Formation

Notre expertise est votre avenir

Unix / Linux
Les fondamentaux

UNLI-FO - Rev. 7

1. PRESENTATION DU MONDE UNIX / LINUX

OBJECTIFS

Comprendre le contexte historique et définir :

- Les systèmes d'exploitation Unix et Linux
- Les logiciels libres et de l'Open Source
- GNU, Linux et les distributions
- Définition du Shell

1.1 Unix

1.1.1 Historique d'UNIX

Unix est un système d'exploitation supporté par un grand nombre d'architectures matérielles.

Exemples :

P.C Intel - Pentium	= SCO, SOLARIS
Serveurs IBM/BULL - Risc Power PC	= AIX
Serveurs SUN - Risc Ultra Sparc	= SOLARIS
Serveurs Risc HP-Precision	= HP-UX
Serveurs Compaq/DEC – Risc Alpha	= True64

Développé dans les années 1960, par les laboratoires Bell's d'AT&T, il est multitâche, multi-utilisateurs, temps partagé.

Unix devient portable en 1973, après réécriture en langage C.

Elaboré pour garder une totale indépendance vis à vis des plates-formes matérielles, Unix introduit l'idée de compatibilité au niveau source. Cela veut dire qu'il suffit de recompiler le ou les fichiers sources d'une application pour la faire fonctionner sur un autre type de machine Unix.

Suite à une convention signée avec le Gouvernement Américain, AT&T n'a plus eu le droit de commercialiser le code source « Unix System V ». N'étant plus rémunérée sur les licences Unix vendues l'entreprise décide de distribuer les « programmes sources » aux universités.

L'université de BERKELEY décide alors de développer sa propre version d'Unix, baptisée BSD (Berkeley Software Distribution).

Unix-BSD est à l'origine des plus importantes avancées dans le monde Unix avec notamment les développements de produits tels que « TCP/IP », éditeur « vi », C-Shell, job control...

Par la suite, certains constructeurs d'ordinateurs ont adapté leur propre système Unix (SunOS, MS-Xenix, SCO, DEC/ULTRIX, HP-UX, Sun/Solaris, IBM/AIX...).

Fin des années 1960, AT&T

Système d'exploitation Unix

- Multitâches,
- Multi-utilisateurs,
- Temps partagé.

1973 : Unix devient portable après recompilation en langage C

BSD Berkeley Software Distribution.

- TCP/IP
- vi
- C-Shell
- job control…

Systèmes Unix constructeurs

- MS-Xenix
- SCO
- DEC : ULTRIX
- Hewlett-Packard : HP-UX
- Sun : Solaris
- IBM : AIX...

1.2 Logiciel libre

1.2.1 Définition

Un logiciel libre est un logiciel diffusé sous une licence Copyleft, généralement la licence publique générale GNU (GNU GPL), défini par la Free Software Foundation.

Ils constituent une vraie alternative aux logiciels commerciaux qualifiés de « propriétaires » ou encore de « privateurs ».

Le terme « logiciel libre » fait référence à la liberté d'expression et non à la gratuité.

Elle repose sur 4 types de liberté :

1. La liberté d'exécuter le programme pour tous les usages.
2. La liberté d'étudier le fonctionnement du programme et de l'adapter à vos besoins. L'accès au code source est une condition requise.
3. La liberté de redistribuer des copies.
4. La liberté d'améliorer le programme et de publier vos améliorations, pour en faire profiter toute la communauté. L'accès au code source est une condition requise.

La documentation d'un logiciel libre devrait aussi être sous une licence libre :

GNU Free Documentation Licence (GNU FDL).

Ne confondez pas la notion de Logiciel libre avec :

- Le freeware (graticiel), logiciel mis gratuitement à disposition par son créateur en tant que logiciel propriétaire, auquel cas il est soumis à certaines contraintes quant à sa diffusion.
- Le shareware (partagiciel), un logiciel protégé par le droit d'auteur, qui peut être utilisé gratuitement durant une certaine période ou un certain nombre d'utilisations. Après la période de gratuité, l'utilisateur doit acheter le logiciel s'il veut continuer à l'utiliser.

Des associations se sont créées pour promouvoir les logiciels libres :

- L'APRIL
- L'AFUL

1.2.2 L'APRIL

L'APRIL est constituée de personnes, d'entreprises, d'associations et d'organisations provenant de divers horizons. Depuis 1996, c'est un acteur majeur de la démocratisation et de la diffusion du logiciel libre et des standards ouverts auprès du grand public, des professionnels et des institutions dans l'espace francophone.

Elle a pour objectifs de :

- Promouvoir le logiciel libre dans toutes les sphères de la société,
- Sensibiliser le plus grand nombre aux enjeux des standards ouverts et de l'interopérabilité,
- Obtenir des décisions politiques, juridiques et réglementaires favorables au développement du logiciel libre et aux biens communs informationnels,
- Favoriser le partage du savoir et des connaissances.

1.2.3 L'AFUL

L'Association Francophone des Utilisateurs de Logiciels Libres (AFUL), fondée en 1998 un par Stéfane Fermigier, Jean-Pierre Laisné, Bernard Lang, Thierry Stœhr et Nat Makarevitch, regroupe des utilisateurs, des professionnels du logiciel libre, des entreprises commerciales, des associations qui sont installés dans les régions francophones. Son objectif principal est de promouvoir les logiciels libres comme les systèmes libres GNU / Linux ou BSD (Berkeley Software Distribution).

Information :

BSD désigne une famille de systèmes d'exploitation Unix, développés à l'Université de Californie (Berkeley) : FreeBSD, NetBSD et OpenBSD.

1.2.4 Le GNU

L'acronyme récursif GNU signifie « GNU's Not Unix » (GNU n'est pas Unix) et se prononce gnou.

Le 27 septembre 1983, Richard Stallman, chercheur au laboratoire d'intelligence artificielle du MIT (Massachusetts Institute of Technology) annonce le développement du projet GNU qui a pour objectif de construire un système d'exploitation compatible avec Unix et dont la totalité des logiciels sont libres.

En janvier 1984, il démissionne de son poste pour commencer à écrire les logiciels du projet GNU. C'est aussi pour empêcher le MIT de s'immiscer dans la distribution du projet GNU, de prétendre la propriété de son travail et de lui imposer ses propres conditions de distribution, voire en faire un paquetage de logiciels propriétaires.

1.2.5 La Free Software Foundation

En octobre 1985, Richard Stallman fonde la FSF (Free Software Foundation), une organisation américaine à but non lucratif, pour aider le financement du projet GNU et la communauté du logiciel libre. Elle vend des tee-shirts, CD-ROM et des distributions GNU pour gagner des fonds.

« Notre mission est de préserver, protéger et promouvoir la liberté d'utiliser, étudier, copier, modifier et redistribuer les programmes informatiques, et de défendre les droits des utilisateurs de logiciel libre. »

1.2.6 Le GPL

En janvier 1989, Richard Stallman rédige la 1ère version de GNU General Public License (GNU GPL ou tout simplement GPL).

Le tribunal de grande instance de Paris a jugé applicable la licence GPL version 2 en France le 28 mars 2007.

Publiée le 29 juin 2007, la version 3 de cette licence a été réalisée avec l'aide d'Eben Moglen, professeur de droit et d'histoire du droit à l'école de droit de Columbia, Manhattan (New York).

1.2.7 Le LGPL

La GNU LGPL (GNU Lesser General Public licence) a été créée pour que certains logiciels libres puissent exister dans certains domaines où la publication libre entièrement d'un code était impossible.

Cette licence, limitée ou amoindrie, s'applique le plus souvent aux bibliothèques (librairies).

LGPL signifiait au tout début GNU Library General Public Licence. Le « L » de Library est devenu par la suite Lesser : GNU Lesser General Public Licence car certains développeurs pouvaient choisir cette licence au lieu du GPL du fait que le « L » signifiait Library.

1.2.8 La GPL Affero

La licence publique générale GNU Affero (GNU AGPL) est une version modifiée de la version 3 de la GNU GPL ordinaire. Elle a une seule exigence supplémentaire : si vous exécutez un programme sur un serveur et laissez d'autres utilisateurs communiquer avec celui-ci, votre serveur doit aussi leur permettre de télécharger le code source correspondant au programme qui est exécuté. Si ce qui s'exécute est votre version modifiée du programme, les utilisateurs du serveur doivent obtenir le code source tel que vous l'avez modifié.

1.2.9 GNU Hurd

En 1990, le système GNU est presque terminé. Il ne manque plus que le composant principal : le noyau.

Le noyau d'un système d'exploitation (kernel en anglais) est un logiciel fondamental et critique qui gère les ressources de l'ordinateur et permet aux matériels et aux logiciels de communiquer entre eux. Il devait tout d'abord s'appeler Alix mais Thomas Bushnell, connu aussi sous le nom de Michael Bushnell, a préféré GNU Hurd. Ce dernier est un double acronyme récursif. Hurd signifie :

- « Hird of Unix-Replacing Daemons »,
- « Hurd of Interfaces Representing Depth ».

Le GNU Hurd (« horde de gnous ») est une série de serveurs qui fonctionnent au-dessus de Mach, un micronoyau développé à l'université Carnegie-Mellon puis à l'université de l'Utah et qui remplissent les diverses fonctions d'un noyau Unix. Le développement a pris du retard. Il fallait attendre que Mach soit publié sous forme de logiciel libre.

1.2.10 CeCILL

La licence CeCILL, créée par le CEA (Commissariat à l'énergie atomique), le CNRS (Centre national de la recherche scientifique) et l'INRIA (Institut national de recherche en informatique et en automatique), permet à des établissements publics de publier leurs logiciels sous licence libre rédigée selon le droit français.

Elle prévoit que si un logiciel sous la licence CeCILL intègre du code sous licence GPL, ou inversement, alors c'est la GPL qui s'applique.

La 1ère version a été publiée en 2004 suivi de la version 2 en mai 2005.

Le projet contient également deux autres licences :

- CeCILL-B pour la compatibilité des licences de type BSD.
- CeCILL-C pour les composants sous licence LGPL.

Sortie de la version 2.1 de CeCILL. Issue des discussions dans le cadre de l'Open Source Initiative (OSI), elle est maintenant reconnue officiellement comme une licence *Open Source* par cet organisme. La compatibilité a également été améliorée (avec les licences GNU Affero GPL et EUPL).

1.3 GNU / Linux

1.3.1 Naissance de Linux

En 1991, Linus Torvalds, étudiant finlandais à l'université de Helsinki, désire comprendre le fonctionnement de son ordinateur personnel. Les systèmes d'exploitation de l'époque n'exploitent pas correctement les capacités 32 bits et la gestion mémoire d'Intel 80386. Il fait alors son apprentissage sur le système d'exploitation Minix.

Le professeur Andrew Tanenbaum, de l'Université libre d'Amsterdam, a développé Minix un clone gratuit d'Unix qu'il utilise pour son enseignement. Les sources sont disponibles mais non libres. On peut apprécier ce dernier pour son aspect pédagogique. La simplicité est privilégiée par rapport aux performances. On peut donc être insatisfaits pour ses limitations techniques.

Du fait que l'auteur de Minix refuse d'intégrer les contributions visant à l'améliorer, Linus décide d'écrire un remplaçant de Minix. Il oriente son projet vers le développement d'un noyau aux normes POSIX.

POSIX IEEE 1003 est une série de standards pour Unix qui spécifie les interfaces utilisateurs, la ligne de commande standard, l'interface de script (Korn Shell), les services...

Le projet de Linus prend le nom de « Freax », une combinaison de « free », de « freak » et du « X » d'Unix.

Ari Lemmke met à disposition le projet sur un server ftp (ftp://ftp.funet.fi).

N'appréciant pas trop le nom de « Freax », il nomme le répertoire de stockage « Linux », une combinaison de « Linus » et du « X » d'Unix. Linux devient alors le nom du projet et l'acronyme « Linux Is Not Unix ».

1.3.2 Noyau Linux

Le 17 septembre 1991, la version 0.01 du noyau (kernel en anglais) est diffusée de façon confidentielle.

Le 5 octobre 1991, Linus Torvalds fait une annonce sur les usenets et propose une version 0.02. Puis dans le même mois, une version 0.03 incluant le binaire du Shell GNU Bash et du compilateur GCC.

Fin 1991, plusieurs versions du noyau se succèdent pour ajouter de nouvelles fonctionnalités. Il faut attendre le 5 janvier 1992 pour que Linux soit diffusé en GNU GPL. La jonction de Linux et du système GNU propose un système d'exploitation libre et complet. Cette version du système d'exploitation est alors appelé « GNU/Linux ». Par abus de langage de nos jours, nous l'appelons Linux.

La communauté de développeurs au travers d'Internet va s'accroître si bien que les forums comp.os.linux.* doivent se réorganiser plusieurs fois.

C'est en mars 1994 que la version 1.0 du noyau Linux est stable, prête pour la production. Elle fournit les services d'un Unix classique.

La version 2.0 sort en juillet 1996.

Au fil du temps, le noyau évolue pour prendre en compte de nouvelles technologies.

1.3.3 Mascotte TUX

En 1996, un concours est organisé pour trouver un logo. On préfère plutôt parler de mascotte. Alan Cox, l'un des programmeurs les plus impliqués dans le développement du noyau depuis ses débuts, suggère un manchot.

Le manchot pygmée de Larry Ewing est remarqué et remporte le concours. Il est ensuite affiné par Linus Torvalds.

James Hughes propose le nom de TUX en jouant sur l'acronyme Torvalds UniX, sachant que Tux veut aussi dire « tuxedo » qui signifie smoking en anglais Nord-Américain, petit nom qui désigne aussi les manchots.

1.4 Distributions Linux

1.4.1 Définition

Une distribution GNU / Linux rassemble les composants d'un système d'exploitation : le noyau Linux, des drivers, un Shell (interpréteur de commandes), un serveur X, un gestionnaire de bureau, un logiciel d'installation, des outils de configuration, des logiciels bureautiques, des utilitaires, etc.

De nombreuses distributions existent ayant chacune des objectifs et une philosophie propre. Certaines sont dédiées à un usage particulier : firewall (pare- feu), routeur, multimédia... D'autres sont spécifiques à un matériel.

1.4.2 Historique

MCC Interim Linux, créée par Owen Le Blanc du Manchester Computing Centre (Université de Manchester, Angleterre), est la première distribution capable de s'installer de façon autonome sur un ordinateur en février 1992. Le créateur la considère comme provisoire et non officiel.

Mi-1992, Peter MacDonald conçoit la 1ère distribution Linux SLS (Softlanding Linux System). Elle propose bien plus qu'un noyau et des utilitaires de base. Elle comprend l'implémentation libre de X11R5 (XFree86 1.0m), le serveur d'affichage du MIT (Massachusetts Institute of Technology).

En 1993, Patrick J. Volkerding crée la distribution Slackware en modifiant Linux SLS.

Ian Murdock crée le projet Debian à partir de Linux SLS dans le deuxième semestre de 1993.

De nombreuses distributions vont naître par la suite.

1.4.3 Des centaines de distributions

Pour avoir un aperçu de l'immense variété des distributions vous pouvez aller voir :

http://distrowatch.com/index.php?language=FR

- Le tableau suivant contient les principales distributions généralistes.

Nom	Site anglais	Site français
Slackware	http://www.slackware.com	http://slackfr.org/
Debian	http://www.debian.net/	http://www.debian.org/
Fedora	http://fedoraproject.org	http://www.fedora-fr.org/
Redhat	http://www.redhat.com/	http://www.redhat.fr/
Mandriva	http://www.mandriva.com/en	http://www.mandriva.com/fr
CentOS	http://www.centos.org/	http://fr.centos.org/
Suse	http://www.novell.com/linux/	http://www.novell.com/fr-fr/linux/
OpenSuse	http://www.opensuse.org/en/	http://www.opensuse.org/fr/
Ubuntu	http://www.ubuntu.com/	http://www.ubuntu-fr.org/
gentoo	http://www.gentoo.org/	http://www.gentoo.fr/
Knoppix	http://www.knoppix.net/	http://knoppix-fr.org/

- Le second tableau énumère quelques distributions spécialisées.

Nom	Site Web	Description
Backtrack	ttp://www.remote-exploit.org/	Live CD - Outils pour l'analyse réseau et le test d'intrusion.
ipcop	http://www.ipcop.org	Firewall (pare-feu)
Geexbox	http://geexbox.org/	Media center
System-RescueCD	htttp://www.sysresccd.org/	Live CD – Outils d'administration
Gparted	http://gparted.sourceforge.net/	Live CD pour gérer le partitionnement de disques.

debian

gentoo linux

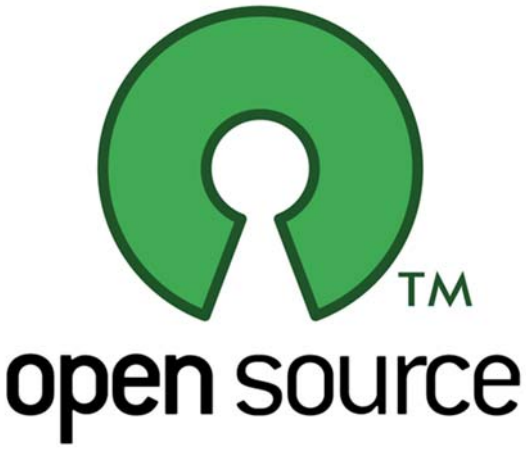
1.5 Open Source

1.5.1 Historique

En février 1998, Christine Peterson, Présidente de Foresight Nanotech Institute, a suggéré l'utilisation de la désignation Open Source pour lever l'ambiguïté du mot anglais Free Software qui signifie libre au sens de « liberté » mais surtout « gratuit ». D'une part, il faut rappeler aux utilisateurs que la conception d'un logiciel a un coût. D'autre part, il faut choisir un vocabulaire adapté au monde des entreprises.

La désignation Open Source n'a fait qu'ajouter une nouvelle ambiguïté. Elle est détournée de son objectif premier et elle est appliquée à des logiciels ne respectant que la disponibilité du code source.

Eric Steven Raymond, hacker américain, a essayé de déposer Open Source mais en vain. En 1997, il crée alors avec Bruce Perens, ancien leader du projet Debian, l'Open Source Initiative, qui délivre le label « OSI approved » aux licences qui respectent les critères définis dans l'OSD (Open Source Definition).



1.5.2 Définition de l'Open Source (the Open Source definition)

Pour qu'un logiciel soit conforme, il doit répondre aux exigences suivantes :

1. La redistribution libre
2. Le code-source
3. Les œuvres dérivés
4. L'intégrité du code source de l'auteur
5. La non-discrimination contre des personnes ou groupes
6. La non-discrimination contre des champs d'application
7. La distribution de licence
8. La licence ne doit pas être spécifique à un produit
9. La licence ne doit pas restreindre d'autres logiciels
10. La licence doit être neutre sur le plan technologique

Remarque :

Sur le site Web de l'Open Source Initiative, vous pouvez consulter l'intégralité de l'OSD : http://www.opensource.org/docs/definition.php

1.5.3 Distinction entre le Logiciel libre et l'Open Source

Des logiciels libres sous licence GPL sont Open Source, tandis que des logiciels Open Source peuvent ne pas être libres.

La différence terminologique entre Free Software et Open Source a pour but de souligner une divergence de points vus :

- Richard Stallman met en avant les mérites plutôt éthiques et philosophiques des logiciels libres.
- Eric Raymond préfère souligner la qualité des logiciels à code source ouvert d'un point de vue purement technique.

D'un point de vue économique, la désignation Open Source contribue à la création d'une nouvelle forme de marché et d'économie qui est porté par les entreprises traditionnelles de l'informatique (SSII) mais également par des sociétés de services spécialisées : les SSLL (sociétés de services en logiciels libres).

1.6 Caractéristiques d'Unix et Linux

- Systèmes multitâches / multi-utilisateurs

Plusieurs utilisateurs peuvent se connecter en même temps sur le même système d'exploitation où plusieurs programmes tournent simultanément.

- Architecture de la machine cachée à l'utilisateur

Tout élément matériel d'un système Unix ou Linux est vu comme un fichier. L'utilisateur ne connaît pas la structure matérielle du système.

- Systèmes écrits en langage C

Le noyau Unix ou Linux, comme la plupart des outils, sont écrits dans un langage évolué : le C.

- Fichiers de structure simple

Ni Unix ni Linux ne gèrent des structures de fichiers particulières.

- Grande quantité d'outils (utilitaires)

En particulier avec Linux certaines distributions sont livrées avec des milliers d'utilitaires.

- Langage de commande évolué :

Le Shell ksh (Unix) ou bash (Linux) (cf. pages suivantes)

- Compatible Posix

Linux a hérité d'un très grand nombre de fonctionnalités et de concepts d'Unix. L'un comme l'autre sont compatibles POSIX.

- Orienté réseaux

Unix et Linux offrent tous les outils et tous les mécanismes pour la mise en place de réseaux locaux, l'intégration dans des réseaux existants, la connexion à Internet.

- Multiplateformes (Linux)

Linux est disponible pour un grand nombre de plateformes techniques : Intel, Motorola 68K, PowerPC, Alpha, PalmPilot…

- Open Source (Linux)

Aujourd'hui, plusieurs milliers de programmeurs, communicant par Internet, participent au développement de Linux.

- Système multitâches / multi-utilisateurs
- Architecture de la machine cachée à l'utilisateur
- Système écrit en langage C
- Fichiers de structure simple
- Grande quantité d'outils et d'utilitaires
- Langage de commande évolué : le Shell
- Compatible Posix
- Orienté réseaux
- Multi plates-formes (Linux)
- Open Source (Linux)

1.7 Le Shell

1.7.1 Qu'est-ce qu'un Shell ?

Un Shell est un interpréteur de commandes.

Un Shell interactif est un programme lancé juste après la procédure de connexion en interface ASCII ou appelé via une icône en interface graphique.

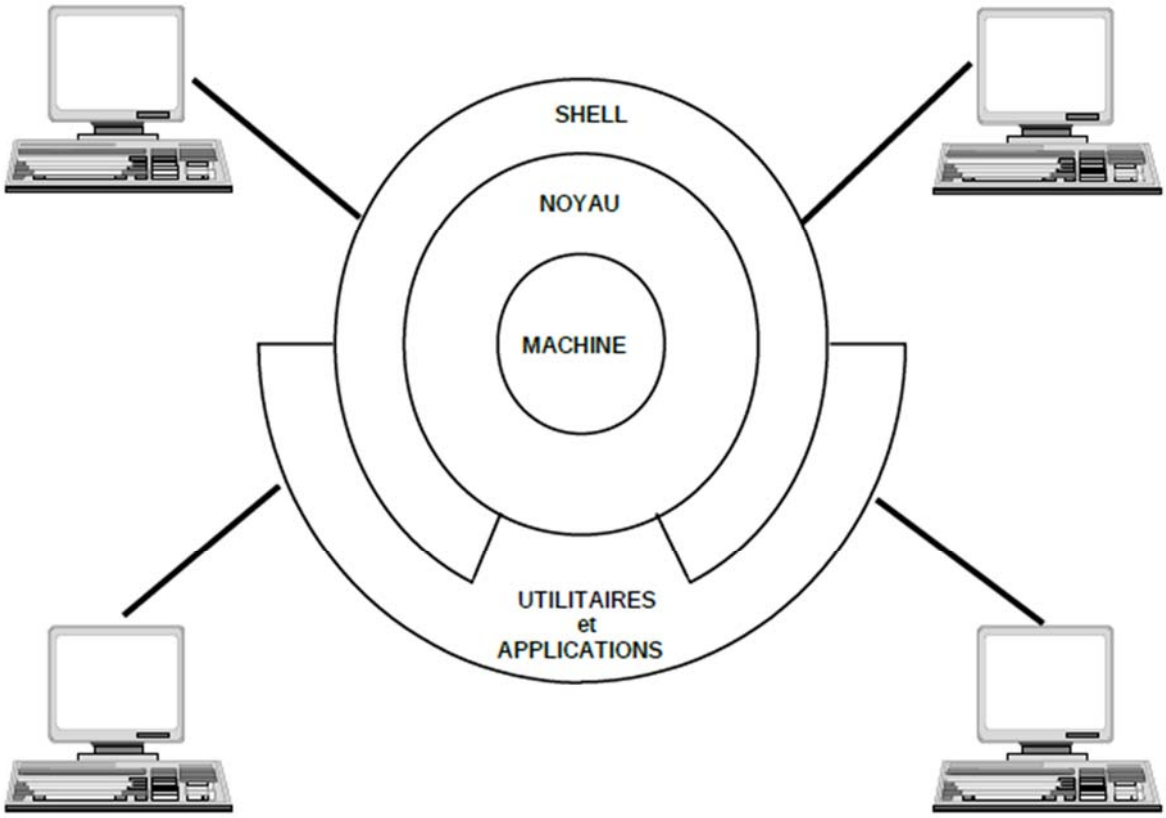
Un Shell permet de dialoguer avec le système au moyen de lignes de commandes. Il va analyser la ligne de commande puis lancer l'exécution du ou des programmes correspondants.

Un Shell possède tous les outils permettant d'en faire un langage de programmation. Les procédures interprétées par un Shell s'appellent des Shell-scripts.

1.7.2 Les principaux Shell

En fait, « Shell » est un terme générique : il existe plusieurs types de Shell, dont les syntaxes sont très proches.

sh	Bourne Shell	Shell d'origine des systèmes Unix
ksh	Korn-Shell	Le standard des systèmes Unix
csh	C Shell	Syntaxe proche du langage C
bash	Bo(u)rn(e) Again Shell	Le Shell standard de Linux et Mac OS X
tcsh	Tenex C Shell	Amélioration du *C Shell*
zsh		Shell orienté utilisateur reprenant le meilleur de bash, ksh et tcsh.
ash	Almquist Shell	Petit Shell dérivé du sh, très utilisé dans les systèmes embarqués


SHELL
NOYAU
MACHINE
UTILITAIRES
et
APPLICATIONS

2. PRISE EN MAIN

OBJECTIFS

- Se connecter à un poste de travail
- Identifier les principaux éléments du bureau (connexion graphique)
- Ouvrir une session Shell
- Saisir des commandes en respectant leur syntaxe

2.1 Connexion à Linux

2.1.1 Appellations

On se connecte le plus souvent à un poste de travail via une interface graphique, à un serveur en ligne de commandes.

Dans la littérature on trouvera souvent les appellations suivantes :

- CLI = Command Line Interface = interface lignes de commandes = interface ASCII
- GUI - Graphic User Interface = interface graphique = bureau = espace de travail graphique

2.1.2 Ouverture d'une session utilisateur

Le bouton Type de session permet de choisir l'interface graphique : KDE, Gnome… (ici l'interface par défaut est le Gnome).

Le bouton Entrer permet de valider la connexion.

Le bouton Menu, lors d'une connexion distante depuis un terminal-X ou un PC, permet de quitter la session X.

Le bouton Arrêter, sur la console graphique locale, permet de choisir l'arrêt ou le reboot du système, le redémarrage du serveur X ou le passage en mode console.

2.1.3 Connexion à Linux

Dans tous les cas de connexion il faudra entrer :

- Un nom d'utilisateur
- Un mot de passe

Nota : par défaut, chaque utilisateur peut changer son mot de passe, il est fortement déconseillé de ne pas avoir de mot de passe.

Exemple de fenêtre de connexion :

2.2 Le bureau sous Linux

Linux propose différents environnements de bureau.

Les bureaux les plus utilisés sont Gnome et KDE, mais il en existe bien d'autres tels que xfce, lxde, MATE, Cinnamome, Enlightenment, Unity, AnotherLevel, AfterStep, BlackBox, Fluxbox…

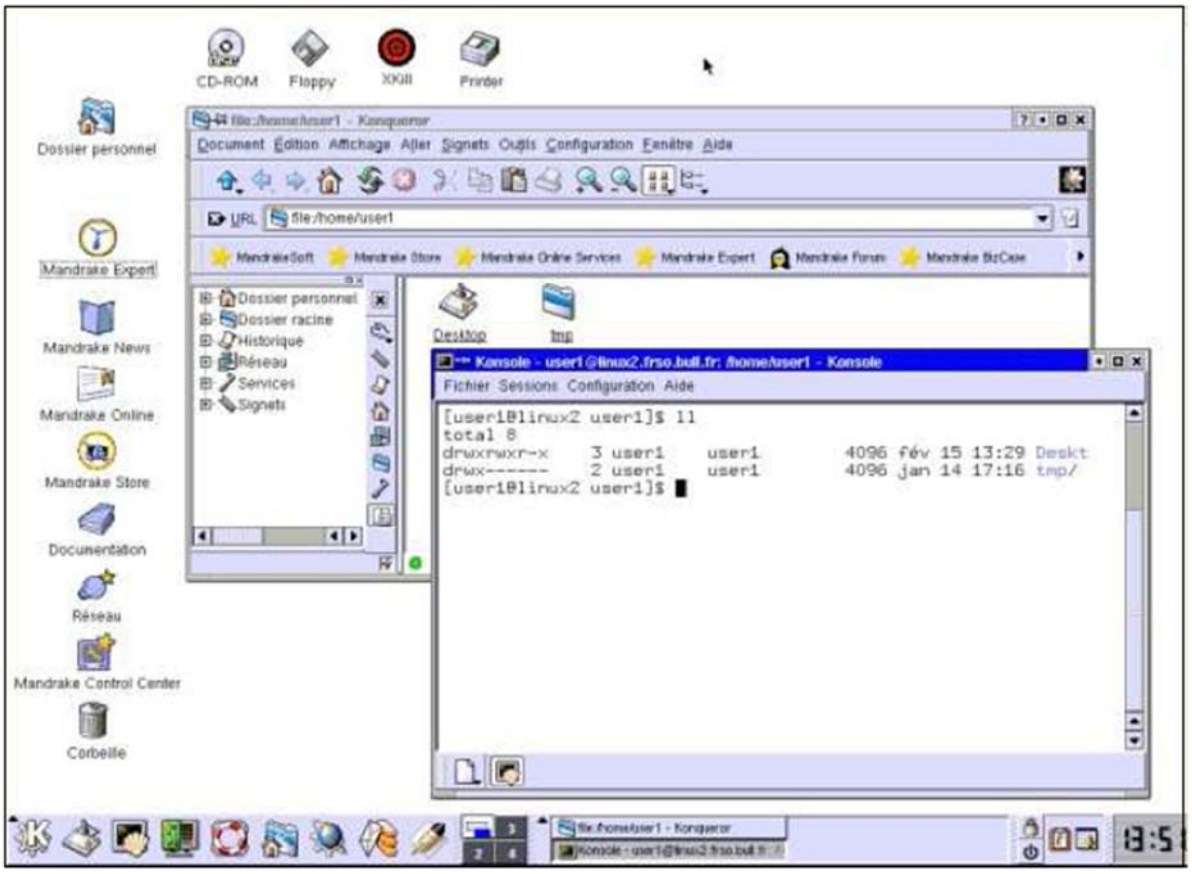
Le bureau permet une gestion graphique de l'espace de travail, il permet aussi l'utilisation des applications graphiques.

