2021/2022

Département Informatique

GUIDE
UNIX UTILISATEURS

F. GUILLET

- Troisième année -

Reproduction interdite sans autorisation de l'auteur et de l'école

École polytechnique de l'université de Nantes
Rue C. Pauc - La Chantrerie - BP 50609 - 44306 NANTES Cédex 3 Tél : 02 40 68 32 00 Fax : 02 40 68 32 32

Chapitre I Généralités

Ce document propose une initiation au système d'exploitation Unix, niveau utilisateur. On y trouvera un historique, une description du système, un inventaire des principales commandes, un résumé des commandes de l'éditeur vi, une description de la programmation en Bourne shell, les possibilités des commandes orientés réseau, et enfin des travaux pratiques et leur corrigé.

1 Historique :

- **Mi-1960** : Projet ambitieux entre le **M.I.T**., **BELL** laboratories (communs à **AT&T** et **Western Electric**), et **General Electric** pour un *système d'exploitation multi-utilisateurs* : **MULTICS** (MULtiplexed Information and Computing Service) qui devait être capable de gérer une ville entière équipée de terminaux!
- **1968** : Le projet prenant un retard important, BELL laboratories quitte le consortium.
- **1969** : **K. Thompson** (BELL laboratories) développe sur un **PDP7** (DEC) un système d'exploitation *mono-utilisateur* qui comporte un *assembleur*, un *interpréteur de commandes* et un *système de fichiers* rudimentaires. Par dérision, il le baptise **UNIX**. Il n'imaginait pas alors l'avenir de son système.
- **1970** : Le système est **amélioré** dans une seconde édition sur un **PDP 11/20** apportant : traitement de texte, tubes de communication (pipelines), et surtout un système *multi-utilisateurs*.
- **1973 :** Unix devient **portable** sur tout type de machine : en collaboration avec **D. Ritchie**, l'un des inventeurs du langage C avec Kernighan, Unix est réécrit en langage **C** à 95% (+5% d'assembleur).
- **1975 :** Des licences sont accordées aux universités américaines (gestion et enseignement) qui l'améliorent. En particulier l'Université de Berkeley (qui produisit l'éditeur **vi**).

  A partir de cette date trois grandes familles d'UNIX ont commencé à se former pour arriver en :
- **1987** : à **UNIX SYSTEM V release 3**[1], **UNIX BSD 5.0**[2], et **OSF/1**[3].

D'autres constructeurs offrent, sur leurs machines, une version *unix-like* compatible avec l'une de ces trois familles. C'est le cas de SunOs et Solaris (Sun Microsystems), Xenix (Microsoft), HP-UX (Hewlett-Packard), Ultrix (Digital Equipement Corporation), AIX (IBM), Linux (freeware PC 1991)...

Ces trois dialectes convergent aujourd'hui, dans un effort de normalisation, pour se rejoindre dans la version **4** de **Unix SYSTEM V**.

1. développé à l'UCB (University of California, Berkeley).
2. version industrielle développée par AT&T.
3. version normalisée développée par OSF (Open Software Fondation).

ANNEES	Laboratoires BELL	BERKELEY	AUTRES
1974	Releases annuels		
1975	Licences commerciales version 6		
1977	Version 7	1.0 BSD Pascal	
1978		2.0 BSD vi	
1979	System III	3.0 BSD demand paging lisp	
1980		4.0 BSD C SHELL	XENIS MICROSOFT UNIPLUS UNISOFT
1981	Release 4	4.1 BSD	
1982	SystemV	4.1 1c BSD	PC/IX IBM
1983		4.2 BSD ingres	ULTRIX DEC
1984	System V, Release 2		

Tableau 1 : Récapitulatif des versions

2 Principales caractéristiques d'Unix

Le système d'exploitation Unix dote les machines qui le supportent de capacités généralement attendues sur de puissants systèmes de calcul :

- *Mutli-utilisateurs* : plusieurs utilisateur peuvent être connectés simultanément.
- *Multitâches* : un environnement en temps partagé permet d'entrelacer l'exécution d'un ensemble de tâches sur la machine.
- *Interactif* : l'utilisateur communique à travers une interface de commande (shell).
- *Portable* : disponible la plupart des machines (du PC au super-calculateur) .
- *Maniable* : possibilité d'écriture de procédures unix.
- Orienté *réseau* d'ordinateurs (communications, partage de fichiers).

3 Rôles du système d'exploitation

Un système d'exploitation est constitué d'un ensemble de programmes dont la fonction est de gérer les différents éléments matériels d'une machine (mémoires, processeurs, disques durs, disquettes, prise réseau...). Il s'agit donc d'une interface entre les applications (partie logique) et le matériel (partie physique).

Les principales fonctions du système d'exploitation unix sont les suivantes ***:***

- ***Gérer la mémoire vive*** : Le système d'exploitation segmente la mémoire en portions qui sont allouées aux applications qui le demandent. Les application peuvent ensuite utiliser cette portion de mémoire qui leur est allouée, mais ne peuvent agir sur les autres portions. Seul le système d'exploitation peut gérer la répartition de cette mémoire et les supports physiques utilisés (mémoire principale, extensions, mémoire virtuelle sur disque). De plus, lorsque une application est temporairement inactive, le système peut transférer automatiquement la portion de mémoire physique utilisée vers dans une zone spéciale du disque dur (swap), puis la rétablir en mémoire lorsque l'application sera à nouveau active.
- ***Gérer l'accès aux périphériques*** : Tout accès à un périphérique (liaison série, disquette...) passe obligatoirement par le système d'exploitation, et ne peut s'effectuer directement. L'accès s'effectue à travers des fichiers spéciaux rattachés à des pilotes de périphériques. Ainsi, le système offre un niveau d'abstraction: les périphériques sont vus par les applications comme des fichiers et sont utilisables comme ceux-ci.
- ***Gérer le(s) disque(s) dur(s)*** : Le système d'exploitation permet une gestion fiable (récupération d'erreurs), sécurisée (droits d'accès, utilisateurs multiples), rapide (accélérée par cache mémoire) du disque dur et un rangement sous forme de fichiers hiérarchisés des données qu'il contient.

- ***Gérer les programmes*** : unix étant multitâches, les applications ont la possibilité d'être exécutée en parallèle (généralement du parallélisme simulé) en se partageant, sans chevauchement, les ressources (cpu, mémoire, .) système. Sous unix, les programmes exécutés en mémoires sont appelés des processus.
- ***Gérer les statistiques de fonctionnement*** : cette fonction permet de récolter des données sur le fonctionnement du système d'exploitation (utilisateur connectés, processus en cours, erreurs...).

4 Organisation d'unix

Comme la majorité des systèmes d'exploitation, unix est organisé en plusieurs couches qui permettent des abstractions successives des couches inférieures, de la structure matérielle vers l'utilisateur.

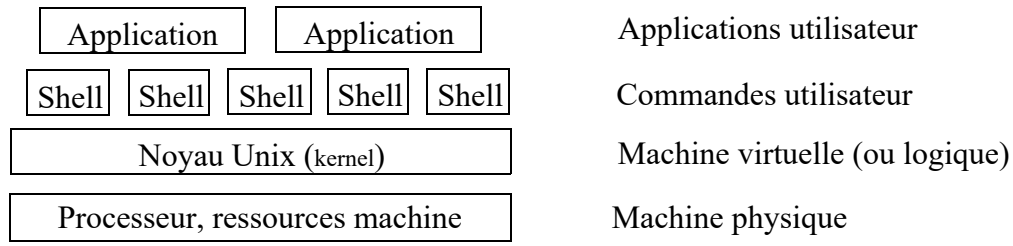

Figure 1 : Structuration en couches

Le noyau Unix (kernel)

C'est un programme qui commande le matériel et qui exécute les fonctions de base concernant les périphériques, les fichiers et le processeur (gestion du temps partagé entre autres). Il est écrit à 95% en langage C et 5% en assembleur (drivers de périphériques).

Il se trouve sous la racine (même organisation hiérarchique du disque que DOS) sous le nom d'UNIX et est résident en mémoire après le chargement.

Le shell

C'est l'interface utilisateur du système Unix. Ce programme permet la communication avec le noyau unix en interprétant les commandes de l'utilisateur, soit depuis le clavier, soit depuis des fichiers de commandes (fichiers de traitement de lots, ou scripts : procédures écrites en langage de commande). Il existe de nombreux types de shells au choix de l'utilisateur. Dans la suite du document, le shell utilisé sera le **BOURNE SHELL** (du nom de son concepteur) noté **sh**.

Les applications complexes

Ce sont des fichiers exécutables répartis dans les différents répertoires sur le disque. Les utilisateurs qui y sont autorisés peuvent les lancer et les utiliser.

5 Accès au système

5.1 Prérequis

il faut que l'administrateur système définisse un **compte utilisateur**, constitué de :

- Un **nom de login** ou nom de connexion. utilisateur, qui est Il s'agit d'une chaîne d'au plus 8 caractères et qui aura préalablement été défini par le gestionnaire du système. Ce nom constitue votre nom d'utilisateur pour le système et est associé à un numéro entier : uid (User IDentity).
- Un **mot de passe** facultatif, que vous aurez déjà défini ou non lors d'une précédente session.
- Un **répertoire racine utilisateur** (**HOME**) ou répertoire de connexion, dans lequel vous serez automatiquement connecté à chaque session.
- Un **groupe** d'autres utilisateurs dans lequel l'utilisateur est inclus (associé à un numéro entier : gid (Group IDentity)).
- Un **shell** par défaut qui sera lancé au début de chaque session et vous permettra de communiquer avec Unix à travers des commandes en mode textuel.

5.2 Connexion et Déconnexion

Pour se connecter au système Unix :

Pour se connecter sur un système fonctionnant sous Unix, les étapes sont les suivantes :

1. La console Unix affiche un message général d'invitation à la connexion (login banner) :
   login :
   Entrer alors votre nom de login, en minuscules, puis tapez la touche **<entrée>** (ou **<Enter>**).
2. Si vous avez défini un mot de passe, Unix affiche :
   password :
   Entrer alors votre mot de passe, puis **<Entrée>**.
3. S'il y a une erreur, Unix affiche :
   login incorrect
   login :
4. Sinon Unix affiche le message du jour qui peut être différent pour chaque utilisateur. Puis un shell est lancé et un prompt est affiché.

Le **prompt** est un message indiquant le mode interactif : Unix est prêt à exécuter vos commandes dans votre répertoire de connexion. Le prompt est configurables par l'utilisateur. En Bourne shell (différent en C shell), on a par défaut le prompt dollar (**$**).

Remarques :

- Il faut éviter de se connecter en étant en mode majuscules.
- UNIX fait la différence entre les minuscules et les majuscules.
- Les commandes sont à priori en minuscules.
- La touche backspace permet d'effacer le dernier caractère saisi.
- Le caractère @ annule tous les caractères entrés.
- La combinaison de touches ^U annule la ligne en cours de frappe.
- La combinaison de touches ^C interrompt le traitement en cours.

Pour quitter le système Unix :

Il suffit de taper **^d** (touches <Ctrl>d) ou **logout**. Alors Unix revient à l'étape 1 :
login :

6 Système de fichiers

En Unix comme dans les autres systèmes d'exploitation, les données sont stockées sur des mémoires de masse (principalement des disques dur) sous forme de **fichiers**.

Il y a 4 types de fichiers sous UNIX :

- Les **fichiers ordinaires** ou fichiers pleins qui contiennent des données, des programmes... UNIX ne fait aucune différence entre un fichier de texte, un programme exécutable[1] et un fichiers de données.
- Les **fichiers répertoires** ou dossiers, ou catalogues qui sont des « boites » à fichiers et permettent de classer l'information. Comme sous DOS., ils contiennent un certain nombre d'informations sur les fichiers qu'ils contiennent et permettent en particulier de faire le lien entre le nom d'un fichier et son adresse sur le disque (par l'intermédiaire de l'i-node).
- Les **fichiers spéciaux** : (périphériques, mémoires, tubes, liens, sockets...). Comme sous DOS, tout périphérique est considéré comme un fichier et donc chaque périphérique a au moins un fichier spécial qui lui correspond.
- Les **tubes nommés** : normalement, un tube est un ensemble de données transitoires résidant en mémoire. Il est cependant possible d'écrire ces données sur le disque.

Chaque fichier est identifié par:

- **.** un **nom** donné sous forme d'une chaîne de caractères,
- **.** un **chemin** qui précise sa localisation dans l'arborescence des répertoires.

7 Organisation Hiérarchique des fichiers

L'organisation des fichiers sur disque sous Unix est basée sur une *hiérarchie arborescente de répertoires*, très proche de celle de DOS, avec l'existence de répertoires et de sous-répertoires. Cette arborescence, contrairement à dos, est unique et son répertoire racine s'appelle slash (*/*).

N'importe quel utilisateur peut se promener à travers les répertoires et découvrir la structure générale du système. Les commandes à utiliser sont :

- **cd chemin** pour changer de répertoire
- **pwd** pour afficher le répertoire courant
- **ls -l** pour afficher la liste du répertoire courant.

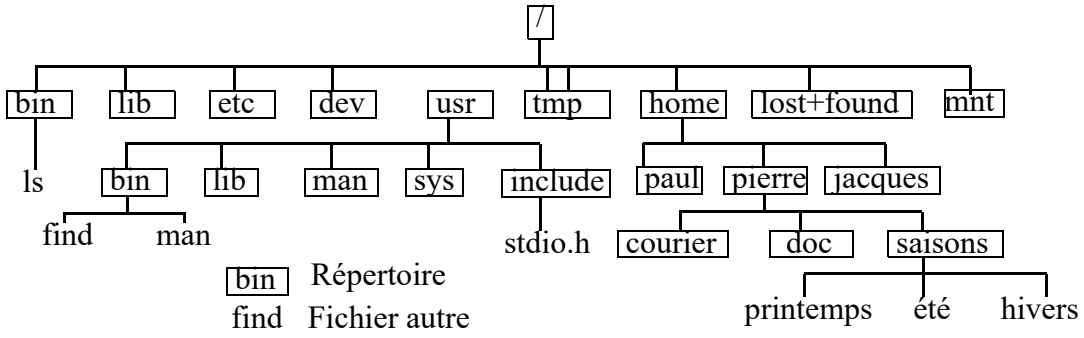

Figure 2 : Exemple de hiérarchie de fichiers

1. Il faut noter que toutes les commandes de base, comme par exemple l'effacement d'un fichier ou l'affichage de la liste d'un répertoire, ne font partie ni du noyau ni de l'interpréteur de commandes, mais constituent des fichiers à part entière sur le disque et sont rangées dans différents répertoires standards.

/bin et /usr/bin (binary)

Répertoires qui contiennent les commandes habituelles d'UNIX.

C'est là que l'interpréteur va chercher les commandes (**ls, find, man**...) que l'utilisateur tape au clavier. Il faut au préalable avoir inclus ces répertoires dans le chemin de recherche des commandes (stocké dans la variable **PATH**).

Il y a 2 répertoires concernés historiquement, lorsque les disques à accès rapide étaient chers, les commandes fréquemment utilisées se trouvaient dans **/bin**, stockées alors sur un disque différent et plus rapide que **/usr/bin**.

/dev (device)

Répertoire contenant les fichiers spéciaux des différents pilotes de périphériques de la machine (console, mémoire, terminaux, disques...).

Chaque terminal, disque et imprimante, apparaît comme une entrée dans ce répertoire. Les programmes de gestion de ces périphériques (**device, driver**) font partie intégrante du noyau. Il est nécessaire de recompiler le noyau lorsque l'on veut insérer un nouveau driver.

Exemples : **tty***x* les ports asynchrones (terminaux), **ttyp***x* les terminaux virtuels, **null** le périphérique nul, **mem** la mémoire vive, **kmem** la mémoire du noyau unix, **swap** le swap disque, **drum** l'espace de pagination, *x***sd***xx* partition de disque scsi, *x***hd***xx* partition de disque ide, *x***fd***xx* disquette, , ...

/etc (config)

Répertoire plus spécialement réservé à l'administrateur du système.

Il contient des commandes et des fichiers de configuration lus au démarrage de la machine (**/etc/passwd /etc/group /etc/inittab /etc/printcap**...).

/lib et /usr/lib : (library)

Répertoires qui contiennent des routines compilées, définies par l'utilisateur ou utilisées par des programmes tels des compilateurs C ou Fortran.

Des fichiers utilisés par certaines commandes peuvent aussi y être présents.

/tmp et /usr/tmp (temporary)

Répertoires qui contiennent des fichiers temporaires créés par le système.

Un utilisateur doit de préférence créer ses fichiers provisoires sous ce répertoire.

Certaines procédures systèmes (dont le redémarrage) effacent le contenu de ces répertoires.

/usr/man (manual)

Répertoires qui contient des fichiers du manuel en ligne.

/usr/include (include)

Répertoires qui contient les fichiers C d'en-tête (suffixés **.h**).

/lost+found :

Répertoire qui contient des informations concernant des erreurs rencontrées lors d'un diagnostic du système de fichiers (commande fsck).

/mnt : (mount)

Lorsque l'on dispose de plusieurs unités de stockage d'information, il est nécessaire de « monter » ces périphériques sur le système de fichiers de base. Le montage est en général fait dans ce répertoire.

N.B. : Le montage est la greffe d'une sous-arborescence provenant d'un autre disque à l'arborescence générale du système.

/home

Répertoire qui contient tous les répertoires de connexion des utilisateurs.

8 Syntaxe des commandes UNIX

8.1 Syntaxe

Les commandes utilisateurs sous les shell unix ont la forme suivante:

Syntaxe : commande [options] [arguments]

Les commandes doivent, le plus souvent, être écrites en minuscules (unix différencie majuscules et minuscules). Dans le texte, les crochets (**[ ]**) signifient que l'élément encadré est optionnel.

- la chaîne de caractères **commande** est généralement un nom de fichier.
- Une **option** commence par un tiret (**-**), par exemple : **-a**. Plusieurs options peuvent se suivre, par exemple : **-a -l** ... ou **-al** ...
- Les **arguments** sont généralement des noms de fichiers précédés, si nécessaire, de leur chemin d'accès.

De plus chaque commande:

- est activée dans le contexte d'un **répertoire courant** (cf **pwd**), et d'un environnement de variables;
- lance l'exécution, sous forme d'un **processus** unix, d'un **fichier exécutable** de même nom dans le système de fichier. Pour que le shell puisse exécuter la commande, il doit trouver le fichier exécutable à lancer dans une liste de chemins de répertoires d'exécutables contenue dans la variable **PATH** (par défaut : /bin:/usr/ucb:/usr/bin).

8.2 Nom de fichiers

Un nom de fichier est une **chaîne de caractères** devant respecter les contraintes suivantes :

- 14 caractères au maximum (parfois plus, selon les systèmes).
- Tous les caractères peuvent être utilisés sauf **<slash>** (**/**).

Il est recommandé de choisir une lettre comme premier caractère et de ne pas employer de caractères invisibles, ni les caractères spéciaux interprétés au niveau du shell comme : **$, ; & <espace> * ? [ ] ( ) \ - ' ` { } | # ~ " < >**.

Tout nom de fichier préfixé par un caractère point (**.**) est considéré comme un **fichier caché** (exemple : **.profile** le fichier de configuration du Bourne shell). Les fichiers cachés **.** et **..** désignent respectivement le **répertoire courant** et son **père**.

Ex : Essayer **$ ls .** et **$ ls ..**

8.3 Génération de noms de fichiers

L'interface shell permet de passer, en argument d'une commande unix, un nom composé de caractères spéciaux (comme sous DOS). Ces caractères sont interprétés par le shell avant le lancement de la commande : tout argument composé de caractères spéciaux est remplacé par l'ensemble des fichiers qu'il décrit.

Les **caractères spéciaux** ou méta-caractères sont : ***** , **?** , **[-]** :

- ***** décrit toute occurence de caractères (sauf un point (.) initial);
- **?** décrit tout caractère;
- **[*ensemble*]** décrit tout caractère parmi ***ensemble*** (**[a-z]** pour une lettre minuscules, **[abc]** pour une des trois premières lettres minuscules de l'alphabet).

Ex : **$ ls fic[A-Z].c** *(afficher les noms de fichiers ficA.c, ficB.c,..., ficZ.c s'ils existent)*

Pour annuler la signification des caractères spéciaux, il faut : soit les encadrer d'apostrophes (**'**); soit les précéder par le caractère anti-slash (\).

Ex : Essayer **ls *.doc** et **ls '*.doc'** et **ls *.doc**

8.4 Nom de chemin

Un chemin permet de décrire la position d'un fichier dans le système de fichiers (arborescence). Un chemin est nommé par la liste des noms de répertoires à parcourir pour atteindre un fichier, chaque nom étant séparé par un caractère **/**. On distingue deux types de chemin :

- **absolu** : à partir de la racine **/** .

Ex : **$ ls /home/pierre/saisons** *pour lister le répertoire /home/pierre/saisons*

- **relatif** : à partir du répertoire courant **.** .

Ex : **$ ls pierre/saisons** *pour, si l'on est dans **/home**, lister le répertoire saison*

9 Documentation en ligne

Le système Unix contient un manuel de référence qui peut être consulté en ligne. Le nombre de commandes Unix et la diversité de leurs options est tel que ce manuel s'avère indispensable et décharge l'utilisateur d'un trop grand effort de mémorisation.

