout utilisateur peut lancer un client *telnet*.

➤ smtp (Simple Mail Transfer Protocol)

Utilisée pour l'échange de messages ascii par courrier électronique, cette application n'est pas utilisée directement, mais par l'intermédiaire de la commande *mail*

ou d'un autre client de messagerie tel que *elm*, *pine*, *mutt*, *kmail*, *exmh*, *netscape*,...

b) Les applications étendues

Ces applications, de haut niveau, ne font pas partie de TCP/IP, mais ont été développées sur ces protocoles. Elles ne sont pas limitées au monde Unix/Linux, et sont disponibles sur d'autres systèmes d'exploitation. Elles ont largement contribué au développement des réseaux hétérogènes. Il s'agit de :

- RPC Remote Procedure Call (UDP)

Ce protocole permet l'exécution de procédures sur une machine distante. Equivalent à distance d'un appel conventionnel de procédure, ce mécanisme est dit synchrone : le processus en cours attend la fin de l'exécution de la procédure distante pour continuer. Ce protocole est utilisé par des applications de plus haut niveau, en particulier par NFS.

- NFS Network File System (UDP)

NFS, développée par Sun, est universellement utilisée au point d'être le standard de fait. Ce type d'application permet de greffer une partie de l'arborescence de fichiers d'une machine alpha en un point de l'arborescence d'une machine beta. La machine alpha « exporte » un répertoire, la machine beta le monte sur un répertoire de sa propre arborescence. (Cette opération de montage peut être effectuée par plus d'une machine.)

Exemple

La machine alpha exporte le répertoire */usr/share/lib*. La machine beta monte ce répertoire sur son répertoire local */usr/lib*. Le fichier */usr/share/lib/naglib.a* de alpha est accessible sous le nom */usr/lib/naglib.a* sur la machine beta, et rien ne le différencie d'un fichier local.

Les opérations d'export et de montage sont effectuées par l'administrateur, qui peut restreindre les droits d'accès aux répertoires exportés. Ce point ne sera pas détaillé ici.

- http Hyper-Text Transfer Protocol (TCP)

Protocole utilisé pour l'accès à des pages hypertextes au format HTML. Le client http (par exemple Netscape) d'une machine demande au serveur http (par exemple Apache) d'une autre machine de lui transmettre une page. En première analyse, une URL comme :

`http://cigale.u strasbg.fr/Doc/README.html`

décrit le protocole (ici, *http*),
l'adresse IP du serveur (ici, *cigale.u strasbg.fr*)
et le nom de fichier de la page recherchée (ici, */Doc/README.html*)

La connexion TCP est établie à la demande du client, utilisée pour le transfert de cette page, puis refermée. Du point de vue du principe de base, ce protocole peut être considéré comme une simplification à l'extrême d'un *ftp* anonyme. L'interprétation du contenu de la page est à la charge du client, qui va y repérer de nouvelles URL, les présenter à l'utilisateur de la façon décrite par le format HTML de la page, et la navigation continue...

➤ X11 - X Window (TCP)

Cette application sera présentée au chapitre 16. C'est une application réseau développée sur TCP. Lorsque X Window est utilisée entièrement sur une machine, les clients X locaux accèdent au serveur X local de la même façon qu'ils le feraient pour un serveur X distant.

c) Les applications Unix

Variantes à distance de commandes Unix standard, les remote commands sont des commandes habituelles précédées de la lettre **r** (pour **remote**). Toutes ces commandes nécessitent évidemment l'identification de l'utilisateur sur la machine distante. Pour toutes les remote commands, cette identification repose sur des fichiers de configuration réseau que nous décrirons plus loin.

➤ rlogin (Remote LOGIN)

```
rlogin nom_de_machine [  l nom_login]
```

Cette application est fonctionnellement équivalente à telnet, mais de machine Unix à machine Unix (Linux). L'option *l nom login* permet de spécifier sous quel nom l'utilisateur se connecte à la machine distante ; sinon le nom local est utilisé.

rlogin présente par rapport à *telnet* deux avantages : une partie de l'environnement est transmis à la machine distante, et l'identification sur la machine distante peut être automatisée sans que la sécurité en soit gravement affectée (au contraire). En pratique, pour des raisons trop longues à détailler, l'éditeur *vi* fonctionnera souvent mieux si la connexion est établie par *rlogin* plutôt que par *telnet*.

Exemple

L'identification n'est pas automatisée (*rlogin* est la seule « remote command » pouvant fonctionner sans identification automatique).

```
xstra> rlogin beta  l platane
platane's Password :  § entrée du mot de passe sans écho
Welcome to IBM AIX Version 3.2
Last login : Mon Feb 22 10 :33 :22 1993 on pts/3 from delta.osi
ris.fr
<beta>/home/platane> ls  F
bin/ lib/ mailbox/ tools/
<beta>/home/platane> exit
Connection closed
xstra>
```

➤ rcp (Remote CoPy)

rcp permet la copie de fichiers entre la machine locale et une machine distante, ou entre deux machines distantes.

```
rcp source destination
rcp  r répertoire_source répertoire_destination
```

Cette extension de la commande *cp* nécessite, pour la désignation des noms de fichiers ou de répertoires à copier, la définition de la machine sur laquelle ils se trouvent :

```
utilisateur@nom_de_machine:nom_de_fichier
```

Le champ *utilisateur@* peut être omis si le nom de l'utilisateur est le même sur les deux machines.

Exemple

```
xstra> hostname § l'utilisateur xstra est
courlis         § connecté à la machine courlis
xstra> rcp .cshrc beta:fich1
xstra> rcp  r beta:~/tools tmp
xstra> cd tmp   § vérification
xstra> ls -F    § le répertoire tools
tools/          § a bien été copié
xstra>
```

Pour pouvoir utiliser les caractères de génération des noms de fichiers sur la machine distante, il est nécessaire de les placer entre '…' (paire de "simple quotes"), pour éviter leur interprétation localement.

Exemple

```
xstra> rcp 'beta:tools/w*' tmp
xstra> cd tmp
xstra> ls  F
where* where.bak* where.bsh* where.ksh*
xstra>
```

➤ rsh (Remote shell)

rsh permet de lancer une commande Linux sur une machine distante.

```
rsh nom_de_machine [  l nom_login] [commande]
```

L'option *l nom login* a le même sens que pour *rlogin*. Si la commande est omise, *rsh* est équivalent à *rlogin*.

Exemples

```
xstra> rsh beta ls
Mail
```

```
backup
bin
fich1
sendmail
tools
xstra> rsh beta pwd
/home/xstra
/xstra>
```

Si votre machine Linux est connectée à l'Internet, il n'est pas très prudent d'utiliser les applications *rlogin, rsh, rcp* : dans ce cas, *ssh* et *scp* fournissent exactement les mêmes fonctionnalités, avec la même syntaxe, mais en établissant entre les deux machines un tunnel chiffré évitant tout risque d'intrusion ou d'attaque. Dans toutes les distributions récentes de Linux, par souci de sécurité, les serveurs *rlogind* et *rshd* ne sont pas démarrés dans la configuration par défaut (pas plus que les serveurs *telnetd* et *ftpd*).

13.2.6 Les fichiers associés au réseau

a) Les fichiers de configuration globale

Ces fichiers concernent l'administrateur système et nous n'entrerons pas dans le détail de leur description. Signalons simplement l'existence des fichiers suivants :

/etc/hosts

Ce fichier contient une liste de couples (numéro IP, nom symbolique). C'est par son intermédiaire que IP transforme un nom symbolique de machine en une adresse IP.

/etc/resolv.conf

Dans un grand réseau, il serait fastidieux (voire impossible) de garantir que le fichier */etc/hosts* est à jour sur chaque machine. Il est alors possible de définir un "serveur de noms" (DNS : Domain Name Server), sur lequel une base de données fonctionnellement équivalente à un fichier */etc/hosts* est soigneusement maintenue à jour. Le fichier */etc/resolv.conf* décrit la liste des serveurs de noms disponibles et leurs adresses IP.

/etc/hosts.equiv

Ce fichier permet de déclarer que plusieurs machines du réseau sont équivalentes. Des utilisateurs ayant des noms identiques sur des machines déclarées équivalentes n'ont pas besoin de s'identifier lors d'une session `rlogin`, `rcp` ou `rsh`. C'est une facilité, mais également un trou potentiel de sécurité (à n'utiliser qu'en toute connaissance de cause).

/etc/services

Ce fichier contient la liste des services réseaux éventuellement disponibles sur cette machine : nom - numéro de port - protocole.

Exemple

```
# Network services, Internet style
ftp      21/tcp
telnet   23/tcp
smtp     25/tcp   mail
tftp     69/udp
```

b) Les fichiers de configuration personnelle

Ces fichiers sont créés par l'utilisateur pour configurer son propre environnement réseau.

$HOME/.rhosts

Sur la machine courlis, le fichier */home/xstra/.rhosts* contient :

```
beta.u strasbg.fr xstraplus
```

L'utilisateur xstra de courlis déclare faire confiance à l'utilisateur xstraplus de beta. Dans ce cas, une remote command lancée sur la machine courlis par l'utilisateur xstraplus de la machine beta sera exécutée sans identification préalable. Il est bien évident que l'utilisateur xstraplus de beta est la même personne que l'utilisateur xstra de courlis. Il faut noter que cette confiance n'est pas automatiquement réciproque : l'utilisateur xstraplus de beta peut ne pas avoir créé de fichier *$HOME/.rhosts*. Les permissions de ce fichier doivent impérativement être : *rw*

Ce fichier établissant la confiance n'est normalement pas suffisant pour une utilisation de *ssh*. Pour que la confiance soit établie, le fichier *$HOME/.ssh/authorized keys* doit contenir la clef de l'utilisateur client. La description des mécanismes de sécurité mis en œuvre dans *ssh* dépasse largement le cadre de cet ouvrage. Pour plus d'information, le lecteur intéressé devra se reporter à la commande : *man ssh* ou consulter le site : *http://www.openssh.com/*

$HOME/.netrc

Ce fichier est l'équivalent du précédent pour *ftp* et *telnet*; il contient des mots de passe en clair ! NE L'UTILISEZ JAMAIS, sauf pour automatiser des transferts ftp anonymes, et dans ce cas seulement.

$HOME/.forward

Ce fichier permet de rediriger les messages vers un autre destinataire ou, lorsque l'utilisateur possède un compte sur des machines différentes, de rediriger ses messages sur le compte de sa machine "principale". Il est situé dans le répertoire d'accueil ($HOME) des utilisateurs qui souhaitent renvoyer leurs messages dans une autre boîte aux lettres. Les permissions de ce fichier doivent impérativement être : *rw*

Si les protections de ce fichier ou de votre répertoire d'accueil ne sont pas prudentes, un autre utilisateur peut détourner votre courrier.

Exemple

```
xstra> whoami
jpaul
xstra> more .forward
\jpaul
xstra@courlis
```

L'utilisateur *jpaul* redirige la totalité des messages reçus vers la boîte aux lettres de l'utilisateur *xstra* de la machine courlis, tout en en conservant une copie localement (ligne \ jpaul).

13.3 EXERCICES

Exercice 13.3.1

Dans une équipe disposant de plusieurs machines Unix/Linux reliées en réseau, un utilisateur a pour nom de login pierre sur la machine courlis et pierrot sur la machine sapin. La machine sapin est équipée d'une unité de sauvegarde sur cartouche, mais pas la machine courlis. Comment fait cet utilisateur pour sauvegarder sur l'unité de sauvegarde de sapin toute son arborescence personnelle de la machine courlis.

Exercice 13.3.2

Récupérez sur le ftp anonyme de leonardo.u-strasbg.fr dans le *répertoire /pub/livre linux/exemples* le fichier tout.tar.gz.

Exercice 13.3.3

Man multi-plateforme.

Une équipe dispose de plusieurs machines unix de constructeurs différents, dont Linux, et l'utilisateur Pierre Colin dispose d'un login sur chacune de ces machines. Son nom de login n'est pas le même sur chaque machine. Cet utilisateur est enregistré sur les machines suivantes :

```
machine :  login   : Version d'Unix :
----------+----------+-----------------
courlis   : pierre  : HP-UX
sapin     : pierrot : SUN Solaris
mickey    : pierre  : IBM AIX
dingo     : pcolin  : Linux
```

Les commandes de même nom n'ont pas exactement les mêmes options et comportements sur ces différents Unix, et le *man* de chaque machine précise exactement leur fonctionnement. L'utilisateur pierre travaille normalement sur mickey, mais il veut pouvoir simplement consulter le *man* sur les autres machines, sans avoir à suivre une fastidieuse procédure de login. Il aimerait disposer des commandes suman, hpman, liman, lui permettant de consulter le man respectivement de SUN, HP et Linux lorsqu'il travaille sur sa machine habituelle.

man ls : le man de ls sur la machine locale (AIX)

suman ls : le man de ls sur la machine SUN (sapin)

hpman ls : le man de ls sur la machine HP (courlis)

liman ls : le man de ls sur la machine Linux (dingo)

Écrivez ces commandes.

Rassurez-vous : la solution est beaucoup plus courte que l'énoncé.

Exercice 13.3.4

Ftp en différé.

Si votre machine Linux est directement connectée à l'Internet, vous pouvez souvent constater que les temps de transfert par *ftp* peuvent devenir très longs à certaines heures de la journée. Il est très intéressant de lancer ce *ftp* à des heures moins chargées, ce qui est possible par la commande at. Comment faites-vous pour récupérer dans 8 heures et par un *ftp* anonyme, le fichier texte */pub/README* et le fichier binaire */pub/shell/prog.tar.gz* sur la machine ftp.machine.domaine et placer ces deux fichiers dans le répertoire */home/florent/bidon* ?

Chapitre **14**

Outils de manipulation de texte

Linux comporte un certains nombre d'utilitaires permettant de manipuler du texte. Il est important de bien définir ce terme : nous parlons ici d'utilitaires permettant de réaliser sur des textes des opérations de recherche, remplacement, extraction, comptage, analyse, formatage,.. et ceci de façon très rapide et automatisable. Il ne s'agit pas d'utilitaires de **traitement de texte** au sens de **Publication Assistée par Ordinateur**. Dans le contexte qui nous intéresse ici, le terme **texte** s'applique à un flot de caractères constitué de lignes. Tous les utilitaires décrits dans ce chapitre sont capables de traiter des textes contenus dans des fichiers, mais seront le plus souvent utilisés en tant que filtres : ils lisent un texte ligne par ligne sur leur entrée standard, le traitent et produisent un texte et/ou un résultat sur leur sortie standard. Grâce au mécanisme des tubes (voir le paragraphe 7.3), il sera possible de réaliser des traitements très complexes sur des flots de texte de taille illimitée. Le flot d'entrée peut être la sortie d'une commande ou le contenu d'un fichier, le résultat pouvant être redirigé vers un fichier. Tous ces utilitaires interprètent la même description des motifs de caractères : les **expressions régulières**.

Ces utilitaires présentent une gradation de possibilités : du plus simple au plus puissant, chacun d'entre eux peut faire ce que font les précédents. On pourrait se contenter de ne connaître que le plus complet. En pratique, il vaut mieux utiliser le plus simple dans chaque cas. Ces utilitaires sont les suivants, par ordre croissant de possibilités :

`grep` Recherche des chaînes de caractères conformes à un motif donné et les affiche.

`sed` (Stream Editor, éditeur de flot)
Recherche des chaînes conformes à un motif donné et les modifie.
Petit langage de programmation permettant l'écriture de programmes : les scripts `sed`.

`awk` Permet de rechercher, modifier, formater, compter, afficher, traduire, …
`awk` est un langage de programmation spécialisé permettant d'effectuer des traitements relativement complexes sous forme de programmes très courts.

Nous présenterons également `tr`, complémentaire des utilitaires précédents, bien qu'il ne traite pas les expressions régulières. Il rend de grands services dans la même classe d'applications.

Nous allons d'abord présenter les expressions régulières d'une façon très générale et indépendamment des utilitaires `grep`, `sed`, `awk` puis nous présenterons les utilitaires eux–même.

14.1 LES EXPRESSIONS RÉGULIÈRES

Les expressions régulières permettent de décrire des motifs formés de caractères. Ce mécanisme est très similaire au mécanisme de génération de noms de fichiers par le shell que nous avons décrit au paragraphe 7.7. Il en diffère toutefois par les aspects suivants : la génération de noms est un mécanisme rudimentaire qui décrit de façon générique des **noms** de fichiers, alors que les expressions régulières représentent une description générique très sophistiquée de chaînes de caractères **contenues dans** des fichiers (ou un flot de caractères).

Exemples

Rechercher toutes les chaînes de caractères alphabétiques dans :

- le fichier `fich1.f`
- tous les fichiers dont les noms se terminent par `.f`
- tous les fichiers dont les noms commencent par une majuscule et se terminent par `.f`

```
xstra> grep    '[a zA Z]'     fich1.f
xstra> grep    '[a zA Z]'     *.f
xstra> grep    '[a zA Z]'     [A Z]*.f
                    ↑              ↑
§     expression régulière  génération de noms de fichier
```

La syntaxe des expressions régulières est en outre différente de la syntaxe de génération de noms. Ces expressions sont utilisées par un grand nombre d'utilitaires : non seulement *grep*, *sed* et *awk*, mais également par le langage *perl* (the Practical Extraction and Report Language) et tout éditeur de texte, dont *vi* et *emacs*.

Les expressions régulières servent à décrire de façon concise et inambiguë des motifs de caractères, comme par exemple : « toutes les chaînes de 1 à 8 caractères alphanumériques, le premier étant alphabétique ».

Cette périphrase traduite en expression régulière devient :

```
[A Za z][A Za z0 9]{0,7}
```

Les expressions régulières (et les utilitaires qui les comprennent) ne sont pas limitées au monde Unix : *grep*, *sed*, *awk*, et le langage *perl* ont été portés sur d'autres systèmes d'exploitation.

Il existe deux sortes d'expressions régulières : les **expressions régulières étendues (ere)** que nous décrirons ici, et les **expressions régulières de base**, considérées comme obsolètes dans le monde linux, dont nous dirons quelques mots.

14.1.1 Conventions d'écriture

Dans ce chapitre, on dira qu'une chaîne de caractères **vérifie** une expression régulière (l'aide en ligne utilise le terme : **match**), et le terme **caractère** exclut le caractère <new-line>.

- ch désigne un caractère quelconque sauf un caractère spécial
- sp désigne un caractère spécial

Les caractères spéciaux pour les expressions régulières sont les suivants :

```
| . * + ? ^ $ ( ) [ ] {} \
```

Les expressions régulières sont construites progressivement, à partir de briques de base appelées **expressions régulières atomiques**.

14.1.2 Les expressions régulières atomiques : era

On appelle expression régulière atomique (**era**) un motif constitué d'**un seul caractère** appartenant à un **ensemble** précis.

cette era	définit l'ensemble de caractères suivants
ch	l'ensemble constitué du seul caractère **ch**
\sp	l'ensemble constitué du seul caractère spécial **sp**
.	l'ensemble de tous les caractères
[gkl]	l'ensemble constitué des caractères placés entre [ ] (**g,k,l**)
[^gkl]	l'ensemble constitué des caractère autres que **g,k,l.** Le caractère ^ placé après [indique la négation.
[a-z]	l'ensemble constitué des caractères compris entre **a** et **z** (ordre alphabétique)
[^a-z]	l'ensemble constitué des caractères non compris entre **a** et **z**

Exemples

```
a       § le caractère a
[i n]   § un caractère appartenant à l'ensemble ijklmn
[ i n]  § un caractère appartenant à l'ensemble  ijklmn
.       § un caractère quelconque
\.      § le caractère .
[^0 9]  § tout caractère autre qu'un chiffre
[^0 9^] § tout caractère autre qu'un chiffre ou ^
        § car le dernier ^ n'est pas après [
```

14.1.3 La construction d'une expression régulière : er

Si l'on recherche une chaîne de caractères (un "mot") conforme à un certain motif, il faut pouvoir définir l'ensemble des caractères constituant ce "mot", le nombre de caractères dans ce mot, et à quel endroit le chercher (quoi ? combien ? où ?). Par exemple : un mot de 3 à 6 lettres minuscules au début de la ligne. Les briques élémentaires (**era**) définissent des ensembles de caractères (quoi ?) et seront combinées entre elles par trois opérations élémentaires : la concaténation, la quantification (combien ?) et l'ancrage (où ?).

La concaténation

Une expression régulière (**er**) peut être obtenue par concaténation (juxtaposition) d'expressions régulières atomiques (**era**). La concaténation est décrite sans opérateur ou caractère spécial :
abc signifie : a suivi de b suivi de c.

Exemples

```
abc             § la chaîne abc
[Oo]ui          § la chaîne Oui ou la chaîne oui
[A Z][0 9]..    § une majuscule suivie d'un chiffre suivi
```

```
                  § de deux caractères quelconques :
                  § A8b5 ou Z444 mais pas h7fu
[a z][0 9].\.     § une minuscule suivie d'un chiffre suivi
                  § d'un caractère quelconque suivi d'un
                  § point : a8b. ou h6.. mais pas Z67f
```

La quantification

Les **quantifieurs** permettent de définir combien de fois l'**era** qui précède est répétée. Si ⊗ désigne une expression régulière atomique les quantifieurs utilisables sont les suivants :

quantifieur	signification
⊗*****	tout mot de 0 à N caractères vérifiant ⊗
⊗**+**	tout mot de 1 à N caractères vérifiant ⊗
⊗**?**	tout mot de 0 à 1 caractère vérifiant ⊗
⊗**{n}**	tout mot de n caractères vérifiant ⊗
⊗**{n1,n2}**	tout mot de n1 à n2 caractères vérifiant ⊗
⊗**{n1,}**	tout mot d'au moins n1 caractères vérifiant ⊗
⊗**{,n2}**	tout mot de 0 à n2 caractères vérifiant ⊗

Dans ce tableau, la phrase « tout mot de n caractères vérifiant ⊗ » signifie exactement : « tout mot de n caractères appartenant à l'ensemble défini par l'expression régulière atomique ⊗ ».
La notation **{n,m}** est générale mais lourde pour les cas simples, c'est pourquoi :

- ***** est utilisé à la place de **{0,}**
- **+** est utilisé à la place de **{1,}**
- **?** est utilisé à la place de **{0,1}**

Exemples

```
abc       § la chaîne abc
a\.       § la chaîne a.
a.        § a suivi de n'importe quel caractère
a*        § rien ou a ou aa ou aaa ou ...
a.*       § a suivi de n'importe quelle chaîne (même vide)
          § (a ou ab ou abc ...)
a+        § a ou aa ou aa ou aaa ou ...
a?        § rien ou a
a{2}      § la chaîne aa
[a b]{2}  § aa ou ab ou bb ou ba
```

L'ancrage

Deux caractères spéciaux servent à préciser la position de la chaîne recherchée dans la ligne :
^ désigne le début de la ligne (s'il est placé au début de l'expression)
$ désigne la fin de la ligne (s'il est placé à la fin de l'expression)

Exemples

```
^linux  § linux en début de ligne
linux$  § une ligne terminée par linux
^linux$ § une ligne ne contenant que linux
^$      § une ligne vide
```

14.1.4 Combinaison d'expressions régulières

Deux opérations permettent de combiner entre elles des expressions régulières : **l'alternative** et le **groupage**.

L'alternative

Les expressions régulières permettent de désigner "ceci ou cela" en séparant deux expressions régulières (er) par le caractère **|** comme dans : "*ceci|cela*". Cette combinaison est une nouvelle expression régulière.

Exemples

```
linux|unix          § le mot linux ou le mot unix
^linux|^unix        § le mot linux ou le mot unix au début
[A Z]{8}|[0 9]{4}   § 8 lettres majuscules ou 4 chiffres
```

Le groupage par (...) et la notation \W ...

Mettre une expression régulière entre parenthèses ne change rien à cette expression : l'expression **(er)** représente toute chaîne qui vérifie **er**. Les parenthèses semblent inutiles. Cette notation offre cependant deux possibilités :

- les quantifieurs vus plus haut (voir le paragraphe 14.1.3) peuvent s'appliquer à de telles expressions,
- la notation **\N**, où **N** est un nombre de 1 à 9, désigne la chaîne de caractères qui a vérifié l'expression placée dans la N^{ème} paire de parenthèses.

Exemples

```
tsoin           § tsoin et aucune autre chaîne
tsoin{2}        § tsoinn et aucune autre chaîne :
                § Le quantifieur {2} n'est appliqué qu'au
                § dernier n
(tsoin){2}      § tsointsoin et aucune autre chaîne :
                § Le quantifieur {2} est appliqué à
```

```
                § l'expression tsoin
(bla|tsoin){2}  § blabla ou tsointsoin
                § ou blatsoin ou tsoinbla
(bla|tsoin)\1   § blabla ou tsointsoin
                § MAIS PAS blatsoin ni tsoinbla
```

Attention

\0 désigne le caractère <**null**> et **\N** ne référence la nième paire de parenthèses que si elle existe.

Dans le groupage par **()**, la notation **\N** peut être complétée par ***** **+** ou **?** avec les significations habituelles *:*

- **\1*** signifie 0 à N fois la chaîne qui a vérifié le premier sous-motif
- **\3+** signifie 1 à N fois la chaîne qui a vérifié le troisième sous-motif
- **\2?** signifie 0 ou 1 fois la chaîne qui a vérifié le deuxième sous-motif

Cette notation est appelée *:* référence arrière à une sous-chaîne (back-reference to a sub-string). Elle permet de désigner une chaîne inconnue au moment de l'écriture de l'expression, comme par exemple :
"une chaîne identique au premier mot de la ligne ".

14.1.5 D'autres expressions régulières atomiques

Nous avons présenté les expressions régulières atomiques comme étant un moyen de définir un ensemble de caractères, par exemple :

```
[_A Za z0 9]
```

désigne un caractère alphanumérique ou le signe _.

D'autres possibilités permettent de rendre les expressions régulières atomiques plus complètes (c'est utile) et utilisables avec des jeux de caractères autre que l'ASCII US (en particulier avec des caractères accentués).

- plus complètes : comment désigner un caractère spécial,
- internationales : comment définir une classe de caractères indépendamment de la langue locale.