Un très grand nombre d'applications graphiques sont proposées par les diverses distributions.

On trouve habituellement des outils graphiques tels que :

- Tableau de bord
- Gestionnaires de fichiers
- Navigateurs
- Outils d'administration
- Editeurs de texte
- Manuels d'information
- Jeux
- …

Exemple de bureau KDE :



2.3 Le tableau de bord Linux

Par défaut, il y a 4 espaces de travail sur un écran, une application pouvant occuper un ou tous les espaces de travail.

Le tableau de bord contient des menus déroulants et des icônes actives. Il peut se trouver le plus souvent en bas ou en haut de l'écran.

Un clic sur le bouton Applications permet d'ouvrir un menu déroulant et de sélectionner le programme à lancer.

Un clic sur une icône active l'application associée à cette icône.

Notez bien que ces environnements de travail standard peuvent être personnalisés :

- Selon la distribution de Linux
- Par l'administrateur du système
- Par l'utilisateur

2.4 Connexion à Unix

2.4.1 Ouverture d'une session utilisateur

Le bouton OK permet de valider la connexion.

Le bouton Lancement permet

Le bouton Options permet de choisir la langue de l'interface, le type de session (bureau ou mono-fenêtre), une connexion sur console ASCII.

Le bouton Aide permet d'accéder à la description de ce panneau.

2.4.2 Connexion à Unix

Dans tous les cas de connexion il faudra entrer :

- Un nom d'utilisateur
- Un mot de passe

Nota : par défaut, chaque utilisateur doit changer son mot de passe à la première connexion, il est fortement déconseillé de ne pas avoir de mot de passe.

2.5 Le bureau sous Unix

Unix propose un environnement de bureau appelé Common Desktop Environment : CDE.

Le bureau permet une gestion graphique de l'espace de travail, il permet aussi l'utilisation des applications graphiques.

Un très grand nombre d'applications graphiques sont proposées.

On trouve habituellement des outils graphiques tels que :

- Panneau de contrôle
- Gestionnaires de fichiers
- Navigateurs
- Outils d'administration
- Editeurs de texte
- Manuels d'information
- …

Remarque :

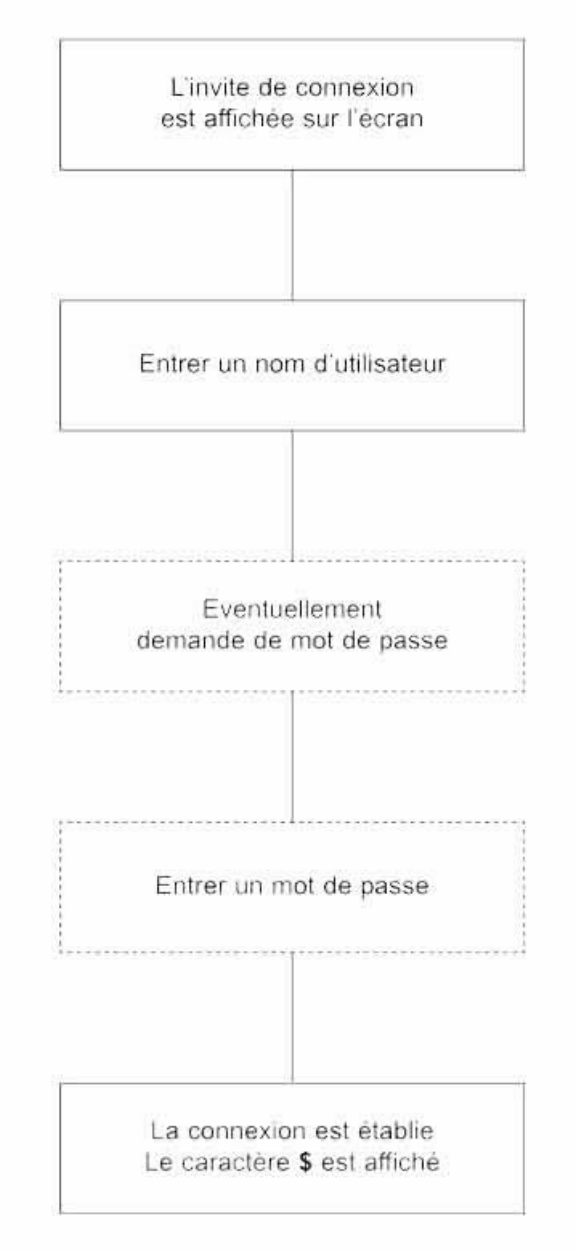
Il devient très rare de se connecter en graphique sur les serveurs Unix, la plupart du temps on y accède par un émulateur de terminal.

putty est l'émulateur de terminal le plus couramment utilisé.

2.6 Connexion ASCII

Il est fréquent de se connecter à un serveur Unix ou Linux en mode caractères (ASCII).

- Via un émulateur de terminal sur un poste de travail (p. ex. *Putty*).
- En utilisant un des pseudo-terminaux sous Linux : Ctrl+Alt+F1 à F6



2.7 Ouverture d'une fenêtre Shell

2.7.1 Le Shell

On peut accomplir énormément de tâches en utilisant l'interface graphique…mais pas tout !

En fait, avec un peu d'habitude, il est souvent bien plus efficace de travailler à partir d'une interface Shell en mode ligne de commandes qu'à partir d'une interface graphique.

Pour travailler avec le Shell ksh ou bash il est nécessaire d'ouvrir une fenêtre terminale.

2.7.2 Gestion d'une fenêtre Shell

L'ouverture d'une fenêtre Shell se fait :

- En cliquant sur l'icône correspondante ;
- En sélectionnant l'émulateur de terminal.

La terminaison du Shell se fait :

- En tapant la commande exit ;
- En tapant simultanément sur les touches <Ctrl> et <D> ;
- En fermant la fenêtre terminale (bien que cela manque de rigueur !)

2.7.3 Déconnexion

- A la terminaison du Shell si on travaille à partir d'un terminal, ou d'un pseudo terminal, en mode ligne ;
- A la clôture de la session si on travaille en graphique.

2.8 Lancement d'une commande

2.8.1 Vos premières commandes

date	Affiche la date et l'heure
ls [répertoire]	Liste le contenu d'un répertoire
ls -l [répertoitre]	Liste le contenu d'un répertoire en format long
wc [fichier]	Compte le nombre de lignes de mots et de caractères d'un fichier
wc -l [fichier]	Compte le nombre de lignes d'un fichier
wc -lc [fichier]	Compte le nombre de lignes et de caractères d'un fichier
cal [mois] [année]	Affiche le calendrier
less [fichier]	Affiche le contenu d'un fichier page à page <Entrée> affiche ligne après ligne <Barre Espace> affiche page à page <b> retour arrière d'une page <q> quitte less
more [fichier]	Affiche le contenu d'un fichier page à page même syntaxe que less
passwd	Permet de changer le mot de passe
echo "message"	Affiche "message" dans la fenêtre de Shell

```
$
$ date
lun jan 21 16:16:38 EST 2002
$
$ ls
Desktop/  azer  bin/  office52/  tmp/
$
$ ls -l
total 16
drwxr-xr-x    3 pgar     pgar      4096 jan 18 17:05 Desktop/
-rw-r--r--    1 pgar     pgar         0 jan 14 16:28 azer
drwxr-xr-x    2 pgar     pgar      4096 jan 14 16:45 bin/
drwxr-xr-x    3 pgar     pgar      4096 jan 10 10:10 office52/
drwx------    2 pgar     pgar      4096 nov  5 10:55 tmp/
$

$ wc -l /etc/passwd
     50 /etc/passwd
$
$ wc -lc /etc/passwd /etc/group
     50    2095 /etc/passwd
     61     853 /etc/group
    111    2948 total
$
$ cal 09 1752
...
$
$ cal 1970 | more
...
$
$ azer          # notez bien que azer n'est pas une commande !
bash: azer: commande introuvable
$
```

2.9 Syntaxe des commandes

2.9.1 Syntaxe de base des commandes Shell

- **`commande [-options][--mot_option][paramètre] [paramètre]...`**

La validation de la ligne de commande se fait par la touche <Entrée>.

Un ou plusieurs espaces ou tabulations sont utilisés comme séparateur.

Plusieurs commandes peuvent se suivre sur une même ligne à condition de les séparer par un ";" (point-virgule).

2.10 Informations sur les commandes

2.10.1 Manuel en ligne

Les pages de manuel, connues sous le nom de « man pages » constituent la documentation de référence pour l'utilisateur, l'administrateur et le programmeur.

Elles contiennent l'ensemble des commandes et des fonctions disponibles qui sont organisées en plusieurs sections :

- Section 1 - Commandes utilisateurs

Elle décrit les commandes de l'utilisateur telles que la manipulation de fichiers, les outils de compilation, etc.

- Section 2 - Appels système

Un appel système (system call ou syscall) est une fonction fournie par le noyau (kernel) d'un système d'exploitation. Elle est utilisée par les programmes qui s'exécutent dans l'espace utilisateur.

- Section 3 - Librairies

Toutes les fonctions des librairies (à l'exception des fonctions des appels système) y sont décrites comme la Librairie Standard C (libc), la Librairie Math (libm), la Librairie Temps Réel (librt)...

- Section 4 - Fichiers spéciaux

GNU / Linux accède aux composants matériels de la machine par l'intermédiaire de fichiers spéciaux.

- Section 5 - Format des fichiers

Elle décrit les formats de fichiers et de protocoles.

- Section 6 - Jeux

Les jeux et les petits programmes amusants y sont décrits.

- Section 7 - Divers

Elle contient divers topiques et conventions.

- Section 8 - Commandes d'administration et de maintenance

Sont décrits les commandes d'administration système, relatives aux daemons et matériels.

Des sections ne sont pas toujours présentes dans une distribution GNU / Linux :

• Section 0 - Fichiers headers des librairies du langage C

• Section 9 - Sous-programmes du noyau

• Section L - Librairies mathématiques

• Section N - Commandes TCL / TK (Tool command Language / ToolKit)

• Section x - X Window System

Normalement, une application en ligne de commande possède sa propre page de manuel. Dans le cas contraire, certains utilisateurs ressentent cette absence comme un signe de mauvaise qualité. Ainsi, quelques distributions tels que Debian écrivent ses propres pages de manuel pour les commandes qui en sont dépourvues.

```
$ man wc

WC(1)              Manuel de l'utilisateur Linux              WC(1)

NOM
       wc - Afficher le nombre d'octets, de mots et de lignes d'un fichier.

SYNOPSIS
       wc [-clw] [--bytes] [--chars] [--lines] [--words] [--help]  [--version]
       [fichier...]

DESCRIPTION
       Cette page de manuel documente la version GNU de wc ([NDT] wc = Word
       Count).

       wc compte le nombre d'octets, de mots séparés par des blancs, et de
       sauts de lignes (New Lines) dans chacun des fichiers indiqués.

       Si aucun fichier n'est fourni, ou si le nom `-' est mentionné, la lecture
       se fait depuis l'entrée standard.

       Une ligne de statistiques est affichée pour chaque fichier, précédée du
       nom du fichier s'il a été indiqué en argument.
       (...)

       Par défaut wc affiche les trois valeurs. Les options permettent de n'en
       afficher que certaines d'entre elles. Les options ne se surchargent
       pas, mais cumulent leurs effets, ainsi wc --bytes --words affiche à la
       fois le nombre d'octets et de mots.

OPTIONS
       -c, --bytes, --chars
       Afficher uniquement le nombre d'octets.

        -w, --words      Afficher uniquement le nombre de mots
        -l, --lines
       (...)
```

Pour afficher le texte page à page man utilise :

- more en ksh
- less en bash.

2.10.2 Informations sur les options d'une commande

```
$ wc --help          # en bash

Usage: wc [OPTION]... [FICHIER]...
Afficher le décompte de lignes, mots et octets de chaque FICHIER, et
le nombre total de ligne si plus d'un FICHIER est spécifié.
Sans FICHIER, ou quand FICHIER est -, lire de l'entrée standard.
  -c, --bytes, --chars   afficher le nombre d'octets
  -l, --lines            afficher le nombre de sauts de lignes
  -L, --max-line-length  afficher la longueur de la ligne la plus longue
  -w, --words            afficher le nombre de mots
```

```
$ wc -? /etc/group        # L'option help n'existe pas en ksh
wc :option non reconnue : ?
Usage :wc [-c|-m][-lw] [Fichier...]
       wc [-k] [-c [-lw]] [Fichier...]
```

2.10.3 Autres commandes utiles

```
$ whatis ls
ls                   (1)  - Afficher le contenu d'un répertoire
ls                   (1)  - list directory contents
ls                   (1p)  - list directory contents
```

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
/usr/share/man/man1p/ls.1p.gz
```

```
$ info ls
File: coreutils.info,  Node: ls invocation,  Next: dir invocation,
Up: Directory listing

10.1 `ls': List directory contents
==================================
The `ls' program lists information about files (of any type,
including
directories).  Options and file arguments can be intermixed
arbitrarily, as usual.
...
```

2.10.4 Documentation Linux au format HTML

Les pages man en HTML et en français !
http://www.linux-france.org/article/man-fr/

Encore une mine d'informations en français
http://www.linux-france.org/

Manuel de référence bash du GNU (anglais)
http://www.gnu.org/software/bash/manual/HTML_node/index.HTML

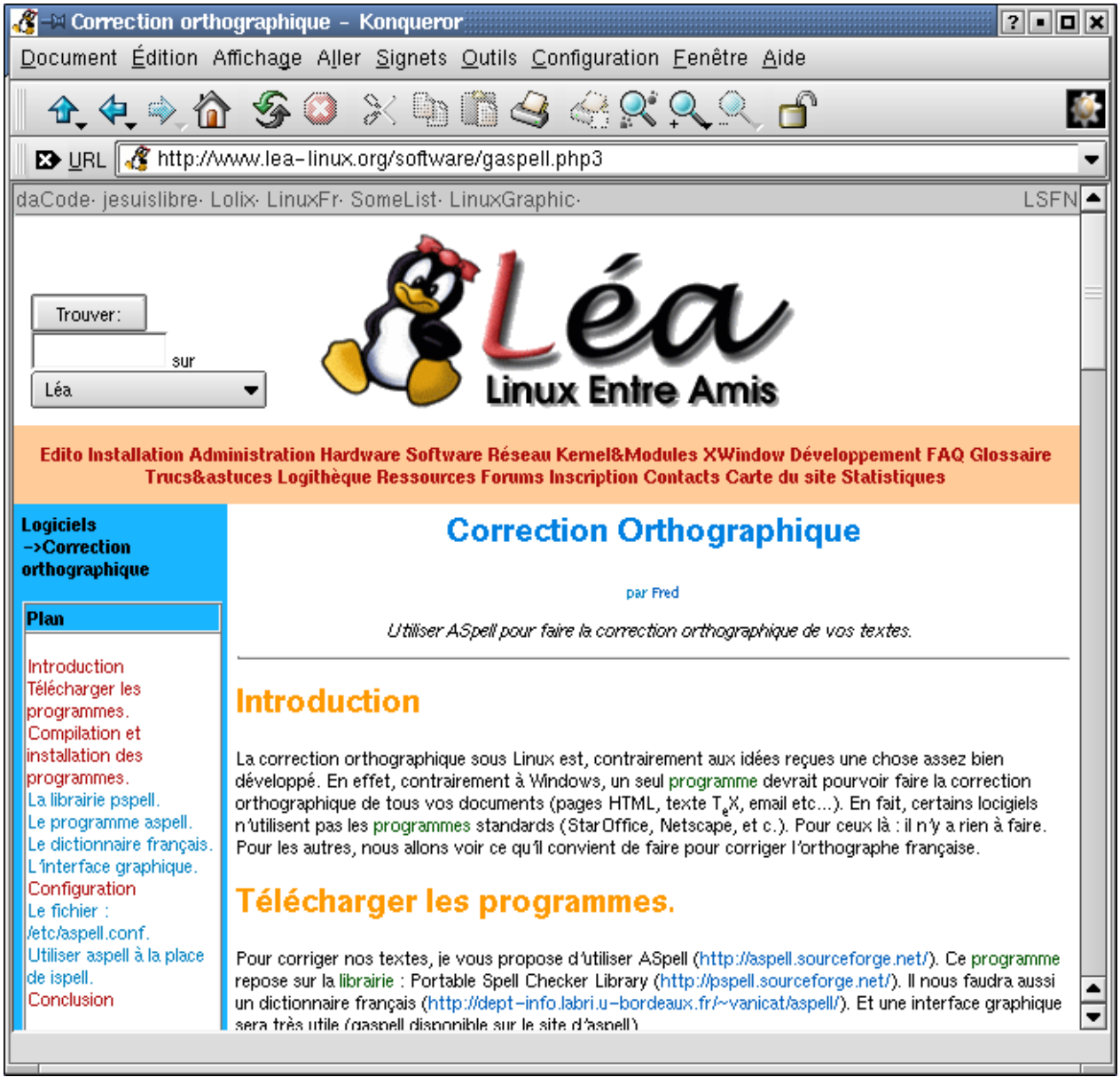
Bibliothèque de documentation Gnome
http://library.gnome.org/users/user-guide/stable/overview-desktop.HTML.fr

2.10.5 Documentation Unix au format HTML

Manuel de référence du ksh (anglais)
http://www.kornShell.com

Introduction au ksh d'AIX (anglais)
http://www.ibm.com/developerworks/aix/library/au-kornShellscripting/index.HTML

Accédez aux informations sur le Web grâce au navigateur Web Konqueror



3. MANIPULER LES FICHIERS

OBJECTIFS

- Décrire l'arborescence
- Identifier les principaux types de fichiers
- Manipuler des fichiers en utilisant les commandes Shell et des outils de l'environnement graphique
- Manipuler l'historique des commandes

3.1 Les types de fichiers

Tout est fichier pour Unix ou Linux !

3.1.1 Fichier ordinaire

Un fichier contient un ensemble d'informations.

Unix et Linux ne reconnaissent pas de structure particulière à un fichier ordinaire.

Un fichier n'est qu'une simple suite d'octets enregistrés sur disque.

Cependant, le fichier peut être organisé selon le type d'informations qu'il contient : lignes de texte ASCII, tables de base de données, trames de vidéo… Ainsi on ne pourra accéder à un fichier qu'en utilisant la commande ou l'outil correspondant au type de données qu'il contient.

3.1.2 Fichier répertoire (directory)

Un répertoire est un fichier particulier qui contient un ensemble de noms de fichiers. Chaque nom de fichier est associé à un numéro permettant l'accès physique au fichier sur disque.

3.1.3 Fichier spécial

Un fichier spécial est un nom associé à un élément matériel du système tel qu'un périphérique, un mécanisme de communication…

Ce type de fichier sera étudié au cours du stage d'administration.

Exemples de fichiers spéciaux :

- Le clavier est un fichier d'entrée
- Une fenêtre est un fichier de sortie
- L'imprimante est un fichier de sortie

3.1.4 Fichier lien symbolique

Un lien symbolique est un fichier faisant référence à un autre fichier.

3.2 Organisation des fichiers

3.2.1 Arborescence

Les fichiers Unix ou Linux sont organisés en une arborescence unique.

Le répertoire qui amorce cette arborescence est appelé répertoire racine ou root directory.

Ce répertoire racine est représenté par / (barre oblique droite ou slash).

L'ensemble des répertoires et fichiers est vu par l'utilisateur dans une seule et unique arborescence.

L'organisation physique des fichiers est cachée à l'utilisateur, pour accéder et utiliser un fichier inutile d'indiquer sur quel périphérique il se trouve.

3.2.2 Système de fichiers

Un disque physique peut être découpé en différentes partitions.

Une partition doit être formatée en système de fichiers pour pouvoir contenir des fichiers rangés dans une arborescence.

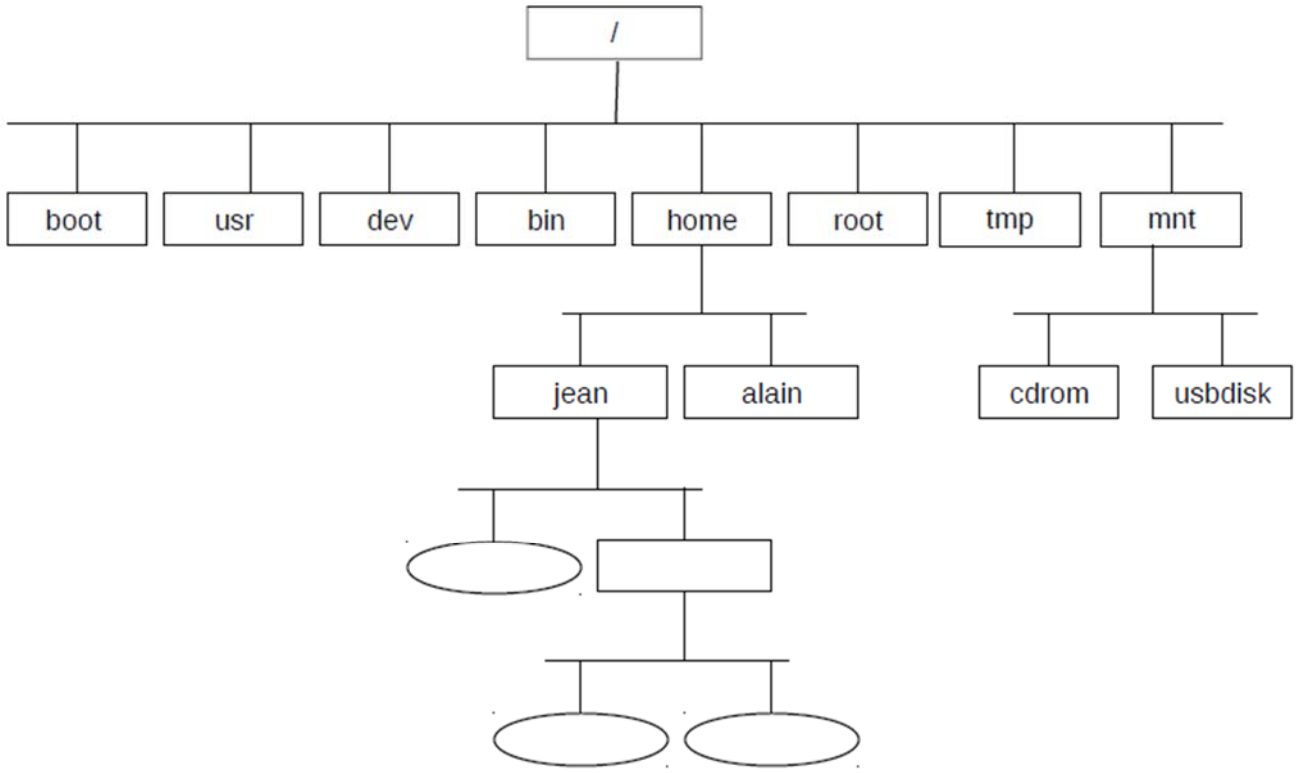
Les fichiers sont référencés dans les systèmes de fichiers par une structure appelée inode, chaque fichier est identifié par son numéro d'inode.

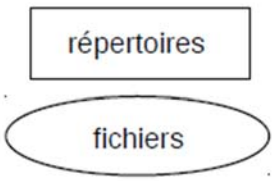
Tous les inodes du système de fichiers sont regroupés dans une table appelée i-list.

Un inode contient les caractéristiques du fichier ainsi que des pointeurs vers les blocs de données du fichier.

Nota : ces notions seront développées et détaillées au cours du stage d'administration.

Exemple d'arborescence (Linux)





3.3 Répertoires

Pour travailler avec des fichiers on peut utiliser un gestionnaire de fichier graphique ou des commandes du Shell.

Un gestionnaire de fichier graphique permet de visualiser aisément l'arborescence sous différentes formes en fonction du choix des options.

Cependant la maîtrise des commandes permet d'effectuer des opérations plus sophistiquées au moyen de lignes de commandes ou de Shell scripts.

3.3.1 Répertoire de connexion

Chaque utilisateur possède un répertoire de connexion encore appelé répertoire d'accueil ou home directory.

Par défaut ce répertoire porte le nom de l'utilisateur et se trouve dans le répertoire /home (cf. schémas pages précédentes et suivantes).

Lorsqu'un utilisateur se connecte, il est automatiquement positionné dans son répertoire de connexion.

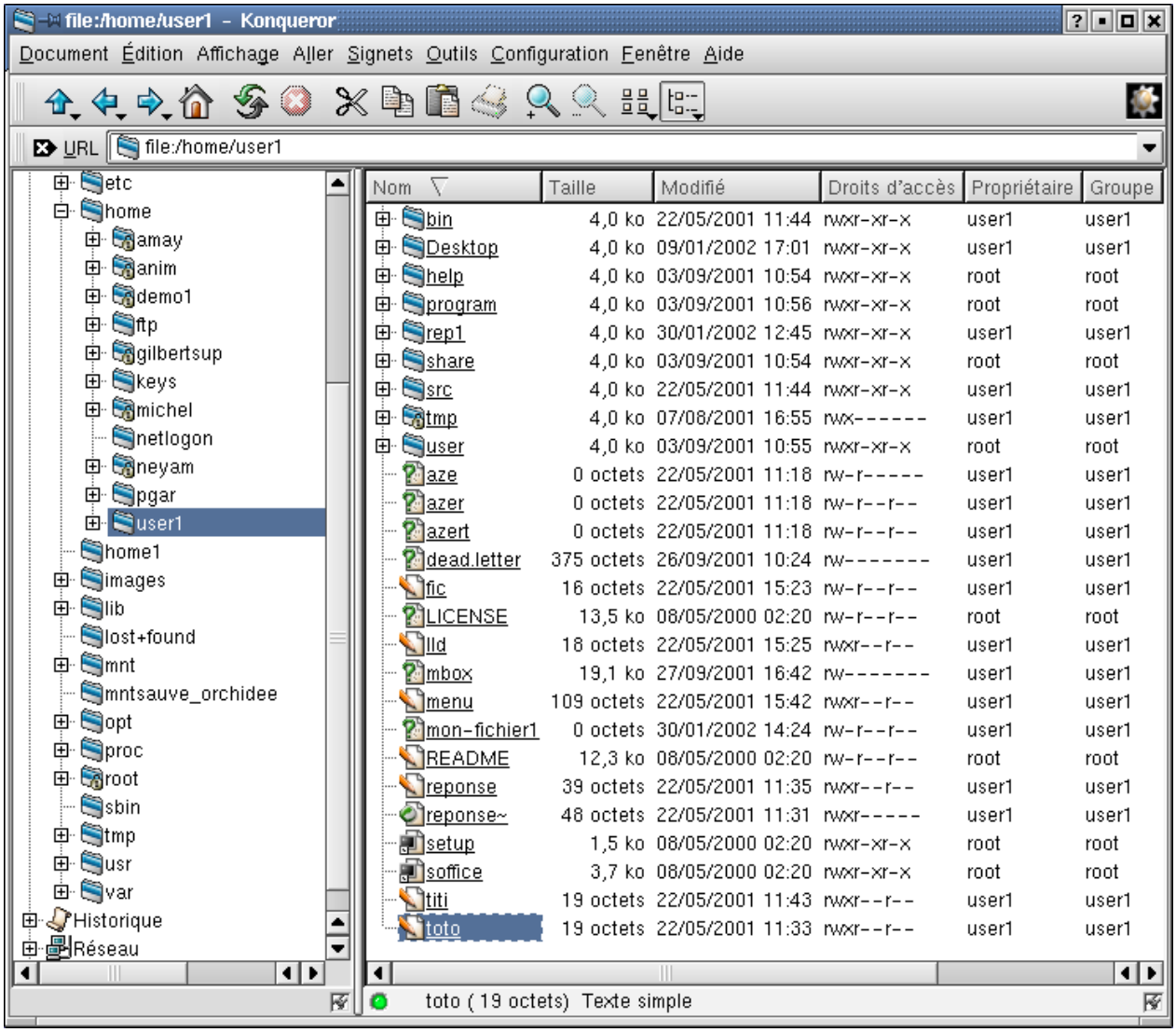
3.3.2 Répertoire de travail

Un utilisateur peut se déplacer dans l'arborescence et se positionner dans le répertoire de son choix. Ce répertoire sera alors son répertoire de travail ou répertoire courant.

La commande pwd (Print Working Directory) permet d'afficher le nom du répertoire de travail dans lequel se trouve positionné l'utilisateur à un instant donné.

