Cahiers de l'Admin

Collection dirigée par Nat Makarévitch

BSD

Emmanuel **Dreyfus**

Avec la contribution
d'Ollivier **Robert** et d'Olivier **Tharan**

2e édition

EYROLLES

Dans la collection

Les Cahiers de l'Admin

dirigée par Nat Makarévitch

Mac OS X Server

Jacques FOUCRY – N°11192, 2003.

On verra dans ce cahier que Mac OS X Server version 10.2 (Jaguar) facilite considérablement la vie de l'administrateur : outre ses fonctions d'administration et d'automatisation évoluées, il prend en charge tous types de clients (Mac OS, UNIX/Linux, Windows) et offre en standard la panoplie désormais indispensable d'outils Open Source MySQL, PHP, Samba, NFS, FTP, CUPS…

Sécuriser un réseau Linux 2e édition

Bernard BOUTHERIN, Benoit DELAUNAY - N°11445, 2004.

À travers une étude de cas générique mettant en scène un réseau d'entreprise, l'administrateur apprendra à améliorer l'architecture et la protection de ses systèmes connectés, notamment contre les intrusions, dénis de service et autres attaques : filtrage des flux, sécurisation par chiffrement avec SSL et (Open) SSH, surveillance quotidienne… On utilisera des outils Linux libres, réputés pour leur efficacité.

BSD 2e édition

Emmanuel DREYFUS – N°11244, 2003.

Ce cahier révèle les dessous d'UNIX et détaille toutes les opérations d'administration UNIX/BSD : gestion des comptes, initialisation de la machine, configuration des serveurs web, DNS et de messagerie, filtrage de paquets… Autant de connaissances réutilisables sous d'autres systèmes UNIX, et en particulier Linux.

Chez le même éditeur

J.-L. BÉNARD, L. BOSSAVIT, R. MÉDINA, D. WILLIAM

L'Extreme Programming

N. 11051, 2002, 350 pages

V. STANFIELD & R..W. SMITH

Guide de l'administrateur Linux

N°11263, 2003, 654 pages.

C. AULDS.

Apache 2.0 Guide de l'administrateur Linux.

N°11264, 2003, 582 pages.

G. MACQUET – **Sendmail**

N°11262, 2003, 293 pages.

C. HUNT. – **Serveurs réseau Linux.**

N°11229, 2003, 650 pages.

R. RUSSELL et al. – **Stratégies anti-hackers**

N°11138, 2e édition 2002, 754 pages.

Dans la collection

Les Cahiers du programmeur

Mac OS X

Gestionnaire de photos avec Cocoa, REALbasic et WebObjects.

Alexandre Carlhian, Jacques Foucry, Jean-Philippe Lecaille, Jayce Piel – avec la collaboration d'Olivier Gutknecht – N°11192, 2003.

Réalisez un gestionnaire de photos consultable via le Web avec Cocoa et Objective-C, REALbasic et WebObjects.

PHP 5

Application de chat avec PHP 5 et XML

Stéphane Mariel – N°11234, 2004.

De la conception à l'exploitation, on créera une application de discussion en ligne en PHP 5 respectant les méthodes éprouvées du développement web : architecture MVC, conception modulaire avec les interfaces, sessions, gestion d'erreurs et exceptions, échanges XML et transformations avec DOM, XPath et SimpleXML.

PHP (2)

Ateliers Web professionnels avec PHP/MySQL et JavaScript.

Philippe CHALEAT et Daniel CHARNAY – N°11089, 2002.

En une douzaine d'ateliers pratiques, allant de la conception d'aides multi-fenêtrées en JavaScript à la réalisation de services Web, en passant par les templates PHP et les annuaires LDAP, on verra qu'autour de formulaires HTML, on peut sans mal réaliser des applications légères ergonomiques et performantes.

PostgreSQL

Services Web professionnels avec PostgreSQL et PHP/XML.

Stéphane MARIEL – N°11166, 2002.

Ce cahier montre comment réaliser simplement des services Web avec PostgreSQL, PHP et XML. Le développeur apprendra à modéliser sa base, tirer parti de la richesse de PostgreSQL (transactions, procédures stockées, types de données évolués…), optimiser ses performances et en automatiser l'administration, sans oublier la réalisation d'un affichage dynamique avec XSLT.

Dans la collection

Accès libre

Débuter sous Linux.

S. Blondeel, H. Singodiwirjo. – N°11349, 2004, 328 pages.

Cet ouvrage guidera des utilisateurs motivés, qu'ils aient ou non utilisé un système MS-Windows, vers la connaissance, l'utilisation et l'administration du système libre et gratuit Linux, qui offre, outre sa puissance, les indispensables de tout poste de travail : traitements de texte, tableurs, mail, navigation Web, messagerie instantanée, jeux, multimédia.

OpenOffice.org efficace.

S. Gautier, C. Hardy, F. Labbe, M. Pinquier.

N°11348, 2004, 350 pages.

OpenOffice.org, suite bureautique gratuite tournant sous Windows, Linux et Mac OS X, inclut tous les modules habituels : traitement de texte, tableur de calcul, présentation, dessin, formules… Écrit par les chefs de file du projet français OpenOffice.org, cet ouvrage montre comment optimiser son environnement de travail et l'utilisation de chaque module d'OOo, comment s'interfacer avec des bases de données telle MySQL, et offre enfin un précis de migration.

Réussir un site Web d'association...

avec des outils gratuits.

A.-L. Quatravaux, D. Quatravaux. – N°11350, 2004, 280 pages.

Souvent peu dotée en moyens, une association doit gérer ses adhérents et membres, faciliter l'organisation du travail entre eux, être visible et soigner son image. Depuis le choix de l'hébergement jusqu'au référencement, en passant par la personnalisation graphique sous SPIP, la configuration d'un serveur d'e-mailing et la création de listes de diffusion, ce livre explique comment gagner un temps précieux en confiant ces tâches à des outils adéquats et gratuits.

Emmanuel **Dreyfus**

Cahiers de l'Admin BSD

2e édition

Les dessous d'Unix

Collection dirigée par Nat **Makarévitch**

Avec la contribution d'Ollivier **Robert**, Olivier **Tharan**
Florence **Henry** et Sébastien **Blondeel**

EYROLLES

Remerciements

Cet ouvrage ne serait pas ce qu'il est sans le travail des relecteurs. Un grand merci donc à Manuel Bouyer, Emmanuel Eisenstaedt, Gérard Henry, et Xavier Humbert pour les remarques et suggestions qu'ils ont pu apporter à ce livre. Merci aussi à Sébastien Blondeel, Ollivier Robert, et Olivier Tharan, qui furent les acteurs de la grande relecture finale. S'il reste des coquilles, c'est de leur faute !

Ce livre est écrit en DocBook XML avec l'éditeur **vi**. Il est difficile d'imaginer le résultat final lorsque l'on travaille avec de tels outils, mais les formats XML ont pour principal avantage d'être faciles à transformer.

Le document DocBook est ainsi transformé en un document LaTeX grâce aux feuilles de styles XSL du projet DB2LaTeX, de Ramon Casellas et James Devenish, dont il faut au passage saluer le remarquable travail. Le fichier LaTeX obtenu est ensuite traité par une feuille de style pour produire une mise en page proche de la maquette de la collection mise au point par l'éditeur. Je dois cette feuille de style à Florence Henry, que je remercie chaleureusement.

Avant-propos

Unix a la peau dure. Alors que ses versions pour les stations de travail des grands constructeurs (IBM, HP, SGI, Sun) sont sur le déclin, on le voit ressurgir en force à travers MacOS X et les Unix libres comme GNU/Linux et les BSD.

Unix a bien des qualités : fiable, performant, flexible (il a survécu à plus de 30 ans d'évolution informatique), et gratuit avec les Unix libres. Hélas, Unix a un défaut : c'est un système complexe.

Mais cette complexité est contrebalancée par une autre de ses qualités : Unix est transparent. On peut comprendre pourquoi et comment les choses se passent sous Unix, ce qui n'est pas forcément le cas sur d'autres systèmes. Ceux qui ont déjà écumé la base de connaissances de Microsoft à la recherche du nom de la clef de base de registres à changer pour modifier un comportement donné apprécieront.

Sa complexité est donc maîtrisable, au prix d'un investissement personnel : la pratique. Ce livre jette les bases pour le lecteur disposé à y consacrer du temps.

Nous verrons les bases d'administration système Unix dans le cas des BSD. Le chapitre 1 présente la famille Unix et la place que les systèmes BSD y occupent. On y expliquera également pourquoi les BSD sont de bons systèmes d'apprentissage et de bons choix en production. Le chapitre 2 présente l'étude de cas familière aux habitués de la collection.

Nous enchaînerons sur les bases de l'utilisation d'une machine Unix au chapitre 3, pour continuer avec l'installation au chapitre 4. Cet ordre peut surprendre, mais il faudra parfois connaître les bases pour pouvoir installer.

Les chapitres 5 à 8 traitent d'administration système Unix. Nous exposons d'abord la procédure de démarrage de la machine, bien utile à connaître pour résoudre les problèmes en cas de panne. Vient ensuite la présentation de quelques tâches quotidiennes sur une machine Unix : gestion des utilisateurs, des groupes, et des permissions. Enfin, nous abordons les tâches de configuration les plus fréquentes sur une station Unix : le réseau, l'interface graphique, l'intégration dans un réseau de machines Unix.

Le chapitre 10 traite de la configuration de systèmes BSD en pare-feu, domaines où ils sont très prisés. La compréhension de ce type de configuration exige des connaissances assez approfondies sur les réseaux locaux et TCP/IP, thèmes qui font l'objet d'une rapide introduction au chapitre 9 .

Le chapitre 11 explique l'installation de logiciels de tierces parties, par recompilation manuelle ou via un système de paquetages. Sa lecture permettra d'installer les nombreux logiciels libres disponibles gratuitement.

VOCABULAIRE **Logiciel tierce partie**

Ce terme désigne les logiciels non fournis dans le système BSD mais écrits par des tiers. Des milliers de logiciels libres et propriétaires sont ainsi disponibles.

Retour sur un plan plus opérationnel : le chapitre 12 détaille la configuration de services classiques de l'Internet ou de l'Intranet : le Web, le DNS, et la messagerie.

Enfin, le chapitre 13 analyse les catastrophes : il passe en revue différentes difficultés qui peuvent se présenter sur des systèmes Unix, et propose quelques solutions. Il traite des mises à jour de sécurité, des sauvegardes, et de la surveillance du système.

Ce que vous ne trouverez pas dans cet ouvrage

Ce volume ne comporte pas de fichier de configuration prêt à l'emploi ni de recette universelle. L'apprentissage de l'administration Unix demande des efforts et de la pratique : c'est en écrivant des fichiers de configuration et en mettant au point ses propres solutions que l'on devient un administrateur Unix confirmé. Ce livre n'est pas un livre de recettes ; il tente plutôt d'aider à comprendre.

On ne trouvera pas non plus ici une compilation de la documentation existante, ni des modes d'emploi « pas à pas » de mise en place d'une fonction précise. Cet ouvrage renverra souvent aux sources d'information en ligne, sans s'y substituer, pour éviter une obsolescence rapide : le Web est toujours plus à jour que les livres.

Ce que vous trouverez dans cet ouvrage

Ce livre présente des informations sur l'administration et le fonctionnement des systèmes Unix en général, et sur les systèmes BSD en particulier. Lorsque cela

est possible, il mettra en exergue les différences entre Unix distincts pour que le lecteur acquière des connaissances réutilisables d'un système à l'autre.

Certains chapitres valent donc pour la plupart des Unix. D'autres, spécifiques aux systèmes BSD, traitent de domaines où les systèmes Unix diffèrent beaucoup entre eux. Les chapitres sur l'installation, la configuration d'une station, et la mise en place d'un pare-feu sont ainsi largement orientés BSD.

Des mises en gardes évoquent les chausse-trapes qui piègent fréquemment les débutants sous Unix : on proposera des solutions pour les éviter ou contourner.

À qui s'adresse cet ouvrage ?

Ce livre s'adresse à qui désire acquérir de solides bases d'administration système Unix, dans le cadre d'une utilisation domestique comme professionnelle. Les études de cas proposées vont de la mise en place d'un pare-feu personnel à l'intégration de serveurs d'entreprise pour des réseaux éventuellement hétérogènes. Beaucoup d'autres choses sont possibles, moyennant de prendre le temps de se plonger dans la configuration de la machine.

Ce manuel cible des débutants sous Unix, dotés d'une bonne culture informatique ; il se veut le compagnon d'un apprentissage pratique. Le lecteur prêt à installer un système BSD sur une machine et à passer du temps à le manipuler pourra réellement rompre l'os et profiter de la substantifique moelle.

Quiconque se sera reconnu dans cette description pourra sans tarder se précipiter au chapitre 1 !

Conventions typographiques

Un certain nombre de conventions typographiques ont été adoptées pour faciliter la lecture de ce livre et permettre l'identification rapide de certains éléments.
Les noms ou expressions en langue étrangère apparaissent comme ceci : *if it is not broken, do not fix it*.
Les noms de commandes apparaissent ainsi : **`disklabel`**, et les options qui leur sont associées ainsi : **`-e -I`**. Les noms de fichiers sont imprimés ainsi : /etc/passwd.
Les extraits de fichiers sont rendus dans le style suivant :

```
if [ "$1" = autoboot ]; then
        autoboot=yes
        rc_fast=yes
fi
```

Et les sessions interactives sont rendues ainsi :

```
$ grep manu /etc/passwd|awk '{print $3}'
manu:*:500:500:Emmanuel Dreyfus:/home/manu:/bin/ksh
$ grep manu /etc/passwd|awk -F: '{print $3}'
500
```

Dans ces sections, et dans le reste du livre, ce qui doit être saisi par l'utilisateur apparaît ainsi : **`shutdown -r now`**, et ce qui est affiché par la machine apparaît ainsi : `ksh: help: not found`.

Table des matières

1

Unix et BSD

La famille Unix est très nombreuse. Essayons d'en démêler les branches pour comprendre ce qui lie des systèmes tels que les BSD, Linux, et les Unix constructeurs.

SOMMAIRE

MOTS-CLEFS

- Unix
- Systèmes BSD

Un peu de généalogie

Vous connaissez probablement déjà un peu Unix ou Linux, car vous avez choisi de feuilleter ce livre. Linux fait beaucoup parler de lui ces derniers temps ; on peut même lire dans certains journaux qu'il saura délivrer la bureautique du joug microsoftien. C'est tout le mal qu'on lui souhaite. Unix reste moins connu du grand public, qui a parfois du mal à se figurer en quoi il consiste exactement, et quels sont les liens qui le rattachent à Linux. Quant aux systèmes BSD, ils ne sont souvent connus que des spécialistes. Essayons de dissiper le brouillard qui semble entourer tout cela.

Aux temps préhistoriques

Unix est un système d'exploitation dont la première version date de la fin des années 1960. Il a été mis au point par *Bell Labs*, le centre de recherche de l'opérateur téléphonique historique américain *American Telephone and Telegraph* (AT&T). Très rapidement, Unix a été distribué à l'extérieur d'AT&T sous forme de code source. C'est ce qui a causé l'apparition d'une véritable famille de systèmes Unix, chaque entreprise détentrice d'une licence Unix faisant évoluer le système en ajoutant ses idées et en reprenant celles du voisin.

Une des plus anciennes branches de la famille Unix a été développée à l'Université de Californie à Berkeley (UCB), à partir de 1977. Ces systèmes étaient connus sous le nom de *Berkeley Software Distribution* (BSD). À bien des égards, BSD était la branche de recherche d'Unix. On lui doit l'intégration de nombreuses fonctionnalités, dont par exemple l'implémentation originale de TCP/IP, dans 4.2BSD en 1983. L'Unix BSD a été utilisé comme source de nouveautés ou tout simplement comme point de départ par de nombreux systèmes Unix.

Une autre branche assez répandue est la branche UNIX System V, largement vendue par AT&T à des constructeurs de machines tels que Sun, IBM, HP, ou SGI. Chacun de ces constructeurs a ensuite créé son Unix à partir de cette première version.

Des Unix libres

On compte aujourd'hui plusieurs systèmes Unix libres. Trois d'entre eux descendent de BSD : NetBSD, FreeBSD, et OpenBSD. Le système GNU/Linux est quant à lui une ré-implémentation complète dont l'inspiration provient surtout d'UNIX System V.

Il existe d'autres systèmes Unix libres. Citons Darwin, qui constitue les couches Unix de MacOS X, ou des systèmes plus expérimentaux tels que xMach ou GNU/-Hurd.

CULTURE **Les sources**

Les sources d'un programme sont un ensemble de fichiers contenant du code, que l'on compile pour obtenir le programme exécutable. Unix a toujours eu une culture de distribution sous forme de code source : les entreprises ayant acquis la licence adéquate y avaient accès, ce qui leur permettait de le modifier à leur guise.
La forme binaire est l'autre forme de distribution. Le programme, déjà compilé, est prêt à l'emploi, mais on ne peut pas le modifier.

CULTURE **Les descendants de BSD et de System V**

À la fin des années 1980, la famille Unix commence à s'étoffer. On y trouve les dérivés de System V, comme AIX d'IBM et HPUX de HP. Les dérivés de BSD sont assez nombreux : IRIX de SGI, Digital UNIX de Digital, SunOS de Sun, NeXTStep de NeXT… Certains changent de famille : SunOS et IRIX deviennent par la suite des dérivés de System V, mais en général, tout le monde s'inspire des idées du voisin.

VOCABULAIRE **Linux et GNU/Linux**

Quelle est la différence entre Linux et GNU/-Linux ? Linux est le noyau du système, c'est-à-dire le composant qui attribue les ressources de la machine à tous les programmes, alors que GNU/Linux est le système d'exploitation dans son ensemble, donc le noyau flanqué de nombreux programmes utilitaires.

Quel système est Unix ?

La propriété intellectuelle liée à Unix est passée dans les mains de nombreuses sociétés, dont Novell, qui en 1993 revend le droit de distribuer le code source d'Unix à SCO (*the Santa Cruz Operation*), et la marque déposée Unix au consortium X/Open (lequel sera rebaptisé plus tard *Open Group*).

L'*Open Group* a pour but la standardisation des systèmes Unix. Il publie pour cela des normes dictant les comportements des commandes et des interfaces de programmation. La détention de la marque déposée Unix permet à l'*Open Group* de n'accorder la dénomination « Unix » qu'aux systèmes conformes à ces normes.

Il existe ainsi plusieurs manières de définir Unix, ce qui permet d'alimenter des débats sans fin. Certains sont pour une interprétation stricte de la marque : sont des Unix les seuls systèmes qui ont passé la certification de l'*Open Group* donc présentant la marque Unix. À ce compte, seuls les Unix des constructeurs sont des Unix ; GNU/Linux et les BSD libres n'en sont que des dérivés.

D'autres retiennent l'origine, et accordent le titre d'Unix aux BSD libres, car ils descendent de BSD, qui lui-même descend de l'Unix originel. Mais ils refusent à GNU/Linux l'appellation Unix, puisqu'il a été créé de toutes pièces, sans code issu de l'Unix originel. Ce point de vue amène à différencier Linux et Unix.

Enfin, d'autres vont retenir l'aspect familial, le plus large, et appeler Unix les systèmes ayant un ensemble de comportements et de fonctionnalités en commun. Des systèmes comme GNU/Linux ou les BSD libres font ainsi partie de la famille Unix. C'est ce point de vue que nous retiendrons dans ce livre, afin d'éviter d'avoir à faire d'incessantes distinctions entre « systèmes Unix » et « systèmes dérivés d'Unix ».

Pour résumer, rechercher un véritable système Unix aujourd'hui revient à chercher un véritable Gaulois en France. Tout a progressé, tout s'est mêlé, et pour utiliser Unix, il faut choisir un système Unix particulier. Chaque système contient des éléments issus du tronc commun Unix, et d'autres éléments qui lui sont propres. Dans cet ouvrage, on a choisi de s'intéresser aux systèmes BSD ; nous expliquerons pourquoi dans la section suivante.

CULTURE **SCO et Novell**

À l'heure où ces lignes sont écrites, une guerre de communication fait rage entre SCO et Novell, chacun prétendant avoir des droits sur les sources d'Unix. Cette discorde s'inscrit dans une affaire mettant en jeu SCO contre IBM, pour une obscure affaire de portions du code source d'Unix qui auraient été incluses dans Linux.
Au fil des semaines, la plus grande confusion s'instaure dans cette affaire, et l'auteur ne sait plus très bien quels droits SCO détient réellement sur les sources d'Unix. Les procès à venir et les précisions qu'apportera SCO devraient dissiper les malentendus.

ATTENTION **Un point de vue de plus**

Mentionnons également le cas des nombreux enquêteurs qui vérifient la conformité d'une installation à l'informatique « standard », que tout le monde est censé pratiquer. Ceux-là doivent faire rentrer tout système dans une case : soit « Windows » (sous-entendu sur PC), soit « Unix » (sous-entendu sur station Unix constructeur), soit « Linux » (sous-entendu PC sous GNU/Linux).
La distinction qu'ils marquent entre « Unix » et « Linux » est un point de vue de plus à connaître, même s'il est assez inexact.