Dans la suite de ce document, il est fortement conseillé d'utiliser systématiquement ce manuel pour chacune des commandes pésentées. On y trouvera d'une part une description plus complète, d'autre part les éventuelles variantes existant sur votre système.

Syntaxe : man [options] [section] [commande]

Cette commande provoque la recherche et édition de la documentation Unix en ligne (manuel de référence).

Par défaut, la section est celle des commandes utilisateurs (niveau 1). Il existe aussi des sections pour la programmation C sous unix (niveaux 2...).

La liste des chemins des répertoires du manuel doit être décrite dans la variable **MANPATH** (par défaut : **/usr/man**).

Par défaut le manuel est affiché page par page à l'aide de la commande **more**. Différente touches de commande peuvent alors être utilisée, dont les principales sont :

h ou **?** : une aide en ligne.
<espace> : page suivante; **<entrée>** : ligne suivante.
/mot : recherche d'un mot; puis **n** : occurrence suivante, **.** précédente.
q ou **:q** : pour quitter.

Voir aussi les commandes **apropos** et **whatis** qui permettent d'obtenir une description très succincte d'une commande.

Syntaxe : whatis mot-clé1 [mot-clé2 ...]

Recherche un mot clé dans une base de donnée, qui doit avoir été préalablement créée par l'administrateur à l'aide de la commande **makewhatis**. Cette base de donnée est obtenue par compilation des information contenu dans les fichiers du manuel en ligne.

Syntaxe : apropos mot-clé1 [mot-clé2 ...]

Recherche une chaîne de caractère dans la base de donnée utilisée par la commande whatis.

Ex : **$ man man** *(voir figure 3, page 15)*
$ man whatis
$ whatis man
man (1) - format and display the on-line manual pages .br
man (7) - macros to format man pages
man.config (5) - configuration data for man

man(1) man(1)

NAME
man - format and display the on-line manual pages
manpath - determine user's search path for man pages

SYNOPSIS
man [-adfhktw] [-m system] [-p string] [-C config_file] [-M path] [-P pager] [-S section_list] [section] name ...

DESCRIPTION
man formats and displays the on-line manual pages. This version knows about the MANPATH and PAGER environment variables, so you can have your own set(s) of personal man pages and choose whatever program you like to display the formatted pages. If section is specified, man only looks in that section of the manual. You may also specify the order to search the sections for entries and which preprocessors to run on the source files via command line options or environment variables. If name contains a / then it is first tried as a filename, so that you can do man ./foo.5 or even man /cd/foo/bar.1.gz.

OPTIONS
-C config_file
Specify the man.config file to use; the default is /usr/lib/man.config. (See man.config(5).)
-M path
Specify the list of directories to seach for man pages. If no such option is given, the environment variable MANPATH is used. If no such environment variable is found, the default list is found by consulting /usr/lib/man.config. An empty substring of MANPATH denotes the default list.
-P pager
Specify which pager to use. By default, man uses /usr/bin/less -s This option overrides the PAGER environment variable.
-S section_list
List is a colon separated list of manual sections to search. This option overrides the MANSECT environment variable.
-a By default, man will exit after displaying the first manual page it finds. Using this option forces man to display all the manual pages that match name, not just the first.
-c Reformat the source man page, even when an up-to-date cat page exists. This can be meaningful if the cat page was formatted for a screen with a different number of columns.
-d Don't actually display the man pages, but do print gobs of debugging information.
-D Both display and print debugging info.
-f Equivalent to whatis.
-h Print a one line help message and exit.
-k Equivalent to apropos.
-m system
Specify an alternate set of man pages to search based on the system name given.
-p string
Specify the sequence of preprocessors to run before nroff or troff. Not all installations will have a full set of preprocessors. Some of the preprocessors and the letters used to designate them are: eqn (e), grap (g), pic (p), tbl (t), vgrind (v), refer (r). This option overrides the MANROFFSEQ environment variable.
-t Use /usr/bin/groff -Tps -mandoc to format the manual page, passing the output to stdout. The output from /usr/bin/groff -Tps -mandoc may need to be passed through some filter or another before being printed.
-w or --path
Don't actually display the man pages, but do print the location(s) of the files that would be formatted or displayed. If no argument is given: display (on stdout) the list of directories that is searched by man for man pages. If manpath is a link to man, then "manpath" is equivalent to "man --path".

CAT PAGES
Man will try to save the formatted man pages, in order to save formatting time next time these pages are needed. Traditionally, formatted versions of pages in DIR/manX are saved in DIR/catX, but other mappings from man dir to cat dir can be specified in /usr/lib/man.config. No cat pages are saved when the required cat directory does not exist.

It is possible to make man suid to a user man. Then, if a cat directory
has owner man and mode 0755 (only writable by man), and the cat files have owner man and mode 0644 or 0444 (only writable by man, or not writable at all), no ordinary user can change the cat pages or put other files in the cat directory. If man is not made suid, then a cat directory should have mode 0777 if all users should be able to leave cat pages there.

The option -c forces reformatting a page, even if a recent cat page exists.

ENVIRONMENT
MANPATH
If MANPATH is set, its value is used as the path to search for manual pages.
MANROFFSEQ
If MANROFFSEQ is set, its value is used to determine the set of preprocessors run before running nroff or troff. By default, pages are passed through the table preprocessor before nroff.
MANSECT
If MANSECT is set, its value is used to determine which manual sections to search.
PAGER If PAGER is set, its value is used as the name of the program to use to display the man page. By default, /usr/bin/less -s is used.
LANG If LANG is set, its value defines the name of the subdirectory where man first looks for man pages. Thus, the command 'LANG=dk man 1 foo' will cause man to look for the foo man page in .../dk/man1/foo.1, and if it cannot find such a file, then in .../man1/foo.1, where ... is a directory on the search path.
NLSPATH, LC_MESSAGES, LANG
The environment variables NLSPATH and LC_MESSAGES (or LANG when the latter does not exist) play a role in locating the message catalog. (But the English messages are compiled in, and for English no catalog is required.) Note that programs like col(1) called by man also use e.g. LC_CTYPE.
PATH PATH is used in the construction of the default search path for man pages.
SYSTEM SYSTEM is used to get the default alternate system name (for use with the -m option).

SEE ALSO
apropos(1), whatis(1), less(1), groff(1).

BUGS
The -t option only works if a troff-like program is installed.

Figure 3 : Résultat de la commande $man man

10 Commandes élémentaires

10.1 Visualisation du contenu de fichiers de texte

Syntaxe : cat [options] [fichier1 fichier2 ...]

Affiche sur l'écran le texte ascii contenu dans les fichiers donnés en argument.

Ex : **$ man cat**
$ cat /etc/passwd
root:..jFpm/gt/fSlY:0:0:System Administrator:/root:/bin/bash
bin:*:1:1:Demons systemes:/bin:
daemon:*:2:2:daemon:/sbin:
....

10.2 Visualisation avec défilement page par page

Syntaxe : more [options] [fichier1 fichier2 ...]

Affiche sur l'écran le texte contenu dans les fichiers en arguments; puis s'arrête après chaque page. Différente touches de commande peuvent ensuite être utilisée, dont les principales sont :

h ou **?** : une aide en ligne.
<espace> : page suivante; **b** ou **^B** : page précédent.
<entrée> : ligne suivante; **y** ligne précédente.
/mot : recherche d'un mot; puis **n** pour l'occurrence suivante, **.** précédente.
q ou **:q** : pour quitter.

Une commande équivalente à **more -c** est **page** ou **pg**. Il est aussi possible de trouver la commande équivalente **less**.

Ex : **$ man more**
$ more /etc/passwd
$ more -c /etc/passwd

11 Entrées/sortie d'une commande unix

11.1 Principe : le filtre unix

La plupart des commandes Unix ont une fonctionnalité de **filtre**, chaque commande :

- accepte en **entrée** un **fichier** de texte passé en argument;
- puis délivre en **sortie** un texte (par traitement du texte d'entrée) sur l'**écran** de la console.

Si aucun nom de fichier n'est passé en argument, la commande passe en mode interactif : elle lit un texte provenant du **clavier** de la console.

Ce texte doit se terminer par le caractère spécial ^D.

Ex : **$ sort**
d *Début de la saisie*
b
a
^d *^D clot le fichier d'entrée*
a
d
b

} Entrée au clavier

} Sortie sur l'écran

11.2 Fichiers d'entrées/sorties standard

Sous UNIX, toutes des entrées/sorties sont réalisées à travers des fichiers. Chaque commande est associée à trois fichiers référencés 0, 1, 2 :

- 0 le fichier d'entrée standard (nommé stdin), qui contient le texte d'entrée de la commande. Par défaut l'entrée standard provient du fichier pilote du clavier du terminal.
- 1 le fichier de sortie standard (stdout), dans lequel est écrit de texte de sortie de la commande. Par défaut le fichier pilote de l'écran du terminal sert de sortie standard.
- 2 le fichier d'erreur standard (stderr) réservé aux messages d'erreur. Par défaut, l'écran du terminal.

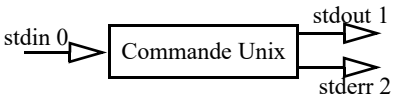

Figure 4 : Filtre unix

11.3 Redirection des entrées/sorties

Chaque fichier associé par défaut aux entrée/sortie peut être remplacé par un fichier quelconque du disque. Ce mécanisme géré au niveau du shell, offert dans tous les types de shell Unix, est nommé : **redirection des entrées/sorties**. Les caractères, communs à tous les shell, indiquant une redirection sont : **<**, **<<**, **>**, **>>**, **|**, ' '.

11.3.1 Redirection de la sortie standard

Syntaxe : commande > resultat

La sortie standard de la commande n'est plus l'écran du terminal, mais le fichier **resultat**. L'opérateur **>** transfert le résultat de la commande vers un fichier sur le disque. Si le fichier n'existait pas, alors il est créé, sinon il est écrasé (si ses droits le permettent)!

Ex : **$ sort > apres**
b
a
c **^D** *pour clore l'entrée*

Accès au résultat : **$ cat apres**
a
b
c

Syntaxe : commande >> resultatCumule

Cette commande est identique à la précédente, à ceci près : l'opérateur **>>** n'écrase pas le fichier **resultatCumule** s'il existait, mais ajoute le résultat de la commande à la fin de ce fichier.

11.3.2 Redirection de l'entrée standard

Syntaxe : commande < fichier

D'une façon complémentaire, on peut rediriger un fichier existant comme entrée d'une commande. Certaines commandes ne peuvent être utilisées que de cette manière, pour d'autres, il est équivalent de fournir en argument le nom du fichier.!

Ex : **$ cat avant**
b
a
c

Puis : **$ sort < avant**
a
b
c

NB : Le résultat serait identique avec le nom du fichier en paramètre (**sort avant**).

Syntaxe : commande <<mot

Lit l'entrée de la commande à la suite de la commande jusqu'à la rencontre d'une ligne débutant par **mot**.

Ex : **$ sort <<fin** donne en sortie : a
b b
a c
c
fin

11.3.3 Chaînage des commandes

Syntaxe : commande1 | commande2

Ce mécanisme, appelé tube (pipe) et déjà présent sous dos, permet d'utiliser directement la sortie d'une commande comme entrée d'une autre commande.

Ainsi, en combinant des commandes simples, on peut composer facilement des commandes complexes.

Ex : **$ ls /dev | more** *Liste le contenu du répertoire **/dev** page par page.*
$ who | cut -f1 -d" " | sort | uniq *Affiche les noms de tous les utilisateurs connectés, chacun une seule fois et par ordre alphabétique.*

11.3.4 Redirection de la sortie standard vers une chaîne de caractères

Syntaxe : ‘commande‘ (‘ est le caractère back quote)
ou **$(commande)** en Bourne shell

Ce mécanisme, permet d'utiliser directement la sortie d'une commande comme une chaîne de caractères, et pourra ainsi devenir le paramètre d'une autre commande.

Ex : **$ echo "nom de la machine : ‘hostname‘ "** *affiche (echo) une chaîne de caractère.*
$ grep ‘hostname‘ /etc/hosts *Recherche la ligne du fichier host avec le nom de la machine.*

11.3.5 Redirections simultanées et spécifiques

On peut enfin combiner les différentes redirections dans un même appel de commande. Toutefois, on ne peut rediriger le même fichier à la fois en entrée et en sortie. En effet, le fichier de sortie est créé avant exécution de la commande et lecture du fichier d'entrée, et provoquera donc son écrasement.

Ex : **$ sort < avant > apres** *trie "avant" puis inscrit le résultat dans "apres"*
$ sort < fichier > fichier *ne fonctionne pas !*

Les redirections **<**, **<<**, **>**, **>>**, **|**, ‘ ‘ sont communes aux shell. Mais, chaque shell possède des redirections spécifiques, dont :

■ En Bourne shell

On peut rediriger la sortie d' erreur vers un fichier :

Syntaxe : commande 2> erreur

Ex : **$ commande 2>/dev/null**
(Redirige la sortie d'erreur vers **/dev/null** le périphérique inactif).

On peut rediriger les sorties entre elles à partir de leur numéro. Ainsi pour rediriger la sortie d'erreur vers la sortie standard :

Syntaxe : commande 2> &1.

Ex : **$ commande 2>&1 1>/dev/null**
*(Affecte l'erreur standard au fichier précédemment destiné à la sortie standard puis redirige la sortie standard vers **/dev/null**.)*

Ex : **$ commande 1>/dev/null 2>&1**
*(Redirige la sortie standard vers **/dev/null** puis redirige l'erreur standard vers le fichier précédemment affecté à*

*la sortie standard, c'est-à-dire **/dev/null** !)*

■ En C shell

On peut rediriger simultanément les sorties d'erreur et standard vers un fichier :

Syntaxe : commande >& sortie

Ex : **$ commande >& /dev/null**

(Tout le texte en sortie standard et d'erreur ira à la poubelle /dev/null)

Chapitre II Manipulation des fichiers et des répertoires

12 Accès aux répertoires

12.1 affichage du répertoire de travail

Syntaxe : pwd

Affiche de chemin absolu du répertoire courant

Ex : **$ pwd**
home/jacques

12.2 Changement de répertoire

Syntaxe : cd [répertoire]

La commande **cd** sans argument équivaut à **cd $HOME**, le répertoire racine de utilisateur.

Un répertoire contient, comme sous DOS, 2 fichier au minimum :

- le répertoire lui-même noté : **.**
- le répertoire père noté : **..**

NB : On peut, en particulier utiliser ce dernier répertoire pour changer de répertoire courant sans repasser par la racine

Ex : **$pwd**
home/paul

$ cd ../pierre
$ pwd
home/pierre

13 Affichage du contenu d'un répertoire

Syntaxe : ls [-options] [fichiers ou repertoires]

Affiche les fichiers donnés en argument. Si un fichier est un répertoire, son contenu est affiché. Si aucun argument n'est donnée, le contenu du répertoire courant est affiché. Par défaut, la sortie est triée par ordre alphabétique et donne tous les fichiers sauf ceux dont le nom commence par un point (.).

Les options permettent d'avoir plus ou moins de renseignements sur les fichiers.

-a : contenu complet du répertoire même ceux, cachés, commençant par un point (.).
-R : contenu aussi de tous les sous-répertoires.
-l : contenu du répertoire (sans .) avec les renseignements complets.
-t : contenu trié par date de dernières modifications.
-p : une barre oblique suit les noms de sous-répertoires.
-g : précise le nom du groupe auquel appartient le fichier.
-F : ajoute un suffixe (/,@,*, =, |) au nom du fichier selon sa nature.
-d : liste les caractéristiques du dossier en argument, au lieu de son contenu.

Ex : **$ ls -l**

```
drwxr-xr-x  2  jean  users     1024  Mar 21  1996  News/
1   2       3   4     5         6        7         8
```

- Signification de chaque groupe de caractères en format long (-l) :

1 : nature du fichier (premier caractère) **2** : autorisations d'accès (9 caractères suivants)
3 : nombre de liens du fichier **4** : nom du propriétaire
5 : nom du groupe d'utilisateurs **6** : taille en octets
7 : date de dernière modification **8** : nom du fichier

- Si l'option **i** est choisie, on rajoute en début de ligne le numéro d'i-node du fichier.
- Nature du fichier :

- : fichier ordinaire; **d** : répertoire; **l** : fichier lien (raccourci); **b** : fichier spécial de type blocs;
c : fichier spécial de type caractères; **p** : fichier tube nommé; **s** : fichier socket communication.

Ex : **$ ls -F**

```
Repertoire/   lien@   executable*   fichier_normal   tube|   socket=
```

$ ls -lF

```
total 10
drwx------ 6 pierre  users  1024 Mar  3 17:56 Repertoire/
lrwxrwxrwx 9 pierre  users  1024 Mar 24 17:32 lien -> ../fichier_lie
-rwxr-xr-x 4 pierre  users 10125 Dec 12 17:56 executable*
-rw-r--r-- 2 pierre  useers  250 Nov 19 12:08 fichier_normal
prw-rw-r-- 2 pierre  users     0 Nov 19 12:08 tube|
srwxrwxrwx 2 pierre  users     0 Nov 19 12:08 socket=
```

$ ls -aF

```
./   ../   .cache   Repertoire/   executable*   fichier_normal   tube|   socket=
```

$ ls -aF *

```
Repertoire/   executable*   fichier_normal   tube|   socket=
```

$ ls -aF .*

```
./   ../   .cache
```

$ ls /dev/?ty[tp][1-2]*

```
/dev/ptyp1   /dev/ptyp2   /dev/ttyp1   /dev/ttyp2   /dev/ttyp20
```

14 Recopie, Déplacement, suppression, création de liens

14.1 Copie de fichier

Syntaxe : cp fichier1 fichier2
Syntaxe : cp fichier1 [fichier2 …] répertoire

La première syntaxe nécessite deux fichiers non répertoires, elle duplique le fichier **fichier1** et le renomme **fichier2**.

La deuxième forme recopie un ensemble de fichiers non répertoires (**fichier1…**) dans un répertoire (**répertoire**). Les fichiers sont copiés dans le répertoire en conservant leurs noms.

Ex : **$ ls -F**

```
Repertoire/  fichier1
```

$ cp fichier1 fichier2

$ **ls**

```
Repertoire  fichier1   fichier2
```

$ cp fichier1 fichier2 Repertoire
$ ls -F Repertoire

```
fichier1  fichier2
```

14.2 Changement de nom d'un fichier ou répertoire

Syntaxe : mv fichier1 fichier2

Syntaxe : mv fichier1 [fichier2 …] répertoire

Le premier cas provoque un renommage (avec déplacement si les chemins différent) d'un fichier ou d'un répertoire. Dans le deuxième, il y a déplacement des fichiers vers un répertoire.

Ex : **$ ls -F**

```
Repertoire/  fichier1    fichier2
```

$ mv fichier1 fichier3

$ ls

```
Repertoire/  fichier2    fichier3
```

14.3 Création d'un lien avec un fichier

Syntaxe : ln -s fichier1 fichier2

La commande créé un fichier lien (un raccourci, un renvoi) nommé **fichier2** vers un fichiers de nature quelconque **fichier1**. Il est ensuite possible d'accéder au contenu du fichier1 avec le nom de fichier2 (qui peut être dans un autre répertoire).

On peut créer un grand nombre de liens pour un même fichier (Utile si un grand nombre d'utilisateurs utilisent un même fichier de données en lecture).

Ex : **$ cd Repertoire**

$ ls -F

fichier1 fichier2

$ ln -s ../fichier1 fichier3

$ ls -l

```
total 0
-rw-r--r--   1 pierre users    0 Apr 22 16:58 fichier1
-rw-r--r--   1 pierre users    0 Apr 22 16:58 fichier2
lrwxrwxrwx   1 pierre users 1024 Apr 22 18:20 fichier3 -> ../fichier1
```

14.4 Suppression d'un fichier

Syntaxe : rm [options] fichier1 [fichier2 …]

La commande permet la suppression d'un ou de plusieurs fichiers à l'exclusion des répertoires.

Il faut avoir la permission d'écriture (**w**) sur les fichiers à supprimer. Si le fichier est protégé en écriture et que l'utilisateur qui demande l'effacement est son propriétaire, alors il y a demande de confirmation de la suppression, puis suppression forcé du fichier en cas d'accord.

14.5 Remarques

Les commandes cp et mv sont inactives si les fichiers sources et destination sont identiques.

Ex : **$ cp fichier1 fichier1**

cp: 'fichier1' and 'fichier1' are the same file

Par défaut, les commandes cp, mv (pas ln) écrasent le fichier destination si par malheur un fichier de même nom préexistait avec le droit d'écriture (**w**)! On peut y pallier par l'option -i (fortement conseillée!).

Ex : **$ ls -l**

```
total 0
-rw-r--r--   1 pierre users     10 Apr 22 16:58 fichier1
-rw-r--r--   1 pierre users     20 Apr 22 18:39 fichier2
```

$ cp fichier1 fichier2 *(actif ! fichier2 est écrasé!!)*

Les options suivantes sont communes aux commandes rm, mv, cp :

-i : option de sécurité demande de confirmation avant chaque opération destructrice ou écrasante.