Les caractères spéciaux et la notation \X

Il est possible de désigner des caractères spéciaux de la façon suivante :

\0	le caractère <**null**>
\a	le caractère <**alert**> (bell)
\b	le caractère <**backspace**>
\f	le caractère <**form-feed**>
\n	le caractère <**new-line**>

\r	le caractère <**carriage-return**>
\t	le caractère <**tab**>
\v	le caractère <**vertical-tab**>
\013	le caractère dont le code ASCII est **013** en octal

Les classes de caractères

Les classes de caractères sont définies par la notation *[:code:]* et ne peuvent être utilisées que dans la définition d'une liste de caractères (era). La correspondance entre ces classes et les éléments qui les composent sont donnés ici pour la langue anglaise :

[[:alnum:]]	un alphanumérique (**[0-9a-zA-Z]**)
[[:alpha:]]	un alphabétique (**[a-zA-Z]**)
[[:cntrl:]]	un caractère de contrôle (**[\a\b\r\f\t\n\v]**)
[[:digit:]]	un digit (**[0-9]**)
[[:graph:]]	un caractère autre qu'alphanumérique ou ponctuation
[[:lower:]]	une lettre minuscule (**[a-z]**)
[[:print:]]	un caractère imprimable
[[:punct:]]	un caractère de ponctuation
[[:space:]]	un caractère d'espacement (**[\t\r\n\f]**)
[[:upper:]]	une lettre majuscule (**[A-Z]**)
[[:xdigit:]]	un digit hexa (**[0-9A-Fa-f]**)

Par exemple en français, **[[:alpha:]]** désigne toutes les lettres alphabétiques **y compris** les caractères accentués, alors que **[a-zA-Z]** ne désigne que les 26 lettres de l'alphabet en minuscule ou majuscule.

Attention

Ne pas oublier **[:** et **:]** dans les classes :

[[:print:]]	correct : un caractère imprimable
[^[:print:]]	correct : un caractère non imprimable
[^:print:]	incorrect
[:^print:]	incorrect
[[:lower:][:digit:]]	correct : une minuscule (y compris accentuée) ou un chiffre. en anglais : **[a-z0-9]**

Exemple détaillé

L'exemple suivant illustre l'utilisation des classes de caractères dans le cas des caractères accentués français : rechercher dans un texte les chaînes représentant un prénom suivi d'un nom (dans la même ligne). Un nom est une suite d'au moins deux lettres majuscules précédées d'un espace et d'un prénom. Un prénom est une suite d'au moins deux lettres, la première étant majuscule et les suivantes minuscules. Le prénom est précédé d'un espacement ou du début de la ligne. Le nom est suivi d'un espace ou d'une ponctuation ou de la fin de la ligne. Les lignes suivantes en sont des exemples :

Ce monsieur s'appelle François GUICHARD, il est assis à droite.
Frédérique OSTRÉ préfère perl : c'est bien compréhensible.
Cet exemple vous est proposé par Pierre COLIN.
Mais cette ligne ne contient PAS ce que l'on cherche.

L'expression régulière est donc construite de la façon suivante :
un nom : **`[[:upper:]]{2,}`**
un prénom : **`[[:upper:]][[:lower:]]+`**
un prénom suivi d'un nom (séparés par un espace) :

```
[[:upper:]][[:lower:]]+ [[:upper:]]{2,}
```

Il faut maintenant exprimer ce qui peut se trouver avant :

```
(^|[[:space:]])
```

... et ce qui peut se trouver après :

```
([[:space:][:punct:]]|$)
```

Et voila : on peut tout mettre ensemble en une seule ligne.
(replié en deux lignes ci-dessous pour des raisons typographiques) :

```
(^|[[:space:]])[[:upper:]][[:lower:]]
+ [[:upper:]]{2,}([[:space:][:punct:]]|$)
```

Cette expression trouvera les trois premières des quatre lignes proposées en exemple. La quatrième ligne ne sera pas prise, bien que « Mais » puisse passer pour un prénom et « PAS » pour un nom (d'après nos critères). Il est important de noter que cette expression est la même pour un utilisateur fancophone et pour un utilisateur anglais : c'est le grand avantage des classes **`[:alpha:]`**, **`[:upper:]`**, **`[:lower:]`** par rapport à une écriture comme **`[A-Za-z]`**, etc.

Le lecteur intéressé pourra améliorer l'expression proposée pour tenir compte des noms composés (OSTRÉ-WAERZEGGERS), des prénoms composés (Jean-Paul), et de la possibilité d'avoir un à trois prénoms comme :

Charles-Henri Stanislas Alfred-Marie VILLEROY-BRANDT.

14.1.6 Les expressions régulières de base

La présentation précédente se veut assez générale, c'est pourquoi les exemples n'ont présenté que les expressions régulières étendues, sans mettre en œuvre les commandes *grep*, *sed* ou *awk*, qui seront présentées dans la suite de ce chapitre.

Les expressions régulières de base, considérées comme obsolètes sous linux, ont une syntaxe différente : les caractères **? + { } ( ) |** n'ont pas de signification spéciale. Il faut les précéder du caractère **** pour qu'ils soient interprétés comme dans les expressions régulières étendues. C'est un point à vérifier dans la documentation en ligne pour chaque utilitaire.

14.2 LA COMMANDE GREP

La commande *grep* est la plus simple. Elle permet de rechercher dans un flot de texte les lignes qui vérifient une expression régulière, et ne transmet sur sa sortie standard que ces lignes. Le *grep* linux comporte trois modes de fonctionnement selon l'option :

grep -G	utilise les expressions régulières de base	défaut
grep -E	utilise les expressions régulières étendues	à préférer
grep -F	recherche des chaînes littérales	**fgrep**

Sous Linux, nous conseillons d'utiliser systématiquement la commande *grep E*, ce qui permet l'utilisation des expressions régulières étendues. La commande *fgrep* (voir ci-dessous) rend de grands services pour chercher rapidement une chaîne contenant un point ou une étoile. La commande *grep* sans option (équivalente à *grep G*) ne devrait être utilisée que pour compatibilité avec d'anciens shell scripts. La commande *grep* est décrite en Annexe A, avec des exemples. L'option *-i* permet d'inhiber la différence majuscules/minuscules dans la recherche et l'option *-v* signifie "afficher toutes les lignes qui ne vérifient **pas** l'expression régulière".

Exemples

Le fichier *prodct.f* contient (entre autres) les lignes suivantes :

```
J J518+K
J J*5
paj 5*sqrt(alpha)
```

Les deux exemples suivants montrent la différence entre *grep* et *fgrep* :

```
xstra> grep  i 'j=j*5' *.f
  § ici j*5 est une expression régulière
prodct.f : J J518+K
prodct.f : paj 5*sqrt(alpha)
  § Attention la chaîne j j*5 ne vérifie pas
  § l'expression régulière j j*5
xstra> fgrep  i 'j=j*5' *.f
  § ici * est pris tel quel
```

```
prodct.f : J J*5
xstra>
```

14.2.1 Les caractères spéciaux dans les expressions régulières

Nous allons présenter ici l'interprétation des caractères spéciaux dans une expression régulière sur une suite d'exemples simples mettant en œuvre la commande *grep*.

Soit le fichier *fichier source* contenant les trois lignes suivantes :

```
ceci est un essai
message 1 arrive, continuons
fin de l'essai.
```

***** remplace zéro fois ou n fois le caractère qui le précède.

Exemple

```
xstra> grep 'es*a' fichier_source
ceci est un essai
message 1 arrive, continuons
fin de l'essai.
xstra>
```

. désigne un caractère quelconque.

Exemple

```
xstra> grep 'c.ci' fichier_source
ceci est un essai
xstra>
```

[...] désigne un caractère quelconque appartenant à la liste donnée entre crochets. Deux caractères séparés par un tiret (-) définissent une liste : *[c g]* équivaut à *[cdefg]*.

Exemple

Recherche de toutes les lignes contenant un caractère numérique.

```
xstra> grep '[0 9]' fichier_source
message 1 arrive, continuons
xstra>
```

^ placé en début de motif, désigne le début de la ligne.

Exemple

```
xstra> grep '^c' fichier_source
ceci est un essai
xstra>
```

$ placé en fin de motif, désigne la fin de la ligne.

Exemple

```
xstra> grep 'es*a.$' fichier_source
ceci est un essai
xstra>
```

[^...] désigne une liste de caractères à exclure.

Exemple

Recherche de toutes les lignes contenant autre chose que <espace> et chaînes alphabétiques en minuscule.

```
xstra> grep '[^ a z]' fichier_source
message 1 arrive, continuons
fin de l'essai.
xstra>
```

Le tableau 14.1 présente une synthèse des différences d'interprétation des caractères spéciaux utilisés dans la génération des noms de fichiers (voir le paragraphe 7.7) et les expressions régulières étendues.

	Génération de noms de fichier	Expressions régulières
?	un caractère quelconque, sauf <new-line>	remplace zéro ou une fois le caractère qui précède
.	le caractère point	un caractère quelconque sauf <new-line>
*	zéro ou un nombre quelconque de caractères	remplace zéro fois ou n fois le caractère qui précède
[a-i]	un caractère entre a et i	un caractère entre a et i
[!a-i]	un caractère qui n'est pas entre a et i	un caractère entre a et i, ou !
[^a-i]	un caractère entre a et i, ou ^	un caractère qui n'est pas entre a et i
\	banalise le caractère qui suit	banalise le caractère qui suit
^	le caractère ^	ce qui suit est en début de ligne
$	référence une variable	ce qui précède est en fin de ligne

TABLEAU 14.1. SYNTHÈSE DES DIFFÉRENCES D'INTERPRÉTATION DES CARACTÈRES SPÉCIAUX.

14.3 LA COMMANDE SED

La commande *sed* (Stream EDitor) est un éditeur non interactif. Cette commande peut être utilisée comme un filtre. Elle traite son entrée standard ligne par ligne. La forme générale de la commande *sed* est :

```
sed [ n] [ e commandes_sed] fichier_source
sed [ n] [ f fichier_commande] fichier_source
```

La commande *sed* copie le fichier *fichier source* sur la sortie standard après application des commandes d'édition. L'option *-n* a pour effet d'inhiber la sortie à l'écran du résultat des commandes d'édition. Sans cette option, *sed* copie chaque ligne d'entrée sur sa sortie standard, après l'avoir éventuellement modifiée.

Les commandes d'édition (*commandes sed*) interprétées par *sed* peuvent être stockées dans le fichier *fichier commande* ou placées directement après l'option *-e*.

14.3.1 La commande d'édition de sed

La commande d'édition de *sed (commandes sed)* est de la forme :

```
[adresse_debut [, adresse_fin] ] fonction [arguments]
```

Les adresses peuvent avoir la forme suivante :

- une valeur numérique désignant le numéro de la ligne dans le fichier,
- */motif/* désignant la première ligne contenant *motif*,
- *$* désignant la dernière ligne.

14.3.2 Quelques commandes d'édition

Impression des lignes n à m : la fonction p

```
xstra> sed  n 'n,mp' fichier_source
```

Affichage à l'écran des lignes *n* à *m* du fichier *fichier source*, du fait de la fonction *p (print)*.

Impression sur critère

```
xstra> sed  n '/motif_début/,/motif_fin/p' fichier_source
```

Affichage à l'écran des lignes du fichier *fichier source*, lignes comprises entre la première ligne contenant la chaîne de caractères *motif début* et la première ligne contenant la chaîne *motif fin*.

Substitution d'une chaîne de caractères par une autre : la fonction s

```
xstra> sed 's/ancien_motif/nouveau_motif/g' fichier_source
```

Affichage à l'écran du fichier *fichier source*. Les lignes contenant la chaîne de caractères *ancien motif* seront affichées après substitution d'*ancien motif* par *nouveau motif*, et ceci pour toutes les occurrences d'*ancien motif* ; cela est dû au modifieur *g* qui signifie global. Sans le modifieur *g*, la substitution n'est faite que pour la première occurrence de *ancien motif* pour chaque ligne. Cette commande a été décrite au chapitre traitant *vi* (paragraphe 5.2.3).

14.4 LA COMMANDE AWK

AWK a été écrit par A.V. Aho, P.J. Weinberger, B.W. Kernighan, d'où son acronyme. *awk* est un langage de programmation spécialisé dans la manipulation de texte, et optimisé dans le sens d'une extrême concision : beaucoup de programmes *awk* ne font qu'une seule ligne. La logique de fonctionnement de *awk* est celle de *sed* : la boucle sur les lignes du flot d'entrée est implicite, le programme à appliquer est passé en paramètre ou enregistré dans un fichier dont le nom est passé en paramètre.

14.4.1 Caractéristiques du langage

- *awk* supporte des opérateurs arithmétiques et de manipulation de chaînes
- *awk* supporte les expressions régulières étendues
- La boucle sur les lignes d'entrée est implicite
- Les variables *awk* sont non déclarées et non typées, normales ou spéciales

Les variables spéciales : numéro de ligne, nombre de champs, longueur, nom du fichier, etc. existent toujours et sont mises à jour automatiquement au cours de l'exécution.

L'utilisateur peut définir des variables : elles sont simultanément de type chaîne **et** numérique, le contexte d'utilisation déterminant automatiquement la conversion appliquée par *awk*. De plus, ces variables sont créées et initialisées automatiquement à zéro (ou à la chaîne vide) dès leur première apparition dans le programme *awk*.

14.4.2 La ligne de commande awk

Deux cas peuvent se présenter : soit le programme *awk* est suffisamment court pour être placé entre apostrophes sur la ligne de commandes, soit il est plus long et enregistré dans un fichier dont le nom est passé à la commande *awk* avec l'option *-f*. L'option *-Fd* permet de fixer le séparateur de champs au caractère *d*.

La première forme de la commande *awk* est la suivante :

```
awk [ Fd] 'programme_awk' [fichiers]
```

Important

Le programme *awk* est placé entre apostrophes simples, car il comporte presque toujours des caractères susceptibles d'être interprétés par le shell.

Exemple

```
xstra> awk -F: '$5 == ""' /etc/passwd
```

Ce qui signifie : dans le fichier */etc/passwd*, sélectionner les lignes dont le cinquième champ est vide, le séparateur de champs étant le caractère :

La deuxième forme de la commande *awk* est la suivante :

```
awk [ Fd]  f fichier_awk [fichiers]
```

Exemple

```
xstra> ls  l | awk -f formatls
```

Ce qui signifie : appliquer le programme *awk* enregistré dans le fichier *formatls* à toutes les lignes produites par la commande *ls l.*

awk lit une ligne sur son entrée standard, la découpe en champs selon le délimiteur de champs et lui applique toutes les instructions du programme *awk* (la copie sur la sortie standard n'est pas automatique). Dans toute la suite le terme « **sortir** » signifie : recopier sur la sortie standard.

14.4.3 Le programme awk

Un programme *awk* est une suite de lignes du type :

```
condition { action }
```

Le champ *condition* ou le champ *action* peut être omis.

Un champ *condition* manquant est vérifié par toute ligne d'entrée.

Un champ *action* manquant sort toute ligne qui vérifie *condition.*

Exemple

```
xstra> awk '$1 == 5 {print $2}' fich1
  §  condition : si le premier champ vaut 5
  §  action : sortir le deuxième champ
```

Le champ condition

Le champ *condition* peut être soit une condition de base, soit une condition composée.

Les conditions de base sont les suivantes :

Condition	À quel moment l'action correspondante est appliquée

BEGIN	Avant lecture de la première ligne d'entrée.
END	Après lecture de la dernière ligne d'entrée.
expr	À toute ligne d'entrée pour laquelle l'expression **expr** est vraie, c'est à dire différente de zéro ou non vide (selon le contexte, évaluation numérique ou alphanumérique).
/er/	À toute ligne d'entrée contenant une chaîne vérifiant l'expression régulière **er**.
expr ~/er/	À toute ligne d'entrée pour laquelle la valeur alphanumérique de **expr** contient une chaîne vérifiant l'expression régulière **er**.
expr !~/er/	À toute ligne d'entrée pour laquelle la valeur alphanumérique de **expr** ne contient pas une chaîne vérifiant l'expression régulière **er**.

Les conditions de base, sauf **BEGIN** et **END**, peuvent être composées entre elles selon les combinaisons suivantes :

Condition	**À quel moment l'action correspondante est appliquée**
cond1 && cond2	À toute ligne d'entrée vérifiant **cond1 et cond2**.
cond1 || cond2	À toute ligne d'entrée vérifiant **cond1 ou cond2**.
!cond	À toute ligne d'entrée ne vérifiant **pas cond**.
cond1, cond2	À toute ligne d'entrée à partir de la première ligne vérifiant **cond1** et jusqu'à la prochaine ligne vérifiant **cond2** (incluses).

Les variables prédéfinies

Nom de variable	**Contenu**
FILENAME	Nom du fichier d'entrée courant
FNR	Numéro de ligne dans le fichier d'entrée courant
FS	Séparateur de champs en entrée
IGNORECASE	Si différent de 0, ne jamais faire de différence entre majuscule et minuscule
NF	Nombre de champs dans la ligne courante
NR	Numéro de ligne dans le flot d'entrée
OFS	Séparateur de champs en sortie
ORS	Séparateur de lignes en sortie
RS	Séparateur de lignes en entrée
$0	Ligne d'entrée courante
$1,$2,...,$NF	1er, 2ème, ..., dernier champ de la ligne d'entrée courante

Les expressions

Les expressions dont il est question ici ne doivent pas être confondues avec des expressions régulières. Il s'agit d'expressions composées à partir de variables et d'opérateurs, et que *awk* peut évaluer numériquement ou sous forme de chaîne selon le contexte. Toute expression peut également être considérée comme un booléen, la valeur vrai signifiant différent de zéro dans un contexte numérique et non vide dans un contexte de chaîne.

Exemples

Expression	Signification
$1+$3	La somme des 1er et 3ème champs.
NF > 5	Vrai si le nombre de champs est supérieur à 5.
$4 == ""	Vrai si le 4ème champ est vide.
($1+$2)/NF	La somme des 1er et 2ème champs divisée par le nombre de champs.
$7 ~/ksh/	Vrai si le 7ème champ contient la chaîne `ksh`.

Attention

Ne pas confondre NF et $NF

Pour un habitué de la programmation en shell débutant en *awk*, une erreur fréquente consiste à référencer une variable par *$variable* au lieu de *variable*. Tout particulièrement pour la variable *NF*, nombre de champs, *NF* représente le nombre de champs alors que *$NF* représente le dernier champ. En effet, si la ligne courante comporte 5 champs, *NF* vaut 5 et *$NF* vaut *$5* donc le 5ème (et dernier) champ.

Les opérateurs de comparaison s'appliquent aussi bien à des comparaisons numériques que lexicographiques, sauf **~** et **!~**. Ces opérateurs sont les suivants :

Opérateur	Signification	Exemples numériques et / ou lexicographiques
==	Egal à	**1** est égal à **1** et **linux** est égal à **linux**
!=	Différent de	**linux** est différent de **DOS**
>	Supérieur à	**linux** est supérieur à **dos**
<	Inférieur à	**dos** est inférieur à **dosread**
>=	Supérieur ou égal à	
<=	Inférieur ou égal à	
~	Vérifie l'ere	**/usr/bin/ksh** vérifie **/ksh/**
!~	Ne vérifie PAS l'ere	**/home/yannick** ne vérifie pas **/[0-9]+/**

Les actions

L'action par défaut est *print* : copier la ligne d'entrée courante vers la sortie standard. Les actions possibles sont celles d'un langage de programmation conventionnel, et la syntaxe est voisine de celle du langage C.

Très brièvement, les actions possibles sont des combinaisons de :

- affectation de variables, incrémentation, décrémentation
- opérations arithmétiques dont un certain nombre de fonctions prédéfinies telles que *sin*, *cos*, *atan2*, *exp*, *log*, *sqrt*, *rand*,..
- concaténation de chaînes *(var1 var2)*
- fonctions prédéfinies sur les chaînes telles que *index*, *length*, *split*, *match*, *sub*, *substr*, *sprintf*,..
- opérations de sortie telles que *print*, *printf*, *getline*,.. avec redirection possible vers un fichier ou un tube
- structures de contrôle telles que *if*, *else*, *do*, *while*, *for*,..

Une description complète de *awk* nécessiterait un livre entier, et le lecteur intéressé est invité à se référer au *man* de *awk* ou à un ouvrage plus spécialisé. Nous préférons présenter des exemples mettant en évidence la simplicité et l'extrême concision des programmes *awk*.

Exemple

Tout d'abord quelques exemples d'analyse du fichier */etc/passwd*. Dans ce fichier, les différents champs sont séparés par le caractère :, *awk* est donc invoqué avec l'option *-F:* pour plus de simplicité. Une solution équivalente, mais moins élégante, consisterait à commencer le programme *awk* par une ligne *BEGIN{FS=":"}*. Rappelons que chaque ligne du fichier */etc/passwd* décrit un utilisateur du système selon les champs :

```
login:!:uid:gid:gecos:home_dir:login_shell
1    :2: 3 : 4 :  5  :    6   :     7
yannick:!:202:245:perl
guru:/home/yannick:/usr/bin/ksh
```

Parmi les *login* décrits dans le fichier */etc/passwd*, certains correspondent à des utilisateurs et d'autres sont des *login* d'administration ne correspondant pas à des personnes. Voici quelques exemples de sélection faciles avec *awk* et qui seraient impossible avec *grep* : chercher les lignes comportant un certain motif dans un certain champ (la notion de champ n'existe pas pour *grep*).

```
#lister les login ayant un login shell
xstra> awk -F: '$7 != "" {print $1}' /etc/passwd
#peut s'écrire (plus bref, mais moins lisible)
xstra> awk -F: '$7 {print $1}' /etc/passwd

# lister les login et uid pour lesquels
# le login shell est un shell bash
xstra> awk -F: '$7 ~/bash/ {print $1,$3}' /etc/passwd
```

```
# lister les logins dont l'UID est inférieur à 100
# et qui n'ont pas de login shell
# l'action par défaut est print
xstra> awk -F: '$3 <= 100 && $7 == ""' /etc/passwd
```

Exemple

Voici un autre exemple de programme *awk* très court, et néanmoins très utile. Il sera abondamment commenté. Il s'agit de compter le nombre de mots et de lignes dans un fichier texte.

```
xstra> awk '{mot=mot+NF}END{print NR,mot}' fichtext
```

Explications pas à pas :

La variable mot est créée et initialisée à zéro par *awk*, puis pour chaque ligne (condition manquante) exécuter l'action : mot = mot + nombre de champs dans la ligne courante. Après avoir lu et traité la dernière ligne (condition END) exécuter l'action : sortir le nombre de lignes et le contenu de la variable mot.

Attention à la virgule

`print NR,mot`	sortir NR et mot séparés par un espace : 12 63
`print NR mot`	sortir la chaîne obtenue par concaténation de NR et mot : 1263

Exemples

Dans les exemples suivants, nous ne présenterons que le programme *awk* et non plus la ligne de commandes toute entière comme dans l'exemple précédent.

Sortir chaque ligne précédée de son numéro :

```
{ print NR,$0}
```

Sortir chaque ligne précédée de son numéro, ignorer les lignes commençant par le caractère # :

```
!/^#/{ n++;print n,$0}
```

Sortir les 15 premières lignes :

```
 NR <  15
```

Sortir le 1er champ, le 2ème champ, leur somme et leur différence :

```
{ print $1,$2,$1+$2,$1 $2}
```

Compter combien de lignes dépassent 80 caractères :

```
length($0) > 80 { n++}
END { print n, "lignes de plus de 80 car."}
```

Sortir les lignes dans lesquelles la somme de tous les champs est inférieure à 100 :

```
{ som 0
  for (i 1;i< NF;i++) som som+$i
  if (som < 100) print
}
# ce qui peut aussi s'écrire
{ som 0;for(i 1;i< NF;i++)som som+$i;if(som < 100)print}
```

14.5 LA COMMANDE TR

La commande *tr* (TRanslate) existe en deux versions légèrement différentes selon l'origine du système, la version system V se trouvant généralement dans */usr/bin/tr* et la version BSD dans */usr/ucb/tr*. La commande *tr* est un filtre strict : elle traite son entrée standard et produit le résultat sur sa sortie standard. Pour traiter le contenu d'un fichier, la redirection de l'entrée standard sur le fichier doit être explicite. Cette commande peut être utilisée sur des données non textuelles et ne connaît pas les expressions régulières. Selon les options et les paramètres qui lui sont passés, *tr* effectue trois types de transformation sur le flot d'entrée :

- transcodage : *tr* sans option : *tr string1 string2*
- suppression de certains caractères : option *-d* : *tr d string1*
- suppression de répétitions : option *-s* : *tr s string1*

Dans le premier cas, le transcodage, *tr* remplace chaque caractère du flot d'entrée appartenant à la chaîne *string1* par le caractère de même rang dans la chaîne *string2*. Dans toutes les utilisations de *tr*, la notation *a z* est autorisée pour représenter la suite des caractères de *a* à *z* (inclus) et les caractères non imprimables sont représentés par leur code ASCII en octal (\015 : le caractère <carriage-return>.

Les exemples suivants montrent le transcodage :

```
# remplacer "a" par "A", "," par ";" et "/" par "_"
xstra> tr 'a,/' 'A;_' <fich1 >fich2
# remplacer $ par F, () par {} et <tab> par <espace>
# <tab> vaut 011 en octal
xstra> tr '$()\011' 'F{} ' <fich1 >fich2
# transformation majuscules vers minuscules
xstra> tr 'A Z' 'a z' <fich1 >fich2
```

Dans le deuxième cas, suppression de certains caractères, *tr* supprime tout caractère du flot d'entrée appartenant à la chaîne *string1*. Voici un exemple très utile :

```
# supprimer les caractères <carriage return> et Ctrl Z
# ce qui convertit un fichier texte ASCII de
# DOS vers Unix
xstra> tr d '\015\032' <fich1 >fich2
```

Dans le troisième cas, suppression de répétitions de certains caractères, *tr* recherche dans le flot d'entrée toute séquence constituée de plusieurs exemplaires consécutifs d'un caractère appartenant à la chaîne *string1* et remplace cette séquence par un seul exemplaire du même caractère. Voici quelques exemples :

```
# remplacer plusieurs espaces consécutifs par un seul
xstra> tr  s ' ' <fich1 >fich2
# remplacer plusieurs <new line> consécutifs par un seul
xstra> tr  s '\012' <fich1 >fich2
# ce qui peut se combiner comme ici pour les séquences
# de <espace> ou <tab> ou <new line>
xstra> tr  s ' \011\012' <fich1 >fich2
```

Pour terminer, voici un exemple mettant en évidence l'élégance du mécanisme des tubes Unix permettant d'associer très simplement des briques de base. Le problème posé consiste à construire le dictionnaire alphabétique de tous les mots d'un texte. Les étapes du traitement sont les suivantes :

tr coupe le texte en mots,

sort trie ces mots par ordre alphabétique et élimine les doublons

Le résultat final est redirigé vers le fichier résultat.

```
# dictionnaire alphabétique de tous les mots d'un texte
xstra> tr  cs 'a zA Z' '\012' < text | sort  u > dico
```

14.6 EXERCICES

Exercice 14.6.1

Recherchez dans le fichier */etc/passwd* :

- L'entrée correspondant à votre login (solution indépendante du login).
- Le nombre de lignes contenant la chaîne de caractères : MICHEL ou michel.
- Le nombre de lignes contenant la chaîne de caractères : */home/*.