```
$ pwd
/home/user1
$
```

Exemple de gestionnaire de fichiers graphique : Konqueror



3.4 Opérations sur les répertoires

3.4.1 Changer de répertoire

La commande cd (Change Directory) permet à un utilisateur de se déplacer dans l'arborescence et se positionner dans le répertoire de son choix.

Le chemin d'accès du répertoire de destination sera utilisé en paramètre de cette commande.

On peut utiliser deux types de chemins d'accès :

- Le chemin d'accès absolu

```
$ pwd
/home/user1
$ cd /home/user1/program/resource
$ pwd
/home/user1/program/resource
```

- Le chemin d'accès relatif

```
$ pwd
/home/user1
$ cd program/resource
$ pwd
/home/user1/program/resource
```

3.4.2 Répertoire père

Il est possible de remonter dans l'arborescence grâce à un "fichier" particulier nommé ".." (point point) qui représente le répertoire immédiatement supérieur.

```
$ pwd
/home/user1/program/resource
$ cd ..
$ pwd
/home/user1/program
$ cd ../..
$ pwd
/home
```

3.4.3 Positionnement automatique sur le répertoire de connexion

Pour retourner directement sous son répertoire de connexion à partir de n'importe quel endroit de l'arborescence, il suffit de taper cd.

```
$ pwd
/home/user1/program/resource
$ cd
$ pwd
/home/user1
$
```

3.4.4 Déplacements rapides

On peut aussi utiliser le caractère ~ (tilde) pour indiquer de manière implicite le chemin d'accès au répertoire de connexion.

```
$ pwd
/home/user1/program/resource
$ cd ~
$ pwd
/home/user1
$
```

Pour revenir sous le répertoire précédent :

```
$ pwd
/home/user1
$ cd -
/home/user1/program/resource
$ pwd
/home/user1/program/resource
$
```

Nota : le répertoire courant est représenté par un nom de "fichier" particulier nommé "." (point).

3.4.5 Lister le contenu d'un répertoire

La commande ls (LiSt content of a directory) permet d'afficher la liste des fichiers contenus dans le répertoire.

- ls affichage du contenu du répertoire courant
- ls /usr/bin affichage du contenu du répertoire nommé
- ls -l affichage au format long
- ls -F ajoute un caractère à chaque nom de fichier pour indiquer son type

(/=répertoire, *=exécutable...)

- ls --color utilise les couleurs pour distinguer les types de fichiers

3.4.6 Repérer le type de fichier

La commande ls –l permet d'identifier le type des fichiers par le premier caractère des lignes affichées :

d	répertoire
-	fichier ordinaire
b, c	fichier spécial
l	lien symbolique

Par défaut, Linux utilise la commande ls avec des options de couleurs particulières.

On peut avoir par exemple :

Bleu	répertoire
Noir	fichier ordinaire
Vert	fichier exécutable ou Shell-script
Jaune	fichier spécial
Cyan	lien symbolique

Attention, ceci n'est pas une règle absolue !

Les couleurs dépendent des valeurs de la variable LS_COLORS.

```
$ ls -l
total 116
drwxr-xr-x    3 user1    user1        4096 jan  9 17:01 Desktop
-rw-r--r--    1 root     root        13795 mai  8  2012 LICENSE
-rw-r--r--    1 root     root        12589 mai  8  2012 README
-rw-r-----    1 user1    user1           0 mai 22  2013 aze
-rw-r--r--    1 user1    user1           0 mai 22  2013 azer
-rw-r--r--    1 user1    user1           0 mai 22  2013 azert
drwxr-xr-x    2 user1    user1        4096 mai 22  2013 bin
-rw-------    1 user1    user1         375 sep 26 10:24 lettre
-rw-r--r--    1 user1    user1          16 mai 22  2013 fic
drwxr-xr-x    3 root     root         4096 sep  3 10:54 help
-rwxr--r--    1 user1    user1          18 mai 22  2013 lld
-rw-------    1 user1    user1       19537 sep 27 16:42 mbox
-rwxr--r--    1 user1    user1         109 mai 22  2013 menu
drwxr-xr-x    5 root     root         4096 sep  3 10:56 prog
-rwxr--r--    1 user1    user1          39 mai 22  2013 reponse
-rwxr-----    1 user1    user1          48 mai 22  2013 reponse~
lrwxrwxrwx    1 root     root           25 sep  3 10:55 setup -> prog/setup
drwxr-xr-x   15 root     root         4096 sep  3 10:54 share
drwxr-xr-x    2 user1    user1        4096 mai 22  2013 src
-rwxr--r--    1 user1    user1          19 mai 22  2013 titi
drwx------    2 user1    user1        4096 aoû  7 16:55 tmp
-rwxr--r--    1 user1    user1          19 mai 22  2013 toto
drwxr-xr-x   18 root     root         4096 sep  3 10:55 user
$
```

3.4.7 Créer un répertoire

La commande mkdir permet de créer un répertoire.

```
$ mkdir rep1
$ ls -l
...
drwxr-xr-x    2 user1    user1        4096 jan 30 12:45 rep1/
...
```

La commande mkdir crée un répertoire avec deux entrées :

- Le répertoire . lié au répertoire lui-même
- Le répertoire .. lié au répertoire père

En pareil cas, on dit que le répertoire est vide

```
$ cd rep1
$ ls -la
total 8
drwxr-xr-x    2 user1    user1        4096 jan 30 12:45 ./
drwxr-xr-x   13 user1    user1        4096 jan 30 12:45 ../
$
```

3.4.8 Supprimer un répertoire

La commande rmdir permet de supprimer un répertoire vide.

```
$ rmdir rep1
```

Si le répertoire contient des fichiers cette commande est rejetée.

Pour supprimer un répertoire non vide ainsi que tout ce qu'il contient on utilisera la commande rm.

```
$ rm -r rep1
```

3.5 Les fichiers ordinaires

3.5.1 Les types de fichiers ordinaires

Nous avons vu que Linux ne tient pas compte du contenu des fichiers, cependant leur utilisation dépendra bien évidemment de ce contenu : affichage de texte, exécution d'une commande, audition d'une musique, affichage d'une vidéo…

La commande file permet d'analyser un fichier et affiche la nature des informations contenues.

```
$ file program
program: directory
$
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), stripped
$
$ file /etc/passwd
/etc/passwd: ASCII text
$
$ file /usr/bin/passwd
/usr/bin/passwd: setuid executable, can't read '/usr/bin/passwd'
(Permission denied).
$
```

Nota : la commande file est très utile pour éviter d'afficher à l'écran des fichiers non ASCII notamment au travers de la commande cat (voir page suivante) qui ne gère pas le type de contenu de fichiers.

3.5.2 Lecture d'un fichier ordinaire ASCII

Les commandes pg en ksh, less en bash et more dans tous les cas permettent d'afficher, page à page, le contenu d'un fichier ASCII.

- more commande d'origine BSD

<Entrée> avance d'une ligne
<Espace> avance d'une page
<b> recule d'une page
<h> aide
<q > sortie de more

- less alternative à more (nombreuses améliorations)

-? ou --help informations sur la commande ou <h>
<f> avance d'une page
<b> recule d'une page
<e> avance d'une ligne
<y> recule d'une ligne
<q> sortie de less

- pg commande AIX

<Entrée> avance d'une page
<-1> recule d'une page
<+3> avance de 3 pages
<+1l> avance d'une ligne
<-1l> recule d'une ligne
<1> affiche la page no 1
<$> affiche la dernière page
<q> sortie de pg

- cat affiche et concatène le contenu de fichiers sans gérer le page à page ni le type de contenu des fichiers.

Remarque : les pages du manuel (commande man) sont, par défaut, affichées par la commande more en Unix et par less en Linux.

3.5.3 Création d'un fichier ordinaire

Sous Unix et sous Linux il n'existe pas de commande spécialisée pour créer des fichiers.

Chaque utilitaire (copies, éditeurs de textes…) crée des fichiers si besoin.

Bien que les mécanismes utilisés ci-dessous soient détaillés plus tard dans ce cours, voici des moyens simples de créer un fichier ordinaire.

Création d'un fichier vide :

```
$ >fichier1
$
$ ls –l
...
-rw-r--r--    1 user1    grp1      0  7 avril 13:09   fichier1
...
$
```

Ou bien :

```
$ touch fichier2
$
$ ls –l
...
-rw-r--r--    1 user1    grp1      0  7 avril 13:10   fichier2
...
$
```

Attention

Dans les exemples ci-dessus,

Si fichier1 existait déjà, il serait écrasé sans avertissement !!!

Si fichier2 existait déjà, il conserverait ses données.

3.5.4 Création d'un petit fichier de texte avec la commande cat

cat sans argument se met en attente d'une entrée au clavier et affiche à l'écran chaque ligne saisie au clavier.

```
$ cat
les sanglots longs des violons de l'automne
les sanglots longs des violons de l'automne
blessent mon coeur d'une langueur monotone
blessent mon coeur d'une langueur monotone
<CTL><d>
$
```

Le signe > est une redirection de flux de données (étudié plus tard dans ce cours). Les données qui devraient être affichées à l'écran sont envoyées dans fichier3.

```
$ cat >fichier3
les sanglots longs des violons de l'automne
blessent mon coeur d'une langueur monotone
<CTL><d>
```

cat avec argument affiche le contenu du ou des fichiers passés en argument.

```
$ cat fichier3
les sanglots longs des violons de l'automne
blessent mon coeur d'une langueur monotone
$ ls –l
...
-rw-r--r--    1 user1    grp1      0  7 avril 13:12   fichier3
...
$
```

3.5.5 Suppression d'un fichier ordinaire

La commande rm (ReMove) permet de supprimer un fichier ordinaire.

Si l'on ne possède pas les autorisations nécessaires, la commande sera rejetée.

rm nom_de_fichier

Options utiles :

- i : interactif, l'utilisateur doit confirmer ou non la suppression du fichier option souvent par défaut en Linux.
- f : forcé, ne tient pas compte des protections du fichier (dangereux !)
- r : récursif, supprime récursivement le contenu des répertoires.

```
$ rm –i fichier2
rm: détruire `fichier2'? o
$
```

Astuce

Si l'option -i est installée par défaut, cela signifie que l'administrateur a voulu imposer une protection en créant un alias. Il est toutefois possible de neutraliser cela en procédant comme suit :

```
$ \rm fichier1
```

3.6 Les noms de fichiers

3.6.1 Caractéristiques

Un fichier est caractérisé par un nom, unique dans le répertoire où il se situe. Un même nom peut être utilisé dans des répertoires différents, il s'agira donc de fichiers différents.

Le nom d'un fichier peut comporter jusqu'à 255 caractères.

Le chemin d'accès à un fichier peut comporter jusqu'à 1024 caractères.

Il n'y a en théorie aucune restriction quant à l'orthographe des noms de fichiers, mais voici quelques conseils à suivre si vous voulez éviter des surprises parfois fort incommodantes :

- Unix et Linux font la distinction entre minuscules et majuscules, respectez la casse (par habitude on travaille le plus souvent en minuscules).
- La plupart des caractères autres que lettres et chiffres peuvent avoir une interprétation particulière, évitez-les, sauf. (point) et _ (souligné).
- Un fichier dont le nom commence par «.» (point) est appelé « fichier caché ».
  Pour visualiser les fichiers cachés utilisez : `ls -a`

3.6.2 Caractères génériques

Le Shell permet l'emploi de caractères génériques qui sont utilisés pour construire des noms de fichiers à partir des fichiers sous le répertoire courant ou le répertoire cité.

Ces caractères génériques peuvent être utilisés dans toutes les commandes où des noms de fichiers sont cités (ls, less, cp, mv, rm…)

Signification des caractères génériques en fonction des noms des fichiers présents sous le répertoire courant ou le répertoire cité :

*	:	remplace n'importe quelle chaîne de caractères y compris la chaîne vide.
?	:	remplace un caractère quelconque.
[ ]	:	remplace un des caractères contenus entre les crochets.
[!]	:	remplace un des caractères non contenu entre les crochets.

Attention :

ls -F fait suivre les noms de fichiers exécutables par une "*", les répertoires par "/", les liens symboliques par "@". Ces caractères ne font pas partie du nom du fichier.

Exemples :

```
$ ls -l /usr/bin/c*
lrwxrwxrwx    1 root   root            3 mai 10  2001 /usr/bin/c++ -> g++
-rwxr-xr-x    1 root   root        57308 avr  8  2001 /usr/bin/c++filt
-rwxr-xr-x    1 root   root         6124 mar 30  2001 /usr/bin/c2050
-rwxr-xr-x    1 root   root         7644 mar 30  2001 /usr/bin/c2070
-rwxr-xr-x    2 root   root        36353 avr  4  2001 /usr/bin/c2ph
-rwxr-xr-x    1 root   root        10184 mar 30  2001 /usr/bin/cZ11
-rwxr-xr-x    1 root   root         3353 mar 22  2001 /usr/bin/c_rehash
-rwxr-xr-x    1 root   root        10620 avr  2  2001 /usr/bin/cal
-rwxr-xr-x    1 root   root        13336 avr  3  2001 /usr/bin/calendar
-rwxr-xr-x    1 root   root        20156 mar 30  2001 /usr/bin/calibrate_p
-rwxr-xr-x    1 root   root         6700 avr 12  2001 /usr/bin/cancel
...

$ ls -l /usr/bin/c?
lrwxrwxrwx    1 root   root            3 mai 10  2001 /usr/bin/cc -> gcc
-rwxr-xr-x    1 root   root        74172 mar  7  2001 /usr/bin/ci
-rwxr-xr-x    1 root   root        70588 mar  7  2001 /usr/bin/co
-rwxr-xr-x    1 uucp   uucp       139708 mar 13  2001 /usr/bin/cu
$
```

3.7 Historique des commandes sous Unix

Le ksh comporte une interface utilisateur qui permet d'améliorer le confort d'utilisation du Shell.

Les commandes saisies au clavier sont enregistrées dans un fichier historique dont le nom par défaut est $HOME/.sh_history

HOME est une variable standard qui contient le chemin d'accès absolu au répertoire de connexion ; son contenu est obtenu par :

$ echo $HOME

Le fichier .sh_history est mis à jour, en ajout, en fin de fichier, avec chaque ligne de commande saisie au clavier. Il contient par défaut 126 lignes, lorsqu'il est plein, c'est la ligne la plus ancienne qui tombe.

La commande history affiche les 16 dernières lignes saisies au clavier et examinées par le Shell.

La commande r exécute des commandes du fichier historique et met à jour ce fichier :

r	exécute la dernière commande de l'historique ;
r n	exécute la nième commande de l'historique ;
r -2	exécute l'avant dernière commande de l'historique ;
r ls	exécute la dernière commande commençant par ls.

Plus généralement, l'utilisation de l'historique des commandes se fait au travers du clavier. L'appel à l'historique des commandes se fait par la touche <Echap>.

Pour un bon fonctionnement du rappel des commandes, il faut, au préalable, avoir défini un éditeur de texte pour l'historique des commandes. La variable d'environnement EDITOR, non renseignée par défaut, est prévue pour contenir le chemin d'accès au code d'un éditeur de texte. L'utilisateur doit renseigner cette variable dans un Shell script appelé $HOME/.profile qui est exécuté à chaque connexion.

- EDITOR=/usr/bin/vi
- export EDITOR

Si ces deux lignes sont incluses dans $HOME/.profile, les commandes suivantes sont valides pour l'historique des commandes.

<Echap>	Entre dans l'éditeur de texte
k ou -	Remnote de ligne en ligne dans le fichier historique
j ou +	Descend de ligne en ligne dans le fichier historique
l ou espace	Avance d'un caractère
h ou backspace	Recule d'un caractère
x	Efface le caractère courant
a … <Echap>	Insère du texte à droite du curseur
A … <Echap>	Insère du texte en fin de ligne
i … <Echap>	Insère du texte à gauche du curseur
I … <Echap>	Insère du texte en début de ligne
/hist	Recherche dans l'historique la ligne contenant la chaîne "hist"
r	Remplace le caractère sous le curseur

3.8 Historique des commandes sous Linux

Le bash comporte une interface utilisateur disposant de petits "raccourcis" très astucieux et utiles, qui vous épargneront de taper sur quelques touches.

Le fichier historique sous Linux s'appelle $HOME/.bash_history, son fonctionnement est similaire à celui de $HOME/.sh_history.

3.8.1 Rappel d'une commande

<tab>	Taper une fois la touche <Tab> permet de compléter automatiquement un nom de fichier s'il est unique. Si lors du premier appui sur <Tab>, le nom n'a pas été complété, un deuxième appui vous donne la liste de toutes les possibilités.
<Flèche haut> (ou Ctrl-P)	La flèche vers le haut permet de monter dans l'historique des commandes.
<Flèche bas> (ou Ctrl-N)	La flèche vers le bas permet de descendre dans l'historique des commandes.
<Flèche droite> <Flèche gauche>	Les flèches droite et gauche permettent de se déplacer sur la ligne de commande, toute modification est possible par frappe directe.
<Entrée>	L'appui sur la touche <Entrée> valide la ligne de commande et la relance.
<Maj><Fl. haut> <Maj><Fl. bas>	Permet de dérouler le contenu du terminal texte vers le haut ou le bas, ligne à ligne.
!!	Rappelle la dernière ligne de commande, ce raccourci peut être utilisé à l'intérieur d'une autre ligne de commande.

!*	Rappelle les arguments de la dernière ligne de commande, ce raccourci peut être utilisé à l'intérieur d'une autre ligne de commande.
	$ mkdir rep1
	$ cd !*

4. PROTECTION DES FICHIERS

OBJECTIFS

- Interpréter les droits d'accès attribués aux fichiers
- Modifier les droits d'accès aux fichiers

4.1 Notion d'utilisateur

4.1.1 Définition d'un utilisateur

Afin de pouvoir ouvrir une session de travail Unix ou Linux, il faut être reconnu du système et donc posséder un nom d'utilisateur.

Un utilisateur est défini par :

- Un nom (8 caractères maximum)
- Un numéro (UID = User IDentifier)
- Un répertoire de connexion
- Un numéro de groupe principal (GID = Group IDentifier)

Un compte utilisateur est protégé par un mot de passe.

Un utilisateur fait partie d'un groupe et est propriétaire de ses fichiers.

Les utilisateurs sont déclarés par l'administrateur du système et font l'objet d'une ligne dans le fichier /etc/passwd.

Une ligne de /etc/passwd est composée de sept champs séparés par des " : "

- Nom utilisateur
- Champ mot de passe (*)
- uid
- gid
- Commentaire (*)
- Répertoire d'accueil
- Programme à lancer à la connexion (*)

(*) Ces champs peuvent être vides.

(*) Si le champ programme à lancer à la connexion est vide, il est lancé par un Shell de Bourne (Unix seulement).

Plusieurs connexions peuvent être établies simultanément avec le même nom d'utilisateur.

Un utilisateur n'est pas lié à un terminal précis.

4.1.2 Le super utilisateur

Il existe un utilisateur particulier ayant pour nom root.

C'est un utilisateur privilégié :

- Il a accès à tous les fichiers du système
- Tant en mode ligne qu'avec l'utilisation d'une interface graphique root est seul habilité à exécuter certaines commandes.

C'est en tant que "super utilisateur" que l'administrateur gère le système.

L'invite de "root" est # en standard.

Il est fermement recommandé de mettre un mot de passe au super utilisateur car pour "root", toutes les protections sur les fichiers sont levées.

4.2 Notion de groupe

4.2.1 Définition d'un groupe

Un groupe rassemble un certain nombre d'utilisateurs ayant en commun des autorisations d'accès à des fichiers, utilisateurs travaillant, par exemple, avec la même application.

Un groupe est décrit par une ligne du fichier /etc/group (4 champs séparés par des « : ») :

- Nom du groupe
- Champ mot de passe
- Numéro de groupe (GID = Group IDentifier)
- Une liste d'utilisateurs invités dans le groupe

Un utilisateur appartient toujours à un groupe dit « groupe principal ».

Par défaut, Linux crée un groupe par utilisateur, le groupe porte le même nom que l'utilisateur.

Tous les utilisateurs d'un groupe sont égaux quant aux autorisations d'accès aux fichiers dont le groupe est propriétaire.

Il est possible pour un utilisateur d'être invité dans d'autres groupes afin d'avoir accès aux fichiers appartenant à ces groupes.

Identification complète d'un utilisateur

Après votre connexion au système, il est bon de vous assurer de votre identité et des groupes auxquels vous appartenez.

```
$ id
uid=501(user1) gid=508(user1) groupes=22(cdrom),43(usb),503(xgrp)
```

Dans cet exemple :

- L'utilisateur se nomme *user1* et possède le numéro d'utilisateur 501,
- Son groupe principal est le groupe *user1* dont le numéro est 508,
- Il est aussi invité des groupes *cdrom*, *usb* et *xgrp*.

4.3 Les droits d'accès

4.3.1 Généralités

Pour l'accès à un fichier, les utilisateurs sont classés en trois ensembles :

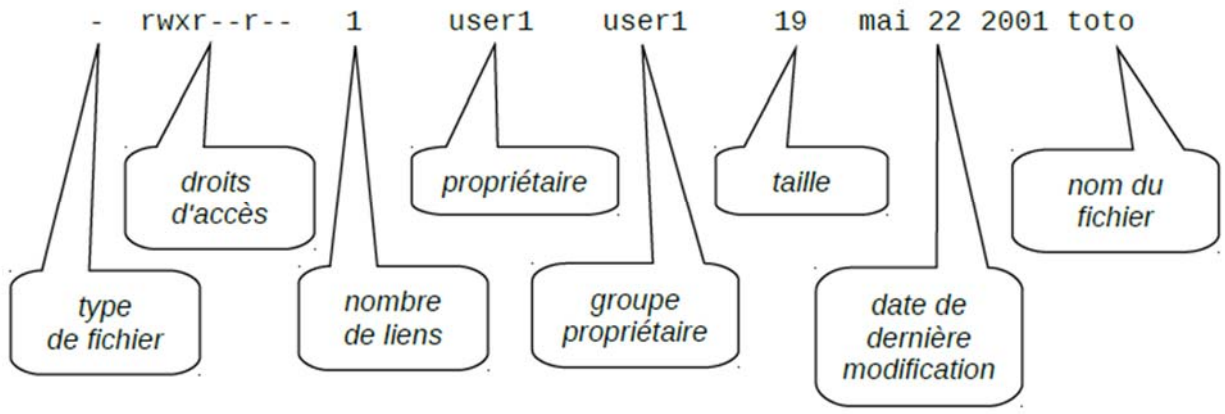
- Le propriétaire du fichier (en général l'utilisateur qui l'a créé),
- Les utilisateurs du groupe propriétaire du fichier,
- Les utilisateurs des autres groupes.

r w x suid	r w x sgid	r w x sticky-bit
Propriétaire (user - u)	Groupe (group - g)	Autres (other - o)

A chaque ensemble est associé trois types d'autorisations dont la signification dépend du type du fichier

Fichiers ordinaires	Répertoires
• autorisation de lire (r) • autorisation d'écrire (w) • autorisation d'exécuter (x)	• autorisation de lister (r) • autorisation de créer ou supprimer des fichiers (w) • autorisation de passer dans le répertoire (x)

```
$ ls -l
total 120
drwxr-xr-x    3 user1    user1        4096 jan  9 17:01 Desktop
-rw-r--r--    1 root     root        13795 mai  8  2000 LICENSE
-rw-r--r--    1 root     root        12589 mai  8  2000 README
-rw-r--r--    1 user1    user1           0 mai 22  2001 azer
drwxr-xr-x    2 user1    user1        4096 mai 22  2001 bin
-rw-------    1 user1    user1         375 sep 26 10:24 dead.letter
-rw-r--r--    1 user1    user1          16 mai 22  2001 fic
drwxr-xr-x    3 root     root         4096 sep  3 10:54 help
-rwxr--r--    1 user1    user1          18 mai 22  2001 lld
-rw-------    1 user1    user1       19537 sep 27 16:42 mbox
-rwxr--r--    1 user1    user1         109 mai 22  2001 menu
-rw-r--r--    1 user1    user1           0 jan 30 14:24 mon-fichier1
drwxr-xr-x    5 root     root         4096 sep  3 10:56 program
drwxr-xr-x    2 user1    user1        4096 jan 30 12:45 rep1
-rwxr--r--    1 user1    user1          39 mai 22  2001 reponse
drwxr-xr-x   15 root     root         4096 sep  3 10:54 share
drwxr-xr-x    2 user1    user1        4096 mai 22  2001 src
-rwxr--r--    1 user1    user1          19 mai 22  2001 titi
drwx------    2 user1    user1        4096 aoû  7 16:55 tmp
-rwxr--r--    1 user1    user1          19 mai 22  2001 toto
drwxr-xr-x   18 root     root         4096 sep  3 10:55 user
```



Champ droits d'accès : Lorsque la lettre apparaît, le droit est positionné, lorsque la lettre est remplacée par un "-", le droit est enlevé.

NB : En Linux, il est fréquent que propriétaire et groupe portent le même nom, ce qui n'est pas le cas en Unix. Les deux cas de figures seront présentés dans ce support.

4.3.2 Droits d'accès supplémentaires

suid : set user-id

sgid : set group-id

Ces deux autorisations supplémentaires permettent à un utilisateur de prendre, le temps de l'exécution d'un programme, l'identité du propriétaire ou du groupe propriétaire du programme.

sticky-bit :

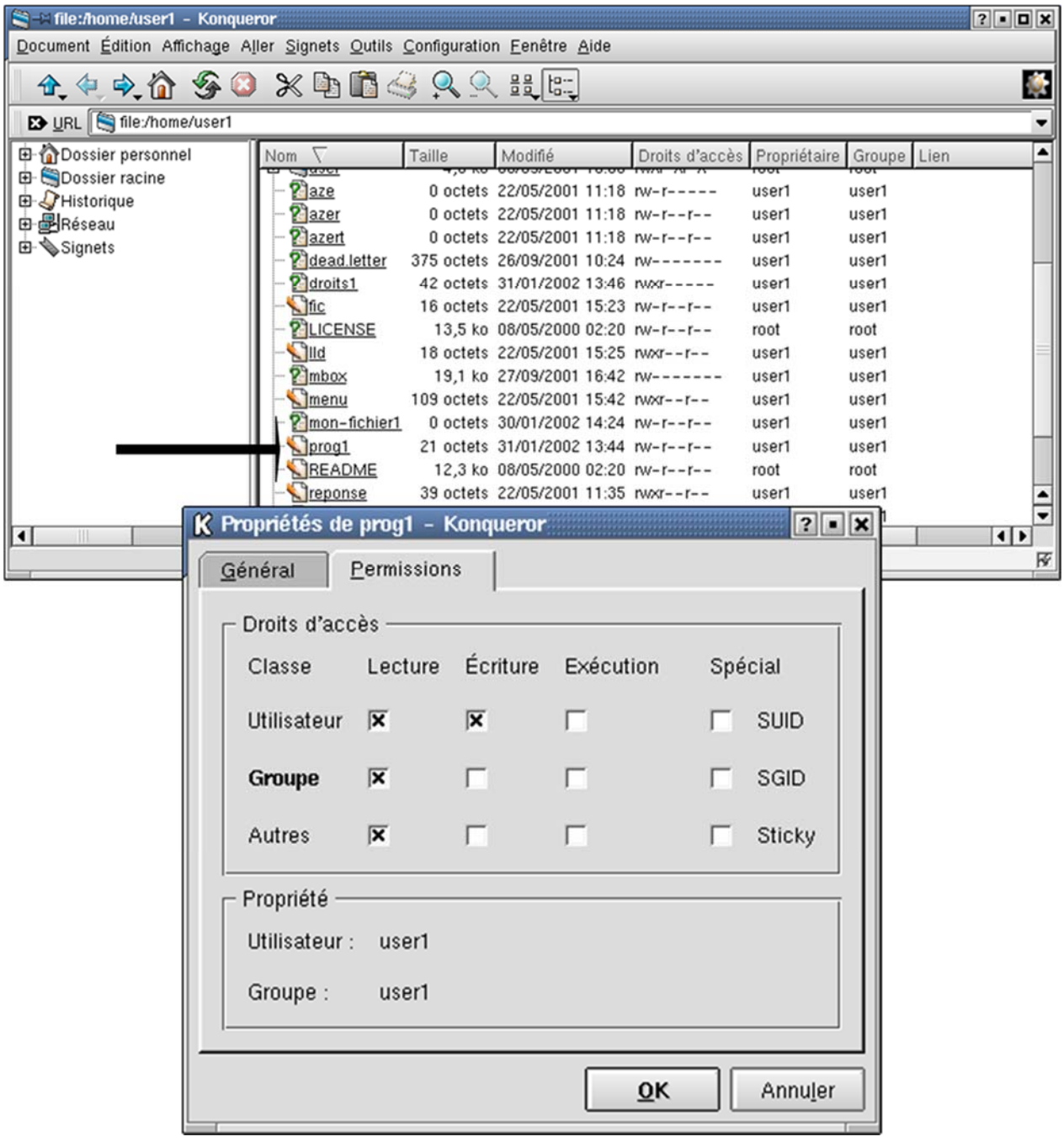
Sert uniquement pour les répertoires. Il indique que seuls le propriétaire du répertoire, et le propriétaire d'un fichier qui s'y trouve ont le droit de supprimer ce fichier (typiquement utilisé pour les répertoires comme /tmp ayant une autorisation d'écriture générale).

Exemple de la commande passwd possédant le suid permettant à tout utilisateur de modifier son mot de passe dans des fichiers qu'il n'a pas le droit de modifier directement :

```
$ ls -l /usr/bin/passwd
-r-s--x--x    1 root     root        13044 mai 21  2001 /usr/bin/passwd

$ passwd
Changer le mot de passe pour user1
Entrer le mot de passe :
Nouveau mot de passe:
Nouveau mot de passe:
...