Ex : **$ cp -i fichier1 fichier2**
cp: overwrite ‘fichier2’? *(demande de confirmation (y/n)!)*
$ rm -i fichier1
rm: remove ‘fichier1’? *(demande de confirmation (y/n)!)*

-f : option dangereuse qui déjoue la protection d’écriture sur fichiers (écrasement, destruction).

Ex : **$ ls -l**
```
total 0
-rw-r--r--   1 pierre users      10 Apr 22 16:58 fichier1
-rw-r--r--   1 pierre users      20 Apr 22 18:39 fichier2
```
$ ln -sf fichier1 fichier2 *(fichier2 est écrasé!)*
$ ls -l
```
total 0
-rw-r--r--   1 pierre users     10 Apr 22 16:58 fichier1
lrwxrwxrwx   1 pierre users   1024 Apr 22 18:50 fichier2 -> fichier1
```

-r : agit sur la totalité de la branche à partir du répertoire donné en premier argument (tous ces sous-répertoires et tous ses fichiers).

Ex : **$ ls -F ***
fichier1 fichier3
Repertoire:
fichier1 fichier2@
$ cp -ri Repertoire copieDeRepertoire *(copie de branche avec vérification d’écrasement)*
$ ls -F *
fichier1 fichier3
Repertoire:
fichier1 fichier2@
copieDeRepertoire:
fichier1 fichier2@
$ rm -ri copieDeRepertoire *(suppression de branche avec confirmation)*

15 Création, suppression de répertoires

Syntaxe : mkdir répertoire1 [répertoire2 …]

Création de nouveaux répertoires vides.

Syntaxe : rmdir répertoire1 [répertoire2 …]

Suppression de répertoires vides.

16 Protection des fichiers

16.1 Catégories d’utilisateurs associées à un fichier

Le système UNIX reconnait **3 catégories d’utilisateurs** pour un fichier:

- le **propriétaire** (owner) : l’utilisateur (user) qui a créé le fichier.
- le **groupe** (group) : tous les utilisateurs membres du groupe du propriétaire (exclu).
- les **autres** (world) : tous les utilisateurs des autres groupes.

Le propriétaire est donné soit par le nom de connexion, soit par le numéro de propriétaire (voir /etc/passwd). Il en est de même pour le groupe (voir /etc/group).

L’administrateur (root) qui possède toutes les permissions.

16.2 Modification du propriétaire ou du groupe

Syntaxe : chown groupe fichiers et **chgrp owner fichier**

Commandes réservées à l’administrateur du système. **chown** permet changement du propriétaire d’un fichier, et **chgrp** le changement du groupe d’un fichier

16.3 Droits courants

La commande **ls -l** permet d'afficher les autorisations sous la forme : **rwxrwxrwx**. Chaque groupe de trois caractères **rwx** spécifie dans l'ordre : les autorisations pour le propriétaire, le groupe, les autres.

Un tiret (**-**) indique que l'autorisation n'est pas accordée.

- Pour un **fichier** :
  - **r** accorde le droit de lecture,
  - **w** le droit d'écriture,
  - **x** le droit d'exécution.

Ex : **rw-r--r--** limite l'accès en lecture (**r--**) pour les membres du groupe et les autres, et donne l'accès en écriture/lecture (**rw-**) pour le propriétaire.

- Pour un **répertoire** :
  - **r** accorde le droit de lire (par **ls**) le contenu du répertoire, pas d'y entrer;
  - **w** : droit de créer ou de supprimer un fichier du répertoire;
  - **x** : droit d'y entrer (par **cd**), un chemin d'accès peut le contenir;
  - **r-x** : droit d'accès (lecture et entrée) au répertoire.

Ex : **rwxr-x-r-x** limite l'accès en lecture et entrée (**r-x**) pour les membres du groupe et les autres, et donne l'accès en écriture/lecture/entrée (**rwx**) pour le propriétaire.

N'oubliez jamais que sous le système Unix, l'utilisateur est responsable de la protection de ses fichiers (voir la commande umask pour la protection automatique)!

Quelques règles de bon sens :

- ne jamais autoriser l'écriture (**rwx**) dans vos répertoires aux autres utilisateurs (sous peine d'y trouver des fichiers parasites ou pire de voir les fichiers qu'il contient disparaître).
- ne jamais autoriser le droit d'écriture (**rw**) sur un de vos fichiers aux autres utilisateurs (sous peine de le voir disparaître).
- vous pouvez, en revanche, autoriser la lecture et l'exécution aux autres utilisateurs (à vous de gérer la confidentialité de vos documents)

16.4 Modification des droits d'accès à un fichier

Syntaxe : chmod [mode] mode fichier1 [fichier2...]

Commande de changement des autorisations sur un ensemble de fichiers.

Seul le propriétaire d'un fichier peut en changer les droits.

-R est une option permettant de propager la commande à l'ensemble de la branche de fichiers débutant au répertoire donné en argument.

Mode numérique : valeur octale

La notation se fait sur **3** chiffres en **octal** (4 avec les droits spéciaux). Chaque chiffre se rapporte, dans l'ordre de lecture, à l'utilisateur propriétaire, aux utilisateurs du groupe et aux autres utilisateurs.

Ex : **$ chmod 761 fichier**
$ ls -lF fichier
-rwxrw---x 1 pierre users 0 Apr 24 14:31 fichier*
(Le propriétaire 700 a toutes les autorisations; le groupe 060 ne peut pas exécuter, les autres utilisateurs 001 ne peuvent qu'exécuter.)

Classe utilisateur	Octal					Symbolique				
	r	w	x	s	t	r	w	x	s	t
Propriétaire (u)	400	200	100	4000	-	**r--------**	**-w-------**	**--x------**	**--s------**	**-**
Groupe (g)	040	020	010	2000	-	**---r-----**	**----w----**	**-----x---**	**-----s---**	**-**
Autres (o)	004	002	001	-	1000	**------r--**	**-------w-**	**--------x**	**-**	**--------t**

Tableau 2 : Récapitulatif des droits élémentaires à composer

Mode symbolique : classe action droits

La notation emploie les caractères **ugoa** (classe), **rwxst** (droits), **+-=** (action).

- **u g o a** : **u** pour désigner l'utilisateur, **g** le groupe, **o** les autres et **a** tous;
- **+ - =** : **+** pour ajouter, **-** pour retrancher, **=** pour affecter une autorisation;
- **r w x** pour désigner les droits (voir aussi les droits avancés **s t**).

Ex : **$ ls -ld R***

```
drwxr-xr-x   2 pierre users  1024 Apr 22 19:20 Repertoire
```

$ chmod go-rx `R*` (*pour interdire, au groupe et aux autres, les droit d'accès (rx) au répertoire*).

$ ls -ld R*

```
drwx------   2 pierre users  1024 Apr 22 19:20 Repertoire
```

il est possible de spécifier plusieurs classes successivement en les séparant par des virgules (,).

Ex : **$ chmod u=rwx,go-w,o-r R***

16.5 Droits avancés

Il existe aussi des droits avancés :

- **s** SUID bit (04000 en octal) et SGID bit (02000 en octal).

ils accordent des privilèges spéciaux aux fichiers exécutables en se superposant aux droit **x**. Le propriétaire peut placer ce droit SUID pour le propriétaire (resp. SGID pour le groupe) en plus du droit **x** d'exécution. Lorsque le processus qui exécutera le fichier sera lancé par un utilisateur, celui-ci disposera des droits du propriétaire du fichier (resp. du groupe) et non de l'utilisateur qui le lance!

Ex : D'abord chez pierre (L'exécutable **ecrit fichier texte** créé pouis écrit dans **fichier** la chaîne **texte**)

$ id

uid=16(pierre) gid=200(users) groups=200(users)

$ pwd

/ home/pierre/toucher_couler

$ ls -alF

total 4

```
drwxr-xr-x  2 pierre  512 Apr 24 16:47 ./
drwxr-xr-x  8 pierre 1536 Apr 24 16:16 ../
-rwxr-xr-x  1 pierre   33 Apr 24 16:47 ecrit*
```

$ chmod u+s ecrit

$ ls -lF ecrit

```
-rwsr-xr-x  1 pierre   33 Apr 24 16:47 ecrit*
```

Puis chez nicolas

$ id

uid=100(nicolas) gid=20(other) groups=20(other)

$ cd /home/pierre/toucher_couler

$ ecrit fichier "coule !!"

$ ls -lF

total 26

```
-rwxr--r--  1 pierre         11 Apr 24 18:11 fichier
-rwsr-xr-x  1 pierre      24576 Apr 24 18:10 ecrit*
```

$ cat fichier

coule !!

Nicolas peut donc créer grâce au suid bit un fichier propriété de pierre, chez pierre!

- **t** STICKY bit (01000 en octal).

Ce droit **t** vient surimprimer le droit x de la dernière classe d'utilisateur (world ou others) pour des répertoires ou des exécutables. Sur des **répertoires** partagés (droits rwxrwxrwt) comme **/tmp // var/spool/mail**, ce droit n'accorde la possibilité de supprimer un fichier de ce répertoires qu'à son propriétaire. Si ce droit n'était placé, tout utilisateur pourrait y supprimer n'importe quel fichier, même s'il n'en est pas le propriétaire. Sur des **fichiers exécutables**, le sticky bit permet d'accélérer

le chargement en mémoire d'exécutables fortement sollicités en les conservant en mémoire swap après leur première exécution.

Ex : **$ ls -ldF /tmp**

```
drwxrwsrwt  4 bin          13312 Apr 24 18:10 /tmp/
```

16.6 Définition des droits de création

Syntaxe : umask [mode]

Cette commande shell permet la définition des droits maximaux accordés à la création d'un fichier.

L'argument **mode** est un masque de permission (trois caractères octaux) qui permet de **retirer** (pas de donner) les droits spécifiés. Si le mode n'est pas spécifié, la valeur courante du masque est affichée.

Le mode normal de création, lorsque le masque vaut **000**, est :

- **666** (**rw-rw-rw-**) pour un fichier normal,
- **777** (**rwxrwxrwx**) pour un fichier exécutable ou un répertoire.

Le masque de création sera bien sur actif qu'une fois installé par la commande umask, et interviendra sur tous les fichiers créés ultérieurement.

Il est conseillé de placer le masque à 022 si l'on veut assurer une protection minimale. Si on veut une protection maximal, utiliser 077.

Ex : **$ rm -r nouveau_fichier nouveau_repertoire**
$ **umask 000** *(pas de masque, droits de création du système)*
$ touch nouveau_fichier
$ ls -lF nouveau_fichier
-rw-rw-rw- 1 pierre users 0 Apr 24 14:11 nouveau_fichier
$ mkdir nouveau_repertoire
$ **ls -ldF nouveau_repertoire**
drwxrwxrwx 2 pierre users 1024 Apr 24 14:18 nouveau_repertoire/

Ex : **$ rm -r nouveau_fichier nouveau_repertoire**
$ **umask 022** *(suppression des droits d'écriture au groupe et autres)*
$ touch nouveau_fichier
$ ls -lF nouveau_fichier
-rw-r--r-- 1 pierre users 0 Apr 24 14:11 nouveau_fichier
$ mkdir nouveau_repertoire
$ ls -ldF nouveau_repertoire
drwxr-xr-x 2 pierre users 1024 Apr 24 14:18 nouveau_repertoire/

Ex : **$ umask 077**
077
077 -> 000111111 -> **---rwxrwx** : retire tous les droits aux utilisateurs autres que le propriétaire.
A sa création, un fichier aura les droits **rw------**
et un exécutable ou un répertoire les droits **rwx------**.

Chapitre III Traitement de Fichiers

17 Manipulation de fichiers de texte

17.1 Visualisation de début et de fin de fichier

Syntaxe : head [options] [fichier1 fichier2 ...]
Syntaxe : tail [options] [fichier1 fichier2 ...]

La commande **head** (resp. **tail**) affiche le début (resp. fin) des fichiers donnés en argument. Quelques options possibles sont :

-n *nb* ou ***-nb*** affiche les *nb* premières (resp. dernières) lignes (par défaut 10)
-c *nb* affiche les *nb* premiers (resp. dernières) caractères.

Ex : **$ head -3 /etc/group**
root::0:root
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
$ tail -1 /etc/passwd

17.2 Fractionnement vertical

Syntaxe : cut [options] [fichier1 fichier2...]

La commande extrait une séquence de champs ou de colonnes sur chaque ligne d'un fichier de texte ascii.

Ex : **$ cut -c1 /etc/passwd** *(Extrait la première lettre de chaque ligne du fichier **/etc/passwd**.)*
$ **cut -c10-15 /etc/passwd** *(Extrait les colonnes de caractères 10 à 15.)*

Pour extraire des champs dont la liste est introduite par **-f**, il est nécessaire de préciser le caractère séparation de champ avec **-d**.

Ex : $ **cut -d: -f1,3 /etc/passwd** *(Extrait les champs 1 et 3. Le séparateur utilisé est **:**)*
$ cut -d: -f1,3-5 /etc/passwd *(Extrait les champs 1, 3, 4, 5, du fichier **/etc/passwd**.)*
$ cut -c1 -d: -f1,3 /etc/passwd *(Extrait le premier caractère et les champs 1, 2 et 3.)*

17.3 Tri de lignes

Syntaxe : sort [options] [clé[option]...] [fichier1…]

Cette commande permet de trier un fichier contenant des lignes de texte. Le tri s'effectue ligne par ligne sur des clés spécifiées ou par défaut sur toute la ligne.

Une **clé** est une zone de texte servant au tri. Le format d'une clé est **+*début* -*fin***, ou ***début*** (resp. ***fin***) spécifie le début *inclus* (resp. la fin *exclue*) de la zone de clé. Si la position ***début*** (resp. ***fin***) est omise, la clé débute (resp. fini) au début (resp. à la fin) de chaque ligne. Les position ***début*** et ***fin*** sont indentifiées par ***m*[.*n*]**, où ***m*** est un numéro de champ (0 pour le premier champ) et ***n*** un numéro de caractère dans ce champ. Ainsi, **10.3 -15.2** définit la zone du 4éme caractère du 11ème champs, au 2ème caractère du 16ème champs. **10.3 -15** s'arrêterait au 15ème champs.

Quelques options possibles :

-c : vérifie si un fichier est trié;
-m : fusion seule de fichiers déjà triés;
-u : ne conserve qu'un seul exemplaire de lignes identiques;
-o *fic* : fichier de sortie;
-d : ordre du dictionnaire; seuls les lettres, les chiffres, les blancs interviennent dans le tri;
-b : espaces non significatifs ignorés;
-f : de minuscules en majuscules;
-n : tri sur des valeurs numériques;
-r : tri en sens inverse;
-t *car* : spécifie le caractère séparant les champs. Les séparateurs par défaut sont <espace> et <tab>.

Ex : **sort -n -t: +2 -3 /etc/passwd** *(Tri numérique sur le 3iémé champ (uid), champs séparés par* **:** *)*

17.4 Jonction de 2 fichiers sur une zone commune

Syntaxe : join [options] fichier1 fichier2

Les fichiers doivent être dans l'ordre lexicographique (sort -d) sur le champ concerné, en principe le premier.

La commande compare les lignes des 2 fichiers et cherche si le champ concerné est identique, auquel cas, en sortie, est produite une ligne comportant le champ commun, le reste de la ligne de **fichier1** et celui de **fichier2**.

-t*car* : ***car*** est le caractère séparateur de champ, par défaut l'espace ou la tabulation;
-j*n m* ou **-j*n.m*** : jonction sur le champ ***m*** du fichier ***n*** (1 ou 2 pour premier ou deuxième argument).
-o *n.m* [*n.m* ...] : liste des champs affichés en sortie.

17.5 Sélection/rejet de lignes communes à 2 fichiers

Syntaxe : comm [options] fichier1 fichier2

Les fichiers doivent être dans l'ordre lexicographique (sort -d). La sortie est sur 3 colonnes : lignes seulement dans fichier1, lignes seulement dans fichier2, lignes communes.

-123 : supprime l'affichage de la colonne correspondante

17.6 Recherche de lignes identiques dans un fichier

Syntaxe : uniq [options] fichier1 fichier2

En mode normal, les lignes identiques du fichier source (qui doit être trié) ne sont pas répétées dans le fichier de sortie. Quelques options :

-u : seules les lignes qui n'apparaissent qu'une seule fois sont recopiées
-d : seules les lignes répétées sont recopiées

17.7 Fractionnement d'un fichier en plusieurs

Syntaxe : split [options] [-n] fichier1 fichier2

Coupe fichier1 en une série de fichiers de ***n*** (par défaut 1000) lignes nommées fichier2aa fichier2ab ...

Ex : **$ split -10 /etc/passwd pswd**
Le fichier **/etc/passwd** est découpé en une série de fichiers de 10 lignes dont les noms sont : **pswdaa, pswdab, pswdac**...
La commande est inversée par : **cat pswd* > /etc/passwd**

17.8 Comparaison du contenu de 2 fichiers

Syntaxe : cmp [options] fichier1 fichier2

Compare deux fichiers de *données quelconques* (exécutables, textes...). Si les fichiers diffèrent, la commande affiche la position et la valeur du premier octet à partir duquel le contenus des fichiers est différent.

Quelques options :

-l : toutes les différences

-s : pas d'affichage mais un code de retour

Syntaxe : diff [options] fichier1 fichier2

Affiche les différences trouvées entre les deux fichiers de *texte ascii*. A la différence de cmp, diff essaie de resynchroniser les contenu des deux fichiers après chaque différence. Les différences de fichier2 par rapport à fichier1 peuvent être décrites dans une syntaxe compréhensible par l'éditeur sed d'unix. Il devient alors possible de générer fichier2 à partir de fichier1 et d'un fichier contenant son différentiel avec fichier1 (résultat de diff fichier1 fichier2).

L'option **-r** permet de comparer deux branches du système de fichier.

17.9 Compteur de caractères, de mots, de lignes

Syntaxe : wc [options] fichier1 [fichier2...]

La commande wc permet de compter les mots, les lignes, les caractères d'un fichier.

Ex : **$ wc -w /etc/passwd** *(Compte les mots)*
$ wc -c /etc/passwd *(Compte les caractères)*
$ wc -l /etc/passwd *(Compte les lignes)*
$ ls -l | wc -l *(compte le nombre de fichiers d'un répertoire.)*

17.10 Transformation de caractères

Syntaxe : tr [options] chaine1 chaine2

La commande présente la particularité d'être un filtre pur, c'est-à-dire de ne pas accepter de nom de fichier en arguments. On ne peut traiter qu'un seul fichier à la fois en redirigeant son entrée. Les caractères de **chaine1** sont remplacé dans l'ordre par ceux de **chaine2**.

Ex : **$ tr a-z A-Z < fic > maj** (*copie* **fic** *dans* **maj**, *en remplaçant les minuscules par des majuscules.)*
$ tr 'a-e' '0-4' < fic *(Affiche* **fic**, *en remplaçant les lettres a à e par les chiffres 0 à 4.)*

17.11 Traitement de flots de caractères

Syntaxe : sed [options] [fichier1 fichier2...]

La commande **sed** (stream editor) lit une à une les lignes des fichiers dont les noms figurent en arguments et y applique les commandes qui suivent l'option **-e**.

La commande **p** demande l'affichage de la ligne.

Ex : **$ sed -n -e 20p /etc/passwd** *(Affiche à l'écran la ligne 20 du fichier /etc/passwd. En l'absence de l'option* **-n**, *toutes les lignes auraient été affichées avec la ligne 20 en double.)*
$ sed -n -e 18,21p /etc/passwd *(Affichage des lignes 18, 19, 20, 21.)*
$ sed -n -e '18,$p' /etc/passwd *(de la ligne 18 à la fin du fichier.)*
$ sed -n -e '18,/djld/p' /etc/passwd *(de la ligne 18, à la ligne contenant le mot djld.)*
$ sed -n -e '/djld/,/msiou/p' /etc/passwd *(de la ligne contenant le mot djld, à la ligne contenant le mot msiou.)*
$ sed -n -e '/^a/, /^s/p' /etc/passwd *(de la ligne commençant par un a, à la ligne commençant par un s)*
$ sed -n -e '1,3s/a/5' -e '1,3p' fic1 *(remplace dans les 3 premières lignes du fichier fic1, le caractère a par le chiffre 5. Puis affiche les lignes 1 à 3.)*

18 Recherche

18.1 Recherche arborescente de fichiers

Syntaxe : find [repertoire] expressions

find est une commande essentielle du système unix. Elle permet d'explorer des fichiers dans une arborescence, suivant les conditions booléennes **expressions** qui lui sont fournies en argument.

Le premier argument **repertoire** est le répertoire de départ, par défaut le répertoire courant. Tous les fichiers situés dans les répertoires sous-jacents seront explorés.