Exercice 14.6.2

Recherchez dans le fichier */etc/inetd.conf* :

- Le nombre de lignes contenant un point .
- Le nombre de lignes ne commençant pas par # .
- Toutes les lignes de plus de 50 caractères.
- Toutes les lignes de plus de 50 caractères (hors commentaires).

Exercice 14.6.3

Écrivez une commande *awk* permettant de lister le contenu d'un fichier sans les lignes vides et en les numérotant. Testez votre solution avec l'entrée standard.

Exercice 14.6.4

Le fichier annuaire.txt contient des adresses sous la forme suivante :
Prénom NOM Date-de-naissance Sexe Numéro-de-téléphone
Le fichier est supposé très long, et il peut contenir des erreurs comme dans l'exemple ci-dessous :

```
Georges TARTEMPION 12 2 1967 M 038898422524
Hortense HYABLANT 7 4 1944 F 0146872556
Adrien DUBOUCHON 27 11 1938 M 0154896372
Ludwig SCHMILLBLEERCK 28 12 1957 M 0258459887
Antonio VAVILDA 16 5 1937 M
Hughes CARPETTE 8 11 1958 M 0123568629
J'en ai assez de ce travail fastidieux!
Dagobert ELOY 14 7 1947 M 0225415487
      ligne vide
Anatole SUISSE 1 2 1965 n 4
Antonino SZWPRESWKY 16 5 8937 M 0298358745
```

Écrire les trois commandes *grep* permettant de :
- supprimer les lignes vides ;
- trouver les gens qui n'ont pas le téléphone ;
- trouver les lignes non conformes ;

Écrire une solution *awk* pour le dernier cas.

Exercice 14.6.5

Listez les noms de login de tous les utilisateurs de votre système qui n'ont pas de programme de démarrage (dans */etc/passwd*).

Exercice 14.6.6

Les messages électroniques SMTP sont constitués exclusivement de caractères ASCII imprimables (pas de caractères spéciaux). Leur structure est simple : un en-tête précède le corps du message.
L'en-tête commence toujours par une ligne : "From " (From suivi d'un espace). La fin de l'en-tête est toujours signalée par une ligne vide. Vous savez que votre boite d'entrée est le fichier */var/spool/mail/$LOGNAME*. Vous savez donc tout ce qu'il faut savoir pour écrire un shell script qui envoie sur sa sortie standard :
- le nombre de messages en attente dans votre boite d'entrée ;
- les champs "From: " et "Subject: " du dernier message.

Exercice 14.6.7

Améliorez le script précédent pour qu'il fasse son travail deux fois par minute.

Exercice 14.6.8

Améliorer le script précédent pour qu'il reste silencieux si aucun message n'est arrivé dans la période qui précède.

Exercice 14.6.9

Si vous travaillez dans l'environnement X11, lancez ce script en background dans une petite fenêtre dans un coin de votre écran. Il est supérieur au programme `xbiff` à deux points de vue :
- vous savez si le message qui vient d'arriver mérite le dérangement ;
- vous avez la satisfaction de l'avoir fait vous-même.

Exercice 14.6.10

Écrire un script qui lit un message sur son entrée standard et recopie :
- l'en-tête du message dans le fichier "./_header" ;
- le corps du message dans le fichier "./_body".

Un tel script est l'outil de base permettant d'automatiser le traitement des messages à l'arrivée. Écrire une solution `sed` et une solution `awk`.

Exercice 14.6.11

Chercher un motif dans un fichier avec `egrep`, c'est facile :

```
egrep 'ere' fichier
```

Remplacer une chaîne qui vérifie une expression régulière par une nouvelle chaîne, c'est facile aussi, mais moins, et c'est surtout long à taper, et il faut rediriger, ... Ce qu'il faudrait, c'est un sed simplifié "à la grep", nommé `remplace`, à utiliser dans les cas tous simples, comme par exemple :

```
remplace 'pays' 'paysage' *.txt
```

Ce qui signifie : dans tous les fichiers d'extension `.txt`, remplacer toute occurrence de la chaîne `pays` par `paysag`e.

Bien sûr, comme pour `egrep`, le premier argument est une expression régulière.

Chapitre **15**

Sécurité

La sécurité et la qualité des systèmes d'information sont des critères essentiels du fait de leur implication sur le plan économique. En effet, les évènements provoquant des pertes d'informations peuvent être nombreux : destruction accidentelle ou volontaire, vol du matériel ou des informations, dysfonctionnement du système ou des logiciels, virus, cheval de Troie, intrusion, espionnage, sabotage, … La liste est longue.

Les sinistres graves ont des origines multiples et complexes. Il s'agit d'incohérence des moyens de sécurité ou de l'inadéquation de ces moyens par rapport aux enjeux. Le coût des pertes peut être important pour ne pas dire inestimable lorsque les informations perdues ne peuvent plus être récupérées. La vie d'une société dépend aussi de sa sécurité informatique.

Parmi toutes les définitions de la sécurité informatique, nous avons retenu celle-ci : **un système est sécurisé si vous pouvez faire confiance en son comportement et s'il peut vous prémunir contre les pertes d'informations**.

Trois critères permettent de définir la sécurité d'un système d'information :

- La **disponibilité** : les données et ressources du système sont disponibles pour un utilisateur autorisé lorsqu'il en a besoin.
- L'**intégrité** : les données sont exactes et à jour. Les programmes exécutent exactement les fonctions décrites dans la documentation.
- La **confidentialité** : les données ne peuvent être accédées que par les personnes autorisées.

Le système d'exploitation Linux est un système multi-utilisateur. Par conséquent, les ressources d'une même machine seront partagées par plusieurs utilisateurs. De

plus, l'informatique est actuellement basée sur la communication et l'échange d'informations par les réseaux. **La sécurité** peut être divisée en deux parties : la **sécurité physique** et la **sécurité logique**. Néanmoins l'une n'a pas de sens sans l'autre.

La sécurité physique concerne la sécurité des locaux où sont installées les machines et les sauvegardes. Elle est souvent négligée. Les menaces physiques les plus courantes sont le vol, l'incendie, l'inondation, les intrusions, etc. Il existe un autre aspect de la sécurité physique : sur certaines machines Linux, toute personne ayant physiquement accès à la machine peut obtenir plus ou moins aisément les privilèges de l'administrateur (de root). De telles machines doivent être protégées physiquement sinon aucune sécurité logique ne sera utile.

La sécurité logique est la sécurité fournie par le système Linux et par les logiciels. Les menaces logiques peuvent être divisées en deux catégories : les infections informatiques (Cheval de Troie, bombe logique, virus, ver) et les intrusions.

Le cheval de Troie, comme son nom l'indique, est un programme qui se fait passer pour un autre programme.

La bombe logique est un programme à déclenchement différé qui va réaliser une action néfaste.

Le virus est un programme infectant un autre programme, qui pourra se reproduire lors du lancement du programme infecté et réaliser éventuellement une action néfaste.

Le ver est similaire au virus dans sa fonction de reproduction et son action néfaste, mais contrairement au virus il est capable de se reproduire tout seul et ne requiert pas l'exécution d'un programme. Le ver de l'Internet du 3/10/1988 est le plus connu.

L'**intrusion** consiste en l'utilisation illicite d'un accès au système. Ce cas est de plus en plus fréquent du fait de l'interconnexion des ordinateurs. En effet, par le réseau Internet des millions d'individus du monde entier peuvent tenter de pénétrer votre système.

Ce chapitre se limitera à la sécurité logique sous Linux. Il n'abordera pas les différents points de sécurité à respecter lors de l'administration d'une machine. Ce rôle est dédié à l'administrateur et nécessiterait un approfondissement des connaissances sur Linux ne faisant pas l'objet de cet ouvrage.

Cependant, la sécurité n'est pas exclusivement de la responsabilité de l'administrateur de la machine ou du responsable de la sécurité. Chaque utilisateur, par sa négligence, peut permettre l'intrusion d'un non-utilisateur de la machine et ainsi fragiliser l'édifice de la sécurité. N'oubliez jamais que **la sécurité est l'affaire de tous**.

Nous allons donc présenter quelques règles simples permettant à l'utilisateur de sécuriser son environnement et par conséquent de participer activement à la sécurité globale de la machine. Notez bien que la négligence d'un utilisateur peut avoir pour

conséquence l'intrusion du loup dans la bergerie. Bien que Linux puisse être bien sécurisé, cette intrusion amènera un risque important pour le bon fonctionnement de la machine et pour les données des autres utilisateurs.

15.1 LA CONNEXION

15.1.1 L'entrée en session

L'entrée en session est un élément essentiel de la sécurité concernant l'utilisateur. En effet, le seul et unique contrôle d'accès à une machine pour un utilisateur est le couple (**nom de login**, **mot de passe**). Une personne en possession de cette clé peut accéder à vos données (confidentialité), les modifier (intégrité) et accéder à toutes les ressources auxquelles vous avez accès (disponibilité). Cette clé doit donc rester strictement personnelle et confidentielle. Dans ce couple d'accès à une machine, le mot de passe est l'élément essentiel, le nom de login étant généralement votre propre nom. Le choix du mot de passe est donc primordial.

Le nom de login, et d'autres informations, se trouvent dans le fichier `/etc/passwd`, qui est accessible en lecture à tous les utilisateurs. Le mot de passe est dans un fichier sécurisé (ce fichier n'est pas accessible en lecture par les utilisateurs). Il est chiffré par une méthode non inversible, c'est-à-dire qu'il est impossible de retrouver un mot de passe à partir de sa valeur chiffrée. Lors d'une connexion, l'authentification est réalisée par comparaison du mot de passe actuel chiffré avec le mot de passe chiffré contenu dans un fichier système.

Si votre connexion est réalisée *via* le réseau, il est très imprudent d'utiliser `telnet` : le mot de passe est transféré en clair sur le réseau. Il est obligatoire d'utiliser `ssh`. Dans toutes les distributions récentes de Linux, le serveur `telnet` n'est pas activé.

15.1.2 Comment protéger votre mot de passe contre le piratage ?

Le chiffrement du mot de passe n'étant pas inversible, un individu mal intentionné ne peut trouver votre mot de passe que du fait de votre négligence. Ainsi certaines règles doivent être observées :

- **Il ne faut jamais noter son mot de passe**. Après avoir trouvé un **bon** mot de passe, ne le notez pas sur la console, sous le clavier, dans votre agenda, sur une étiquette dans un tiroir de votre bureau… Un bon mot de passe doit être très difficile à deviner, mais très facile à mémoriser. Il peut être tentant d'écrire son mot de passe ainsi que son login dans des fichiers de connexion automatique : n'enregistrez jamais votre mot de passe en clair dans un fichier de connexion automatique (paragraphe 13.2.6).
- **Il ne faut jamais partager son mot de passe avec un autre utilisateur**. On peut être tenté de donner son mot de passe pour partager des données avec un autre utilisateur, ou lui permettre de copier ces données. Le partage de l'accès au système est toujours synonyme de dilution des responsabilités. Le système

d'exploitation Linux permet, grâce aux protections des fichiers et à la notion de groupe, de partager des données dans de bonnes conditions de sécurité sans pour autant diminuer la facilité d'accès.

- **Le mot de passe doit être original**. Un individu cherchant à connaître votre mot de passe essaiera les informations personnelles vous concernant : nom de la machine, votre prénom et ceux de votre entourage, le nom de vos animaux préférés, information de votre société, date de naissance,……Cette première approche ayant échoué, l'intrus utilisera un logiciel avec dictionnaire permettant de découvrir (**craquer**) votre mot de passe. Ces logiciels et dictionnaires sont du domaine public (accessibles par tout utilisateur connecté au réseau Internet) et peuvent être facilement récupérés et mis en œuvre. Ils comportent généralement : les noms communs, noms propres, prénoms, le tout dans plusieurs langues. Le logiciel essaiera toutes les combinaison de un, deux, trois, quatre, voire cinq caractères pour votre mot de passe. Un tel logiciel permet souvent de découvrir près de 20 % des mots de passe.
- **Attention à votre environnement lors de la saisie de votre mot de passe**. Vous faites une démonstration avec saisie de votre mot de passe devant un auditoire très attentif. Il est dans ce cas très facile de noter la position des doigts sur le clavier et de trouver ainsi le mot de passe : changez votre mot de passe immédiatement après votre démonstration.

15.1.3 Choix du mot de passe

Le choix du mot de passe est important. Il faut donc prendre le temps de le choisir. Si l'administrateur de la machine vous impose un choix immédiat de mot de passe, prenez-en un facile, réfléchissez, puis à l'aide de la commande `passwd` modifiez-le. Certaines règles pour un bon choix de mot de passe doivent être respectées :

- Ne jamais choisir un mot de passe de moins de sept caractères. En général, seuls les huit premiers caractères sont utilisé sur Linux.
- Combiner des caractères alphabétiques majuscules et minuscules, des caractères numériques et des signes de ponctuation.
- Comme indiqué au paragraphe précédent, éviter les mots de passe pouvant faire partie de dictionnaires.
- Il doit être facile à mémoriser pour éviter de le noter.
- Il faut changer de mots de passe dès que l'on a le moindre doute.
- Éviter les exemples de mots de passe donnés dans les cours ou documents.
- Enfin on adoptera une fréquence de changement de mot de passe raisonnable.

Exemples

Mauvais mots de passe :

racine, root, Paris, 1234567, linux, xstra

Exemples

Bons mots de passe :
Malvslr, Beau!C, 3kmAPied, vlmpdJP!

15.1.4 Gestion des connexions

En tant qu'utilisateur, il vous est possible, par des contrôles très simples, de participer à la sécurité de votre machine. Lors de votre connexion, il est intéressant d'afficher à l'écran l'heure de début et de fin de votre dernière connexion (`last login`). Un rapide contrôle de ces informations permettra de détecter une intrusion, c'est-à-dire une connexion avec votre nom à une date et une heure où vous étiez absent. Il est également possible, à l'aide de l'une des commandes `who`, `w`,… de contrôler si un collègue malade ou en déplacement est connecté à la machine. Dans tous ces cas, il faut immédiatement avertir l'administrateur du système.

15.1.5 Sortie de session

L'une des grandes négligences des utilisateurs est de quitter leur poste de travail « juste pour quelques minutes » qui peuvent parfois se transformer en heures (café, téléphone dans un autre bureau,…) et de laisser ainsi un terminal connecté. A partir de ce moment il est très facile à un individu indélicat d'utiliser votre session pour copier un logiciel (cheval de Troie, virus, …) dans un de vos répertoires, de détruire vos fichiers ou d'exécuter d'autres actions néfastes.

Il faut pour éviter cela toujours se déconnecter avec la commande `exit` **et attendre le message de** `login` **qui indiquera l'attente d'une nouvelle connexion avant de partir**.

Afin d'éviter une sortie de session, il est également possible de bloquer sa console alphanumérique à l'aide de la commande `lock` et sa sessionl X11 à l'aide de la commande `xlock`.

Un mécanisme de « logout » automatique est mis à votre disposition par le shell. Le temps d'inactivité avant la sortie automatique de session est programmable à l'aide de la variable TMOUT pour le Bash. Ce mécanisme ne devrait intervenir qu'exceptionnellement en cas d'étourderie ou de force majeure. Il ne dispense absolument pas l'utilisateur de terminer volontairement une session.

Exemple : cheval de Troie

Couramment utilisé sur les terminaux en libre service, le cheval de Troie est un programme lancé par un utilisateur mal intentionné. Il simule sur le terminal l'invite login du système et capture le nom de login et le mot de passe d'un nouvel arrivant, les sauvegardes dans un fichier, donne le message "invalid login or passwd" puis redonne la main au programme `login` du système. L'utilisateur pensera avoir commis une erreur de frappe dans son mot de passe. Un nouvel essai lui permettra de se connecter normalement et il oubliera l'incident.

La solution consiste à donner volontairement un mot de passe erroné au premier essai de connexion. Une autre solution consiste à changer son mot de passe après un tel incident.

Remarque

Pour les utilisateurs de machines Linux, il est extrêmement dangereux de redémarrer la machine (couper le secteur, *reset*…). En effet, un arrêt brutal de Linux risque de rendre incohérents les systèmes de fichiers (*file systems*). Ainsi, si vous êtes « **planté** », c'est-à-dire si l'une de vos applications bloque l'écran, vous pouvez d'abord essayer de détruire le programme à l'aide de la combinaison de touches <ctrl-c>. En cas d'échec, contrairement au DOS, **n'éteignez surtout pas l'ordinateur**. Une solution possible consiste à ouvrir un écran virtuel par la combinaison de touches *<Alt Ctrl F2>*. Dans cet écran, vous pouvez vous logger et tuer les processus fautifs (*kill KILL PID*) pour résoudre le problème. Pour éteindre la machine, utiliser la commande *shutdown*.

15.2 PROTECTION DES FICHIERS ET DES RÉPERTOIRES

La protection des fichiers et des répertoires contribue à la sécurisation de Linux en permettant le contrôle de l'accès aux fichiers. La signification et le changement des permissions des fichiers à l'aide des commandes *ls l* et *chmod* sont décrits au chapitre 4.

15.2.1 Protection par défaut

La protection par défaut attribuée aux fichiers et aux répertoires lors de leur création est définie par la commande *umask masque*. La valeur de masque est en standard 022, impliquant les permissions suivantes : `rwx r x r x` pour les répertoires et fichiers exécutables et `rw r r` pour les fichiers lors de leur création. Ainsi tous les utilisateurs peuvent accéder à vos données sans toutefois pouvoir les modifier ou les supprimer.

Il est possible d'augmenter la confidentialité en prenant un masque égal à 066 (`rwx x x`) pour les répertoires et les fichiers exécutables, (`rw `) pour les fichiers. Cette option permet une protection des fichiers contre la lecture pour tous les utilisateurs du système. Elle permet par contre l'accès à vos répertoires. Les membres du groupe ou les autres pourront ainsi traverser vos répertoires pour accéder à un sous-répertoire sur lequel ils possèdent des droits de lecture ou d'écriture.

Le *masque* égal à 077 (`rwx `) permet la meilleure protection car il interdit à tous les utilisateurs, même à ceux de votre groupe, l'accès à vos répertoires et fichiers. C'est un cas extrême d'utilisateur isolé.

Le choix du masque est dicté par le niveau de sécurité souhaité. Un bon compromis entre souplesse et sécurité est de choisir un **masque de 026** (`rwx r x x`) pour

les répertoires et fichiers exécutables, (`rw r `) pour les fichiers permettant aux membres du groupe d'accéder à l'ensemble des données sans toutefois les modifier et permettant aux autres de traverser vos répertoires pour atteindre un fichier ou sous-répertoire que vous aurez autorisé en lecture ou écriture.

Exemple

HOME (`rwx r x x`)

Commun	personnel	projet1
(rwx r-x r-x)	(rwx --- ---)	(rwx r-x --x)

Vérifiez toujours si la commande *umask* a été initialisée à la valeur que vous souhaitez. Pour plus de sécurité vous pouvez initialiser la valeur de *umask* dans l'un de vos fichiers personnels d'initialisation. Cela annulera la valeur par défaut mise par l'administrateur de la machine.

Le répertoire HOME de chaque utilisateur peut avoir les permissions :

`rwx`	Pour un utilisateur isolé.
`rwx r x` ou `rwx x`	Pour un utilisateur qui travaille en groupe.
`rwx rwx rwx`	Ces permissions sont à proscrire. Tout utilisateur du système peut détruire vos fichiers, détourner votre courrier (*.forward*), commettre des malveillances qui vous seront attribuées, vous empêcher de vous connecter, etc.

15.2.2 Travail en groupe

En cas de souhait de partage de fichiers il faut ni donner son login et mot de passe, ni permettre à tous les utilisateurs d'accéder à vos données (*rwx rwx rwx*). Le partage de fichiers peut être réalisé par la création de groupes. Il faut demander à l'administrateur de la machine de créer un groupe (*/etc/group*) et d'y enregistrer tous les membres de ce groupe.

Un utilisateur peut appartenir à plusieurs groupes, ce qui procure une très grande souplesse. Nous avons vu au paragraphe 4.5 que la philosophie BSD diffère sur ce point de celle du système V. Dans l'usage courant et du point de vue de la sécurité, le résultat est le même : si le fichier *fich1* appartient au groupe *group1* et si je suis moi-même membre de ce groupe, les permissions de groupe de *fich1* me sont applicables.

Il suffit pour cela de créer un répertoire appartenant à ce groupe avec les protections *rwx rwx x*. Tous les membres du groupe pourront créer et supprimer des fichiers. L'accès aux données dépendra évidemment des permissions des fichiers.

Ainsi la création d'un groupe est utile pour :

- séparer les utilisateurs en groupes d'intérêt commun,
- mettre en commun un ensemble de fichiers accessibles par les membres du groupe,
- limiter l'accès à un ensemble de commandes à un groupe d'utilisateurs.

La meilleure utilisation de cette notion de groupe est celle dans laquelle le nombre et la composition des groupes reflètent exactement l'organisation du travail.

15.2.3 Fichiers sensibles

Les fichiers d'initialisation qui vous appartiennent doivent obligatoirement avoir la protection *rw* ou *r* . Il s'agit principalement des fichiers standard d'initialisation : *.bash profile*, *.bash login*, *.bashrc*, *.bashscript* et *.bash logout* pour le Bash, *.exrc* pour l'environnement de travail de *vi*, *.rhosts*, *.forward* pour le travail en réseau, et vos fichiers personnels d'initialisation lancés par les précédents. N'inscrivez jamais votre mot de passe en clair dans un fichier *.netrc* ou *.ncftprc*.

Un programme modifiant ou supprimant des informations personnelles ne doit pouvoir être exécuté que par son propriétaire. Il faut aussi savoir que l'administrateur du système (root) peut lire tous vos fichiers ainsi que votre boîte aux lettres.

15.2.4 Audit de votre arborescence

Un audit permet de rechercher les anomalies dans votre arborescence. La commande *find* permet de réaliser un audit en balayant votre arborescence et en recherchant des informations de manière sélective :

- Recherche des fichiers et répertoires ne vous appartenant pas :

```
xstra> find $HOME ! user $LOGNAME print
```

- Recherche des répertoires avec permissions différentes de celles définies par *umask* :

```
xstra> find $HOME type d ! perm $((777 $(umask))) \
       exec ls ld {} \;
```

- Recherche des fichiers exécutables par tous les utilisateurs :

```
xstra> find $HOME type f exec ls l {} \; \
       |grep '^.{9}x'
```

15.3 SAUVEGARDE

L'utilisateur doit sauvegarder régulièrement l'ensemble de ses données. Un programme de sauvegarde est présenté au chapitre 11. Les sauvegardes sont fondamentales pour récupérer des informations perdues après un incident. La fréquence des sauvegardes dépend de l'importance que l'on accorde à ses données.

Il existe plusieurs stratégies de sauvegarde. Il est possible de réaliser une sauvegarde complète de l'ensemble des données. Dans ce cas le temps de sauvegarde est long et le nombre de supports probablement important. Il est également possible de réaliser une sauvegarde incrémentale, qui permet de ne sauvegarder que les fichiers modifiés depuis la sauvegarde précédente. Il ne faut pas oublier que vos sauvegardes peuvent également subir l'effet du temps et ainsi être détruites. Pour cela vérifiez régulièrement l'ensemble de vos sauvegardes.

Un autre point important est de garder au moins une sauvegarde de la totalité de vos données hors du site de travail, dans des locaux physiquement distincts et protégés. En effet, un accident (incendie, dégât des eaux…) détruira non seulement vos données sur la machine mais également vos sauvegardes.

N'oubliez jamais que vos sauvegardes peuvent être relues sur un autre ordinateur. Il ne sert donc à rien de protéger votre ordinateur contre le piratage en le mettant dans une salle à code d'accès et de ranger les sauvegardes dans une armoire accessible à tout le monde.

15.4 VARIABLES D'ENVIRONNEMENT

La variable PATH décrit l'ordre dans lequel le shell recherche la commande entrée par l'utilisateur. Pour éviter qu'un programme parasite portant le même nom qu'une commande Linux standard ne soit lancé à la place de cette commande :

- vérifiez que la variable PATH ne contient que des répertoires sûrs,
- placez toujours les répertoires systèmes avant les répertoires personnels dans la variable PATH,
- placez le répertoire courant à la fin de PATH (ou supprimez-le).

Exemple

```
xstra> echo $PATH
/bin:/usr/bin:/usr/local/bin:/home/xstra/bin:.
xstra>
```

Il est également conseillé de ne jamais modifier la variable LOGNAME contenant le nom de l'utilisateur et la variable SHELL qui indique le shell utilisé.

Il est possible à un autre utilisateur de la machine d'envahir votre écran par des messages à l'aide de la commande *write*. La commande *mesg n* permet d'inhiber l'envoi de messages par *write* sur votre écran.

15.5 EXERCICE

Exercice 15.5.1

Vérifiez que les permissions de votre répertoire d'accueil et votre umask sont conformes aux relations de confiance que vous entretenez avec les autres utilisateurs. Faites cette vérification pour vos répertoires principaux et vos fichiers de démarrage.

Chapitre **16**

L'interface graphique X11

X Window est un environnement graphique de fenêtrage comprenant une interface graphique utilisateur (GUI) et une interface de programmation d'applications (API). C'est un moyen de communication entre la machine et l'utilisateur, non pas en mode texte, c'est-à-dire caractère par caractère, mais en mode graphique ou bitmap, c'est-à-dire pixel par pixel (un pixel est un point lumineux sur un écran).

Pour X Window, les caractères d'un texte sont des graphiques parmi d'autres. Un pointeur représenté par une flèche, une croix ou tout autre signe, permet comme son nom l'indique, de pointer un pixel. Ce pointeur est déplacé à l'aide d'une souris.

L'interface graphique utilisateur X Window peut s'apparenter à d'autres interfaces graphiques telles que l'interface Macintosh de Apple ou Windows de Microsoft pour les PC. Ces deux interfaces graphiques ont deux points communs : leur fonctionnement en mode mono-utilisateur et leur lien étroit avec le matériel pour lequel elles ont été développées. X Window est en revanche une interface graphique pour les stations de travail. Elle est indépendante du matériel et basée sur le réseau. Le but des concepteurs de X Window était de délocaliser calcul et affichage graphique : un graphique créé sur une machine peut être visualisé sur la même machine ou sur une autre machine du réseau, indépendamment du matériel du constructeur, et du système d'exploitation.