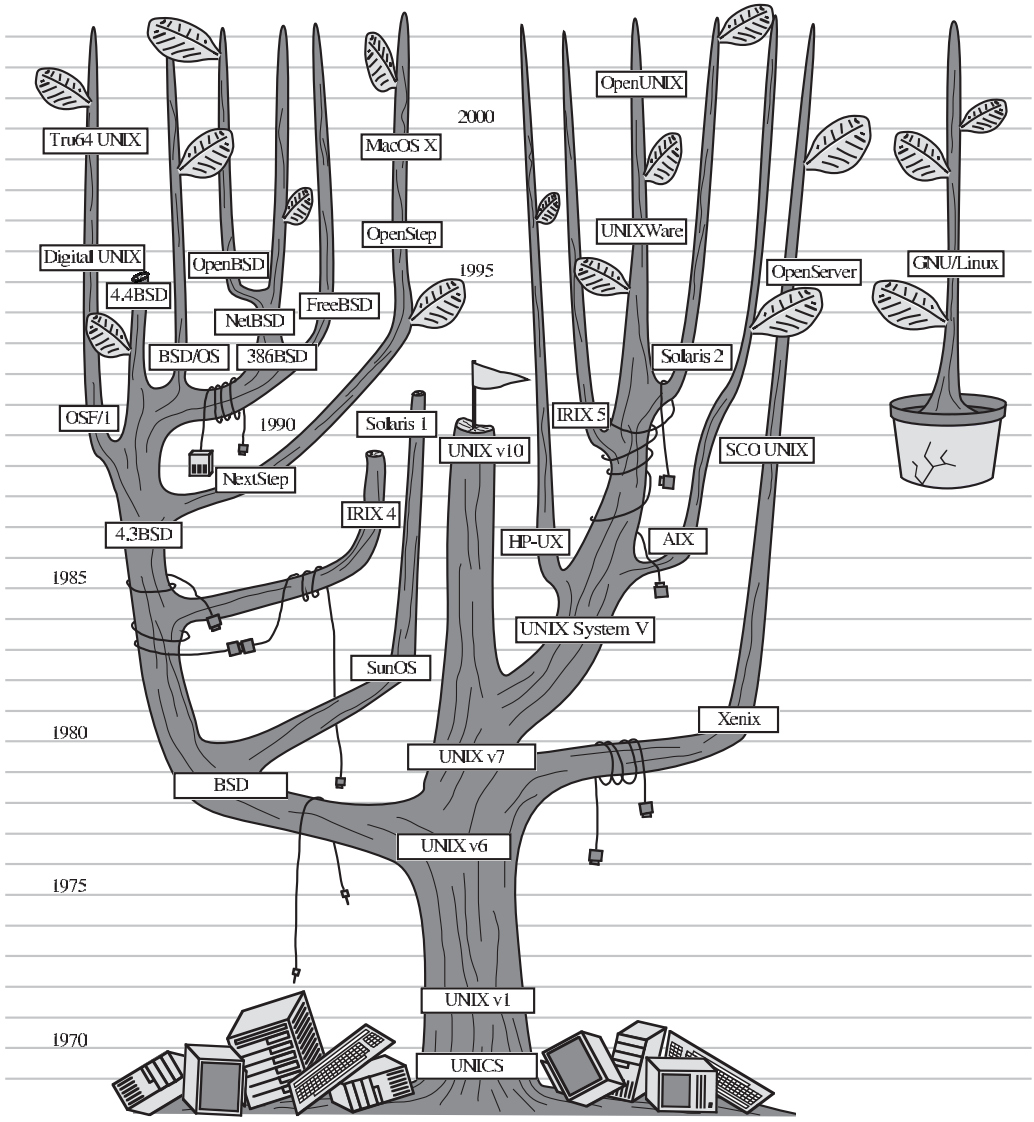
VOCABULAIRE **Unix-like**

En anglais, les systèmes dérivés d'Unix mais ne pouvant pas prétendre à la dénomination Unix *stricto sensu* sont souvent appelés « *Unix-like* », ce qui se traduit assez bien par « système à la Unix ».

B.A.-BA **Logiciel libre**

Le mouvement du logiciel libre a été lancé par Richard Stallman via la *Free Software Foundation* (FSF). Ces logiciels libres fournissent à leurs utilisateurs quatre libertés fondamentales : exécuter le programme, pour tous les usages ; étudier le fonctionnement du programme et l'adapter à ses besoins ; redistribuer des copies du programme ; et améliorer le programme et publier ces améliorations. L'accès au code source du programme est une condition nécessaire.
Un mouvement plus récent, l'*Open Source Initiative* (OSI), propose une définition en dix points qui dans la pratique est quasiment équivalente à celle du logiciel libre selon la FSF. Le contraire de « logiciel libre » est « logiciel propriétaire » : est propriétaire tout logiciel qui n'est pas libre, car il ne remplit pas au moins l'une des quatre libertés fondamentales.
Le projet lancé par Richard Stallman s'appelle GNU (GNU N'est pas Unix). Le but était tout d'abord de concevoir un système d'exploitation libre complet. De nombreux logiciels à ce titre nécessaires furent développés mais le noyau manqua jusqu'à l'apparition de Linux, en 1991. Les logiciels GNU fonctionnent désormais avec d'autres noyaux (Hurd, NetBSD). La FSF s'intéresse désormais aux applicatifs et à la documentation libre.

CULTURE **Un arbre généalogique simplifié d'Unix**



Une représentation en arbre est assez inadéquate pour montrer toutes les influences d'une branche à l'autre : les branches BSD et System V ont régulièrement alimenté les autres Unix, et les récupérations de code entre Unix libres sont monnaie courante.
Les migrations de BSD vers System V sont également représentées de façon peu satisfaisante. IRIX et Solaris n'ont bien entendu pas tout abandonné lors de cette transition. La représentation des branches BSD de ces systèmes comme des branches mortes est donc excessive.
Enfin, il est nécessaire de mentionner le fait que la taille des branches ne représente pas leur importance culturelle ou en nombre d'installations. Les feuilles et les câbles sont purement décoratifs.

Les systèmes BSD

Cet ouvrage aborde des bases d'administration système Unix, en s'appuyant sur les systèmes NetBSD, FreeBSD et OpenBSD. Contrairement à ce que la similitude de leurs noms pourrait laisser croire, ces trois systèmes sont bien distincts, tant au niveau du code source que des fonctionnalités qu'ils proposent.

Ces systèmes ont néanmoins quelques points communs. Un air de famille, pourrait-on dire. Les BSD sont des systèmes Unix : ils sont performants, fiables, et mûrs. Ce sont des systèmes modernes, qui prennent en charge des technologies récentes, comme USB ou IP version 6. Ils sont capables de faire fonctionner des applications telles qu'OpenOffice.org, Mozilla, KDE, ou GNOME, tout comme ils peuvent servir de nombreux protocoles : le Web, FTP, le DNS, les news, etc.

Tous trois sont de plus des systèmes Open Source, libres et gratuits : tous leurs sources et binaires peuvent être téléchargés librement sur l'Internet.

Les trois BSD partagent encore le même mode de développement centralisé, où une unique équipe de développeurs écrit l'ensemble du système et l'intègre en une distribution. Par opposition, les différents éléments des systèmes GNU/Linux sont développés par des équipes nouant peu de liens. Des projets rassemblent ces morceaux et apportent leur propres contributions pour constituer des « distributions Linux ». Le mode de développement des systèmes BSD leur confère ainsi une grande qualité dans l'intégration et la cohérence du système.

En conséquence, et contrairement aux distributions GNU/Linux, les systèmes BSD n'ont chacun qu'une seule distribution : il existe un seul NetBSD, un seul FreeBSD, et un seul OpenBSD. Les distributions GNU/Linux, quant à elles, sont très nombreuses : citons Red Hat, Debian, Slackware, SuSE, etc. On peut y voir un avantage ou un handicap. Quoi qu'il en soit, le dilemme du choix d'une distribution ne se posera pas à ceux qui auront choisi BSD.

Un des avantages principaux des BSD est d'avoir su rester assez proches du tronc commun Unix. Ces systèmes introduisent peu de commandes ou de formats de fichiers particuliers, ce qui rend les compétences acquises sur un système BSD facilement transférables sur un autre Unix, tel que GNU/Linux ou Solaris. J'irai

CULTURE **Histoires de familles**

Il existe d'autres systèmes BSD. Citons Darwin, qui constitue les couches Unix de MacOS X, ou le projet xMach. Ces deux systèmes sont Open Source, mais ils sont assez différents de NetBSD, FreeBSD et OpenBSD. Ils reposent en effet sur le micro-noyau Mach plutôt que sur un noyau Unix monolithique traditionnel.
On trouve également des systèmes Open Source dérivés de FreeBSD et OpenBSD : DragonflyBSD, fondé par une figure historique de FreeBSD pour cause de désaccords sur des choix techniques, et EkkoBSD, démarré à partir des sources d'OpenBSD pour pouvoir travailler « plus démocratiquement ». Ces systèmes bénéficient pour l'instant de très peu de moyens humains, et il est difficile de prédire leur évolution.
Enfin, il existe également un BSD propriétaire : BSD/OS, distribué par la société BSDI, mais cette dernière a annoncé son intention d'en arrêter le développement.

CULTURE **Applications sous Unix**

Si vous êtes totalement étranger au monde Unix, les noms des applications citées ici ne vous diront peut-être rien. Sous Unix, pas de Microsoft Office, pas d'Internet Explorer, pas d'Outlook. Mais l'offre logicielle est vaste (et souvent gratuite). Associons donc quelques fonctions aux noms. OpenOffice.org est une suite bureautique se posant en concurrent de Microsoft Office (il en existe d'ailleurs une version pour Windows). Mozilla est un navigateur Web. KDE et GNOME sont des environnements de bureau évolués et intégrés.

CULTURE **Unix constructeur**

Par ce terme, on désigne les Unix développés par les fabricants de machines. Le tableau ci-dessous donne un aperçu rapide des Unix constructeur, des processeurs sur lesquels ils fonctionnent, et de leurs fabricants.

Fabricant	Système	Processeur
Sun	Solaris	Sparc
IBM	AIX	POWER/PowerPC
SGI	IRIX	MIPS
Digital/Compaq	Tru64 Unix	Alpha
HP	HPUX	PA-RISC/Itanium

même jusqu'à dire, à titre personnel, que j'ai plus appris sur GNU/Linux en utilisant BSD qu'en utilisant GNU/Linux lui-même...

Bien sûr, installer et administrer un BSD ne fera pas de vous un expert Unix tous systèmes confondus. La famille Unix comporte quelques moutons à cinq pattes, qui ne font rien comme les autres, tels qu'AIX d'IBM. Mais en maîtrisant l'administration d'un système BSD, vous ferez l'acquisition d'une culture Unix et de méthodes de travail qui vous permettront de prendre en main relativement facilement la plupart des autres systèmes Unix.

Voyons maintenant les principales caractéristiques de nos trois systèmes BSD.

NetBSD : la portabilité avant tout

NetBSD, fondé en 1993 à partir du projet 386BSD, est l'aîné des systèmes BSD existant actuellement. Il a pour but de créer un système d'exploitation Unix libre, gratuit, stable, performant, sûr, et surtout multi-plateformes.

Au moment de la version 1.0, NetBSD proposait déjà six architectures, utilisant quatre types de processeurs différents : amiga, hp300, i386, mac68k, pc532, et sparc. Une décennie plus tard, NetBSD 1.6 fonctionne sur plus de cinquante plateformes différentes et sur plus de dix types de processeurs.

NetBSD est donc le système de choix lorsqu'il faut redonner un second souffle à une vieille station dont le fabricant a abandonné la prise en charge dans les versions récentes de son système. NetBSD peut transformer un Macintosh II ou une vieille station Sun en un pare-feu ou un petit serveur Web tout à fait valable. C'est aussi le seul BSD capable de fonctionner sur des assistants personnels (PDA) tels que le Jornada de HP, que l'on peut admirer sur la figure 1.1.

Pour autant, il serait réducteur de cantonner NetBSD aux vieilles machines et aux PDA. NetBSD fonctionne très bien sur un certain nombre de serveurs à processeurs d'architecture 64 bits. Il fut par exemple le premier système libre à fonctionner sur stations alpha, ainsi qu'à exploiter ces machines en mode multi-processeurs. Tout aussi à l'aise sur compatibles PC, il peut être utilisé pour mettre en place des serveurs ou des stations de travail à peu de frais.

NetBSD est également très compatible. Avec lui-même tout d'abord : un grand soin est apporté à la compatibilité ascendante. Ainsi, NetBSD 1.6 est encore capable de faire fonctionner des programmes compilés pour NetBSD 0.8. NetBSD cultive aussi la compatibilité avec les autres systèmes : il est capable d'exécuter de nombreux programmes compilés pour d'autres Unix, comme par exemple, FreeBSD, GNU/Linux, Solaris, SCO Unix, Digital UNIX... Ceci lui permet de bénéficier de nombreuses applications qui ne lui étaient pas originellement destinées.

La grande portabilité de NetBSD résulte de l'attention portée à la qualité de son développement : avant tout, les choses y sont bien faites. C'est en adoptant les meilleures solutions – quitte à passer plus de temps à les mettre en œuvre – que l'équipe de NetBSD peut obtenir un système où les mêmes codes sources pilotent

VOCABULAIRE **Portabilité**

La « portabilité » est un anglicisme désignant la capacité d'un logiciel à fonctionner sur différents systèmes d'exploitation ou d'un système d'exploitation à fonctionner sur différentes plateformes matérielles, sous réserve de recompilation ou d'adaptations légères et simples, souvent automatisées.

CULTURE **386BSD**

386BSD est le maillon qui mène du système BSD, développé à Berkeley, aux BSD libres modernes. Ce projet a été abandonné après le lancement de NetBSD et FreeBSD.

ATTENTION **Compatibilité binaire**

La compatibilité binaire ne concerne que des programmes compilés pour le même processeur mais pour un système d'exploitation différent. Ainsi, NetBSD/i386 pourra exécuter les binaires Linux/i386, mais pas les binaires Linux/powerpc – et inversement.

VOCABULAIRE **Pare-feu**

« Pare-feu » est la traduction officielle de l'anglais « *firewall* », qui se traduirait littéralement par « mur antiflammes », terme désignant à l'origine une plaque placée entre le moteur et l'habitacle d'un véhicule afin de réduire la vitesse de propagation d'un incendie. Il existe d'autres traductions moins usitées, telles que « coupe-feu », « barrière de sécurité », ou encore le très élégant « garde-barrière ».

Figure 1–1 NetBSD sur le Jornada 720 de HP. La disquette sert uniquement d'indicateur de l'échelle

SUR LES AUTRES UNIX **NetBSD comparé à Linux**

En matière de portabilité, le seul concurrent sérieux de NetBSD est Linux. Lui aussi a été porté sur un très grand nombre de plateformes, mais ces différentes versions (appelées « ports ») de Linux partagent moins de code que les ports de NetBSD. Par exemple, ils partagent peu les pilotes de matériel, ce que les auteurs de NetBSD s'efforcent de faire. Davantage testé, NetBSD a donc toujours tendance à mieux fonctionner sur les plateformes les moins populaires, comme par exemple mac68k.

plus de 50 architectures différentes. On dispose ainsi d'un code source très bien écrit et agréable à lire, ce qui a son importance quand on souhaite le reprendre.

La compatibilité ascendante est une autre conséquence de cette politique, avec des implications immédiates pour l'administrateur. Ainsi, NetBSD lui épargne autant que possible le désagrément des programmes et des fichiers de configuration à mettre à jour en même temps que le système.

Le lecteur attentif aura peut-être remarqué un net parti pris en faveur de NetBSD dans cet ouvrage. Cela n'a rien de surprenant : l'auteur est membre de l'équipe de développement du projet NetBSD.

FreeBSD : le spécialiste du PC

Le projet FreeBSD a vu le jour peu après NetBSD, dans le but identique de fournir des modifications non officielles à 386BSD. Pourquoi démarrer un projet différent ? Parce que les motivations étaient distinctes. L'équipe de NetBSD voulait un système fonctionnant sur le plus grand nombre de machines possible ;

CULTURE **Des processeurs**

Outre les 80x86 d'Intel, qui équipent les PC classiques, et les 680x0 de Motorola, qui ont équipé de nombreuses machines, dont les anciens Macintosh d'Apple, le monde des processeurs regorge de puces créées par les fabricants de machines Unix. Citons le Sparc pour les stations Sun, le MIPS pour les stations SGI, le POWER et le PowerPC pour les stations IBM (également utilisés dans les nouveaux Macintosh), le PA-RISC et l'Itanium pour les stations HP, et l'Alpha et le Vax pour les stations Digital.
L'informatique embarquée a elle aussi apporté de nouvelles puces, comme le SuperH ou les ARM.

celle de FreeBSD préférait un système fonctionnant au mieux sur une plateforme donnée : le compatible PC.

FreeBSD s'est donc concentré sur l'optimisation pour les PC. Ainsi, il a été capable d'utiliser des PC multi-processeurs plusieurs années avant NetBSD. FreeBSD reconnaît également un grand nombre de cartes d'extension pour PC, beaucoup plus que NetBSD.

FreeBSD est aussi plus connu, ce qui lui confère davantage d'utilisateurs, et donc plus de bras pour réaliser des paquetages. Le système de paquetages de FreeBSD est par conséquent le plus riche des trois systèmes BSD. N'en concluez pas pour autant que les autres sont pauvres ou inutilisables !

Enfin, FreeBSD a une véritable avance sur les autres BSD en ce qui concerne la compatibilité binaire avec Linux. Il peut accueillir des programmes tels que Matlab ou Oracle pour GNU/Linux, alors que les autres BSD échouent faute de compatibilité binaire complète.

Finissons par un mot sur l'aspect multi-plateformes : FreeBSD s'est un peu diversifié. Il dispose d'une version pour machines Alpha bien rôdée, et des versions pour Sparc, PowerPC et Itanium, plus récentes – donc moins éprouvées.

CULTURE **Oracle et Matlab**

Oracle est un puissant logiciel de base de données, proposant des fonctionnalités très complexes comme la répartition d'une base de données sur plusieurs machines, pour gagner en fiabilité. Matlab est un environnement de calcul scientifique et technique précieux dans certains domaines. Ces deux logiciels sont propriétaires.

ATTENTION **Tout évolue rapidement**

C'est une caractéristique des systèmes libres : la situation peut changer très vite. Même si l'on peut supposer qu'au moment où vous lirez ces lignes, FreeBSD comptera toujours plus de paquetages que NetBSD et OpenBSD, il est possible qu'entre-temps quelqu'un ait décidé de travailler sur la compatibilité binaire de NetBSD ou OpenBSD, et que ces systèmes soient donc devenus capables d'exploiter Oracle ou Matlab compilés pour GNU/Linux.

OpenBSD : l'obsession de la sécurité

NetBSD et FreeBSD sont des projets distincts pour des raisons d'orientation technique, mais OpenBSD est né de différends d'ordre personnel. Le fondateur d'OpenBSD était un membre de l'équipe dirigeante de NetBSD et en a été évincé par ses pairs. En 1995, il a donc démarré son propre BSD : OpenBSD, un système reposant sur NetBSD 1.0.

Les partisans de chaque camp ont leurs versions des faits quant aux motifs de cette séparation. Pour bénéficier d'un point de vue neutre sur la question, le plus simple est de prendre son courage à deux mains et de lire les centaines de courriers électroniques des archives des listes de diffusion où les coups se sont échangés.

OpenBSD dérivant de NetBSD, il en est très proche. La séparation n'a pas eu lieu pour des motifs techniques, et ce fut la sécurité qui lui permit de se démarquer.

Au moment de sa sortie, OpenBSD bénéficiait d'un avantage certain sur ses deux cousins BSD : NetBSD et FreeBSD étaient basés aux États-Unis d'Amérique et devaient se conformer à des lois strictes, interdisant l'exportation d'algorithmes de cryptographie forte, alors assimilés à des armes de guerre. Installé au Canada, OpenBSD pouvait quant à lui distribuer librement tous les outils de cryptographie qu'il souhaitait. Cette situation a duré plusieurs années, conférant à OpenBSD un avantage indéniable sur le plan de la sécurité. Mais les lois américaines ont depuis été assouplies, et les BSD sont désormais tous trois distribués avec une panoplie complète d'outils de cryptographie forte.

OpenBSD a fait beaucoup d'efforts sur le plan de la sécurité, en particulier en conduisant un audit du code hérité de NetBSD, et en en corrigeant de très nombreuses failles. Mais il ne faut pas imaginer que les autres BSD se sont contentés

de rester passifs : dans chaque projet BSD, on trouve des développeurs qui surveillent de près les modifications des voisins, et qui réintègrent les corrections de bogues qu'ils observent. La recopie est de mise, c'est ainsi que le logiciel libre fonctionne.

Lors de la première édition de ce cahier, OpenBSD avait par exemple pris une avance significative sur NetBSD et FreeBSD pour réduire l'impact de certains trous de sécurité, en imposant une pile non exécutable sur les architectures qui le permettent. Les deux autres BSD ont depuis travaillé à l'intégration de cette fonctionnalité : les innovations sécuritaires sont trop intéressantes pour que le système où elles ont été mises au point puisse seul s'en prévaloir très longtemps.

On doit aussi au projet OpenBSD le développement d'outils de sécurité tels qu'OpenSSH, actuellement très utilisé même au-delà de la sphère BSD.

Tous ces efforts louables ont contribué à améliorer la sécurité des trois BSD et d'autres systèmes Unix, mais ils n'ont pas forcément fait d'OpenBSD un système plus sûr que les autres BSD. Le projet OpenBSD clame haut et fort sur sa page d'accueil n'avoir eu qu'un seul trou de sécurité depuis 1997 dans son installation par défaut. C'est une escroquerie intellectuelle : personne n'utilise un système dans sa configuration par défaut. On utilise une machine Unix pour employer des programmes clients et serveurs, et quand on les démarre, on quitte la configuration par défaut, exposant ainsi de nouveaux trous de sécurité. OpenBSD n'en a pas forcément moins que FreeBSD ou NetBSD, et pour cause : les applications que l'on emploie sur ces systèmes sont souvent les mêmes.