$ ls -l /etc/passwd
-rw-r--r--    1 root     root         2677 jui  5 16:56 /etc/passwd

$ ls -l /etc/shadow
-r--------    1 root     root         2130 jui  5 17:04 /etc/shadow
```

4.4 Modification des droits d'accès

Avec l'outil graphique, il suffit de sélectionner un fichier, de faire un clic droit pour afficher ses propriétés, de cocher les droits souhaités et de valider.



La commande chmod permet de modifier les droits d'accès à un fichier. Seuls root ou le propriétaire d'un fichier peuvent en changer les droits.

Il y a deux syntaxes possibles :

4.4.1 En octal

➢ **chmod valeur_octale fichier**

Valeurs octales :

r	w	x	r	-	x	r	-	-
1	1	1	1	0	1	1	0	0
7			5			4		

Exemple :

```
$ chmod 754 prog1
```

4.4.2 En symbolique

➢ **chmod [qui] opération [permission] fichier**

qui :	u	user
	g	group
	o	others
	a	all
opération :	+	ajoute des droits
	-	enlève des droits
	=	positionne des droits, enlève les précédents
permission :	r	read, lecture
	w	write, écriture
	x	exécution
	s	positionne suid ou sgid (fonction de [qui])
	t	positionne sticky-bit

```
$ chmod u+x,g-w,o=r prog1
```

```
$ id
uid=501(user1) gid=501(user1) groupes=501(user1), 503(xgrp)
$
$ cat >droits1
echo "fichier d'essai des droits d'accès"
$
$ ls -l droits1
-rw-r--r--    1 user1    user1          42 jan 31 13:46 droits1
$
$ chmod 754 droits1
$
$ ls -l droits1
-rwxr-xr--    1 user1    user1          42 jan 31 13:46 droits1
$
$ chmod g+w,o=x droits1
$
$ ls -l droits1
-rwxrwx--x    1 user1    user1          42 jan 31 13:46 droits1
$
$ cat droits1
echo "fichier d'essai des droits d'accès"
$
$ droits1
fichier d'essai des droits d'accès
$
$ chmod 000 droits1
$ ls -l droits1
----------    1 user1    user1          42 jan 31 13:46 droits1
$
$ cat droits1
cat: droits1: Permission non accordée
$
$ droits1
bash: ./droits1: Permission non accordée
$
$ echo fin >droits1
bash: droits1: Permission non accordée
$
```

4.5 Changement de groupe et de propriétaire

La commande chgrp permet de changer le groupe propriétaire d'un fichier.

La commande chown permet de changer soit simplement le propriétaire soit à la fois le propriétaire et le groupe d'un fichier.

- `chgrp groupe fichier`
- `chown utilisateur fichier`
- `chown utilisateur:groupe fichier`

Les commandes `chgrp` et `chown` s'appliquent à tous les types de fichiers (fichiers ordinaires, répertoires, fichiers spéciaux…).

Seul root a le droit de changer le propriétaire d'un fichier.

Un utilisateur peut changer le groupe propriétaire d'un fichier à la condition d'appartenir à ce nouveau groupe ou d'en être invité (cf. la commande id).

« Root » peut attribuer tout fichier à n'importe quel utilisateur ou groupe.

```
$ ls -l azer
-rw-r--r--    1 user1    user1           0 mai 22  2011 azer
$ chgrp user2 azer
chgrp: Vous n'êtes pas membre du groupe `user2'.: Opération non
permise
$ id
uid=501(user1) gid=501(user1) groupes=501(user1),503(xgrp)
$ chgrp xgrp azer
$ ls -l azer
-rw-r--r--    1 user1    xgrp            0 mai 22  2011 azer
$
$
$ su                      # on passe administrateur root
Password:*****
#
# chown user2 azer
# exit                    # on retourne à la connexion user1
$
$ ls -l azer
-rw-r--r--    1 user2    xgrp            0 mai 22  2011 azer
$
```

4.6 Changement d'identité

La commande su, substitute user, elle permet de changer d'identité.

su nom_utilisateur

```
$ whoami
user1
$ su user2
Mot de passe :
xxxxx
$ whoami
user2
$ exit
$
$ whoami
user1
$
$ su root ou bien $ su
Mot de passe :
xxxxx
# whoami
root
#
```

5. OPERATIONS SUR LES FICHIERS

OBJECTIFS

- Copier, déplacer, rechercher des fichiers
- Créer des liens
- Rechercher des fichiers dans l'arborescence

5.1 Environnement graphique

5.1.1 Exemples d'outils graphiques développés sous Linux

L'environnement graphique de Linux est riche de différents outils de gestion de fichiers.

Chaque distribution, chaque nouvelle version propose des outils toujours plus sophistiqués offrant toute une palette de configurations possibles.

Les plus récents couplent navigation locale et manipulation des fichiers avec la navigation Web sur l'Internet.

Pour information citons quelques gestionnaires de fichiers :

- Konqueror (KDE)
- Nautilus (Gnome)
- Linux Commander
- Krusader
- BusyBox
- ...

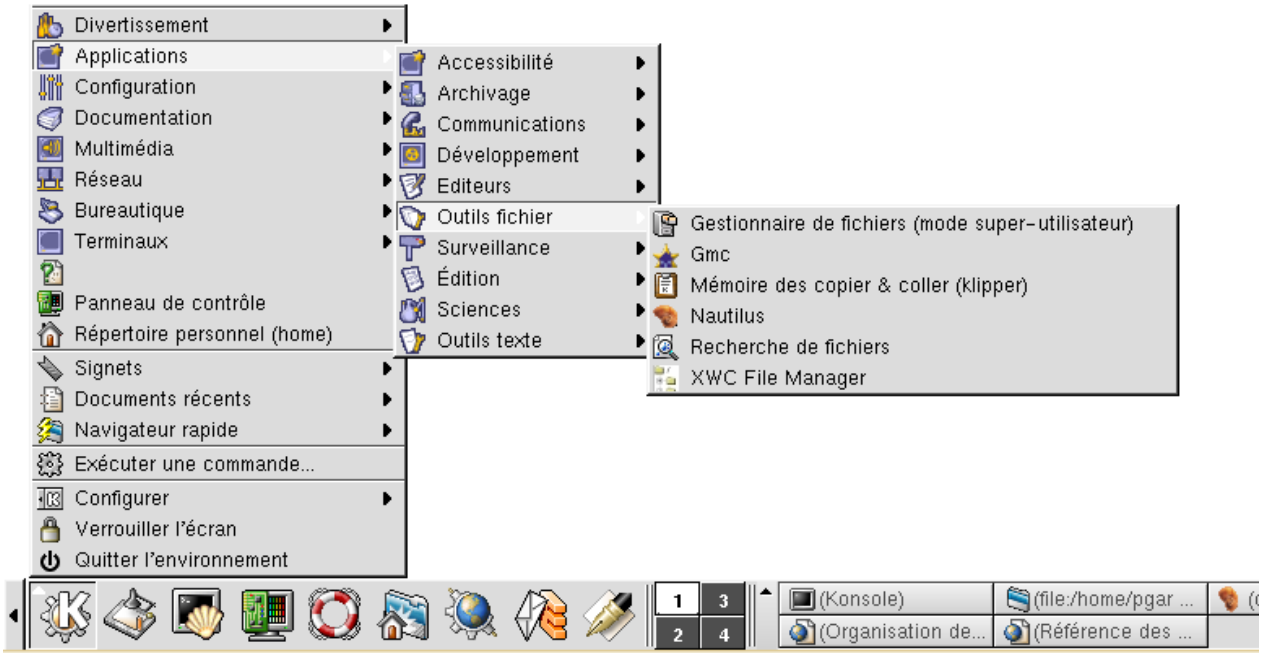
Conseils :

Inutile de vouloir tout connaître et tout maîtriser, essayez les différents gestionnaires de fichiers et adoptez celui qui vous convient le mieux.

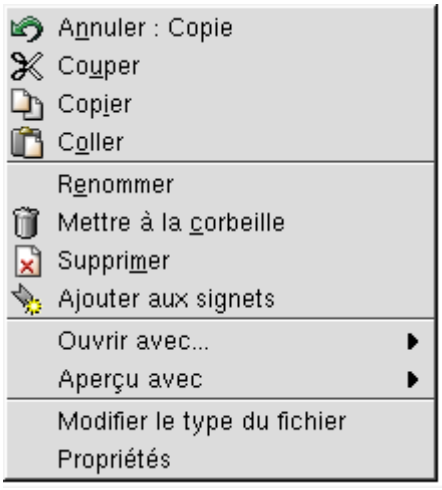
En utilisant les aides en ligne, vous ne devriez pas rencontrer de difficultés pour vous familiariser avec les possibilités et particularités de chaque outil.

Remarques :

- Dans le reste du chapitre vous allez apprendre plus particulièrement les commandes permettant de manipuler les fichiers.
- Sous Unix, CDE (Common Desktop Environment) offre des outils similaires à celui qui est présenté à la page suivante.



Avec Konqueror on peut ; soit utiliser les outils de la barre de tâche, soit ouvrir les menus déroulant ou encore cliquer droit sur un fichier pour avoir accès aux opérations sur les fichiers :



5.2 Copier des fichiers

La commande `cp` (copy) permet de copier des fichiers ordinaires.

Copie d'un seul fichier ordinaire :

- **`cp source destination`**

Copie d'un ensemble de fichiers ordinaires vers un répertoire :

- **`cp fichier1 fichier2 … répertoire`**

Emploi des caractères génériques :

- **`cp * répertoire`**

Options utiles :

- **`-i`** demande confirmation avant d'écraser des fichiers existants
- **`-R`** copie récursivement les répertoires

```
$ pwd
/home/user2

$ ls -l
total 16
drwxr-xr-x    3 user2    base        4096 fév  1 09:40 Desktop/
drwxr-xr-x    2 user2    base        4096 fév  1 09:43 copie/
drwxr-xr-x    2 user2    base        4096 fév  1 09:44 source/
drwx------    2 user2    base        4096 jan 31 09:20 tmp/

$ cd source

$ ls -l
total 12
-rw-r--r--    1 user2    base          14 fév  1 09:44 original1
-rw-r--r--    1 user2    base          19 fév  1 09:44 original2
-rw-r--r--    1 user2    base          23 fév  1 09:44 original3

$ ls -l ../copie
total 0

$ cp original1 ../copie/original1.cp

$ ls -l ../copie
total 4
-rw-r--r--    1 user2    base          14 fév  1 09:46 original1.cp

$ cp * ../copie

$ cd ../copie
$ ls -l
total 16
-rw-r--r--    1 user2    base          14 fév  1 10:16 original1
-rw-r--r--    1 user2    base          14 fév  1 09:46 original1.cp
-rw-r--r--    1 user2    base          19 fév  1 10:16 original2
-rw-r--r--    1 user2    base          23 fév  1 10:16 original3
```

5.3 Déplacer, renommer un fichier

La commande `mv` (move) permet de déplacer, renommer un fichier.

Déplacement d'un fichier :

- **`mv source destination`**

Déplacement de plusieurs fichiers vers un répertoire :

- **`mv fichier1 fichier2 … répertoire`**

Emploi des caractères génériques :

- **`mv * répertoire`**

Déplacement d'un répertoire :

- **`mv répertoire1 répertoire2`**

Pour renommer un fichier :

- **`mv ancien-nom nouveau-nom`**

```
$ pwd
/home/user2/copie

$ cd ../source

$ pwd
/home/user2/source

$ ls -l
total 12
-rw-r--r--    1 user2    base          14 fév  1 09:44 original1
-rw-r--r--    1 user2    base          19 fév  1 09:44 original2
-rw-r--r--    1 user2    base          23 fév  1 09:44 original3

$ mv original1 original1.bis

$ ls -l
total 12
-rw-r--r--    1 user2    base          14 fév  1 09:44 original1.bis
-rw-r--r--    1 user2    base          19 fév  1 09:44 original2
-rw-r--r--    1 user2    base          23 fév  1 09:44 original3

$ cd .. ; pwd
/home/user2

$ ls -l
total 16
drwxr-xr-x    3 user2    base        4096 fév  1 09:40 Desktop/
drwxr-xr-x    2 user2    base        4096 fév  1 10:16 copie/
drwxr-xr-x    2 user2    base        4096 fév  1 10:38 source/
drwx------    2 user2    base        4096 jan 31 09:20 tmp/

$ mv source originaux

$ ls -l
total 16
drwxr-xr-x    3 user2    base        4096 fév  1 09:40 Desktop/
drwxr-xr-x    2 user2    base        4096 fév  1 10:16 copie/
drwxr-xr-x    2 user2    base        4096 fév  1 10:38 originaux/
drwx------    2 user2    base        4096 jan 31 09:20 tmp/

$ ls -l originaux
total 12
-rw-r--r--    1 user2    base          14 fév  1 09:44 original1.bis
-rw-r--r--    1 user2    base          19 fév  1 09:44 original2
-rw-r--r--    1 user2    base          23 fév  1 09:44 original3
```

5.4 Créer un fichier - Notion d'inode

Les systèmes Unix et Linux mettent des systèmes de fichiers à la disposition des utilisateurs.

Les systèmes de fichiers sont des structures logicielles construites sur une portion de disque et capables de recevoir des fichiers rangés dans une arborescence.

Sommairement, un système de fichiers est composé :

- D'un super bloc qui décrit le système de fichiers
- D'une i_list composée d'inodes numérotées
- D'une zone bloc de données

Les inodes (contraction de « index » et « node » en français : *nœud d'index*) sont des structures de données contenant des informations concernant les fichiers stockés dans les systèmes de fichiers. A chaque fichier correspond un numéro d'inode (*i-num*).

Lorsqu'un fichier est créé, quelle qu'en soit la manière, le système lui attribue un inode libre. Cet inode est déclaré occupé et attribué au fichier. Cette opération entraîne une modification du super bloc.

Une entrée est créée dans le répertoire contenant le fichier. Cette entrée comporte le nom du fichier et son numéro d'inode.

Le système recherche également un (ou plusieurs) bloc(s) de données libre(s) pour enregistrer les données du fichier. Ce(s) bloc(s) est(sont) déclaré(s) occupé(s) et attribué(s) au fichier. Cela suppose une modification du super bloc et un enregistrement du(des) numéros de bloc(s) de données dans l'inode préalablement attribué au fichier.

L'inode du fichier recevra toutes les caractéristiques du fichier : type du fichier, droits sur le fichier, propriétaire, groupe, taille, date de création, date de dernier accès, date de dernière modification des données... Ainsi que le(les) numéro(s) de blocs de données contenant les données du fichier. Ce qui signifie qu'atteignant l'inode, on atteint les données du fichier. L'inode est atteint par consultation de répertoire père.

La commande ls -i permet de connaître le numéro d'inode d'un fichier.

5.5 Créer un lien sur un fichier

La création de lien sur un fichier consiste à donner plusieurs chemins d'accès aux données d'un fichier. Autrement dit, à créer plusieurs chemins d'accès atteignant le même inode.

La commande `ln` (link) permet de créer des liens.

Création d'un lien physique :

- **`ln source destination`**

Le compteur de liens est une des caractéristiques du fichier, il est contenu dans l'inode.

La commande **`ls-l`** affiche la valeur du compteur de liens d'un fichier.

La commande `rm` permet de supprimer un lien. Elle décrémente le compteur de lien.

Pour supprimer un fichier, il est nécessaire de supprimer tous les liens à ce fichier.

Création d'un lien symbolique avec l'option **`-s`** :

- **`ln -s fichier_source fichier_lien`**

`fichier_source` Existe, a son propre inode.

`fichier_lien` Est créé par la commande, son inode est différent de celui de **`fichier_source`**

Contient le chemin d'accès à **`fichier_source.`**

```
$ pwd
/home/user2

$ ls -l
total 16
drwxr-xr-x    2 user2    base        4096 fév  1 10:49 appli/
drwxr-xr-x    2 user2    base        4096 fév  1 10:16 copie/
drwx------    2 user2    base        4096 jan 31 09:20 tmp/

$ ls -li appli
total 12
 277710 -rw-r--r--    1 user2    base          14 fév  1 09:44 prog1
 277711 -rw-r--r--    1 user2    base          19 fév  1 09:44 prog2
 277712 -rw-r--r--    1 user2    base          23 fév  1 09:44 prog3

$ ln appli/prog1 prog1.lien

$ ls -li
total 20
 277701 drwxr-xr-x    2 user2    base        4096 fév  1 10:49 appli/
 357709 drwxr-xr-x    2 user2    base        4096 fév  1 10:16 copie/
 277710 -rw-r--r--    2 user2    base          14 fév  1 09:44 prog1.lien
  37294 drwx------    2 user2    base        4096 jan 31 09:20 tmp/

$ rm appli/prog1

$ ls -li
total 20
 277701 drwxr-xr-x    2 user2    base        4096 fév  1 10:58 appli/
 357709 drwxr-xr-x    2 user2    base        4096 fév  1 10:16 copie/
 277710 -rw-r--r--    1 user2    base          14 fév  1 09:44 prog1.lien
  37294 drwx------    2 user2    base        4096 jan 31 09:20 tmp/
$ cd

$ ln -s /etc/passwd pass

$ ls -li /etc/passwd pass
 611698 -r-xr-xr-x    1 root     security    1948 jan  6 13:48 /etc/passwd
 823800 lrwxrwxrwx    1 user2    base          11 fév  1 11:58 pass-->
/etc/passwd
```

5.6 Rechercher un fichier

5.6.1 La commande `find`

La commande `find` permet de rechercher des fichiers dans l'arborescence.

➢ **`find [chemins] [critères] [actions]`**

- **`Chemins`** Liste de répertoires à utiliser comme point de départ de la recherche, les éventuels sous-répertoires sont également parcourus.
- **`Critères`** Conditions que les fichiers doivent satisfaire pour être retenus.
- **`Actions`** Série d'actions à effectuer sur les fichiers qui répondent aux critères donnés.

Le répertoire de recherche par défaut est le répertoire de travail (.)

```
$ cd /

$ find . -name core -print
find: ./mnt/cdrom: Erreur d'entrée/sortie
find: ./mnt/floppy: Erreur d'entrée/sortie
./dev/core
./etc/X11/xdm/core
find: ./etc/skel/tmp: Permission non accordée
find: ./etc/default: Permission non accordée
find: ./etc/gconf/gconf.xml.defaults: Permission non accordée
find: ./etc/Bastille: Permission non accordée
find: ./etc/uucp: Permission non accordée
find: ./etc/Webmin: Permission non accordée
find: ./etc/linuxconf/archive: Permission non accordée
<CTL><c>

$ find . -name core -print 2>/dev/null
./dev/core
./etc/X11/xdm/core
./home/ftp/core
./var/log/httpd/core
./proc/sys/net/core
```

Nota : **`-print`** est la primitive d'action par défaut, il n'est pas nécessaire de l'écrire.

5.6.2 Exemples de critères de recherche

Recherche par nom de fichier :

```
find / -name fichier
```

Recherche par numéro de fichier (i-num) :

```
find /home -inum numéro
```

Recherche par nom de propriétaire de fichier :

```
find /tmp -user toto
```

Recherche des fichiers modifiés il y a exactement 8 jours :

```
find /home -mtime 8
```

Recherche des fichiers modifiés à une date plus récente que le fichier cité dans la commande :

```
find /home/user1 -newer ficref
```

Recherche des fichiers selon leur type (f=ordinaire, d=répertoire) :

```
find /usr/bin -type d
```

Recherche selon les droits des fichiers :

```
find /home -perm 764
```

Recherche des fichiers à 3 liens :

```
find /usr/bin -links 3
```

Astuce : pour éviter l'affichage des messages d'erreurs, possibles lorsqu'on n'est pas root, on peut rediriger ces erreurs dans un "fichier poubelle". (Notion de redirection étudiée plus loin).

- `find /home -name fichier 2>/dev/null`

5.6.3 find et les caractères génériques

L'utilisation des caractères génériques dans le champs critère peut s'avérer délicate.

Le résultat de la commande find dépend de la présence ou non de fichiers répondant à l'expression générique dans le répertoire de travail.

Pour éviter les résultats non significatifs il est vivement conseillé de :

Toujours protéger l'expression générique par des " " (guillemets).

```
$ ls -l
total 12
drwxr-xr-x. 2 pat gp1 4096 1 déc. 2013 rep1
drwxr-xr-x. 2 pat gp1 4096 20 août 13:02 rep2
drwxr-xr-x. 2 pat gp1 4096 20 août 13:03 rep3
$
$ ls -R
.:
rep1 rep2 rep3
./rep1:
./rep2:
fic1
./rep3:
fic1 fic2 fic3
$
$ find . -name f*
./rep2/fic1
./rep3/fic1
./rep3/fic2
./rep3/fic3
$
$ cd rep1
$ find . -name f*
$
$ cd ../rep2
$ find . -name f*
./fic1
$
$ cd ../rep3
$ find . -name f*
find: les chemins doivent précéder l'expression : fic2
Utilisation : find [-H] [-L] [-P] [-Olevel] [-D
help|tree|search|stat|rates|opt|exec] [chemin...] [expression]
$
$ find . -name "f*"
./fic1
./fic2
./fic3
$
```

5.6.4 La commande whereis

La commande **whereis** recherche les fichiers exécutables, les sources et les pages de manuel des fichiers spécifiés dans une liste de répertoires standard tels que :

```
/bin
/usr/bin
/etc
/usr/etc
/sbin
/usr/sbin
...
```

```
$ whereis rm
rm: /bin/rm /usr/share/man/man1/rm.1.bz2
```

6. REDIRECTIONS PIPE

OBJECTIFS

- Rediriger les entrées et sorties des commandes
- Enchaîner des commandes au moyen de tubes ou « pipes »

6.1 Entrées / sorties standard

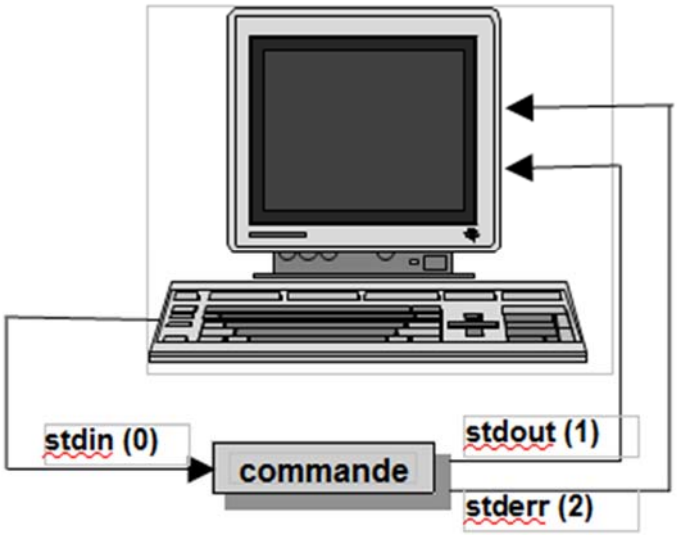
Chaque fichier ouvert par une commande se voit affecté un descripteur de fichier (file descriptor).

Lorsqu'une commande est lancée, trois fichiers standard sont ouverts par défaut :

- Le fichier « Entrée standard » (stdin) : le clavier
- Le fichier « Sortie standard » (stdout) : l'écran
- Le fichier « Sortie d'erreur » (stderr) : l'écran

Les « file descriptor » associés à ces fichiers sont :

- Entrée standard : fd = 0
- Sortie standard : fd = 1
- Sortie d'erreur : fd = 2



6.2 Les redirections

6.2.1 Redirection des données en sortie

La sortie standard (écran) peut être redirigée à l'aide du symbole ">".

La sortie des données se fera dans un fichier qui sera créé ou dont le contenu sera écrasé.

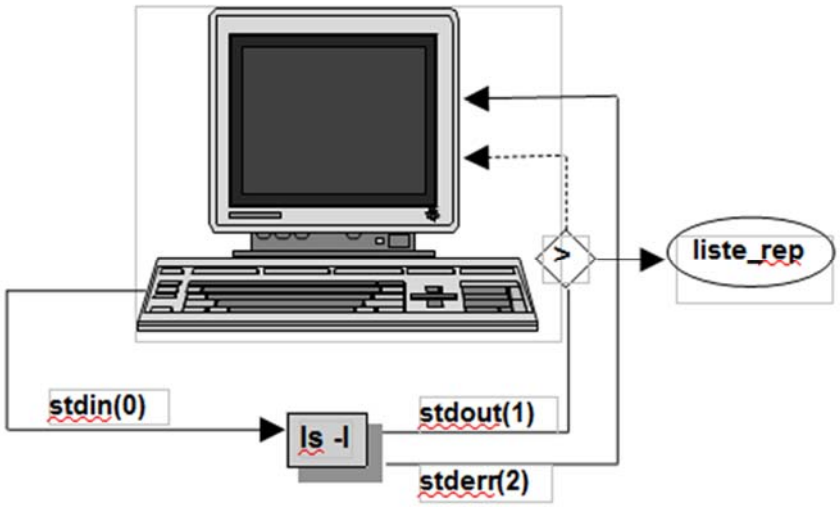
Syntaxe :

- **commande > fichier**

```
$
$ ls -l > liste_rep
$
```

Pour ajouter les données en fin d'un fichier existant

- **commande >>fichier**



6.2.2 Redirection des erreurs en sortie

La sortie d'erreur (écran) peut être redirigée à l'aide du symbole "2>".

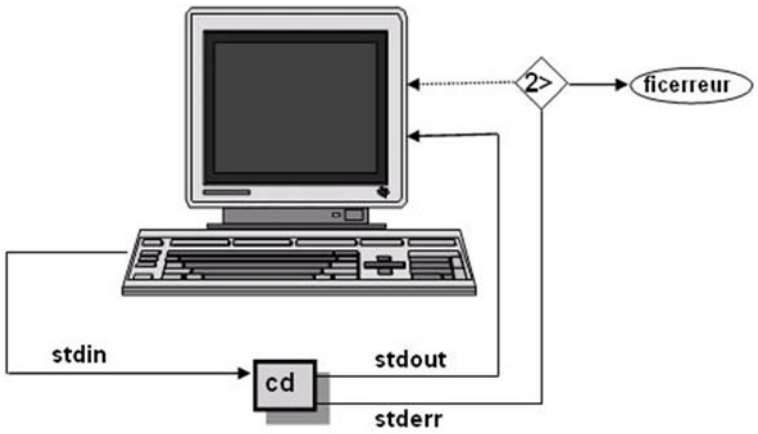
La sortie des messages d'erreurs se fera dans un fichier qui sera créé ou dont le contenu sera écrasé (comme pour la sortie standard).

Syntaxe :

- **commande 2> fichier**

```
$
$ ls bidon  2>fic_err
$
```

Pour ajouter les données en fin d'un fichier existant



- commande 2>> fichier

Il existe un pseudo-périphérique appelé « fichier poubelle » permettant de se débarrasser des sorties inutiles.