X11 a été développé à partir de 1984 au Massachusetts Institute of Technology (MIT) dans le cadre du projet Athena. En 1988, le MIT créa un consortium comprenant les plus importants constructeurs d'ordinateurs afin de veiller à la standardisation X Window, encore appelé X11. La dernière version est X11R6 (X11 release 6). Dans la suite du texte, nous n'emploierons plus que le terme X11.

L'élément de base de X11 est la **fenêtre** (window). Une fenêtre est un rectangle sur l'écran. X11 permet de gérer sur un même écran de multiples fenêtres qui peuvent se cacher et se chevaucher. Il permet également l'affichage de texte dans un grand nombre de polices de caractères différentes (fontes). Le **gestionnaire de fenêtres** (**window manager**) est un programme indépendant de X11 qui gère l'aspect des fenêtres et leurs manipulations. Il existe plusieurs gestionnaires de fenêtres.

X11 étant très spartiate, il est apparu depuis quelques années la notion d'**environnement de travail**, permettant d'unifier l'interface graphique entre les différents ordinateurs et de proposer un ensemble standard d'applications et d'utilitaires bureautiques. L'environnement de travail est basé sur X11. Depuis 1996, existe dans le monde des stations de travail, un environnement de travail commun appelé « **Common Desktop Environment** » (**CDE**). Dans le monde Linux, il en existe plusieurs, nous ne citerons que les deux plus importants à cette date : **GNOME** et **KDE**. La présentation de chaque environnement de travail peut faire l'objet d'un ouvrage. De plus il existe, soit dans la distribution Linux comme aide en ligne, soit disponible sur Internet, une documentation en français d'initiation et d'utilisation de chaque environnement de travail.

L'indépendance matérielle de X11 se fonde sur la séparation entre génération du graphique et visualisation de ce dernier. Cette séparation s'intègre dans le concept de client-serveur que nous avons présenté au paragraphe 13.1.4 et sur lequel nous allons revenir maintenant dans le cadre de X11.

Le paragraphe X11 de cette ouvrage se limitera à une présentation des concepts et de l'utilisation de base de X11. Il décrira quelques caractéristiques communes aux différents environnements de travail.

16.1 PRÉSENTATION GÉNÉRALE DE X11

16.1.1 Le concept client–serveur

L'une des idées novatrices de X11 est d'être conçu suivant une architecture distribuée appelée **client–serveur** (Fig. 16.1).

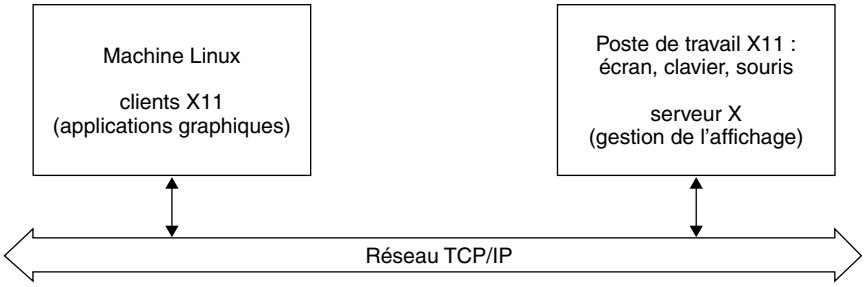


FIGURE 16.1. UN EXEMPLE D'UTILISATION DE X11 EN MODE CLIENT–SERVEUR : LE POSTE DE TRAVAIL ET LA MACHINE LINUX SONT DISTINCTS.

Le **serveur X** est un programme qui fournit des services graphiques. Il prend en charge la gestion des périphériques d'entrée (clavier et souris) et de sortie (écran graphique). Cet ensemble est appelé un affichage (**display**). Il faut un serveur X par affichage ; néanmoins celui-ci peut gérer plusieurs écrans (**screen**), si le matériel le permet. Le serveur est dépendant du matériel et exécuté au niveau local, mais fournit des services normalisés indépendants du matériel.

Le **client** X est un programme applicatif qui ne se préoccupe ni des affichages ni de la gestion des entrées. Il fait appel au serveur X pour lui confier cette tâche. Le client X va communiquer avec le serveur X.

La communication du client avec le serveur est réalisée par l'intermédiaire d'un protocole spécifique (**protocole X11**). Le client est ainsi indépendant de la machine sur laquelle se font les affichages. Il adresse des requêtes (requests) au serveur qui les exécutera (demande de création d'affichage, de destruction de fenêtre…). Cette communication est réalisée de manière asynchrone, c'est-à-dire que le serveur X traite les requêtes dans leur ordre d'arrivée. Les clients n'attendent pas forcément une réponse du serveur.

Le dialogue client-serveur peut être local ou distant par l'intermédiaire d'un réseau (Ethernet par exemple). Ainsi le processus client peut être exécuté sur une machine autre que celle du serveur. De plus, plusieurs clients s'exécutant sur des machines différentes peuvent être connectés simultanément à un même serveur et donc utiliser le même affichage.

N'importe quel client peut communiquer avec n'importe quel serveur, sous réserve de respecter le protocole X11 et d'en avoir l'autorisation (voir le paragraphe 16.3.4). Ceci permet l'utilisation de matériels informatiques différents. Cette particularité de X11 permet de tirer parti des possibilités de l'informatique répartie. Ainsi, un programme peut s'exécuter de façon répartie : le calcul est réalisé sur une machine A qui délègue ensuite les tâches de gestion de l'affichage à une autre machine B sur laquelle est implanté le serveur X.

Le client est constamment informé d'une action d'un périphérique, de l'affichage (déplacement de la souris, état du clavier…) par des **événements**.

16.1.2 Le gestionnaire de fenêtres

La fenêtre constitue l'élément de base dans X11. Les fenêtres sont organisées de façon hiérarchique, comme la gestion des fichiers. La fenêtre de base, appelée **fenêtre racine** (**root window**), occupe la totalité de l'écran et ne peut à aucun moment être supprimée. Un client va créer une ou plusieurs fenêtres filles de la fenêtre racine.

Néanmoins, l'aspect des fenêtres et leurs manipulations ne sont pas réalisés par le serveur X mais pris en charge par un client particulier : le **gestionnaire de fenêtres** (**window manager**). De plus le gestionnaire de fenêtres prend en charge un menu pour la fenêtre racine. Le gestionnaire de fenêtres est l'intermédiaire entre l'utilisateur et X11. Il participe à la convivialité. Plusieurs gestionnaires de

fenêtres différents ont été développés : **twm** (Tom's Window Manager) livré en standard avec X11, **mwm** (Motif Window Manager de l'OSF), **dtwm** (gestionnaire de fenêtre basé sur mwm et utilisé par CDE), **enlightenment** (GNOME), **kwm** (KDE).

Les éléments d'utilisation d'un gestionnaire de fenêtres présentés dans cet ouvrage sont communs à tous les gestionnaires de fenêtres.

16.1.3 L'environnement de travail

L'environnement de travail est indépendant du gestionnaire de fenêtres bien qu'il existe toujours un gestionnaire de fenêtres conseillé par environnement de travail. Nous présentons dans cet ouvrage quelques caractéristiques communes aux deux environnements graphiques actuellement les plus utilisés dans le monde Linux : GNOME et KDE.

Le but de l'environnement de travail est de fournir une interface agréable et facile à utiliser, homogène pour l'ensemble des applications. L'environnement de travail inclus généralement un tableau de bord permettant de lancer les applications et d'afficher des informations, un bureau sur lequel sont déposés les fichiers et icônes. L'environnement de travail est configurable à l'aide d'outils graphiques. De plus il utilise un système de gestion des sessions ce qui lui permet de conserver les paramètres de configuration pour chaque utilisateur. Une session est l'ensemble des applications, des paramètres et des ressources disponibles sur le bureau de l'utilisateur. Lorsqu'un utilisateur se connecte pour la première fois, une session initiale est créée par défaut. Lorsque l'utilisateur quitte une session, le gestionnaire de sessions sauvegarde automatiquement tout l'environnement de la session. Cela signifie que le bureau lors d'une prochaine connexion sera restauré dans le même état que lorsque l'utilisateur s'est déconnecté la dernière fois.

Au démarrage de l'environnement de travail apparaît à l'écran le **bureau,** puis au milieu de l'espace de travail un pointeur représentant la position de la souris (un déplacement de la souris entraîne un déplacement du pointeur), ainsi qu'une ou plusieurs zones rectangulaires appelées fenêtres**.** Il est possible d'avoir plusieurs fenêtres ouvertes sur l'écran comme si l'on avait plusieurs dossiers sur son bureau. Une fenêtre représentera une application, par exemple l'émulation d'un terminal connecté à la machine (`xterm`, `konsole`, `kvt` pour KDE, `gnome-terminal` pour GNOME), un gestionnaire de fichiers ou bien une horloge (`xclock`). Il apparaît également le **tableau de bord**. Il s'agit d'une fenêtre particulière, généralement affichée dans la partie inférieure de l'écran. Elle contient des indicateurs, des icônes de commandes, des menus secondaires.

Il est possible de modifier à tout moment la taille des fenêtres et de les déplacer. De plus, si vous avez trop de fenêtres sur votre bureau, vous pouvez réduire une fenêtre en icône (représentation graphique de la fenêtre réduite). L'icône s'affiche au bas de votre bureau jusqu'à ce que vous la retransformiez en fenêtre. Ces tâches sont prises en charge par le gestionnaire de fenêtres.

16.1.4 Configuration matérielle d'un serveur X11

La configuration matérielle traditionnelle d'une machine supportant un serveur X est une **station de travail**, un PC sous Linux par exemple, avec un écran graphique bitmap de résolution au moins égale à 1024x700 pixels, avec un clavier et une souris.

L'émulateur X11, programme s'exécutant sous Windows sur un PC et qui est en fait un serveur X11. Il permet à un utilisateur d'exécuter un programme sur une station de travail Linux et d'afficher le résultat sur l'écran de son PC. Le PC est serveur X11 de l'application (client X11) s'exécutant sur une machine Linux.

16.1.5 Démarrage de X11

Quand vous démarrez Linux, deux situations peuvent se présenter. Soit vous êtes en mode texte et obtenez une invite, soit vous avez une fenêtre d'identification (login) graphique. Dans le premier cas, vous devez vous connecter (vous identifier auprès du système) et ensuite taper la commande *startx*. Si l'installation est réussie, X11 est lancé et le bureau de l'environnement de travail devrait apparaître après quelques secondes d'initialisation. Si une fenêtre d'identification (login) graphique se présente, alors seuls votre nom d'utilisateur et votre mot de passe sont nécessaires. Dans ce cas X11 et votre environnement de travail doivent démarrer sans aucune intervention.

16.1.6 Arrêt de X11 - arrêt de l'ordinateur

Il est possible de quitter X11 de plusieurs manières différentes qui dépendent parfois des distributions et de votre environnement de travail. Le plus souvent, vous trouverez sur votre bureau un menu arrêt ou sortie (logout en anglais) qui vous permettra de choisir entre une fin de session ou un arrêt de votre ordinateur. De plus il est possible de basculer du fonctionnement X11 vers un mode de fonctionnement terminal en utilisant la combinaison de touches `<Alt Ctrl F1>` à `<Alt Ctrl F6>`. Pour revenir à X11 utilisez la combinaison de touches `<Alt Ctrl F7>`. Il est possible d'arrêter (tuer) le serveur X11 en saisissant la combinaison de touches `<Alt Ctrl backspace>`.

Remarque

Il est fortement déconseillé d'arrêter votre ordinateur en coupant le secteur. L'arrêt de Linux ne peut être réalisé proprement que par la commande *shutdown now* ou en utilisant le menu arrêter l'ordinateur de votre environnement de travail.

Une autre solution consiste à basculer en mode terminal et à utiliser la combinaison de touches `<Alt Ctrl sup>` qui permet de redémarrer proprement votre ordinateur.

16.2 UTILISATION DE X11

16.2.1 Emploi de la souris

Toutes les actions sous X11 sont réalisées au moyen de la souris. Nous allons décrire les techniques fondamentales d'emploi de la souris :

Pointer (point)	Déplacer la souris jusqu'à ce que le pointeur atteigne l'élément visé.
Cliquer (click)	Appuyer sur l'un des boutons de la souris et le relâcher, sans déplacer la souris entre-temps.
Cliquer deux fois (double click)	Cliquer rapidement deux fois de suite avec l'un des boutons de la souris.
Glisser (drag)	Appuyer sur l'un des boutons de la souris et le maintenir enfoncé en déplaçant la souris. Nous dirons : glisser le bouton gauche…

X11 peut être utilisé avec une souris ayant deux ou trois boutons. Pour les fonctions nécessitant le bouton central, il est possible d'obtenir la même action avec une souris à deux boutons en enfonçant les deux boutons simultanément.

16.2.2 La fenêtre

La figure 16.2 page 226 représente une fenêtre X11. L'aspect et les fonctionnalités des bordures de la fenêtre X11 dépendent du gestionnaire de fenêtres. Voici le détail des différents éléments qui la composent :

A Case du menu système — Si vous cliquez sur cette case, un menu apparaît vous proposant un certain nombre de manipulations sur la fenêtre.

B Curseur de sélection

C Barre de titre — La barre de titre affiche l'identification de la fenêtre. Si plusieurs fenêtres sont ouvertes dans l'espace de travail, la barre de titre de la fenêtre active a une couleur différente des autres.

D Titre de la fenêtre

E Espace de travail dans la fenêtre

F Bordure de la fenêtre — Elle correspond au périmètre de la fenêtre. Il est possible de modifier la taille de la fenêtre en faisant glisser la souris sur la bordure. Des

flèches apparaissent alors pour indiquer les sens possibles de déplacement.

G Case de réduction Cliquer sur cette case réduit la fenêtre en icône.

H Case d'agrandissement Cliquer sur cette case agrandit la fenêtre à la taille de l'écran.

I Coin de la fenêtre Le coin de la fenêtre permet, selon le même principe que les bordures, de réduire ou d'agrandir la fenêtre.

J Pointeur de souris

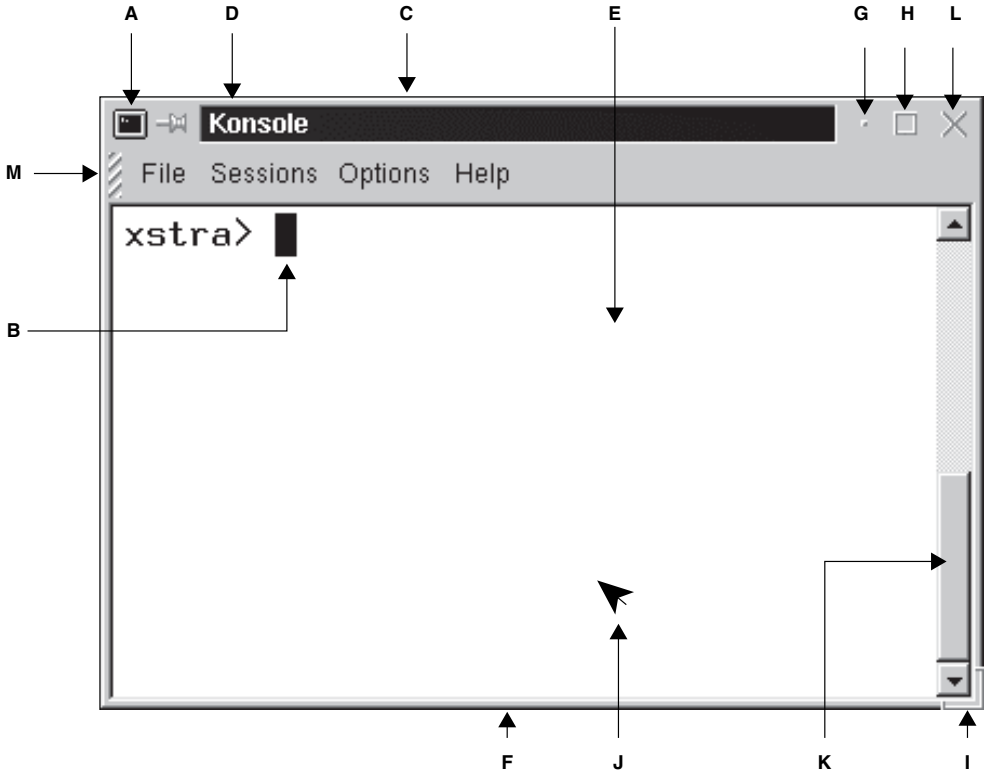


FIGURE 16.2. UNE FENÊTRE X11 SOUS LE GESTIONNAIRE DE FENÊTRE KWM.

K Barre de défilement (ascenseur) Le déplacement de cette barre permet de visualiser l'information non visible.

L Case fermeture Cliquer sur cette case permet de fermer la fenêtre et quitter l'application.

M Barre menu Barre de menu.

16.2.3 Manipulation des fenêtres

C'est le client gestionnaire de fenêtres (**window manager**) qui permet de déplacer, réduire, redimensionner les fenêtres. Pour activer une fenêtre, vous la pointez puis vous cliquez avec le bouton gauche de la souris. Lorsqu'une fenêtre devient active, son cadre change de couleur. Sélectionnez une fenêtre émulateur de terminal à l'aide du tableau de bord en KDE ou GNOME. Puis exécutez la commande `xterm geometry 80x24 0 0 sb&`. Cette commande permet de créer une fenêtre émulation de terminal dans le coin bas droit de votre espace de travail.

a) Déplacer la fenêtre

Pour déplacer une fenêtre, ou une icône, procédez comme suit :

> Pointez la barre de titre de l'élément à déplacer et faites-le glisser jusqu'au nouvel emplacement. Le déplacement de la fenêtre est représenté par le déplacement de son contour. Après avoir placé la fenêtre à l'endroit choisi, relâchez le bouton de la souris.

b) Redimensionner la fenêtre

Vous pouvez aussi changer les dimensions d'une fenêtre. Pour redimensionner une fenêtre, procédez comme suit :

> Pointez la bordure (ou le coin) de la fenêtre et faites-la glisser jusqu'à ce que la fenêtre ait la taille voulue, puis relâchez le bouton de la souris.
>
> Si vous faites glisser une bordure, seul est redimensionné le côté de la fenêtre que vous faites glisser. Si vous faites glisser un coin, les deux côtés de l'angle changent de dimension en même temps.

Le gestionnaire de fenêtres modifie la taille de la fenêtre mais c'est le client de la fenêtre qui doit gérer la nouvelle dimension de cette dernière.

c) Agrandissement et réduction d'une fenêtre

Lorsque la fenêtre encombre votre bureau, il est possible, à l'aide de la case de réduction, de la réduire en icône. Pour agrandir ou réduire, procédez comme suit :

Pointez la case de réduction de la fenêtre que vous souhaitez réduire, et cliquez.
Pour agrandir une icône (voir le paragraphe 16.2.4), il suffit de cliquer deux fois sur l'icône souhaitée et cette dernière redeviendra la fenêtre avec sa taille initiale et la position qu'elle occupait avant toute réduction.

Pointez la case d'agrandissement de la fenêtre que vous souhaitez agrandir.
Pour que la fenêtre reprenne sa taille initiale, il suffit de cliquer à nouveau sur la case d'agrandissement.

d) Manipulation de la barre de défilement (ascenseur)

La fenêtre ne visualise qu'une partie de l'information contenue dans une zone mémoire tampon (buffer). L'ascenseur (ressource `sb`) représente graphiquement la position de la fenêtre actuelle dans le buffer et peut être déplacé avec la souris (**scrolling**). Pour faire défiler, procédez comme suit :

Pointez la souris sur la barre de défilement.
Si vous cliquez avec le bouton de droite, le défilement du contenu de la fenêtre se fera vers le bas.
Si vous cliquez avec le bouton de gauche, le défilement du contenu de la fenêtre se fera vers le haut.
Si vous cliquez sur le bouton central, vous pouvez, en faisant glisser la souris, déplacer le texte de la fenêtre vers le haut ou vers le bas.

e) Fermer la fenêtre

Pour quitter une application, c'est-à-dire fermer la fenêtre correspondante, il suffit de choisir la commande `close` en cliquant sur **la case du menu système.**

16.2.4 L'icône

Lorsque l'espace de travail est encombré par plusieurs fenêtres, il est possible de réduire une fenêtre en **icône**. Une icône est une petite image graphique représentant la fenêtre. Cette icône est mise dans la barre supérieure de votre écran en KDE et dans le tableau de bord en GNOME. Un programme en cours d'exécution dans une fenêtre continuera à être exécuté lorsque la fenêtre est réduite en icône.

16.3 LES CLIENTS X11

16.3.1 Lancement d'un client

Il est possible de lancer un client X11 (une application qui fait appel au protocole X11), à partir d'une fenêtre émulation de terminal. La syntaxe est la suivante :

```
client [ ressources] &
```

Un client est exécuté de préférence en tâche d'arrière-plan, ce qui permet une autre action avec la fenêtre d'émulation de terminal. Cela n'est pas une obligation, et vous pouvez lancer un client en avant-plan, c'est-à-dire sans le caractère *&* en fin de ligne.

Chaque client dispose de *ressources* qui permettent de contrôler et personnaliser son apparence. Un certain nombre de ces *ressources* sont communes (voir le paragraphe 16.3.3). La *ressource* débute par le caractère **-**, suivi du nom de la ressource et de sa valeur si elle existe. Le nom de la ressource et sa valeur sont séparés par un espace. En exécutant le client X11 avec l'option --*help*, la liste des ressources de ce dernier est affichée à l'écran.

Exemple

Lancement du client émulation de terminal avec le fond de la fenêtre en bleu (*bg*) et le texte en rouge (*fg*) ainsi que la barre de défilement (*sb)* :

```
xstra> xterm  bg blue  fg red  sb &
xstra>
```

Liste des ressources pour le client xclock.

```
xstra> xclock   help
Usage: xclock [ analog] [ bw <pixels>] [ digital]
       [ fg <color>] [ bg <color>] [ hd <color>]
       [ hl <color>] [ bd <color>]
       [ fn <font_name>] [ help] [ padding <pixels>]
       [ rv] [ update <seconds>] [ display displayname]
       [ geometry geom]
xstra>
```

16.3.2 Arrêt d'un client

Chaque client dispose d'une procédure d'arrêt qui lui est propre (*exit* pour le client *xterm*, le menu *file exit* pour le client *netscape*).

Fermer la fenêtre permet également d'arrêter un client. Pour cela cliquez sur la case du menu système. Un menu apparaît. Cliquez sur la commande *close* de ce menu et la fenêtre va disparaître. Le client associé à la fenêtre est arrêté. Si vous interrompez ainsi un client ayant des données à sauvegarder ou une émulation de terminal avec un programme en cours d'exécution, il est évident que les données seront perdues ou que le programme en cours d'exécution sera arrêté.

Si, pour une raison quelconque, il est impossible d'arrêter le client de cette manière, vous pouvez toujours tuer le processus à l'aide de la commande *kill* après avoir recherché le numéro de processus à l'aide de la commande *ps*.

16.3.3 Quelques ressources communes à certains clients

Chaque client dispose d'un certain nombre de ressources qui lui sont propres ainsi que de ressources communes à tous les clients s'appuyant sur les bibliothèques standard.

a) Couleurs

Il est possible de modifier les couleurs des différents éléments des clients (à condition de posséder un écran couleur naturellement). Les ressources les plus courantes concernant les couleurs sont décrites dans le tableau 16.1.

Ressources	Description
-fg couleur	Couleur des caractères
-bg couleur	Couleur du fond
-cr couleur	Couleur du curseur
-ms couleur	Couleur de la souris

TABLEAU 16.1. RESSOURCES CONCERNANT LES COULEURS.

La *couleur* peut être déterminée, soit par le nom de la couleur définie dans le fichier */usr/lib/X11/rgb.txt*, soit par la représentation hexadécimale de la couleur dans les trois couleurs de base (rouge, vert et bleu). La commande *more /usr/lib/X11/rgb.txt* permet de connaître les noms de couleurs prédéfinies.

Exemples

```
xstra> xterm  bg blue  fg red &
```

est identique à :

```
xstra> xterm  bg #0000ff  fg #ff0000 &
```

b) Taille et localisation dans l'espace de travail

Chaque client que vous lancez est positionné dans l'espace de travail. Vous pouvez choisir la position lors du lancement à l'aide de la ressource :

geometry LARGEURxHAUTEUR+colonne+ligne

où :

LARGEUR représente la largeur de la fenêtre en pixels (en caractères pour le client *xterm*).

HAUTEUR représente la hauteur de la fenêtre en pixels (nombre de lignes pour *xterm*).

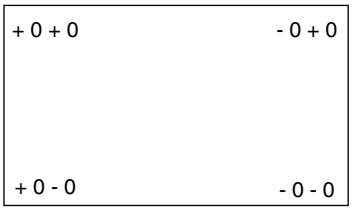
colonne représente la position de la colonne de la fenêtre en pixels. Si un **+** précède *colonne*, alors la position est déterminée par rapport au bord gauche de la fenêtre racine. Dans le cas d'un alors la position est déterminée par rapport au côté droit de la fenêtre racine.

ligne représente la position de la ligne de la fenêtre en pixels. Si un **+** précède *ligne* alors la position est déterminée par rapport au haut de la fenêtre racine. Dans le cas d'un elle est déterminée par rapport au bas de la fenêtre racine.

Voici un résumé des positions possibles par rapport à la fenêtre racine :

Vous pouvez omettre soit *LARGEURxHAUTEUR*, soit *+colonne+ligne*.

Dans ce cas la ressource par défaut est prise en compte.



Exemples

```
xstra> xterm  geometry 80x60 0 0 &
```

La fenêtre *xterm* aura 80 colonnes et 60 lignes. Elle sera positionnée en bas à droite de l'espace de travail.

```
xstra> xterm  geometry  0+0 &
```

La fenêtre *xterm*, de taille 80 colonnes et 24 lignes, sera positionnée en haut à droite de la fenêtre racine.

c) Les polices

Il est également possible de modifier les polices de caractères du texte apparaissant dans les fenêtres à l'aide de la ressource *fn nom fonts*. Les polices de caractères définissent la typographie des caractères utilisés pour la visualisation du texte.

Les polices de caractères disponibles sous X11 sont décrites dans des fichiers contenus dans le répertoire */usr/lib/X11/fonts* et ses sous-répertoires. Le nom de la police à indiquer, *nom fonts*, est, soit une chaîne de caractères définissant la police, soit un alias qui est déterminé dans le fichier *fonts.alias*. Si le fichier *fonts.alias* contient la chaîne «FILE_NAMES_ALIASES», la référence à la police sera réalisée par son nom sans le chemin et l'extension.