Quelques expressions possibles :

-name ***nom_fic*** : recherche le nom nom_fic dans une branche.
-print : provoque l'affichage des fichiers sélectionnés.
-type ***type_fic*** : recherche un fichier de type ***type_fic***. les types possibles sont les lettres : **b** pour block (buffered) spécial, **c** pour caractère (unbuffered) spécial, **d** pour directory, **p** pour named pipe (FIFO), **f** pour regular file, **s** pour socket.
-perm [+-] ***droits_en_octal*** : recherche un fichier dont les droits sont ***droits_en_octal***. L'utilisation d'un + signifie qu'au moins un des droits soit actif, tandis qu'un **-** nécessite que tous les droits donnés soient actifs.
-user ***utilisateur*****.** : recherche les fichiers qui appartiennent à un utilisateur
-goup ***groupe*** : recherche les fichiers qui appartiennent à un groupe
-exec ***commande*** : exécution d'une commande s'appliquant aux fichiers trouvés.
-ok ***commande*** : idem option précédente mais avec demande de confirmation avant exécution.

La sélection de fichiers, par rapport à une date d'écriture, de modification, ..., peut se faire avec les conditions suivantes :

-atime ***nbjour*** cette condition est vraie et participe à la sélection des fichiers qui ont été lus dans les ***nbjour*** derniers jours.
-mtime ***nbjour*** condition vraie si les fichiers ont été modifiés dans les ***nbjour*** derniers jours.
-ctime ***nbjour*** condition vraie si les fichiers ont été modifiés dans les ***nbjour*** derniers jours (modification de propriétaire, permission, taille...).
-newer ***fichier*** condition vérifiée par tous les fichiers modifiés ou crées après le fichier ***fichier***.

Chacune de ces expressions renvoie un booléen selon qu'elle est satisfaite ou non. Le fait de juxtaposer des expressions est interprété comme une conjonction lexicographique : les expressions sont successivement testées de gauche à droite tant qu'elles sont vraies.

Ex : **$ find /usr -name "*p" -print**
$ find /usr -type f -name "*p" -print

Ces commandes permettent de rechercher dans la branche démarrant au répertoire **/usr**, tous les fichiers dont le nom se termine par un caractère **p**. Le caractère ***** est un symbole de génération de noms signifiant : "n'importe quoi ou rien". La deuxième condition : **-print** implique que le nom trouvé soit affiché à l'écran. Cette action ne sera effectuée que pour les fichiers satisfaisant aux expressions qui le précèdent (**-name "*p"**). La deuxième possibilité, filtre d'emblée par **-type f** les fichiers pleins (contenant des données); alors que la premières porte sur tous les types de fichiers (répertoires...).

Ex : **$ find /usr -type f -perm-444 -print**

Cette commande permet de rechercher à partir du répertoire **/usr** tous les fichiers pleins exécutables pour tous. L'option **-perm -444** recherche les fichiers ayant les droits d'exécution pour tous les utilisateurs (444->100100100->--x--x--x). Le signe moins (**-**) indique que les autres droits ne sont pas pris en compte. En l'absence de ce signe (**-**), la recherche porte sur les fichiers qui ont exactement ce type de permissions.

Ex : **$ find / -xdev -type f -perm-4000 -print**

Recherche de fichiers possédant un suid bit, dans les disques locaux (**-xdev**).

Ex : **$ find / -xdev -type f -perm-2000 -print**

Recherche de fichiers possédant un sgid bit.

Ex : **$ find / -xdev -type f -perm-1000 -print**

Recherche de fichiers possédant un sticky bit.

Ex : **$ find /etc -type d -name "*[Vv]*" -print**

Recherche à partir du répertoire **/etc**, tous les sous-répertoires dont le nom contient la lettre V majuscule ou v minuscule.

Ex : **$ find /usr -user root -print**

Cette commande est très utile pour vérifier qu'un utilisateur ne met pas de fichier en dehors de son répertoire.

Ex : **$ find /etc -newer defaultdomain -print**
Liste les fichiers créés après le fichier nommé defaultdomain**.**

Les conditions : **-exec**, **-ok**, permettent de lancer une commande à laquelle on associe le nom du fichier sélectionné. Il y a autant de fois exécution qu'il y a de fichiers sélectionnés.

Ex : **$ find /usr -type f -name fic1 -exec rm {} \ ;**

Les accolades **{}** symbolisent le nom du fichier sélectionné par les expressions précédentes. Le point virgule est une autre manière de terminer une commande, au même titre que le retour chariot. Mais, comme il s'agit d'un caractère spécial analysé par le shell, il est nécessaire de le précéder d'un back slash (\) afin d'en faire un caractère normal. Noter la présence d'un séparateur (blanc, tabulation) obligatoire devant **** : **`rm {} \;`**.

L'option **-ok** demande confirmation avant de lancer la commande.

Ex : **$ find /usr -type f -name "*p*" -ok ls -l {} \;**
< ls -l /usr/.exerc> ? y
`-rw-r--r-- 1 root other 65 may 15 11:50 .exerc`

18.2 Recherche d'une chaîne dans un fichier

Syntaxe : grep [options] expr [fichier1 fichier2...]

La commandes **grep** permet d'extraire, des fichiers en argument, certaines lignes en fonction de leur contenu. Il existe aussi **egrep** et **fgrep** qui ne diffèrent par la syntaxe de **expr**.

Le premier argument **expr** est une expression régulière, modèle de chaîne de caractères, qui sera recherchée dans les fichiers donnés en arguments. Des symboles peuvent être utilisés dans le premier argument pour former une expression régulière qui servira de modèle de recherche. Le caractère chapeau (**^**) symbolise le début d'une ligne, alors que dollar (**$**) représente sa terminaison. Le symbole point (**.**) représente un caractère quelconque. * signifie 0 à n occurrences du caractère précédent,

Toute ligne contenant une séquence de caractères correspondant au modèle sera extraite. Le nom du fichier concerné est donné s'il y en a plusieurs.

Quelques options :

-c : affichage du nombre de lignes trouvées uniquement
-i : ignore la distinction entre majuscules et minuscules
-n : affichage de la ligne précédée de son numéro
-v : affichage de toutes les lignes sauf celles qui contiennent la combinaison

Ex : **$ grep root /etc/passwd** *(lignes contenant le mot root.)*
$ grep -v root /etc/passwd *(lignes ne contenant pas le mot root)*
$ grep -i "[ad]" /etc/passwd *(lignes contenat les caractères a ou d)*
$ grep "^r" /etc/passwd *(lignes qui débutent par la lettre r)*
$ grep ':$' /etc/passwd *(lignes qui finissent par le caractère deux points (:))*
$ grep "^..e" /etc/passwd *(lignes qui ont la lettre e en 3ième position)*
$ grep "^..[aeo]" /etc/passwd *(lignes qui ont un a , e, ou o en 3ième position)*
$ grep "^[a-e]" /etc/passwd *(lignes qui commencent par l'une des cinq premières lettres de l'alphabet)*

18.3 Filtrage élaboré (awk)

Syntaxe : awk [options] [expr] [{action}] [fichier1 fichier2...]

La commande **awk** est un filtre puissant permettant de sélectionner à partir du paramètre expr, d'extraire et de transformer selon le paramètre action, les lines, les champs, les caractères des fi-

chiers en argument. La force de cette commande est d'offrir un véritable langage interprété pour réaliser ces traitements.

dd : copie de fichier avec conversion

dd [option=valeur] ...

Cette commande permet par exemple de convertir un fichier ASCII en EBCDIC.

file : nature d'un fichier

file fichier

Délivre une description sur la nature du fichier passé en argument.

which : localisation rapide d'un exécutable

which fichier

Renvoie, s'il existe, le chemin absolu menant au fichier exécutable. La recherche en basée sur la variable $PATH contenu la liste des chemins d'accès aux exécutables.

whereis : localisation rapide d'un fichier

whereis fichier

Renvoie, s'ils existent, les chemins absolus menant au fichier.

19 Gestion de l'impression

Pour imprimer un fichier, il faut s'assurer que celui-ci peut être imprimé. Les fichiers binaires comportant des caractères de contrôle sont à exclure. Les fichiers *postscript* ne sont imprimables que sur une imprimante dite postscript. Sur la majorité des systèmes, le standard d'impression est postscript.

Il existe toutefois des logiciels qui peuvent transformer :

- des fichier texte ascii en fichiers postscript. Ex : pr, a2ps, ascii2ps.
- des fichiers binaires en ascii. Ex : uuencode (utilisé pour envoyer des codes par mail).

Les imprimantes reconnue sont déclarée dans le fichier **/etc/printcap**. Le premier champs de description de chaque imprimante est son nom dans le système. Lorsque plusieurs imprimantes sont disponibles, les commandes **lp*** ci-dessous accepte l'option **-P*printer***, où ***printer*** est le nom de l'imprimante choisie. La première imprimante du fichier printcap est l'imprimante par défaut du système.

Chaque imprimante est associée à une file d'attente de travaux d'impression, dans laquelle sont placées les demandes d'impression des utilisateurs. Ces files d'attentes sont dans le répertoire système **/var/spool/*printer***, où ***printer*** est le nom de l'imprimante associée.

19.1 Imprimer un fichier

Syntaxe : lpr [options] [fichier1 fichier2 …]

Cette commande place les fichiers d'impression (généralement postscript) donnés en argument dans la file d'attente d'impression de l'imprimante. Les travaux seront ensuite imprimés chacun leur tour. Si aucun fichier n'est spécifié, l'entrée standard est prise comme un fichier à imprimer.

Ex : **$ lpr fic1**

Si l'imprimante est occupée par l'impression d'un fichier appartenant à un autre utilisateur, votre demande d'impression sera mise en attente.

19.2 Examiner la file d'attente d'impression

Syntaxe : lpq [options]

Permet de visualiser la file d'attente des travaux d'impression.

Ex : **$ lpq**

"imprimante is ready and printing"

Rank	owner	Job	Filer	Total size
printing	pierre	71	fic0	1236 bytes
active	paul	72	fic1	4030 bytes

19.3 Retirer une demande d'impression de la file d'attente

Syntaxe : lprm [options] numéro

Permet de supprimer un travail de la file d'attente d'impression, à partir de son numéro d'ordre (champs job de la commande lpq).

Ex : **$ lprm 72**
$ lpq

"imprimante is ready and printing"

Rank	owner	Job	Filer	Total size
printing	pierre	71	fic0	1236 bytes

19.4 Mise en page de fichiers pour l'impression (pretty print)

Syntaxe : pr [options] fichier1 [fichier2…]

Cette commande, qui génère une sortie postscript, permet de déterminer le format d'impression des fichiers ascii en argument, C'est-à-dire : le nombre de colonnes, de lignes par page, les titres, la taille de l'interligne, la numérotation des pages, effectuer une fusion de plusieurs fichiers...

Ex : **$ pr /etc/group | lpr**

Chapitre IV Contrôle d'exécution des processus

20 Notion de Processus

Toute commande unix s'exécute sous forme d'un processus. Ainsi lorsque la commande **ls** est interprétée par le shell dans lequel est tapée, il y a création d'un nouveau processus qui exécute l'opération correspondante, c'est-à-dire le fichier **/bin/ls**.

Un processus est une unité d'exécution correspondant à l'image en mémoire d'un fichier exécutable. Cette image en mémoire a une durée de vie limitée : son temps d'exécution, puis disparait de la mémoire. Chaque processus possède *en propre* :

- du *code exécutable* pour le(s) processeur(s),
- une zone de *données* : pile, tas, variables statiques (globales, constantes).
- un *contexte d'exécution* : variables d'environnement shell, répertoire de travail, table de fichiers ouverts (dont les entrées/sorties), identification, priorité, ...

20.1 Filiation, hiérarchie de processus

Il existe une filiation (« héritage ») entre processus. En effet : chaque processus est le **fils** d'un unique **père** : celui qui l'a créé. (Sauf le premier processus du système lancé automatique au boot). De plus le fils hérite de son père, par un mécanisme de recopie, de certaine caractéristiques :

- ses variables d'environnement,
- son répertoire de travail,
- sa table de fichiers ouverts (dont les entrées/sorties).

Il est important de noter que ces données ne sont pas partagées mais ***dupliquées*** et transmises au fils. En effet, père et fils évoluent indépendamment et les modifications apportées par le père (resp. le fils) sur les valeurs des données précédentes sont sans effet sur les valeurs correspondantes du fils (resp. du père).

Par contre ils restent liés : la mort d'un shell père entraîne la mort des ses shells fils (signal SIGHUP).

20.2 Identification

Chaque processus est identifié dans le système par un numéro entier unique : son **PID** (Process IDentity).

Les processus possèdent d'autres caractéristiques d'identification, dont :

- **PPID** (Parent Process ID): le pid de son père.
- **UID** (User ID): un entier identifiant son propriétaire (le même que pour les fichiers).
- **GID** (Group ID): un entier identifiant le groupe du processus.

UID et GID sont hérités du père pour les utilisateur normaux (hors root).

- **EUID** (effective UID), **EGID** (Effective GID) sont respectivement l'utilisateur et le groupe effectif, c'est-à-dire ceux utilisés pour les droits accordés en cours d'exécution. Ils ont généralement les mêmes valeurs que UID et GID sauf lors du positionnement des droits SUID SGID sur le fichier exécutables du quel est issu le processus (cf SUID SGID bits).

20.3 Exécution en temps partagé

Le système unix est multi-tâches, et les processus s'exécutent en "parallèle". En fait, le processeur (cpu) est souvent unique, et le parallélisme est alors simulé par le temps partagé (pseudo parallélisme).

Dans ce cas, le temps de traitement du processeur est partagé en intervalles élémentaires (slices); et chaque processus du système se voit accordé une suite intervalle de temps, généralement non contiguës, dans laquelle il est exécuté par le processeur.

La distribution (ordonnancement, planification, scheduling) des intervalles de temps est gérée par le système et dépend de la priorité accordée à chaque processus. Plus un processus est prioritaire, plus il se verra accordé rapidement des intervalles de temps rapprochés, et par conséquent il sera exécutés rapidement. Moins un processus est prioritaires, plus il se verra accordé tardivement (après les autres) des intervalles de temps éloignés les uns des autres, et donc il sera exécuté lentement.

20.4 Etats d'un processus

L'exécution d'un processus étant discontinue, il peut prendre plusieurs états:

Elu : il s'exécute sur le cpu. Etat 1 : mode noyau (préemptif). Etat 2 : mode utilisateur.
Prêt : en attente de mise en exécution par le scheduler. Etats 5 : prêt et en mémoire. Etats 6 : prêt mais swappé.
Bloqué : en attente d'un événement de reprise. Etats 4 : Endormi (Sleeping), suspendu par l'utilisateur. Etats 3 : Suspendu (Idle): en attente d'être exécuté
Terminé (état 9): exécution terminée.
Zombie (état 8): terminé mais son père n'en a pas eu connaissance.

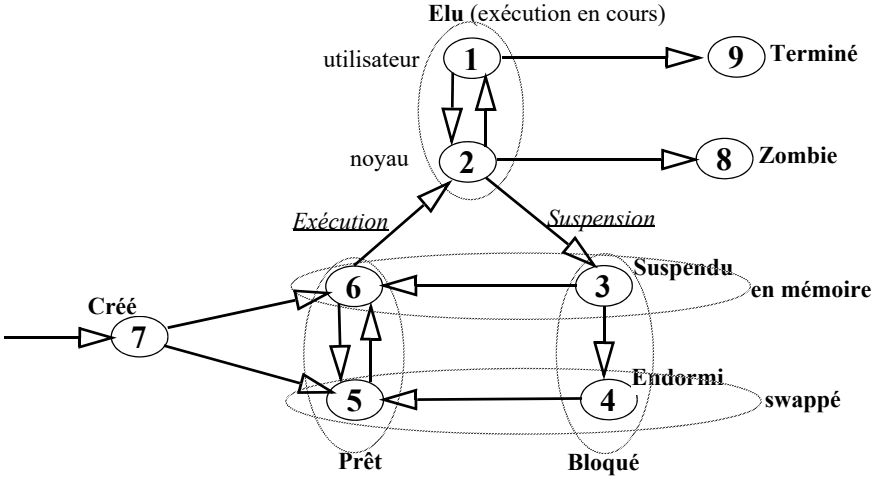

Figure 5 : Etats d'un processus

21 Principales commandes

21.1 Listage des processus

Syntaxe : ps [options] [pid]

Indication sur l'état des processus actifs. S'il n'y a pas d'option, seule une liste abrégée des processus liés au terminal est affichée. Les principales options sont les suivantes :

- modes de listage :
  - **-l**, **u, j, s, v, m** . Choisir un unique mode parmi : long (**l**), utilisateur (**u**), ids processus (**j**), signaux (**s**), mémoire virtuelle (**v**), statistiques mémoire (**m**).
  - **-n** force l'affichage numérique de certains champs (ex: **USER**)
  - **-S** ajoute le temp cpu des fils.
  - **-c** ajoute le nom de la commande (par défaut)
- Options de sélection :
  - **-a** inclut les informations sur les processus appartenant aux autres
  - **-x** inclut les processus sans terminal de contrôle
  - **-g** montre tous les processus (par défaut ps sélectionne les plus intéressants)
  - **-r** sélectionne les processus exécutables (état R)
  - **-f** montre la hiérarchie des processus (sous certains unix).
  - **-t***x* seuls les processus liés au terminal *x* sont concernés. **t***x* doit être spécifié comme listé par ps : **t**3 pour **/dev/tty3**, **t**co pour **/dev/console**…
- Autres options :
  - **-e** affiche l'environnement en plus de la ligne de commande.
  - **-w, ww** mode 132 colonnes (**w**), et non tronqué (**ww**).

Signification des différentes informations affichées :

	PID	numéro du processus
	TT, **TTY**	terminal de contrôle
u	**STAT**	état du processus :
	R	(**r**unnable) processus dans la file d'attente d'exécution (prêt à s'exécuter)
	T	(**t**erminated) processus terminé
	S	(**s**leeping) processus endormi (en attente depuis moins de 20 secondes)
	I	(**i**dle) processus suspendu (en attente depuis plus de 20 secondes)
	Z	(**z**ombie) processus terminé dont le père n'a pas été avisé de la terminaison
	N <	indiquent que la priorité a été augmentée (**N**) ou diminuée (**<**)
	W	(**s**wapped out) processus swappé de la mémoire vers le disque.
u	**%CPU**	taux d'utilisation du cpu (moyenne calculée sur une minute)
l	**NI**	valeur utilisée pour le calcul de la priorité (nice)
l	**PRI**	priorité du processus (chiffre élevé priorité moindre)
lu	**SIZE, SZ**	taille des segment de données et de la pile (en Ko)
lu	**RSS**	taille réelle résidente (en Ko)
	%MEM	pourcentage de mémoire résidente allouée à ce processus
	USER	nom du propriétaire du processus
	UID	numéro du propriétaire du processus
lj	**PPID**	numéro du processus père
j	**SID**	numéro de la session à laquelle appartient le processus
j	**PGID**, **TPGID**	numéro de groupe du processus, du groupe du terminal du processus
l	**CP**	exploitation du processeur à court terme pour le partage du processeur
	START	date ou heure de démarrage du processus
	TIME	durée d'exécution du processus
l	**WCHAN**	événement par lequel le processus s'est endormi (ou son adresse si l'option **n**). si champ vide ∅ : processus en cours d'exécution.
	COMMAND	nom de la commande

Exemples :

$ ps

```
  PID TTY STAT  TIME COMMAND
11676 pp2 S     0:01 -bash
11736 pp2 R     0:00 ps
```

$ ps u

```
USER      PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND
guillet 11676  0.3  1.7  384  548 pp2 S    15:59   0:01 -bash
guillet 11738  0.0  0.5   57  184 pp2 R    16:06   0:00 ps u
```

$ ps j

```
 PPID   PID  PGID   SID TTY TPGID  STAT   UID   TIME COMMAND
11675 11676 11676 11676 pp2 11737  S     5188   0:01 -bash
11676 11737 11737 11676 pp2 11737  R     5188   0:00 ps j
```

$ ps l

```
 F   UID   PID  PPID PRI NI SIZE  RSS WCHAN      STAT TTY   TIME COMMAND
 0  188 11676 11675   2  0  384  548 11bbe8     S    pp2   0:01 -bash
 0  188 11750 11676  20  0   57  184 0          R    pp2   0:00 ps l
```

$ ps m

```
  PID TTY MAJFLT MINFLT  TRS  DRS SIZE SWAP  RSS SHRD  LIB  DT COMMAND
11757 pp2     21     59   20   36  372    0  372  308  316  16 ps m
11676 pp2     41    783  252  100  724    0  724  564  372  40 -bash
```

$ ps v

```
  PID TTY STAT  TIME  PAGEIN TSIZ DSIZ  RSS   LIM %MEM COMMAND
11676 pp2 S     0:01      41  268  116  548    xx  1.7 -bash
11767 pp2 R     0:00      21   16   41  184    xx  0.5 ps v
```

$ ps s

```
 UID   PID SIGNAL   BLOCKED  IGNORED  CATCHED  STAT TTY   TIME COMMAND
 188 11676 00000000 00010000 7fffffff 07803efb S    pp2   0:01 -bash
 188 11786 00000000 00000000 7fffffff 00000000 R    pp2   0:00 ps s
```

$ ps f (linux?)

```
  PID TTY STAT  TIME COMMAND
11676 pp2 S     0:01 -bash
11788 pp2 R     0:00  \_ ps f
```