De plus, beaucoup de trous de sécurité sont ouverts par l'administrateur lui-même, quand il fait des erreurs de configuration et ouvre grand la porte aux pirates. Il ne faut donc pas se leurrer : même si le projet OpenBSD fait un travail remarquable sur le plan de la sécurité, ne croyez pas que ce système est intrinsèquement sûr. Comme tout programme écrit par des humains, il contient des erreurs, dont certaines ont des implications sur la sécurité. Utiliser OpenBSD ne vous dispensera donc pas de tenir à jour votre système, et comme les développeurs d'OpenBSD sont très forts pour trouver et corriger des bogues, les mises à jour seront nombreuses.

Pour ce qui ne relève pas de la sécurité, OpenBSD est un projet de taille nettement inférieure à FreeBSD et NetBSD. Le faible nombre de développeurs a pour conséquence un système de paquetages moins fourni que celui de NetBSD et FreeBSD – il n'est pas pour autant ridicule. L'utilisation d'OpenBSD comme machine de bureautique est toutefois plus difficile à mettre en œuvre qu'avec NetBSD et FreeBSD, à cause du faible nombre de paquetages disponibles pour cette utilisation.

Signalons un autre point important : OpenBSD assure très peu de compatibilité ascendante. À chaque nouvelle mise à jour, il faut suivre rapidement. Certains développeurs du projet OpenBSD prétendent que ces ruptures de compatibilité ont un but sécuritaire : forcer les gens à faire les mises à jour les contraint à récupérer les corrections des trous de sécurité dont souffrait la version précédente. On peut aussi imaginer que le faible nombre de développeurs n'aide pas à maintenir plusieurs versions du système d'exploitation à la fois.

PLUS LOIN **La pile et les débordements de variables**

Un trou de sécurité très classique consiste à oublier de vérifier les bornes d'une variable dans un programme. Un attaquant peut exploiter cet oubli en allant écrire à certains endroits non prévus de la zone de mémoire où sont stockées les variables locales (cette zone s'appelle la pile).
L'attaquant dépose du code sur la pile, et il ne lui reste plus qu'à trouver un moyen de le faire exécuter pour détourner le programme. En imposant une pile non exécutable, on minimise les conséquences d'un mauvais contrôle de bornes, puisque le code écrit sur la pile ne pourra pas être exécuté. Cette mesure est extrêmement efficace contre toute une classe d'attaques, mais elle ne peut être mise en œuvre qu'avec les processeurs capables de rendre une zone de mémoire accessible en lecture/écriture mais pas en exécution.

CULTURE **OpenSSH**

OpenSSH est une implémentation libre de SSH, un protocole permettant d'assurer des connexions sécurisées entre deux machines, pour faire par exemple de l'administration système à travers Internet.

CULTURE **Compatibilité ascendante**

Les logiciels, architectures, et systèmes d'exploitation évoluent au cours du temps. Quand plusieurs éléments fonctionnent de concert, ils doivent être compatibles. On appelle compatibilité ascendante le fait qu'un nouveau produit, mettant en place une nouvelle norme, continue à comprendre et interagir avec une norme plus ancienne, qu'il rend pourtant obsolète. Ainsi, du code compilé pour processeur i386 ou i486 fonctionnera sur des Pentium ; un lecteur de DVD comprendra les CD-ROM ; etc.
L'inconvénient de la compatibilité ascendante est évidemment le volume et la complexité sans cesse croissants de l'héritage à prendre en compte. Certains projets logiciels, plus soucieux de simplifier les choses que de ménager leurs utilisateurs, font régulièrement table rase du passé, imposant ainsi une mise à jour et une reconfiguration plus complètes.

Lequel est le meilleur ?

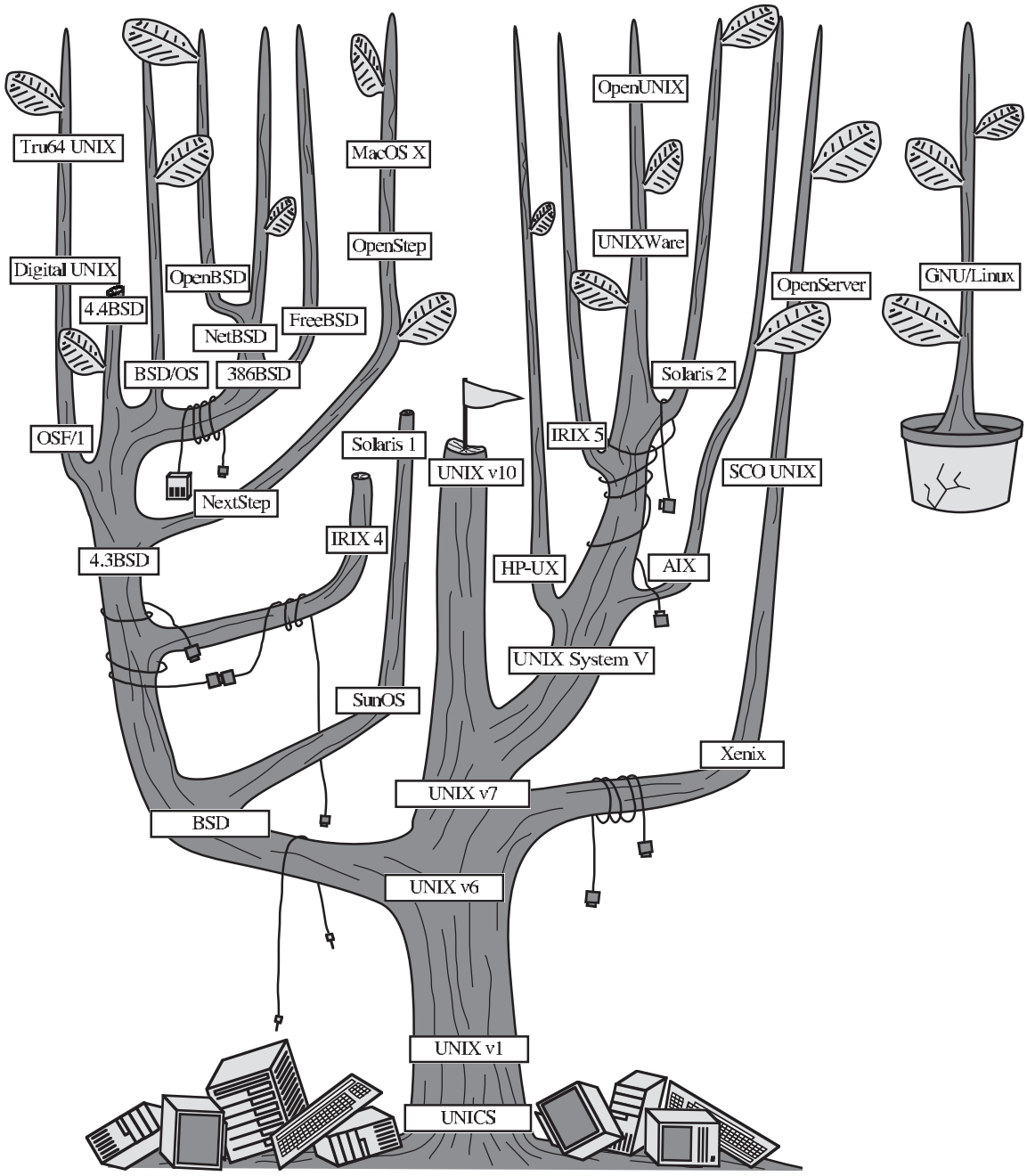
C'est un fait avéré : le public a horreur de la diversité. Puisqu'il y a trois BSD, il me faut obligatoirement indiquer lequel est le meilleur, ce qui permettra de négliger les deux autres.

La vie n'est pas si simple, et chacun des BSD a ses avantages. C'est ce qui justifie leurs existences même ! Certaines personnes, qui trouvent rassurant de ranger les choses dans des petites boîtes, vous expliqueront peut-être que suivant l'utilisation que l'on en fait, un seul BSD est valable. On entend par exemple souvent que du matériel exotique impose le choix de NetBSD, que pour l'utilisation sur PC il faut retenir FreeBSD, et que seul OpenBSD mettra en place un pare-feu convenable.

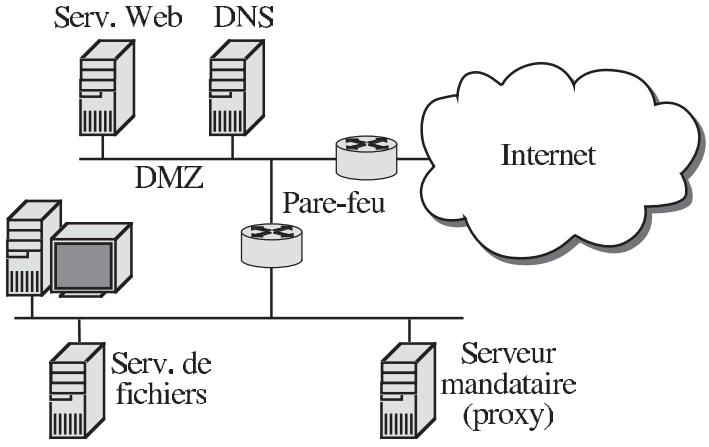
Ce genre de classement ne rime pas à grand-chose. OpenBSD reconnaît du matériel exotique, NetBSD et OpenBSD fonctionnent très bien sur PC, et ces trois systèmes font de très bons pare-feu. Mais tout cela ne vous dit pas lequel choisir...

L'idéal serait de tester les trois. À défaut, le conseil de l'auteur va vers NetBSD, mais n'oubliez pas que son jugement est biaisé. NetBSD vous procurera un environnement performant et fiable quels que soient la mission et le matériel que vous lui confierez. En particulier, vous pourrez l'employer tout à la fois sur une vieille machine sauvée du rebut pour vous faire la main, sur un PDA, ou sur un gros serveur.

Autre avantage, les exemples de ce cahier ont plus tendance à s'appuyer sur NetBSD que sur ses deux cousins, autre conséquence de la préférence de l'auteur...



2


Serv. Web
DNS
Internet
DMZ
Pare-feu
Serv. de
fichiers
Serveur
mandataire
(proxy)

Présentation de l'étude de cas

Plusieurs raisons peuvent vous amener à vous intéresser à l'apprentissage des bases d'Unix avec un système BSD. Examinons-en quelques-unes à l'aide d'une étude de cas.

SOMMAIRE

Unix domestique

Première situation, vous désirez découvrir Unix sur une configuration montée à la maison, probablement par curiosité personnelle. Vos motivations peuvent être assez diverses :

- Vous êtes frustré par l'informatique microsoftienne, coûteuse et peu fiable. On vous a présenté Linux comme le remède miracle, ce qui a piqué votre curiosité : quel dommage ce serait de manquer une telle panacée !
- Vous avez longtemps utilisé les services Internet (le Web, IRC, les news, la messagerie...) comme client, et avez voulu tester les choses côté serveur. Après avoir expérimenté quelques serveurs sur votre système d'exploitation habituel, vous avez la sensation de passer à côté de quelque chose. Vous voulez donc expérimenter tout cela sous Unix, par acquit de conscience.
- Vous savez que les compétences Unix se vendent bien sur le marché du travail, et avez donc décidé de l'inclure sur votre CV. Par souci d'honnêteté, vous allez d'abord l'apprendre.

Quelle que soit votre motivation, il est probable que vous ayez d'abord tenté votre chance avec GNU/Linux : il est gratuit et célèbre. Si tel n'est pas le cas, c'est presque dommage, car vous risquez de ne pas remarquer tous les avantages de BSD. Mais ce qui suit devrait quand même vous aider un peu à voir ce que vous avez raté.

Vous avez donc installé votre machine en double amorçage avec GNU/Linux, puis commencé à faire quelques allers-retours avec votre système habituel pour consulter la documentation en ligne qui vous permettrait de configurer l'accès à Internet. À chaque nouvel échec, un redémarrage s'imposait.

B.A.-BA **Double amorçage**

Le double amorçage est une configuration où la machine est capable de démarrer sur deux systèmes d'exploitation différents. En général, on met en place un menu de démarrage qui donne le choix du système à invoquer. En anglais, on parle de *dual-boot*.

Une fois votre installation de GNU/Linux à peu près fonctionnelle, vous avez constaté que c'était très joli, très coloré, avec une belle interface graphique, mais que votre maîtrise de ce système n'était pas bien supérieure à celle que vous aviez de Windows. Dès votre première utilisation, le système proposait déjà de nombreux services et vous n'aviez pas la moindre idée des relations liant ses divers éléments. Il est même possible qu'il se mettait à jour seul, modifiant son comportement et son organisation à mesure que vous tentiez de le comprendre.

Le système, livré avec de nombreux frontaux d'administration en mode graphique, vous a permis de configurer facilement la couleur du fond de l'écran, mais pas encouragé à essayer de comprendre comment les choses fonctionnaient : pourquoi se compliquer la vie à visiter des arborescences de fichiers de configuration alors que l'on l'impression de tout pouvoir faire à la souris ?

Un jour, quelque chose a mal tourné, et comme vous ne compreniez rien au fonctionnement de tout ce fatras, vous avez appliqué la méthode traditionnelle pour régler un problème sous Windows : tout réinstaller.

Le pire, c'est que vous avez peut-être trouvé tout cela normal. Cela ne l'est pas : sur les systèmes Unix, on peut comprendre comment fonctionne le système, et on peut corriger les problèmes sans tout réinstaller à chaque fois. Encore faut-il,

pour cela, que le système vous aide un peu. C'est là que les BSD peuvent vous apporter quelque chose.

Les BSD vous rendront les mêmes services que GNU/Linux, mais après configuration. Au départ, le système ne fait rien ; aucun service n'est démarré. Tout ce que vous ferez fonctionner sur la machine, vous l'aurez mis en place ; vous en comprendrez donc le fonctionnement et saurez y intervenir. En prime, les connaissances que vous aurez acquises s'appliqueront très bien aux autres Unix, en particulier à GNU/Linux.

PLUS LOIN **Mauvais, GNU/Linux ?**

Il faut tout de même dissiper certains malentendus : GNU/Linux n'est pas un mauvais système. On ne fera pas ici le procès de sa performance ou de sa stabilité, elles sont toutes les deux excellentes.
Ce que l'on peut reprocher aux distributions Linux les plus axées grand public (que vous essaierez si vous en choisissez une au hasard, puisque ce sont les plus répandues), c'est leur volonté de masquer la complexité du système. Pour l'usager qui dispose d'un gourou pour administrer sa machine ou pour l'administrateur confirmé, cela ne pose pas de problème, au contraire.
En revanche, pour l'administrateur débutant, ces choix sont lourds de conséquences : il est très difficile de prendre le système en main dans ces conditions.

Le plan de vol

Utiliser BSD n'est pas la solution à tous vos maux : il y a quelques autres écueils à éviter. Le premier, c'est sans doute le double amorçage, et ses allers-retours entre les systèmes pour pouvoir configurer la connexion à Internet. Vous gagnerez un temps précieux en ayant une deuxième machine, dédiée à BSD. En plus, vous pourrez essayer les services que vous monterez sur la machine BSD depuis votre machine habituelle, ce qui sera très gratifiant.

Seulement voilà : vous n'avez pas les moyens de vous offrir une deuxième machine. Pas de problème : pour apprendre, une vieillerie sauvée des ordures fera tout à fait l'affaire (l'auteur a longtemps fait fonctionner un serveur Web personnel sur une Sun IPC trouvée dans une benne de l'université). Les vieux Macintosh ou PC achetés d'occasion sont eux aussi de bonnes pistes : ils sont lents, mais sous réserve de gonfler un peu leurs configurations mémoire et disque, ils feront des petits serveurs tout à fait valables pour apprendre.

PERFORMANCES **Matériel préhistorique**

Si vous devez acheter une vieille machine d'occasion pour essayer un Unix, pensez tout de même à vérifier qu'elle est capable de l'accueillir. Pour les PC, il vous faudra au minimum un 386 ; pour les Macintosh, un 68030. Il est à noter que les vieux Macintosh sont dotés de disques SCSI, ce qui les rend nettement plus performants que les PC du même âge, bridés par des contrôleurs IDE de première génération.
Il faut aussi avoir à l'esprit que les vieilles machines ne sont pas forcément aussi fantastiques qu'elles peuvent en avoir l'air. S'essayer à Unix sur une machine multipliant les problèmes matériels n'est pas très agréable. De plus, si un Macintosh II ou un 486 sont assez puissants pour jouer le rôle d'un petit serveur, ils risquent d'être assez décevants si vous essayez d'y installer une interface graphique.

Deuxième problème, il vous faut un but. Si vous installez un système Unix pour le plaisir, vous aurez vite fait le tour de la question. Ayez des projets : installer un pare-feu pour votre connexion Internet (chapitres 9 et 10), monter un serveur Web personnel (chapitre 12), ou encore une installation bureautique en logiciel libre (les pistes sont données au chapitre 11).

Avant d'atteindre votre but, il vous faudra acquérir des fondamentaux, valables pour tous les systèmes Unix. Le chapitre 3 vous apprendra l'utilisation de la ligne de commande. Sa maîtrise vous permettra de n'être jamais pris au dépourvu, même quand l'interface graphique refusera de démarrer.

Les chapitres 5 à 8 seront l'occasion d'apprendre un peu d'administration Unix pour maîtriser le fonctionnement du système : comment se passe le démarrage, comment régler les problèmes qui peuvent le gêner, et comment gérer les utilisateurs et les services fonctionnant sur la machine. La configuration du réseau et de l'interface graphique sont abordées dans le chapitre 7.

L'administrateur de systèmes Microsoft

Autre profil, autres besoins. Vous gérez un parc Microsoft, avec des clients Microsoft, ce qui est hélas courant, mais disposez de plus – ce qui est plus dur – de serveurs tous équipés en Microsoft : le serveur Web IIS, la messagerie Exchange,

le serveur DNS de Microsoft, et ainsi de suite. Vous avez hérité cette situation de votre prédécesseur, ou avez monté le réseau en utilisant ce que vous aviez l'habitude d'utiliser pour faire vos rapports et vos présentations : du Microsoft.

Êtes-vous content de votre installation ? Peut-être pas. Tous les nouveaux virus et vers sont pour vous. Certes, les mises à jour vous protégeront, mais il y en a une quantité phénoménale et vous n'avez pas le temps de suivre. Pour couronner le tout, à chaque mise à jour, vous risquez de compromettre le bon fonctionnement de tout l'édifice, somme toute assez capricieux.

Si vous avez beaucoup de serveurs, vous savez combien le déploiement de quoi que ce soit sur toutes les machines est pénible. Vous êtes las de devoir rechercher dans la base de connaissances Microsoft quelle clef de base de registres créer pour modifier tel ou tel comportement. Les messages d'erreur cryptiques vous fatiguent. De plus, tout cela coûte très cher. Vous avez envie d'autre chose.

Vous avez peut-être discuté avec un collègue qui gère des serveurs Unix, et visiblement sa vie est bien moins insupportable. Mais il y a une grosse barrière à l'entrée : il faut connaître Unix. Vous n'êtes pas moins compétent qu'un autre, vous décidez d'apprendre, et de confier à Unix quelques services, s'il fait l'affaire.

Les systèmes BSD peuvent se révéler un bon choix ; ils font d'excellents serveurs. Ils sont gratuits, ce qui vous permettra enfin de faire des économies (si vous avez un budget à dépenser, vous vous rattraperez sur le matériel). Ils sont fiables, performants, bref, tout ce qu'il vous faut. Les mises à jour de sécurité des systèmes BSD tombent à un rythme supportable, et ces derniers bénéficient de l'avantage de la minorité : les vers et les virus sont écrits pour Windows et Linux, mais beaucoup plus rarement pour les BSD.

Même si l'activité professionnelle de votre entreprise vous empêche de choisir le système (par exemple, on a décidé pour vous le couple Solaris/Oracle), un système BSD sera probablement un bon terrain pour faire votre apprentissage.

Le plan de vol

Entre deux coups de fil d'utilisateurs qui n'arrivent pas à imprimer, vous allez donc monter un serveur expérimental. Vous projetez probablement de monter sous Unix un serveur de fichiers ou d'impression, un contrôleur de domaine NT, un serveur DNS, un serveur DHCP, un serveur mandataire, un pare-feu, ou un serveur Web. Rappelez-vous, tout cela ne vous coûtera pas un centime en licences.

Vous allez donc suivre le même parcours que pour Unix à la maison, mais en sautant probablement le chapitre 9, traitant de choses que vous devez déjà connaître : les réseaux Ethernet et TCP/IP. Le chapitre 13 retiendra toute votre attention, puisqu'il est consacré à l'étude des catastrophes informatiques, sujet que vous connaissez bien – votre métier c'est de les éviter.

ATTENTION **Mises à jour**

Le fait que les vers ne soient pas prévus pour votre système ne dispense pas de faire des mises à jour. Cela laisse juste un peu plus de temps avant d'être piraté.