Exemple

```
| xstra> xterm  fn 9x15 &
```

Le nom du fichier de la police de taille 9X15 : */usr/lib/X11/fonts/9X15.snf*

16.3.4 Client local et client distant (remote)

Jusqu'à présent nous avons décrit l'exécution d'un client local. Une seule machine exécute les commandes de l'utilisateur, le serveur et les clients X.

Un client distant est un client exécuté sur un ordinateur distant, différent de celui exécutant le serveur X. L'ordinateur distant exécute le client. L'ordinateur local exécute le serveur X prenant en charge l'affichage (voir le paragraphe 16.1).

Plusieurs conditions préalables sont indispensables à l'exécution d'un client distant :

- la machine où s'exécute le client et la machine où s'exécute le serveur X doivent être connectées à un réseau TCP/IP,
- l'adresse Internet de la machine distante doit être définie sur la machine locale,
- l'utilisateur doit avoir un login sur la machine distante.

Ces conditions dépendent du réseau (chapitre 13). Du point de vue de X11, il faut également autoriser l'accès au serveur X et indiquer au client le nom du serveur X.

L'autorisation d'accès au serveur X est obtenue, soit en mettant le nom du serveur X dans le fichier texte */etc/X0.hosts*, soit en utilisant la procédure « magic cookies », soit en utilisant un client particulier *xhost*. Cette dernière méthode est conseillée car elle permet dynamiquement de modifier (additionner, supprimer) les ordinateurs ayant accès au serveur X. *xhost* ne modifie qu'une table interne d'autorisation et en aucun cas le fichier */etc/X0.hosts*. La syntaxe du client *xhost* est :

xhost	affiche la liste des ordinateurs ayant accès au serveur.
xhost +	permet l'accès à tous les ordinateur distants.
xhost	interdit l'accès à tous les ordinateurs distants.
xhost +nom	autorise l'accès au serveur pour l'ordinateur *nom*.
xhost nom	interdit l'accès au serveur X pour l'ordinateur *nom*.

Le nom du serveur X est indiqué au client distant, soit à l'aide de la variable d'environnement DISPLAY, soit à l'aide de la ressource *display nom serveur:x.y* (*x* : numéro du display et *y* : numéro du screen).

Exemple

Vous êtes sur l'ordinateur dont le nom est *courlis*. Vous établissez une connexion en utilisant *rlogin* avec l'ordinateur dont le nom est *ulp*.

```
xstra> hostname  § indique le nom de l'ordinateur local
courlis
xstra> xhost +ulp § Autorisation d'accès à ulp
xstra> xhost  § Affiche la liste des ordinateurs ayant
              § accès à courlis.
access control enabled, only authorized clients can connect
INET:ulp.u strasbg.fr
xstra> rlogin ulp
passwd
ulp> hostname
ulp
ulp> xterm  geometry  0 0  sb  display courlis:0.0 &
  § ou
ulp> export DISPLAY=courlis:0.0
ulp> xterm  geometry  0 0  sb&
ulp>
```

16.3.5 L'émulation de terminal xterm

Le client *xterm* est un émulateur de terminal alphanumérique DEC vt100. Il permet ainsi d'exécuter des commandes Linux comme sur un terminal traditionnel. Chaque environnement de travail a son propre émulateur de terminal, par contre l'émulateur *xterm* est disponible sur toutes les distributions Linux. . Par défaut, *xterm* crée un processus shell fils. Celui-ci est en communication constante avec *xterm*. Le shell créé est déterminé par la variable d'environnement SHELL. D'habitude *xterm* positionne automatiquement la variable TERM à *xterm*.

xterm s'utilise comme un terminal traditionnel. Il est possible d'utiliser la commande *stty* pour sa configuration. La position d'entrée d'une commande est déterminée par un curseur. La position de la souris est indépendante de ce curseur.

xterm utilise les ressources présentées au paragraphe 16.3.3. Cependant, il existe de nombreuses ressources et possibilités offertes par *xterm*. Nous allons en décrire quelques-unes :

- **Barre de défilement (ascenseur)**

  La ressource *sb* permet de sauvegarder un certain nombre de lignes qui défilent à l'écran dans un buffer interne et d'ajouter une barre de défilement à la fenêtre afin de visualiser les lignes sauvegardées.

- **Taille de la fenêtre**

  La modification de la taille de la fenêtre est décrite au paragraphe 16.2.3. Néanmoins, afin que des programmes tels que *vi* tiennent compte de telles modifications, il est nécessaire d'exécuter dans la fenêtre la commande

`eval ´resize´` qui met à jour les variables d'environnement TERM, LINES et COLUMNS.

Afin d'éviter la saisie trop fréquente de la commande complète, on crée une fonction en Bash :

```
xr () {eval ´resize´;}
```

- **Copier-coller**

  La fonctionnalité *copier-coller* est fournie par X11. Elle permet ainsi à des clients d'échanger des données. L'échange est réalisé en deux phases :

  – Sélection du texte à copier.

  Elle s'effectue à l'aide de la souris. La sélection est visualisée par une inversion vidéo du texte. Positionnez la souris au début du texte à copier. Cliquez avec le bouton gauche et faites glisser la souris afin de sélectionner le texte souhaité. En relâchant le bouton gauche vous avez copié le texte sélectionné (visualisé en vidéo inverse) dans le buffer interne.

  Vous pouvez également sélectionner un texte en cliquant avec le bouton de gauche au début du texte, puis en cliquant avec le bouton de droite à la fin. En faisant glisser le bouton de droite, il est possible d'agrandir ou de réduire la portion du texte sélectionné.

  La troisième manière de sélectionner un texte est de sélectionner un mot en cliquant deux fois sur le mot et une ligne en cliquant trois fois sur la ligne avec le bouton gauche.

  – Recopie du texte sélectionné à l'emplacement souhaité.

  Le texte est recopié en cliquant sur le bouton central de la souris (simultanément sur les deux boutons si la souris n'en possède que deux) à l'endroit où se trouve le curseur dans la fenêtre active.

  Ce mécanisme est extrêmement utile lorsqu'il est employé avec `xterm`. En effet il permet de sélectionner la totalité ou une partie des commandes dans l'historique, des noms de fichiers résultats d'une autre commande. Il permet par exemple de faire du «copier-coller» entre deux applications standard telles que `vi` et `vi`, ou `vi` et `ls`,…

Exemple

Ouvrez deux fenêtres `xterm`. Dans chaque fenêtre, ouvrez un fichier texte à l'aide de `vi`. Dans une fenêtre, sélectionnez un texte à l'aide de la souris. Dans une autre, placez-vous en mode insertion à l'endroit où vous souhaitez insérer le texte sélectionné dans la première fenêtre. Déplacez la souris dans la fenêtre où vous souhaitez insérer le texte et cliquez sur le bouton central. Vous constaterez que le texte sélectionné dans l'autre fenêtre a été copié et inséré dans le nouveau fichier.

- **Titre de la fenêtre et de l'icône**

Il est possible de fixer le titre de la fenêtre à l'aide de la ressource `title nom fenêtre`. Le titre de l'icône est par contre modifié par la ressource `n nom icône`.

Exemple

```
xstra>xterm  n "systeme"  title "Emulation terminal" &
xstra>
```

16.4 L'ENVIRONNEMENT DE TRAVAIL

L'environnement de travail est conçu principalement pour éviter à l'utilisateur, le processus toujours complexe d'apprentissage des commandes avec ses options associées, et la difficulté de configuration de son ordinateur. Ainsi les clients présentés au paragraphe précédent (16.3) peuvent être facilement exécutés sans taper une ligne de commandes au clavier, en utilisant simplement la souris. Pour cela il suffit de sélectionner dans le menu principal le client désiré. Enfin, la plupart des ressources du client sont accessibles à partir des menus du client. Que vous soyez débutant ou que vous connaissiez bien Linux, l'utilisation d'un environnement de travail facilitera votre tâche. Néanmoins, il ne vous dispensera pas d'un bon apprentissage des principes et des commandes Linux.

Un environnement de travail est agréable et convivial ; il possède un gestionnaire de fichiers puissant et facile à utiliser. Il est de configuration simple et possède une aide en ligne. De plus il permet d'avoir une interface homogène pour vos applications et vous propose des utilitaires faciles à utiliser. Il comprend les fonctionnalités suivantes :

a) Un tableau de bord

Le tableau de bord est une fenêtre spéciale qui apparaît généralement en bas de l'écran. De là, vous lancerez vos programmes et basculerez entre les différents espaces de travail. Il comprend généralement :

- des icônes de lancement d'applications qui vous permettent d'exécuter des programmes (le gestionnaire de fichiers, le gestionnaire de configuration, le verrouilleur d'écran...),
- un icône (en forme de pied pour GNOME, un grand K pour KDE) qui, lorsqu'on clique dessus, affiche un menu déroulant avec toutes les applications disponibles,
- des icônes d'indications (l'horloge, la charge de l'ordinateur...),
- des icônes qui remplissent les deux fonctions de lancement d'applications et d'indicateurs (messagerie, agenda...)
- des icônes de pose qui permettent en y glissant des objets de réaliser une fonction (poubelle, imprimante...)

- des menus secondaires, indiqués par des flèches situées au-dessus d'une icône, qui contiennent au moins une icône de pose d'installation et une icône de lancement,
- un sélecteur d'espace de travail permettant de choisir parmi plusieurs espaces de travail différent,
- un icône permettant de quitter son environnement de travail.

Il est possible de personnaliser facilement à l'aide de la souris ou d'un utilitaire, le tableau de bord en ajoutant des icônes de lancement aux menus secondaires (utilisation de l'icône de pose du menu secondaire), en ajoutant des menus secondaires, en modifiant le nom des espaces de travail et leur nombre.

b) Un gestionnaire de fichiers

Le gestionnaire de fichiers permet de manipuler (créer, utiliser, déplacer, copier, rechercher, renommer) les fichiers, répertoires et objets de votre système à l'aide d'une interface graphique puissante et conviviale. La plupart des fonctionnalités est réalisée en utilisant la souris (sélection simple ou multiple, glisser-déplacer, ouverture...).

c) Le centre de contrôle

À la différence de beaucoup d'environnements graphiques Unix, vous n'avez pas besoin d'éditer des fichiers de configuration abscons. Le centre de contrôle de l'environnement de travail permet de configurer presque chaque aspect de l'apparence et du comportement de votre bureau. Cet utilitaire, permet de personnaliser l'aspect de votre bureau en modifiant entre autre :

- la couleur des cadres des fenêtres actives ou inactives, du sélecteur d'espace de travail, des zones de texte et listes....
- l'apparence du bureau comme les couleurs ou images de fond d'écran, les couleurs des fenêtres,
- la police de caractères, la langue,
- certaines caractéristiques du clavier telles que le volume sonore, la vitesse de répétition des touches,
- la vitesse, accélération de déplacement de la souris,
- le volume, la tonalité et la durée du « bip »,
- les caractéristiques de l'économiseur et du verrouillage de l'écran,
- les réglages pour le gestionnaire du bureau et le tableau de bord incluant les bureaux virtuels.
- le mode de sélection des fenêtres,
- les réglages relatifs aux sons,
- et les paramètres de fin de session (sauvegarde et restauration de la session en cours).

d) Un logiciel de messagerie

Le logiciel de messagerie disponible est bien plus convivial et facile à utiliser que le logiciel standard de Linux : `mail`. Ses caractéristiques sont proches de celles des logiciels de messagerie disponibles sur les PC. Il permet entre autre de lire, classer, rechercher, supprimer, composer les messages ainsi que d'y répondre ou faire suivre. Il gère aussi des boîtes aux lettres de rangement. L'impression d'un message est très aisé. De plus ce logiciel gère aussi les fichiers attachés et peut décoder les fichiers en format MIME.

e) Un éditeur de texte

L'éditeur de texte permet de créer et éditer un document en format texte. Ce logiciel simple d'utilisation peut remplacer `vi`.

f) Un émulateur de terminal

Chaque environnement de travail a son propre émulateur de terminal plus performant et convivial que `xterm` qui est l'émulateur par défaut de X11. Des options disponibles, soit à partir du menu de la fenêtre, soit à partir d'un clic sur le bouton droit de la souris, permettent de modifier l'aspect de l'émulateur de terminal, les caractéristiques du curseur, le mode de défilement, les paramètres du clavier et de l'écran.

h) Un verrouilleur d'écran

Le verrouilleur d'écran permet, sans quitter la session, de verrouiller l'écran ; ceci empêche un autre utilisateur d'utiliser votre compte en votre absence. Préférez la sortie de votre environnement de travail lorsque votre absence est prolongée. Vous libérerez des ressources de l'ordinateur ainsi que votre écran.

L'utilisation de chacune de ces fonctionnalités est relativement intuitive. Pour les utilisateurs souhaitant acquérir une pratique plus complète, nous conseillons dans une première étape l'aide en ligne puis la consultation d'ouvrages spécialisés. Il existe sur le web une documentation en français pour chaque environnement de travail :

KDE (*http://www.kde.org/fr/documentation.html*)

GNOME (*http://perso.club-internet.fr/nicolovi*).

16.5 EXERCICE

Exercice 16.5.1

Vous êtes connecté à la machine davinci dans l'environnement graphique X11. Vous souhaitez lancer un client X (par exemple `xclock`) sur la machine courlis. Que faites-vous ?

Annexe A
Les commandes

Quelques commandes avec les options les plus utilisées sont présentées dans cette annexe. Elle ne remplace pas la consultation du manuel de référence mais propose plutôt une aide rapide.

at

Exécute une suite de commandes à une heure fixée (exécution différée).

Syntaxe

```
at time [date] [+increment]
at  d job
at  l [job]
```

Options

-d permet d'effacer un *job* précédemment initialisé par *at*.

-l permet d'afficher toutes les commandes différées (*job*) initialisés par *at* pour l'utilisateur.

Description

Cette commande permet de lancer une suite de commandes à partir de l'entrée standard à une heure et une date fixées. Les commandes en attente se trouvent dans

le fichier */var/spool/at/spool*. La sortie standard et la sortie d'erreur standard sont envoyées par *mail* à l'utilisateur, sauf si elles sont redirigées.

Les variables d'environnement, le répertoire courant, le masque de création des fichiers (*umask*) sont conservés lorsque la commande est exécutée.

L'autorisation d'utilisation de *at* pour un utilisateur est indiquée dans le fichier */etc/at.allow*. Si ce fichier n'existe pas, le fichier */etc/at.deny* est vérifié pour déterminer si l'accès à la commande *at* doit être interdit à l'utilisateur. Si *at.deny* est vide, l'utilisation de *at* est permise pour tous les utilisateurs. Si aucun fichier n'existe, seul l'administrateur de la machine (root) a la permission d'utilisation de *at*.

L'heure (time) est spécifiée sous la forme heure:minute. Un indice *am* ou *pm* peut y être ajouté ; sinon c'est une heure d'horloge sur 24 heures qui est utilisée. On peut spécifier en option une date sous plusieurs formes possibles, dont l'écriture française (25/02/92).

Il existe des noms spéciaux reconnus par *at* pour *time* : *noon*, *midnight*, *now*, *next* ; pour *date* : *today*, *tomorrow* ; pour l'incrément : *day*, *week*, *month*, *year*.

Exemples

1) Exécution le 23/9/2000 à 8 h 15 des commandes contenues dans le fichier *test1*.

```
xstra> at 08:15 sep 23 <test1
warning: commands will be executed using /bin/sh
job 6 at 2000 09 23 08:15
xstra>
```

2) Exécution dans un jour de *test2*.

```
xstra> at now +1 day <test2
warning: commands will be executed using /bin/sh
job 7 at 2000 08 10
xstra>
```

3) Exécution le jour même à 18h15 de *test3*.

```
xstra> at 18:15 <test3
warning: commands will be executed using /bin/sh
job 8 at 2000 08 09 18:15
xstra>
```

Les commandes soumises à *at* sont rangées dans des fichiers qui se trouvent dans le répertoire */var/spool/at/spool*. L'affichage de la date peut être modifié en positionnant la variable d'environnement POSIXLY_CORRECT.

Exemple

```
xstra> at  l     § affichage des commandes différées
6       2000 09 23 08:15 a
7       2000 05 25 10:05 a
8       2000 05 24 18:15 a
xstra> at  d 8 § suppression d'une commande différée
xstra> at  l
6       2000 09 23 08:15 a
7       2000 05 25 10:05 a
xstra> declare  x POSIXLY_CORRECT
xstra> at  l
6       Sat Sep 23 08:15:00 2000 a
7       Thu May 25 10:05:00 2000 a
xstra>
```

awk

Filtre permettant de travailler sur des fichiers ASCII : extraction de lignes et de champs selon des critères (voir chapitre 14).

Syntaxe

Deux formes d'invocation de *awk* sont fréquemment rencontrées.

La première est utilisée dans le cas où le programme *awk* est court :

```
awk [programme] [ Fc] [variable valeur] fichiers
```

programme	programme awk : critères de sélection et actions
−Fc	définition du séparateur de champs, ici le caractère c
variable=valeur	assignation de valeurs à des variables internes de awk
fichiers	liste de fichier à traiter, ou l'entrée standard si cette liste est vide

La deuxième est utilisée dans le cas où le programme *awk* est long :

```
awk [ f prograwk][ Fc] [variable valeur] fichiers
```

progawk	nom du fichier : awk cherchera son programme dans ce fichier

Options

-f *progawk* utilise le contenu du fichier *progawk* comme source de programme : critères de sélection et actions

-Fc utilise le caractère *c* comme caractère séparateur de champs.

Description

awk effectue les opérations de recherche définies par *grep*. Surtout, il peut y avoir des actions sur les champs des lignes concernées pour le(s) fichier(s) à traiter. Dans chaque fichier texte *awk* cherche à traiter les lignes qui correspondent à un des critères contenus dans programme ou dans *fichier programme* (le *fichier programme* ' ' veut dire l'entrée standard). Si *programme* est spécifié sur la ligne de commandes, il doit être délimité par des simples quotes pour éviter qu'il ne soit interprété par le shell (voir exemple 2).

Les fichiers à traiter sont lus dans l'ordre. L'entrée standard est lue si il n'y a pas de fichier. Le fichier ' ' indique l'entrée standard.

Les variables *awk* peuvent être initialisées sur la ligne de commandes en utilisant les arguments de la forme *variable=valeur*. Cela initialise *variable* à *valeur* avant que le premier enregistrement de *fichier* ne soit lu (voir exemple 3).

À chaque critère, une action peut être associée : *sélection actions*. Chaque ligne de chaque fichier d'entrée est analysée en fonction de chaque ligne de *fichier programme*. Chaque ligne est divisée en champs notés $1, $2… $0 désignant la ligne entière. Les actions suivent une syntaxe voisine de celle du langage C. Il sera donc possible de faire des calculs, des tests, des impressions.

Exemple 1

Affichage des valeurs des champs utilisateur et groupe pour toutes les lignes du fichier */etc/passwd*.

```
xstra> awk  F: '{ print $1,$4 }' /etc/passwd
root 1
soline 205
. . .
xstra>
```

Exemple 2

Pour toutes les lignes du fichier */etc/passwd*, récupération des champs utilisateur, groupe, Home et shell et affichage dans cet ordre. (Cette commande est en une seule ligne, repliée ici pour des raisons typographiques).

```
xstra> awk  F: '{ printf("  \tUtil= %s \tGroup= %s \t Home= %s
\tShell= %s \n", $1, $4, $6, $7) }' /etc/passwd
Util who    Group 100 Home /            Shell /bin/who
…
```

```
Util xstra  Group 200 Home /home/xstra  Shell
Util soline Group 300 Home /home/soline Shell /bin/sh
…
xstra>
```

Exemple 3

Affichage de tous les utilisateurs du groupe 200 :
D'abord, création du script *lstgrp* qui contient la commande *awk*.
Ensuite, exécution de ce script avec comme paramètre le groupe 200.

```
xstra> cat >lstgrp
awk  F: '$4 == Groupe {print $1}' Groupe=$1 /etc/passwd
<ctrl d>
xstra> chmod  a+x  lstgrp
xstra> lstgrp 200
xstra
xstra1
xstra2
xstra>
```

basename, dirname

Permet d'extraire une partie d'un chemin d'accès.

Syntaxe

```
basename chaine [suffix]
dirname chaine
```

Description

La commande *basename* extrait et affiche sur la sortie standard la dernière chaîne de caractères précédée par / d'un chemin d'accès défini par *chaine*. On obtient ainsi le nom d'un fichier. Le nom du fichier est affiché sans suffixe en cas de présence de *suffix*. La commande *dirname* extrait l'autre partie du nom de fichier.

Exemples

```
xstra> basename /home/xstra/passwd.tri
passwd.tri
xstra> dirname /home/xstra/passwd.tri
/home/xstra
xstra> basename /home/xstra/passwd.tri .tri
passwd
xstra>
```

batch

Exécute une suite de commandes en mode batch.

Syntaxe

```
batch
```

Description

La commande *batch* permet de mettre en file d'attente des commandes qui seront exécutées lorsque le niveau de charge du système le permettra.

La commande sera exécutée lorsqu'elle sera en tête de la file d'attente ; ainsi les commandes seront exécutées quel que soit l'utilisateur qui a mis la commande dans la file d'attente.

Exemple

```
xstra> batch
prog1 >sortie
<ctrl d>
xstra>
```

cat (conCATenate)

La commande *cat* est une commande multi-usage qui permet d'afficher, de créer, de copier et de concaténer des fichiers.

Syntaxe

```
cat [ b] [ n] [ s] [ e] [ t] [ v ] [ fichier…]
```

Options

-b numéroter les lignes en omettant les lignes vides.

-n numéroter les lignes.

-s remplacer plusieurs lignes vides consécutives par une seule.

-e visualiser les caractères non imprimables et les fins de ligne par un $.

-t visualiser les caractères non imprimables et les tabulations par un <ctrl-i>.

-v afficher en clair des caractères non imprimables.

Description

La commande *cat* lit le ou les fichiers spécifiés et les affiche sur l'écran (sortie standard). Si aucun fichier n'est spécifié, *cat* lit l'entrée standard.

Le mécanisme de la redirection s'applique et rend cette commande multi-usage, comme l'illustrent les exemples ci-après.

Exemples

1) Lecture des données à partir du clavier (faire <ctrl-d> en fin de saisie des données) et affichage à l'écran.

```
xstra> cat
coucou c'est moi
salut
<ctrl d>
xstra>
```

2) Copie du fichier *f1* dans le fichier *f2*.

```
xstra> cat f1 >f2
xstra>
```

3) Concaténation des fichiers *f1*, *f2*, *f3* dans le fichier *f123*.

```
xstra> cat f1 f2 f3 > f123
xstra>
```

cd (Change Directory)

Permet de changer de répertoire de travail.

Syntaxe

```
cd répertoire
```

Description

Cette commande permet de changer de répertoire de travail. Si *répertoire* n'est pas indiqué, le contenu de la variable interne HOME est utilisé comme nouveau répertoire de travail. Si *répertoire* est un chemin absolu, ou un chemin relatif débutant par **.** ou **..**, la commande *cd* applique le changement de répertoire.

Pour les autres cas de chemin relatif, *cd* essaye de découvrir le répertoire désigné par rapport à l'un des chemins spécifiés par la variable CDPATH. Il est conseillé de mettre le répertoire courant . en premier dans la variable d'environnement CDPATH.

Attention : *cd* est une commande interne qui dépend donc de l'interpréteur de commandes.

Exemples

```
xstra> pwd
/home/xstra/rep1
xstra> cd    § retour au répertoire de travail d'accueil.
xstra> pwd
/home/xstra
xstra> cd rep1  § rep1 devient le nouveau
                § répertoire de travail.
xstra> pwd
/home/xstra/rep1
xstra> cd rep2  § rep2 devient le nouveau répertoire
                § de travail si /home/xstra est
                § dans la variable cdpath.
xstra> pwd
/home/xstra/rep2
xstra>
```

chmod (CHange MODe)

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

Syntaxe

```
chmod [ R] mode fichier
```

Option

-R descend récursivement à travers les répertoires, positionne le *mode* pour chacun des fichiers décrits.

Description

La protection d'un fichier ou de plusieurs fichiers peut être modifiée par le propriétaire du ou des fichiers (ou par le super-utilisateur) à l'aide de la commande *chmod*.

Il existe deux modes d'utilisation de cette commande. Le premier (le plus ancien) utilise la description des protections par un nombre octal. Le deuxième mode d'utilisation permet une description symbolique des droits d'accès.

a) Utilisation du mode octal

`rwx rw  r x` est représenté par le code octal 765 (une lettre = 1, un tiret = 0)

⇒ *rwx rw r x = 111 110 101 = 7 6 5*

b) Description symbolique

chmod [who]op[permission] fichier

où

who	est une combinaison de lettres **u** (user=propriétaire), **g** (groupe), **o** (other=autre) ou **a** (all=tous) pour **ugo**.
op	est un opérande ; **+** permet d'établir la permission, **–** d'enlever la permission, et **=** d'établir exclusivement la permission en enlevant les autres.
permission	est la permission choisie ; **r** (read=lecture), **w** (write=écriture), **x** (exécution).

Exemples

Changement des protections du fichier *f1* en utilisant le mode octal, et du fichier *f2* en utilisant la description symbolique.

```
xstra> ls  l
 rw  r   r   1 xstra  staff  35  jan 12 15:54  f1
 rw  r   r   1 xstra  staff  23  fev 01 10:12  f2
xstra> chmod ug+x f1
xstra> chmod 765 f2
xstra> ls  l
 rwx r x r   1 xstra  staff  35  jan 12 15:54  f1
 rwx rw  r x 1 xstra  staff  23  fev 01 10:12  f2
xstra>
```

cp (CoPy)

La commande *cp* permet la copie de fichiers. Elle s'utilise sous trois formes :

1) la copie d'un fichier source dans un fichier destination,
2) la copie d'une liste de fichiers dans un répertoire,
3) la copie d'un répertoire dans un autre.

Syntaxe

```
cp [ i] fsource fdestination
cp [ i] fsource … répertoire
cp [ i] [ r] repsource repdestination
```

Options

-i demande de confirmation, en cas de copie sur un fichier existant.

-r copie récursive, c'est-à-dire copie des fichiers du répertoire *repsource* et de ses sous-répertoires.

Attention

Le répertoire destination ne devrait pas dépendre (être un sous-répertoire) du répertoire source.