$ ps e

```
  PID TTY STAT  TIME COMMAND
11676 pp2 S     0:01 -bash + TERM=vt100 HOME=/home/profs/fguillet PATH=/usr/loca
11812 pp2 R     0:00 ps e + ignoreeof=10 HOSTNAME=tequila.silr.ireste.fr LOGNAME
```

$ ps ps auxfgw

```
USER       PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND
root         1  0.0  1.0   52  336  ?  S    13:23   0:00 init [5]
root         6  0.0  0.8   35  284  ?  S    13:23   0:00 bdflush (daemon)
root         7  0.0  0.8   35  284  ?  S    13:23   0:00 update (bdflush)
root        23  0.0  1.1   49  352  ?  S    13:24   0:01 /usr/sbin/crond -l10
root        25  0.0  0.9   31  312  ?  S    13:24   0:00 /usr/sbin/lpd
root        27  0.0  1.0   44  344  ?  S    13:24   0:00 /usr/sbin/klogd
root        35  0.0  1.1   42  364  ?  S    13:24   0:00 gpm -m /dev/mouse -t mman
root        64  0.0  1.1   54  356  ?  S    13:24   0:00 /usr/sbin/inetd
root       409  0.1  1.3   62  432  ?  S    17:16   0:00  \_ in.telnetd
fguillet   410  0.3  2.0  117  652 pp0 S    17:16   0:01      \_ -bash
fguillet   445  0.0  1.0   67  340 pp0 R    17:21   0:00          \_ ps auxfgw
fguillet   446  0.0  0.8   45  280 pp0 R    17:21   0:00          \_ less
root        66  0.0  1.1   53  372  ?  S    13:24   0:01 /usr/sbin/syslogd
root        69  0.0  1.9  107  616  ?  S    13:24   0:00 sendmail: accepting connections
root        72  0.0  0.7   36  224  ?  S    13:24   0:00 /var/config/sbin/configd
root       131  0.0  0.9   52  304 v01 S    13:24   0:00 /sbin/agetty -L 38400 tty1 linux
root       135  0.0  2.3   69  732  ?  S    13:24   0:00 /usr/X11/bin/xdm -nodaemon
root       137  0.0  7.9 1633 2528  ?  S    13:24   0:03  \_ /usr/X11R6/bin/X -auth /usr/X11R6/lib/
X11/xdm/A:0-a00135
root       138  0.0  3.7  117 1180  ?  S    13:24   0:00  \_ -:0
```

$ ps ajxf

```
 PPID   PID  PGID   SID TTY TPGID  STAT   UID   TIME COMMAND
    0     1     1     1  ?     -1  S        0   0:00 init [5]
    1     6     6     6  ?     -1  S        0   0:00 bdflush (daemon)
    1     7     7     7  ?     -1  S        0   0:00 update (bdflush)
    1    23    18    18  ?     -1  S        0   0:01 /usr/sbin/crond -l10
    1    72    18    18  ?     -1  S        0   0:00 /var/config/sbin/configd
    1    25    25    25  ?     -1  S        0   0:00 /usr/sbin/lpd
    1    27    27    27  ?     -1  S        0   0:00 /usr/sbin/klogd
    1    35    35    35  ?     -1  S        0   0:00 gpm -m /dev/mouse -t mman
    1    64    64    64  ?     -1  S        0   0:00 /usr/sbin/inetd
   64   409    64    64  ?     -1  S        0   0:00  \_ in.telnetd
  409   410   410   410 pp0   453  S     5188   0:01      \_ -bash
  410   453   453   410 pp0   453  R     5188   0:00         \_ ps ajxf
    1    66    66    66  ?     -1  S        0   0:01 /usr/sbin/syslogd
    1    69    69    69  ?     -1  S        0   0:00 sendmail: accepting connect
    1   131   131   131 v01   131  S        0   0:00 /sbin/agetty -L 38400 tty1
    1   135   135   135  ?     -1  S        0   0:00 /usr/X11/bin/xdm -nodaemon
  135   137   137   135  ?     -1  S        0   0:03  \_ /usr/X11R6/bin/X -auth
  135   138   138   135  ?     -1  S        0   0:00  \_ -:0
```

21.2 Processus et signaux

■ **kill** : envoi d'un signal à un processus

Syntaxe : kill [-signal] pid

Envoie le signal numéro **signal** au processus numéro **pid**.

-l : affiche une liste des signaux

Le numéro de processus (pid) est donné par la commande **ps**. Les signaux sont listés dans le fichier **/usr/include/signal.h**. Par défaut **kill** envoie le signal 15 de terminaison (TERM).

Pour tuer un processus récalcitrant, utiliser le signal 9 (KILL) qui ne peut être capturé. Tous les processus fils d'un processus tué sont tués (propagation d'un signal TERM).

Exercice : essayer les signaux STOP et CONT

```
$ find / -name guillet 2>/dev/null &
$ ps
  PID TTY STAT  TIME COMMAND
25007 pp1 R         0:00   find / -name guillet
$ kill -STOP 25007
$ ps
  PID TTY STAT  TIME COMMAND
25007 pp1 T         0:10   find / -name guillet
[1]+  Stopped (signal)       find / -name guillet 2>/dev/null
$ kill -CONT 25007
$ ps
  PID TTY STAT  TIME COMMAND
25007 pp1 R         0:12   find / -name guillet
```

■ **trap** : interception de signaux (interne au Bourne shell)

Syntaxe : trap [suite de commandes] signaux

Active la redirection des signaux dans le processus courant.

Ex : **trap 'rm /tmp/temporaire; exit' 2 3** pour supprimer un fichier temporaire puis terminer le processus à la réception d'un signal SIGINT (touches <ctrl c>) ou SIGQUIT (touches <ctrl \>), ou SIGSTOP (touche <ctrl z>).

trap ' ' 2 pour ne rien faire à la réception d'un signal SIGINT.

trap 2 pour rétablir le comportement par défaut (interruption) à la réception d'un SIGINT.

Ex : taper, puis exécuter le fichier suivant

```
trap 'echo touche <ctrl c>' 2
trap 'echo touche <ctrl \>' 3
while [1=1]
    do
    done
```

■ **nohup** : immunisation d'une commande contre les coupures de ligne (SIGHUP)

Syntaxe : nohup commande

S'il y a coupure, la sortie standard et la sortie d'erreur standard sont détournées dans **nohup.out**. Si l'utilisateur quitte sa session alors que des processus tournent en arrière-plan, le système lui demande une confirmation. Si le logout est confirmé, tous les processus attachés à la session sont supprimés, sauf ceux lancés avec la commande **nohup**.

Ex :
```
$ nohup find / -name pierre &
nohup: appending output to 'nohup.out'
$ ps
```
Sortez du terminal, puis dans un autre faites ps.

21.3 Lancement des processus

■ commande : exécution bloquante (avec attente de terminaison, mode synchrone)

Syntaxe : nom_commande [arguments]

Il y a création d'un processus shell fils, le processus shell père étant celui qui interprète la commande **nom_commande**. Le shell fils est exécuté en parallèle, mais le shell père est suspendu jusqu'à ce que le shell fils se soit terminé.

■ **exec** : exécute une commande à la place du shell

Syntaxe : exec commande

exec exécute une commande à la place du shell, qui termine ; il n'y a pas création d'un nouveau processus.

Ex : Comparer les résultats des deux lignes suivantes :
```
$ ps
$ exec ps
```

■ **&** : exécution non bloquante (tâche de fond, mode asynchrone)

Syntaxe : nom_commande [arguments] &

Il y a création d'un processus shell fils, le processus shell père étant celui qui interprète la commande **nom_commande**. Le shell fils est exécuté en parallèle sans bloquer le shell père qui continue à s'exécuter. Renvoie le pid du processus fils. Noter que la variable **$!** contient le pid du dernier fils lancé.

Ex :
```
$ ps &
  PID TTY STAT  TIME COMMAND
  273 pp0 S     0:05 -bash
 1106 pp0 R     0:00 ps
$ echo $!
1106
```

21.4 Lancement des processus différés

■ **at, atq, atrm** : exécution différée datée

Syntaxe : at [-q*file*] date [< fichier_de_commandes]

La commande **at** place la demande d'exécution différée dans une liste d'attente ***file*** identifiée par une lettre de **a** à **z** et **A** à **Z**. Le fichier de commande sera automatiquement exécuté par le système à la date donnée. Pour pouvoir être utilisée, cette commande nécessite que le démon **crond** soit lancé au boot système.

Il est préférable de rediriger la sortie standard, car par le résultat de la commande différée est envoyée à la messagerie électronique.

Syntaxe : atq [-q*file*] [*atpid*]

Syntaxe : atrm [-q*file*] *atpid*

Les commandes **atq** et **atrm** permettent de lister les exécutions différées enregistrées et de les supprimer à partir de leur numéro. **atq** est équivalent à **at -l**, et **atrm** à **at -d**.

Ex : **$ at now + 1 minute > ps.result**
ps
<ctrl d>
Job 130 will be executed using /bin/sh
(ou encore **$ echo ps | at now + 1 minute > ps.result**)
$ atq

Date	Owner	Queue	Job#
19:17:00 04/17/97	pierre	c	130

Ex : **$ at 0:15am May 30 < shell_script**

■ **batch** : mise en file d'attente d'exécution de faible priorité

Syntaxe : batch [< fichier_de_commandes]

batch est équivalent à **at -b** . Cette commande place une demande d'exécution différée dans une liste d'attente à faible priorité. Le fichier **fichier_de_commandes** sera automatiquement exécuté par le système dès que la charge cpu sera faible.

Ex : **$ echo ps | batch > ps.result**
Job 10 will be executed using /bin/sh
$ atq

Date	Owner	Queue	Job#
20:07:00 04/28/97	pierre	E	10

■ **crontab** : exécution différée cyclique

Syntaxe : crontab fichier_cron

Le fichier crontab **fichier_cron** contient la description de l'ensemble des commandes cycliques à lancer automatiquement. Le fichier contient des lignes de 5 champs successifs :

- 1) minute 0-59
- 2) heure 0-23
- 3) jour du mois 1-31
- 4) mois 1-12
- 5) jour de la semaine 0-6 (0 pour dimanche)
- 6) commande à lancer

Il peut y être placé des caractères spéciaux : ***** signifie 'tous', **1-5** signifie de 1 à 5, et **1,4,5** signifie les valeurs 1, 4 et 5.

Exemple de ligne du fichier **fichier_cron** :

30 20 * * 0 batch_du_dimanche_a_20H30

crontab -l liste votre fichier crontab.

21.5 Gestion de la priorité

■ **nice** : lancement d'un processus avec gestion de sa priorité

Syntaxe : nice [-*priorite*] *commande*

La valeur ***priorite*** est un entier permettant d'indiquer la priorité souhaitée pour le processus ***commande*** dans le système. Plus cette valeur est élevée, moins le processus est prioritaire et plus il s'exécutera lentement. Un utilisateur commun ne peut accorder qu'une valeur de positive. Le super-utilisateur root peut accorder une valeur négative. Généralement la valeur est comprise -20 et 19 (la plus faible priorité), et par défaut, elle est de 10.

Il y a deux versions distinctes de nice, l'une est intégrée au C shell, l'autre est la commande /usr/bin/nice.

■ **renice** : changement de priorité d'un processus en cours d'exécution

Syntaxe : renice [*priorite*] *pid*

L'incrément est compris entre 1 et 19 (la plus faible priorité). Par défaut, il est de 10.

Le super-utilisateur peut utiliser une priorité plus grande que la normale en employant un incrément négatif jusqu'à -20, -10 par exemple :

Ex : **$ nice -5 find / -name ireste 2>/dev/null & ps -l**

```
F  UID PID  PPIDPRI NI SIZE  RSS WCHAN     STAT TTY   TIME COMMAND
0  188 24768 23929 22  5  118  264           0       R N  pp3   0:01 find / -name ireste
0  188 24775 23929 20  0   57  184           0       R      pp3   0:00 ps -l
```

$ renice 9 24768 & ps -l

```
24768: old priority 19, new priority 9
F  UID PID PPIDPRI NI SIZE  RSS WCHAN     STAT TTY   TIME COMMAND
0  188 24768 23929 25  9  118  264           0       R N  pp3   0:01 find / -name ireste
0  188 24775 23929 20  0   57  184           0       R     pp3   0:00 ps -l
```

$ renice -10 24768 *réservé à root*

```
renice: 24768: setpriority: Permission denied
```

21.6 Autres commandes

■ **sleep** : suspend de l'exécution du processus pendant une durée donnée

Syntaxe : sleep durée

le temps **durée** est donné en secondes.

Ex : **$ (sleep 10; ps -l) & ps -l**

■ **wait** : attend la fin des processus en tâche de fond (commande interne au shell)

Syntaxe : wait [*pid*]

Attend la fin du processus fils dont le PID est ***pid***, et renvoie son statut de terminaison (émis par **exit**). Si ***pid*** est omis, attend la fin de tous les processus en tâche de fond courants. La commande **wait** est une commande de synchronisation interne au shell (elle est exécutée sans création d'un sous-processus).

Ex : Comparer :

$ ps -l & ps -l

$ ps -l & wait ; ps -l

■ **exit** : termine le shell (commande interne au shell)

Syntaxe : exit [n]

Sort du shell avec le statut **n** ou, si **n** est omis, le statut de la dernière commande exécutée. Le statut émis par une commande peut être récupéré dans le shell père par soit le statut de ls commande **wait** (cas asynchrone) soit dans la variable **$?** pour la dernière commande (cas synchrone).

Ex : **$ (echo fils; exit 50)**
$ echo $?
50
$ echo $?
0

Ex : **$ (echo fils; exit 50) &**
$ wait $!
$ echo $?
50

■ **time** : temps d'exécution d'une commande (commande interne au shell)

Syntaxe : time [-apv] commande

Ex : **$ time ps**
0.03user 0.07system 0:00.10elapsed 100%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+0minor)pagefaults 0swaps

22 Les Démons et processus système

Les démons (daemons) sont des processus système qui fonctionnent en permanence. Ce sont des fils du processus 1 (init). Ils se comportent généralement comme des serveurs à l'écoute de requêtes de processus clients.

- init : processus d'initialisation (boot avec /etc/ttys et /etc/inittab))
- bdflush et update : processus qui vident régulièrement les tampon de cache du disque (fichiers à accès bufferisés).
- swapper : processus réalisant les transfert entre mémoire vive et swap disque.
- pagedeamon : processus de pagination de la mémoire.
- crond : démon de commandes différées.
- lpd : démon de gestion des files d'impression.
- klogd : démon d'enregistrement des messages du noyau.
- syslogd : démon d'enregistrement et de distribution des messages du système (/etc/syslog.conf).
- inetd : super démon internet (/etc/inetd.conf).
  - in.telnetd :sous démon de inetd pour telnet
  - in.rshd : sous démon de inetd pour rsh
  - in.rlogind : sous démon de inetd pour rsh
- httpd : démon serveur http (serveur de pages html).
- sendmail : démon d'émission du courrier électronique (/etc/sendmail.cf).
- named : démon du service de correspondance de noms DNS (/etc/named.boot...).
- rpc.portmap : le démon des services rpc (dans /etc/rpc)
- rpc.mountd : démon rpc de montage à distance (sur le serveur).
- rpc.nfsd : démon serveur NFS (/etc/exports), utilise rpc.mountd.

23 Caractéristiques du terminal

■ **tty** : affichage du numéro de terminal

Syntaxe : tty

■ **stty** : affichage et définition des différentes options d'un terminal

Syntaxe : stty [a] [-g] [option]

-a : affiche toutes les définitions des options
-g : affichage des options courantes

Ex : **$ stty intr ^C** active la touche <ctrl>c pour interrompre de processus en cours.

```
$ stty -a
speed 9600 baud; rows 24; columns 80;
intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch = ^Z; start = ^Q; stop = ^S; susp = ^Z; dsusp = ^Y;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 4; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -ignbrk brkint ignpar -parmrk -inpck istrip -inl-
    cr -igncr icrnl ixon -ixoff -iuclc -ixany -imaxbel opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -
    ofdel nl0 cr0 tab0 bs0 vt0 ff0 isig icanon -iexten echo echoe echok -echonl -noflsh -xcase -
    tostop -echoprt -echoctl echoke
```

24 Contrôle des ressources disque

24.1 Commandes générales

■ **du** : occupation disque d'un répertoire

Syntaxe : du [-s] [répertoires]

Affiche l'occupation disque du répertoire courant ou les répertoires spécifiés ainsi que celle de tous les sous-répertoires sans l'option **-s**.

-s affiche seulement le total de chaque répertoire
-a génère une entrée pour chaque fichier

■ **quota** : occupation disque d'un utilisateur

Syntaxe : quota [-v] [utilisateur]

Affiche l'espace occupé sur disque par un utilisateur ainsi que ses limites. Seul le super-utilisateur peut visualiser les informations sur les autres utilisateurs.

-v concerne tous les systèmes de fichiers sur lesquels des quotas existent

■ **df** : occupation disque des systèmes de fichiers montés

Syntaxe : df [options] [répertoire/filesystem]

Affiche les quantités d'espace libre et occupé sur les systèmes de fichiers spécifiés. Sans paramètre, concerne sur tous les systèmes de fichiers montés.

-t typeconcerne les systèmes de fichiers d'un type donné (eg. nfs ou 4.2)
-a concerne tous les systèmes de fichiers y compris ceux qui ont 0 bloc au total (automounter)

24.2 Commandes spécifiques aux disquettes

■ Pilotes de disquette

Les pilotes de disquette généraux sont (voir **man fd**) :

/dev/fd0 pour le premier lecteur de disquette (souvent 3$_{1/2}$ 1.44 Mo),

/dev/fd1 pour le second (souvent 5$_{1/4}$ 1.2 Mo).

■ **fdformat** : formattage d'une disquette

Syntaxe : fdformat [options] pilote_disquette

Formate la disquette associée au pilote.

Ex : **$ fdformat /dev/fd0**

■ **mount, umount** : montage et démontage d'une disquette dos

Syntaxe : mount [-rv] [-t type] [-o options] pilote_disquette repertoire

Monte la disquette accédée par pilote_disquette dans le répertoire déjà créé **repertoire**. Il est ensuite possible de se déplacer dans le système de fichier de la disquette avec les commandes unix traditionnelles (ls, cd...)

-o *options* : spcécifie les options de montage. Les options sont séparées par des virgules (,). Certaines de ces options peuvent être :

ro ou **rw** (read-only) accès en lecture seule, ou (read-write) en lecture-écriture.

suid ou **nosuid** autorisation ou masquage des set uid bits.

Par défaut : rw,suid.

Ex : **$ mount -v -t pcfs -o rw,noauto /dev/fd0 /pcfs**

Mais, il faut avoir les permissions de root pour lancer cette commande. Une alternative consiste à ajouter dans le fichier /etc/fstab, la ligne :

/dev/fd0 /pcfs pcfs rw,user,noauto 0 0

Tout utilisateur (grâce à l'option **user**) pourra alors monter la disquette sous le répertoire/pcfs en tapant :

$ mount /pcfs

Pour démonter la disquette taper :

$ umount /pcfs

Consulter aussi man pcfs, man fstab.

■ **eject** : éjection d'une disquette

Syntaxe : eject

Ejecte automatiquement la disquette du lecteur (uniquement sur les systèmes disposant d'un lecteur à éjection automatique).

25 Connexion au système

■ **login** : connexion au système d'un utilisateur à partir d'un shell

Syntaxe : login [utilisateur]

Ex : **$ login**
login: **pierre**
Password:
SunOS Release 4.1.3C (GENERIC) #8: Fri Jul 9 15:10:57 PDT 1993
You have new mail.

■ **passwd, yppaswd** : modification du mot de passe de connexion

Syntaxe : passwd [utilisateur]

yppasswd [utilisateur] (en cas de pages jaunes NIS)

Un mot de passe doit avoir au moins 6 caractères et doit contenir au moins 2 caractères alphabétiques et un caractère spécial ou numérique.

Ex : **$ yppasswd**
Changing NIS password for pierre on balsa.
Old password:
New password:
Retype new password:

■ **su** : changement du nom d'utilisateur sans sortir du système

Il n'y a pas changement du répertoire courant (on ne se retrouve pas dans le répertoire habituel du login). On retourne à l'utilisateur précédent par la touche **^D**, ou la commande **exit**.

Ex : **$ su paul**
Password:

26 Communication entre utilisateurs

■ **write** : utilitaire de communication interactif entre terminaux

Syntaxe : write utilisateur [terminal]

Si l'utilisateur est connecté sur plusieurs terminaux, il faut spécifier lequel doit recevoir le message.

La commande attend que l'utilisateur entre son message, celui-ci est terminé par **^D**. Le message s'affiche ligne par ligne sur le terminal récepteur et finit par **<EOT>**.

■ **wall** : envoie un message à tous les utilisateurs connectés

Syntaxe : wall [-a] [fichier]

Envoie le fichier ou le message entré au clavier et validé par **^D** à tous les utilisateurs (sur leur console).

Seul le super-utilisateur peut passer outre les protections invoquées par les utilisateurs (cf. **mesg**)

-a envoie sur tous les terminaux

■ **mesg** : autorise ou interdit les messages

Syntaxe : mesg [n] [y]

mesg avec l'argument **n** interdit les messages sur le terminal, avec l'argument **y**, les autorise, sans argument, affiche l'état courant sans le modifier.