CULTURE **SunOS et Solaris**

Le système d'exploitation développé par Sun s'appelait à l'origine SunOS et c'était un BSD. Sun l'a plus tard rebaptisé Solaris, puis l'a fait virer de bord pour le baser sur un UNIX System V. Ainsi, Solaris 1 est un BSD, et Solaris 2 est un System V.
Par un caprice de numérotation, Sun s'est mis à faire tomber le numéro majeur de version : ainsi, le successeur de Solaris 2.6 s'appelle Solaris 7 ; suivent Solaris 8 et 9.
Pour compliquer encore l'affaire, Sun a continué à donner un nom SunOS à ses systèmes baptisés Solaris. Pour chaque version de Solaris, il y a un deuxième numéro de version qui correspond à SunOS : Solaris 1 correspond à SunOS 4.1.1, Solaris 2.x à SunOS 5.x... La numérotation des versions du système de Sun est extrêmement compliquée.

VOCABULAIRE **Serveur mandataire, ou serveur proxy**

On trouve souvent le terme « serveur *proxy* », qui se traduit officiellement en français par « serveur mandataire ».

CULTURE **Serveurs Unix sous Windows**

Certains serveurs du monde Unix, tel le serveur DNS, sont capables de fonctionner sous Windows. Comme dans la version Unix, ces serveurs ne coûtent rien en licences. Si cette possibilité vous intéresse, il faut savoir que ces logiciels sont peu conformes aux standards d'interface utilisateur de Windows. Une culture Unix aide à les mettre en œuvre.

L'administrateur Unix

Dernier lecteur potentiel, l'administrateur de systèmes Unix constructeur dont le parc vieillit. Vous avez la charge de moyens informatiques d'une entreprise ou d'une université, comprenant de vieilles stations Sun, Silicon Graphics, des Alpha, des VAX, etc.

Vous êtes relativement content de ces machines, elles vous ont rendu des fiers services par le passé, mais l'heure de leur retraite approche. Non pas que le matériel fatigue (il est en pleine forme), mais parce que ces vieux systèmes multiplient les trous de sécurité. Vous n'avez pas envie de payer le prix fort pour une mise à jour du système d'exploitation qui, si elle existe, sera peut-être inadaptée à votre machine, un peu trop vieille.

Vous êtes face à un double problème : d'une part migrer les services vers des machines plus récentes, et d'autre part conserver en état de fonctionnement des applications développées exprès pour vos stations.

Pour vos nouvelles machines, un Unix libre vous assurera un système fonctionnel et pérenne. La migration vers les Unix libres est dans l'air du temps : c'est plus économique. Vous êtes habitué aux outils Unix, et BSD vous plaira sans doute plus que les distributions Linux, car ces dernières semblent jouer à celle qui introduira le plus de changements gratuits dans l'administration du système.

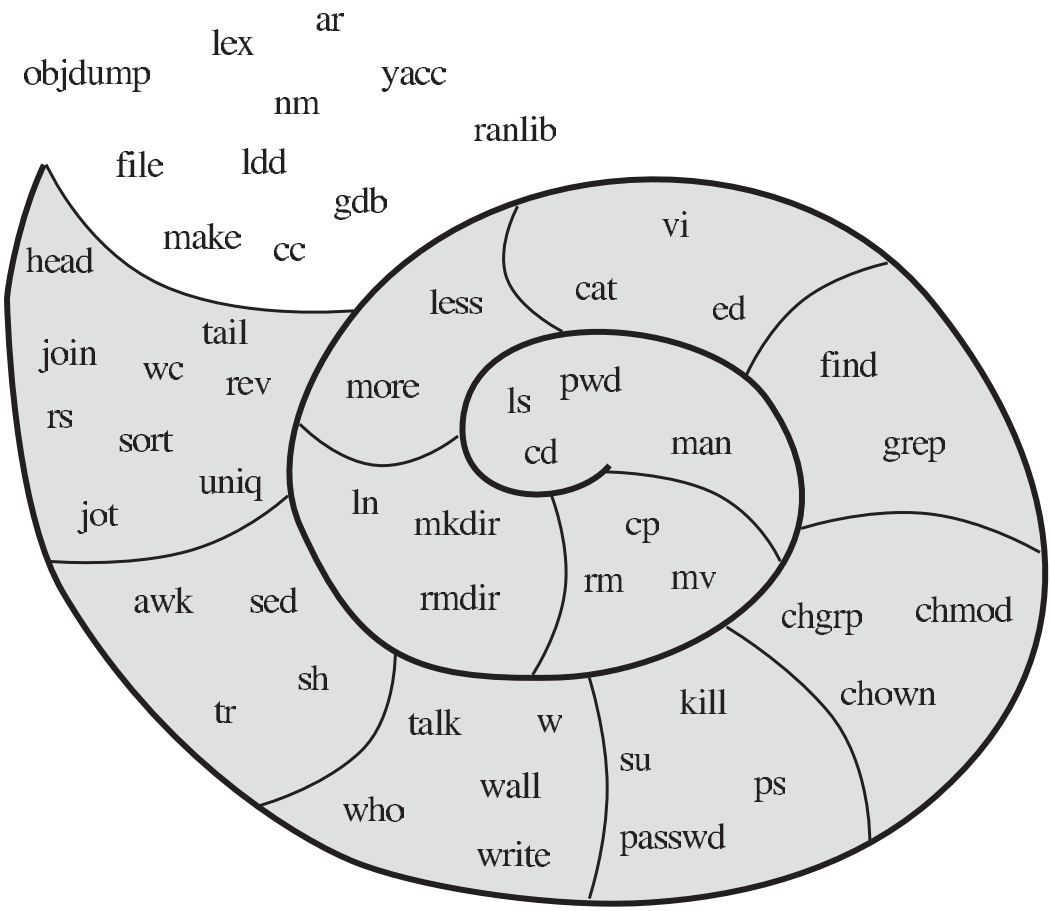
Si vous décidez de migrer vos vieilles stations sous BSD, il est probable que vous pourrez toujours faire fonctionner les programmes compilés pour votre Unix propriétaire, en employant les capacités de compatibilité binaire présentes dans les systèmes BSD. La compatibilité binaire permet de faire fonctionner les applications écrites pour le même processeur mais pour un Unix différent. La compatibilité n'est jamais exacte à 100 %, mais elle permet souvent de faire fonctionner les applications pour lesquelles il n'existe pas d'équivalent en logiciel libre – en fait c'est même sa raison d'être.

Le plan de vol

Vous allez donc monter une machine sous BSD pour utiliser les services anciennement assurés par vos stations de travail. Vous êtes déjà un usager confirmé d'Unix, et vous n'avez pas vraiment besoin de ce livre pour vous débrouiller. Vous voudrez peut-être examiner la séquence de démarrage (chapitre 5) ou la gestion des utilisateurs et des services (chapitres 6 et 8) ; vous y retrouverez des concepts familiers.

Les seuls chapitres qui vous apporteront quelque chose de nouveau sont sans doute celui traitant du système de paquetages et de la compatibilité binaire (chapitre 11) et celui portant sur les mises à jour du système (chapitre 13). C'est peu, mais il en faut bien pour tout le monde !

3


ar
lex
objdump
yacc
nm
ranlib
file
ldd
gdb
make
cc
head
vi
less
cat
ed
tail
join
wc
rev
more
pwd
find
ls
rs
sort
cd
man
grep
uniq
jot
ln
mkdir
cp
rm
mv
awk
sed
rmdir
chgrp
chmod
sh
tr
talk
w
kill
chown
su
wall
ps
who
passwd
write

Bien utiliser Unix pour mieux l'administrer

Nous allons introduire l'utilisation d'un système Unix avec les commandes de base, permettant de manipuler et d'éditer des fichiers, et de gérer les processus en cours. Les usagers déjà familiers de la ligne de commande Unix peuvent sans doute passer directement au chapitre suivant.

SOMMAIRE

MOTS-CLEFS

- Ligne de commande
- Le shell Unix
- Manipulation des fichiers
- Processus

Pourquoi maîtriser la ligne de commande ?

À l'ère des interfaces graphiques prétendument conviviales et ergonomiques, la ligne de commande peut sembler dépassée. Datant des années 1970, elle semble cryptique et complexe au profane. Qu'apporte-t-elle donc par rapport aux interfaces homme-machine se pilotant à la souris ? Cette question a plusieurs réponses.

Expressivité de l'interface texte

Tout d'abord, la ligne de commande est bien plus expressive que n'importe quelle interface graphique. Toute interface utilisateur, graphique ou textuelle, vise à communiquer avec la machine. Une interface graphique limite la complexité des instructions qu'elle permet de construire. Par exemple, à moins d'écrire un programme *ad hoc*, il y est impossible de demander à la machine de « tuer les processus de l'utilisateur toto occupant plus de 4 Mo ».

La ligne de commande bien maîtrisée permet des constructions bien plus complexes. Donnons un exemple percutant.

Pour un fanatique de l'interface graphique, l'équivalent de la ligne de commande suivante sera long à reproduire : j'envoie un courrier électronique à tous les utilisateurs occupant plus de 10 Mo sur leur compte (message préparé dans le fichier homefull.txt)

```
while read pwent ; do user=`echo $pwent|awk '{print $1}'`; home=`echo $pwent |
awk '{print $5}'`; test `du -ks $home|awk '{print $1}'` -ge 10240 &&
mail $user < homefull.txt ; done < /etc/passwd
```

À défaut d'être agréable à lire, cette ligne de commande s'exécute très rapidement : quelques secondes pour une centaine d'utilisateurs. Réaliser la même chose avec une interface graphique sur un système Windows prendrait plusieurs heures, et mettrait à rude épreuve les nerfs de l'administrateur système.

L'économie des ressources

Deuxième argument pour la ligne de commande : sa faible consommation de ressources. Gérer l'interface graphique consomme du temps de calcul et de la mémoire, ressources mieux employées à assurer les services pour lesquels la machine a été mise en place. Pour un serveur où le seul utilisateur de l'interface graphique est un administrateur qui opère en maintenance, le gâchis de ressources est patent.

L'utilisation de l'interface graphique est parfois difficile, voire impossible. Lors de l'administration à distance d'une machine située à l'autre bout du monde, la latence des communications peut dépasser la seconde.

Autre exemple de situation à latence : l'administration d'une machine à travers un lien saturé. La saturation limite la bande passante disponible et augmente la

CULTURE **Communications par satellite**

Lorsqu'elle transite par satellite, l'information se propage à la vitesse de la lumière (environ 300.000 km/s). Or, les satellites sont placés en orbite géostationnaire, à 36.000 km du sol. Pour se propager du sol au satellite puis du satellite au sol, l'information franchit donc 72.000 km ; et cette seule étape dure au minimum un quart de seconde. Les liens satellites peuvent être multiples, d'où des latences encore plus importantes.

latence. Il est fréquent d'observer des délais de plusieurs secondes entre l'action effectuée et la réaction de la machine.

Dans toutes ces situations, une interface graphique se révèle inutilisable. Tout ceux qui ont déjà travaillé sur une machine surchargée et dont l'interface graphique est peu réactive comprendront aisément. Rien n'est plus énervant que d'attendre plusieurs secondes à chaque clic de souris.

L'interface texte souffre elle aussi des augmentations de latence, mais il est toujours possible de travailler sans piquer de crise de nerf. On peut par exemple taper ses commandes en aveugle et attendre patiemment le résultat, taper ses commandes sur la machine locale et les copier/coller dans la session distante, ou préparer des scripts et les envoyer d'un bloc pour exécution sur la machine distante.

CULTURE **Les seaux percés**

Les machines communiquent entre elles par l'envoi de petits « paquets » d'information, appelés datagrammes. L'Internet est composé de réseaux reliés par des routeurs. On modélise parfois les routeurs comme des seaux percés : les paquets entrant dans le routeur sont l'eau versée, et les paquets sortants, l'eau qui s'écoule.
Si un routeur reçoit plus de paquets qu'il ne peut en envoyer sur un lien donné, il mets les paquets en file d'attente : le seau se remplit. Les paquets tardent plus à traverser le routeur, et la latence augmente.
Quand la file d'attente est pleine, le routeur doit abandonner des paquets (le seau déborde). Les paquets perdus devront être retransmis, ce qui augmente encore la latence.

Quand l'interface graphique ne répond plus

Dernier argument en faveur de la ligne de commande : c'est parfois la seule possibilité. Les systèmes BSD, lors de leur installation, ne configurent pas d'interface graphique : il faut pour cela jouer de l'interface texte. De plus, même après configuration d'une interface graphique, le système de base ne contient aucun programme permettant d'administrer la machine via des frontaux graphiques. Il faudra donc installer de nombreux paquetages pour pouvoir se passer éventuellement de l'interface texte.

L'utilisateur ayant découvert l'informatique récemment pourrait s'interroger sur l'intérêt d'utiliser un système BSD, alors que tant de distributions Linux démarrent immédiatement l'interface graphique.

Mais cet usager moderne oublie une chose : tôt ou tard, quand on administre une machine, on finit par casser quelque chose. Ou bien quelque chose tombe en panne : les disques dur meurent, eux aussi. Les coupures de courant peuvent encore amener leur lot de problèmes. Et un jour, la machine, très malade, ne sera plus capable de démarrer l'interface graphique, ni la plupart des services pour lesquels on l'a configurée.

B.A.-BA **Les distributions Linux**

Les distributions Linux abondent, et se divisent en deux catégories : d'une part, celles, faciles à installer, qui finiront par vous faire tourner en bourrique à tout automatiser : Red Hat, SuSE, Mandrake. D'autre part celles, plus proches de la philosophie BSD, ne prenant pas d'initiatives et vous laissant gouverner le système : Slackware, Debian, Gentoo (listes non exhaustives).

À ce stade, deux voies sont possibles : l'administrateur maîtrise la ligne de commande d'Unix, répare son système, et la machine est rapidement remise sur pied. Deuxième possibilité, l'administrateur connaît trop mal l'interface texte, et la seule issue est alors la réinstallation complète de la machine, avec éventuellement quelques pertes de données à la clef.

Mon point de vue est que finalement, les BSD rendent un fier service à l'administrateur en l'obligeant à utiliser l'interface texte. Il y gagne sur plusieurs tableaux :

- Sa capacité à traiter efficacement des problèmes qui peuvent prendre plusieurs jours si on n'essaie de ne les traiter qu'avec l'interface graphique.
- La possibilité de prendre la main sur des machines distantes dont la connectivité est mauvaise, à cause de problèmes structurels (machine à l'autre bout du monde), ou conjoncturels (lien réseau saturé).

B.A.-BA **Modes mono-utilisateur et multi-utilisateurs**

Le mode de fonctionnement normal d'une machine Unix est le mode multi-utilisateurs, où la machine est susceptible d'accueillir en parallèle les processus de dizaines d'utilisateurs – d'où son nom.
Quand le système détecte un problème majeur au démarrage, il passe en mode mono-utilisateur (*single user*). Dans ce mode, la machine n'offre qu'une interface texte sur la console.

- La capacité à remettre en état une machine qui n'est plus capable de fonctionner en mode multi-utilisateurs.

Après avoir exposé les raisons pour lesquelles il est utile de maîtriser la ligne de commande, nous pouvons passer à son utilisation.

Où se faire la main ?

On peut se faire les dents sur le système d'un autre ou sur un système récemment installé. Si vous optez pour la deuxième solution, faites un détour par le chapitre 4, qui traite de l'installation des systèmes BSD.

Au terme de l'installation, le système propose une interface de ligne de commande, avec les droits de l'administrateur du système, l'utilisateur appelé root. Cet usager étant omnipotent, une fausse manœuvre faite en son nom peut détruire tout le système.

Apprendre la ligne de commande en tant que root vous exposera à tout réinstaller à chaque erreur grave. Il vaut donc mieux créer un compte utilisateur aux pouvoirs limités pour faire son apprentissage des commandes Unix (la création des comptes est abordée au chapitre 6). Mais pour créer ce compte, il faudra faire appel à la ligne de commande !

VOCABULAIRE **Le super-utilisateur**

Sous Unix, l'utilisateur root a les pleins pouvoirs. En anglais, on parle de *superuser* (super-utilisateur).

B.A.-BA **SSH et Telnet**

SSH signifie *Secure SHell*. C'est un protocole permettant d'avoir accès à distance à la ligne de commande d'une machine Unix, en chiffrant les communications (ce qui évite de se faire capturer son mot de passe ou d'être espionné par des indélicats). Telnet joue le même rôle que SSH, mais sans chiffrement – il est donc moins sûr.
Pour se connecter ainsi sur une machine distante, il faut disposer d'un client (Telnet ou SSH) sur la machine locale, et du serveur correspondant activé sur la machine distante. Par exemple, sous MacOS X, on active le serveur SSH dans le tableau de bord **partage** en cochant la case **Connexion à distance** dans l'onglet **Service**. Ensuite, il ne reste plus qu'à invoquer le client, taper l'adresse du serveur, et saisir son nom d'utilisateur et mot de passe.

La solution la plus simple est donc de vous faire la main chez quelqu'un d'autre. Dans le cadre professionnel, il vous est peut-être possible d'obtenir un compte sans privilège particulier sur une machine Unix. Et les universités proposent souvent des comptes Unix à leurs étudiants.

Autres pistes possibles : sur tout système MacOS X, on dispose d'un Unix ; il suffit pour cela d'invoquer le terminal ou de se connecter à distance via SSH – à condition d'y avoir un compte, bien entendu.

Dernière possibilité : certains fournisseurs de services Internet proposent des accès gratuits au shell de leur machine Unix : une requête dans votre moteur de recherche favori en indiquant « *free Unix shell* » devrait vous en donner quelques-uns. Le service fourni est en général assez limité : peu d'espace disque, pas de possibilité d'utiliser le réseau ou de compiler un programme. Ces fournisseurs de services vendent en général un service complémentaire sur la même machine pour bénéficier des fonctions inexistantes dans la prestation gratuite.

Le shell, quelques commandes sur les fichiers

Lorsque l'usager interagit avec la machine via la ligne de commande, c'est le plus souvent par le biais d'un « shell ». Le shell est un programme qui attend des commandes, les exécute, puis attend à nouveau les commandes suivantes.

Pour signaler à l'usager qu'il attend une commande, le shell affiche une invite de commande, dont la forme par défaut dépend du type de shell utilisé. Elle se

termine en général par un $ (dollar), remplacé par un # (dièse) lorsque l'usager travaille en tant qu'administrateur. Cela permet de se rappeler les pouvoirs illimités dont on dispose, et les responsabilités qui en découlent. Ces comportements correspondent aux réglages par défaut, mais il peuvent être configurés différemment. Chaque shell a aussi quelques comportements propres.

Outre le shell, un système Unix comprend un système de fichiers, organisé en arborescence en partant d'un répertoire appelé la racine, et noté / (*slash*, ou barre de division). À tout moment, le shell dispose d'un répertoire courant : l'endroit de l'arborescence sur lequel la prochaine commande agira.

Voyons quelques commandes pour naviguer dans l'arborescence. Pour les tester, il suffit de les taper à l'invite de commande, puis de valider.

- **pwd** affiche le répertoire courant, ainsi que le chemin qui y mène depuis la racine.
- **ls** affiche le contenu du répertoire courant.
- **cd rep** permet d'entrer dans le répertoire **rep**, qui devient alors le répertoire courant.
- **cd ..** est une forme particulière de la commande **cd**, qui permet de remonter dans le répertoire père du répertoire courant (celui qui contient le répertoire courant)

Il est possible de fournir à **cd** des chemins au lieu d'un simple répertoire. Par exemple, pour se rendre dans le répertoire share, situé dans répertoire usr, lui-même à la racine, on peut taper :

```
$ cd /usr/share
$ pwd
/usr/share
```

CULTURE **Les systèmes de fichiers**

Le système de fichiers Unix diffère un peu de celui du Mac (avant MacOS X) ou de Windows. En particulier, il compte une unique racine, à laquelle tout se rattache. On y trouve le contenu d'une partition (la partition racine) ; les autres partitions ou disques sont visibles dans des répertoires appelés « points de montage », tels que /home ou /cdrom.
MacOS avant sa version X et Windows ont un système de fichiers distinct pour chaque partition, tous ces volumes apparaissant soit avec leurs noms (sous MacOS), soit avec des noms de lettres comme C: ou D: (sous Windows).
Autre différence : sous Unix, les répertoires sont séparés par des caractères / (*slash*) alors que Windows utilise des \ (*backslash*) et que MacOS (avant X) utilise : (deux points).

ATTENTION **Invite de commande**

Dans l'exemple ci-contre, le $ est l'invite de commande : ne le tapez pas ! Dans tous les exemples, les commandes à taper apparaissent **comme ceci** et ce qu'affiche l'ordinateur est représenté ainsi.

HISTOIRE **Les types de shells**

Le plus ancien shell disponible s'appelle *Bourne Shell*, invoqué par la commande /bin/sh. Il est apprécié pour réaliser des scripts, car c'est le seul qui soit standardisé (norme ISO 9945-2, dite POSIX.2) Grâce à cette standardisation, son comportement ne varie pas trop d'un Unix à l'autre, ce qui est capital pour pouvoir réutiliser les scripts.
En revanche, il n'est pas très prisé pour l'utilisation interactive, car il était à l'origine peu confortable (pas de complètement, pas de rappel des commandes tapées précédemment...). Deux shells visant à le remplacer pour des sessions interactives sont donc rapidement apparus : le *Korn Shell* (**ksh**) et le *C Shell* (**csh**). Le *Korn Shell* s'efforce de reprendre la syntaxe du *Bourne Shell*, et le *C Shell* utilise une syntaxe proche du langage C.
Récemment, des évolutions de ces shells s'en sont démarquées, essentiellement pour des question de licences des logiciels : Le Bourne-Again Shell (**bash**) est un *Korn Shell* amélioré fourni par défaut sous GNU/Linux, ce qui lui a donné une forte popularité. **tcsh** est une évolution du *C Shell*.
Enfin, certains shells, comme **zsh**, tentent une synthèse entre les syntaxes du *Bourne Shell* et du *C Shell*, poussant plus loin les facilités d'utilisation interactive.
Le choix du shell interactif est avant tout une affaire de goût personnel. Pour faire des scripts, le *Bourne Shell* est difficilement contournable, puisqu'il est le seul qui soit standardisé. Le *Korn Shell* a l'avantage de permettre une utilisation interactive relativement conviviale (une fois configuré correctement), tout en gardant la syntaxe du *Bourne Shell*. Certains lui préféreront des shells plus récents et confortables, tels que **bash**.
La meilleure solution pour choisir son shell est encore d'en essayer plusieurs. Aller demander dans un forum de discussion lequel est le meilleur ne produira qu'une guerre passionnée et stérile entre les partisans de chaque shell.
La commande **echo $SHELL** permet de découvrir le shell utilisé.