Exemples

1) Dans le répertoire *xstra*, copie du fichier *essai* dans *essai1*.

```
xstra> cd /home/xstra
xstra> cp essai essai1
xstra>
```

2) Copie du fichier *essai* de *xstra* dans *xstra/projet1*.

```
xstra> cd /home/xstra
xstra> cp essai /home/xstra/projet1
xstra>
```

3) Copie du contenu du répertoire *xstra/projet1* dans */home/xstra/projet2* (copie uniquement des fichiers).

```
xstra> cd  /home/xstra/projet1
xstra> cp * /home/xstra/projet2
xstra>
```

4) Copie de l'arborescence de *xstra/projet1* dans */home/xstra/projet2*.

```
xstra> cd /home/xstra/projet1
xstra> cp  r *  /home/xstra/projet2
xstra>
```

crontab

La commande *crontab* permet d'initialiser le lancement cyclique d'une commande.

Syntaxe

```
crontab [fichier]
crontab  r
crontab  l
crontab  e
```

Options

-r permet de supprimer un utilisateur de *crontab*.

-l permet d'afficher le fichier *crontab* pour l'utilisateur.

-e permet d'éditer directement le fichier *crontab*.

Description

La commande *crontab fichier* copie *fichier* dans le répertoire */var/spool/cron* qui contient tous les utilisateurs ayant une crontab. Les utilisateurs ont le droit d'utiliser la commande *crontab* si leur nom figure dans le fichier */etc/cron.allow*. Si ce fichier n'existe pas, le fichier */etc/cron.deny* est consulté. Si le fichier *cron.deny* est vide et si *cron.allow* n'existe pas, tous les utilisateurs ont le droit d'utiliser *crontab*. Les sorties standard sont envoyées à l'utilisateur par la messagerie, sauf en cas de redirection.

Le fichier utilisé par *crontab* est composé de six champs :

Champ 1 : les minutes (0-59)
Champ 2 : l'heure (0-23)
Champ 3 : le jour du mois (1-31)
Champ 4 : le mois de l'année (1-12)
Champ 5 : le jour de la semaine (0-6 avec 0 = dimanche)
Champ 6 : la tâche à exécuter

Un astérisque dans un des champs signifie que toutes les valeurs du champ sont légales. Une liste d'éléments est indiquée par un tiret ( ). Il est à noter que deux champs peuvent spécifier le jour. La spécification des jours dans un champ impose à l'autre champ de contenir un astérisque. Dans le sixième champ est indiquée la commande qui sera exécutée par l'interpréteur de commandes aux dates spécifiées.

L'interpréteur de commandes est appelé à partir du répertoire HOME. Les utilisateurs qui souhaitent que leur *.bashrc* soit exécuté doivent le faire explicitement dans le fichier *crontab*.

Exemples

Création du fichier *essai* qui contiendra les commandes *test1* et *test2*. Ces tâches vont être lancées cycliquement.

```
xstra> cat >essai
0 0 1,15    * * test1
0 0     *   * 1 test2
<ctrl d>
xstra> crontab essai
xstra> crontab  l
0 0 1,15 * * test1
```

```
0 0 * * 1 test2
 xstra>
```

La commande *test1* sera exécutée le 1er et le 15e jour de chaque mois de l'année, à minuit.

La commande *test2* sera lancée tous les lundis à minuit.

cut

La commande *cut* est un filtre qui permet de couper en plusieurs champs des lignes d'un fichier ou de l'entrée standard.

Syntaxe

```
cut  cliste [ f1 …]
cut  fliste [  dx ] [ f1 …]
```

Options

-dx	choix du caractère *x* comme nouveau séparateur de champs.
-cliste	indique la position des caractères (position en colonne).
-fliste	indique la position des champs.

Description

La commande *cut* affiche une portion seulement des lignes des fichiers *f1*… ou de l'entrée standard. La portion sélectionnée peut être donnée par la position des caractères (option *cliste*) ou la position des champs (option *fliste*), le séparateur de champs par défaut étant le caractère *Tabulation*.

La liste est une suite d'entiers croissants séparés par des virgules. On peut préciser un intervalle sous la forme *n p*. La notation *n* équivaut à *1 n* et *p* équivaut à *p x*, où *x* représente le dernier élément (champ ou caractère). On peut également combiner une suite de valeurs et des intervalles (exemple : 1-4,7,12,30- correspond aux éléments 1, 2, 3, 4, 7, 12 et du 30e au dernier).

Exemples

1) Affichage des portions de lignes se trouvant en vingtième position pour le fichier *f1*.

```
xstra> cut  c20 f1
```

2) Affichage du nom et du numéro (UID) de chaque utilisateur (champs 1 et 3 avec le séparateur : du fichier */etc/passwd*).

```
xstra> cut  d:  f1,3 /etc/passwd
```

du (Disk Usage)

Donne l'occupation disque des fichiers et sous-répertoires du répertoire spécifié ou, si aucun répertoire n'est précisé, du répertoire courant.

Syntaxe

du [k] [s] [a] [répertoire …]

Options

-k indique la taille en Koctets (option par défaut).

-s (summary) n'indique que l'espace occupé par chaque répertoire.

-a (all) indique l'occupation disque de chaque fichier ou sous-répertoire.

Exemples

1) Place occupée par chaque répertoire et sous-répertoire du répertoire courant. Cette commande donne aussi la hiérarchie des répertoires à partir du répertoire courant.

```
xstra> du
2180    ./lib
6       ./bin
19      ./ludo/sauvegarde
20      ./ludo
…
10768   .
xstra>
```

Dans cet exemple, le sous-répertoire *lib* a 2 180 Ko et le répertoire courant 10 768 Ko.

2) Idem, mais pour le répertoire *ludo*.

```
xstra> du ludo
19      ./ludo/sauvegarde
20      ./ludo
xstra>
```

find

Cette commande scrute récursivement les répertoires spécifiés, d'après certains critères de recherche. Si ces critères sont vérifiés, une commande peut être lancée.

Syntaxe

```
find chemin … expression
```

Options

-name fichier vrai si *fichier* correspond au nom de fichier courant ; on peut utiliser les caractères spéciaux associés à la génération des noms de fichiers, à condition de neutraliser leur interprétation par le shell.

-type c vrai si le type de fichier est *c*, où *c* indique fichier spécial caractère (on a aussi les conventions : *b* pour fichier spécial bloc, *d* pour répertoire ou *f* pour fichier ordinaire).

-links n vrai si le fichier a *n* liens.

-user nom vrai si le fichier appartient à l'utilisateur *nom*.

-group gnom vrai si le fichier appartient au groupe *gnom*.

-size n[c] vrai si le fichier a une taille de *n* blocs (512 octets par bloc). Si *n* est suivi de *c*, la taille est en caractères (sur certains Linux, il est obligatoire d'associer l'option *size* à l'option *type*).

-mtime n vrai si le(s) fichier(s) a été modifié depuis *n* jours. Pour ces primitives, la valeur entière associée signifie :

- exactement *n* si elle n'est pas signée,
- plus de *n* si elle est signée par +,
- moins de *n* si elle est signée par .

-print toujours vrai : entraîne l'impression du chemin d'accès courant.

-exec cde vrai si la commande *cde* exécutée renvoie un code d'achèvement 0. La fin de la commande doit être marquée par **\ ;** et elle peut comporter en argument l'entrée courante symbolisée par **{ }** .

-perm mode vrai si les permissions du fichier correspondent exactement au *mode* (octal ou symbolique).

-perm -mode vrai si tous les bits de permission du mode sont positionnés pour le fichier.

-perm + mode vrai si l'un des bits de permission du mode est positionné pour le fichier.

Une expression composée résulte de la combinaison de primitives à l'aide des opérateurs suivants (par ordre de priorité décroissante) :

- expression entre parenthèses (ces dernières sont banalisées par \),
- l'opérateur *non* symbolisé par *!*,
- l'opérateur *et* résultant de la juxtaposition de primitives,
- l'opérateur *ou* symbolisé par *o*.

Description

Scrute récursivement le(s) répertoire(s) dont le chemin est spécifié. Recherche alors des entrées dont les caractéristiques vérifient l'expression booléenne composée d'une liste de conditions définies par les options précisées ci-dessus.

Exemples

1) Recherche de tous les fichiers source C pour tous les utilisateurs. Puis affichage à l'écran de la liste des fichiers.

```
xstra> find /home  name '*.c' -print
```

2) Recherche, dans toute l'arborescence, de tous les fichiers de taille supérieure à 300 000 caractères. Puis affichage à l'écran de la liste de ces fichiers.

```
xstra> find /  type f  size +300000c  print 2>/dev/null
```

3) Recherche des fichiers de l'utilisateur xstra qui ont été modifiés il y a plus de 20 jours. Puis affichage à l'écran de la liste des fichiers.

```
xstra> pwd
/home/xstra
xstra> find .  mtime +20  print
…
```

4) Suppression des fichiers, *core* et *a.out* auxquels on n'a pas accédé depuis plus de 2 jours :

```
xstra> find $HOME \(  name core  o  name a.out \) \<return>
 atime +2  exec rm {} \;
```

5) Liste à partir du répertoire courant de tous les programmes Fortran sauf les fichiers *toto.f*

```
| xstra> find .  type f  name '*.f' !  name toto.f  print
```

grep

Cette commande est un filtre qui permet de rechercher, dans un ou plusieurs fichiers, toutes les lignes qui contiennent une chaîne de caractères donnée (expression régulière).

Syntaxe

grep [n] [v] [l] [i] [c] expression [fichier…]

Options

-n fait précéder chaque ligne affichée par son numéro de ligne dans le fichier source.

-v affiche toutes les lignes sauf celles contenant *expression*.

-l n'affiche que les noms des fichiers dont au moins une ligne satisfait à la recherche.

-i ne fait aucune distinction entre les majuscules et les minuscules.

-c affiche le nombre de lignes qui contiennent l'expression.

-q n'écrit rien sur la sortie standard. Renvoie un code de retour 0 si la recherche est fructueuse. Option utilisée dans les shell scripts.

Description

grep recherche dans les fichiers spécifiés en entrée (par défaut l'entrée standard) les lignes qui contiennent une chaîne de caractères et l'affiche sur la sortie standard. La chaîne de caractères recherchée peut être décrite par une expression régulière réduite.

Les caractères `$ * [ ^ ( )` et \ ont une signification particulière pour le shell. Pour pouvoir les utiliser dans une expression régulière, il faut mettre l'expression entre quotes. L'interprétation des caractères spéciaux dans une expression régulière est décrite au paragraphe 14.

Exemples

1) Affichage de toutes les lignes du fichier *prog.f* contenant *read*.

```
| xstra> grep read prog.f
```

2) Affichage avec la numérotation de toutes les lignes du fichier *prog.f* contenant *read*.

```
| xstra> grep  n read prog.f
```

3) Affichage de toutes les lignes du fichier *prog.f*. contenant la chaîne de caractères *call sub* en majuscules ou minuscules.

```
| xstra> grep  i 'call sub' prog.f
```

4) Affichage de la ligne (ou des lignes) contenant un caractère numérique dans tous les fichiers du répertoire courant.

```
| xstra> grep '[0 9]' *
```

5) Recherche de tous les fichiers contenant la chaîne *toto* et affichage de leur nom.

```
| xstra> grep  l 'toto' *
```

head

Liste les n premières lignes d'un ou plusieurs fichiers avec impression du nom de chaque fichier comme séparateur. Voir la commande *tail* dans cette même annexe.

Syntaxe

```
head [ n] fichier…
```

Exemples

1) Affichage des 20 premières lignes du fichier *essai.f.*

```
| xstra> head  20 essai.f
```

2) Affichage des 10 premières lignes de tous les fichiers ayant comme suffixe *.f*.

```
| xstra> head  10 *.f
```

kill

Cette commande permet d'envoyer un signal à un processus.

Syntaxe

```
kill [ signal] pid
```

```
kill [ signal] %job
kill  l
```

Option

-9 parmi les signaux qu'on peut envoyer à un processus, nous ne retenons que le signal 9 qui demande l'arrêt du processus désigné par son numéro (PID).

-l liste des signaux.

Description

Pour demander l'arrêt d'un processus, il faut spécifier le numéro (PID) de ce processus ou, pour les travaux en arrière-plan, le numéro du travail à arrêter. La commande *ps* permet de trouver le numéro du processus. *jobs l* donne la liste des travaux en cours.

Exemples

1) Cette commande tue le processus dont le numéro (PID) est 1635.

```
xstra> kill  9 1635
xstra>
```

2) Affichage à l'écran de tous les signaux disponibles.

```
xstra> kill  l
 1) HUP  2) INT  3) QUIT  4) ILL  5) TRAP  6) IOT
 7) EMT  8) FPE  9) KILL  10) BUS …..
xstra>
```

Remarque

Le signal KILL est égal au signal 9.

less

Permet d'afficher un fichier ou l'entrée standard page par page. *less* est une version améliorée de la commande *more*, décrite plus loin. Le lecteur est prié de s'y rapporter pour une description des deux commandes.

ln (LiNk)

Permet d'établir un lien entre deux ou plusieurs noms et un fichier quelconque. Les fichiers liés à l'aide de cette commande n'occupent pas de place sur le disque.

Syntaxe

```
ln   option  fichier_source  fichier_cible
```

Option

`-s` permet de créer un lien symbolique

Description

Cette commande permet de créer un lien sur un fichier. Créer un lien correspond à donner un nouveau nom à un fichier déjà existant. Le fichier *`fichier source`* existe. *`fichier cible`* sera en fait le nouveau nom de *`fichier source`*.

En Linux, à chaque nom de fichier est associé un index qui va pointer sur un inode. La création d'un lien consiste à associer au nouveau nom de fichier le même index qui va donc pointer sur le même inode que celui du fichier *`fichier source`*.

Le lien n'est possible que dans un même *`file system`* et avec des fichiers. Cependant, l'option *`-s`* permet de créer des liens symboliques entre *file system* et entre répertoires.

La commande *`ln`* a été présentée en détail au paragraphe 10.1.6.

Exemple

Le fichier *`New f2`* est créé sans occupation disque et il est lié au fichier *`f1`*.

```
xstra> ln f1 New_f2
xstra> ls  l
 rw r  r     xstra 20  jan 17 10:54  f1
 rw r  r     xstra 20  jan 17 10:54  New_f2
xstra>
```

lpr lpq lprm

Envoie, à travers un spooler, un fichier sur une imprimante.

Syntaxe

```
lpr [ Pimprimante] [ #copies] [ m] [ r] [fichier…]
```

Options

`-Pimprimante` impression sur l'imprimante indiquée.

-#copies *copies* correspond aux nombres de copies à imprimer pour le ou les fichiers spécifiés.

-m le spooler envoie un message lorsque le fichier a été imprimé.

-r supprime le fichier en cours d'impression.

-a liste l'ensemble des imprimantes disponibles et leur état.

Description

La commande *lpr* envoie le ou les fichiers spécifiés au spooler d'impression qui le(s) stocke jusqu'à ce que l'imprimante soit disponible. Si aucune imprimante n'a été spécifiée avec l'option *P*, par défaut l'impression est envoyée dans le spool *lp*.

Il est possible de connaître le statut de l'imprimante : *lpq*. On peut ainsi retrouver le numéro de la demande (job) des fichiers en cours d'impression pour une éventuelle suppression. La suppression d'une impression de la file d'attente du spooler se fait par la commande :

```
lprm job
```

en précisant éventuellement l'imprimante avec l'option *P*(*lprm Plp2 job*).

ls (LiSt files)

Permet d'obtenir la liste et les caractéristiques des fichiers contenus dans un répertoire. Si aucun argument n'est donné, la commande *ls* affiche la liste des noms des fichiers du répertoire courant par ordre alphabétique.

Syntaxe

```
ls [aFlgutCR] [fichrep]
```

Options

-F indication du type des fichiers par un caractère suffixe :
- / pour les répertoires,
- * pour les exécutables,
- @ pour les liens symboliques,

-l liste détaillée des caractéristiques de chaque fichier du répertoire.

-ld rep liste les caractéristiques du répertoire en argument, plutôt que les fichiers qu'il contient, avec date de la dernière modification.

-lu avec date du dernier accès.

-lt trie les fichiers en fonction de la dernière modification.

-CR liste récursive des répertoires et sous-répertoires à partir du répertoire courant.

-a liste des fichiers cachés (par exemple : *.cshrc*, *.profile*, ...).

Exemple

```
xstra> ls  l
drwx r   r      1 xstra staff 540 Jan 21 16:15 rep1
....
xstra>
```

Le premier caractère indique le type de fichier (**d** pour répertoire, pour un fichier normal, **b** pour un fichier bloc, **c** pour un fichier caractère, **l** pour un lien symbolique). On trouve ensuite le nombre de liens du fichier, le propriétaire du fichier, le groupe, la taille, la date de dernière modification et le nom du fichier.

mkdir (MaKe DIRectory)

Commande de création de répertoire.

Syntaxe

mkdir option nom_répertoire

Options

-p avec cette option, la commande *mkdir* va créer le répertoire ainsi que tous les répertoires pères s'ils n'existaient pas.

-m mode permet d'affecter la protection définie par *mode* au répertoire créé.

Description

La commande *mkdir* crée un répertoire pour chaque nom passé en argument. La protection de ce(s) répertoire(s) est déterminée par la valeur de *umask* ou à l'aide de l'option *m*.

Exemples

1) Création du répertoire *projet1* puis du répertoire *program* dans *projet1*.

```
xstra> mkdir projet1 projet1/program
xstra>
```

2) Création de *projet2* et *program*.

```
xstra> mkdir  p /home/xstra/projet2/program
xstra>
```

more, less

Permet d'afficher sur l'écran un fichier page par page. Au bas de chaque page apparaît le message *more* suivi du pourcentage de texte déjà lu. Il suffit d'appuyer sur la touche <espace> pour afficher la page suivante ou la touche <return> pour passer à la ligne suivante. D'autres actions sont également possibles.

Syntaxe

more [cs] [lignes] [+numéroligne] [+/chaîne] fichier…

Options

-c	efface l'écran avant d'imprimer une page.
-s	remplace n lignes blanches consécutives par une seule.
-lignes	nombre de *lignes* par page, exemple : -20.
+numéroligne	visualise le fichier à partir de la ligne indiquée.
+/chaine	visualise le fichier deux lignes avant la première ligne contenant *chaine*.

Après l'affichage d'une page, il est possible d'avoir l'une des actions suivantes :

return	affichage d'une ligne supplémentaire à l'écran.

espace	affichage d'une nouvelle page.
q	sortie de *more*.
=	affiche le numéro de la ligne courante.
h	commande *help* (affiche les commandes disponibles).
v	passage sous *vi*, au même endroit du fichier.
:f	affiche le nom du fichier courant.
/chaine	recherche vers l'avant de la prochaine occurrence de la *chaine*.
?chaine	recherche vers l'arrière de la prochaine occurrence de la *chaine* (*less* seulement).
n	recherche de l'occurrence suivante.
N	recherche de l'occurrence suivante dans le sens opposé (*less* seulement).
is	saut de *i* lignes avant d'afficher la prochaine page. Exemple : 20s pour un saut de 20 lignes.

Exemples

1) Affichage du fichier *essai* à partir de la dixième ligne.

```
xstra> more  10 essai
Nous sommes sur la 10e ligne
Nous sommes sur la 11e ligne
…
Nous sommes sur la 20e ligne
<more>
```

2) Suite à l'affichage d'une partie du fichier *essai* (1ère page à l'écran, à partir de la 10e ligne), plusieurs réponses sont possibles. L'utilisateur choisit de quitter l'affichage.

```
q       § réponse de l'utilisateur
xstra>
```

L'utilisateur récupère la main.

Les principaux avantages de *less* sur *more* sont les suivants :

- *less* est capable de remonter dans un texte qu'il lit sur son entrée standard, alors que *more* ne peut le faire que sur un fichier ;
- les 4 touches de curseur et les touches « page suivante » « page précédente » permettent de naviguer dans le texte ;
- l'option *i* permet d'ignorer les différences Maj/Min dans une recherche, sauf si la chaîne recherchée comporte au moins une Majuscule ;
- l'option *a* permet de poursuivre la recherche à partir de la première ligne après le bas de la page ;
- l'option *z* permettant de fixer la taille de la fenètre accepte un nombre négatif : *z 2.* Cela signifie : deux lignes de moins que la taille de l'écran ;
- La variable *LESSCHARSET* permet de fixer le jeu de caractères parmi un vaste choix.

mv (move)

Transfère le nom d'un fichier ou d'un répertoire.

Syntaxe

```
mv [ i]  fichier_ancien  fichier_nouveau
mv [ i]  repertoire_ancien  repertoire_nouveau
mv [ i]  fichier  répertoire
```

Option

-i demande de confirmation, en cas de renommage sur un fichier existant. Cette option est conseillée pour éviter la destruction de *fichier nouveau*.

Description

La commande *mv* permet de changer le nom d'un fichier ou bien de transférer un ou plusieurs fichiers dans un autre répertoire, mais cette dernière opération est à utiliser avec précaution car les fichiers ne figureront plus dans leur répertoire d'origine. Il est donc possible d'effectuer des transferts de fichiers dans des répertoires, et des transferts de répertoires vers d'autres répertoires. Pour cela, les répertoires source et destination doivent appartenir au même "*file system*".

La commande *mv* est détaillée au paragraphe 10.1.6.

Exemples

1) Renommer le fichier *dmasse* en *dmasse.old*.

```
xstra> mv dmasse dmasse.old
xstra>
```

2) Renommer le répertoire *doc* en *doc.dir*.

```
xstra> mv doc doc.dir
xstra>
```

ps

Donne un ensemble de renseignements sur les processus en cours.

Syntaxe

ps [auxw]

Options

a affiche des renseignements sur tous les processus attachés à un terminal.

u donne, pour chaque processus, le nom de l'utilisateur (*user*), le pourcentage de cpu (*%cpu*), la taille virtuelle du processus (*size*) et la taille de la partie résidente du processus (*rss*) en Ko.

x affiche également des informations sur les processus non liés à un terminal.

w affichage sur 132 colonnes, utile pour voir le nom complet de la commande associée à chaque processus.

Description

La commande *ps* affiche la liste des processus. Suivant les options utilisées, les informations ci-dessous sont affichées :

PID	numéro du processus.
TTY	type de terminal :
	co /dev/console,
	mn /dev/ttymn.
STAT	état du processus :
	R en cours d'exécution,
	T arrêté (par <ctrl-z> par exemple),

	P en attente de page,
	D en attente sur le disque,
	S en sommeil depuis moins de 20s,
	I en sommeil depuis plus de 20s,
	Z processus fils terminé,
	W “swappe” sur disque.
TIME	temps écoulé (user+sys).
COMMAND	la commande en cours d’exécution.

Cette commande possède de nombreuses autres options dont celles compatibles System V et qui sont précédes par le signe -. Ainsi *ps ax* correspond à *ps ef* en System V.

Exemples

1) N’affiche que les processus de l’utilisateur.

```
xstra> ps u
...
xstra>
```

2) Affiche les processus de tous les utilisateurs.

```
xstra> ps au
...
xstra>
```

3) Affiche tous les processus en cours sur la machine.

```
xstra> ps ax
...
xstra>
```

pwd (Print Working Directory)

Affiche le chemin d’accès complet du répertoire courant.

Exemple

```
xstra> pwd
/home/xstra
xstra>
```

rm (ReMove)

Supprime un ou plusieurs fichiers.

Syntaxe

```
rm [ irf] nom …
```

Options

-i	demande de confirmation de chaque suppression pour éviter un effacement par inadvertance. Cette option est fortement conseillée.
-f	si le droit d'écriture *w* est absent, cette option ne demande pas d'autorisation pour supprimer le fichier.
-r	suppression récursive : si *nom* désigne un répertoire, cela permet de supprimer toute la sous-arborescence qu'il contient.

Description

Cette commande permet de supprimer un ou plusieurs fichiers. Il s'agit en fait de l'effacement du lien. Si celui-ci était unique, il y a effacement des données.

Si l'accès en écriture au fichier cité n'est pas autorisé, la commande affiche les protections du fichier en mode octal et attend la confirmation de la suppression du fichier (*yes*).

Attention

Le droit de supprimer un fichier n'est pas lié au droit d'écriture dans le fichier mais à celui d'écriture dans le répertoire qui le contient.

Exemples

1) Effacement du fichier *essai.o*.

```
xstra> rm essai.o
xstra>
```

2) Effacement de tous les fichiers objet (finissant par *.o*) du répertoire courant.

```
xstra> rm *.o
xstra>
```

3) Effacement du répertoire *projet1* et de toute sa sous-arborescence.

```
xstra> rm  r projet1
xstra>
```

rmdir (ReMove DIRectory)

Suppression d'un ou de plusieurs répertoires.

Syntaxe

```
rmdir repertoire
```

Description

Cette commande permet de supprimer un répertoire. Cela n'est possible que si le répertoire est vide et si l'utilisateur n'est pas positionné sur le répertoire qu'il veut supprimer.

Exemple

Suppression des répertoires *projet1* et *projet2*

```
xstra> pwd
/home/xstra
xstra> ls  l
drwx r x r x  xstra  staff  40  Jan    21 17:15 projet1
drwx r x r x  xstra  staff  540 Feb    21 10:15 projet2
xstra> ls  l projet1  § le répertoire projet1 est vide
xstra> ls  l projet2
drwx r x r x  xstra  staff  220 Sep    11 17:15 cours
xstra> ls  l projet2/cours     § le répertoire cours
                               § est vide
xstra> rmdir projet1
xstra> rmdir projet2/cours projet2
xstra>
```

Il faut d'abord supprimer le répertoire *cours* pour pouvoir supprimer le répertoire *projet2*.

sdiff

Affiche côte à côte deux fichiers en signalant les lignes qui sont différentes.

Syntaxe

```
sdiff [ l] [ s] [ wn] fic1 fic2
```

Options

-l quand les lignes sont identiques, n'affiche que celles de *fic1*.

-s n'imprime que les lignes différentes.

-wn définit la longueur des lignes (*n*) pour la sortie sur écran ; par défaut la longueur est de 130 caractères.

Description

Dans la présentation côte à côte des deux fichiers, chaque ligne de *fic1* est séparée de celle de *fic2* par l'un des délimiteurs suivants :

un espace de plusieurs blancs sépare l'affichage de lignes identiques,

< signale que la ligne n'existe que dans le fichier *fic1*,

> indique que la ligne n'existe que dans le fichier *fic2*,

| indique qu'une différence existe entre les deux lignes.