■ **talk** : établit une connexion interactive entre deux utilisateurs

Syntaxe : talk utilisateur [terminal]

talk est un programme de communication interactif entre deux utilisateurs. L'invocation de cette commande provoque sur le terminal du correspondant l'affichage d'un message lui demandant d'établir une liaison par la commande **talk**. Lorsque la connexion est établie, l'écran est coupé en deux verticalement, les deux correspondant écrivent simultanément, chacun dans sa moitié d'écran. ^L rafraîchit l'écran et ^C termine la session.

On peut joindre un utilisateur sur une autre machine en utilisant la syntaxe : **talk user@host**.

■ **who** : affichage de la liste des connexions au système

Syntaxe : who [option]

Affiche la liste des connexions au système : nom d'utilisateur et terminal utilisé.

■ **whoami** : affichage du nom de l'utilisateur connecté sur le terminal

27 Archivage et Compression de fichiers

27.1 Archivage avec tar (Tape ARchive)

Le programme d'archivage avec tar peut être rapprochée de la commande ms-dos backup.

A l'origine tar a été conçu pour l'interface avec des bandes de sauvegarde. Cette permet de créer une archive afin d'y d'ajouter, puis d'en extraire **séquentiellement** des fichiers ou des répertoires avec leur contenu.

Si l'on ajoute des fichiers qui se trouvent déjà dans l'archive, les fichiers sont ajoutés en fin d'archive et n'écrasent pas ceux qui existent déjà.

Le support de sortie est indiqué sous la forme d'un fichier-:

- /dev/hd0 : support périphérique bande
- /dec/fd0 : support périphérique disquette.
- file : un support fichier dans un répertoire.

Syntaxe : tar [-][rxtuc] [vwfbFmkenpA] <archive|périphérique> <liste fichiers|répertoires>

Options :

r ajoute des fichiers à la fin de l'archive existante
x extraction des fichiers de l'archive. Sans indication de fichier, c'est tout ce qui se trouve sur le support de données qui est extrait; les fichiers plus récents écrasent les plus anciens.
t lecture simple de l'archive sans écriture des fichiers inclus
u des fichiers sont archivés s'ils n'existent pas encore ou ont été modifiés depuis le dernier archivage.
c crée une nouvelle archive ; écrase l'ancien contenu
v (verbose) informations à l'écran pendant l'archivage
w traitement interactif avec demandes de confirmation
f l'argument suivant est un nom de périphérique ou de fichier archive
b<n>facteur de blocking; pour les systèmes à bande uniquement
F<nom>fichier dans lequel se trouvent les arguments suivants
l sortie de messages d'erreur si la résolution de liens de fichiers s'avère impossible
m la date et l'heure de la dernière modification sont maintenues; sans ce commutateur, ce sont la date et l'heure de la commande **tar** qui sont considérées.
k<n>spécifie la taille du support en Ko
e les gros fichiers ne sont pas découpés
p lors de l'extraction de fichiers les doits d'accès originaux sont conservés

Ex : **$ tar cf archi.tar . $ tar tf archi.tar $ tar xf archi.tar**
$ tar cf /dev/fd0 .

Attention : l'écriture et la lecture ne se différencient que par les commutateurs c et x.Si on observe le clavier, on voit que x et c sont juste à côté l'un de l'autre!

27.2 Compression de fichiers

compress, uncompress, gzip, gunzip

Syntaxe : compress fichier1[fichier2…]
Syntaxe : uncompress fichier1 [fichier2…]

La commande **compress** comprime les fichiers spécifiés et leur ajoute l'extension **.Z**. **uncompress** décomprime les fichiers comprimé avec **compress** et leur enlève l'extension **.Z**. Si l'opération aboutit, le fichier de départ est détruit.

compress et **uncompress** sont les commandes standard de compression sous UNIX, on leur préfère toutefois **gzip** et **gunzip** (distribution GNU) plus efficaces qui fonctionnent de façon identique, avec l'extension **.gz**.

Chapitre V Éditeur vi

Contrairement aux éditeurs modernes toujours prêts à intégrer au texte ce que l'on frappe, **vi** interprète le texte tapé au clavier comme une suite de commandes. Certaines permettent de se placer dans un mode spécial où l'on peut entrer du texte. Il faut donc un certain apprentissage pour maîtriser **vi** qui est loin d'être intuitif. Cependant, **vi** est **incontournable** dans le monde UNIX car c'est de loin l'éditeur le plus standard. C'est également un éditeur très complet et adapté à la programmation.

28 Modes de vi

Modes de vi

Mode commande/déplacement

C'est le mode dans lequel on trouve **vi** au lancement et après 'ESC'

Le texte frappé au clavier est interprété comme une suite de commandes mais n'est pas intégré au texte.

Ex : **j** : descend le curseur d'une ligne
D : efface la fin de la ligne
u : annule la dernière modification

Mode insertion

Certaines commandes permettent de faire entrer dans un mode d'insertion où le texte frappé est directement intégré au texte.

Ex : après **a**, **i**, **C**, etc…

Il est important de toujours savoir dans quel mode est **vi**.

Dans le doute : taper la touche **<esc>** qui fait revenir au mode commande avant de continuer.

29 Entrée dans l'éditeur

vi fichier édition de fichier
vi +n fichier édition de fichier avec positionnement sur la ligne n
vi + fichier avec positionnement sur dernière ligne
vi + /chaîne fichier avec positionnement sur ligne contenant chaîne

30 Syntaxe des commandes vi

La syntaxe simple de **vi** permet d'associer à une commande de modification de texte, un moyen de décrire sur quelle partie de texte elle doit agir.

Certaines commandes peuvent être précédées d'un nombre **[*n*]** indiquant que la commande doit être répétée ***n*** fois. Par défaut ***n*** vaut 1.

Commandes de déplacement

Syntaxe : **[*n*]*d***

d lettre ou mot de la commande de mouvement

Ex : **h** déplace d'un caractère à gauche; donc **5h** de 5 caractères à gauche.

Commandes d'altération du texte

Syntaxe : **[*n*]*c*[*d*]** ou ***c*[*nd*]**

c lettre de commande

d lettre ou mot de déplacement

Applique l'action définie par la commande ***c*** entre la position courante du curseur et la position désignée par le déplacement ***nd***.

Ex : **d** (**d**elete) est la commande d'effacement, donc
d5h ou **5dh** efface 5 caractères à gauche
d5G ou **5dG** efface de la ligne courante à la ligne n°5

Syntaxe : **[*n*]*c***

c est une lettre de commande

Certaines commandes définissent implicitement le texte sur laquelle elles agissent, elle n'utilisent pas d'argument de déplacement.

Ex : **x** : efface un caractère (équivalent à **dl**); **5x** : efface cinq caractères à droite.

Syntaxes particulières

Doublement du mot de commande (avec déplacement)

Pour les commandes qui nécessitent de spécifier un déplacement, le doublement du mot de commande vaut pour toute la ligne courante.

Ex : **dd** : efface la ligne courante;**yy** : copie la ligne courante;**cc** : change la ligne courante

Mot de commande en majuscule (avec déplacement)

Pour les commandes qui nécessitent de spécifier un déplacement, la mise en majuscules du mot de commande vaut pour la fin de la ligne courante

Ex : **D** : efface la fin de ligne;**C** : change la fin de ligne

Mot de commande en majuscule (sans déplacement)

Pour une commande qui contient implicitement une position, la lettre de commande majuscule entraîne une action identique à une position différente.

Ex : **I** : insère en début de ligne;**A** : ajoute en fin de ligne;**O** : ouvre une ligne avant;
P : colle le buffer avant;**X** : efface à gauche.

31 Principales commandes

Commandes de déplacement (*d*)

Déplacement relatif

[n]e fin du mot courant

mais encore :

Unité de déplacement	Déplacement en arrière	Déplacement vers l'avant
Colonne (caractère quelconque)	**h**	**l** <espace>
ligne (précédente/suivante)	**k ^P**	**j ^N**
ligne (premier/dernier caractère)	**O**	**$**
ligne (premier caractère imprimable)	**-**	**+**
recherche	**?**chaîne	**/**chaîne
Mot	**b**	**w**
Mot (ignore la ponctuation)	**B**	**W**
Page	**^B** (backward)	**^F** (forward)
Demi-page	**^U** (up)	**^D** (down)

Tableau 3 : Commandes de déplacement relatif

Déplacement absolu

[*n*]G ligne spécifiée (par défaut la dernière)
[*n*]| colonne n
[*n*]H première ligne de la page (ou énième ligne)
[*n*]L dernière ligne de la page (n-1 avant la fin)
M début de la ligne au milieu de l'écran

Recherche de chaîne

[*n*]f*car* place le curseur sur la n^{ième} occurrence de ***car*** sur la même ligne
[*n*]F*car* idem f vers le début de ligne
[*n*]t*car* idem t, place le curseur sur le caractère précédent
[*n*]T*car* idem F, place le curseur sur le caractère précédent
; répète la dernière commande f, F, t, T
; répète la dernière commande f, F, t, T dans le sens opposé
[*n*]/*chaîne* recherche de ***chaîne*** dans le fichier, après le curseur
[*n*]?*chaîne* idem, avant le curseur
n répète le dernier /, **?**
N répète le dernier /, **?** dans le sens opposé

Commandes d'altération de texte

En mode insertion (la commande est validée par 'ESC')

[*n*]i (**i**nsert) insère n fois avant le curseur
[*n*]I (**i**nsert) insère n fois avant le premier caractère imprimable de la ligne
[*n*]a (**a**ppend) insère n fois après le curseur
[*n*]A (**a**ppend) insère n fois en fin de ligne
o (**o**pen) insère sur une nouvelle ligne au-dessous
O (**o**pen) insère sur une nouvelle ligne au-dessus
c[*n*d] (**c**hange) change le texte jusqu'à **nd** par le texte inséré
[*n*]s (**s**ubstitute) change n caractères ('**s**' = '**ch**')

Dans ces commandes **^V** permet d'introduire des caractères spéciaux.

En mode commande

r (**r**eplace) remplace un caractère
d[*n*d] (**d**elete) efface le texte entre le curseur et **nd**
D efface la fin de la ligne
dd efface toute la ligne
y[*n*d] (**y**ank) copie le texte entre le curseur et **nd**
[*n*]p (**p**aste) copie n fois le buffer après le curseur
[*n*]x efface n caractères à droite ('**x**' = '**dl**')
[*n*]X efface n caractères à gauche ('**X**' = '**dh**')
[*n*]J (**j**oin) joint les n-1 lignes suivantes à la ligne courante
u (**u**ndo) annule la dernière modification
. (point) répète la dernière commande

Commandes de récupération

Il existe un buffer standard de récupération où se trouve le dernier texte coupé ou copié (équivalent au presse-papier des éditeurs modernes). Il existe une pile (1 à 9) où sont placés les dernières lignes et groupes de lignes supprimés. 26 autres buffers (a à z) où l'on peut stocker des morceaux de texte sont également disponibles.

Quand la lettre n'est pas spécifiée, le buffer standard est concerné.

"*lettre*	sélectionne un buffer avant une opération couper, copier ou coller
y[*n*]d	(**y**ank) copie le texte désigné par **[n]d**
Y	copie la ligne courante
p	transfère le contenu du buffer après le curseur
P	transfère le contenu du buffer avant le curseur

Commandes de marquage

Il est possible de mémoriser 26 (a à z) positions

m*lettre*	la position du curseur est mémorisée et pose d'une marque identifiée par la lettre ***lettre***.
'*lettre*	positionnement au début de la ligne où se trouve la marque ***lettre***.
`*lettre*	positionnement à l'endroit de la marque ***lettre***.

Commandes d'intérêt général

:f, ^G	référence du fichier et ligne en cours
:w	(**w**rite) sauvegarde du fichier
:q	(**q**uit) sortie avec contrôle de sauvegarde du tampon
:q!	sortie forcée (sans contrôle de sauvegarde)
:wq, ZZ	sortie de vi et sauvegarde du tampon sous le nom d'entrée
:e [+n] ***fichier***	(**e**dit) édition d'un fichier
:e! [+n] ***fichier***	édition d'un fichier sans contrôle de sauvegarde du tampon
:r ***fichier***	(**r**ead) ajout d'un fichier après la ligne courante
:! ***commande***	exécution d'une commande shell
^L	rafraîchissement de l'écran

32 Commandes avancées

32.1 Abréviations

ab : crée une abréviation

Syntaxe : :ab <abréviation> <texte complet>

La commande **:ab** permet de créer une abréviation qui pourra être utilisée avec toutes les commandes d'insertion de texte. À chaque fois que l'abréviation est tapée comme un mot entier (entouré de caractères de ponctuation, d'espaces…) **vi** la remplace par le texte complet spécifié. L'abréviation doit elle-même être un mot complet. En revanche, le texte complet peut intégrer des signes de ponctuation et même plusieurs lignes à condition de précéder le retour chariot par **^V**.

Ex :: **:ab AdrIR IRESTE^V^MRoute de Gachet^V^M44087 Nantes cedex 03**

unab : supprime une abréviation

Syntaxe : :unab <abréviation>

Note : pour désactiver la substitution juste une fois, il suffit de taper **^V** tout de suite après l'abréviation et avant le signe de ponctuation.

map, map! : crée une macro-commande

Syntaxe : :map <macro> <suite de commandes>

Sur le même principe que **ab**, **map** associe une séquence de touches avec une suite de commandes **vi.** La macro peut contenir des caractères quelconques mais ne doit pas commencer par une lettre de

commande **vi** (p, a, i, x…). Dans la séquence de commandes, les caractères spéciaux (<cr>, <esc>…) doivent être précédés par **^V**. La version **map** fonctionne en mode commande, alors que **map!** fonctionne en insertion de texte.

Ex : **:map // 0i/* ^V<esc>A */^V<esc>**

unmap, unmap! : supprime une macro-commande

Syntaxe : :unmap <macro>

32.2 Substitution

s : substitution

Syntaxe : :<intervalle>s/<chaîne1>/<chaîne2>/[g|c]

Dans chaque ligne de l'intervalle, on échange la chaîne de caractères <chaîne 1> par <chaîne 2>. L' intervalle est déterminé par :

%	tout le fichier
l1,l2	de la ligne l1 à la ligne l2
n	(nombre) ligne n°n
?<chaîne>?	ligne qui contient la précédente occurence de <chaîne>
/<chaîne>/	ligne qui contient l'occurence suivante de <chaîne>
.	ligne courante
$	dernière ligne du fichier

Les chaînes **<chaîne1>** et **<chaîne2>** peuvent contenir les caractères **^** et **$** :

- **^** début de ligne
- **$** fin de ligne

Deux options sont possibles : **g** et **c**.

- **g** (**g**lobal) agit sur toutes les occurrences dans la même ligne.
- **c** (**c**onfirm) demande confirmation à chaque occurrence trouvée.

Ex : **:.,$s/bien/très bien/** *Remplace, à partir de la ligne courante (**.,$**), la chaîne **bien** par **très bien**.*
%s/^/>/ *Place, au début (^) de chaque ligne (%), un caractère >.*

32.3 Utilisation de commandes Unix

:! : exécute une commande UNIX

Syntaxe : :<intervalle>!<commande>

Prend le texte de l'intervalle précisé et le remplace par son résultat.

Ex : **:%!sort** *Trie les lignes du fichier par ordre alphabétique*

32.4 Fichier de commandes

:so : interprète un fichier de commandes

Syntaxe : :so <fichier>

vi interprète chaque ligne du fichier comme une commande en la précédant de ':'.

33 Configuration de vi

Paramètres de vi

:set : affiche ou affecte la valeur des paramètres

:set all : affiche tous les paramètres et leurs valeurs

:set <paramètre> et **:set no<paramètre>** : paramètre vrai, faux

:set <paramètre> <valeur> : affecte la valeur au paramètre

Paramètres principaux :

[no]autoindent indentation automatique
[no]timeout temps maximal pour entrer les macro-commandes
tabstop nombre de colonnes pour un caractère <tab>
shiftwidth espacement des niveaux d'indentations (avec **<**, **>**, **^D**, **^T**)

fichier de configuration

vi démarre en interprétant, comme un fichier de commandes, le fichier **.exrc** du répertoire ($HOME) de l'utilisateur. Ce fichier peut contenir des abréviations, des macro-commandes, ainsi que des paramètres.

Chapitre VI Le Bourne Shell

L'interpréteur de commande étudié dans cette partie est le Bourne Shell (programme **sh**).

34 Mécanisme d'exécution des processus shell

Après le démarrage du système et la connexion d'un utilisateur, phases pendant lesquelles un certain nombre de processus systèmes sont lancés, le prompt du shell s'affiche : **$**.

A ce moment, pour chaque utilisateur connecté le système lance un processus : le shell de connexion (défini par l'administrateur du système dans le fichier de configuration du compte utilisateur : /etc/passwd). Ce shell entre alors en mode interactif et attend une commande que l'utilisateur tape une commande au clavier. La touche entrée (CR) provoque l'exécution par le shell de la commande.

Le shell exécute alors une commande de l'utilisateur en créant un processus fils.

Ex1 : **ps**

On note que l'utilisateur possède **2** processus en cours d'exécution :

sh et **ps** et que **ps est le fils de sh** (**sh** exécute **ps**).

Pour **exécuter** une commande, le shell utilise le mécanisme de **fork** (fourche) du langage C :

1- Le shell initial ou processus **père** est dupliqué dans un processus **fils**.
2- Le processus père (le shell) est placé en **attente**.
3- La commande à exécuter prend la place du processus shell fils.
4- Le processus fils s'exécute.
5- Après exécution le processus fils envoie un signal au processus père qui reprend la main.
6- La place occupée par le processus fils est alors libérée.

De plus, un processus **hérite** d'un certain nombre de données du processus père, à savoir :

- L'**environnement** du processus père (dont il a en partie hérité). C'est-à-dire, l'ensemble des variables qui ont été définies comme exportables dans le processus père.
- Les **fichiers** ouverts par le père (ou dont celui-ci a hérité).

35 La redirection des entrées/sorties

Les 3 fichiers ouverts en début de session sont :

-.**L'entrée standard** (descripteur **0**) : le clavier

- **La sortie standard** (descripteur **1**) : l'écran

- **La sortie erreur standard** (descripteur **2**) : l'écran

• Redirection des sorties

commande > fichier ou **commande >> fichier**

• Redirection de l'entrée

commande < fichier ou **commande << mot**

Redirections simultanées

Ex : **sort < avant > apres** trie "avant," inscrit le résultat dans "apres"

sort < fichier > fichier ne fonctionne pas !

On peut rediriger la sortie standard vers l'erreur standard ou vice-versa. (Attention, contrairement aux précédentes, ce type de redirection est spécifique au Bourne shell) :

Ex : **commande 2>&1 1>/dev/null**

affecte l'erreur standard au fichier précédemment destiné à la sortie standard puis redirige la sortie standard vers **/dev/null**.

commande 1>/dev/null 2>&1

redirige la sortie standard vers **/dev/null** puis redirige l'erreur standard vers le fichier précédemment affecté à la sortie standard, c'est-à-dire **/dev/null** !

Chaînage des commandes : **'pipeline'**

commande1 | commande2

• Substitution du résultat d'une commande

commande2 `commande1`

Ce mécanisme n'est pas une redirection mais une substitution de texte réalisée par le shell. L'argument de commande2 est le résultat renvoyé par l'exécution de commande1. Par exemple une liste de nom de fichiers obtenue par la commande **find** (ou **grep**…) est passée à une autre commande qui va donc agir uniquement sur ces fichiers.

L'ordre d'exécution est le suivant :

-1- La commande *commande1* encadrée par des caractères backquotes (`);

-2- La commande *commande2*, avec le paramètre chaîne de caractères résultant de l'exécution de *commande1*.

Ex : **ls -lg `grep -l main*`** (Liste en détail les noms de fichiers commençant par "main".)
mv -i `find . -atime +7 -name '*~' -print` .wastebasket (Déplace vers **.wastebasket** les fichiers, dont le nom fini par **~**, qui n'ont pas été lus depuis 7 jours.)

36 Exécution de commandes par le Bourne shell

sh est l'interpréteur (SHELL) de commandes écrites en BOURNE SHELL. sh permet d'exécuter différent types de commandes :

■ **Commande simple :**

Une chaîne de caractères comportant le nom de la commande et un ou plusieurs arguments.

La valeur d'une commande simple (donnée par **$**?) est son code retour si son appel se termine correctement, 200 + code retour en cas d'erreur.

■ **commande pipeline :**

Séquence d'une ou plusieurs commandes séparées par le caractère |.

■ **Liste de commandes:**

Séquence d'un ou plusieurs commandes séparés par :

; exécution **séquentielle (synchrone)**,

& exécution **asynchrone** (tâche de fond),

&& et || exécution **conditionnelle**;

et terminée facultativement par : **;** ou **&**.

Dans tous les cas, il faut que la liste d'instruction se termine par la touche entrée (CR) pour être interprétée.

37 fichiers de commandes

Il est possible de créer un fichier ascii, à partir d'un éditeur de texte, contenant une suite de commandes shell afin de les automatiser (shell script ou traitement par lot). Il y a deux possibilité pour l'exécuter :

sh nom_fichier [arguments]si le fichier est accessible en lecture.

nom-fichier [arguments] ou

./nom-fichier [arguments]si le fichier est accessible en exécution.