Ce chemin est absolu, car il commence par un / (*slash*), ce qui signifie qu'il part de la racine du système de fichiers. On peut aussi indiquer des chemins relatifs, par exemple pour aller dans le répertoire sys du répertoire include, lui-même dans le répertoire père du répertoire courant :

```
$ pwd
/usr/share
$ cd ../include/sys
$ pwd
/usr/include/sys
```

La commande **ls** liste le contenu d'un répertoire

- **ls** s'applique au répertoire courant.
- **ls var** concerne le répertoire var.

De même qu'avec **cd**, on peut passer à **ls** des chemins plus complexes, dont voici quelques exemples :

```
$ ls /usr/share
calendar        groff_font      man             openssl         syscons
dict            info            me              pcvt            tabset
doc             isdn            misc            perl            tmac
examples        libg++          mk              sendmail        vi
games           locale          nls             skel            zoneinfo
$ cd /usr/share
$ pwd
/usr/share
$ ls
calendar        groff_font      man             openssl         syscons
dict            info            me              pcvt            tabset
doc             isdn            misc            perl            tmac
examples        libg++          mk              sendmail        vi
games           locale          nls             skel            zoneinfo
$ ls ../../var
account         cron            games           mail            rwho
at              db              heimdal         msgs            spool
```

CULTURE **Organisation du système de fichiers Unix**

Lors de l'exploration du système de fichiers, vous vous interrogerez peut-être sur la fonction de chacun des répertoires. Voici celle de quelques répertoires courants :

/etc	Fichiers de configuration.
/home	Répertoires personnels des utilisateurs.
/dev	Pseudo-fichiers donnant accès au matériel.
/var	Fichiers de taille variable, comme par exemple les journaux, ou les files d'attente de messagerie et d'impression.
/bin	Commandes utilisateur indispensables à la maintenance de la machine.
/sbin	Commandes d'administration indispensables à la maintenance de la machine.
/usr	Tout ce qui n'est pas indispensable à la maintenance de la machine, mais sert lors de son fonctionnement normal.
/usr/bin	Les autres commandes utilisateur.
/usr/sbin	Les autres commandes d'administration.
/usr/local	Arborescence réservée à l'administrateur. Le système n'y installe jamais rien.
/tmp	Fichiers temporaires, effacés à chaque redémarrage.
/var/tmp	Fichiers temporaires, préservés entre les redémarrages.

La documentation comprend une description exhaustive du système de fichiers, qu'on peut consulter en tapant **man hier** à l'invite de commande. Nous détaillerons sous peu l'utilisation de cette aide en ligne ; sachez d'ores et déjà que la touche **Espace** permet d'avancer d'un écran et que la touche **q** quitte l'aide en ligne.

```
backups         dnews           log             preserve        tmp
crash           empty           lost+found      run             yp
$ cd /
$ pwd
/
$ ls var
account         cron            games           mail            rwho
at              db              heimdal         msgs            spool
backups         dnews           log             preserve        tmp
crash           empty           lost+found      run             yp
```

La commande **ls** admet des options. Dans les commandes Unix, les options prennent la forme d'une ou plusieurs lettres, précédées par un **–**. Par exemple, l'option **–a** (comme *all*) de **ls** permet de lister tous les fichiers, y compris les fichiers « cachés » :

```
$ ls /tmp
truc.txt
$ ls -a /tmp
.               ..              .X11-unix       truc.txt
```

De même, l'option **–l** (comme *long*) permet de voir plus d'informations sur chaque fichier. Voici des exemples de listes fournies par **ls –l** et **ls –a –l** :

```
$ ls -l /
total 4816
drwxr-xr-x   2 root  wheel      512 Jul 13 22:24 altroot
drwxr-xr-x   2 root  wheel     1024 Aug 27 03:22 bin
drwxr-xr-x   4 root  wheel    10752 Sep 14 21:50 dev
drwxr-xr-x  19 root  wheel     2048 Oct 22 21:47 etc
drwxr-xr-x   4 root  wheel      512 Aug 27 06:51 home
drwxr-xr-x   2 root  wheel      512 Jul 11 12:51 mnt
-rw-r--r--   1 root  wheel  2410055 Aug 31 09:42 netbsd
drwxr-xr-x   4 root  wheel      512 Aug 28 18:39 root
drwxr-xr-x   2 root  wheel     2048 Jul 12 23:27 sbin
drwxr-xr-x   2 root  wheel      512 Jul 11 12:51 stand
lrwxr-xr-x   1 root  wheel       11 Jul 13 11:29 sys -> usr/src/sys
drwxrwxrwt   3 root  wheel      512 Oct 24 12:34 tmp
drwxr-xr-x  15 root  wheel      512 Aug 28 11:34 usr
drwxr-xr-x  23 root  wheel      512 Jul 16 15:11 var
$ ls -l -a /tmp
total 68
drwxrwxrwt   3 root  wheel    512 Mar 15 04:34 .
drwxr-xr-x  20 root  wheel    512 Mar  6 10:37 ..
drwxrwxrwt   2 root  wheel    512 Mar  6 10:51 .X11-unix
-rw-r--r--   1 root  wheel  51200 Feb  8 16:27 truc.txt
```

On observe dans ces exemples la liste détaillée des fichiers et répertoires présents à la racine. Décrivons-la :

Dans la première colonne, on trouve une première lettre indiquant le type de l'objet considéré. Voici les types les plus courants :

- `–` fichier normal
- `d` répertoire
- `l` lien symbolique (équivalent des alias sur Mac et des raccourcis sous Windows)

Les autre caractères indiquent les droits sur le fichier (voir la note ci-contre).

La deuxième colonne indique le nombre de références existant sur l'objet. En l'absence de liens durs (voir l'aparté), ce nombre est égal à 1 pour un fichier, et au nombre d'éléments contenus pour un répertoire.

ASTUCE **Les fichiers cachés**

Les fichiers et répertoires dont le nom commence par un . (point) sont invisibles lorsque l'on utilise **ls** (sauf pour l'utilisateur root, qui voit toujours tout). **ls –a** permet de voir tous les fichiers et répertoires. Cela permet d'alléger la présentation en manipulation courante, ces fichiers étant usuellement des fichiers de configuration qu'on ne souhaite pas voir apparaître en permanence.
Chaque répertoire contient au moins deux éléments dont le nom commence par un . (point) : le répertoire . (point), qui désigne le répertoire courant, et le répertoire .. (point point), qui désigne le répertoire père du répertoire courant. Le répertoire racine est son propre père.
Le répertoire .. (point point) est d'utilisation courante ; nous en avons déjà vu un exemple. On utilise le répertoire . (point) dans certains cas plus rares ; nous en verrons un plus loin.

EN PRATIQUE **L'affichage des droits**

La figure 3.1 décrit l'attribution des droits affichés par **ls**. Un `r` indique un droit en lecture (*read*), un `w` un droit en écriture (*write*), et un `x` un droit en exécution. Un – signale une absence de droit.

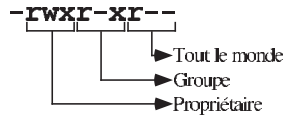


Figure 3–1 L'affichage des droits

PLUS LOIN **Autres types d'objets**

Il existe d'autres types d'objets, qui ne sont pas des vrais fichiers, et qui ne vous intéresseront pas immédiatement. Pour les nommer rapidement :

- `c` périphériques en mode caractères
- `b` périphériques en mode blocs
- `p` tubes nommés (*named pipes*, ou FIFO)
- `s` point d'entrée de protocoles de réseau locaux à la machine (*sockets* UNIX)

PLUS LOIN **Taille des répertoires**

Un répertoire est en quelque sorte un fichier contenant la liste des éléments qu'il renferme. C'est la taille (en octets) de ces informations qui est indiquée dans la cinquième colonne du résultat de **ls -l**. Pour obtenir la véritable taille (en kilo-octets) d'un répertoire, utilisez la commande **du -ks rep**, où **rep** est le nom du répertoire concerné.

Les troisième et quatrième colonnes indiquent respectivement l'utilisateur et le groupe propriétaires du fichier. Nous aborderons les groupes d'utilisateurs dans le détail au chapitre 6.

Le cinquième champ contient la taille de l'objet en octets. Pour les fichiers, il s'agit de la taille de leur contenu ; pour les autres objets, c'est plus complexe.

Viennent ensuite la date et l'heure de dernière modification du fichier, et enfin le nom du fichier lui-même. Le nom des liens symboliques est suivi du chemin de l'objet cible.

À l'aide !

La commande **ls** compte encore beaucoup d'options. Où trouver leur liste ? Comment interpréter la colonne donnant les droits d'accès aux fichiers ?

La réponse à ce type d'interrogations légitimes est donnée par la commande **man**, dont le rôle est d'afficher la documentation (page **man**) d'une commande donnée. Exemple pour la commande **ls** :

```
$ man ls
LS(1)                      NetBSD Reference Manual                      LS(1)

NAME
     ls - list directory contents


(...)
```

Les pages **man** sont nombreuses, exhaustives et souvent bien à jour. C'est sans doute la meilleure source de documentation que l'on puisse trouver sur un système Unix. Elles sont également difficiles à lire, austères, et en anglais.

PLUS LOIN **Liens symboliques, liens durs, et nombre de références**

Il y a deux types de liens : les liens symboliques et les liens durs. Un **ls -l** ne montre les destinations que pour les liens symboliques, petits fichiers contenant l'emplacement du fichier destination – qui peut éventuellement ne pas exister. On crée les liens symboliques avec la commande **ln -s** :

```
$ ln -s /etc/profile test
$ ls -l
total 0
lrwxr--r-- 1 manu  wheel  12 Jul 21 16:18 test -> /etc/profile
```

Les liens durs, qui se créent en utilisant la commande **ln** sans l'option **-s**, consistent à faire pointer plusieurs noms de fichiers vers le même *inode*. Les *inodes* sont des structures internes utilisées par le système pour gérer les objets présents dans le système de fichiers : chaque fichier ou répertoire est associé à un et un seul *inode*.
ls -l ne montre pas la destination d'un lien dur, car il n'y a pas de nom de fichier destination. Il y a simplement plusieurs noms de fichier donnant accès au même contenu. Par contre, les liens durs sont décelables en observant la deuxième colonne de l'affichage de **ls -l**, qui indique le nombre de références vers un objet. Pour un fichier, il est égal au nombre de noms pointant sur le même *inode*. S'il est supérieur à 1, il y a un lien dur.

```
$ ls -l
total 2
-rw-r--r--  1 manu  wheel  6 Jul 21 16:26 test
$ ln test nom2
$ ls -l
total 4
-rw-r--r--  2 manu  wheel  6 Jul 21 16:26 nom2
-rw-r--r--  2 manu  wheel  6 Jul 21 16:26 test
$ ln test nom3
$ ls -l
total 6
-rw-r--r--  3 manu  wheel  6 Jul 21 16:26 nom2
-rw-r--r--  3 manu  wheel  6 Jul 21 16:26 nom3
-rw-r--r--  3 manu  wheel  6 Jul 21 16:26 test
```

Il est important de signaler qu'on ne peut faire des liens durs qu'entre des objets présents sur la même partition.

Détaillons :

- Difficiles à lire : c'est vrai, du moins au début. Utiliser efficacement cette documentation suppose un certain apprentissage. Nous donnerons plus loin quelques clefs pour mieux digérer les pages **man**.
- Austères : c'est encore vrai. Tout comme d'ailleurs le reste de ces systèmes BSD. Il faut prendre les choses du bon côté : lorsque vous prenez place au volant d'une formule 1, vous ne vous attendez pas à trouver un allume-cigare ou un autoradio, mais la formule 1 a quand même certains plus par rapport à une Lada.
- En anglais : qu'on le déplore ou qu'on s'en réjouisse l'anglais est la langue internationale pour la plupart des disciplines techniques, et en particulier pour l'informatique. La documentation est donc d'abord écrite en anglais puis parfois traduite. Qui refuse d'apprendre l'anglais technique, risque de se priver d'une partie de la documentation, ou se de limiter à des versions dépassées.

  Sur les systèmes BSD, les pages **man** ne sont pas traduites, plus par manque de moyens humains que par choix politique. Si lire l'anglais technique représente un obstacle insurmontable, les BSD ne sont probablement pas faits pour vous.

Revenons sur les pages **man** proprement dites. Toutes les pages documentant des commandes ont la même structure, dont il faut s'imprégner pour être efficace dans sa lecture.

On trouve dans chaque page le nom de la commande, suivi entre parenthèses de la section du manuel dans laquelle elle se trouve. Il y a 8 sections sur un Unix traditionnel, et parfois quelques sections supplémentaires, spécifiques au système. Par exemple, les BSD ont une section 9 qui documente le noyau, certains systèmes Unix commerciaux ont une section 3F pour les interfaces de programmation en Fortran, etc.

- section 1 : commandes pour l'utilisateur
- section 2 : appels système, interfaces pour les programmes en C

PLUS LOIN **La visionneuse**

Les pages **man** sont affichées à l'aide d'une commande appelée visionneuse (*pager* en anglais). C'est par défaut **less**, ou **more** sur les Unix anciens. Son rôle est de présenter la documentation écran par écran : sur un terminal de 25 lignes, elle affiche 24 lignes, puis attend une pression sur la touche **Espace**, qui affichera les 24 lignes suivantes, et ainsi de suite.
Avec **less**, on peut remonter ou redescendre avec les touches fléchées. Avec **more**, la touche **u** (*up*) permet de monter, la touche **entrée** de descendre. À tout moment, on peut quitter **less** ou **more** en enfonçant la touche **q** (quitter).

CULTURE **Pages man et pages info**

Les pages **man** ne sont hélas pas universelles. La *Free Software Foundation*, à l'origine de beaucoup de programmes pour Linux et BSD (et en particulier de tous les outils de compilation) a décidé que sa documentation serait produite en pages **info** plutôt qu'en pages **man**. Les pages **info** s'utilisent comme les pages **man**, mais avec la commande **info**. La navigation y est assez différente.

```
$ info gcc
File: gcc.info,  Node: Top,  Next: G++ and GCC,  Up: (DIR)

Introduction
************

   This manual documents how to run, install and port the GNU compiler,

(...)
```

Certains volontaires écrivent des pages **man** pour les programmes de la FSF, mais elles ne sont pas toujours très à jour. Ce sont les pages **info** qui font autorité pour ces programmes.

- section 3 : fonctions des bibliothèques du langage C
- section 4 : points d'entrée du noyau, pilotes
- section 5 : fichiers de configuration
- section 6 : jeux
- section 7 : standards
- section 8 : commandes d'administration
- section 9 : fonctions présentes dans le noyau

Certaines pages **man** existent dans plusieurs sections : on trouve par exemple une commande **passwd** (pour changer son mot de passe) et un fichier de configuration passwd (base de données des utilisateurs existant sur le système). Il y a donc deux pages **man**, en sections 1 et 5. Si l'on tape **man passwd**, la commande **man** affichera la première page trouvée, en section 1. Pour obtenir la page en section 5, il faut la demander explicitement : **man 5 passwd**.

EN PRATIQUE **Plusieurs pages au même nom**

Comment savoir s'il y a des pages de même nom dans d'autres sections ? On peut tout simplement demander à la commande **man** d'aller chercher la page dans toutes les sections une par une. On peut aussi jeter un œil à la rubrique SEE ALSO de la première page trouvée : elle précise la plupart du temps s'il y a d'autres pages **man** de même nom dans d'autres sections du manuel.

SUR LES AUTRES UNIX **Choisir la section**

Sur certains systèmes Unix, on sélectionne la section du manuel avec l'option **-s** : **man -s 5 passwd**

La structure des pages man

Pour une section donnée, les pages **man** respectent toujours la même structure. Si la lecture est un peu pénible les premières fois, on gagne en efficacité une fois que l'on sait où chercher l'information. Voici la structure des pages pour les sections 1 et 8 (les commandes) :

- NAME : nom de la commande
- SYNOPSYS : mode d'emploi synthétique de la commande. La notation est toujours la même : les éléments optionnels entre crochets. les choix possibles séparés par un caractère | (barre verticale), etc.
- DESCRIPTION : description complète de la commande, comprenant notamment le détail de chaque option disponible. La liste des options est une portion fréquemment consultée.
- SEE ALSO : les autres pages **man** peu ou prou liées à celle-ci. Cette rubrique est importante car elle permet de trouver une commande dont on ne connaît pas encore le nom.

On trouve aussi quelques sections optionnelles : ENVIRONMENT, STANDARD, HISTORY, COMPATIBILITY, pas présentes dans toutes les pages.

Lorsque l'on consulte une page **man**, on peut chercher un mot en tapant **/** (*slash*) suivi d'une portion significative de ce mot, et validation. En appuyant ensuite sur

EN PRATIQUE **Le synopsis**

Quelques exemples de synopsis valent mieux qu'un long discours. Le synopsis (syntaxe) indique comment utiliser une commande :

- cmd [-ab] indique que l'on peut faire **cmd**, **cmd -a**, **cmd -b** ou encore **cmd -a -b**.
- cmd -b [-a] donne les possibilités **cmd -b -a** ou **cmd -b**.
- cmd [-a|-b] autorise à utiliser **cmd**, **cmd -a** ou **cmd -b**.

L'ordre des options est en général sans importance, et on peut compacter ces dernières comme suit : **cmd -ab** représente **cmd -a -b**

n, on passe à l'occurrence suivante du mot ; **N**, permet de passer à l'occurrence précédente. La recherche de mots-clef dans une page **man** est un véritable réflexe à prendre pour lire plus efficacement. Imaginons que vous recherchiez l'option de **ls** qui permet d'afficher une barre de division (*slash*) à la fin de chaque répertoire pour les distinguer des autres éléments. Sur un système BSD, rechercher le mot « *Directory* » dans la page **man** de **ls** vous livrera la réponse en quelques secondes : **ls -p**.

ASTUCE **Chercher sans la casse**

Les mots en début de phrase commençant par des majuscules, il faut penser à chercher avec et sans majuscule initiale, ou omettre la première lettre : taper **irectory** au lieu de **directory**.

Trouver la bonne page man

Dernier point sur la documentation : comment trouver une commande sans connaître son nom ? Les pages **man** n'aident pas tellement dans un tel cas de figure. On peut essayer de rebondir de page en page avec la rubrique SEE ALSO, mais ça n'est pas toujours possible. Par exemple, comment retrouver les commandes qui permettent d'utiliser un modem ?

C'est l'objet de la commande **apropos** :

```
$ apropos modem
chat (8) - Automated conversational script with a modem
mhzc (4) - Megahertz Ethernet/modem card driver
umodem (4) - USB modem support
```

SUR LES AUTRES UNIX **apropos ou pas ?**

Sur certains Unix, **apropos** n'existe pas ; il faut remplacer cette commande par **man -k**.

CULTURE **Désignation des pages man**

Pour préciser que l'on parle d'une page **man**, on indique souvent le nom de la commande suivie de la section de manuel dans laquelle on peut la trouver, mis entre parenthèses. Exemples : ls(1), pwd_mkdb(8), wtf(6).

Cela fournit une seule commande (en section 8). La lecture de chat(8) et des autres pages listées dans la section SEE ALSO devrait permettre de découvrir peu à peu comment configurer une connexion PPP via un modem.

Plus de commandes portant sur les fichiers

Nous avons vu comment se déplacer dans l'arborescence et lister des fichiers. Voici d'autres commandes fondamentales pour la manipulation des fichiers et des répertoires, accompagnées de quelques exemples d'utilisation :

Copier :

```
$ cp /etc/services /tmp/services
$ cd /tmp
$ ls
services
$ cp services services.bak
$ ls
services        services.bak
$ cp /etc/services /etc/protocols /etc/hosts /tmp
$ ls
hosts           protocols       services        services.bak
```

Déplacer :

```
$ mv /tmp/services /var/tmp/services
$ cd /var/tmp
$ ls
services
$ mv /tmp/protocols /tmp/hosts ./
$ ls
hosts      protocols  services
```

```
$ mv protocols services hosts /tmp
$ ls ./ /tmp
./:

/tmp:
hosts           protocols       services        services.bak
```

Changer de nom (une forme de déplacement) :

```
$ mv /tmp/protocols protocols.old
$ ls
protocols.old
$ mv protocols.old protocols2.old
$ ls
protocols2.old
$ mv protocols2.old /tmp/protocols
$ ls
$
```

Effacer :

```
$ cd /tmp
$ ls
hosts           protocols       services        services.bak
$ rm services.bak
$ ls
hosts           protocols       services
$ cd /var
$ rm /tmp/services /tmp/hosts
$ ls /tmp
protocols
$ rm ../tmp/protocols
$ ls /tmp
$
```

Créer un répertoire ou le supprimer :

```
$ cd /tmp
$ ls -p
$ mkdir bidule
$ ls -p
bidule/
$ mkdir bidule/truc
$ ls -p bidule
truc/
$ mkdir -p /tmp/test/foo/bar/buz
$ ls -p
bidule/         test/
$ rm -R /tmp/test
$ ls -p
bidule/
$
```

Et ainsi de suite... Tout cela est assez monotone, mais il faut bien un vocabulaire minimal avant de passer à des choses plus compliquées. Vous découvrirez les sens exact des options **`-p`** de **`mkdir`**, **`-p`** de **`ls`**, et **`-R`** de **`rm`** dans leurs documentations respectives.