Exemples

```
xstra> sdiff fic1 fic2
x     |    y     § lignes différentes
abc         abc   § lignes identiques
oh    <           § ligne dans fic1 uniquement
      >    labo   § ligne dans fic2 uniquement
```

Exemples

1) Comparaison de *prog1.f* et *prog2.f*. Le résultat est affiché page par page à l'écran grâce à *more*.

```
xstra> sdiff  l prog1.f prog2.f | more
```

2) Comparaison de *proga.f* et *progb.f*. Affichage à l'écran uniquement des lignes différentes.

```
xstra> sdiff  s proga.f progb.f
```

sort

Cette commande permet de lire et de trier le contenu des fichiers.

Syntaxe

```
sort [ options] [+c1 [ c2]] .. [fichier …]
```

Options

-c	test si l'entrée est déjà triée.
-m	fusion des fichiers d'entrée qui doivent être triés.
-d	tri seulement sur l'ordre alphabétique.
-i	caractères de contrôle ignorés.
-n	tri numérique.
-r	ordre du tri inversé.
-tx	choix du séparateur de champs *x* au lieu de la valeur par défaut (espace ou tabulation).
-o fres	résultat du tri dans le fichier *fres*.

Description

sort trie un ou plusieurs fichier(s) suivant des critères spécifiques en s'appuyant sur la notion de champs. Un champ commence en début de ligne ou à la fin du précédent champ et il se termine en fin de ligne ou au premier caractère blanc, ou au premier caractère de tabulation qui suit. Par défaut le tri porte sur un seul champ (la ligne), selon l'ordre lexicographique.

c1 et *c2* sont des numéros de champs : l'option *+c1* permet d'exclure du tri les champs *1* à *c1* ; l'option *c2* permet d'exclure du tri les champs qui suivent le champ *c2*. *c1* et *c2* peuvent être suivis d'une ou de plusieurs options *dinr*. Si *c2* n'est pas précisé, le tri se fera du champ *c1 + 1* jusqu'à la fin de la ligne.

Les valeurs *c1* et *c2* sont de la forme *nbchamp.nbcar* où :

- *nbchamp* est le numéro du champ,
- *nbcar* est le nombre de caractères à ignorer depuis le début de ce champ (*.nbcar* est optionnel).

Exemples

1) Tri du fichier *essai* ; le résultat du tri sera dans le fichier *essai.tri*.

```
xstra> sort  o essai.tri essai
xstra>
```

2) Tri du fichier */etc/passwd* sur le numéro d'utilisateur (UID) qui est le troisième champ de chaque ligne du fichier. Ce champ est numérique. Chacun des

champs dans le fichier */etc/passwd* est séparé par le caractère ":". Le résultat du tri sera dans le fichier */etc/passwd.tri*.

```
xstra> sort  t: +2n  3  o /etc/passwd.tri /etc/passwd
xstra>
```

stty

La commande *stty* permet de visualiser et de modifier les caractéristiques de la liaison associée à l'entrée standard.

Syntaxe

```
stty [options] [arguments]
```

Option

Les très nombreuses options de cette commande sont décrites dans la documentation Linux, et le lecteur peut s'y référer par la commande *man stty*. Nous ne présenterons ici que les options les plus courantes.

-a	affiche toutes les caractéristiques de la liaison.
erase	définit le caractère d'effacement, par défaut <ctrl-h>.
intr	définit le caractère d'interruption, par défaut <ctrl-c>.
isig	permet la comparaison de chaque caractère entré au caractère INTR.
istrip	tronque les caractères à 7 bits à l'entrée.
susp	définit le caractère de suspension de processus, par défaut <ctrl-z>.

Description

Sans option, la commande *stty* affiche la vitesse de transmission de la liaison et les caractéristiques de cette liaison différentes de celles prises par défaut. L'option *a* (all) affiche toutes les caractéristiques de la liaison. Les autres options sont de la forme :

1) Activation (*option*) ou désactivation (*option*) d'un mode de fonctionnement de la liaison :

```
xstra> stty echo    § écho de chaque caractère saisi
xstra> stty  echo   § suppression de l'écho
```

2) Définition d'un paramètre de la liaison :

```
xstra> stty 9600 § vitesse de transmission de 9600 bauds
```

3) Définition d'un caractère de contrôle :

```
xstra> stty erase '<ctrl h>' § <ctrl h> devient le
                             § caractère d'effacement
```

tail

Liste le contenu d'un ou plusieurs fichiers à partir de n lignes du début (*+n*) ou de la fin (*n*) de chaque fichier. Voir la commande *head* dans cette même annexe.

Syntaxe

```
tail +n[lbc] fichier…
tail  n[lbc] fichier…
```

Options

l, k, c — *l, k, c* désignent respectivement l'unité ligne, bloc de 1K et caractère. En effet, *tail* sait commencer l'affichage à *n* lignes, blocs ou caractères du début ou de la fin du fichier, suivant qu'on spécifie le code *l, k* ou *c*. *tail +20c fic* affiche à partir du vingtième caractère. Par défaut, *tail* choisit la ligne comme unité.

Description

tail lit l'entrée standard si aucun fichier n'est indiqué. Cela implique que redirection et tube s'appliquent pleinement à *tail*. *tail* et *head* sont des filtres, comme le montre l'un des exemples ci-dessous.

Exemples

1) Lister à partir de la 90e ligne.

```
xstra> tail +90 fichier
```

2) Lister les 30 dernières lignes.

```
xstra> tail  30 fichier
```

3) Lister les lignes 200 à 239.

```
| xstra> tail +200 fichier | head  40
```

tee

Copie l'entrée standard sur la sortie standard et simultanément vers un (des) fichier(s).

Syntaxe

```
tee [ a] fichier
```

Option

-a	(append) : écrit à la suite du fichier spécifié. Par défaut, écrase le précédent contenu du fichier spécifié.

Exemple

```
| xstra> ls  l | tee  a mouchard
```

La sortie de la commande *ls l* apparaît normalement sur l'écran et simultanément est enregistrée à la suite du fichier *mouchard*.

wc (Word Count)

Cette commande compte le nombre de lignes, de mots et de caractères dans un fichier. C'est un filtre.

Syntaxe

```
wc [options] [fichier]
```

Options

-w	compte le nombre de mots.
-c	compte le nombre de caractères (octets).
-l	compte le nombre de lignes.

Description

La commande *wc* permet de compter le nombre de lignes, de mots et de caractères contenus dans les fichiers indiqués ou sur l'entrée standard si aucun fichier n'est précisé.

Un mot est une chaîne de caractères suivie d'au moins un espace, une tabulation ou un retour à la ligne.

L'option par défaut est *lwc*.

Exemples

1) Compter le nombre de mots de *fichier*.

```
xstra> wc  w fichier
30  fichier
xstra>
```

2) Recherche du nombre de lignes contenant le mot format dans le fichier *program.f*.

```
xstra> grep " format" program.f | wc  l
50
xstra>
```

who

Donne la liste des utilisateurs connectés au système avec indication du nom du terminal, de la machine utilisée et de l'heure d'entrée en session.

Syntaxe

```
who [am i]
```

Option

am i donne le nom de l'utilisateur actuellement en session sur le terminal.

Exemple

Recherche des utilisateurs connectés.

```
xstra> who
xstra    ttyp0     Sep 26    09:39
yannick  ttyp1     Sep 26    15:10
ludovic  ttyp2     Sep 26    11:35
xstra>
```

xargs

Construit dynamiquement une commande à partir de son entrée standard. Cette commande n'apparaît jamais que dans un tube.

Syntaxe

```
cmd1 | xargs cmd2
cmd1 | xargs [ t]  i cmd2 {}
```

Options

-i insère une ligne de l'entrée standard à la place de *{}* et lance la commande *cmd2* obtenue.

-t copie la commande obtenue sur la sortie erreur standard avant de la lancer.

Exemples

```
xstra> ls *.c | xargs -t  i cp {} ~{}
```

Tous les fichiers d'extension *.c* du répertoire courant sont copiés sous le même nom précédé de ~ (convention fréquemment utilisée).

```
xstra> ls *.c | xargs -t  i cp {} {}.bak
xstra> ls *.c | xargs -t  iFICH cp FICH FICH.bak
```

Ces deux lignes sont équivalentes : tous les fichiers d'extension *.c* du répertoire courant sont copiés sous le même nom suivi de *.bak*. Cet exemple montre que l'option *i* peut prendre un argument permettant d'améliorer la lisibilité.

```
xstra> find . -type f -name core | xargs rm
xstra> find . -type f -name core -exec rm {} \;
xstra> rm $(find . -type f -name core)
```

Ces trois lignes sont fonctionnellement équivalentes : supprimer tous les fichiers de nom *core* situés sous le répertoire courant. Mais ces trois solutions ne sont pas techniquement équivalentes si le nombre de fichiers trouvés est très élevé :

La deuxième solution fonctionne toujours, mais un processus *rm* est généré pour chaque fichier à supprimer, ce qui n'est pas efficace.

Dans la troisième solution, un seul processus *rm* est généré pour tous les fichiers à supprimer, ce qui est très efficace. Mais la ligne de commandes générée par le shell risque d'être trop longue ce qui produit une erreur : *bash :/bin/rm : Argument list too long*

La première solution est la meilleure : *xargs* construit la commande *rm* avec les noms passés sur son entrée standard. Si la liste est plus longue que la taille

maximum d'une ligne de commandes, *xargs* générera plusieurs commandes *rm* consécutives, aussi longues que possible, en coupant entre deux noms de fichier. Il n'y aura que peu de processus *rm* générés.

Annexe B
Les caractères spéciaux

La liste suivante présente les caractères spéciaux du shell, mais aussi les caractères (ou association de caractères) ayant, dans certains contextes, une signification particulière. Après une très brève description, le ou les numéros de paragraphes indiqués entre parenthèses permettent de se reporter à la définition précise de ces caractères spéciaux. Le tableau 7.3 n'est pas référencé dans cette annexe, bien qu'il présente une synthèse de l'utilisation de certains caractères spéciaux décrits ci-dessous. Le lecteur est invité à s'y référer dans tous les cas.

`< > >>`	- redirections (7.2)
`<<chaîne`	- lecture sur l'entrée standard avec condition d'arrêt (7.2.3.f)
`|`	- tube (7.3) - dans les expression régulières (14.1.4) - génération de noms en Bash (8.2.5d)
`||`	- séparateur conditionnel de commandes (7.1.5) - dans les expressions régulières (14.4.3)
`? *`	- génération des noms de fichiers (3.2)(7.7) - génération de noms en Bash (8.2.5d) - dans les expressions régulières (14.1)
`[ ]`	- commande `test` en Bash (8.1.6) (8.2.3) - ensemble de caractères pris en compte (3.2) (7.7) - élément d'un tableau de variables en Bash (8.2.2) - dans les expressions régulières (14.1)

`( )` - commandes groupées (7.6)
- génération de noms en Bash (8.2.5d)
- dans les expressions régulières (14.1)

`{ }` - fonction shell (6.4)
- option `exec` de `find` (annexe A)
- variable en Bash (8.2.2)
- génération de noms de fichier en Bash (8.2.5d)
- dans les expressions régulières (14.1)

`' " \` - caractères de neutralisation (7.8)
`` ` `` - backquoting d'une commande (7.5)

`&` - lancement d'une commande en arrière-plan (7.4)(12.5)
- `>&` redirections en bash (7.2.1)
- `&&` séparateur conditionnel de commandes (7.1.5)
- `&&` dans les expressions `awk` (14.4)

`;` - séparateur de commandes (7.6)

`;;` - séparateur dans une construction case (8.1.6)

<ctrl-c> - interrompt le processus en cours

<ctrl-d> - fin de fichier (7.2.2.a)

<ctrl-k> - efface à droite du curseur en Bash (6.5.2)

<ctrl-u> - efface à gauche du curseur en Bash (6.5.2)

<ctrl-z> - suspend la processus en cours (stopped) (12.5)

`$` - référence à une variable : `$variable` (6)
- désigne la dernière ligne pour `sed` (14.3)
- fin de la ligne (5.2.3) (14.1.3)
- numéro de champ dans `awk` (14.4)
- `$(commande)` substitution en Bash (7.5)

`$0` - nom sous lequel un script a été invoqué (8.1.3)
- nom d'une fonction interne dans un script Bash
- enregistrement courant pour `awk` (14.4)

`$1...$9` - référence le nième argument passé à un script (8.1.2)
- référence le nième champ pour `awk` (14.4)

`$#` - référence le nombre d'arguments passés à un script (8.1.3)

`$*` - liste de tous les arguments passés à un script (8.1.3)

`$?` - code de retour de la dernière commande exécutée (8.1.3)

`$!` - en Bash, dernier processus lancé en arrière plan (8.2.1)

`$_` - en Bash, dernier mot de la dernière commande exécutée (8.2.1)

`$$` - référence le numéro du processus shell (8.1.3)

`%` - désigne un job en background : `fg %job` (12.5)
- dans `vi` : recherche la parenthèse correspondante

`#` - tout ce qui suit `#` est du commentaire (sauf `#!` et `$#`)

`#!` - en première ligne d'un script, indique le chemin d'accès au shell (8.1)

`!` - gestion de l'historique en Bash (6.5.1)
- insertion dans `vi` (5.2.4)
- commande `map!` de `vi` (5.3.3)
- opérateur de négation pour le Bash (8.1.6)
- opérateur de négation pour `awk` (14.4)
- génération de noms de fichiers en Bash (8.2.5d)

`~` - désigne le répertoire d'accueil en Bash (6.6.3)
- vérifie une expression régulière pour `awk` (14.4.3)

`@` - génération de nom de fichiers en Bash (8.2.5d)
- séparateur dans une adresse Internet (13.2.5.3)

`/` - séparateur dans les chemins d'accès (3.3.1)
- recherche d'une chaîne dans `vi` (5.1.5)
- recherche d'une chaîne dans `more` (annexe A)
- séparateur de recherche pour `sed` (14.3)
- séparateur de recherche pour `awk` (14.4)

`^` - début de la ligne dans une expression régulière (14.1.3)
- négation d'une liste dans une expression régulière (14.1.2)

`.` (point) - comme premier caractère d'un nom de fichier (3.2)
- pour suffixer les noms de fichiers : *`fichier.c`*
- désigne le répertoire courant (3.3.2)
- `..` désigne le répertoire père (3.3.2)
- exécution d'un script en Bash (6.8)
- tout caractère dans les expressions régulières (14.1.2)
- séparateur de champs dans une adresse Internet (13.2.3.3)

<esc> - sortie du mode insertion de *`vi`* (5.1.6)

<tab> - complètement des noms en Bash (6.6.1)

`:` - passage en mode *`ex`* de *`vi`* (5.2)

`: :+ := :?` - substitution de variable en Bash (8.2.5d)

`=` - en Bash : *`variable=chaîne`* (8)
- en Bash : *`declare`* (8.2.2)

- voir *`$`*

`+ =` - le signe - précède les options dans la ligne de commandes
- dans le mode symbolique de la commande *`chmod`* (4.2.1)
- liste de caractères *`[a z]`* (3.2) (7.7) (14.1)
- voir *`: :+ :=`*

Annexe C
Corrigés des exercices

Cette annexe présente le corrigé des exercices proposés tout au long de cet ouvrage.

CHAPITRE 2

Exercice 2.4.1

```
jerome> passwd
Changing password for jerome
Old password:  §  Saisie de votre ancien mot de passe
New password:  §  Saisie de votre nouveau mot de passe
Re enter new password: §  Seconde saisie de votre
                       §  nouveau mot de passe
jerome>
```

Exercice 2.4.2

```
xstra> sleep 60
§  Vous n'avez pas la main. Vous ne pouvez pas
§  executer un autre programme. Pour interrompre le
§  programme sleep il faut tapez <ctrl C>. Le prompt
§  apparait à nouveau.
<ctrl c>
xstra>
```

CHAPITRE 3

Exercice 3.7.1

```
xstra> cd
xstra> mkdir rep1 rep1/rep2 rep1/rep3
```

```
xstra> cd rep1
xstra> cat >fich11
tout va bien
<ctrl d>
xstra> cp fich11 fich12
xstra> cp fich11 rep2/fich21
xstra> cp fich11 rep2/fich22
xstra> cp fich11 rep3/fich31
xstra> cp fich11 rep3/fich32
```

Exercice 3.7.2

```
xstra> cd
xstra> ls  RF rep1
rep1:
fich11 fich12 rep2/ rep3/

rep1/rep2:
fich21 fich22

rep1/rep3:
fich31 fich32
xstra>
```

Exercice 3.7.3

```
xstra> cd rep1
xstra> mv rep3 rep2/rep3
xstra> cd ..
xstra> ls -RF rep1
rep1:
fich11 fich12 rep2/

rep1/rep2:
fich21 fich22 rep3/

rep1/rep2/rep3:
fich31 fich32
xstra> cd rep1/rep2/rep3
xstra> rm fich31 fich32
xstra> cd ..
xstra> rmdir rep3
xstra> ls fich2* § Vérifiez avant de supprimer
xstra> rm fich2*
xstra> cd ..
xstra> rmdir rep2
xstra>
```

Exercice 3.7.4

La commande *man* vous permet de connaître la syntaxe et le résultat de la commande *id*.

```
xstra> man id
§  affichage page par page de la documentation
§  relative à la commande id
...
xstra> id
 uid 2001(xstra) gid 200(staff)
xstra> grep staff /etc/group
staff:!:200:benoit,jocelyne,martine,xstra
xstra>  § Il y a 4 utilisateurs dans le groupe staff.
```

CHAPITRE 4

Exercice 4.6.1

```
xstra> mkdir reptest
xstra> cd reptest
xstra> cat >bienvenue
echo Bienvenue dans le monde Linux
<ctrl D>
xstra> chmod u+x bienvenue
xstra> ./bienvenue
Bienvenue dans le monde Linux
xstra>
```

Exercice 4.6.2

```
xstra> mkdir rep
xstra> cd rep
xstra> echo lire modifier supprimer >ficha
xstra> ls  l ficha
 rw  r   r    1  xstra staff  24  Jan 28 16:08 ficha
xstra> cat ficha
lire modifier supprimer
xstra> echo et remodifier >>ficha § Permet de rajouter
                                  § (et remodifier) dans ficha.
xstra> cat ficha
lire modifier supprimer
et remodifier
xstra> rm ficha
xstra> ls
xstra>
```

Exercice 4.6.3

```
xstra> mkdir rep
xstra> cd rep
xstra> echo lire supprimer >ficha
xstra> chmod u w ficha
xstra> ls -l ficha
 r   rw       1  xstra staff  15  Jan 28 11:54 ficha
xstra> cat ficha
lire supprimer
xstra> echo modifier >>ficha
ficha: permission denied
xstra> rm ficha
rm: Do you wish to override protection 460 for ficha? y
xstra>
```

Exercice 4.6.4

```
xstra> mkdir rep
xstra> cd rep
xstra> echo lire >ficha
xstra> chmod u w ficha
xstra> chmod u w . § Modifie les protections du
                 § répertoire courant (.) c'est à dire rep.
xstra> cat ficha
lire
xstra> echo modifier >>ficha
ficha: permission denied
xstra> rm ficha
rm: remove write protected file `ficha'? y
rm: cannot unlink `ficha': Permission denied
xstra> ls
ficha
xstra>
```

Exercice 4.6.5

Quand le nouveau fichier est créé par la commande :

cp fichier existant nouveau fichier, il prend les permissions du *fichier existant*. Il ne faut pas oublier également que la permission *x* n'est jamais attribuée à un fichier à sa création, même si *umask* le spécifie.

```
xstra> umask 007
xstra> touch fichtest
xstra> mkdir reptest
xstra> ls  l reptest fichtest
total 8
 rw rw      1  xstra staff     0 Jan 11 11:54 fichtest
drwxrwx     2  xstra staff   512 Jan 11 11:54 reptset
xstra>
```

Exercice 4.6.6

```
xstra> chmod g+x reptest
  § les membres de mon groupe peuvent faire cd reptest
xstra> cd reptest
xstra> chmod g=rx bienvenue
  § les membres de mon groupe peuvent lire et executer
  § le fichier bienvenue
xstra> ls  l bienvenue
 rwxr x     1  xstra staff   29  Jan 25 17:54 bienvenue
xstra> chmod u w,g+w bienvenue
  § Je ne peux plus modifier bienvenue, alors que
  § les membres de mon groupe peuvent le faire
xstra> ls  l bienvenue
 r xrwx     1  xstra staff   29  Jan 25 17:57 bienvenue
xstra>
```

Exercice 4.6.7

La permission d'effacer un fichier n'est pas attribuée au fichier lui-même, mais elle dépend uniquement du droit d'écriture dans le répertoire qui le contient. Voici comment créer un fichier que les membres de mon groupe peuvent modifier mais pas supprimer :

```
xstra> mkdir –m 750 rep1
xstra> cd rep1
xstra> echo bonjour >fich1
xstra> chmod g=rw fich1
```

Les membres de mon groupe ne peuvent pas supprimer *fich1*, mais ils peuvent effacer tout son contenu par : *cp /dev/null fich1*

Maintenant, voici comment créer un fichier que les membres de mon groupe peuvent supprimer mais pas modifier :

```
xstra> mkdir –m 770 rep2
xstra> cd rep2
xstra> echo bonjour >fich2
xstra> chmod g=r fich2
```

Les membres de mon groupe ne peuvent pas modifier *fich2*, mais ils peuvent le supprimer ! Ceci est une gaffe techniquement faisable !
De telles permissions semblent illogiques. Les permissions linux sont codées sous la forme *rwx rwx rwx*, ce qui conduit à 512 possibilités. Il est évident que parmi les 512 possibilités, toutes ne correspondent pas à des permissions réellement intéressantes. Tout ceci montre que l'utilisateur de Linux doit être vigilant non seulement sur les permissions des fichiers, mais aussi sur les permissions des répertoires dans lesquels il place des fichiers accessibles aux autres utilisateurs.

CHAPITRE 6

Exercice 6.4.1

La solution la plus élégante en Bash consiste à créer une fonction.

```
xstra> dir ()
{                       § cette fonction peut être
ls  Al $* | more     § définie dans un fichier "d'initialisation"
}
xstra> dir *.c
 rw r  r   1 xstra  staff  146 Jan 11 17:41 hello.c
 rw r  r   1 xstra  staff  198 Jan 11 17:47 plc.c
xstra>
```

Exercice 6.4.2

```
xstra> alias rm="rm  i"
xstra> alias psmoi="ps  f  u $LOGNAME"
```

Exercice 6.4.3

```
xstra> PS1="\h:$PWD> "
courlis:/home/xstra> umask 027
  § Ces deux commandes doivent être placées dans
  § le fichier $HOME/.bash_profile ou le fichier .bashrc
  § afin d'être prise en compte lors de votre
  § prochaine entrée en session.
```

Exercice 6.4.4

La solution la plus élégante en Bash consiste à créer une fonction.

```
xstra> fd ()
{                       § cette fonction peut être
find .  name "$1"  type d  print § définie dans le
}                                 § fichier .bashrc
xstra> fd 'r*'
./medimax/rendu3d
./prog/resultats
xstra>
```

CHAPITRE 7

Exercice 7.9.1

```
# Notez bien le "\" à la fin de certaines lignes
# Ce caractère indique que la ligne de commandes
# continue sur la ligne suivante.
xstra> echo Il y a $(grep bash /etc/passwd | wc -l) \
utilisateurs de ce systeme dont le login shell est bash
```

Exercice 7.9.2

```
§ lister les fichiers sous le répertoire /etc
§ dont les noms commencent par rc
§ Ici, il y a un piège : il est certain qu'un
§ utilisateur n'ayant pas les privilèges de root ne peut
§ pas connaître la vraie réponse : des sous répertoires de
§ /etc sont rwx r x     et l'utilisateur courant ne peut
§ pas y pénétrer.
§ ATTENTION : ligne suivante, remarquez bien : 'rc*'
xstra> find /etc -type f -name 'rc*' -print
/etc/rc
/etc/rc.net         § Rien ne permet de savoir si cette
/etc/rc.powerfail   § liste est exhaustive : des fichiers
/etc/rc.nfs         § dont les noms commencent par rc
/etc/rc.tcpip       § peuvent exister dans des répertoires
/etc/rc.local       § auxquels je n'ai pas accès. Les
...                 § messages d'erreur permettront de
...                 § s'en convaincre.
xstra>
```

```
§ lister les fichiers réguliers qui m'appartiennent.
§ Ici, attention : des fichiers peuvent m'appartenir
§ sans se trouver dans mon arborescence personnelle.
§ Pour autant, il ne peut pas y en avoir partout !
§ Chercher à partir de / est inutile.
xstra> RACINE="$HOME /tmp /var/spool"
xstra> find $RACINE -type f -user xstra -print \
>/tmp/findmoi 2>/dev/null
...  §  > redirige le résultat vers /tmp/findmoi
     § 2> redirige les erreurs vers le fichier /dev/null
     § où les données vont disparaître.
xstra>
```

```
§ lister tous les sous répertoires de /etc
xstra> find /etc -type d -print
§ donnera un résultat incomplet. Les messages d'erreur
§ permettront de s'en convaincre.
```

```
§ lister tous les fichiers réguliers se trouvant sous
§ mon répertoire d'accueil et qui n'ont pas été modifiés
§ dans les 10 derniers jours.
xstra> cd
xstra> find . -type f -mtime +9  print
```

Exercice 7.9.3

```
§ Il suffit de rediriger le résultat de la commande find
§ dans le filtre tee pour stockage sur le disque puis
§ le filtre wc  l pour compter
```

```
§ le nombre de lignes trouvées par find.
xstra> find $HOME -size 1000000c -print | tee \
       /tmp/findres | wc  l
10
xstra>
```

Exercice 7.9.4

```
xstra> mkdir rep1 ; cd rep1
xstra> touch fich1 fich2 fich11 fich12 ficha ficha1 \
       fich33 .fich1 .fich2 toto afich
xstra> ls fich*
fich1 fich11 fich12 fich2 fich33 ficha ficha1
xstra> ls fich?
fich1 fich2 ficha
xstra> ls fich[0 9]
fich1 fich2
xstra> ls .??*   § Ne pas prendre . et ..
.fich1 .fich2    § comparer à ls .*
xstra> ls [!f]*
afich toto
xstra> ls *fich*
afich fich1 fich11 fich12 fich2 fich33 ficha ficha1
xstra>
```

Exercice 7.9.5

```
xstra> alias en="env|sort|less"
xstra>
```

Exercice 7.9.6

```
xstra> cat > i
Salut
<ctrl d>
xstra> rm  i             § Essayez rm  i
Usage: rm [  Rfir] file
xstra>  § Le fichier  i est considéré par la commande rm
        §  comme étant l'option  i.
xstra> rm \ i          § L'utilisation des caractères de
Usage: rm [  Rfir] file § neutralisation (voir paragraphe
                       § 7.8) ne permet pas non plus
                       § la suppression du fichier  i.
xstra> rm ./ i  § La solution consiste à définir le
                § fichier à supprimer en chemin relatif
                § ou absolu, ce qui évite de commencer
                § par le signe  .
xstra>
```

Exercice 7.9.7

L'alias *p* est créé en utilisant le rappel de la dernière commande mémorisée et la notion de tube :

```
xstra> alias p="fc  s | less"
```

Cet alias peut être placé dans votre fichier *.bashrc*.