Dans le premier cas il y a de création d'un sous-shell (pas dans le deuxième).

Il est possible d'insérer des commentaires après le caractère **#** :

 # commentaires la proportion de lignes à droite du # est ignorée.

Des variables dans les fichiers de commandes

Il est possible de définir des variables dans les fichiers de commandes. Chaque variables a pour valeur une chaîne de caractères qui lui est substituée, grâce au caractère $, à l'exécution du fichier de commande. L'utilisation en est la suivante :

nom_var=une_chaîne_de_caractèrespour l'affectation de la valeur.

set nom_var=une_chaîne_de_caractères2ème possibilité d'affectation de la valeur.

$nom_varpour la substitution de la valeur.

unset nom_varpour supprimer la variable nom_var.

Ex:

pwd : #donne le répertoire en cours

c=`pwd` : #affecte à c la valeur du répertoire en cours

echo $c : #affiche le répertoire en cours

Les arguments du fichier de commande

Il est possible de transmettre des arguments à la suite du nom du fichier de commande, comme pour ls commandes Unix habituelles, à son l'appel.

Ces arguments sont référencés dans le fichier par les variables $0, $1, $2, ..., $9

$0 Le nom de la commande
$1 le premier argument
$n le ième argument

Voir aussi l'opérateur **shift**.

Les autres variables prédéfinis :

$0 : nom de la commande
$# : nombre de paramètres d'appel
$n : valeur du nième paramètre d'appel
$* : liste des paramètres d'appel de la commande
$? : valeur du code retour de la dernière commande exécutée
$$: numéro de processus en cours
$! : numéro du dernier processus lancé et en cours en arrière-plan

L'opérateur shift (décalage des paramètres).

Cet opérateur permet de décaler d'un argument la liste des paramètres d'appel en renommant les paramètres à partir du second (le premier est donc perdu).

Affinage du mécanisme de substitution

Il est possible de jouer sur le mécanisme de substitution de valeur :

${variable:-mot} : substitue la valeur de la variable si elle est définie, sinon le mot (sans modifier la variable).
${variable:=mot} : idem, mais modifie la variable
${variable:+mot} : substitue le mot si la variable est définie, sinon la chaîne vide (variable non modifiée)
${variable:?mot} : substitue la valeur de la variable si elle est définie, sinon le mot; puis sort du shell.

Différence entre les deux délimiteurs ' et "

Dans une chaîne délimitée par **"** le mécanisme de substitution de variables est opérant, alors qu'il ne l'est pas avec **'**

Ex: **$ c=youcou** #affecte à **c** une chaîne de caractères
$ echo "$c" #affiche les caractères **youcou**
$ echo '$c'** #affiche les caractères **$c

Exportation de variables

Un shell appelé hérite des variables de ses ascendants qui sont déclarés dans ceux-ci par la commande :

Syntaxe : export variable

Les variables exportées sont dupliquées dans chaque shell fils. Elles y ont donc une valeur locale, initialisée à l'appel dans le shell père, et leur modification est sans effet sur le shell père.

En cas d'homonymie, la valeur exportée est celle dans le plus proche parent ayant demandé l'exportation.

La commande **env** permet de connaître, dans un shell, la liste des variables exportables (transmises à la descendance).

37.1 structure conditionnelle if then else

```
if <cond1>
then <action1>
[elif <cond2>
then <action2>]
[else <action3<]
fi
if <cond1>; then
    <action1>
[elif <cond2>; then
    <action2>]
[else
    <action3<]
fi
```

Abréviations possibles :

cond1 && action1 (au lieu de : **if cond1 then action1**)
cond1 || action1(pour exécuter action1 si cond1 a échoué)

« **Action** » peut être une commande ou une procédure,
« **cond** » doit rendre comme état de fin **0** (pour vrai!) ou **1** (pour faux!).

Pour cela, on utilise fréquemment la commande du SHELL aux talents multiples : test.

Commande : **test**
Syntaxe1:
test -[rwxfdsbugkstzn] <argument> commande _interactive
Syntaxe 2 :
[-[rwxfdcbugkstzn] <argument>] dans les commandes if-then-else.
La commande test renvoie 0 si le test est valide, 1 sinon.
Les options suivantes définissent les tests possibles:

-r <nom> le fichier existe avec autorisation de lire
-w <nom> le fichier existe avec autorisation d'écrire
-x <nom> le fichier existe avec autorisation d'exécuter
-f <nom> fichier existant
-d <nom> répertoire de fichier existant
-s <nom> le fichier existe et n'est pas vide
-b <nom> le fichier est orienté caractères et existe
-z <nom> la longueur de la suite de caractères est 0 (vide)
-n <nom> longueur de la suite de caractères >0
<s1>=<s2> les chaînes s1 et s2 sont identiques (espaces!)
<s1>!=<s2> s1 est différent de s2 (espaces!)
!<condition> négation de la condition
<cond1>-a<cond2> ET logique
<cond1>-0<cond2> OU logique (inclusif)
\(cond1cond2cond3\) conditions à réunir dans un groupe

Comparaison algébrique : **val1 -opérateur val2** :

<val1>-eq <val2> signifie val1 = val2
<val1>-ne <val2> signifie val1 <> val2
<val1>-gt <val2> signifie val1 > val2
<val1>-ge <val2> signifie val1 >= val2
<val1>-lt <val2> signifie val1 < val2
<val1>-le <val2> signifie val1 =< val2

37.2 structure conditionnelle case

La construction case permet de vérifier si une expression (variable $VAR, $1 ou une constante) figure dans la liste de modèles.

```
case <expression> in
    modele)    action;;
    modele1 | modele2)action;;
    modele)    action
esac
```

Toutes les actions (commandes ou procédures) jusqu'à la dernière, doivent être suivies de « **;;** ».

Plusieurs modèles peuvent être liés à la même action par | (correspond à OU). Si l'on utilise * comme modèle, aucun autre modèle ne pourra plus être comparé.

37.3 structure itérative for

Une boucle for avec une fin et un pas définis permet d'exécuter la même action pour une liste d'objets.

```
for <variable> [in<liste>]
do action
done
```

La valeur de la liste est assignée à la variable avant chaque passage dans la boucle.

37.4 structures conditionnelles while et until

Contrairement aux boucles précédentes avec une fin définie (nombre de paramètres par exemple), il existe des boucles qui ne se terminent que pour une certaine condition.

La boucle **while** est exécutée tant que la condition est remplie, c'est-à-dire qu'elle renvoie 0;

elle est orientée vers la réalisation de la condition.

La boucle until est exécutée tant qu'une certaine condition n'est pas remplie, que l'état est égal à 1.

```
while <condition> ; do
    <action>
done
until <condition> ; do
    <action>
done
```

38 séquence de connexion

La séquence de connexion est initialisée par le démon **getty** qui attend qu'un utilisateur se manifeste. Selon les informations contenues dans le fichier /etc/gettydefs, les paramètres de ligne sont ajustés et le prompt par défaut s'affiche. Quand l'utilisateur a validé son nom par la touche <Return>, getty passe la main au programme **login** qui demande alors le mot de passe. En le comparant à celui du fichier /etc/passwd, le programme accepte ou non la connexion. Si le mot de passe est incorrect, c'est cette fois-ci le programme login qui demande à nouveau le nom de l'utilisateur et le prompt "login:" est alors affiché. La bannière unix est affiché par getty et peut être n'importe quelle chaîne de caractère définie dans le fichier /etc/gettydefs.

Le programme login change ensuite le propriétaire du périphérique /dev/ttyxx sur lequel est connecté l'utilisateur. Durant toute la durée de la connexion, le propriétaire est l'utilisateur. Lorsque ce dernier se déconnecte le programme login rétablit le propriétaire original (généralement root).

Puis, le contrôle est donné au **shell** par défaut indiqué dans le fichier passwd (sh,csh,ksh... ou en fait n'importe quel autre programme). En l'exécutant, login positionne dans son environnement des variables telles que HOME (voir plus loin).

L'enchaînement des programmes getty, login puis du shell se fait par une série d'appels système **fork()**, la seule manière que connaisse Unix pour créer des processus.

Les fichiers d'initialisation du shell dépend de la nature de ce dernier. Dans tous les cas, un **script d'initialisation global** est exécuté, suivi par un **script propre à l'utilisateur** qui se trouve dans le répertoire de connexion de ce dernier (**.profile** pour le Bourne Shell).

39 variables d'environnement

Tout utilisateur démarrant une session dispose d'un environnement standard qu'il peut personnaliser. Cet environnement est caractérisé par un ensemble de variables prédéfinies qu'il peut modifier. Il peut également définir d'autres variables.

Lors de l'exécution d'un fichier de commandes, on a vu qu'il y avait ou non création d'un sous-shell (**sh01.sh**). Dans le premier cas (par défaut), seules les variables marquées en vue d'une exportation dans le shell père sont connues du shell fils. Dans le deuxième cas (**exec**), toutes les variables sont connues puisqu'on est toujours dans le même shell.

Exportation de variables

La commande **export** permet de marquer les variables destinées à être exportées (transmises aux processus fils par recopie).

exportliste les variables exportées.

export nom_de_variablemarque la variable en vue de son exportation.

Aucune modification des variables d'un shell fils n'a d'effet sur les variables de même nom du shell père.

La commande **env** liste toutes les variables exportables (idem **export** + les variables héritées).

Lors de la procédure de connexion, des variables d'environnement prédéfinies sont positionnées par le programme login selon les informations contenues dans le fichier passwd.

PATH: la liste des répertoires de recherche des exécutables (séparée par des **:**).

HOME: le répertoire de connexion de l'utilisateur tel qu'il est spécifié dans le fichier passwd.

SHELL: le nom du shell qui est exécuté

MAIL: est positionné par défaut à /usr/mail/<user> ou <user> est le nom de connexion de l'utilisateur tel que trouvé dans le fichier passwd

IFS: Définit l'ensembles des caractères de séparation assimilés à un espace (par défaut : espace, tabulation, CR)

USER ou **LOGNAME** (selon les systèmes): indiquent le nom de compte de l'utilisateur.

40 scripts d'initialisation à la connexion

Ces scripts sont appelés automatiquement par les shells lors de la connexion de l'utilisateur. Il existe **un script système** commun à tous les utilisateurs qui est exécuté en premier, puis **un script personnel** pour chaque utilisateur. Le script personnalisé configure généralement l'environnement de l'utilisateur, telles des variables shell ou d'environnement.

En Bourne Shell, le script global est défini dans **/etc/profile** et le script personnel dans **$HOME/.profile** (voir ces fichiers).

Des variables importantes doivent être initialisées dans le script de connexion général.

TERM: définit le type de terminal. Utilisé par vi, more et généralement tous les programmes qui affichent des données à l'écran. La commande tset permet de faire correspondre un nom de terminal à un périphérique de terminal.

TMOUT: indique au shell le temps d'inactivité maximal autorisé. Une valeur de 0 indique l'éternité.

HZ: la fréquence horloge de la machine

TZ: (Time Zone) donne les informations sur le fuseau horaire. Des commandes comme date ou mail utilisent cette variable pour convertir l'heure du système en un format lisible.

D’autres variables spécifiques aux shell sont positionnées dans les scripts de connexion personnel.

PWD: indique le chemin d’accès au répertoire courant

HISTFILE: indique le fichier dans lequel seront enregistrées les commandes pour l’historique. par défaut c’est le fichier **.sh_history** dans le répertoire de connexion

HISTSIZE: précise le nombre de commandes pouvant être enregistrées dans le fichier HISTFILE

LANG: indique le langage utilisé par certaines applications

Chapitre VII Communications réseau

Elles se divisent en deux catégories. La première permet de communiquer avec des systèmes hétérogènes (non Unix) elle comprend les commandes **ftp** et **telnet**. La deuxième est formé des commandes qui sont spécifiques aux systèmes Unix. Il s'agit des commandes à distances **rlogin**, **rcp**, **rsh**,...

41 mécanisme d'autorisation.

Ce mécanisme est utilisé par les commandes à distances préfixées par **r** (ou r-commandes). La machine cliente interprétant la commande peut ne plus s'authentifier par mot de passe auprès du serveur distant. Une condition préalable pour qu'un serveur puisse autoriser une machine cliente est que celle-ci soit déclarée dans le fichier **/etc/hosts** du serveur. Cette opération peut être centralisé ou non.

Dans le premier cas, c'est l'administrateur système du serveur qui centralise les autorisations. Il utilise pour cela le fichier **/etc/hosts.equiv** qu permet de configurer un ensemble de machines en réseau.

Dans le deuxième cas, c'est l'utilisateur qui déclare les autorisations le concernant dans un fichier .**rhosts** situé dans son répertoire de connexion $**HOME**.

Dans ce paragraphe nous ne décrirons que l'autorisation gérée par le fichier **.rhosts** de l'utilisateur. Ce fichier contient une ligne par autorisation selon le format : **hostname [logname]**

Dans le cas où la ligne est seulement composée d'un nom de machine, l'autorisation s'applique à l'utilisateur de même nom de login venant de la machine ***hostname*.**

Dans le cas plus général où la ligne est composée d'un nom de machine, d'un espace et d'un nom de login, l'autorisation s'applique à cet utilisateur.

Le fichier .**rhosts** est soumis à deux ou trois contraintes de sécurité. Si ces contraintes ne sont pas respectées, il sera ignoré. Ces contraintes sont les suivantes :

- son propriétaire est identique à celui du répertoire **HOME** qui le contient ;
- ce n'est pas un lien symbolique;
- ses permissions sont **-rw-------**.

Ex : Le fichier .**rhosts** ci-dessous est situé dans le répertoire **HOME** de l'utilisateur jean de la machine osiris. Il autorise pour les r-commandes les utilisateurs paul, xavier et jean de la machine isis, et marie depuis osiris.

```
# .rhosts
# hostname [logname]
isis
isis paul
isis xavier
osiris marie
```

42 Connexion à distance entre machines Unix

rlogin hostname [-l user]

La commande **rlogin** permet le login sur une machine Unix distante. Lors du login il y a contrôle du mot de passe lorsqu'aucun mécanisme d'autorisation n'est déclaré.

Ex : $ **rlogin osiris -l jean**

Cette commande sera autorisée avec le fichier **.rhosts** du paragraphe précédent, situé dans le répertoire **HOME** de l'utilisateur jean de la machine osiris, qui permet l'accès par **rlogin** à trois utilisateurs d'isis et à un utilisateur d'osiris.

43 Connexion entre machines distantes quelconques

telnet hostname

La commande **telnet** permet, comme **rlogin,** la connexion sur un système distant. Mais ici, le mot de passe et le nom de login doivent obligatoirement être fournis lors de la connexion.

Ex :
```
$ telnet tanis
Trying 193.52.85.12 ...
Connected to tanis.
Escape character is '^]'

SunOS Unix (tanis)
login:
```

La commande **telnet** possède une aide en ligne accessible à partir du mode commande, lui même accessible par le caractère d'échappement :

Ex :
```
$ telnet
telnet> ?
Commands may be abreviated. Commands are:
close     close current connection
display   display operating parameters
mode      try to enter line-by-line or character-at-a-time mode
open      connect to a site
.....
?   print help information
telnet> ?
```

Exécution à distance

rsh [-l user] hostname commande

La commande ***rsh*** permet de faire exécuter une commande sur une machine distante. Cette commande ne fonctionne que dans le cas où il y a un mécanisme d'autorisation.

Ex : **$ rsh osiris uptime 2:26pm up 2 days., 8:23, 37 users, load average : 0.08, 0.12, 0.00**

Pour lire à distance une bande au format tar qui se trouve dans le dérouleur de la machine osiris :

Ex : $ **rsh osiris dd if=/dev/rmt0m bs=20b | tar xpbf 20 -**

Pour transférer un répertoire **HOME** d'une machine à une autre en conservant les permissions :

Ex : $ **rsh osiris -l peli tar cf - . | tar xpf -**

Pour lister les fichiers **main.c** (avec mise en page : ***pr***) sur l'imprimante **lw** de la machine isis :

Ex : $ **pr main.c | rsh isis lpr -Plw**

44 Transfert de fichiers à distance

Syntaxe : **rcp** ***source destination***

La commande **rcp** permet de copier une arborescence de fichier d'un système à un autre. Elle ne fonctionne que dans le cas où il y a un mécanisme d'équivalence. Cependant, une fois la barrière du contrôle franchie, l'accès à tous les fichiers public est possible.

Ex : L'utilisateur paul de la machine isis effectue la commande suivante qui copie dans le répertoire courant le fichier **/u/projet**/ax.c de la machine osiris.

```
$ rcp osiris.jean:/u/projet/ax.c  .# avec la syntaxe BSD 4.2
$ rcp jean@osiris:/u/projet/ax.c  .# avec la syntaxe BSD 4.3
```

Les deux conditions requises pour le bon fonctionnement de cette commande sont :

- l'utilisateur est déclaré autorisé pour le **login** qu'il utilise sur la machine distante ; cela signifie que l'utilisateur jean de la machine osiris possède dans son répertoire **HOME** un fichier .**rhosts** contenant la ligne suivante : **isis paul** ;
- l'utilisateur jean de la machine osiris a le droit d'accès en lecture sur le fichier **/projet/ax.c** que paul va copier sur isis.

La copie entre deux machines peut être effectuée depuis une tierce machine.

Ex : Ici paul depuis la machine isis copie le fichier .**login** de l'utilisateur alfred de la machine tanis dans le répertoire **HOME** de l'utilisateur marie sur la machine osiris. Cette commande ne fonctionne que si le fichier .**rhosts** de alfred sur tanis contient la ligne isis paul et si le .**rhosts** de marie sur osiris contient la ligne tanis alfred.

```
$ rcp alfred@tanis:.login  marie@osiris:
```

45 La commande *rusers*

Syntaxe : **rusers [-al] [host ...]**

La commande ***rusers*** est similaire à la commande ***who*** mais pour une liste de machine ou tout le réseau local. Pour chaque machine répondant à la requête de la commande **rusers**, est imprimé à l'écran le nom de la machine distante avec tous les noms d'utilisateurs connectés.

Les options suivantes sont disponibles :

- **-a** Affiche le nom de toutes les machines qui répondent même si aucun utilisateur n'est connecté.
- **-l** Affiche au format long. Cela inclut le nom d'utilisateur, le **hostname**, le **tty** de l'utilisateur loggé, la date et l'heure de début de login, le temps depuis la dernière frappe de touche sur le clavier, et la machine distante depuis laquelle il s'est loggé (si possible).

46 Redirection de l'écran X11

Dans l'environnement graphique X11, il est possible de combiner l'exécution à distance d'une commande utilisant les services X11 et la redirection de l'affichage sur la machine demandant l'exécution distante. Il est nécessaire pour cela d'ajuster la variable **DISPLAY** et d'utiliser la commande **xhost**.

La variable d'environnement DISPLAY

Cette variable d'environnement permet de désigner l'écran graphique utilisé par les processus clients du serveur **X**. Dans sa forme la plus complète, la syntaxe de la variable **DISPLAY** est :

hostname:i.[j] ou **adresse_IP:i.[j]**.

La lettre **i** représente le numéro du serveur **X** que l'on souhaite atteindre. Certaines stations de travail sont en effet capables d'exécuter plusieurs serveurs X. La valeur 0 désigne le premier serveur **X**.

Le serveur **X** pilote en général un clavier, une souris, et un écran. Toutefois certaines configurations pilotent plusieurs écrans ; dans ce cas **j** est utilisé pour désigné l'écran.

En général DISPLAY vaut : hostname**:0.0**.

Ex : (En Bourne shell)

```
$ DISPLAY=ramses:0.0; export DISPLAY
$ DISPLAY=193.52.85.25:0.0; export DISPLAY
```

Si à partir de la station de travail ramses on ouvre un connexion **telnet** sur la machine isis, tous les clients lancés à partir de cette session s'afficheront sur le serveur **X** ramses si sur isis la variable d'environnement **DISPLAY** vaut **ramses:0.0**.

Notons que la variable DISPLAY est une commodité puisque tous les clients **X** acceptent l'option **-display** permettant de désigner explicitement le serveur **X** choisi.

Ex : **$ xterm -display ramses:0.0 &**

La commande *xhost*

Le serveur **X** scrute au moment de son lancement le fichier **/etc/X0.xhosts** qui contient la liste des machines autorisées. La commande **xhost** permet de contrôler le mécanisme d'autorisation en mettant en ou hors service une machine, et en permettant de modifier la liste initiale.

Ex :

```
$ xhost +
access control disabled, clients can connect from any host
$ xhost -
access control disabled, only authorized clients can connect
$ xhost + horus
horus being added to access control list
$ xhost
access control enabled, only authorized clients can connect
horus
```

Autoriser l'accès au serveur **X** n'est pas sans conséquences pour la sécurité. Il est en effet possible de capturer la totalité des informations affichés sur le serveur **X** ainsi que la totalité des événements. Chaque frappe d'une touche du clavier se traduisant par un événement **X**, il est donc possible de capturer les mots de passe au moment de leur introduction !