```
$ find / -name prog1 2>/dev/null
...
```

6.2.3 Redirection des erreurs dans le même fichier que les données

- **commande > fichier 2>&1**

```
$
$ find / -name prog1 >ficimage 2>&1
...
```

Le fichier ficimage sera l'image exacte de ce qui aurait été obtenu à l'écran sans redirection du stdout et du stderr.

Cette écriture permet la construction d'un fichier avec l'arrivée simultanée de deux flux.

6.2.4 Caractères génériques après une redirection

Les caractères génériques ne sont valables que dans la partie commande et non dans la partie des redirections, sauf s'il en découle la génération d'un seul nom de fichier.

Exemple :

```
$
$ echo bonjour > *.c
$
```

Remarque :

`ksh` Crée le fichier « *.c » sauf si présence sous le répertoire courant d'un seul fichier dont le nom se termine par « .c »

`bash` Si aucun fichier du répertoire courant ne porte un nom se terminant par « .c », création du fichier « *.c »

Si un seul fichier du répertoire courant porte un nom se terminant par « .c », écrasement de ce fichier.

Si plusieurs fichiers du répertoire courant portent un nom se terminant par « .c », message d'erreur.

6.2.5 Redirection des données en entrée

L'entrée standard (clavier) peut être redirigée à l'aide du symbole "<".

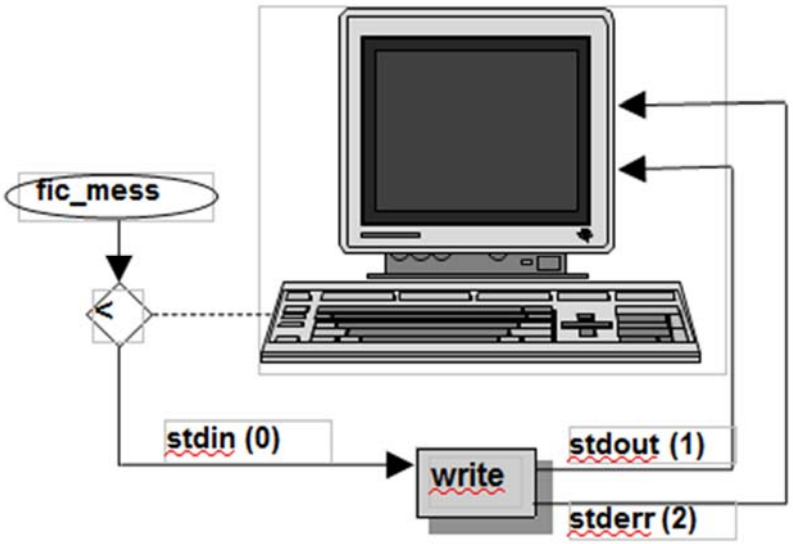
L'entrée des données se fera à partir d'un fichier.

Syntaxe :

- **commande < fichier**

```
$ whoami
user2
$ write user1  # 3 lignes suivantes envoyées à l'écran de user2
coucou user1
Tu reçois à l'écran chaque ligne que je saisi
Sur mon stdin
<Ctl><D>
$ write user1 < fic_mess      # Contenu de fic_mess
                              # envoyé à l'écran de user2
$
```

La commande write permet d'envoyer un message à un autre utilisateur connecté en Shell sur le même système.



6.3 Communication entre processus

Possibilité de faire communiquer deux commandes : l'une utilisant sa sortie standard et l'autre son entrée standard.

Utilisation du symbole | (barre verticale : <Alt Gr> <6>).

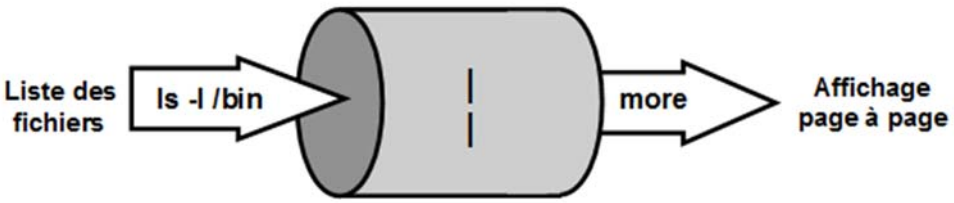
Appelé tube de communication ou plus communément pipe.

Syntaxe :

- **commande1 | commande2**

Avantage : inutile de créer un fichier temporaire (gain de temps et d'espace disque).

Toute sortie produite sur le stdout de commande1 est immédiatement transféré sur le stdin de commande2.




```
$ ls -l /bin | more
total 8080
-rwxr-xr-x. 1 root root  27776 12 mars  12:12 arch
lrwxrwxrwx. 1 root root      4 23 avril  2013 awk -> gawk
-rwxr-xr-x. 1 root root  26264 12 mars  12:12 basename
-rwxr-xr-x. 1 root root 938832 18 juil.  2013 bash
-rwxr-xr-x. 1 root root  48568 12 mars  12:12 cat
...
--Plus--
```

Le nombre de pipe dans un pipeline n'est pas limité.

7. EDITEUR DE TEXTE

OBJECTIFS

- Identifier les différents éditeurs de texte à votre disposition
- Utiliser les fonctions de base de l'éditeur de texte vi
- Utiliser un éditeur de texte graphique de votre choix

7.1 Les éditeurs de texte

Ne confondons pas éditeur de texte avec un traitement de texte.

Un **éditeur de texte** est un programme qui permet de modifier des fichiers de texte brut, sans mise en forme (gras, italique, souligné…).

Sous Windows, on dispose d'un éditeur de texte très basique : le Bloc-Notes.

Sous Unix, bien souvent, seul vi est disponible par défaut.

Sous Linux, on a le choix entre Nano, Vim, Emacs et bien d'autres, sachant qu'au moins un de ceux-là est installé par défaut sur la plupart des distributions.

Un **traitement de texte** est fait pour rédiger des documents mis en forme.

Sous Windows, Word est le plus célèbre traitement de texte ; sous Linux, on possède l'équivalent : Libre Office Writer.

Ces programmes ne peuvent être utilisés qu'en mode graphique.

7.1.1 Quand a-t-on besoin d'un éditeur de texte ?

Chaque fois que l'on doit éditer un fichier de texte brut.

Sous Windows, on voit des fichiers de texte brut au format `.txt`.

Sous Linux, l'extension importe peu (on peut trouver des fichiers en texte brut sans extension).

Les éditeurs de texte sont parfaits pour les programmeurs en particulier : ils permettent d'éditer des fichiers .c, .rb, .py, .sh... en fonction du langage de programmation.

On utilisera aussi un éditeur de texte chaque fois qu'il faudra modifier des fichiers de configuration.

7.1.2 Graphique ou ASCII

Que ce soit sous Unix ou sous Linux, les éditeurs de textes, ASCII ou graphiques, sont nombreux. L'essentiel est d'en connaître un pour être en mesure de saisir et corriger ses programmes.

Pour les débutants, les outils graphiques sont d'un abord plus simple.

Cependant, en cas de problème système, ces outils ne sont pas accessibles et il devient indispensable de maîtriser un éditeur non graphique pour pouvoir réparer.

Bien souvent l'accès aux serveurs sous Unix ou Linux se fait au moyen d'un émulateur de terminal (putty) sur une station de travail, dans ce cas l'accès au système est exclusivement en ligne de commandes, la maîtrise d'un éditeur non graphique s'impose.

Le meilleur éditeur est toujours celui que l'on connaît le mieux !

Les éditeurs les plus souvent rencontrés sont :

Editeurs graphiques

gedit	éditeur graphique par défaut de Gnome
kwrite	éditeur graphique élémentaire de l'environnement de bureau KDE
kate	éditeur graphique avancé de l'environnement de bureau KDE
mousepad	éditeur de texte par défaut de l'environnement de bureau xfce
leafpad	éditeur de texte par défaut de l'environnement de bureau lxde
Gvim, Xvim	vim avec une interface graphique

Editeurs ASCII

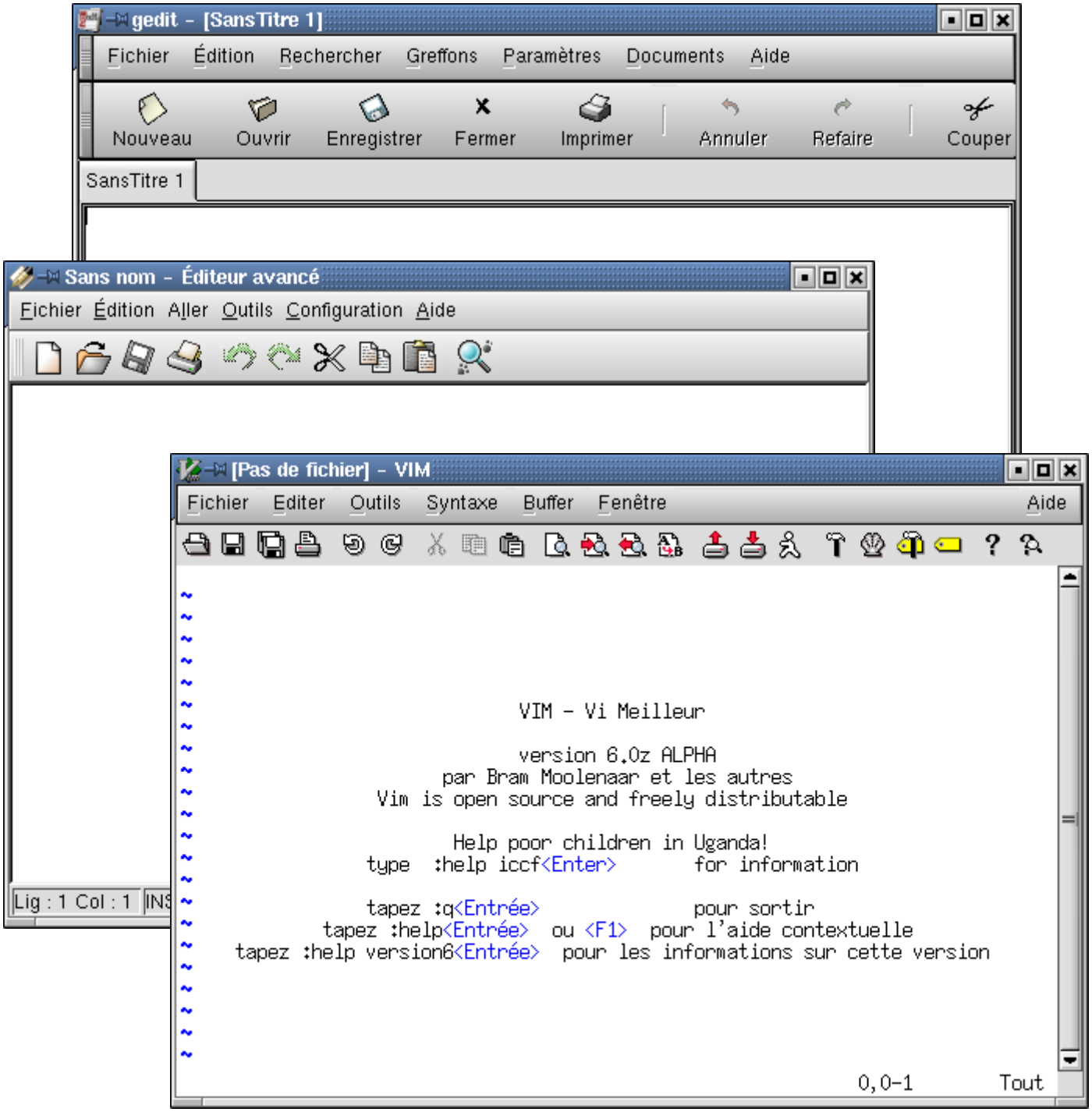
nano	très simple, adapté à une utilisation basique occasionnelle.
emacs	éditeur de texte très complet sert d'environnement de développement pour beaucoup de langages.
vi	éditeur de texte présent par défaut sur la majorité des systèmes Unix.
vim	« VI iMproved», un clone de vi amélioré et massivement utilisé, présent par défaut sur un très grand nombre de systèmes Linux, sert d'environnement de développement pour beaucoup de langages.

Les éditeurs graphiques étant en grande partie d'une utilisation intuitive nous laisserons le soin à chacun de les découvrir selon ses besoins.

L'éditeur "universel", présent sous la majorité des Unix et Linux est vi.

C’est celui que nous étudierons.

Quelques éditeurs graphiques :



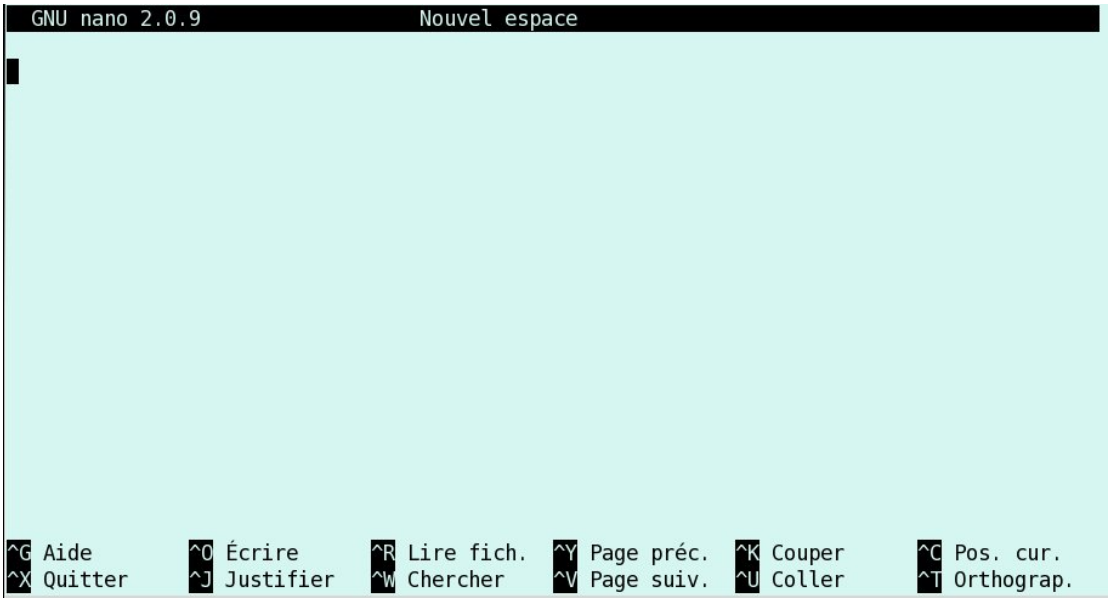
7.2 Nano

Si cet éditeur de texte s'appelle GNU nano, c'est parce qu'il est tout petit.

Il s'agit d'un programme très simple comparé à Vim et Emacs.

Il possède très peu de fonctions par rapport aux deux autres logiciels mais conviendra tout à fait pour une utilisation occasionnelle.

```
$ nano
```



Le symbole ^ signifie <Ctrl> c'est-à-dire la touche Contrôle du clavier.

Raccourcis les plus importants

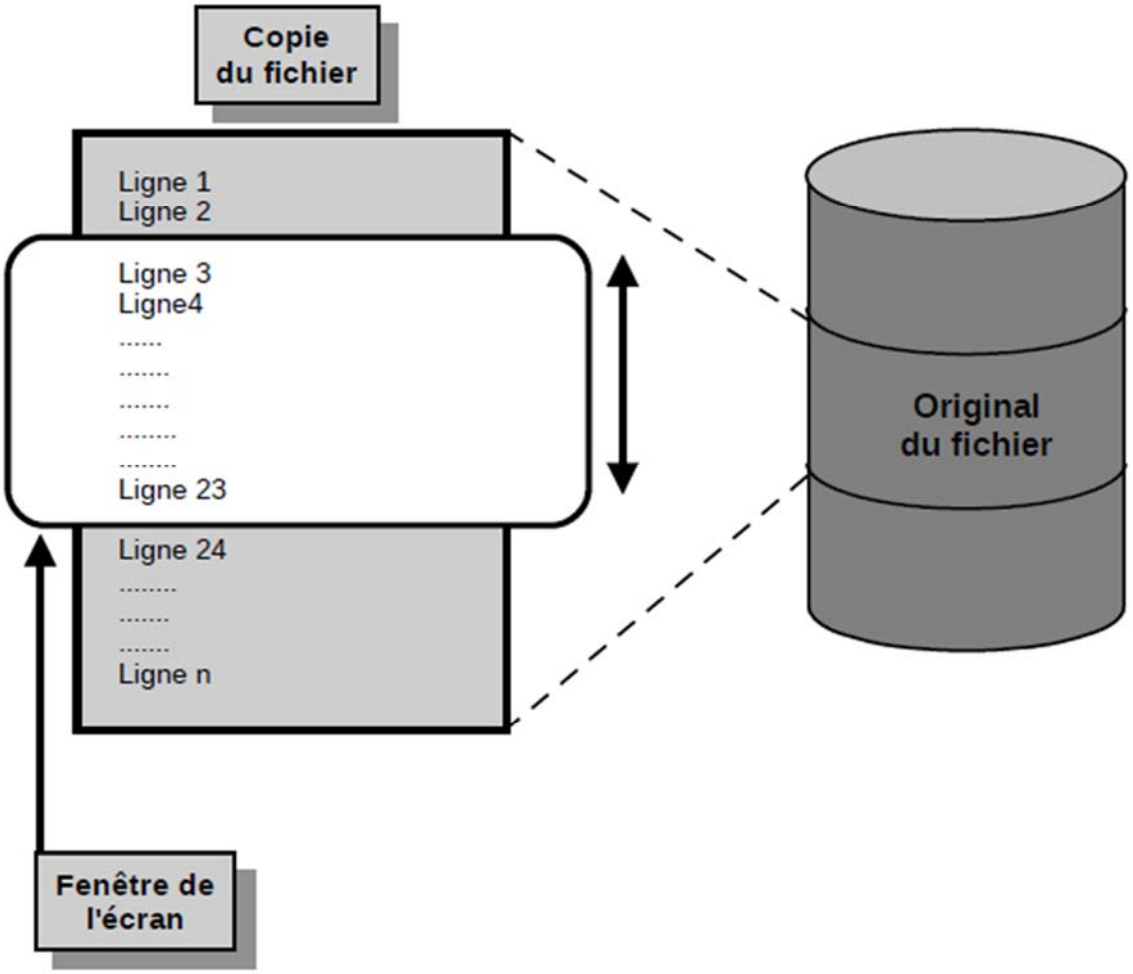
`Ctrl + G`	Afficher l'aide.
`Ctrl + K`	Couper la ligne et la mettre dans le presse-papier.
`Ctrl + U`	Coller la ligne de texte du presse-papier.
`Ctrl + C`	Afficher la position du curseur dans le texte.
`Ctrl + W`	Rechercher dans le fichier.
`Ctrl + O`	Enregistrer le fichier (écrire).
`Ctrl + X`	Quitter Nano.

7.3 Présentation de vi

vi est un éditeur texte ASCII pleine page qui s'ouvre à l'intérieur de la fenêtre de Shell.

vi peut paraître archaïque mais les notions que vous apprendrez dans ce chapitre resteront valables quel que soit l'éditeur que vous utiliserez.

- L'écran est une fenêtre ouverte sur le fichier.
- Accès à l'ensemble de la fenêtre par déplacement du curseur.
- Possibilité de déplacer la fenêtre dans le fichier à sa guise.
- En ouvrant un fichier à l'aide de vi, l'utilisateur travaille sur une copie du fichier dans un tampon de manœuvre. L'original ne sera modifié que lorsque les modifications seront enregistrées.


Copie
du fichier
Ligne 1
Ligne 2
Ligne 3
Ligne4
.....
.....
.....
.....
.....
Ligne 23
Ligne 24
.....
.....
.....
Ligne n
Original
du fichier
Fenêtre de
l'écran

7.3.1 Entrée dans vi

```
$ vi fichier

$ view fichier_en_lecture_seule
```

La fenêtre est au début du fichier.

Si le fichier n'existe pas, il sera créé par des directives de sortie.

La dernière ligne au bas de l'écran sert pour :

- Frapper des directives en mode « éditeur ligne » précédées de « : » (deux points) et validées par <Entrée>,
- Afficher des statistiques,
- Afficher des messages d'erreurs.

7.3.2 Sorties de vi

- Avec sauvegarde du texte

ZZ
Ou
:x <Entrée>
Ou
:w <Entrée> suivi de :q <Entrée>
Ou
:wq <Entrée>

- Sans sauvegarde

`:q <Entrée>` s'il n'y a pas eu de modifications
Ou
`:q!<Entrée>` s'il y a eu des modifications à ignorer

- Vers le Shell

`:!` commande
Ou
:sh
$ commande
$ commande
$<CTL><d>

7.3.3 Déplacer le curseur

Déplacement du curseur dans le texte :

Vers la gauche	<h> ou <backspace> ou <flèche gauche>
Vers le bas	<j> ou <Entrée> ou <flèche basse>
Vers la droite	<l> ou <Espace> ou <flèche droite>
Vers le haut	<k> ou <-> ou <flèche haute>

Déplacement du curseur sur une ligne :

Début de ligne	<^>
Fin de ligne	<$>

7.4 Les modes de vi

Mode « insertion »	Création ou insertion de texte.
Mode « commandes »	Suppression, copie, remplacement.
Mode « éditeur ligne »	Sauvegarde, recherche, substitution, numérotation lignes...

7.4.1 Mode insertion

Commandes d'entrée en mode insertion (entrée de texte) :

a	(append) Le texte sera ajouté après le curseur.
i	(insert) Le texte sera inséré avant le curseur.
o	(open minuscule) Nouvelle ligne sous la ligne courante.
O	(Open majuscule) Nouvelle ligne au-dessus de la ligne courante.

"ligne courante" : ligne sur laquelle se trouve le curseur.

Tous les caractères frappés après une de ces commandes seront insérés dans le fichier à partir de la position du curseur.

Passage d'une ligne à la suivante : `<Entrée>`

Sortie du mode insertion : `<Echap>`

7.4.2 Utilisation basique de vi

Pour créer un nouveau fichier, il faut donc :

- Appeler vi avec le nom du fichier à créer
- a ou i
- Saisir le texte
- <Echap>
- Sortir en sauvegardant le texte par :wq

Ceci est le minimum vital à retenir !

7.4.3 Mode commandes

C'est ce mode qui est présent à l'entrée dans « vi ».

Principales commandes du "mode commandes"

u	(undo) Annule la dernière commande frappée.
U	Annule toutes modifications sur une ligne
ZZ	Sortie de vi avec réécriture du fichier si modification (donc sauvegarde des modifications effectuées).
[n]x	(extract) Suppression du caractère sous le curseur (*) ou bien utiliser la touche <Suppr> du clavier.
[n]r	(replace) Remplacement du caractère sous le curseur par le caractère frappé ensuite (*).
[n]cw	(change word) Remplacement du mot sous le curseur par le mot frappé suivi de <Echap> (*).
[n]yw	(yank word) Mise en mémoire du mot sous le curseur (*).
[n]dw	(delete word) Suppression et mémorisation du mot sous le curseur (*).
[n]dd	Suppression de la ligne et mise en mémoire de celle-ci (*).
[n]yy	(yank) Mise en mémoire de la ligne (*).
p	(put minuscule) Insertion au-dessous de la ligne courante de la ligne en mémoire.
P	(PUT majuscule) Insertion au-dessus de la ligne courante de la ligne en mémoire.
J	(Join) Jonction de la ligne courante avec la suivante.
/mot	Positionne le curseur sur la ligne suivante contenant "mot"
(*) n	n = nombre de répétitions de la commande si indiqué.
5dd	Suppression de 5 lignes à partir de la ligne courante (ligne où se trouve le curseur.

7.4.4 Mode éditeur ligne

Les commandes de l'éditeur ligne, appelées aussi directives, commencent par " :" (deux points) et doivent être validées par `<Entrée>`.

" :" S'affiche sur la dernière ligne en bas de la fenêtre de Shell.

Les plus utiles sont :

:set nu	Numérote les lignes.
:set nonu	Suppression de la numérotation des lignes.
:set showmode	Le message `"-- INSERTION –"` s'affiche sur la dernière ligne (validé par défaut avec vim).
:15	Positionne le curseur en ligne 15 (`"$"` désigne la dernière ligne).
:w [fichier]	(write) Sauvegarde du fichier modifié sous le nom donné (ou le nom initial si le nom du fichier est omis).
:q!	(quit) Sortie de vi sans sauvegarde des modifications.
:f	Donne le nom du fichier en cours.
:1,$s/fic/fichier/	De la première à la dernière ligne, substitution de la 1ère chaîne de caractère `"fic"` par `"fichier"`.
:set showmatch	Permet de retrouver les parenthèses, crochets ou accolades fermantes correspondant au caractère ouvrant.
:set all	Affiche toutes les options de vi.

Remarques :

Vim utilisera la coloration syntaxique si le fichier est identifié comme un Shell-script.

Un fichier de texte sera considéré comme Shell-script s'il est exécutable ou si la première ligne est `#!/bin/sh` ou `#!/bin/bash` ou...

Conseils :

Conservez une copie de secours du fichier originale.

Sauvegardez le fichier (`:w`)de régulièrement pour éviter de tout perdre en cas d'erreur de manipulation.

8. EXPRESSIONS REGULIERES - GREP

OBJECTIFS

- Apprendre les mécanismes des expressions régulières
- Utiliser la commande **grep**

8.1 Les expressions régulières

Les expressions régulières sont des chaines de caractères augmentées de caractères spéciaux.

Ces expressions régulières sont utilisées par différentes commandes travaillant sur le contenu de fichiers de texte. Dans ce chapitre nous allons étudier la commande grep (Get REgular exPression). Ces expressions régulières sont aussi utilisées dans vi pour l'adressage des lignes ou lors de substitutions complexes.

8.1.1 Les caractères spéciaux

Un certain nombre de caractères ont une signification spéciale.

La combinaison de ces caractères permet de composer des critères de sélection pour construire des chaînes de caractères. C'est ce qu'on appelle "les expressions régulières".

$	Fin de ligne
^	Début de ligne
.	N'importe quel caractère
*	Un nombre quelconque de fois le caractère qui précède
.*	N'importe quelle chaîne de caractères y compris la chaîne vide
[ ]	Un des caractères situés entre les crochets
[^]	Un caractère non situé entre les crochets
\	Annule l'interprétation du caractère qui suit

8.1.2 Quelques exemples d'expressions régulières

Expressions régulières	Interprétations
a.	aa ou ab ou ac …
a.c	aac ou abc ou acc …
a\.c	a.c
[abcd] ou **[a-d]**	a, b, c ou d
[a-zA-Z]	une lettre minuscule ou majuscule
[1-37]	1, 2, 3 ou 7
[^abcd] ou **[^a-d]**	un caractère différent de a, b, c ou d
^3	ligne commençant par le chiffre 3
A$	ligne se terminant par le caractère A
xy*	x ou xy ou xyy ou xyyy…
^[0-9].*[0-9]$	ligne commençant et se terminant par un chiffre
/^Sauvegarde/	dans vi, positionne le curseur sur la prochaine ligne commençant par le mot "Sauvegarde"
:g/ *$/s///	dans vi, globalement sur tout le fichier, supprime tous les espaces en fin de ligne

8.2 La commande grep

8.2.1 Recherche d'une expression régulière

La commande grep permet d'effectuer une recherche, dans une liste de fichiers, d'une chaîne de caractères construite à partir d'une expression régulière.

```
grep [-options] 'expr' [fichier1]...[fichiern]
```

8.2.2 Quelques options

`-c` rend le nombre de lignes correspondant à l'expression

`-n` fait précéder chaque ligne par son rang dans le fichier

`-i` minuscules et majuscules identiques pour la recherche

`-s` annule les messages d'erreurs

`-v` rend les lignes ne correspondant pas à l'expression

8.2.3 Exemples d'utilisation de grep

```
$ grep '^user[1-7]:' /etc/passwd
user1:x:501:501::/home/user1:/bin/bash
user2:x:502:502::/home/user2:/bin/bash
user3:x:503:503::/home/user3:/bin/bash
user4:x:504:504::/home/user4:/bin/bash
user5:x:505:505::/home/user5:/bin/bash
user6:x:506:506::/home/user6:/bin/bash
user7:x:507:507::/home/user7:/bin/bash
$

$ grep '^[AG].*[6-8]$' edition
Aboaf Maurice 244748
Allo Jean-Pierre 255398
Gross Pascal 245367
Grosbois Anne 122456
$

$ grep -s '[^fe]grep ' /etc/*
...
```

9. PROCESSUS ET SHELL SCRIPTS

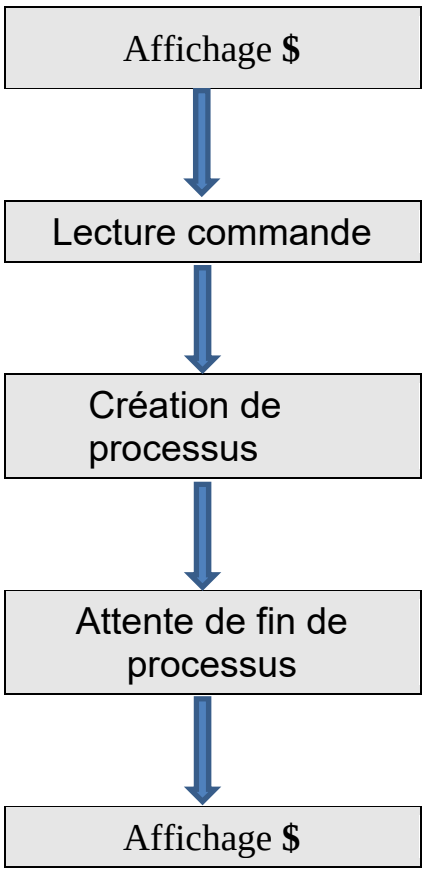
OBJECTIFS

- Décrire les mécanismes d'exécution des processus
- Lister les processus
- Identifier et utiliser les variables d'environnement du Shell

9.1 Gestion des processus

9.1.1 1Le rôle du Shell

Le Shell est un interpréteur de commandes.



9.2 Commandes internes/externes

Le Shell dispose de deux types de commandes :

Les commandes internes dont le code fait partie du code du Shell

Les commandes externes dont le code est dans des fichiers spécifiques

```
$
$ type cd
cd is a Shell builtin

$ type find
find is /usr/bin/find

$ type ll
ll is aliased to `ls -l'
$
```

9.3 Les alias

Un alias est un raccourci pour une commande. L'utilisateur peut ainsi lancer une commande avec options et paramètres en ne tapant qu'un mot.

9.3.1 Liste des alias prédéfinis

La liste des alias prédéfinis pour un utilisateur s'affiche avec la commande alias.

```
$ alias
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
```

9.3.2 Créer un alias

Pour créer un alias on peut utiliser les syntaxes suivantes :

```
alias nomalias=valeur
alias nomalias='commande'
alias nomalias='commande -options'
alias nomalias='commande param1 param2'
alias nomalias='/chemin/daccès/script'
alias nomalias='/chemin/daccès/script param1'
```

Dans l'exemple suivant on crée l'alias c pour accéder plus directement à la commande clear qui efface l'écran.