CHAPITRE 8

Exercice 8.3.1

Le fichier monscript contient :

```
#!/bin/bash
echo mon nom est $0
echo je suis appele avec $# arguments
echo qui sont : $*
```

Et voilà comment on l'invoque :

```
xstra>./monscript Bienvenue dans le monde Linux
mon nom est ./monscript
je suis appele avec 5 arguments
qui sont : Bienvenue dans le monde Linux
xstra>./monscript 'Bienvenue dans le monde Linux'
mon nom est ./monscript
je suis appele avec 1 arguments
qui sont : Bienvenue dans le monde Linux
```

Exercice 8.3.2

Le fichier papa contient :

```
#!/bin/bash
echo avant appel du fils
echo TERM = $TERM
echo PWD = $PWD
./fiston
echo apres appel du fils
echo TERM = $TERM
echo PWD = $PWD
```

Le fichier fiston contient :

```
#!/bin/bash
echo je suis le fils
echo mon heritage est:
echo TERM = $TERM
echo PWD = $PWD
echo je change
TERM='riendutout'
cd /tmp
```

```
echo TERM = $TERM
echo PWD = $PWD
echo je me termine
```

L'exécution donne les résultats suivants :

```
xstra> ./papa
avant appel du fils
TERM   vt100
PWD   /home/moi/testunix
je suis le fils
mon heritage est:
TERM   vt100
PWD   /home/moi/testunix
je change
TERM   riendutout
PWD   /tmp
je me termine
apres appel du fils
TERM   vt100
PWD   /home/moi/testunix
xstra>
```

Exercice 8.3.3

```
#!/bin/bash
exec 2>/dev/null # redirige stderr pour toute la suite
# au cas ou le script est invoque sans argument $1
# n'existe pas, la commande suivante devient  cd .
# voir paragraphe 8.2.6b
cd ${1: .}
for i in * ; do
if [  d $i ] ; then
    echo "$PWD/$i/    <   repertoire"
    $0 $i          # le script s'invoque lui meme
else
    echo $PWD/$i
fi
done
```

Exercice 8.3.4

```
#!/bin/bash
# Si ce script a pour nom : rename
# rename .c .bak
# fait ce que DOS fait en : rename *.c *.bak
#
if [ $#  ne 2  ] ;then
echo  "invoque par rename '.c' '.bak'"
exit 0
fi
```

```
for i in *$1 ; do
    mv $i $(basename $i $1)$2
done
```

Exercice 8.3.5

```
#!/bin/bash
# ce script doit etre invoque avec un seul argument
# svi fich
case $# in   # Doit etre invoque avec un seul argument.
1) if [[ $1 != *.bak ]] ; then # vix fich1.bak ouvre
                               # fich1.bak en lecture seule
     if [ !  s $1 ] ; then   # si fich1 n'existe pas,
       vi $1                 # on le cree
       exit
     fi
     if [  f $1 ] ; then      # si fich1 est un nom
       cp $1 $1.bak           # de fichier on sauvegarde
       if [ $? != 0 ] ; then  # et on verifie
         echo "Erreur : impossible de creer $1.bak" >2
         exit 1
       else
         vi $1
         exit
       fi
     else
       echo "Erreur : $1 n\'est pas un fichier" >2
       exit 1
     fi
   else
     echo "ATTENTION : $1 est ouvert en lecture seule"
     sleep 3
     vi  R $1
     exit
   fi;;
*) echo "Erreur : svi fichier" >2
   exit 2;;
esac
```

Exercice 8.3.6

```
#!/bin/bash
# Pour lancer ce script une fois par semaine, (cron)
# voir corrige de l'exercice 12.6.1
# Les constantes T et MIN sont definies par des variables
typeset T=15         # Plus de 16 jours.
typeset  i MIN=4096  # 4096 octets = 1 bloc pour AIX
# Il est inutile de comprimer un fichier dont la taille
# est inferieure a un bloc : il occupera toujours un
# bloc sur le disque (voir chapitre 10)
```

```
cd
for i in $(find .  type f  atime +$T  size +$MINc  print)
do
  gzip $i
done
```

Exercice 8.3.7

```
#!/bin/bash
# ce script a pour nom process.
# Il montre l'utilisation de la substitution de commande
# et des tableaux.
# Ce script affiche les noms de vos processus.
# exemple: xstra> process
#
declare  i i=7
declare  i fin
declare  a PROCESSTAB
PROCESSTAB=($(ps  u $(id  u)))
FIN=${#PROCESSTAB[*]}
until [ $i  gt $FIN ] ;do
  echo "programme=${PROCESSTAB[$i]}"
  i=$i+4
done
```

CHAPITRE 9

Exercice 9.5.1

Dans le premier cas, *lpr* est invoqué avec en paramètres les trois noms de fichier. Dans le deuxième cas, la commande *cat* crée le flot à imprimer, ce flot est passé à l'entrée standard de *lpr*.

```
xstra> lpr  #2 bibal.h bibal.c README

xstra> cat bibal.h bibal.c README | lpr  #2
```

Exercice 9.5.2

Il suffit d'imprimer sur l'imprimante « void » :

```
xstra> ls  l /usr/bin | a2ps  2 -Pvoid
[ stdin (plain): 42 pages on 21 sheets]
[ Total: 42 pages on 21 sheets]
saved into the file `/dev/null'
[ 198 lines wrapped]
xstra> ls  l /usr/bin | a2ps  3 -Pvoid
[ stdin (plain): 28 pages on 10 sheets]
[ Total: 28 pages on 10 sheets]
```

```
saved into the file `/dev/null'
[ 198 lines wrapped]
```

CHAPITRE 10

Exercice 10.3.1

```
xstra> cd; echo tout va bien >fichtext
xstra> ln fichtext lientext
xstra> ls  il fichtext lientext
13407  rw r  r   2 xstra staff   13 Jan 11 15:26 fichtest
13407  rw r  r   2 xstra staff   13 Jan 11 15:26 lientest
xstra> chmod ug=rwx,o= fichtext
xstra> ls  il fichtext lientext
13407  rwxrwx    2 xstra staff   13 Jan 11 15:26 fichtest
13407  rwxrwx    2 xstra staff   13 Jan 11 15:26 lientest
xstra> echo tout va de mieux en mieux >>fichtext
xstra> cat lientext
tout va bien
tout va de mieux en mieux
xstra> rm fichtext
xstra> cat lientext
tout va bien
tout va de mieux en mieux
xstra> ln lientext /tmp/lientemp
ln: cannot create hard link '/tmp/lientemp' to
'lientetx': Invalid cross device link
```

Un lien dur correspond au fait que un seul inode porte deux noms différents. Du fait que la numérotation des inodes est interne à chaque file system, il est impossible de créer un lien dur entre deux noms appartenant à deux *file system* différents.

Exercice 10.3.2

```
xstra> cd; ls -ld .
drwx  x  x 8 xstra  staff   2560 Jan 11 14:14 ./
```

Mon répertoire d'accueil */home/xstra* porte plusieurs noms, 8 dans cet exemple, bien que je ne puisse pas créer de liens durs sur ce répertoire. En effet, il porte les noms suivants :

- *xstra* dans le répertoire */home*
- . dans le répertoire /home/*xstra*
- .. dans chaque sous-répertoire de */home/xstra*

Le champ "reference count" de *ls l* vaut donc :
2 (*xstra* et .) + le nombre de sous-répertoires de */home/xstra* (.. dans chaque sous-répertoire).

Exercice 10.3.3

```
xstra> echo vive Linux >fich1
xstra> ln  s fich1 sl
xstra> ls -ilF
total 8
398  rw r  r   1 xstra staff 10 Jan 11 15:52 fich1
407 lrwxrwxrwx 1 xstra staff  5 Jan 11 15:52 sl@  > fich1
  § le lien porte un autre inode, sa longueur est de 5
  § (f.i.c.h.1), ses permissions n'ont pas de valeur.
xstra> more sl
vive Linux
xstra> rm fich1
xstra> ls  ilF
407 lrwxrwxrwx 1 xstra staff 5 Jan 11 15:52 sl@  > fich1
xstra> more sl
sl: No such file or directory
  § Le lien pointe sur un nom inexistant (broken link)
xstra>
```

Exercice 10.3.4

Il existe une différence entre l'occupation disque et la taille réelle des fichiers. Ceci est dû à l'adressage en blocs (et fragments) sur le disque.

```
#!/bin/bash
declare  i nbrbyte=0
declare  i discused=0
declare  i n
for i in $(find .  type f  print)
do
  n=$(wc  c <$i)
  nbrbyte=$((nbrbyte+n))
done
echo Total bytes ... $nbrbyte
echo Disc used ......$(du  ks | cut  f1) Kbytes
```

CHAPITRE 11

Exercice 11.3.1

Un fichier d'extension *.tar.gz* ou *.tgz* est un fichier archivé par *tar* puis comprimé par la commande *gzip*. Il faut utiliser *tar* avec les options *–z* et *–t* pour examiner son contenu avant de l'extraire par *tar* option *–z* et *–x*.

```
xstra> tar -zt  f /tmp/prog.tar.gz | more
 § Le contenu du fichier tar s'affiche page par page.
 § Si le contenu est conforme à ce que l'on attend,
 § on peut l'extraire.
xstra> cd; mkdir prog; cd prog
```

```
xstra> tar -zx  f /tmp/prog.tar.gz
 § tar extrait le contenu du fichier archive
```

CHAPITRE 12

Exercice 12.6.1

Dans cet exemple l'utilisateur a pour login : xavier. Le fichier (script) */home/xavier/bin/menage* contient :

```
#!/bin/bash
# la commande xargs cmd construit une commande
# en combinant cmd et les parametres qu'elle
# lit sur son entree standard (voir man xargs)
# exemple :
# echo a b c d | xargs rm
# devient la ligne de commande
# rm a b c d

cd /home/xavier
TMP /tmp/menage.$$
rm  f $TMP
find .  type f  name core  atime +2  print >$TMP
find .  type f  name '*.tmp'  atime +2  print >>$TMP
find .  type f  name 'a.out'  atime +2  print >>$TMP
if [  s $TMP ] ; then
  xargs rm <$TMP
  mail  s "fichiers effaces" xavier <$TMP
fi
rm  f $TMP
```

Ce script est lancé en cron par la commande *crontab e* permettant d'écrire dans la crontab l'entrée suivante :

```
# mn h jourdumois mois jourdelasemaine commande
0 3 * * 1,2,3,4,5 /home/xavier/bin/menage >/dev/null
```

Remarquez que la sortie erreur standard n'a pas été redirigée. Les messages d'erreur, s'il y en a, seront envoyés à l'utilisateur par *mail*.

Exercice 12.6.2

Le shell script */home/xavier/bin/killprog* contient :

```
#!/bin/bash
MOI $(basename $0)       # le nom du script
TEMPO /tmp/$MOI.$$       # un nom de fichier temporaire
CLAVIER $(tty)           # mon tty, servira plus tard
if [ $# !  1 ] ; then    # un argument obligatoire
  ps  fu $(id  u)        # sinon, lister les process
  echo  e "\nUsage : $MOI nom du process a tuer"
```

```
  exit 2                # et sortir en erreur
fi
if ps  fC $1 >$TEMPO  # liste des process candidats
then                  # si au moins 1 de ce nom
  cat $TEMPO          # montrer les candidats
  awk 'NR>1' $TEMPO | while read MONUID PROCESSID RESTE
  do                  # demander confirmation pour chacun
    echo  n "KILL  KILL $PROCESSID ? : "
    read REPONSE <$CLAVIER   # ne PAS lire STDIN (pipe)
    if [  $REPONSE    'oui' ]   ; then
      if kill  KILL $PROCESSID ; then
        echo $PROCESSID killed # il est bien mort !
      fi
    fi
  done
else
  echo aucun process ne porte le nom $1
  rm  f $TEMPO
  exit 1
fi
rm  f $TEMPO
```

CHAPITRE 13

Exercice 13.3.1

```
xstra> cd;hostname
courlis
xstra> tar cf   . | gzip | rsh sapin  l pierrot \
       dd of /dev/rmt0 obs 20b conv sync
```

Bien sûr, il faut aussi que la confiance mutuelle soit établie, c'est-à-dire que les fichiers *~/.rhosts* sur les machines sapin contiennent la ligne :

```
courlis pierre
```

Exercice 13.3.2

```
xstra> ftp leonardo.u strasbg.fr
Connected to leonardo.u strasbg.fr.
220 leonardo FTP server (Version wu 2.5.0(1) Tue Sep 21 16:48:12
EDT 1999) ready.
Name (leonardo.u strasbg.fr:xstra): anonymous
331 Guest login ok, send your complete e mail address as pas
sword.
Password:  § saisie de votre e mail comme mot de passe
230 Guest login ok, access restrictions apply.
ftp> cd /pub/livre linux/examples
250 CWD command successful.
ftp> ls
```

```
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
. . . . . § suit une liste de noms dont le dernier :
tout.tar.gz
226 Transfer complete.
ftp> bin
200 Type set to I.
ftp> get tout.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for tout.tar.gz (5820
bytes).
226 Transfer complete.
5820 bytes received in 0.008273 seconds (687 Kbytes/s)
ftp> quit
221 You have transferred 5820 bytes in 1 files.
221 Total traffic for this session was 6407 bytes in 2 transfers.
221 Thank you for using the FTP service on leonardo.
221 Goodbye.
/home/xstra>
```

Exercice 13.3.3

La solution pourrait être tout simplement :

rsh autre machine man commande | less

Il est également possible de rediriger le résultat de la commande *man* vers un fichier temporaire et de le parcourir avec *less*.

Le fichier (script) *suman* contient :

```
#!/bin/bash
TEMPFILE=/tmp/remoteman.$$
rm  f $TEMPFILE # par precaution
# Ce script aura pour nom suman et hpman et liman
# par des liens durs
# Il prendra une decision selon le nom sous lequel
# il est invoque

MONOM=$(basename $0) # sous quel nom suis je invoque?
case $MONOM in
suman) REMOTEHOST=sapin;   REMOTELOGIN=pierrot;;
hpman) REMOTEHOST=courlis; REMOTELOGIN=pierre;;
liman) REMOTEHOST=dingo;   REMOTELOGIN=pcolin;;
esac
echo "man $1 sur $REMOTEHOST" >$TEMPFILE
rsh $REMOTEHOST  l $REMOTELOGIN man $1 >>$TEMPFILE 2>>&1
more  wvs $TEMPFILE
rm  f $TEMPFILE
```

Il suffit maintenant de créer des liens durs *hpman* et *liman* sur *suman* :

```
xstra> ln suman hpman
xstra> ln suman liman
```

Bien sûr, il faut aussi que la confiance mutuelle soit établie, c'est-à-dire que les fichiers *~/.rhosts* sur les machines sapin, courlis et dingo contiennent la ligne :

```
mickey pierre
```

Exercice 13.3.4

Dans cet exemple l'utilisateur a pour login : florent.

Première étape : création du fichier *~/.netrc* ayant le contenu suivant : (<TAB> représente le caractère tabulation, il ne doit pas être omis)

```
machine ftp.machine.domaine login anonymous password ftp
<TAB>macdef init
<TAB>lcd /home/florent/bidon
<TAB>cd /pub
<TAB>ascii
<TAB>get README
<TAB>cd /pub/shell
<TAB>binary
<TAB>get prog.tar.gz
<TAB>quit
```

Deuxième étape : lancer la commande *ftp* par *at*

```
xstra> at now + 8 hours
at> /usr/bin/ftp ftp.machine.domaine
at> <ctrl D>
Job florent.918048.a will be run at Wed Feb 3 03:14:08
xstra> at -l
florent.918048.a      Wed Feb  3 03:14:08 NFT 1999
xstra>
```

La sortie standard et la sortie erreur standard de cette commande *ftp* seront envoyées à l'utilisateur florent par *mail*.

Le programme *ncftp* est beaucoup plus pratique que le programme *ftp* de base, et il est installé dans la plupart des distributions Linux. La solution avec *ncftp* n'est pas différente.

CHAPITRE 14

Exercice 14.6.1

```
§ Recherche dans /etc/passwd de:
§ mon login :
xstra> grep $LOGNAME /etc/passwd
xstra:!:201:200:Alberto XSTRA:/home/xstra:/bin/bash

§ Nombre de lignes contenant MICHEL ou michel
xstra> grep -i MICHEL /etc/passwd | wc -l
3
```

```
§ Nombre de lignes contenant /home/
xstra> grep -c '/home/' /etc/passwd
27
xstra>
```

Exercice 14.6.2

```
xstra> fgrep . /etc/inetd.conf
§ attention au point avec grep
xstra> grep -cv '^#' /etc/inetd.conf
xstra> awk 'length>50' /etc/inetd.conf
xstra> awk '!/^#/ && length>50' /etc/inetd.conf
```

Exercice 14.6.3

La solution awk est la suivante:

```
xstra> awk '!/^$/ {n++;print n,$0}'
 § tapez plusieurs lignes dont des vides
 § terminez par <ctrl D>
 § observez le résultat
 § Sous linux, ce résultat est obtenu par :
xstra> cat  b
```

Exercice 14.6.4

Supprimer les lignes vides

```
xstra> grep  v '^$' <fich1 >fich2
```

Trouver les gens qui n'ont pas le téléphone

```
xstra> egrep '[^0 9]{8}$' fich1
```

Trouver les lignes non conformes

```
xstra> grep  v '^[A Z][a z]+ [A Z]+ [0 9]{1,2} \
[1 9][0 2]? 19[0 9]{2} F|M [0 9]{8}$' fich1
```

Cette solution est approximative. awk permet une analyse beaucoup plus fine. Le fichier check.awk contient :

```
{error=0
if (NF != 5) error++
if ($1 !~ /[A Z][a z]+/) error++
if ($2 !~ /[A Z]+/) error++
combien=split($3,ladat," ")
if (combien != 3) error++
if (ladat[1] < 1 || ladat[1] > 31 ) error++
if (ladat[2] < 1 || ladat[2] > 12 ) error++
if (ladat[3] < 1789 || ladat[3] > 2002 ) error++
if (length($4) != 1) error++
```

```
if ($4 !~ /[MF]/) error++
if (length($5) != 10) error++
if ($5 !~ /0[0 9]{9}/) error++
if (error != 0) print NR,error,$0
}
```

Ce qui signifie : pour chaque ligne (condition manquante) l'action à entreprendre consiste à : mettre la variable error à zéro, variable qui sera incrémentée pour chaque anomalie détectée. Les vérifications sont ensuite faites dans l'ordre.

Vérifier que la ligne comporte cinq champs
Le premier champ doit être une majuscule suivie de minuscules
Le deuxième champ doit être une suite de majuscules
Couper le troisième champ en sous-champs sur le caractère "-"
Vérifier qu'il y a trois sous-champs
Que ces trois sous-champs ont des valeurs plausibles
Vérifier que le quatrième champ contient un seul caractère
Et que ce caractère est "M" ou "F" et rien d'autre
Vérifier que le cinquième champ comporte dix caractères
Et qu'il contient 0 suivi de dix chiffres

La commande suivante liste les lignes non conformes, en les faisant précéder du numéro de ligne et du nombre d'erreurs détectées :

```
xstra> awk -posix -f check.awk annuaire.txt
1 1 Georges TARTEMPION 12 2 1967 M 038898422524
5 3 Antonio VAVILDA 16 5 1937 M
7 11 J'en ai assez de ce travail fastidieux!
9 11       ligne vide
10 3 Anatole SUISSE 1 2 1965 n 4
11 1 Antonino SZWPRESWKY 16 5 8937 M 0298358745
```

Exercice 14.6.5

Il faut tester si le septième champ du fichier */etc/passwd* est vide.

```
xstra> awk -F: '$7 = "" {print $1}' /etc/passwd
```

Exercice 14.6.6

Le fichier *chkmail.awk* contient :

```
/^From / {n++}
/^From: / {FROM=$0}
/^Subject: / {SUBJ=$0}
END {print n," messages";print FROM;print SUBJ}
```

Le fichier *checkmail* contient le script suivant :

```
#!/bin/sh
awk  f chkmail.awk /var/spool/mail/$USER
```

Le lancement de ce script donne le résultat suivant :

```
xstra> ./checkmail
3  messages
From: jpa@pinson.u strasbg.fr
Subject: ou en est le chapitre 14 ???
```

Exercice 14.6.7

Le script *checkmail* est modifié de la façon suivante :

```
#!/bin/sh
while true ; do
  awk  f chkmail.awk /var/spool/mail/$USER
  sleep 30
done
```

Et il est maintenant lancé en tâche de fond :

```
xstra> ./checkmail &
```

Exercice 14.6.8

Le script *checkmail* est modifié de la façon suivante :

```
#! /bin/sh
MABOITE=/var/spool/mail/$USER
STAMP=/tmp/timestamp.$USER
# ce fichier sert a connaitre l'instant auquel j'ai
# remarque une modification de ma boite aux lettres

rm -f $STAMP
while true ; do
  if [  s $MABOITE ] ; then
    if [ $MABOITE -nt $STAMP ] ; then
      clear;
      awk  f chkmail.awk $MABOITE
      touch $STAMP
    fi
  fi
  sleep 30
done
```

Exercice 14.6.9

```
xstra> xterm -geometry 60x7 0+0 +sb  e ./checkmail &
```

Exercice 14.6.10

Le fichier *splitmail.sed* contient :

```
#n
1,/^$/w _header
/^$/,$ {
```

```
/^$/d
w _body
}
```

La commande *sed* traitant le message est :

```
xstra> cat message | sed -f splitmail.sed
```

Le fichier *splitmail.awk* contient :

```
BEGIN {FICH = "_header"}
/^$/ {FICH = "_body"}
{print > FICH }
```

La commande *awk* traitant le message est :

```
xstra> cat message | awk -f splitmail.awk
```

Exercice 14.6.11

```
#!/bin/bash
if [ $#  lt 3 ] ; then # Il faut au moins trois arguments
  echo erreur : remplace 'ere' 'nchaine' file..
  exit 1
fi
ERE="$1"       # le premier arg est l'expression
REPLACE="$2"   # le deuxieme arg est la nouvelle chaine
SEP='%'        # nous utiliserons % comme separateur
# (le caractere / pourra apparaitre dans l'expression)
shift; shift
# maintenant, $1 contient le premier nom de fichier
# faire une boucle sur tous les fichiers passes
# en arguments
while [ z$1 != z ] # teste si $1 n'est pas vide
                   # (encore un fichier a traiter?)
do
  if [  f $1 ] ; then # $1 est un fichier
    sed "s${SEP}${ERE}${SEP}${REPLACE}${SEP}g" $1 \
            >/tmp/replace.$$
# le resultat est dirige vers un fichier
# temporaire qui est recopie sur le fichier
# d'origine si sed a effectue au moins une modification
        [ $? = 0 ] && cat /tmp/replace.$$ >$1
  else
    echo $1 n\'est pas un fichier, ignore
    # $1 peut etre un repertoire
  fi
  shift # pour passer au suivant, decaler les
        # arguments d'un pas vers la gauche
done    # et passer au suivant : fin de la boucle.
rm  f /tmp/replace.$$
# faire le menage quand tout est fini
```

CHAPITRE 15

Exercice 15.5.1

Nous allons détailler trois cas :

normal : vous avez des relations de confiance avec les membres de votre groupe, et les autres utilisateurs du système ne vous veulent pas de mal.

confiant : vous avez des relations de confiance avec tous les utilisateurs du système.

paranoïaque : vous avez modérément confiance dans les membres de votre groupe, et vous redoutez les autres utilisateurs (situation peu enviable humainement, mais jouable techniquement).

Voilà l'umask et les permissions des principaux fichiers et répertoires que vous pouvez adopter.

	normal	confiant	paranoïaque
umask	026	022	077
~/	rwx r x x	rwx r x r x	rwx r x
~/.profile	rwx r	rwx r r	rwx
~/.forward	rw	rw	rw
~/Mail/	rwx	rwx	rwx
~/bin	rwx r x x	rwx r x r x	rwx
~/.rhosts	rw	rw	r

CHAPITRE 16

Exercice 16.5.1

```
xstra> xhost +courlis.u strasbg.fr
xstra> rlogin courlis
password:   § le mot de passe ne laisse pas d'écho
courlis:xstra> declare -x DISPLAY=davinci.u strasbg.fr:0
courlis:xstra> xclock &
courlis:xstra>
```

Index

A

SCIENCES SUP

2e édition

Jean-Paul Armspach
Pierre Colin
Frédérique Ostré-Waerzeggers

LINUX
Initiation et utilisation

Cet ouvrage s'adresse aux étudiants, universitaires et ingénieurs, et plus généralement à toute personne désireuse d'acquérir une bonne maîtrise de Linux.

La présentation s'applique à toutes les distributions de Linux. Elle est illustrée de nombreux exemples et développe de manière progressive les points suivants : la connexion et les utilisateurs, le système de fichiers et les protections, l'éditeur de textes vi, l'interpréteur de commandes Bash, la programmation en Bash, les utilitaires d'impression et de sauvegarde, la gestion des processus et de l'espace disque, les utilitaires grep, sed, awk et les expressions régulières, les aspects réseau, l'interface graphique X11 et la sécurité du système.

À la fin de chaque chapitre les auteurs proposent une série d'exercices (50 au total) dont les corrigés détaillés figurent en fin d'ouvrage. Un index de plus de 400 entrées permet d'accéder rapidement à l'information cherchée.

Des mêmes auteurs :

Unix. Initiation et utilisation, 2e édition
Dunod, 1999. 302 pages.

JEAN-PAUL ARMSPACH
est ingénieur de recherche à l'université Louis Pasteur de Strasbourg.

PIERRE COLIN
est professeur à l'École Nationale Supérieure de Physique de Strasbourg.

FRÉDÉRIQUE OSTRÉ-WAERZEGGERS
est ingénieur système et administrateur réseau à l'université Louis Pasteur de Strasbourg.

MATHÉMATIQUES
PHYSIQUE
CHIMIE
SCIENCES DE L'INGÉNIEUR
INFORMATIQUE
SCIENCES DE LA VIE
SCIENCES DE LA TERRE

9 782100 076543

ISBN 2 10 007654 X

http://www.dunod.com

DUNOD