ATTENTION **Pas de corbeille !**

Sous Unix, on a de l'hygiène : on ne fouille pas dans les poubelles. Tout fichier effacé est détruit immédiatement et pour toujours.
Pour reproduire le comportement de la corbeille de MacOS ou Windows, il faudra déplacer les fichiers dans un répertoire poubelle au lieu de les supprimer avec la commande **`rm`**.

L'éditeur `vi`

La configuration d'un système Unix est enregistrée dans un certain nombre de fichiers, en général de simples fichiers texte, qui se trouvent pour la plupart dans le répertoire /etc. Nous avons examiné différentes commandes pour manipuler des fichiers, mais tout ceci sera de peu d'utilité sans la maîtrise d'un outil permettant de modifier le contenu des fichiers : un éditeur de textes.

Il existe de nombreux éditeurs de textes sous Unix. Certains d'entre eux fonctionnent avec une interface graphique et ressemblent au bloc-notes de Windows ou à SimpleText sur Mac. Notre but étant la maîtrise de l'interface texte, nous ne les aborderons pas.

Citons certains éditeurs classiques en mode texte : **`ed`**, **`vi`**, **`emacs`**, **`pico`**, **`jed`**, **`vim`**... Les plus répandus sont sans conteste **`vi`** et **`emacs`**. Ils existent sur beaucoup de systèmes Unix, et disposent chacun de leur propre communauté d'utilisateurs. La question du meilleur éditeur est un grand classique sur lequel les adeptes de **`vi`** et **`emacs`** aiment à débattre à l'infini sur Usenet.

Les autres sont tous plus récents et on les trouve plus rarement sur des systèmes Unix trop anciens. **`vim`** est un **`vi`** amélioré (*Vi IMproved*), **`pico`** est l'éditeur fourni avec le logiciel de messagerie en mode texte **`pine`**. Certains éditeurs sont capables de fonctionner en mode graphique le cas échéant : **`gvim`** est la version graphique de **`vim`** ; **`emacs`** et **`xemacs`** peuvent fonctionner à la fois en mode texte et en mode graphique.

On n'a donc que l'embarras du choix. Fort bien, mais quel éditeur choisir ? Cette question reste ouverte pour le novice, auquel on conseillera d'essayer tous les éditeurs mis à sa disposition pour décider lequel lui convient le mieux. Pour l'administrateur, le choix est plus restreint : l'éditeur à apprendre, c'est celui qui est disponible juste après l'installation du système, et avant la configuration de quoi que ce soit d'autre. Sur beaucoup de systèmes et en particulier sur les systèmes BSD, ce choix se limite à **`vi`**.

CULTURE **Usenet**

Usenet est un système distribué de groupes de discussions. Vous en avez peut-être entendu parler sous le nom « les *news* ». C'est un service gratuit qui vous permettra de discuter de tout et de n'importe quoi avec des interlocuteurs du monde entier.
Pour y accéder, il faut disposer d'un logiciel de *news* (KNews, slrn, Netscape, MacSOUP, FreeAgent, WinVN... les clients ne manquent pas), et de l'adresse du serveur de *news* de votre fournisseur d'accès.

VOCABULAIRE **`vi`**

`vi` tient son nom de *VIsual editor*. En anglais, **`vi`** se prononce « vie ail ».

CONFORT **`ee` sous FreeBSD**

Si vous travaillez sous FreeBSD, vous disposez dans la distribution de base d'un éditeur simple à utiliser en plus du traditionnel **`vi`** : **`ee`**.

Une première utilisation de `vi`

Pour démarrer **`vi`**, tapez dans le shell **`vi`** suivi du nom du fichier à éditer, puis validez. On peut aussi invoquer **`vi`** sans indiquer de nom de fichier, auquel cas l'éditeur créera un nouveau fichier, mais il faudra lui indiquer où l'enregistrer au moment de quitter.

```
$ vi /etc/hosts
```

Le shell laisse le terminal à **`vi`** et vous obtenez un écran comme celui-ci :

```
#       $NetBSD: hosts,v 1.6 2000/08/15 09:33:05 itojun Exp $
#
# Host Database
# This file should contain the addresses and aliases
# for local hosts that share this file.
# It is used only for "ifconfig" and other operations
# before the nameserver is started.
#
```

ASTUCES **Si cela ne fonctionne pas**

Si **`vi`** refuse de démarrer, c'est peut-être la variable `TERM`, qui indique le type de terminal, qui est à mettre en cause. Voici un réglage qui convient souvent **`TERM=vt100; export TERM`** en **`sh`**, **`ksh`** ou **`bash`**, ou **`setenv TERM vt100`** en **`csh`** ou **`tcsh`**.
Autres causes d'échec : les répertoires /tmp ou /var/tmp ne sont pas accessibles en écriture. C'est le cas de certaines installations inachevées. Voyez le chapitre 5 pour trouver des solutions à ces problèmes.

```
#
::1                     localhost
127.0.0.1               localhost
#
# RFC 1918 specifies that these networks are "internal".
# 10.0.0.0      10.255.255.255
# 172.16.0.0    172.31.255.255
# 192.168.0.0   192.168.255.255
~
~
~
~
~
~
~
~
/etc/hosts: unmodified, readonly: line 1
```

vi a deux modes de fonctionnement : un mode commande (dans lequel il se place au démarrage) et un mode insertion. En mode commande, **vi** attend des commandes, qui consistent en un ou plusieurs caractères (qui n'apparaîtront pas à l'écran).

Familiarisez-vous avec les déplacements : **h**, **j**, **k** et **l**, qui servent respectivement à déplacer le curseur d'un caractère vers la gauche, le bas, le haut, et la droite.

Le curseur ne se déplace pas toujours exactement comme attendu. En particulier, il ne peut pas atteindre une position située au-delà de la fin d'une ligne, et il se place à la fin des tabulations. Vous pouvez expérimenter ce comportement dans le fichier /etc/hosts : celui des BSD comprend des tabulations dans les lignes décrivant les adresses internes.

Pour taper du texte, il faut passer en mode insertion. Voici certaines des commandes disponibles pour cela (dans un premier temps seules les deux premières vont vous intéresser).

- **i** (*insert*) passe en mode insertion à gauche du curseur.
- **a** (*append*) passe en mode insertion à droite du curseur.
- **o** crée une ligne vide sous la ligne actuelle et passe en mode insertion au début de cette nouvelle ligne.
- **O** de même, au-dessus du curseur.

En mode insertion, tout ce que vous tapez s'affiche à l'écran. Pour se déplacer ou effectuer une quelconque commande, il faudra repasser en mode commande, en enfonçant la touche d'échappement **Échap** .

x est une commande très utile : elle efface le caractère situé sous le curseur. Sans elle, difficile de corriger ses erreurs, dans la mesure où l'on ne peut effacer que ce que l'on vient de taper lorsqu'on est en mode insertion.

CONFORT **Les touches fléchées**

En général, les touches fléchées du clavier permettent aussi de se déplacer, mais elles ne fonctionneront pas sur certains terminaux mal configurés. Il en va de même pour le pavé numérique.

EN CAS DE COUP DUR **Pas de touche Échap ?**

Certains anciens terminaux n'ont pas de touche d'échappement, mais on peut la simuler avec la combinaison **Control**+**[**.

ASTUCE **Pour connaître sa position**

Quand on ne sait plus dans quel mode on se trouve, voici une méthode simple : taper deux fois sur la touche d'échappement **Échap**. Cela a pour effet de revenir en mode commande ou d'annuler toute éventuelle commande complexe, de plusieurs caractères, qui serait en cours de saisie.

PIÈGE À C... **La casse compte**

Attention, la casse des commandes compte. Si **x** minuscule efface le caractère sur lequel est le curseur, **X** majuscule efface le caractère à sa gauche et déplace le curseur d'un cran à gauche. Vous constaterez que **vi** devient très déroutant quand on a verrouillé par mégarde les majuscules (touche **Maj**).

Quitter vi et enregistrer

Si vous avez vraiment pratiqué au fur et à mesure que vous lisiez, vous devez commencer à détester **vi**. Ce fut notre lot commun ; voyons donc comment quitter **vi**.

En mode commande, si vous tapez **:** (deux points), deux points s'affichent en bas de l'écran. **vi** attend une commande longue, et il affichera ce que vous tapez en

bas de l'écran. Une fois la commande terminée, vous devez la valider en pressant sur **Entrée**. Voici les commandes pour enregistrer et sortir :

- **:q** Tente de quitter **vi** (échouera si un fichier en cours d'édition dispose de modifications non enregistrées).
- **:w** Enregistre les modifications.
- **:wq** Enregistre et quitte **vi**.
- **:q!** Quitte **vi** sans rien enregistrer.
- **:w!** Force l'enregistrement (utile pour les fichiers protégés en écriture mais pour lesquels on peut outrepasser cette protection).
- **:w /tmp/hosts.copy** Enregistre sous le nom /tmp/hosts.copy. Cette commande sert notamment quand on a invoqué **vi** sans indiquer de nom de fichier.
- **:e /etc/protocols** Ouvre le fichier /etc/protocols. Cette commande sera rejetée si le fichier en cours de manipulation n'est pas mis à jour sur le disque.

vi agit comme n'importe quel éditeur de textes : il refuse de quitter si toutes les modifications ne sont pas enregistrées. On constate que la ligne de saisie des commandes longues propose toutes les commandes habituelles du menu **Fichier** des éditeurs de textes à interface graphique : **ouvrir**, **enregistrer**, **enregistrer sous**, **quitter**.

Plus loin avec vi

Sous ses dehors abrupts, **vi** est un éditeur de texte très complet, et très apprécié de ceux qui en ont pris l'habitude. Voici quelques commandes utiles :

Annulation, répétition

On peut annuler dans **vi** : ceci se fait avec la commande **u** (*undo*). La répétition de la commande précédente est aussi possible, avec la commande **.** (point).

Numéros de lignes

Parfois un message d'erreur vous indiquera un problème à une ligne donnée d'un fichier de configuration. Dans cette situation, vous trouverez la commande longue **:set ruler** utile : elle vous donnera votre position (colonne, ligne) dans le fichier.

ATTENTION **Overdose de vi**

Si **vi** vous échauffe déjà les méninges, passez à la section suivante, vous en savez déjà assez pour manipuler des fichiers de configuration. Retenez juste que l'on peut annuler, copier/coller, ou rechercher/remplacer. Vous reviendrez ici lorsque vous aurez besoin de ces commandes.

ASTUCES **Autres commandes :set utiles**

Sur une connexion à forte latence, on constate parfois des problèmes avec les touches fléchées. Elles envoient en effet des codes de plusieurs caractères. Quand le dernier arrive trop longtemps après le premier, **vi** considère que cela n'est plus la même commande. Pour régler cela, **:set notimeout**.

Si vous copiez/collez dans **vi** avec des commandes copier/coller externes à **vi** (copier/coller depuis un autre système d'exploitation dans une session SSH d'un XTerm, par exemple), vous serez parfois ennuyé par les indentations automatiques. Pour les supprimer, **:set noai** (*no auto-indent*).

CULTURE **Le caractère $**

Dans les commandes de **vi**, et plus généralement dans beaucoup de programmes Unix, **$** dénote la fin d'une ligne ou d'un fichier. De même, **^** (accent circonflexe) indique souvent un début.

Pour se déplacer rapidement, on peut taper un nombre suivi de la lettre **G**. Cette commande amène au numéro de ligne correspondant. Ainsi, **1G** amène au début du document, et **$G** (ou **G**) vous envoie à la fin.

On peut aussi placer des signets dans le document. À tout moment, la commande longue **:ma a** (ou **ma**) place le signet **a** à la ligne courante (on peut utiliser les 26 lettres majuscules et les 26 lettres minuscules pour les noms de signets). En mode commande, **'a** (apostrophe **a**) amène au signet **a**

Couper, copier, coller

Les copier/coller simples se font ligne par ligne. Il y a pour cela trois commandes : **dd** coupe la ligne courante, **yy** la copie et **p** colle ce qui a été coupé ou copié. Précédées d'un nombre, ces commandes porteront sur le nombre d'occurrences ainsi précisé (nombre de lignes ou de reproductions). Par exemple, **4yy** copie quatre lignes à partir de la ligne courante.

Les commandes copier et coller sont aussi disponibles en commandes longues, sous la forme **:d** et **:y**. On peut les utiliser pour indiquer des zones de fichier plus complexes. Par exemple, **:d** va couper la ligne courante, et **:3,5d** va couper les lignes 3 à 5. **:1,$y** copie l'ensemble du fichier (n'oubliez pas : le dollar représente la fin). **:-1,+3d** coupe à partir de la ligne précédant la ligne courante et jusqu'à la 3ème ligne après la ligne courante. Dernier exemple : **:2,.d** coupe tout de la deuxième ligne à la ligne courante (le **.** (point) représente le numéro de la ligne courante).

On peut aussi utiliser les signets : **:'a,'by** copie les lignes situées entre les signets **a** et **b** ; c'est très pratique.

Rechercher et remplacer

Taper **/** (barre de division), passe **vi** en mode commande longue, et ce caractère s'affiche en bas de l'écran. Tapez un texte à rechercher et validez. **n** passe à l'occurrence suivante, et **N** à l'occurrence précédente. C'est le même comportement que dans les visionneuses classiques **less** et **more**, qui sont utilisées pour les pages **man**.

PLUS LOIN **Les expressions régulières**

Pour approfondir ses connaissances sur les expressions régulières et leur utilisation, le lecteur peut se rapporter à l'ouvrage suivant, référence reconnue dans le domaine : *Maîtrise des expressions régulières*, de Jeffrey E. F. Friedl.
À noter un débat d'expert : certaines personnes plaident pour la traduction de l'anglais « *regular expression* » par « expressions rationnelles », plutôt que « expressions régulières ». Vous pouvez donc être amené à trouver ce terme, qui reste minoritairement employé face à « expressions régulières ».

La commande rechercher/remplacer est un peu complexe. Elle fonctionne comme les couper et copier en commande longue : **:1,$s/foo/bar/** remplace **foo** par **bar** dans tout le fichier. **:'a,'bs/foo/bar/** se contente de remplacer entre les signets **a** et **b**. **:.,+3s/foo/bar/** fait le rechercher/remplacer entre la ligne courante et 3 lignes plus loin.

Par défaut, **vi** ne fait qu'un remplacement par ligne. Pour remplacer plusieurs occurrences sur une même ligne, il faut ajouter le modificateur **g** (global) à la fin de la commande : **:1,$s/foo/bar/g**. Le modificateur **c** demande confirmation pour chaque substitution ; on peut alors choisir **y** (oui), **n** (non), **a** (toutes : *all*) ou **q** (quitter).

Le rechercher/remplacer de **vi** permet des constructions extrêmement complexes à l'aide d'« expressions régulières », langage symbolique permettant de définir

des familles de mots. Il y a beaucoup à dire en la matière, mais trop pour en parler en détail dans ce chapitre. En attendant de devenir un expert, il faut savoir que certains caractères (comme le point) ont un sens particulier dans une recherche pour **vi**. Ainsi, le point correspond à n'importe quel caractère. Pour mentionner un véritable point, il faudra le précéder de **** (*backslash*) comme suit : **\.** (*backslash* point).

Commandes externes

vi est riche de nombreuses possibilités. L'énumération précédente peut décourager, mais il ne faut pas s'inquiéter. Le tout est d'être capable d'éditer des fichiers ; le reste n'est qu'une histoire d'efficacité dans l'utilisation des outils. Très peu de personnes connaissent toutes les commandes de **vi** !

Parfois, malgré ce foisonnement de commandes, on a un besoin que **vi** ne sait pas traiter. Pas de problème dans ce cas, car **vi** sait déléguer à une commande externe. La commande suivante fait appel à **awk** (traitée plus loin) pour échanger les deux colonnes d'un fichier : **:1,$!awk '{print $2,$1}'**

AU SECOURS ! **Je ne supporte déjà plus vi**

vi est indispensable lors de l'installation de la machine et dans les situations critiques, mais en temps normal, vous pouvez tout à fait utiliser un éditeur plus convivial. Nous verrons au chapitre 11 comment installer des éditeurs qui paraîtront plus agréables à utiliser au débutant.

Et si on n'a pas vi ?

Vous aurez peut-être affaire à une situation où **vi** n'est pas disponible. Dans ce cas, il faudra peut-être vous résoudre à utiliser un démon de l'ancien monde : **ed**.

ed est un éditeur adapté aux terminaux imprimante : sans écran, pas d'affichage pleine page, mais ligne à ligne. Vous tapez des commandes, **ed** répond, et il n'est jamais possible de se déplacer dans le document.

ed fonctionne un peu comme **vi**. Il démarre en mode commande. On accède au mode insertion avec les commandes **a** (passe en mode insertion sous la ligne indiquée), **i** (de même, mais au-dessus), et **c** (change la ligne indiquée). Pour ressortir du mode insertion, il faut taper un **.** (point) seul sur une ligne, suivi de la touche **Entrée**.

Voici un exemple de session avec **ed** (à l'ouverture et à l'enregistrement, **ed** indique la taille du fichier : 782 puis 834 octets) :

```
$ ed /tmp/passwd
782
1,5l
root:*:0:0:Charlie &:/root:/bin/ksh
toor:*:0:0:Bourne-again Superuser:/root:/bin/sh
daemon:*:1:31:The devil himself:/:/sbin/nologin
operator:*:2:5:System &:/usr/guest/operator:/sbin/nologin
bin:*:3:7:Binaries Commands and Source:/:/sbin/nologin
2a
insertion après la ligne
numéro 2
.
1,5l
root:*:0:0:Charlie &:/root:/bin/ksh
toor:*:0:0:Bourne-again Superuser:/root:/bin/sh
insertion après la ligne
numéro 2
daemon:*:1:31:The devil himself:/:/sbin/nologin
3d
1,5l
root:*:0:0:Charlie &:/root:/bin/ksh
toor:*:0:0:Bourne-again Superuser:/root:/bin/sh
```

```
numéro 2
daemon:*:1:31:The devil himself:/:/sbin/nologin
operator:*:2:5:System &:/usr/guest/operator:/sbin/nologin
3c
je modifie la ligne 3
Et j'en ajoute une en-dessous
.
1,5l
root:*:0:0:Charlie &:/root:/bin/ksh
toor:*:0:0:Bourne-again Superuser:/root:/bin/sh
je modifie la ligne 3
Et j'en ajoute une en-dessous
daemon:*:1:31:The devil himself:/:/sbin/nologin
w
834
Q
$
```

Heureusement on n'a pas à utiliser **ed** tous les jours. Moins on l'utilise, mieux on se porte.

Faute de **ed**, on peut encore lire des fichiers avec **less** ou **more**, voire avec **cat**, et en créer avec cette dernière commande. L'édition devient impossible, sauf de façon automatisée avec des outils comme **sed** et **awk**, que nous verrons plus tard. Voici un exemple d'utilisation de **cat** pour manipuler des fichiers depuis le shell :

```
$ cd /tmp
$ cat > test
Une fois cette commande tapée, tout ce qui est entré va dans le fichier.
Pour terminer le fichier, il faut taper Control-D en début de ligne
^D
$ ls -l test
-rw-r--r--  1 manu  wheel  142 Mar 15 15:32 test
$ cat test
Une fois cette commande tapée, tout ce qui est entré va dans le fichier.
Pour terminer le fichier, il faut taper Control-D en début de ligne
$ cat >> test
Avec cat >> on ajoute à la fin d'un fichier, alors qu'avec cat > on
écrase le contenu précédent.
^D
$ ls -l test
-rw-r--r--  1 manu  wheel  236 Mar 15 15:33 test
$ cat test
Une fois cette commande tapée, tout ce qui est entré va dans le fichier.
Pour terminer le fichier, il faut taper Control-D en début de ligne
Avec cat >> on ajoute à la fin d'un fichier, alors qu'avec cat > on
écrase le contenu précédent.
$
```

Ainsi se finit cette introduction à l'édition des fichiers. Tout cela peut paraître particulièrement atroce, mais il faut garder à l'esprit que ces outils, s'ils sont difficiles à maîtriser, n'en sont pas moins très puissants. L'apprentissage de **vi** sera sans doute la plus grosse difficulté que vous aurez à affronter dans le monde

CULTURE **Caractères de contrôle**

Les caractères de contrôle sont les caractères obtenus en enfonçant la touche **Control** en même temps qu'une autre touche. Ils ont chacun un sens particulier. **Control**+**C** signifie une interruption, **Control**+**[** signifie un échappement, **Control**+**D** signifie fin de fichier... Dans la littérature technique, les caractères de contrôle sont souvent notés ainsi : **^D** pour **Control**+**D**, **^C** pour **Control**+**C**, et ainsi de suite.

PLUS LOIN **Rechercher/remplacer sans vi**

Anticipons un peu l'édition automatisée avec **sed**, avec ici un rechercher/remplacer dans un fichier :

```
# sed 's/rc_configured=NO/rc.configured=YES' /etc/rc.conf > /etc/rc.conf2
# mv /etc/rc.conf2 /etc/rc.conf
```

La commande **sed** sera abordée plus en détails vers la fin de ce chapitre.