```
$ alias c='clear'
```

Utilisation.

```
$ c <Entrée>
```

9.3.3 Outrepasser un alias

Il est parfois nécessaire de ne pas passer par un alias.

On peut soit taper la commande d'origine en indiquant son chemin d'accès.

```
$ /usr/bin/clear
```

Ou indiquer au Shell de ne pas utiliser le processeur d'alias.

```
$ \c
```

9.3.4 Supprimer un alias

Pour supprimer définitivement un alias on utilisera la commande unalias.

```
$ unalias c
```

9.3.5 Alias permanent

Pour qu'un alias se retrouve présent lors d'une prochaine session il faut le déclarer dans un des scripts qui sont automatiquement lancés à l'ouverture de chaque session de Shell.

Souvent on utilisera le fichier $HOME/.bashrc, certaines distributions utilisent un fichier spécialement dédié aux alias comme $HOME/.aliases par exemple.

```
$ vi .bashrc
...
# User specific aliases and functions
alias c='clear'
...
```

9.4 Processus

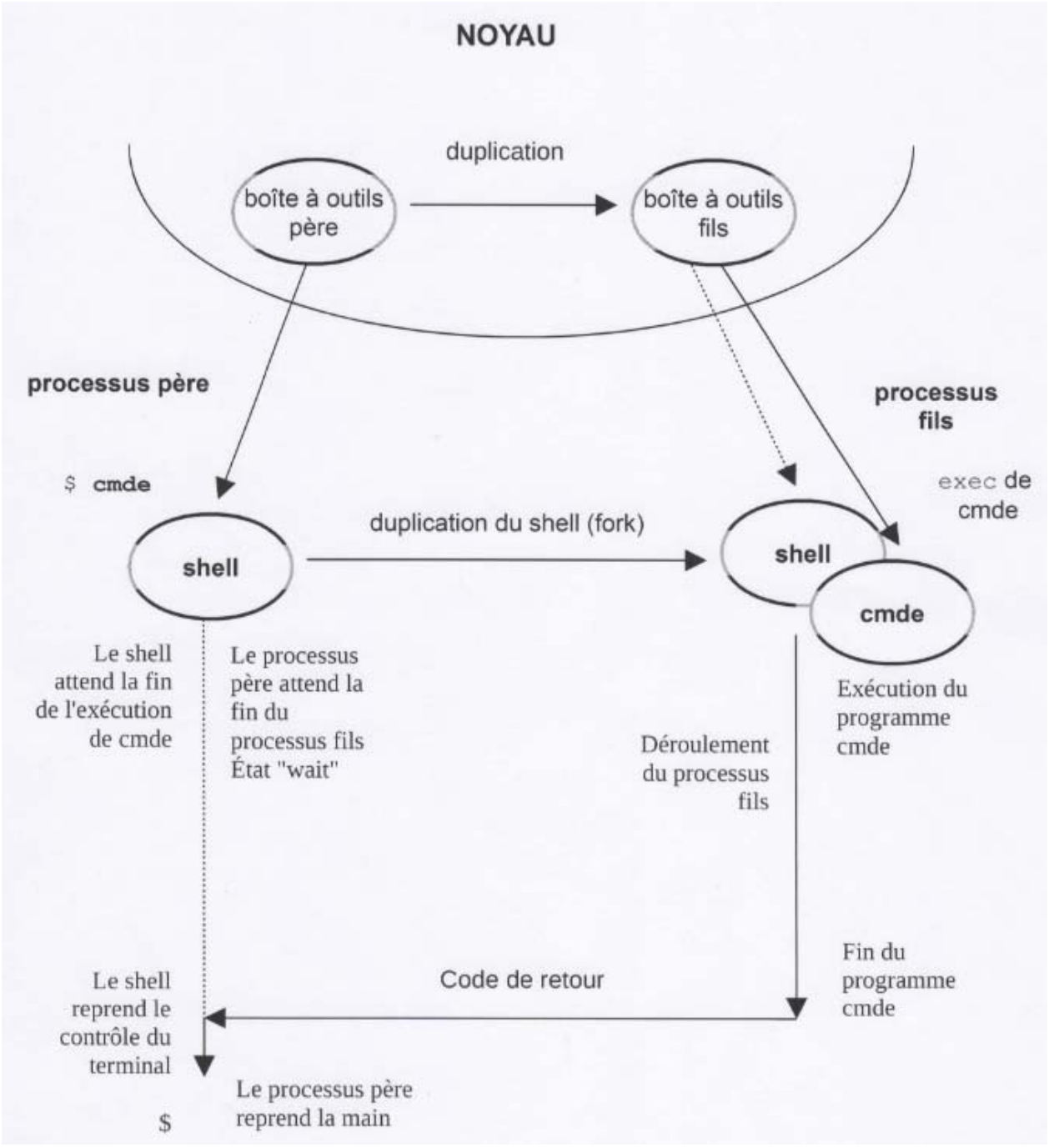
On appelle processus un programme en exécution.

Un processus est créé par un autre processus.

Les commandes internes sont exécutées par le Shell lui-même.

Pour les commandes externes, le Shell crée un processus fils qui se charge de l'exécution.

Les processus sont organisés en arborescence : père → fils


NOYAU
duplication
boîte à outils père
boîte à outils fils
processus père
processus fils
$ cmde
exec de cmde
shell
duplication du shell (fork)
shell
cmde
Le shell attend la fin de l'exécution de cmde
Le processus père attend la fin du processus fils État "wait"
Déroulement du processus fils
Exécution du programme cmde
Fin du programme cmde
Code de retour
Le shell reprend le contrôle du terminal
Le processus père reprend la main
$

9.5 Liste des Processus

La commande ps affiche la liste des processus en cours où :

PID = Process Identifier : numéro du processus

PPID = Parent Process ID : numéro du processus père

```
$ ps
PID TTY TIME CMD
13297 pts/1 00:00:00 bash
13342 pts/1 00:00:00 ps
$
$ ps -f
UID PID PPID C STIME TTY TIME CMD
user1 13297 13296 0 09:44 pts/1 00:00:00 -bash
user1 13343 13297 0 11:43 pts/1 00:00:00 ps –f
$
$ ps –ef | more
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 Jun20 ? 00:00:04 init [3]
root 2 1 0 Jun20 ? 00:00:00 [keventd]
...
root 1225 1 0 Jun20 ? 00:00:00 [nfsd]
root 1226 1 0 Jun20 ? 00:00:00 [nfsd]
root 1227 1 0 Jun20 ? 00:00:00 [nfsd]
root 1228 1225 0 Jun20 ? 00:00:00 [lockd]
root 1229 1228 0 Jun20 ? 00:00:00 [rpciod]
root 1230 1 0 Jun20 ? 00:00:00 [nfsd]
...
root 1821 1 0 Jun20 tty1 00:00:00 /sbin/mingetty tty1
root 1822 1 0 Jun20 tty2 00:00:00 /sbin/mingetty tty2
root 1823 1 0 Jun20 tty3 00:00:00 /sbin/mingetty tty3
...
user1 13297 13296 0 09:44 pts/1 00:00:00 –bash
user1 13344 13297 0 11:45 pts/1 00:00:00 ps –f
$
```

9.6 Commandes et programmes

Une commande est un programme exécutable, le Shell ne fait pas de distinction entre commande système et programme applicatif.

9.6.1 Règle de recherche d'une commande

Pour exécuter une commande le Shell recherche la commande dans les répertoires cités dans la variable PATH.

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:
/home/user1/bin:
```

> *Notez que par défaut le Shell ne recherche pas les programmes dans le répertoire de travail.*

9.6.2 Shell script

Un Shell script ou procédure, est un fichier contenant des lignes de commandes. Pour qu'il soit utilisable il faut rendre ce fichier exécutable.

```
$ cat >mon_programme
echo "Bonjour"
$ ls -l mon_programme
-rw-r--r--. 1 pat gp1 15 14 août 22:33 mon_programme
$ chmod u+x mon_programme
$ ls -l mon_programme
-rwxr--r--. 1 pat gp1 15 14 août 22:33 mon_programme
```

9.6.3 Lancement d'un programme utilisateur

Un programme qui n'est pas une commande système ne sera pas trouvé automatiquement.

```
$ mon_programme
bash: mon_programme : commande introuvable
```

Pour exécuter un programme on peut indiquer son chemin d'accès

```
$ ./mon_programme
Ceci est mon programme
```

ou bien modifier la variable PATH.

```
$ PATH=$PATH:.
$ mon_programme
Ceci est mon programme
```

ou bien

```
$ bash procedure1
...
```

> *Remarques :*
>
> *L'utilisation des variables sera étudiée dans un prochain chapitre.*
>
> *La programmation des Shell scripts sera vu dans le cours suivant.*

Le Shell (ksh ou bash) peut lire un fichier contenant un ensemble de commandes à exécuter.
On appelle ces fichiers de commandes des procédures ou Shell-scripts.

9.6.4 Exécution d'une procédure

```
$
$ chmod u+x procedure1
$
$ procedure1
...
```

9.6.5 Mise au point d'une procédure

La commande set permet de positionner les options du Shell pour la mise au point d'une procédure.

`set –x` Affichage de chaque commande après expansion.

`set –v` Affichage de chaque commande avant expansion.

`set –e` Sortie immédiate sur erreur.

`set +x` Invalide l'option (ici x) positionnée auparavant.

```
$ cat liste
# procédure liste
set -x #affichage après expansion
echo "Le répertoire courant est : "
pwd
echo "Voici la liste des fichiers : "
echo *
$
$ chmod u+x liste
$ liste
++ echo 'Le répertoire courant est : '
Le répertoire courant est :
++ pwd
/home/user1/tmp
++ echo 'Voici la liste des fichiers : '
Voici la liste des fichiers :
++ echo bordeaux brest lille liste lyon paris
bordeaux brest lille liste lyon paris
$
$ cat liste
# procédure liste
set -v #affichage avant expansion
echo "Le répertoire courant est : "
pwd
echo "Voici la liste des fichiers : "
echo *
$
$ liste
echo "Le répertoire courant est : "
Le répertoire courant est :
pwd
/home/pgar/tmp
echo "Voici la liste des fichiers : "
Voici la liste des fichiers :
echo *
bordeaux brest lille liste lyon paris
```

9.6.6 Commentaires dans une procédure

Le caractère # permet de commenter une procédure.

Si le commentaire est à la suite d'une ligne de commande, il doit être précédé d'au moins un espace.

> *Le dièse (♯) est un caractère différent du croisillon ou hashtag (#) ; ce dernier a les deux barres transversales horizontales, alors que celles du dièse sont montantes. (Wikipédia)*

9.6.7 Le shabang

Le shabang ou shebang, représenté par #! est vraisemblablement un mot valise représentant pour sharp (dièse) et bang désignant le point d'exclamation !

Lorsque la première ligne d'une procédure est de la forme :

#!/bin/bash

le système reconnait que ce fichier de texte est un script. L'interpréteur permettant d'exécuter ce script est indiqué à la suite.

9.7 Variables prédéfinies

9.7.1 Environnement

L'environnement d'un Shell contient un certain nombre de variables prédéfinies.

La liste de toutes les variables (prédéfinies ou initialisées pendant la session) peut s'obtenir par la commande set.

(L'exemple ci-dessous n'est qu'un échantillonnage du résultat.)

```
$ set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extquote:force_ignore:hostcomplete:interactive_comments:progcomp:promptvars:sourcepath
BASH_VERSION='4.1.2(1)-release'
COLUMNS=80
DESKTOP_SESSION=gnome
DISPLAY=:0.0
EUID=501
GDMSESSION=gnome
GDM_KEYBOARD_LAYOUT=$'fr\tlatin9
GDM_LANG=fr_FR.UTF-8
HISTFILE=/home/pat/.bash_history
HISTFILESIZE=1000
HOME=/home/pat
HOSTNAME=localhost.localdomain
HOSTTYPE=x86_64
ID=501
IFS=$' \t\n'
LANG=fr_FR.UTF-8
LOGNAME=pat
LS_COLORS='rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:...
MACHTYPE=x86_64-redhat-linux-gnu
MAIL=/var/spool/mail/pat
OLDPWD=/tmp
OSTYPE=linux-gnu
PATH=/usr/lib64/qt-3.3/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:.:/home/pat/bin:.
PPID=11001
PROMPT_COMMAND='printf "\033]0;%s@%s:%s\007" "${USER}" "${HOSTNAME%%.*}" "${PWD/#$HOME/~}"'
```

```
PS1='[\u@\h \W]\$ ' PS2='> ' PS4='+ ' PWD=/home/pat
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactiv
e-comments:monitor
TERM=xterm UID=501 USER=pat USERNAME=pat
```

9.7.2 Définitions

Voici les définitions de quelques-unes des variables du bash les plus importantes.

BASH	Chemin d'accès à l'exécutable bash
BASHOPTS	Options du bash
BASH_VERSION	Version du bash
DESKTOP_SESSION	Nom du gestionnaire de bureau
GDM_KEYBOARD_LAYOUT	Définitions du clavier vu par la session graphique
HISTFILE	Fichier qui contient l'historique des commandes
HISTFILESIZE	Taille maximale de ce fichier
HOME	Répertorie d'accueil de l'utilisateur connecté
HOSTNAME	Nom de la machine
HOSTTYPE	Type de CPU
IFS	Liste des séparateurs de champs dans une ligne de commande
LANG	Langue
LS_COLORS	Codes des couleurs affichées par ls selon types de fichiers
MACHTYPE	Type de CPU – distribution – noyau - système d'exploitation
OLDPWD	Répertoire de travail précédent
OSTYPE	Noyau – système d'exploitation
PATH	Répertoires de recherche des commandes
PS1	Prompt primaire
PS2	Prompt secondaire, Shell en attente (fermeture d'apostrophe p. ex.)
PS3	Prompt secondaire, pour la commande select
PS4	Prompt secondaire, pour la commande set -x
PWD	Répertoire ce travail courant
TERM	Type du terminal
UID	Numéro de l'utilisateur
USERNAME	Nom de l'utilisateur

9.8 Variables communes

9.8.1 Exportation des variables

En général, le Shell n'exécute pas lui-même les commandes externes qui lui sont soumises, mais crée un Shell fils qui se charge de l'exécution.

De ce fait, les variables du Shell père ne sont pas systématiquement connues des processus fils.

Pour qu'une variable soit commune à tous les processus fils d'un processus, elle doit être exportée.

```
$ cat prog
echo $var
$ prog

$ var=18
$ echo $var
18
$ prog

$ export var
$ echo $var
18
$ prog # Après "export var", var est connue de prog
18
```

9.8.2 Syntaxes d'exportation de variable

Voici différentes syntaxes utilisables pour exporter une variable.

```
var=18
export var
```

ou

```
export var=18
```

ou

```
typeset -x var=18
```

Il est possible d'exporter plusieurs variables à la fois. (extrait du fichier .bash_profile)

```
export PATH LANG PS1 EDITOR
```

> ***ATTENTION***
>
> ***- Il n'existe pas l'inverse de l'export (import).***
>
> ***- En conséquence, une variable, même exportée, modifiée dans une procédure appelée, ne voit pas sa valeur modifiée dans la procédure appelante.***

9.8.3 Liste des variables exportées

Pour visualiser la liste des variables transmises utilisez la commande env . (extrait)

```
$ env
HOSTNAME=localhost.localdomain
IMSETTINGS_INTEGRATE_DESKTOP=yes
SHELL=/bin/bash
TERM=xterm
HISTSIZE=1000
USER=pat
LS_COLORS=rs=0:di=01;34:ln=01;36:...
DESKTOP_SESSION=gnome
MAIL=/var/spool/mail/pat
PATH=/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin
:/home/pat:.
QT_IM_MODULE=xim
PWD=/home/pat
GDM_KEYBOARD_LAYOUT=fr_latin9
LANG=fr_FR.UTF-8
...
```

9.8.4 Les variables prédéfinies

Le Shell assigne automatiquement un certain nombre de variables qu'il utilisera.

Certaines variables sont affectées d'une valeur par défaut.

Pour plus d'informations :

```
$
$ man bash
...
Variables du Shell
Les variables suivantes sont remplies par l'interpréteur de
commandes :

BASH Se développe en chemin d'accès complet à l'instance courante
de bash.
...

$ help variables
variables: variables - Names and meanings of some Shell variables
Common Shell variable names and usage.
BASH_VERSION Version information for this Bash.
CDPATH A colon-separated list of directories to search
for directories given as arguments to `cd'.
GLOBIGNORE A colon-separated list of patterns describing
filenames to
be ignored by pathname expansion.
HISTFILE The name of the file where your command history is
stored.
HISTFILESIZE The maximum number of lines this file can contain.
HISTSIZE The maximum number of history lines that a running
Shell can access.
HOME The complete pathname to your login directory.
HOSTNAME The name of the current host.
...
```

```
$ man ksh

$ set
```

9.9 Code de retour

9.9.1 Code de retour d'une commande

A la fin de l'exécution d'une commande, un **code de retour** est toujours positionné.

Le code de retour d'une commande peut être vérifié en affichant sa valeur contenue dans la variable **?** .

```
echo $?
```

Exemples :

Admettons que le fichier fic1 existe dans notre répertoire de travail, et que le fichier fic3 n'existe pas.

```
$
$ cat fic1
Ceci est le fichier fic1
$ echo $?
0
$ cat fic3
fic3 : Aucun fichier ou dossier de ce type
$ echo $?
1
$
```

9.9.2 Valeurs du code de retour

Lorsqu'une commande se déroule "normalement" son code de retour est **égal à 0** (zéro), on dit alors que le code de retour est "**vrai**".

Lorsqu'une commande sort sur un message d'erreur son code de retour est **différent de 0**, on dit alors que le code de retour est "**faux**".

> *Attention il n'y a pas de jugement de valeur lorsqu'on dit "vrai" ou "faux"*

9.9.3 Forcer le code de retour

Les commandes usuelles sont programmées pour donner des codes de retour particuliers si elles ne peuvent se dérouler normalement.

```
$
$ ls azertyu
ls : impossible d'accéder à azertyu : aucun fichier ou dossier de
ce type
$ echo $?
2
$ ls -z
ls : option invalide – 'z'
Saisissez " ls –help " pour plus d'informations.
$ echo $?
2
$ azertyu
bash : azertyu : commande introuvable
$ echo $?
127
$
```

On peut faire de même dans l'écriture d'un Shell script.
La commande exit n provoque la sortie du Shell script en positionnant le code de retour à une valeur donnée.

```
$ cat prog3
exit 3
$ prog3
$ echo $?
3
$
```

9.10 Les opérateurs du Shell

Les opérateurs && et || permettent de programmer les mêmes structures que le if dans des cas plus simples.

9.10.1 L'opérateur &&

commande1 && commande2

Exécution de commande 1, si commande1 fourni un code de retour égal à 0
alors exécution de commande2.

9.10.2 L'opérateur ||

commande1 || commande2

Exécution de commande 1, si commande1 fourni un code de retour différent de 0
alors exécution de commande2.

```
mkdir demo && echo "Le répertoire demo est créé"

mkdir demo || echo "Le répertoire demo n'a pas pu être créé"

cd demo &&ls -l demo ||echo "demo n'est pas un répertoire "
```

10. PARAMETRES ET VARIABLES

OBJECTIFS

- Transmettre les paramètres à une procédure
- Affecter et utiliser les variables du Shell

10.1 Les paramètres

10.1.1 Les paramètres d'une procédure

Au lancement d'une procédure Shell, ou Shell-script, on peut transmettre des valeurs en utilisant des paramètres positionnels sur la ligne de commande.

> *En anglais les "paramètres" sont appelés "arguments". De là à employer le même vocable en français, il n'y a qu'un pas que beaucoup n'hésitent pas à franchir.*

Ces paramètres sont automatiquement affectés à des variables par le Shell.

$	nom_prodédure	paramètre1	paramètre	paramètre	...
	0	**1**	**2**	**3**	

10.1.2 Utilisation des paramètres dans la procédure

A l'intérieur de la procédure on pourra utiliser ces paramètres dans les variables correspondantes :

$1	Premier paramètre
$2	Deuxième paramètre
$3	Troisième paramètre
...	
${10}	Dixième paramètre
...	
${n}	Nième paramètre

> *Notez la syntaxe particulière à partir du dixième paramètre.*

L'utilisation des paramètres répond aux règles d'utilisation des variables du Shell à ceci près qu'on ne peut pas changer la valeur de ces variables donnée dans la ligne de commande. (n=valeur est interdit).

```
$ 3=abc
bash: 3=abc : commande introuvable
$
```

Seule la commande set (étudiée plus loin dans ce chapitre) permettra d'affecter une valeur manuellement aux paramètres.

10.1.3 Notion de variable

Lorsqu'on affecte une valeur à une variable, on récupére cette valeur en appelant la variable $variable.

Exemple :

Si "nom" est le nom de la variable, $nom sera remplacé par "Dupont" c'est à dire le contenu de la variable.

```
$ nom=Dupont
$ echo $nom
Dupont
$
```

10.1.4 Autres variables liées aux paramètres

Il existe d'autres variables gérées par l'interpréteur.

$0	Nom de la procédure (non modifiable)
$#	Nombre total de paramètres
$*	Une chaîne de caractères constituée de tous les paramètres
$$	pid (ou numéro) du processus

Dans cet exemple nous avons créé un fichier de texte appelé ville.

Ce fichier de texte ne contient que des lignes de commandes valides, ici nous n'utiliserons que la commande echo.

```
$ cat ville
echo nom de la procedure : $0
echo le premier paramètre est : $1
echo le troisième paramètre est : $3
echo nombre de paramètres : $#
echo voici tous les paramètres : $*
$
$ ls -l ville
-rw-r--r-- 1 user2 base 169 fév 11 16:21 ville
$ chmod u+x ville
$ ls -l ville
-rwxr--r-- 1 user2 base 169 fév 11 16:21 ville*
$
$ ville Paris Bordeaux LeMans StOuen
nom de la procédure : ville
le premier paramètre est : Paris
le troisième paramètre est : LeMans
nombre de paramètres : 4
voici tous les paramètres : Paris Bordeaux LeMans StOuen
```

10.1.5 Décalage des paramètres

Le décalage des paramètres positionnels se fait par la commande shift.

Après shift :

$0 ne bouge pas

$1 est perdu

Le deuxième paramètre devient premier paramètre : $2 ---> $1

Le troisième paramètre devient deuxième paramètre : $3 ---> $2

etc…

Le nième paramètre devient paramètre n-1 : ${n} ---> ${n-1}

$# et $* sont mis à jour.

Il est possible de préciser le nombre de positions de décalage : shift n.

```
$ cat decalage
echo le premier paramètre est : $1
echo le neuvième paramètre est : $9
echo nombre de paramètres : $#
echo " ATTENTION DECALAGE"
shift
echo le premier paramètre est : $1
echo le neuvième paramètre est : $9
echo nombre de paramètres : $#
$
$ chmod u+x decalage
$
$ decalage
le premier paramètre est :
le neuvième paramètre est :
nombre de paramètres : 0
ATTENTION DECALAGE
le premier paramètre est :
le neuvième paramètre est :
nombre de paramètres : 0
$
$ decalage A B C D E F G H I J K
le premier paramètre est : A
le neuvième paramètre est : I
nombre de paramètres : 11
ATTENTION DECALAGE
le premier paramètre est : B
le neuvième paramètre est : J
nombre de paramètres : 10
$
```

10.2 Les variables du Shell

10.2.1 Les types de variables

Le Shell reconnait 4 types de variables :

- les variables chaine de caractères
- les variables entiers numériques
- les tableaux de chaines de caractères
- les tableaux d'entiers

10.2.2 Variables de chaînes de caractères

```
$
$ chaine='suite de caractères'
$
$ echo $chaine
suite de caractères
$
```

Le nom d'une variable se différencie de son contenu :

***var** nom de la variable*

***$var** contenu de la variable*

Le signe = (égale) sert à affecter une valeur à la variable nommée.

Il n'y a d'espace ni à droite ni à gauche du "=".

10.2.3 Variable de nombres entiers

Les variables de nombres entiers sont souvent appelées variables entières.

Les variables entières sont déclarées au moyen des commandes suivantes.

`typeset -i var1=111`	ksh et bash
`integer var2=222`	ksh uniquement
`declare -i var3=333`	bash uniquement

> *En anglais "nombre entier" se dit 'integer" d'où le -i*

Dans tous les cas l'utilisation des variables est la même.

```
$ echo $var1 $var2 $var3
111 222 333
$
```

> *Dans les exemples du cours nous utiliserons plus volontiers une syntaxe commune au ksh et au bash.*

Une variable entière signifie que le contenu de la variable n'est pas composé de la valeur ascii des caractères mais de la valeur binaire du nombre contenu.

nb=5 le contenu de la variable **nb** est 0x35 (35 en hexadécimal)
soit 0011 0101 en binaire.

typeset -i nbre=5 le contenu de la variable **nbre** est 0000 0101 en binaire

On peut aussi déclarer les variables entières, puis leur affecter une valeur dans un deuxième temps.

```
$ typeset -i a b c
$ a=2
$ b=3 ; c=4
$
```

10.2.4 Les tableaux

Le Shell propose des variables tableaux à une dimension.

N'importe quelle variable peut être utilisée comme tableau.

Il n'y a pas de limite maximale à la taille d'un tableau.

Il n'y a pas d'obligation que les membres d'un tableau soient indexés ou affectés de manière contiguë.

Les tableaux sont indexés par des entiers en commençant à zéro.

10.2.4.1 Les tableaux de chaine de caractères

```
$ chaine='suite de caractères'
$
$ chaine[1]='encore des caractères'
$
$ echo ${chaine[1]}
encore des caractères
$
```

> *Notez la syntaxe particulière avec l'utilisation des {} (accolades)*

La variable chaine[0] existe déjà.

```
$ echo ${chaine[0]}
suite de caractères
$
```

Une variable simple est en réalité l'affectation du premier élément du tableau de même nom et de même type.

Une variable simple est donc potentiellement un tableau de variable.

10.2.4.2 Les tableaux d'entiers

Les index des tableaux sont eux mêmes des entiers, ces index peuvent être eux aussi contenus dans des variables.

```
$
$ typeset -i tabentiers
$ typeset -i i
$ tabentiers[0]=13
$ i=1
$ tabentiers[i]=45
$ echo ${tabentiers[1]}
45
```

10.2.5 Lister, supprimer les variables

typeset –i en ksh et bash, donne la liste des variables entières

declare -i en bash uniquement, donne la liste des variables entières

unset var1 efface la variable var1

Attention à ne pas confondre

unset i efface la variable i

10.3 Les variables interactives

La valeur d'une variable peut être entrée par l'opérateur durant le déroulement d'une procédure.
Pour cela on utilise la commande read.

read variable

En bash comme en ksh, la commande read met la procédure en attente d'une saisie au clavier.

La variable est créée par le Shell et à l'appui sur la touche <Entrée>, ce qui a été saisi est affecté à la variable. Puis le programme reprend son cours.
Si rien n'a été saisi, la variable est créée vide.

En ksh, la commande read fonctionne comme ci-dessus mais on peut lui ajouter un message à afficher.

read variable?"message"

```
$
$ cat menu_ksh
echo ceci est un exemple de menu
read reponse?"Quel est votre choix : 1,2 ou 3 ? "
echo vous avez choisi $reponse
$
```

En pratique on prendra soin d'indiquer ce qu'on attend de l'opérateur en affichant un message grâce à la commande echo.

```
$ cat menu_bash
echo ceci est un exemple de menu
echo -e "Quel est votre choix : 1,2 ou 3 ?\c"
read reponse
echo vous avez choisi $reponse
$
```

Remarques sur les différentes syntaxes de la commande **echo**.

Dans l'exemple ci-dessus, nous avons employé la commande **echo** avec la syntaxe **GNU**, l'option **-e** permet l'interprétation des séquences de caractères précédées d'un backslash. Ici la séquence **\c** supprime le saut de ligne final.

Dans certains Unix la commande **echo** s'emploie sans le **-e**.
On aurait donc la syntaxe suivante.

```
$ cat menu2_ksh
echo ceci est un exemple de menu
echo "Quel est votre choix : 1,2 ou 3 ?\c"
read reponse
echo vous avez choisi $reponse
$
```

Les scripts menu_ksh, menu_bash_ksh et menu2_ksh ont tous le même effet.

```
$ menu_ksh
ceci est un exemple de menu
Quel est votre choix : 1,2 ou 3 ? 2
vous avez choisi 2

$ menu_bash
ceci est un exemple de menu
Quel est votre choix : 1,2 ou 3 ? 2
vous avez choisi 2

$ menu2_ksh
ceci est un exemple de menu
Quel est votre choix : 1,2 ou 3 ? 2
vous avez choisi 2
```

10.4 Protection d'une variable

La commande **readonly** protège une variable en écriture en lui affectant l'attribut lecture-seule..

```
readonly var
```

ou bien

```
typeset -r var
```

On ne peut plus modifier la valeur d'une variable **readonly** lors d'assignations ultérieures ni supprimer la variable.

```
$ var=45
$ echo $var
45
$ var=12
$ echo $var
12
$
$ readonly var
$ echo $var
12
$ var=45
bash: var : variable en lecture seule
$
$ unset var
bash: unset: var : « unset » impossible : variable est en lecture
seule
$
```

En ksh on peut supprimer l'attribut readonly positionné préalablement sur une variable.

```
typeset +r var
```

10.5 Valeur par défaut d'une variable

10.5.1 Variable non définie

`$ echo ${var-val1}`	L'écriture est substituée par val1 si **var** n'est pas définie si non l'écriture est substituée par la valeur de **var**.
`$ echo ${var=val2}`	Si **var** n'est pas définie, var est créée et prend la valeur val2. L'écriture est substituée par la valeur de **var**.

Exemple

Le script imp1 imprime un exemplaire par défaut du fichier dont le nom est donné en premier paramètre.

```
$ cat imp1
nb=${2-1}
pr –o8 $1 | lp –n $nb
echo "Il a été imprimé $nb exemplaires(s) du fichier $1"
$
$ imp1 texte
Il a été imprimé 1 exemplaire(s) du fichier texte
$
$ imp1 texte 5
Il a été imprimé 5 exemplaire(s) du fichier texte
$
```

Notez que le bash rend un message à l'issue de la commande lp qui n'a pas été reproduit dans ces exemples pour ne pas les alourdir et se concentrer sur l'objet de notre étude.

Cet exemple est destiné à illustrer le mécanisme de la substitution des variables par une valeur par défaut, les commandes pr et lp ne sont données qu'à titre d'illustration.

pr *: Mettre en forme des fichiers de texte pour l'impression*

lp *: imprimer des fichiers*

10.5.2 Variables nom définies ou vides

`$ echo ${var:-val1}`	L'écriture est substituée par val1 si **var** n'est pas définie ou vide si non l'écriture est substituée par la valeur de **var**.
`$ echo ${var:=val2}`	Si **var** n'est pas définie ou vide, **var** est créée et prend la valeur val2. L'écriture est substituée par la valeur de **var**.

Exemple

Le script imp2 fait le même traitement que la commande imp1 mais en posant des questions à l'utilisateur.

```
$
$ cat imp2
# Solution bash
echo –e "Donnez le nom du fichier à imprimer : \c"
read fic
echo –e "Donnez le nombre d'exemplaire(s) à imprimer : \c"
read nb
pr –o8 $fic | lp –n ${nb:=1}
echo "Le fichier $fic a été imprimé en $nb exemplaire(s)"
$
$ imp2
Donnez le nom du fichier à imprimer : texte
Donnez le nombre d'exemplaire(s) à imprimer : <Entrée>
Le fichier texte a été imprimé en 1 exemplaire(s)
$
$ imp2
Donnez le nom du fichier à imprimer : texte
Donnez le nombre d'exemplaire(s) à imprimer : 5
Le fichier texte a été imprimé en 5 exemplaire(s)
$
```

Voir les remarques de la page précédente.

10.6 Développement des variables

Nous avons vu que le rôle du Shell est de remplacer (substituer) l'expression $nom_variable par le contenu de la variable.

Il existe de nombreuses syntaxes qui permettent d'aller plus loin dans ce rôle de substitution, ces fonctionnalités sont appelées développement des variables. Parmi lesquelles en voici un échantillon.

```
$ var=chaine
$ echo ${#var}
6
```

Rend la longueur du contenu de la variable

```
$ typeset –i tab
$ tab[0]=1; tab[1]=2;
tab[2]=7
$ echo ${tab[*]}
1 2 7
```

Liste les valeurs du tableau **tab**

```
$ echo ${#tab[*]}
3
```

Donne le nombre d'éléments du tableau

```
$ echo ${#tab[n]}
```

Rend la longueur du contenu de l'élément n du tableau **tab**

```
${var?'message'}
```

Si **var** n'est pas définie, affiche 'message' en plus du message d'erreur

```
$ echo $toto

$ echo ${toto?'La variable toto est inexistante'}
bash: toto: La variable toto est inexistante
$
$ toto=Bonjour
$ echo ${toto?'La variable toto est inexistante'}
Bonjour
$
```

10.7 Récupérer le résultat d'une commande

Il arrive que dans une ligne de commande on doive récupérer le résultat d'une autre commande (faites le parallèle avec la récupération du contenu d'une variable).

10.7.1 Remplacement d'une commande par son résultat

Nous avons à notre dispositions deux syntaxes.

`commande`
ou
$(commande)

Dans les deux cas l'expression est remplacée par le résultat de la commande.

> *Dans la suite du cours nous utiliserons la deuxième syntaxe de préférence.*
> *En effet l'utilisation des `` (accents graves) est souvent source de confusion lorsqu'on utilise conjointement des apostrophes et guillemets.*

```
$ date +%x
05/08/2014
$
$ echo "Nous sommes le $(date +%x)"
Nous sommes le 05/08/2014
$
```

10.7.2 Récupération du résultat d'une commande dans une variable

Ce mécanisme est utilisé dans les procédures Shell si le résultat invariable de la commande doit être utilisé plusieurs fois.

```
$ echo $jour

$ jour="$(date +%x)"
$
$ echo "Nous sommes le $jour"
Nous sommes le 05/08/2014
$
```

10.7.3 Récupération des champs du résultat d'une commande

La commande set suivie de mots, envoie ces mots dans les variables positionnelles. Si elle est suivie d'une commande évaluée, les champs du résultat de cette commande seront donc récupérés dans les paramètres de la procédure ($1 $2 $3 … ${n}).

> *ATTENTION : si la procédure prend des paramètres au lancement, il faudra soit traiter ces paramètres avant la commande* ***set*** *soit les sauvegarder dans des variables non positionnelles.*

```
$
$ echo $1 $2 $3

$
$ set un deux trois
$
$ echo $1 $2 $3
un deux trois
$
$ echo $3 $2 $1
trois deux un
$
```

Cette méthode peut être utilisée pour récupérer les différents champs du résultat d'une commande.

```
$
$ date
ven. sept. 5 11:45:10 CEST 2014
$
$ set $(date)
$
$ echo "Nous sommes le $1 $3 $2 $6"
Nous sommes le ven. 5 sept. 2014
$
```

10.8 Les "quotes"

Il y a trois mécanisme de protection des caractères au sein d'une ligne de commande :

\ barre oblique inverse ou back-slash

' ' apostrophes (quote)

" " guillemets (double-quote)

10.8.1 Barre oblique inverse (\)

Le caractère barre oblique inverse (\) préserve la valeur littérale du caractère qui le suit.

```
$
$ echo Voici l'heure $(date +%Hh%M)
>
```

Ici le Shell a interprété l'apostrophe et attend que la séquence soit fermée par une autre apostrophe.

```
$ echo Voici l\'heure $(date +%Hh%M)
Voici l'heure 11h57
$
```

L'apostrophe est ici protégée par la barre oblique inverse, elle n'est pas interprétée par le Shell.

10.8.2 Les apostrophes

Encadrer des caractères entre des apostrophes préserve la valeur littérale de chacun des caractères y compris des caractères spéciaux.

Lorsque le Shell détecte une apostrophe ouvrant une séquence, il considérera l'apostrophe suivante comme fermant la séquence. En conséquence une apostrophe ne peut pas être placée entre deux apostrophes, même si elle est précédée d'une barre oblique inverse.

```
$
$ ville=Paris
$
$ echo ' ***** A $ville il est $(date +%Hh%M) *****'
***** A $ville il est $(date +%Hh%M) *****
$
```

Dans cet exemple tous les caractères sont entre apostrophes, ils ne sont donc pas interprétés par le Shell.

10.8.3 Les guillemets

Encadrer des caractères entre des guillemets préserve la valeur littérale de chacun des caractères sauf $, `, et \ (dollar, accent grave et barre oblique inverse).

Lorsque le Shell détecte un guillemet ouvrant une séquence, il considérera le guillemet suivant comme fermant la séquence. Un guillemet peut toutefois être protégé entre deux guillemets, à condition de le faire précéder par une barre oblique inverse.

```
$
$ echo " ***** A \"$ville\" il est $(date +%Hh%M) *****"
***** A "Paris" il est 12h26 *****
$
```

Dans cet exemple on constate que :

Les caractères spéciaux <Espace> et * n'ont pas été interprétés par le Shell,

Les guillemets autour de la variable ont été préservés parce que protégés par une barre oblique inverse,

Le $ a été interprété par le Shell, ainsi la variable ville et la commande date ont pu être substitués par le Shell.

11. CONNEXION DISTANTE

OBJECTIFS

- Vérifier l'accès au réseau local
- Se connecter sur un serveur distant

11.1 Connexion à distance

On peut se connecter à distance à un ordinateur sous Unix ou Linux et ouvrir une session Shell. C'est comme cela que l'on administre les serveurs...

Le système qui se connecte à un serveur Linux est appelé le client.

On peut se connecter à un serveur Linux distant depuis un poste de travail sous Linux mais aussi depuis un poste de travail sous n'importe quel autre système d'exploitation Windows, Mac OS...

Sous Windows, on utilisera un émulateur de terminal tel que PuTTY qu'il faudra télécharger, installer et configurer.

A partir d'un Shell sous Linux on utilisera la commande `ssh`.

Les données qui sont échangées entre le client et le serveur sont chiffrées grâce au protocole SSH afin de garantir la confidentialité des échanges.

11.2 Les outils de base

Les outils suivants sont utiles à connaître lorsque l'on veut utiliser le réseau avec un système sous Linux.

11.2.1 ifconfig

La commande `ifconfig` permet de configurer une interface réseau.

Affichage des interfaces actives :

```
$ ifconfig
lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:720 (720.0 b)  TX bytes:720 (720.0 b)

eth0      Link encap:Ethernet  HWaddr 6C:88:14:25:EE:84
          inet adr:192.168.0.11  Bcast:192.168.0.255  Masque:255.255.255.0
          adr inet6: fe80::6e88:14ff:fe25:ee84/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:29966 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19749 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:39267736 (37.4 MiB)  TX bytes:2348069 (2.2 MiB)
```

On peut aussi désactiver une carte réseau :

```
$ ifconfig eth0 down

ou bien

$ ifdown eth0
```

Et bien sûr la réactiver :

```
$ ifconfig eth0 up

ou bien

$ ifup eth0
```

11.2.2 ping

La commande `ping` permet de savoir si une machine distante est accessible.

On peut nommer le système distant en utilisant son adresse IP ou son nom (si configuré).

```
$ ping linux2
ou bien
$ ping 192.168.0.11
PING 192.168.0.11 (192.168.0.11) 56(84) bytes of data.
64 bytes from 192.168.0.11: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from 192.168.0.11: icmp_seq=2 ttl=64 time=0.050 ms
64 bytes from 192.168.0.11: icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from 192.168.0.11: icmp_seq=4 ttl=64 time=0.053 ms
^C
--- 192.168.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3535ms
rtt min/avg/max/mdev = 0.029/0.046/0.053/0.011 ms
```

On arrête les pings en tapant `<Ctrl>+C`.

Pour envoyer un nombre donné de pings on utilisera l'option `-c`.

```
$ ping 192.168.0.11 -c 2
PING 192.168.0.11 (192.168.0.11) 56(84) bytes of data.
64 bytes from 192.168.0.11: icmp_seq=1 ttl=64 time=0.050 ms
64 bytes from 192.168.0.11: icmp_seq=2 ttl=64 time=0.061 ms

--- 192.168.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.050/0.055/0.061/0.009 ms
```

11.2.3 ssh

La commande `ssh` permet de se connecter sur une machine Linux distante.

```
ssh -l nom_utilisateur  nom_système_distant

ssh nom_utilisateur@nom_système_distant
```

Soit deux systèmes : linux1 et linux2

L'utilisateur est "ken", il est connecté sur linux1, il veut ouvrir une session Shell distante sur linux2 en se connectant au compte "brian". *

```
[ken@linux1] $ hostname
linux1
[ken@linux1] $ ssh linux2 -l brian
The authenticity of host 'linux1 (192.168.0.11)' can't be
established.
RSA1 key fingerprint is
1z:2y:3x:4w:56:78:98:78:ab:cd:ef:01:23:45:67:89.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'linux1 (192.168.0.11)' (RSA1) to
the list of known hosts.
brian@linux2's password:
xxxxxx
[brian@linux2] $ hostname
linux2
[brian@linux2] $ exit
[ken@linux1] $
```

La commande `ssh` permet aussi de lancer une commande sur une machine distante.

Maintenant l'utilisateur "ken" veut lancer la commande `ls` sur la machine linux2 et recevoir le résultat sur son écran.

```
[ken@linux1] $ ssh brian@linux2 ls -l rep2
brian@linux2's password:
xxxxxx
total 12
-rw-r--r--. 1 brian  gp1 1251 17 avril 22:54 texte
-rw-r--r--. 1 brian  gp1 1301 17 avril 22:54 texte1
-rw-r--r--. 1 brian  gp1 1251 17 avril 22:54 texte.original
[ken@linux1] $
```

(*) En hommage à Ken Thompson créateur d'Unix et Brian Kernighan créateur du C.

12. GESTION DES IMPRESSIONS

OBJECTIFS

- Vérifier la configuration des impressions
- Lancer des impressions

12.1 Serveur d'impression

Il existe trois standards d'impression sous Linux, un issu du System V, un autre issu de BSD et un dernier, fédérateur, CUPS.

12.1.1 Principe

Quel que soit le standard, le principe est le même. À chaque imprimante déclarée (généralement dans /etc/printcap) correspond une file d'attente (queue). L'ensemble de ces files d'attente est géré par un service indépendant.

Ce principe permet une impression multiutilisateur en local ou en réseau (le service peut être utilisé depuis une autre machine distante).

Tous les travaux d'impression (jobs) sont mis en file d'attente.

En règle générale toutes les imprimantes savent directement imprimer du texte brut ASCII en 80 colonnes.

Pour imprimer des documents formatés ou des images, on peut utiliser un pilote appelé filtre d'impression. Le filtre peut être un script ou un binaire qui récupère le flux entrant (texte, image, document, Postscript...), l'identifie et à l'aide de traitements associés le transforme en langage compréhensible par l'imprimante (Postscript, PCL, Canon, Epson, WPS...).

Linux accepte les commandes issues des Unix de type System V et BSD. Pendant longtemps, le sous-système d'impression était basé sur les services BSD et le démon lpd.

Depuis quelques années, toutes les distributions se basent sur CUPS, compatible (pour les commandes en tout cas) avec les anciens systèmes d'impression.

12.2 Commandes d'impression

Les commandes liées à l'impression sont amenées par le produit `cups` qui gère les imprimantes et les files d'attente.

Il fournit les jeux de commandes issues à la fois de BSD et de System V.

12.2.1 La commande lpr (BSD)

La commande `lpr` permet d'imprimer un fichier.

```
$ lpr nom_fichier
```

Pour imprimer sur une imprimante en particulier.

```
$ lpr -P nom_imprimante nom_fichier
```

Pour spécifier le nombre de copies (ici 2 copies).

```
$ lpr -#2 nom_fichier
```

12.2.2 La commande lp (Sys V)

La commande `lp` permet d'imprimer un fichier.

```
$ lp nom_fichier
```

Pour imprimer sur une imprimante en particulier.

```
$ lp -d nom_imprimante nom_fichier
```

Pour spécifier le nombre de copies (ici 2 copies).

```
$ lp -c 2 nom_fichier
```

12.2.3 La commande lpq

La commande `lpq` permet d'afficher la file d'impression (spool) et connaître les numéros de "job" :

```
$ lpq
HP1200 is ready and printing
Rank    Owner   Job     File(s)                         Total Size
active  alex    12      fstab                           1024 bytes
1st     root    13      fstab                           1024 bytes
```

Pour voir la file d'attente d'une imprimante en particulier.

```
$ lpq -P nom_imprimante
```

12.2.4 La commande lprm

La commande `lprm` permet de supprimer une impression de la file d'attente à l'aide de son numéro "job".

```
$ lprm num_job
```

Pour supprimer toute la file d'attente (cette action n'est permise que pour le compte root).

```
# lprm -
```

12.2.5 La commande lpstat

La commande `lpstat` permet d'obtenir diverses statistiques, comme par exemple :

- La liste des imprimantes installées

```
$ lpstat -a
```

L'imprimante définie par défaut :

```
$ lpstat -d
```

12.2.6 La commande lpc

La commande `lpc` permet d'obtenir des informations sur l'état des imprimantes et files d'attentes.

```
$ lpc status
```

Exemple :

```
$ lpstat -s
destination système par défaut : hp-LaserJet-1010
périphérique pour hp-LaserJet-1010 : usb://HP/LaserJet%201010
périphérique pour hp-LaserJet-1012 : usb://HP/LaserJet%201012
périphérique pour Photosmart-C4200-series : usb://HP/Photosmart%20C4200%20

$ lpstat -d
destination système par défaut : hp-LaserJet-1010

$ lp texte.original
l'identifiant de la requête est hp-LaserJet-1010-52 (1 fichier(s))

$ lpq
hp-LaserJet-1010 n'est pas prêt
Classmt    Proprio   Tâche     Fichier(s)       Taille totale
1st         pat         52      texte.original        2048 octets

$ lpc status
hp-LaserJet-1010:
        l'imprimante correspond au périphérique « usb », débit -1
        la mise en file d'attente est activée
        l'impression est désactivée
        1 entrées
        daemon présent
hp-LaserJet-1012:
        l'imprimante correspond au périphérique « usb », débit -1
        la mise en file d'attente est activée
        l'impression est activée
        aucune entrée
        daemon présent
Photosmart-C4200-series:
        l'imprimante correspond au périphérique « usb », débit -1
        la mise en file d'attente est activée
        l'impression est activée
        aucune entrée
        daemon présent
```

12.3 Installer une "imprimante PDF"

En l'absence d'imprimante physique, il est pratique d'installer une "imprimante PDF". Le résultat des commandes d'impression sera mis dans un fichier .PDF consultable ultérieurement.

Notez bien :

Ceci n'est pas un cours d'administration, cependant voici quelques "recettes de cuisine" qui vont nous permettre de manipuler les commandes étudiées dans ce chapitre.

Vérifier la présence du paquetage `cups` en principe présent à l'installation :

Sous Ubuntu

```
$ sudo apt-cache policy cups | grep Installé
Installé : cups
```

Sous CentOS

```
$ rpm -q cups
cups-1.4.2-50.el6_4.5.x86_64
```

- Télécharger et installer le paquetage `cups-pdf` :

Sous Ubuntu

```
$ sudo apt-get install cups-pdf
```

Sous CentOS

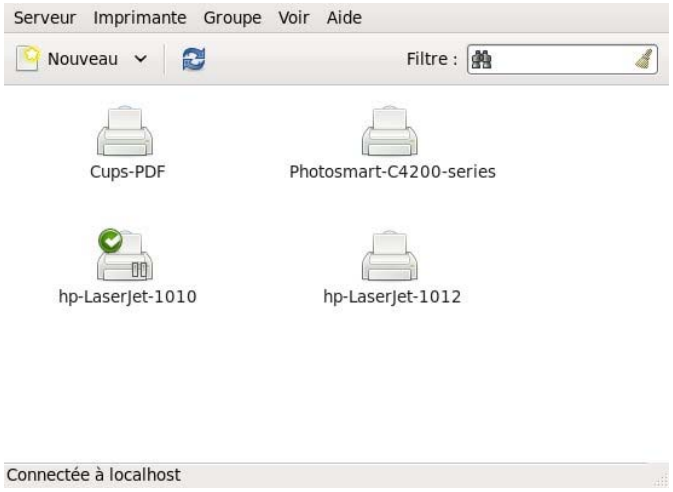
```
# yum install cups-pdf
```

Si on demande une confirmation avant l'installation, tapez O ou Y (pour "Oui") comme indiqué.

On va ensuite redémarrer le service `cups`.

```
# /etc/init.d/cups restart
Arrêt de cups :                                    [  OK  ]
Démarrage de cups :                                [  OK  ]
```

Une fois installée, sous CentOS, on trouvera l'imprimante Cups-PDF dans Système>Administration>imprimante, fenêtre de gestion des imprimantes.



L'imprimante (convertisseur PDF) est installée, l'imprimante s'appelle ici Cups-PDF et les documents produits sont placés dans un dossier « PDF » de votre dossier personnel, dans notre exemple sur le bureau, c'est-à-dire le répertoire $HOME/Bureau.

- Vérification des files d'attentes d'impression

```
$ lpstat -s
destination système par défaut : hp-LaserJet-1010
périphérique pour Cups-PDF : cups-pdf:/
périphérique pour hp-LaserJet-1010 : usb://HP/LaserJet%201010
périphérique pour hp-LaserJet-1012 : usb://HP/LaserJet%201012
périphérique pour Photosmart-C4200-series : usb://HP/Photosmart%20C4200%20
```

- Lancement de l'impression du fichier "texte"

```
$ lp -d Cups-PDF texte
l'identifiant de la requête est Cups-PDF-53 (1 fichier(s))
```

- Vérification

```
$ ls -l Bureau
...
-rw-------. 1 pat gp1  232337 18 avril 00:24 texte.pdf
...
$  evince  texte.pdf    (si  on  est  en  environnement  graphique
```

13. OUTILS DE MANIPULATION DES DONNEES

OBJECTIFS

- Traduire les caractères dans un fichier
- Trier les fichiers
- Réunir, coller des fichiers
- Comparer les fichiers

13.1 Traduction de caractères : tr

13.1.1 La commande tr

Permet de substituer, traduire, supprimer des caractères dans les lignes lues par l'entrée standard, le résultat est envoyé sur la sortie standard.

```
tr  [-options] 'chaîne1' ['chaîne2']
```

13.1.2 Substitution

```
$ ll|tr [:lower:] [:upper:]
TOTAL 80
-RW-R--R-- 1 USER1 USER1  423 SEP 24 13:33 CLIENTS
-RW-R--R-- 1 USER1 USER1  221 SEP 24 12:22 MODELE
-RW-R--R-- 1 USER1 USER1  142 SEP 24 12:29 NOMS
-RW-R--R-- 1 USER1 USER1   40 SEP 24 13:07 NUMERO1
-RW-R--R-- 1 USER1 USER1   39 SEP 24 13:08 NUMERO2
-RW-R--R-- 1 USER1 USER1   79 SEP 24 12:27 NUMEROS
DRWXRWXR-X 3 USER1 USER1 4096 SEP  6 16:12 REP
DRWXRWXR-X 2 USER1 USER1 4096 SEP  6 15:23 REP2
-RW-R--R-- 1 USER1 USER1  423 SEP 24 11:55 TEXTE
-RW-RW-R-- 1 USER1 USER1  493 SEP 25 11:49 TEXTE2
```

13.1.3 Traduction de caractères par d'autres caractères

```
$ ll|tr ' ' '.'
total.80
-rw-r--r--.1.user1.user1..423.sep.24.13:33.clients
-rw-r--r--.1.user1.user1..221.sep.24.12:22.modele
-rw-r--r--.1.user1.user1..142.sep.24.12:29.noms
-rw-r--r--.1.user1.user1...40.sep.24.13:07.numero1
-rw-r--r--.1.user1.user1...39.sep.24.13:08.numero2
-rw-r--r--.1.user1.user1...79.sep.24.12:27.numeros
drwxrwxr-x.3.user1.user1.4096.sep..6.16:12.rep
drwxrwxr-x.2.user1.user1.4096.sep..6.15:23.rep2
-rw-r--r--.1.user1.user1..423.sep.24.11:55.texte
-rw-rw-r--.1.user1.user1..493.sep.25.11:49.texte2
```

13.1.4 Suppression de caractères consécutifs

```
$ cat texte2
1 Le systeme      UNIX comprend          le Noyau, le Shell, les
programmes
2 utilitaires et les langages :
3       - le noyau planifie les taches ;         gere la memoire,
la
4         structure et les acces aux fichiers.
5       - le Shell est un langage         de commande. Il
interprete
6         les commandes et les execute.
7       - les programmes utilitaires,                encore
appeles outils,
8         representent le                 logiciel utilisateur de
base.
9 N'est-ce pas         tout un programme?...

$ tr -s ' ' <texte2
1 Le systeme UNIX comprend le Noyau, le Shell, les programmes
2 utilitaires et les langages :
3       - le noyau planifie les taches ; gere la memoire, la
4        structure et les acces aux fichiers.
5       - le Shell est un langage de commande. Il interprete
6        les commandes et les execute.
7       - les programmes utilitaires, encore appeles outils,
8        representent le logiciel utilisateur de base.
9 N'est-ce pas tout un programme?...
```

13.1.5 Remplacement de caractères consécutifs

```
$ tr -s ' ' '|' <texte2
1|Le|systeme|UNIX|comprend|le|Noyau,|le|Shell,|les|programmes
2|utilitaires|et|les|langages|:
3       -|le|noyau|planifie|les|taches|;|gere|la|memoire,|la|
4       |structure|et|les|acces|aux|fichiers.
5       -|le|Shell|est|un|langage|de|commande.|Il|interprete
6       |les|commandes|et|les|execute.
7       -|les|programmes|utilitaires,|encore|appeles|outils,|
8       |representent|le|logiciel|utilisateur|de|base.
9|N'est-ce|pas|tout|un|programme?...
```

13.1.6 Suppression de caractères

```
$ tr -d ' ' <texte2
1LesystemeUNIXcomprendleNoyau,leShell,lesprogrammes
2utilitairesetleslangages:
3       -lenoyauplanifielestaches;gerelamemoire,la
4       structureetlesaccesauxfichiers.
5       -leShellestunlangagedecommande.Ilinterprete
6       lescommandesetlesexecute.
7       -lesprogrammesutilitaires,encoreappelesoutils,
8       represententlelogicielutilisateurdebase.
9N'est-cepastoutunprogramme?...
```

13.2 Tri de fichiers : sort

13.2.1 La commande sort

La commande sort permet de :

- Trier un ou plusieurs fichiers.
- Fusionner des fichiers déjà triés.
- Vérifier si un fichier est trié.

13.2.2 Tri de fichiers

```
sort [-option(s)] [-k clé_1] [ ... -k clé_n] [fichier_1]
... [fichier_n]
```

Les options :

-o	Précise le fichier de sortie (défaut : stdout).
-d	Tient compte uniquement des blancs et des caractères alphanumériques.
-f	Les minuscules sont transformées en majuscules avant le tri.
-i	Ignore les caractères non affichables.
-n	Clé numérique.
-r	Inverse le résultat du tri (inverse de l'ordre alphabétique, numérique ou alphanumérique).
-u	Si plusieurs lignes ont la même clé de tri seule la première est conservée.
-t	Précise le caractère séparateur de champs.
-c	Vérifie si le fichier en entrée est trié. N'effectue pas de tri.
-m	Fusionne des fichiers déjà triés. N'effectue pas de tri.

- La commande `sort` trie le contenu d'un ou plusieurs fichiers en fonction d'une ou de plusieurs clés de tri.
  Si aucun fichier n'est donné en argument, c'est l'entrée standard qui est prise par défaut.
  Si plusieurs fichiers sont cités, la commande `sort` effectue la concaténation des fichiers avant d'en faire le tri, si bien que :

```
sort fic1 fic2 fic3
```

  Donne le même résultat que :

```
cat fic1 fic2 fic3 | sort
```

La commande `sort` découpe en champs chaque ligne du ou des fichiers.
Un champ est une chaîne de caractères encadrée d'espaces ou d'un autre séparateur précisé par l'option `-t`.

La clé de tri est composée :

- D'un ou plusieurs champs.
- D'une ou plusieurs parties de champ.

Si la clé de tri n'est pas définie, c'est la ligne complète qui est utilisée.

```
$ ll|sort -k 5,5n -k 8,8
total 96
-rw-r--r-- 1 user1 user1   39 sep 24 13:08 numero2
-rw-r--r-- 1 user1 user1   40 sep 24 13:07 numero1
-rw-rw-r-- 1 user1 user1   71 sep 25 12:47 liste1
-rw-r--r-- 1 user1 user1   79 sep 24 12:27 numeros
-rw-rw-r-- 1 user1 user1   81 sep 25 12:48 liste2
-rw-r--r-- 1 user1 user1  142 sep 24 12:29 noms
-rw-r--r-- 1 user1 user1  221 sep 24 12:22 modele
-rw-r--r-- 1 user1 user1  423 sep 24 11:55 texte
-rw-r--r-- 1 user1 user1  423 sep 24 13:33 clients
-rw-rw-r-- 1 user1 user1  493 sep 25 11:49 texte2
drwxrwxr-x 2 user1 user1 4096 sep  6 15:23 rep2
drwxrwxr-x 3 user1 user1 4096 sep  6 16:12 rep
```

Le tri est effectué sur la taille et l'heure de modification des fichiers.
Le champ 5 est défini comme numérique.

```
$ ls -l | sort -k 1.5,1.7
total 96
-rw-r--r-- 1 user1 user1  142 sep 24 12:29 noms
-rw-r--r-- 1 user1 user1  221 sep 24 12:22 modele
-rw-r--r-- 1 user1 user1   39 sep 24 13:08 numero2
-rw-r--r-- 1 user1 user1   40 sep 24 13:07 numero1
-rw-r--r-- 1 user1 user1  423 sep 24 11:55 texte
-rw-r--r-- 1 user1 user1  423 sep 24 13:33 clients
-rw-r--r-- 1 user1 user1   79 sep 24 12:27 numeros
-rw-rw-r-- 1 user1 user1  493 sep 25 11:49 texte2
-rw-rw-r-- 1 user1 user1   71 sep 25 12:47 liste1
-rw-rw-r-- 1 user1 user1   81 sep 25 12:48 liste2
drwxrwxr-x 2 user1 user1 4096 sep  6 15:23 rep2
drwxrwxr-x 3 user1 user1 4096 sep  6 16:12 rep
```

La clé de tri commence au 5ème caractère du 1er champ : 1.5
et se termine au 7ème caractère du 1er champ : 1.7.