Unix (sauf si vous devez un jour configurer **sendmail**), mais une fois qu'on en a acquis la maîtrise, on a du mal à réutiliser le bloc-notes de Windows.

CULTURE **Sendmail**

Sendmail est le serveur de messagerie historique sous Unix. En 2003, il était encore responsable de plus des trois quarts des serveurs de messagerie sur Internet. Il est extrêmement souple, mais sa configuration est hélas très complexe. Courageusement, nous aborderons sa configuration dans le chapitre 12.

Quelques commandes plus avancées pour le shell

Étudions quelques autres commandes utiles du shell.

Variables d'environnement

Sous Unix, tout processus dispose de variables d'environnement. On les manipule comme dans n'importe quel langage de programmation : lire leur valeur, ou la modifier. Exemple en **sh**, **ksh** ou **bash** :

```
$ toto=blablabla
$ echo $toto
blablabla
$ toto="patati patata"
$ echo $toto
patati patata
```

PLUS LOIN **Valeurs avec des espaces**

Si vous voulez donner une valeur comportant des espaces à une variable, il faut ceindre cette valeur d'apostrophes simples (**'**, protection forte) ou doubles (**"**, protection faible).
Pour inclure un caractère spécial, comme une apostrophe simple ou double dans une valeur de variable, on l'« échappe » en le précédant de \ (*antislash*) comme ceci : **pb=l\'incendie**

Le shell transmet ses variables d'environnement aux programmes qu'il démarre. De nombreux comportements en dépendent. Par exemple, la variable `TERM` indique le type du terminal. Un programme comme **vi** l'utilise par exemple pour déterminer le comportement du clavier. Une mauvaise valeur de `TERM`, et les touches fléchées ne fonctionneront plus.

Tout ceci n'est pas très compliqué, à un piège près : les modifications faites aux variables d'environnement ne sont pas visibles aux processus ou programmes invoqués par le shell tant qu'elles n'ont pas été « exportées » avec la commande **export** :

```
$ TERM=vt220
$ export TERM
```

Dans les shells plus récents que le *Bourne Shell*, on peut combiner ces commandes ainsi : **export TERM=vt220**.

La commande **set** (ou **env**) permet de lire la valeur de toutes les variables d'environnement définies. Exemple avec **ksh** :

```
$ set
KSH_VERSION='@(#)PD KSH v5.2.14 99/07/13.2' ❶
LESSCHARSET=latin1 ❷
LINES=24
LOGNAME=manu
MAIL=/var/mail/manu
MAILCHECK=600
MANPATH=/usr/share/man:/usr/pkg/man:/usr/local/man:/usr/X11R6/man ❸
OLDPWD=/home/manu
OPTIND=1
PAGER=less ❹
PATH=/bin:/usr/bin:/usr/pkg/bin:/usr/local/bin:/sbin:/usr/sbin:/usr/pkg/sbin:/us
r/local/sbin: ❺
PPID=264
PS1='$ ' ❻
PS2='> '
PS3='#? '
```

ALTERNATIVE **En csh et tcsh**

En **csh** et **tcsh**, on dispose de commandes différentes selon que l'on veuille définir une variable visible uniquement par le shell ou une variable « exportée ».
Pour une modification locale au shell, on utilise **set** : **set term=vt100**. Pour une variable « exportée », on utilise **setenv** : **setenv TERM vt220**.

```
PS4='+ '
PWD=/home/manu/bsdbook
RANDOM=3018
SECONDS=15572
SHELL=/bin/ksh
TERM=vt102
TMOUT=0
USER=manu
_=set
toto=patati patata ❼
$
```

❶ Variable créée automatiquement par **ksh**, indiquant son numéro de version.

❷ LESSCHARSET indique à la commande **less** le jeu de caractères à utiliser pour l'affichage. La valeur **latin1** permet à **less** d'afficher les accents.

❸ Lors d'une requête d'affichage d'une page **man**, la commande **man** la recherche dans tous les répertoires de la variable MANPATH.

❹ Cette variable est aussi utilisée par la commande **man**. Elle lui indique quelle visionneuse utiliser pour afficher les pages **man**.

❺ PATH joue un rôle similaire à MANPATH : c'est la liste des répertoires où le shell va rechercher toute commande tapée.

❻ PS1 indique l'invite de commande affichée par le shell. Vous pouvez l'enrichir avec un contenu dynamique :

```
$ PS1="--> "
--> PS1="`whoami`@`hostname`$ "
manu@violette$
```

La page **man** du shell utilisé détaillera la syntaxe et les possibilités de cette variable.

❼ La variable toto que nous avons définie dans un exemple précédent, et que le système n'a pas oubliée depuis !

PLUS LOIN **Une invite de commande sophistiquée avec ksh**

En bricolant un peu, on peut obtenir une invite de commande donnant le répertoire courant, sur un **ksh** relativement récent (celui des BSD fait l'affaire) :

```
$ cwd () { cd $@ ; PS1="`whoami`@`hostname -s`<`pwd`> " ; }
$ alias cd='cwd'
$ cd /home/manu
manu@violette</home/manu> cd ..
manu@violette</home> cd /var/spool/mqueue
manu@violette<var/spool/mqueue>
```

Pour configurer automatiquement cette invite ainsi lors de toute nouvelle session, placez la définition de la fonction de shell **cwd** et de l'alias **cd** dans le fichier .profile de votre répertoire personnel. **ksh** exécute les commandes de ce fichier lors de toute connexion utilisateur.

PLUS LOIN PATH **et ./**

Lorsque l'on demande au shell d'exécuter une commande, il la cherche dans tous les répertoires indiqués par la variable PATH.

Pour exécuter un programme situé dans un répertoire absent du PATH, il faudra fournir au shell un chemin menant à la commande, comme par exemple /usr/local/bin/emacs. Si cette commande se trouve dans le répertoire courant, il suffit d'utiliser comme chemin le nom du répertoire courant : **./emacs**.

Signalons une petite différence assez déroutante pour les habitués du DOS : sous DOS, le répertoire courant est toujours utilisé pour chercher des commandes. Pour rétablir ce comportement sous Unix, il suffit d'ajouter le répertoire courant au PATH : **export PATH=$PATH:.** (en **ksh** et **bash**).

Attention toutefois aux implications en matière de sécurité : si vous vous trouvez dans un répertoire où un autre usager a laissé une commande **ms**, faire une faute de frappe (**ms** au lieu de **ls**) vous fera exécuter le programme prévu par l'autre usager. C'est pour cette raison qu'il est déconseillé, et notamment à root, d'avoir . (point) dans son PATH.

Si vous tenez à mettre . dans votre PATH, surtout placez-le à la fin du PATH, sinon une faute de frappe ne sera même plus nécessaire pour vous faire exécuter un faux **ls** !

Gestion des droits des fichiers

On en a parlé brièvement, les droits sur les fichiers sont gérés à travers les droits du propriétaire, du groupe, et du reste du monde. Les détails de la gestion des groupes qui concernent l'administrateur seront abordés au chapitre 6. Voyons comment modifier les droits sur les fichiers.

Trois commandes servent à gérer les droits sur les fichiers : **chmod**, **chown**, et **chgrp**.

chown et **chgrp** servent respectivement à modifier le propriétaire et le groupe d'un fichier. Elles sont assez simples d'emploi et surtout utilisées par root : on indique le nouveau propriétaire ou groupe, suivi de la liste des fichiers affectés.

```
# cd /tmp
# ls -l
total 64
-rw-r--r--  1 root  wheel  29201 Mar 15 12:35 ch3.sgml
drwxr-xr-x  3 root  wheel    512 Apr 16  2001 linux-2.1.14
-rw-r--r--  1 root  wheel    834 Mar 15 15:25 passwd
-rw-r--r--  1 root  wheel    236 Mar 15 15:33 test
# chown manu ch3.sgml test passwd
# chgrp wsrc linux-2.1.14
# ls -l
total 64
-rw-r--r--  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
drwxr-xr-x  3 root  wsrc     512 Apr 16  2001 linux-2.1.14
-rw-r--r--  1 manu  wheel    834 Mar 15 15:25 passwd
-rw-r--r--  1 manu  wheel    236 Mar 15 15:33 test
```

La commande **chmod** permet de modifier les permissions d'accès d'un fichier. Sa forme la plus simple est d'indiquer

- quels droits modifier : **u** pour utilisateur, **g** pour groupe, et **o** pour les autres (*others*). On peut en préciser plusieurs : **ug**, **og**, ou **uog**.
- la nature de l'opération : **+** pour ajouter des droits, **–** pour en supprimer, et **=** pour fixer une nouvelle valeur indépendamment de la valeur précédente.
- les attributs affectés : **r** pour la lecture (*read*), **w** pour l'écriture (*write*), et **x** pour l'exécution.

Bien entendu, un utilisateur ne peut modifier que les droits des fichiers qu'il possède, sauf root, qui modifie ce qu'il veut. Dans la pratique :

```
$ ls -l ch3.sgml
-rw-r--r--  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
$ chmod og-r ch3.sgml
$ ls -l ch3.sgml
-rw-------  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
$ chmod uog+rw ch3.sgml
$ ls -l ch3.sgml
-rw-rw-rw-  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
$ chmod og=r ch3.sgml
$ ls -l ch3.sgml
-rw-r--r--  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
```

Ces trois commandes admettent une option **–R** qui permet de modifier récursivement un répertoire et son contenu.

Recherche de fichiers

La commande **find** permet de faire des recherches de fichiers assez complexes. Exemple : recherche des fichiers et répertoires d'extension .so dans /usr/local/jdk

```
$ find /usr/local/jdk -name '*.so' -print
/usr/local/jdk/jre/lib/ppc/green_threads/libhpi.so
/usr/local/jdk/jre/lib/ppc/native_threads/libhpi.so
/usr/local/jdk/jre/lib/ppc/classic/libjvm.so

(...)
```

PLUS LOIN **Notation octale**

Une autre syntaxe est souvent utilisée pour préciser les droits : on indique un nombre à 3 chiffres. Le premier chiffre indique les droits de l'utilisateur, le deuxième ceux du groupe, le troisième ceux des autres. Chaque chiffre est la somme des nombres suivants : 1 pour exécution, 2 pour écriture, et 4 pour la lecture. C'est la notation octale.
Ainsi, taper **chmod 644 test** affecte les droits `-rw-r--r--` au fichier test. **chmod 000 test** affecte les droits `----------`, **chmod 755 test** donne les droits `-rwxr-xr-x`, et ainsi de suite.

PLUS LOIN **Droits par défaut**

Tout fichier nouvellement créé reçoit des droits par défaut, déterminés par un masque. C'est la commande **umask** qui pilote ce masque avec une valeur octale indiquant les droits qui ne doivent pas être attribués. Pour les fichiers créés, le droit par défaut est de 666 moins l'**umask**, et pour les répertoires, c'est 777 moins l'**umask**. Ainsi, un umask de 022 fera créer les fichiers en mode 644 et les répertoires en mode 755.

SUR LES AUTRES UNIX **find**

Sur BSD et GNU/Linux, l'option **-print** n'est pas nécessaire. Sur le **find** de System V, que l'on trouve sur la plupart des Unix commerciaux, le **-print** est obligatoire pour que la commande **find** affiche ses résultats.

La commande **find** a de nombreuses options, qui permettent de spécifier des critères de taille, de propriétaire, de droits, de dates de création ou de modification... Les possibilités sont vastes ; consultez la page **man**.

On veut parfois chercher un fichier sur tout le système, **find / -name 'trucmuche' -print** fait l'affaire, mais cette commande est lente, car les systèmes de fichiers Unix ne sont en général pas conçus pour faire rapidement des recherches de fichiers. Pour pallier ce défaut, les Unix récents disposent de la commande **locate**, qui cherche un fichier dans un index rafraîchi toutes les nuits.

PLUS LOIN **locate**

Sous BSD, l'index se trouve dans /var/db/locate.database. Il est remis à jour chaque nuit, par la commande **locate.updatedb**, située dans /usr/libexec. **locate.updatedb** est invoquée par **cron**, que nous verrons au chapitre 8.

Seul inconvénient de cette méthode : les fichiers ajoutés depuis le dernier rafraîchissement de l'index sont introuvables avec **locate**. Inversement, les fichiers détruits depuis la dernière mise à jour sont toujours présents dans l'index. Exemple d'utilisation :

```
$ locate libhpi.so
/usr/local/jdk/jre/lib/ppc/green_threads/libhpi.so
/usr/local/jdk/jre/lib/ppc/native_threads/libhpi.so
$
```

ATTENTION **locate et son index**

Attention, si l'index de **locate** est vide ou grossièrement obsolète, **locate** n'affiche aucun message d'erreur. Quand **locate** ne trouve pas un fichier, il est donc prudent de vérifier avec **find** avant de conclure.

Enfin, on doit parfois rechercher un contenu précis dans un ensemble de fichiers. C'est le travail de la commande **grep**. Exemple d'utilisation simple :

```
# grep manu /etc/*
exports:/home/manu/xsrc 10.0.12.190
group:wheel:*:0:root,manu
group:operator:*:5:root,manu
group:wsrc:*:9:manu
group:manu:*:500:
netstart:# see the ifconfig manual page for details.
passwd:manu:*:500:500:Emmanuel Dreyfus:/home/manu:/bin/ksh
security:# the mtree(8) manual page.
#
```

CULTURE **Recherche rapides sur Mac**

Le système de fichiers de MacOS (HFS, et son évolution HFS+) est conçu pour faire des recherches très rapides. L'invocation du programme est en général plus longue que la recherche proprement dite.
HFS doit sa célérité à l'utilisation de structures appelées « arbres binaires ».

grep admet l'argument **-r** pour faire des recherches récursives dans un répertoire. Sur certains Unix, cette option n'est pas disponible, et il faut composer une commande un peu plus complexe avec l'option **-exec** de **find** pour invoquer une commande externe sur chaque fichier trouvé (**-type f** indique de limiter la recherche aux fichiers, par opposition aux répertoires, liens et autres).

IMPORTANT **Astérisques et apostrophes**

Le caractère * (astérisque) est souvent utilisé comme joker : il représente n'importe quelle suite de caractères. Lorsque le shell voit un astérisque, il le remplace par la liste des fichiers correspondants. Exemple :

```
$ echo /var/yp/*
/var/yp/Makefile.main
/var/yp/Makefile.yp
/var/yp/binding
$ echo /var/yp/Make*
/var/yp/Makefile.main
/var/yp/Makefile.yp
```

Si l'on tente de passer un astérisque en argument à une commande, il sera intercepté, interprété et remplacé par le shell. Pour empêcher cela, il suffit de le protéger par des apostrophes (**'**). Le shell le laissera alors tel quel.

```
$ find /usr/share/doc -type f -exec grep Multics {} \; -print
The Unix operating system drew many of its ideas from Multics,
/usr/share/doc/smm/05.fastfs/1.t
similar to the scheme used by Multics [Feiertag71] have been added.
/usr/share/doc/smm/05.fastfs/5.t
"The Multics Input-Output System",
/usr/share/doc/smm/05.fastfs/6.t
$
```

EN PRATIQUE **{} \ ; ? au secours !**

La commande ci-contre vous a peut-être donné des sueurs froides. Le bon réflexe pour comprendre ce qu'elle fait est d'aller voir la page **man** de **find**, et plus précisément son option **-exec**. En bref, **-exec** indique à **find** que nous désirons exécuter une commande sur chaque fichier trouvé. Ici, il s'agit de la commande **grep Multics**. **find** remplacera **{}** par le nom du fichier trouvé, et il faut ajouter un **\;** à la fin de la commande pour que **find** s'y retrouve (ce caractère est échappé : ainsi le shell comprend qu'il ne le concerne pas et le passe tel quel à la commande **find**).

La vie sur la machine

On n'est pas toujours seul sur un système Unix. Les commandes **w** et **who** permettent de savoir qui d'autre est connecté à un instant donné, depuis quand, depuis où, et quelle commande il est en train d'exécuter :

```
$ w
 5:03PM  up 30 days,  7:09, 5 users, load averages: 0.16, 0.12, 0.09
USER TTY FROM              LOGIN@  IDLE WHAT
vpn   p0 radium.network.i 25Feb03  18days -ksh
manu  p1 melancolie.net.e 04Mar03  1day -ksh
manu  p2 melancolie.net.e Wed06PM     0 systat
jdoe  p3 s083.dhcp212-198  5:03PM     0 w
manu  p4 melancolie.net.e Wed06PM     0 vi mach_notify.c
$
```

Les commandes **write** et **talk** permettent d'interagir avec un utilisateur : **write** affiche un message sur son terminal et **talk** est utilisé pour initier un dialogue interactif. Voyez les pages **man** de ces commandes pour en savoir plus, ou essayez-les.

EN CAS DE COUP DUR **talk ne fonctionne pas**

La commande **talk** utilise le serveur **talkd**. Si celui-ci n'est pas activé, **talk** ne pourra pas établir la communication.
Si vous êtes administrateur de la machine, vous pouvez activer **talkd** en configurant **inetd**. Nous étudierons la marche à suivre au chapitre 8.

Ça travaille là-dedans : les processus

Unix est un système d'exploitation multi-tâches, c'est-à-dire qu'il permet d'exécuter à la fois plusieurs programmes. Une instance de programme en cours d'exécution s'appelle un processus. Le shell utilisé pour démarrer des commandes sur la machine est un processus. Pour chacune des commandes invoquées, Unix crée un nouveau processus.

La commande **ps** permet d'observer la liste de ses processus en cours d'exécution :

```
$ ps
  PID TT STAT    TIME COMMAND
2155 p0 Ss   0:00.42 -ksh
2182 p0 R+   0:00.15 ps
1337 p1 Ss   0:01.03 -ksh
2166 p1 S+   0:01.13 systat
1296 p3 Ss   0:00.43 -ksh
1311 p3 S+   0:22.87 vi ch3.sgml
$
```

CULTURE **Le PID**

Le PID (*Process ID*) identifie de façon unique chaque processus. C'est un nombre variant en général de 1 à 32767. Certains Unix sont capables d'utiliser des PID plus élevés.
Certains Unix allouent les PID de façon séquentielle, d'autres les choisissent au hasard. Seule règle universelle : deux processus différents ne peuvent pas partager le même PID simultanément.

On trouve ici le PID, le nom du terminal auquel est attaché le processus, son état (voir la page **man** de **ps** pour les détails), le temps qu'il a consommé en exécution sur le processeur, et le nom de la commande invoquée.

Par défaut, **ps** n'affiche que vos processus attachés à un terminal (un dispositif d'affichage sur écran couplé à un dispositif de saisie au clavier). **ps -x** affiche vos processus non attachés à un terminal, et **ps -a** affiche les processus des autres. **ps -ax** affiche tous les processus.

VOCABULAIRE **Tâche de fond**

On parle souvent de processus fonctionnant en « tâche de fond » pour les processus non attachés à un terminal.

```
$ ps -ax
  PID TT STAT    TIME COMMAND
    0 ?? DKs  0:00.09 [swapper]
    1 ?? Ss   0:00.62 init
    2 ?? DK   0:00.00 [scsibus0]
    3 ?? DK   0:00.00 [scsibus1]
    4 ?? DK   0:00.03 [pagedaemon]
    5 ?? DK   0:03.58 [reaper]
    6 ?? DK   0:58.93 [ioflush]
    7 ?? DK   0:00.15 [aiodoned]
   79 ?? Ss   0:00.37 /usr/sbin/syslogd -s
  306 ?? Ss   0:07.09 /usr/sbin/sshd
  315 ?? Is   0:00.14 /usr/sbin/inetd -l
  320 ?? Ss   0:01.73 /usr/sbin/cron
 1293 ?? Ss   0:00.35 sshd: manu [priv]
 1295 ?? S    0:39.57 sshd: manu@ttyp3
 1334 ?? Ss   0:00.38 sshd: manu [priv]
 1336 ?? S    0:01.82 sshd: manu@ttyp1
 2152 ?? Ss   0:00.37 sshd: manu [priv]
 2154 ?? S    0:00.28 sshd: manu@ttyp0
 2155 p0 Ss   0:00.44 -ksh
 2186 p0 R+   0:00.10 ps -ax
 1337 p1 Ss   0:01.03 -ksh
 2166 p1 S+   0:01.55 systat
 1296 p3 Ss   0:00.43 -ksh
 1311 p3 S+   0:25.17 vi ch3.sgml
  322 E0 S    0:00.56 /usr/libexec/getty std.9600 ttyE0
  323 00 Is+  0:00.20 /usr/libexec/getty std.38400 tty00
$
```

Cette liste est issue d'un NetBSD 1.6. On y trouve un certain nombre de processus fonctionnant en mode noyau, dont les noms sont entre crochets. Ne vous souciez pas trop de ces processus : vous ne pourrez ni les tuer ni les redémarrer, car ils font partie intégrante du système.

On trouve également quelques programmes en tâche de fond, qui assurent des services : **syslogd**, **cron**, **inetd**, **sshd**, deux **getty** qui attendent une identification de session sur leurs terminaux d'attache, et les commandes démarrées par un utilisateur qui est justement en train de taper le présent chapitre dans **vi**.

ps peut afficher de nombreuses autres informations pour chaque processus. Pour en savoir plus, voyez sa page **man**.

La commande **kill** permet de tuer un processus. Pour tuer le processus **systat**, il suffit d'invoquer **kill** en indiquant son PID : **kill 2166**

kill envoie un signal au processus lui demandant de se terminer proprement. Si le processus est vraiment planté, cela peut ne pas suffire, il faut alors requérir sa destruction immédiate et sans délais, avec le signal **-9** : **kill -9 2166**

SUR LES AUTRES UNIX **Des divergences de ps**

La syntaxe de la commande **ps** diffère malheureusement fortement entre BSD et GNU/Linux d'une part et System V d'autre part. Sur les UNIX System V, il faut taper **ps -ef** pour avoir tous les processus.
Les anciens systèmes BSD ont aussi une syntaxe particulière, sans **-** : il faut taper **ps ax** au lieu de **ps -ax**.
Pour compliquer le tout, il y a des hybrides : AIX d'IBM, qui est un System V, utilise la syntaxe BSD, et certains Unix comprennent les deux syntaxes, en faisant la distinction sur la présence ou non du **-**.

B.A.-BA **Le noyau**

Le noyau (*kernel* en anglais) est le cœur du système d'exploitation. C'est lui qui suspend et qui reprend l'exécution des processus des utilisateurs à intervalles de temps régulier, donnant ainsi l'impression d'un système multi-tâches. Il doit aussi fournir aux processus l'accès au matériel et à diverses ressources telles que le système de fichiers ou les terminaux. C'est encore lui qui assure la sécurité du système, en refusant par exemple de laisser un utilisateur modifier un fichier sur lequel il n'a pas de droits.

PLUS LOIN **Processus morts vivants**

Un processus tué peut ne pas disparaître immédiatement si le système a encore besoin de lui pour référence. Le processus apparaît alors toujours sous **ps**, avec un nom entre parenthèses, et un `Z` dans la colonne d'état : c'est un processus zombie. Tant que les zombies apparaissent de façon isolée, ne vous en inquiétez pas : ils ne consomment pas de ressources, et disparaîtront d'eux-mêmes lorsqu'ils auront réglé les affaires de famille qui les retiennent dans le monde des vivants.
Un grand nombre de zombies indique par contre un dysfonctionnement d'une commande. Si votre système est peuplé de zombies, il est sans doute temps de faire un rapport de bogue.

Devenez un grand sorcier avec le shell

Cette section donne des ouvertures vers des utilisations plus avancées du shell. Il est bien entendu inutile de tout maîtriser immédiatement, vous aurez tout le temps de vous familiariser avec tout cela à mesure que vous utiliserez Unix.

Un peu de plomberie

Une des grandes forces d'Unix, c'est sa capacité de renvoyer la sortie d'une commande à l'entrée d'une autre. Cela se fait dans le shell avec les tuyaux (*pipes* en anglais). L'utilisation la plus simple consiste à invoquer une commande qui affiche plus d'une page de résultats et à envoyer sa sortie dans **more** ou **less** pour pouvoir tout examiner convenablement : **find / -type f | less**

On peut donc emboîter des tuyaux de commandes pour faire travailler des commandes à la chaîne. Exemple détaillé :

```
$ ps -ax | grep systat
2198 p0 S+   0:00.13 grep systat
2166 p1 S+   0:02.64 systat
$ ps -ax | grep systat | awk '{print $1}'
2201
2166
$ ps -ax | grep systat | awk '{print $1}' | xargs kill
kill: 2203: No such process
$
```

Le processus **grep**, de PID 2203, est déjà terminé au moment de l'exécution du **kill**, d'où le message d'erreur. En revanche, le processus **systat**, de PID 2166, n'en réchappera pas.

Sortie standard et erreur standard

Chaque commande produit deux flux : la sortie standard et l'erreur standard, où sont évidemment envoyés les messages d'erreur. Ces deux flux s'affichent à l'écran en temps normal. Les commandes disposent également d'un flux d'entrée standard.

Lorsque l'on utilise des tuyaux, seule la sortie standard est transmise dans un tuyau au processus suivant, sur son entrée standard. L'erreur standard va continuer à s'afficher à l'écran. Pour éviter cela et la joindre à la sortie standard, il faut faire une redirection comme ceci (syntaxe valable en **sh**, **ksh** et **bash**) :

```
$ find /etc -type f -exec grep truc {} \;  2>&1 | less
# reports of network infrastructure difficulties
grep: /etc/hosts.equiv: Permission denied
grep: /etc/master.passwd: Permission denied

(...)
```

PLUS LOIN **man, less, et les tuyaux**

Les tuyaux, c'est exactement ce que la commande **man** utilise pour afficher ses pages. On peut la simuler ainsi : **groff -Tascii -man /usr/share/man/man1/ls.1 | less**
La ligne ci-dessus est très utile lorsque l'on veut lire une page **man** située à un endroit où la commande **man** ne va pas la chercher…

ALTERNATIVE **En csh et tcsh**

En **csh** et **tcsh**, la redirection dans un tuyau se fait ainsi : **find /etc -type f -exec grep truc {} \; |& less**
Et comme suit dans un fichier : **find /etc -type f -exec grep truc {} \; >& logerr**

On peut aussi rediriger la sortie standard ou l'erreur standard vers un fichier pour consultation ultérieure, ou vers /dev/null pour s'en débarrasser (syntaxe valable en **sh**, **ksh** et **bash**) :

```
$ grep -r truc /etc 2> err > out
$ ls -l err out
-rw-r--r--  1 manu  wheel  629 Mar 15 17:25 err
-rw-r--r--  1 manu  wheel  220 Mar 15 17:25 out
$ cat err
grep: /etc/hosts.equiv: Permission denied
grep: /etc/master.passwd: Permission denied
grep: /etc/skeykeys: Permission denied
grep: /etc/spwd.db: Permission denied
grep: /etc/ppp/pap-secrets: Permission denied
grep: /etc/ppp/chap-secrets: Permission denied
grep: /etc/ssh_host_key: Permission denied
grep: /etc/ssh_random_seed: Permission denied
grep: /etc/X11/xdm/authdir: No such file or directory
grep: /etc/ssh_host_dsa_key: Permission denied
grep: /etc/ssh/ssh_host_key: Permission denied
grep: /etc/ssh/ssh_host_dsa_key: Permission denied
grep: /etc/ssh/ssh_host_rsa_key: Permission denied
grep: /etc/cgd: Permission denied
$ cat out
/etc/aliases:# reports of network infrastructure difficulties
/etc/mail/aliases:# reports of network infrastructure difficulties
/etc/rc.d/raidframe:    # Initiate parity/mirror reconstruction as needed, in th
e background.
$ grep -r truc /etc 2>/dev/null > out2
$ ls -l out2
-rw-r--r--  1 manu  wheel  220 Mar 15 17:26 out2
$ cat out2
/etc/aliases:# reports of network infrastructure difficulties
/etc/mail/aliases:# reports of network infrastructure difficulties
/etc/rc.d/raidframe:    # Initiate parity/mirror reconstruction as needed, in th
e background.
$
```

PLUS LOIN **/dev/null**

/dev/null est un fichier spécial qui n'a pas de fond : une sorte de trou noir où tout ce qui est écrit est perdu, sans occuper de place sur le disque.

Manipulation de texte avec sed et awk

Quelques commandes, telles que **grep**, **tr**, **sed**, ou **awk**, permettent des manipulations de texte extrêmement puissantes. Nous avons vu que **grep** servait à rechercher des chaîne de caractères dans les fichiers. Voyons rapidement les usages les plus courants de **sed** et **awk**.

sed sert à travailler sur un fichier ligne à ligne. Son usage le plus courant est le rechercher/remplacer, que l'on peut limiter à certaines lignes, comme dans **vi** :

```
$ cat > /tmp/test
Ceci est un test
foo bar buz bar
bidon
^D
$ sed 's/bar/foo/g' /tmp/test
Ceci est un test
foo foo buz foo
bidon
$ sed '2,$s/u/U/' /tmp/test
Ceci est un test
foo bar bUz bar
bidon
$ sed 's/bar$/BAAAR/' /tmp/test > /tmp/test2
$ cat /tmp/test2
Ceci est un test
foo bar buz BAAAR
bidon
$ sed 's/foo \(.\{1,3\}\) buz/\1/' /tmp/test2
Ceci est un test
bar BAAAR
bidon
$
```

PLUS LOIN **Expressions régulières**

Les derniers exemples de **sed** mettent en jeu des expressions régulières. **bar$**, signifie que l'on veut un **bar** ancré en fin de ligne. De même, on pourrait indiquer **^foo** pour un **foo** ancré en début de ligne.

Le bouquet final montre un usage plus complexe d'expression régulière. **.** (point) signifie n'importe quel caractère, **\{1,3\}** l'atome précédent (n'importe quel caractère) de 1 à 3 exemplaires, et **sed** remplace le **\1** dans la partie « remplacer » par ce qu'il a trouvé dans la première paire de **\(\)**. Tout ceci est également utilisable dans la commande rechercher/remplacer de **vi**.

sed fait d'autres choses que des rechercher/remplacer, mais nous allons nous en tenir à cela pour le moment. Vous vous rendrez rapidement compte que **sed** est assez handicapé pour gérer les retours chariot. **tr** est mieux adapté à cet usage :

```
$ cat /tmp/test | tr '\n' ' '
Ceci est un test foo bar buz bar bidon
$
```

awk est plus utilisé pour travailler sur des colonnes. On lui donne des commandes à appliquer à toutes les lignes d'un fichier, où **$1** représente la première colonne, **$2** la deuxième, etc. On peut indiquer le séparateur de colonnes, des conditions sur les lignes à traiter, des instructions à exécuter avant ou après traitement du fichier. Les possibilités sont très vastes : on peut même faire des boucles `while` ou des tests `if` avec la même syntaxe qu'en C.

Quelques exemples simples :

```
$ ps
  PID TT STAT    TIME COMMAND
2155 p0 Ss   0:01.07 -ksh
2336 p0 R+   0:00.10 ps
1337 p1 Ss+  0:01.03 -ksh
1296 p3 Ss   0:00.44 -ksh
1311 p3 S+   0:36.96 vim ch3.sgml
$ ps |awk '{print $2}'
TT
p0
p0
p0
p1
p3
p3
$ ps |awk '{if (NR != 1){print $2}}'
p0
p0
p0
p1
p3
p3
$ ps |awk '{if (NR != 1){print $2}}' | sort -u
p0
p1
p3
$ awk -F: '{if ($4 == 0){print $1}}' /etc/passwd
root
toor
$ awk 'BEGIN {FS=":"; ORS=","} {if ($4 == 0){print $1}}' /etc/passwd
root,toor,
$ awk 'BEGIN {FS=":"; ORS=","} {if ($4 == 0){print $1}}' /etc/passwd | sed 's/,$//'
root,toor
$
```

PLUS LOIN **Traductions des retours chariot**

Les systèmes d'exploitation les plus répandus sont Windows, MacOS et Unix. À eux trois, il ont réussi l'exploit d'adopter trois conventions différentes pour les retours chariot dans les fichiers texte.
Windows utilise *Carriage Return* (noté `CR`, `^M`, ou `\r`) suivi de *Line Feed* (noté `LF`, `^J`, ou `\n`). MacOS utilise *Carriage Return* seul, et Unix utilise *Line Feed* seul.

tr peut être utilisé pour visualiser sous Unix un fichier texte venant de MacOS : **`tr '\r' '\n' < fichiermac.txt`** (cette syntaxe, valable en **sh**, **ksh** et **bash**, redirige l'entrée standard de la commande depuis le fichier indiqué).

Montez votre machine infernale : les scripts shell

Un script shell est une liste de commandes de shell à exécuter. Pour faire un script shell, c'est très simple, il suffit d'une ligne magique au début du fichier, qui indique le chemin de l'interpréteur à invoquer sur le script, précédé par les caractères **#!** (dièse point d'exclamation).

```
$ cat > /tmp/script.sh
#!/bin/sh

echo "Hello world"
^D
$ chmod uog+rx /tmp/script.sh
$ /tmp/script.sh
Hello world
```

Nous pouvons finir cette section par quelques constructions de shell que vous pourrez revenir piocher le jour où vous aurez besoin de faire un script. Ce petit script ne prétend être exhaustif : il se contente d'illustrer quelques exemples de programmation en shell.

```
#!/bin/sh

# Attribution de variable: pas d'espaces autour du =
# pas de dollar devant le nom de la variable
hello="Hello world"

# La valeur d'une variable s'obtient en lui préfixant un $
echo $hello

# Tests sur des chaînes: il faut ajouter un caractère devant
# la variable pour éviter des problèmes si elle est vide.
# Les espaces autour des crochets sont obligatoires.
# Voir la page man test(1) pour les tests disponibles.
if [ "x$hello" = "xHello world" ] ; then echo "yes"; fi

# Les shells anciens ne connaissent pas le «non» logique. Il faut le simuler
# avec un «ou» logique (la seconde partie du «ou» n'étant évaluée que si
# la première partie est fausse, le «ou» a l'effet d'un «si ce qui suit est
# faux, exécute la deuxième partie»)
test -x /etc/passwd || echo "/etc/passwd n'est pas exécutable"

# De même, on peut conditionner l'exécution d'une commande à une autre
# à l'aide d'un «et» logique: si ce qui suit est vrai, exécute la suite
grep manu /etc/passwd > /dev/null && echo "J'ai un compte ici"

# On peut donner à une variable le résultat d'une commande avec $( )
login=$( awk -F: '{if ($4 == 500){print $1}}' /etc/passwd )

# On peut annuler l'effet d'un caractère spécial (ex: $) avec \
# Le shell interprète le contenu dans "" mais pas dans ''
var=1
echo "\$var vaut $var"   # affiche $var vaut 1
echo '\$var vaut $var'   # affiche \$var vaut $var

# Les compteurs se font avec la commande expr.
# Les espaces autour des crochets sont toujours obligatoires.
# Les `` sont un synonyme de $( ), mais on ne peut pas les emboîter.
i=1
while [ $i -lt 10 ] ; do
        echo "i = $i"
        i=`expr $i + 1`
done
```

EN CAS DE COUP DUR Le script shell ne fonctionne pas

Un script shell peut ne pas fonctionner pour différentes raisons : oubli des droits en exécution, ou la ligne **#!** n'introduit pas un chemin d'exécutable existant.
Parfois, le script démarre correctement mais ne fonctionne pas comme on le voudrait. L'option **-x** de **sh** permet d'afficher exactement tout ce qui se passe :

```
$ sh -x /tmp/script.sh
+ echo Hello world
Hello world
```

PLUS LOIN Les scripts Perl

Le shell a l'avantage d'une compatibilité assez large, mais il est assez inconfortable à l'usage. Beaucoup d'administrateurs préfèrent écrire leurs scripts en Perl, un langage interprété taillé pour cela.
Perl est inclus dans la distribution de base d'OpenBSD et de FreeBSD avant la version 5.0. Il est fourni dans le système de paquetages pour NetBSD et FreeBSD 5.0. Pour vous mettre le pied à l'étrier, voici un script **perl** simple (attention, la première ligne doit indiquer l'emplacement de l'exécutable **perl** sur votre système) :

```
#!/usr/bin/perl
print "hello world\n";
```

Perl est un langage aux possibilités très vastes. Pour l'apprendre, un livre sera probablement le bienvenu, mais à défaut vous pouvez commencer par sa page **man** : `perl(1)`.

D'une machine à l'autre

Si vous disposez de comptes sur plusieurs machines UNIX différentes, vous ne tarderez pas à avoir besoin de vous connecter d'une machine à l'autre – pour invoquer des commandes ou copier des fichiers.

L'ancien monde : `telnet`, FTP, `rlogin`, `rsh`, et `rcp`

Sur les UNIX les plus anciens, les services **`telnet`** et **`rlogin`** étaient disponibles pour démarrer une session à distance, et le service **`rsh`** était souvent utilisé pour exécuter une unique commande sur une machine distante. Pour les copies de fichier, que ce soit depuis ou vers la machine proposant le service, FTP et **`rcp`** ont un temps été les standards. Ces protocoles s'utilisaient respectivement avec les commandes **`telnet`**, **`rlogin`**, **`rsh`**, **`ftp`**, et **`rcp`**. Exemple :

```
plume$ telnet spoutnik.example.net
Trying 192.0.2.15...
Connected to spoutnik.example.net
Escape character is '^]'.


UNIX(r) System V Release 4.0 (spoutnik)

login: manu
password: azerty
spoutnik$ exit
Connection closed by foreign host
plume$ ftp spoutnik.example.net
Connected to spoutnik.example.net.
220-
220 192.0.2.15 FTP server ready.
Name (spoutnik.example.net:manu): manu
331 Password required for manu.
Password: azerty
230 User manu logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get temp
local: temp remote: temp
229 Entering Extended Passive Mode (|||15861|)
150 Opening BINARY mode data connection for 'temp' (281500 bytes).
281500 bytes received in 00:06 (44.88 KB/s)
ftp> put group
local: group remote: group
229 Entering Extended Passive Mode (|||17913|)
150 Opening BINARY mode data connection for 'group'.
226 Transfer complete.
317 bytes sent in 00:00 (1.44 KB/s)
ftp> quit
plume$
```

B.A.-BA **Particularités de FTP**

FTP est le seul rescapé de tous ces services, puisqu'il reste couramment utilisé sur Internet comme protocole de mise à disposition de fichiers pour le public. On parle alors de FTP anonyme : la connexion se fait en tant que l'utilisateur **`ftp`** ou **`anonymous`**, sans mot de passe – certains sites exigent qu'on tape à la place son adresse électronique. Dans tous les cas, aucun vrai mot de passe ne transitant, le fait que FTP ne chiffre rien n'est pas un problème.

FTP dispose de fonctionnalités utiles, comme la reprise d'un téléchargement interrompu (commande **`reget`** de **`ftp`**). À l'inverse, la fonctionnalité de transfert de fichiers en mode ASCII ou binaire est une éternelle source de corruption de fichiers pour les usagers qui n'y prennent pas garde : si vous devez transférer des fichiers qui ne contiennent pas que du texte brut, assurez-vous de bien être en mode binaire (commande **`binary`** de **`ftp`**), que vous préférerez au moindre doute.

ATTENTION **Invites de commande**

Dans ces exemples, on travaille sur plusieurs machines différentes. Pour éviter les confusions, l'invite de commande `$` est ici préfixée du nom de la machine à laquelle on s'adresse. Bien entendu, ce nom fait partie de l'invite de commande et il ne faut pas le taper.

Exploitation à distance avec SSH

Tous ces services étaient fondamentalement non sûrs dans la mesure où ils laissaient circuler les mots de passe en clair à travers le réseau, ce qui facilitait la tâche aux pirates. Les services **rlogin**, **rsh** et **rcp** disposaient de plus de mécanismes permettant de passer d'une machine à l'autre sans avoir à donner de mot de passe, véritable catastrophes sur le plan de la sécurité. Ils sont donc tombés en désuétude et ont été remplacés par un unique service : SSH. Ce dernier utilise une cryptographie forte pour assurer des communications authentifiées, confidentielles, et inaltérées.

ATTENTION **Pas de communication sans serveur**

Que ce soit pour les commandes **telnet**, **ftp**, **rlogin**, **rsh**, **rcp**, ou pour les plus récentes **ssh**, **scp** et **sftp**, aucune commande n'est utile si la machine cible n'offre pas le service correspondant. Ainsi, si la machine sur laquelle vous désirez vous connecter n'a pas de serveur SSH en fonction, vous ne pourrez pas utiliser **ssh**, **scp** ou **sftp**.

La commande **ssh** permet de se connecter à une machine proposant le service SSH pour entamer une session, à la manière de **telnet** ou **rlogin**, ou pour exécuter une simple commande, à la manière de **rsh**. Les commandes **scp** et **sftp** permettent les échanges de fichiers avec un serveur SSH, et leurs interfaces ressemblent respectivement à celles de **rcp** et **ftp**. Voici quelques exemples d'utilisation :

```
plume$ ssh violette
manu@violette's password: azerty
violette$ exit
plume$ ssh violette ls -l /dev/wd0a
manu@violette's password: azerty
brw-r-----  1 root  operator  13, 0 Jul  7  2003 /dev/wd0a
plume$ ssh guest@violette
guest@violette's password: guest
violette$ exit
plume$ scp violette:/etc/passwd ./
manu@violette's password: azerty
passwd                                        100%  830    42.1KB/s   00:00
plume$ ls -l passwd
-rw-r--r--  1 manu  manu  830 Jan 17 15:57 passwd
plume$ scp /etc/group violette:/tmp
manu@violette's password: azerty
group                                         100%  317    98.8KB/s   00:00
plume$ ssh violette ls -l /tmp/group
manu@violette's password: azerty
-rw-r--r--  1 manu  wheel  317 Jan 17 15:58 /tmp/group
plume$ sftp violette
Connecting to violette...
manu@violette's password: azerty
sftp> cd /tmp
sftp> ls
.
..
.font-unix
group
sftp> get group
group                                         100%  317     1.4KB/s   00:00
sftp> quit
plume$ ls -l group
-rw-r--r--  1 manu  manu  317 Jan 17 16:00 group
plume$
```

ASTUCE **Quitter une session SSH**

Il arrive parfois qu'une session SSH reste bloquée, quand par exemple le shell sur la machine distante s'est planté. On peut alors forcer la clôture de la session SSH en tapant la séquence d'échappement suivante : **~.** (tilde suivi de point) – ceci nous amène à une autre astuce : comme il sert aux séquences d'échappement, on ne peut pas afficher un tilde directement dans une session SSH. Pour l'obtenir, il faut taper **~~** (tilde deux fois).
La séquence d'échappement pour fermer une session **telnet** est différente : il faut taper **Control+]** puis entrer la commande **quit**.