13.2.3 Fusion de fichiers

sort -m fichier1 … fichiern

Avec l'option -m la commande sort fait la fusion des fichiers déjà triés cités en argument.
Elle n'effectue aucun tri.

```
$ cat liste1_trie
arch
cat
vi
ypdomainname
zcat
```

```
$ cat liste2_trie
a2p
ctie
nano
over
pinfo
```

```
$ sort -m liste1_trie liste2_trie
a2p
arch
cat
ctie
nano
over
pinfo
vi
ypdomainname
zcat
```

13.2.4 Vérification

sort -c fichier

Vérifie si le fichier donné en argument est trié.

S'il est trié, le code de retour est 0.

S'il n'est pas trié, un message d'erreur est affiché et le code de retour est 1.

```
$ cat liste1
usleep
cat
awk
bash
cut
umount
vi
arch
ypdomainname
view
zcat
basename
```

```
$ sort -c liste1 ; echo $?
sort: liste1:2: désordre: cat
1

$ sort -c liste1_trie ; echo $?
0
```

13.2.5 Trier des nombres

Le tri de nombres est un peu particulier. En effet, la commande `sort` ne distingue pas si les caractères sont des nombres et va donc par défaut les trier par ordre alphabétique.

```
$ cat nombres
36
16
42
129
27
364
```

```
$ sort nombres
129
16
27
36
364
42
```

Selon la table ASCII ces nombres sont bien triés. Tout ce qui commence par 1 est en premier, puis vient ce qui commence par 2 et ainsi de suite. Ce que nous lisons comme des nombres est interprété comme de simples chaines de caractères.

Par conséquent, le "mot" 129 précède le "mot" 42 alors que nous nous attendions à l'inverse.

L'option -n permet de trier en considérant le texte comme des nombres.

```
$ sort -n nombres
16
27
36
42
129
364
```

Cette fois, l'ordre de croissance des nombres est bien respecté.

13.3 Suppression de colonnes : colrm

colrm n1 n2

Suppression de la colonne n1 à la colonne n2, le reste est conservé et envoyé sur la sortie standard.

Suppression de la 11ème à la 42ème colonnes :

```
$ ls -l rep1
total 1332
-rw-r--r--. 1 pat gp1    148  7 janv.  2016 fic
-rw-r--r--. 1 pat gp1 306489 26 août  10:39 fic_30.pdf
-rw-r--r--. 1 pat gp1 172018 26 août  10:39 fic_31.pdf
-rw-r--r--. 1 pat gp1 289930 26 août  10:39 fic_32.pdf
-rw-r--r--. 1 pat gp1 289930 26 août  10:39 fic_34.pdf
-rw-r--r--. 1 pat gp1 289930 26 août  10:39 fic_3.pdf
-rw-r--r--. 1 pat gp1    676  7 janv.  2016 sauve.log
-rwxr--r--. 1 pat gp1    177  7 janv.  2016 sauve.sh
```

```
$ ls -l rep1 | colrm 13 43
total 1332
-rw-r--r--.  fic
-rw-r--r--.  fic_30.pdf
-rw-r--r--.  fic_31.pdf
-rw-r--r--.  fic_32.pdf
-rw-r--r--.  fic_34.pdf
-rw-r--r--.  fic_3.pdf
-rw-r--r--.  sauve.log
-rwxr--r--.  sauve.sh
```

13.4 Extraction de champs : cut

cut [-d séparateur] [-c | -b | -f] liste

Extrait et envoie sur la sortie standard les caractères, les octets, les champs définis par la liste. L'option `"-d séparateur"` peut être utilisée pour définir le séparateur de champ à utiliser (par défaut, le séparateur est `"tabulation"`).

Extraction des colonnes 1 à 10 et 43 jusqu'à la fin :

```
$ ls -l rep1
total 1332
-rw-r--r--. 1 pat gp1    148  7 janv.  2016 fic
-rw-r--r--. 1 pat gp1 306489 26 août  10:39 fic_30.pdf
-rw-r--r--. 1 pat gp1 172018 26 août  10:39 fic_31.pdf
-rw-r--r--. 1 pat gp1 289930 26 août  10:39 fic_32.pdf
-rw-r--r--. 1 pat gp1 289930 26 août  10:39 fic_34.pdf
-rw-r--r--. 1 pat gp1 289930 26 août  10:39 fic_3.pdf
-rw-r--r--. 1 pat gp1    676  7 janv.  2016 sauve.log
-rwxr--r--. 1 pat gp1    177  7 janv.  2016 sauve.sh
```

```
$ ls -l rep1 | cut -c1-11,44-
total 1332
-rw-r--r--. fic
-rw-r--r--. fic_30.pdf
-rw-r--r--. fic_31.pdf
-rw-r--r--. fic_32.pdf
-rw-r--r--. fic_34.pdf
-rw-r--r--. fic_3.pdf
-rw-r--r--. sauve.log
-rwxr--r--. sauve.sh
```

Ici `cut` affiche les colonnes 1 à 11 et de la colonne 44 à la fin de la ligne.

13.4.1 Couper selon un délimiteur

Par défaut, le délimiteur de champs est la tabulation. Parfois, le fichier à traiter présente d'autres types de séparateurs.

Exemple avec un extrait du fichier des utilisateurs /etc/passwd, ici le séparateur est " :".

```
$ more fic
pat:x:501:501::/home/pat:/bin/bash
user1:x:505:504::/home/user1:/bin/bash
toto:x:506:506::/home/toto:/bin/bash
user2:x:508:504::/home/user2:/bin/bash
user3:x:509:504::/home/user3:/bin/bash
```

```
$ cut -d : -f 1,6 fic
pat:/home/pat
user1:/home/user1
toto:/home/toto
user2:/home/user2
user3:/home/user3
```

Ici cut affiche le champ 1 (nom de l'utilisateur) et le champ 6 (répertoire d'accueil), notez que le séparateur de champs est conservé.

13.5 Afficher le début d'un fichier : head

head [-n valeur|-c valeur] [fichier …]

Affiche sur la sortie standard l'en-tête du ou des fichiers donnés en arguments (par défaut, l'entrée standard). Lorsqu'il y a plusieurs fichiers leurs noms est indiqué en en-tête.

S'il n'y a pas d'option, affichage des 10 premières lignes (-n 10). Avec une valeur négative, il y a affichage de toutes les lignes sauf des n dernières.

Si l'option -c est utilisée, affichage des n premiers caractères du fichier. Avec une valeur négative, il y a affichage de tout sauf des n derniers caractères.

```
$ cat texte
1 Le systeme UNIX comprend le Noyau, le Shell, les programmes
2 utilitaires et les langages :
3       - le noyau planifie les taches ; gere la memoire, la
4         structure et les acces aux fichiers.
5       - le Shell est un langage de commande. Il interprete
6         les commandes et les execute.
7       - les programmes utilitaires, encore appeles outils,
8         representent le logiciel utilisateur de base.
9 N'est-ce pas tout un programme?...
```

```
$ head -n 3 texte
1 Le systeme UNIX comprend le Noyau, le Shell, les programmes
2 utilitaires et les langages :
3       - le noyau planifie les taches ; gere la memoire, la
```

```
$ head -n -3 texte
1 Le systeme UNIX comprend le Noyau, le Shell, les programmes
2 utilitaires et les langages :
3       - le noyau planifie les taches ; gere la memoire, la
4         structure et les acces aux fichiers.
5       - le Shell est un langage de commande. Il interprete
6         les commandes et les execute.
```

```
$ date
lun sep 24 17:28:22 CEST 2016
```

```
$ date | head -c 10
lun sep 24
```

```
$ date | head -c -9
lun sep 24 17:28:22 CEST
```

13.6 Afficher la fin d'un fichier : tail

tail [-n valeur|-c valeur] [fichiers …]

Affiche sur la sortie standard la fin du ou des fichiers donnés en arguments (par défaut, l'entrée standard). Lorsqu'il y a plusieurs fichiers leur nom est indiqué en en-tête.

S'il n'y a pas d'option, affichage des 10 dernières lignes (-n 10).

```
$ cat texte
1 Le systeme UNIX comprend le Noyau, le Shell, les programmes
2 utilitaires et les langages :
3       - le noyau planifie les taches ; gere la memoire, la
4         structure et les acces aux fichiers.
5       - le Shell est un langage de commande. Il interprete
6         les commandes et les execute.
7       - les programmes utilitaires, encore appeles outils,
8         representent le logiciel utilisateur de base.
9 N'est-ce pas tout un programme ?...
```

```
$ tail -n 3 texte
7       - les programmes utilitaires, encore appeles outils,
8         representent le logiciel utilisateur de base.
9 N'est-ce pas tout un programme ?...
```

Si l'option -c est utilisée, il y a affichage des n derniers caractères du fichier.

```
$ date
lun sep 24 17:28:22 CEST 2016
```

```
$ date | tail -c 5
2016
```

Attention piège :

```
$ date | tail -c 4
016
```

Cas particulier : affichage en temps réel

tail -f fichier

L'option -f permet d'afficher les nouvelles lignes qui s'ajoutent au fichier au fur et à mesure de leur arrivée. Cette commande est généralement utilisée pour suivre l'évolution de fichiers de log.

La sortie de cette commande se fait par <Ctrl><C>.

Par défaut, tail -f recherche les nouveaux changements dans le fichier toutes les secondes. L'option -s suivie d'un nombre permet de préciser le temps entre deux vérifications.

Mise en évidence :

Il n'est pas aisé de générer des événements pour faire évoluer les logs système. Voici une astuce pour mettre en évidence le fonctionnement de tail -f.

Saisissez le programme prog.log suivant et rendez-le exécutable. Créez un fichier fic.log avec une première ligne. Lancez tail -f fic.log, puis à partir d'une autre fenêtre de Shell (sur le même utilisateur) lancez le programme prog.log. Observez l'évolution de tail -f.

```
$ more prog_log
for i in 1 2 3 4 5 6
do
  echo " > entrée $i" >>fic.log
  sleep 2
done
```

```
$ more fic.log
Ceci est le fichier fic.log
```

```
$ chmod u+x prog_log ; ./prog.log
```

```
$ tail -f fic.log
Ceci est le fichier fic.log
 > entrée 1
 > entrée 2
 > entrée 3
 > entrée 4
^C
```

13.7 Affichage de chaînes imprimables de fichiers : strings

strings [-options] [fichiers]

La commande strings envoie sur la sortie standard les chaînes de caractères imprimables du ou des fichiers cités.

Elle permet, aussi la recherche dans des fichiers non ASCII, cas typique des exécutables.

Les options :

-a	Scrute entièrement le fichier.
-n <valeur>	Recherche les chaînes de caractères imprimables suivies d'un caractère non imprimable d'une taille d'au moins <valeur> caractères (par défaut : 4).
-f	Fait précéder les enregistrements trouvés du nom du fichier.
-t o|d|x	Fait précéder les enregistrements de leur position dans le fichier en octal (o), en décimal (d) ou hexadécimal (x).

```
$ strings -a -f -n 40 -t x /bin/sleep
/bin/sleep:    2c41 Pause for NUMBER seconds.  SUFFIX may be `s'for seconds (the default),
/bin/sleep:    2c89 `m' for minutes, `h' for hours or `d' for days.  Unlike most implementations
/bin/sleep:    2cd6 that require NUMBER be an integer, here NUMBER may be an arbitrary floating
/bin/sleep:    2d34       --help     display this help and exit
/bin/sleep:    2d64       --version  output version information and exit
/bin/sleep:    31fd This is free software.  You may redistribute copies of it under the terms of
/bin/sleep:    324a the GNU General Public License <http://www.gnu.org/licenses/gpl.html>.
/bin/sleep:    3291 There is NO WARRANTY, to the extent permitted by law.
/bin/sleep:    3440 Copyright %s %d Free Software Foundation, Inc.
...
```

13.8 Supprimer les lignes identiques : uniq

uniq [-c|-u|-d] [fichier_entrée] [fichier_sortie]

Permet de supprimer des lignes identiques consécutives du fichier cité en argument (par défaut l'entrée standard).

Le résultat est envoyé dans le fichier de sortie cité (par défaut la sortie standard).

Les options :

-c	Affiche le nombre d'occurrences de chaque ligne.
-u	N'afficher que les lignes uniques.
-d	N'afficher que les lignes présentes en double.

```
$ more clients
Aboaf Maurice 244 748
Adda Jen 539 234
Allo Jean-Pierre 255 98
Allo Jean-Pierre 255 98
Allo Jean-Pierre 255 98
Allo Jean-Pierre 255 98
Dupont Jean 111 112
Dupont Jean 111 112
Bernard Jean-Paul 234 567
Chasserat Paul 245 178
Cousin Pascal 222 222
Cousin Pascal 222 222
Cousin Pascal 222 222
Froideceaux Michel 252 423
Gros Lucien 212 121
Gros Lucien 212 121
```

```
$ uniq clients
Aboaf Maurice 244 748
Adda Jen 539 234
Allo Jean-Pierre 255 98
Dupont Jean 111 112
Bernard Jean-Paul 234 567
Chasserat Paul 245 178
Cousin Pascal 222 222
Froideceaux Michel 252 423
Gros Lucien 212 121
```

```
$ uniq -d clients
Allo Jean-Pierre 255 98
Dupont Jean 111 112
Cousin Pascal 222 222
Gros Lucien 212 121
```

```
$ uniq -u clients
Aboaf Maurice 244 748
Adda Jen 539 234
Bernard Jean-Paul 234 567
Chasserat Paul 245 178
Froideceaux Michel 252 423
```

```
$ uniq -c clients
      1 Aboaf Maurice 244 748
      1 Adda Jen 539 234
      4 Allo Jean-Pierre 255 98
      2 Dupont Jean 111 112
      1 Bernard Jean-Paul 234 567
      1 Chasserat Paul 245 178
      3 Cousin Pascal 222 222
      1 Froideceaux Michel 252 423
      2 Gros Lucien 212 121
```

13.9 Fusion de fichiers : paste

La commande paste fusionne les fichiers cités en argument par concaténation ligne à ligne séparées par défaut par une tabulation sinon ce sont les séparateurs cités dans l'option -d.

paste [-d 'séparateurs'] [fichiers …]

```
$ more noms
Aboaf Maurice
Adda Jen
Allo Jean-Pierre
Dupont Jean
Bernard Jean-Paul
Chasserat Paul
Cousin Pascal
Froideceaux Michel
```

```
$ more numero1
244
539
255
111
234
245
222
252
```

```
$ more numero2
748
234
98
112
567
178
222
423
```

```
$ paste -d '/-' noms numero1 numero2
Aboaf Maurice/244-748
Adda Jen/539-234
Allo Jean-Pierre/255-98
Dupont Jean/111-112
Bernard Jean-Paul/234-567
Chasserat Paul/245-178
Cousin Pascal/222-222
Froideceaux Michel/252-423
```

13.10 Concaténation de fichiers : cat

Ne pas confondre les commandes paste et cat.

La commande cat permet de mettre bout à bout, concaténer, plusieurs fichiers. Le résultat est envoyé sur la sortie standard. Si aucun fichier n'est cité en sortie affichage à l'écran, si aucun fichier d'entrée n'est cité le fichier d'entré standard est le clavier.

cat [fichiers1] [fichiers2] [fichiers3] [fichiers_sortie]

$ more noms1 Aboaf Maurice Adda Jen Allo Jean-Pierre	**$ more noms2** Dupont Jean Bernard Jean-Paul	**$ more noms3** Chasserat Paul Cousin Pascal Froideceaux Michel

```
$ cat noms1 noms2 noms3 >noms
```

```
$ more noms
Aboaf Maurice
Adda Jen
Allo Jean-Pierre
Dupont Jean
Bernard Jean-Paul
Chasserat Paul
Cousin Pascal
Froideceaux Michel
```

Il est courant de voir la commande cat utilisée pour afficher le contenu d'un fichier. Ceci ne présente pas d'inconvénient pour de petits fichiers de texte, mais on préférera utiliser more et less qui ont tout de même bien plus de possibilités et de sécurités.

Cat d'un fichier binaire va tenter d'afficher à l'écran des caractères non ASCII, le résultat n'est pas prévisible. A éviter absolument !

13.11 Comparaison de fichiers : cmp

cmp [-b, -s] fichier1 fichier2 [saut1] [saut2]

La commande cmp compare ligne à ligne le contenu de deux fichiers.

Elle s'arrête à la première différence et affiche un message sauf si l'option -s est utilisée.

saut1 et **saut2** indiquent le nombre d'octets à sauter avant de commencer la comparaison.

saut1 pour **fichier1** et **saut2** pour **fichier2** sont des valeurs données en décimal par défaut (possibilité de les citer en hexadécimal 0x…. ou en octal 0…).

Le code de retour est 0 si les fichiers sont identiques et 1 s'ils sont différents.

-b Affiche les octets qui différent

-s N'affiche rien ; fournir seulement le code de retour.

```
$ cat clients1
1-Aboaf Maurice 244 748       <<<
2-Adda Jen 539 234
3-Allo Jean-Pierre 255 98
4-Dupont Jean 111 112         <<<
5-Bernard Jean-Paul 234 567
6-Chasserat Paul 245 178
8-Cousin Pascal 222 222
9-Froideceaux Michel 252 423
10-Gros Lucien 212 121

$ cat clients2
1-Abouf Maurice 244 748       <<<
2-Adda Jen 539 234
3-Allo Jean-Pierre 255 98
4-Dupont Jean 221 112         <<<
5-Bernard Jean-Paul 234 567
6-Chasserat Paul 245 178
7-Martin Pierre 145 865       <<<
8-Cousin Pascal 222 222
9-Froideceaux Michel 252 423
10-Gros Lucien 212 121
```

```
$ cmp clients1 clients2
clients1 clients2 sont différents: octet 4, ligne 1
```

L'option -b donne en plus la valeur en octal et le caractère qui correspondent à la première différence rencontrée.

```
$ cmp -b clients1 clients2
clients1 clients2 diffèrent: octet 4, ligne 1 est 141 a 165 u
```

```
$ cat fic1
78azertyuiop
qsdfghjklm
wxcvbn
```

```
$ cat fic2
1234azertyuiop
qsdfghjklm
wxcvbn
```

```
$ cmp -b fic1 fic2 2 4
```

Dans ce dernier exemple, la commande saute les 2 premiers caractères dans le fichier fic1 et les 4 premiers dans fic2, hormis ces premiers caractères les deux fichiers sont identiques.

13.12 Comparaison de fichiers : diff

La commande diff compare ligne à ligne l'intégralité du contenu de deux fichiers et retourne les différences.

```
diff [-eiwr] fichier1 fichier2
```

Si fichier1 et fichier2 sont des répertoires, il y comparaison des fichiers de ces répertoires sans récursivité sauf si l'option -r est utilisée.

Les options :

-e	Comparaison avec édition des directives de modification de l'éditeur ed.
-i	Ignore les minuscules et majuscules lors de la comparaison.
-w	Les espaces et tabulations ne sont pas pris en compte.
-r	Récursivité lors de la comparaison de répertoires.

13.12.1 Comparaison de fichiers ordinaires

```
$ cat clients1
1-Aboaf Maurice 244 748
2-Adda Jen 539 234
3-Allo Jean-Pierre 255 98
4-Dupont Jean 111 112
5-Bernard Jean-Paul 234 567
6-Chasserat Paul 245 178
8-Cousin Pascal 222 222
9-Froideceaux Michel 252 423
10-Gros Lucien 212 121
```

```
$ cat clients2
1-Abouf Maurice 244 748
2-Adda Jen 539 234
3-Allo Jean-Pierre 255 98
4-Dupont Jean 221 112
5-Bernard Jean-Paul 234 567
6-Chasserat Paul 245 178
7-Martin Pierre 145 865
8-Cousin Pascal 222 222
9-Froideceaux Michel 252 423
10-Gros Lucien 212 121
```

```
$ diff clients1 clients2
1c1
< 1-Aboaf Maurice 244 748
---
> 1-Abouf Maurice 244 748
4c4
< 4-Dupont Jean 111 112
---
> 4-Dupont Jean 221 112
6a7
7-Martin Pierre 145 865
```

Pour que le 1er fichier devienne identique au 2ème il faut :

- Remplacer la 1ière ligne de clients1 par la 1ière ligne de clients2.
- Remplacer la 4ème ligne de clients1 par la 4ème ligne de clients2.
- Rajouter après la 6ème ligne de clients1 la 7ème ligne de clients2.

```
$ diff -e clients1 clients2
6a
7-Martin Pierre 145 865
.
4c
4-Dupont Jean 221 112
.
1c
1-Abouf Maurice 244 748
.
```

Le résultat peut être mis directement dans le fichier de directives de l'éditeur silencieux sed.

13.12.2 Comparaison de répertoires

Si les deux fichiers source et cible sont des répertoires, diff compare les fichiers correspondant dans les deux répertoires, dans l'ordre alphabétique. Cette comparaison n'est pas récursive, à moins d'employer l'option -r.

```
$ ll rep1
total 104
-rw-r--r-- 1 user1 user1 423 sep 26 15:52 clients
-rw-rw-r-- 1 user1 user1  24 sep 26 15:54 fic1                <<<
-rw-rw-r-- 1 user1 user1  71 sep 26 15:52 liste1
-rw-rw-r-- 1 user1 user1  30 sep 26 15:52 liste1_trie
-rw-rw-r-- 1 user1 user1  81 sep 26 15:52 liste2
-rw-rw-r-- 1 user1 user1  25 sep 26 15:52 liste2_trie
-rw-r--r-- 1 user1 user1 221 sep 26 15:52 modele
-rw-r--r-- 1 user1 user1 142 sep 26 15:52 noms                <<<
-rw-r--r-- 1 user1 user1  60 sep 26 15:57 numero1             <<<
-rw-r--r-- 1 user1 user1  39 sep 26 15:52 numero2
-rw-r--r-- 1 user1 user1  79 sep 26 15:52 numeros
-rw-r--r-- 1 user1 user1 423 sep 26 15:52 texte
-rw-rw-r-- 1 user1 user1 493 sep 26 15:52 texte2
```

```
$ ll rep2
total 92
-rw-r--r-- 1 user1 user1 423 sep 26 15:53 clients
-rw-rw-r-- 1 user1 user1  71 sep 26 15:53 liste1
-rw-rw-r-- 1 user1 user1  30 sep 26 15:53 liste1_trie
-rw-rw-r-- 1 user1 user1  81 sep 26 15:53 liste2
-rw-rw-r-- 1 user1 user1  25 sep 26 15:53 liste2_trie
-rw-r--r-- 1 user1 user1 221 sep 26 15:53 modele
-rw-r--r-- 1 user1 user1  40 sep 26 15:53 numero1             <<<
-rw-r--r-- 1 user1 user1  39 sep 26 15:53 numero2
-rw-r--r-- 1 user1 user1  79 sep 26 15:53 numeros
-rw-r--r-- 1 user1 user1 423 sep 26 15:53 texte
-rw-rw-r-- 1 user1 user1 493 sep 26 15:53 texte2
-rw-rw-r-- 1 user1 user1   0 sep 26 15:57 toto                <<<
```

```
$ diff rep1 rep2
Seulement dans rep1: fic1
Seulement dans rep1: noms
diff rep1/numero1 rep2/numero1
11,15d10
< 125
< 785
< 267
< 985
< 762
Seulement dans rep2: toto
$
```

13.13 Différences entre 3 fichiers : diff3

La commande diff3 permet de comparer trois fichiers et de proposer (par défaut) les modifications à apporter pour qu'ils deviennent identiques.

diff3 [-options] fichier1 fichier2 fichier3

```
$ cat clients1
1-Aboaf Maurice 244 748
2-Adda Jen 539 234
3-Allo Jean-Pierre 255 98
4-Dupont Jean 111 112
5-Bernard Jean-Paul 234 567
6-Chasserat Paul 245 178
8-Cousin Pascal 222 222
9-Froideceaux Michel 252 423
10-Gros Lucien 212 121

$ cat clients2
1-Abouf Maurice 244 748
2-Adda Jen 539 234
3-Allo Jean-Pierre 255 98
4-Dupont Jean 221 112
5-Bernard Jean-Paul 234 567
6-Chasserat Paul 245 178
7-Martin Pierre 145 865
8-Cousin Pascal 222 222
9-Froideceaux Michel 252 423
10-Gros Lucien 212 121

$ cat clients3
1-Aboaf Maurice 244 748
2-Addada Jen 539 234
3-Allo Jean-Pierre 255 98
4-Dupont Jean 111 112
5-Bernard Jean-Paul 234 567
6-Chasserat Paul 245 178
9-Froideceaux Michel 252 751
10-Gros Lucien 212 121
11-Maigre gaston 223 141
```

```
$ diff3 clients1 clients2 clients3
====
1:1,2c
  1-Aboaf Maurice 244 748
  2-Adda Jen 539 234
2:1,2c
  1-Abouf Maurice 244 748
  2-Adda Jen 539 234
3:1,2c
  1-Aboaf Maurice 244 748
  2-Addada Jen 539 234
====2
1:4c
3:4c
  4-Dupont Jean 111 112
2:4c
  4-Dupont Jean 221 112
====
1:7,8c
  8-Cousin Pascal 222 222
  9-Froideceaux Michel 252 423
2:7,9c
  7-Martin Pierre 145 865
  8-Cousin Pascal 222 222
  9-Froideceaux Michel 252 423
3:7c
  9-Froideceaux Michel 252 751
====3
1:9a
2:10a
3:9c
  11-Maigre gaston 223 141
```

Remplacer les lignes :

1. *1 et 2 de **clients1** par les lignes 1 et 2 de **clients3**.*
2. *1 et 2 **clients2** par les lignes 1 et 2 de **clients3**.*

*Remplacer la ligne 4 de **clients1** et **clients3** par la ligne 4 de **clients2***

Remplacer :

- *Les lignes 7 et 8 de **clients1** par la ligne 7 de **clients3**.*
- *Les lignes de 7 à 9 de **clients2** par la ligne 7 de clients3.*

*Ajouter après la ligne 9 de **clients1** et la ligne 10 de **clients2**, le contenu de la ligne 9 de **clients3**.*

Avec l'option -e, les informations retournées sont les directives de l'éditeur ed à appliquer sur fichier1 pour qu'il devienne l'identique de fichier3. Ce qui revient à faire :

diff -e fichier1 fichier3

```
$ diff3 -e clients1 clients2 clients3
9a
11-Maigre gaston 223 141
.
7,8c
9-Froideceaux Michel 252 751
.
1,2c
1-Aboaf Maurice 244 748
2-Addada Jen 539 234
```

13.14 Comparer des fichiers triés : comm

La commande comm permet de comparer des fichiers triés.

comm [-1,-2,-3] fichier1 fichier2

L'affichage par défaut se fait sur trois champs :

- Le champ 1 contient les lignes uniques de fichier1.
- Le champ 2 contient les lignes uniques de fichier2.
- Le champ 3 contient les lignes communes aux deux fichiers.

Les options :

-1	Permet de supprimer le champ 1 en sortie.
-2	Permet de supprimer le champ 2 en sortie.
-3	Permet de supprimer le champ 3 en sortie.

```
$ comm commande1 commande2
                        G.T.M.:ciment:64:15/09/07
                        SPIE ET CIE:platre:24:15/09/07
                        SPAC ET CIE:ardoises:70:15/09/07
                        LA MAISON MOD:briques:65:15/09/07
            La HUTTE:ardoises:97:15/09/07
                        MAISON DE MAC:tuiles:270:15/09/07
                        HERNANDEZ ET C:ciment:64:15/09/07
MERREIRA:ardoises:24:15/09/07
                        SPINELLI:briques:24:15/09/07
                        ANTONELLI:platre:48:15/09/07
                        SEONARD:tuiles:20:15/09/07
HERNANDO:ciment:64:16/15/07
            MARTIN:platre:25:15/09/07
            HERNANDO:ciment:64:16/15/07
                        PERREIRA:ardoises:24:16/15/07
            GRANDS TRAVA:ciment:100:11/15/07
LEONARD:tuiles:20:16/15/07
GRANDS TRAVA:ciment:100:11/15/07
```

13.15 Comparer et fusionner : sdiff

La commande sdiff compare et fusionne sur la sortie standard, ligne à ligne, les fichiers cités en argument.

sdiff fichier1 fichier2

Elle signale les différences par les caractères :

- "|" pour indiquer que l'enregistrement est différent dans fichier2 par rapport à fichier1.
- "<" pour indiquer que l'enregistrement correspondant est absent dans fichier2.
- ">" pour indiquer que l'enregistrement correspondant est absent dans fichier1.

```
$ sdiff clients1 clients3
1-Aboaf Maurice 244 748         1-Aboaf Maurice 244 748
2-Adda Jen 539 234                  | 2-Addada Jen 539 234
3-Allo Jean-Pierre 255 98             3-Allo Jean-Pierre 255 98
4-Dupont Jean 111 112                 4-Dupont Jean 111 112
5-Bernard Jean-Paul 234 567           5-Bernard Jean-Paul 234 567
6-Chasserat Paul 245 178              6-Chasserat Paul 245 178
8-Cousin Pascal 222 222       | 9-Froideceaux Michel 252 751
9-Froideceaux Michel 252 423        <
10-Gros Lucien 212 12                 10-Gros Lucien 212 121
                                    > 11-Maigre gaston 223 141
$
```