47 Transfert de fichiers à distance

La commande **ftp** ne permet qu'un nombre limité de commandes sur le système distant. Les commandes **ftp** sont filtrées par le serveur qui les transforme en commandes adaptées au système d'exploitation du serveur. On a ainsi un fonctionnement indépendant du système d'exploitation. Ces commandes sont du type transfert de fichier, listage de répertoire, effacement de fichiers, changement de répertoire ...

On doit fournir un nom de **login** et un mot de passes pour effectuer le **« pseudo login »**. Cette opération est contrôlée à l'aide du fichier **/etc/ftpusers** situé sur le serveur. Elle interdite pour tous les noms de login déclarés dans ce fichier.

```
# /etc/ftpusers
root
uucp
invite
```

On peut automatiser cette opération de **« pseudo login »** grâce au fichier **.netrc** situé dans le répertoire **HOME** de l'utilisateur de la machine cliente. L'utilisateur déclare dans son fichier **.netrc** le nom de la machine serveur, un nom de login sur cette machine, et le mot de passe correspondant en

clair. Ce fichier doit obligatoirement être protégé par les permissions **-rw-------**. La plupart des versions de **ftp** imposent ces permissions.

```
# .netrc
machine isis login peli password alpha1
machine osiris login peli passwd alpha2
machine eole login peli
macdef init
binary
```

48 Le courrier électronique

La commande **mail** est utilisée pour l'émission et la réception de courrier électronique entre utilisateurs de systèmes informatiques. Un message peut être envoyé à un ou plusieurs utilisateurs et il est possible, pour des diffusions générales, de définir des listes d'utilisateurs.

Pour être prévenu de l'arrivée de courrier dans votre boîte aux lettres vous devez informer le système. Cela peut être fait par l'initialisation des variables d'environnement **MAIL** et **MAILCHECK** dans le fichier d'initialisation votre shell de connexion. Exemple pour le Bourne shell, ajouter dans **.profile**

```
$ MAIL=/usr/spool/mail/$USER
$ MAILCHECK=600
$ export MAIL MAILCHECK
```

Appelé sans argument la commande **mail** permet de lire le courrier. Si la boîte aux lettres es vide elle vous retourne le message **no mail for logname** ; dans le cas contraire elle affiche le sujet des messages reçus.

Utilisation de la commande *mail*

Pour lire un message il suffit d'entrer son numéro d'ordre après le caractère d'invitation **&**. Lorsqu'on utilise un ou plusieurs arguments, **mail** permet d'émettre du courrier. L'utilisation de base est représentée par l'exemple suivant :

```
$mail marchand@horus
Subjetct: Presentation SunOS 5.0
La présentation des communications évoluées du Systeme V a lieu Jeudi 23 Mai à 13H30.
Cc:ophilipp@lati
```

Les adresses peuvent être remplacées par un **alias** décrivant une liste d'utilisateurs. Les **alias** personnels sont décrits dans le fichier **.mailrc.** situé dans le répertoire HOME de l'utilisateur; les alias d'intérêt général sont décrits dans le fichier /usr/lib/aliases géré par l'administrateur système.

Paramètres de la commande *mail* : .mailrc

Le fichier **.mailrc** situé dans le répertoire de chaque utilisateur permet de personnaliser les modes de fonctionnement par défaut de la commande **mail.** Toutefois le fichier d'initialisation général **/usr/lib/Mail.rc** positionne les options les plus intéressantes. Ce dernier fichier est géré par l'administrateur du système.

```
# .mailrc
set SHELL=/bin/csh
set EDITOR=/usr/ucb/ex
set VISUAL=/usr/ucb/vi
set folder=~/mail
set record=~/mail/sent
set ask askcc autoprint dot msgprompt save
set crt=23
alias ccl meunier peli soulay zeyons
```

Il existe également la possibilité de définir des listes d'utilisateurs à l'aide d'**alias** soit dans le fichier **.mailrc** soit dans le fichier **/usr/lib/aliases**. Ce dernier est géré par l'administrateur du système.

Les options de la commande **mail** sont contrôlées à l'aide des commandes set et **unset.** Il existe deux types d'options : les options booléennes qui sont positionnées par la commande **set option** et annulées par unset option ; les options initialisées par une chaîne de caractères par la commande

set option=valeur.

Les options booléennes :

- **append** : les messages sauvés dans le fichier **mbox** sont ajoutés à la fin du fichier **mbox** au lieu du début.
- **ask** : lors de l'émission d'un courrier, **mail** commence par vous en demander le sujet.
- **askcc** : lors de l'émission d'un courrier, **mail** vous demande si vous voulez envoyer des duplicatas (**Carbon Copy**) au moyen de l'invitation **Cc** :
- **autoprint** : même comportement pour la commande **delete** que la commande **dp**.
- **dot** : caractère point en colonne 1 suivi du retour ligne repéré comme la fin de message.
- **hold** : rangement après lecture d'un courrier des lettres dans le fichier **mbox**.
- **ignore** : les interruptions ne sont pas prises en compte et sont remplacées par l'affichage du caractère **@**.
- **ignoreeof** : n'accepte pas **Control-D** comme fin de message, seule la sortie par « **.Return** » fonctionne.
- **msgprompt** : indication sommaire de la manière d'opérer pour envoyer une lettre.
- **metoo** : (moi aussi) lorsque vous envoyez un courrier à une liste d'utilisateurs définie par **alias, mail** ne vous adresse pas la lettre si vous faites partie de cette liste. **metoo** vous permet également d'être destinataire.
- **quiet** : suppression de l'affichage de la version du mail.
- **verbose** : affiche la lettre envoyée sur le terminal utilisateur.

Les options initialisées par des chaînes de caractères :

- **EDITOR** : chemin d'accès complet de l'éditeur invoqué à l'aide de la commande **~e**.
- **SHELL** : chemin d'accès complet du shell invoqué par la commende **!** ou **~!**.
- **VISUAL** : chemin d'accès complet de l'éditeur invoqué par la commande **~v.**
- **crt** : seuil pour déterminer quelle doit être la longueur d'un message pour qu'il soit filtré par la commande **more**.
- **escape** : caractère indiquant le caractère d'échappement qui se substitue au caractère standard ~.
- **folder** : nom du répertoire dans lequel on range ses dossiers courrier.
- **toplines** : nombre de ligne affichées par la commande **top**. Par défaut les cinq premières lignes sont affichées.

Les principales commandes de mail.

Il existe deux types de commandes. Les premières sont utilisées en mode réception. Les autres sont utilisées en mode émission et sont toutes précédées du caractères d'échappement ~.

Les commandes du mode réception.

- **-** ou **-n** affiche le message ou les n messages précédents.
- **?** ou **help** affiche un bref résumé des commandes.
- **!cmd** exécute la commande shell cmd.
- **print (p)** affiche le message suivant ou la liste de message indiquée en argument.
- **Print (P)** ou **Type (T)** affiche le message suivant en omettant l'en-tête.
- **Reply (R)** en lecture du courrier passe en mode émission pour répondre à la lettre qui vient d'être lue.
- **alias (a)** sans arguments affiche tous les **alias** connus par **mail** ; avec un argument, affiche l'**alias** désigné par cet argument ; avec au moins deux arguments, crée un nouvel **alias** valable pour la session **mail** courante seulement.
- **dp** efface la lettre courante et affiche la suivante.
- **exit (ex,x)** sortie sans modification de la boîte aux lettres.
- **quit (q)** sortie avec rangement dans le fichier **mbox** de tous les messages lus et non effacés.

Les commandes en mode émission.

- **~!cmd** exécute la commande **cmd**,
- **~?** Affiche un résumé des commandes du mode émission,
- **~:cmd** exécuté la commande **cmd** qui est une commande du mode réception,
- **~e** appel l'éditeur désigné par **EDITOR** dans le fichier .**mailrc** sur le texte en cours de frappe,
- **~v** appel l'éditeur désigné par **VISUAL** dans le fichier .**mailrc** sur le texte en cours de frappe,
- **~f file** inclusion du fichier **file** dans le texte en cours de frappe,
- **~m file** inclusion du fichier **file** dans le texte en cours de frappe en indentant ce texte par rapport au texte courant,
- **~q** annulation du texte en cours de frappe et rangement de ce texte dans le fichier **dead.lettre**,
- **~r file** inclusion du fichier **file** dans le message en cours,
- **~s string** substitution du sujet du courrier par la chaîne **string,**
- **~t names** ...ajoute les noms indiqués à la liste des destinataires de la lettre,
- **~w file** range le texte de la lettre en cours dans le fichier **file**,
- **~|cmd** filtre le texte de la lettre à l'aide du filtre désigné par **cmd** ; les filtres habituellement utilisés sont la commande *fmt* pour effacer une remise en page du texte ou les commandes ***lp*** et ***lpr*** pour imprimer un message.

Les commandes *uuencode* et *uudecode*

Ces deux commandes sont très utiles car elles permettent de coder un fichier binaire sous forme ascii. Elles sont très utilisées avec la messagerie électronique car cette dernière ne peut transmettre que des fichiers ascii.

```
$ uuencode /bin/ls newls > ls.uu
```

Le premier argument de la commande est le nom de fichier à coder, le deuxième est le nom qui sera donné au fichier lorsqu'il sera décoder par *uudecode*. Le résultat affiché sur la sortie standard est redirigé dans le fichier ll.uu.

```
$ uudecode ll.uu
```

L'argument de ***uudecode*** est le fichier codé. La commande va le décoder et créer le fichier **newls**. Parfois les fichiers binaires codés ascii sont trop volumineux pour être transmis via la messagerie électronique. Il existe dans ce cas deux solutions :

- compresser les fichiers avant la commande ***uuencode*****,**
- scinder sous forme de petits fichiers au moyen de la commande ***split*****.**

Les deux méthodes peuvent être utilisées conjointement.

Annexe 1 : Récapitulatif de commandes Unix

Documentation

man [options] [section] commande : accès au manuel de référence Unix

connexion au système

login [utilisateur] : connexion au système

passwd [utilisateur] : modification du mot de passe de connexion

su [utilisateur] : changement du nom d'utilisateur sans sortir du système

tty : affichage du numéro de terminal

stty : affichage et définition des différentes options d'un terminal

manipulation des fichiers et des répertoires

pwd: affichage du répertoire de travail

cd [répertoire] : changement de répertoire

mkdir répertoires: création de répertoires

ls [-options] [fichiers]: affichage du contenu d'un répertoire

cp [options] fic_sources fic_dest : copie physique de fichiers

mv [options] fic_sources fic_dest : changement de nom d'un fichier ou répertoire

ln [options] fic_sources fic_dest: création d'un lien avec un fichier

rmdir, rm [-options] fichiers:: suppression de répertoire ou de fichier

Protection des fichiers

chmod mode fichiers : changement des autorisations d'accès

chown : changement du propriétaire d'un fichier

chgrp : changement du groupe d'un fichier

umask [mode]: définition du mode de protection par défaut

Traitement de Fichiers

find répertoire expressions: recherche de fichiers à partir d'un répertoire.

grep [options] expression régulière [fichiers]: recherche d'une chaîne dans des fichiers

cut [options] fichiers : fractionnement vertical d'un fichier

sed fichiers : traitement de flots de caractères

tr chaine1 chaine2 : transformation de caractères

sort [options] [clé[option]...] fichiers : tri de fichiers

join [options] fichier1 fichier2 : jonction de 2 fichiers sur une zone commune

split [-n] fichier1 fichier2: fractionnement d'un fichier en plusieurs

wc [option] fichiers : compteur de mots

cat [options] fichiers : affichage du contenu d'un fichier

more fichiers : affichage du contenu d'un fichier page par page

cmp [options] fichier1 fichier2 : comparaison du contenu de 2 fichiers

diff [options] fichier1 fichier2: comparaison du contenu de 2 fichiers

dd [option=valeur]: copie de fichier avec conversion de format texte

uniq [options] fichiers : recherche de lignes identiques dans un fichier

comm [options] fichier1 fichier2: sélection/rejet de lignes communes à 2 fichiers rangés
head -n fichier : premières lignes d'un fichier
tail -n fichier : dernières lignes d'un fichier
file fichier : Délivre une description sur la nature du fichier passé en argument.
which fichier : Renvoie, s'il existe, le chemin absolu menant au fichier exécutable.
whereis fichier : Renvoie, s'ils existent, les chemins absolus menant au fichier.

gestion de l'impression

lpr [options] fichiers : imprimer un fichier.
lpq [options]: examiner la file d'attente d'impression.
lprm [options] numéro: retirer votre demande d'impression de la file d'attente.
pr [options] fichiers : mise en page de fichiers pour l'impression.

contrôle d'exécution des processus

ps [options] [pid]: indication sur l'état des processus actifs
kill [-signal] pid : envoi d'un signal à un processus
nice [-incrément] commande : exécution d'une comande avec une priorité faible
nohup commande : immunisation d'une commande contre les coupures de ligne et les arrêts
sleep durée: suspension de l'exécution pendant un intervalle donné
exec commande : exécute une commande à la place du shell
exit [code_retour]: termine le shell
wait [pid]: attend la fin des processus en tâche de fond

contrôle des ressources disque

du [-s] [répertoires] : occupation disque d'un répertoire
df [options] [répertoire/filesystem] : occupation disque des systèmes de fichiers
quota [-v] [utilisateur] : occupation disque d'un utilisateur

communication entre utilisateurs

write utilisateur [terminal] : utilitaire de communication interactif entre terminaux
wall [-a] [fichier] : (write all) envoie un message à tous les utilisateurs connectés
mesg [n] [y] : autorise ou interdit les messages
talk utilisateur [terminal] : établit une connexion interactive entre deux utilisateurs
who [option]: affichage de la liste des connexions au système
whoami : affichage du nom de l'utilisateur connecté sur le terminal

archivage, sauvegardes Compression de fichiers

tar [-][rxtuc] [vwfbFmkenpA] <archive|périphérique> <liste fichiers|répertoires> : archivage
compress, uncompress, gzip, gunzip fichiers : compression et décompression de fichiers

Annexe 2 : Bibliographie

A. Tanenbaum. *Les systèmes d'exploitation - conception et mise en oeuvre*. Informatique Intelligence Artificielle. InterEditions, Paris, 1989.

C. Pélissier. *Unix - utilisation, administration, système et réseau*. Traité des Nouvelles Technologies, Série Informatique. Hermès, Paris, 1996.

P. Charman. *Unix et X window - Guide pratique*. Cépaduès Editions, 1994.

Annexe 3 : Travaux Pratiques

49 Série 1 . Initiation

L'objectif de cette série est de se familiariser avec les commandes UNIX.

1. Tester les commandes **xman, man.**
Par ex : **man man**.

2. Lister le contenu d'un répertoire : entrer **ls** et déterminer les différentes options de cette commande (telles que -l, -a, -F, -g).
Par ex : **ls -l** liste le contenu d'un répertoire en format long, avec le mode de protection des liens, la dimension en octets et la date de la dernière modification.

3. Tester les caractères génériques : *****, **?**, **[ ]**, -.
Par ex : **ls *.txt** pour afficher la liste des fichiers d'extension **.txt**.

4. Tester les commandes sur les chemins : **pwd**, **cd <chemin d'accès>**

5. Créer un répertoire par la commande **mkdir <nom du répertoire>**

6. Copier un fichier avec la commande **cp <source> <destination>**

7. Tester la commande **mv <ancien fichier>** <**nouveau fichier>**.
Par ex : **mv rep1/tot rep2/.**
Développer les différentes options de ces commandes.

8. Effacer un fichier par **rm** et un répertoire par **rmdir**.

9. Tester la commande **cat [nom de fichier] [nom de fichier]**, développer les différentes options.

10. Tester la commande **more [nom de fichier], date**.

Note : Chaque commande est précédée d'un appel à **man** ou **xman**.

50 Série 2 . Traitement des fichiers

L'objectif de cette série est de mettre en oeuvre les techniques de recherche et de manipulation de fichiers sous le système d'exploitation UNIX.

1. Exercices sur find

a . Rechercher dans le répertoire **/etc**, les fichiers dont le nom :

. se termine par **p**

. contient un **a** ou un **b** minuscules

. se termine par une extension **.lock**

b . Rechercher dans le répertoire **/etc**, les fichiers dont seul le propriétaire possède toutes les permissions.

c . Rechercher dans votre répertoire des fichiers qui ont été sollicités durant les 2 derniers jours.

d . Créer dans votre répertoire des fichiers ayant le chiffre 5 dans leur nom (*Ex* : **fic5, fich5**,...). Rechercher les, puis effacer-les après confirmation.

2. Exercices divers

On utilisera le fichier de travail : **/etc/passwd**.

a. Compter le nombre de lignes contenues dans ce fichier.

b. Compter le nombre de lignes dont le 2ème caractère est un **a**.

c. Extraire les mots de passe, les afficher à l'écran.

d. Extraire les mots de passe contenant le chiffre 7.

e. Compter les mots de passe contenant le chiffre 7.

51 Série 3. Traitement des fichiers

L'objectif de cette série est de poursuivre la manipulation des fichiers sous le système d'exploitation UNIX.

1. Créez deux fichiers **fic1.txt** et **fic2.txt** sous **vi** qui diffèrent de 20 mots, en utilisant les commandes spécifiques de l'éditeur qui permettent les manipulations de texte. Puis successivement :

- triez-les (**sort**), en nommant les fichiers triés **fic1** et **fic2**.

- stockez dans un fichiers **10dif12** les 10 premières différences entres ces deux fichiers (utilisez les commandes **comm** et **split** ou **head**).

2. Vous séparerez ensuite la partie commune et les 2 parties différentes pour produire 3 fichiers.

fic1 **fic2**

fic12 **fic1-2** **fic2-1**

partie commune **différence de 1/2** **différence de 2/1**

3. Par redirection, recopier le sommaire de votre directory d'accueil dans un fichier **fic3**

4. Vous séparerez **fic3** en 2 fichiers

. **fic31** : **fic3** jusqu'au premier caractère .

. **fic32** : **fic3** à partir du premier caractère .

. **fic33** : **fic3** reconstitué à partir de FIC31 et FIC32.

5. Dans un répertoire et tous ses fils, rechercher tous les fichiers ayant l'extension **TXT** et recopier cette liste dans un fichier **fic4** dans le répertoire père (**find**).

6. Rechercher dans tous vos répertoires 4 fichiers qui ont été sollicités durant les 8 derniers jours.

Mettre la liste dans **fic5** (**find**)

7. Trier **fic3** sur le 2ème champ (délimité par un caractère <espace> ou <tabulation>).

Trier **fic3** et **fic4** dans **fic6** en ordre inverse sur les deux premiers caractères du champ1.

8. A partir de **fic3**, rechercher tous les fichiers créés le 4 février (ou une autre date).

9. Par redirection, recopier votre sommaire dans **fic7** et rechercher les lignes identiques de **fic3** et **fic7** (**uniq**).

10. Compter le nombre de mots, de lignes et de caractères contenus dans **fic7**.

11. Créer **fic8** : le fichier **fic7** dans lequel les majuscules sont remplacées par des minuscules (**tr**).

Créer **fic9** : le fichier **fic7** dans lequel les majuscules et minuscules sont permutées.

52 Série 4 . Structures de contrôle, variables

L'objectif de cette série est de s'essayer à la programmation en Bourne shell et d'utiliser les structures de contrôle, les variables...

1. Variables et commande test

Tapez au clavier les lignes suivantes :

ex 1 : commande **test** et variable **$?**.

```
pwd
touch zzz
ls -l zzz
echo $?
test -f zzz
echo $?
ls -l proc1
test -f proc1
écho $?
```

ex 2 : substitution et variables

```
var="ls -l"
echo $var
echo "$var"
echo '$var'
echo "\$var"
echo `$var`
echo $HOME
echo $PATH
```

" : double quote, ' : single quote, ` : back quote

Commentez.

2. Alternative **if... then... else... fi**

Compléter, exécuter et commenter :

ex 1 : à créer dans un fichier **nom** puis à exécuter avec : **nom arg1**

```
if test -f $1
then echo $1 existe
else echo $1 n\'existe pas
fi
```

ex 2 :

```
if test -f $1 -o -d $1
then echo le fichier ou le répertoire $1 existe
else echo le fichier ou le répertoire $1 n'existe pas
fi
```

ex 3 :

```
if test -f $1; then
    echo "le fichier $1 existe"
    if test ! -s $1; then
        rm $1
```

```
        else
            echo (insérer message correspondant)
        fi
    else
        if test -d $1; then
            echo (insérer message correspondant)
            # ls $1 Áwc -w > n.fic
            a = `ls $1 Á wc -w`
        if test $a -eq 0; then
            echo(insérer message correspondant)
            rmdir $1
        else
            echo(insérer message correspondant)
        fi
    else
        echo(insérer message correspondant)
    fi
fi
```

3. Structure répétitive **for**... **in**... **do**... **done**

Remplacer les messages et commenter le fonctionnement :

```
    case $# in
        2)  if test -d $1; then
                for i in .o .c .p; do
                    if test -f $1/$2$i; then
                        echo $1/$2$i (insérer message)
                    else
                        echo $1/$2$i(insérer message)
                    fi
                done
            else
                echo $1(insérer message)
            fi;;
        *)  echo(message d'erreur) ;;
    esac
```

4. Structure répétitive **while**... **do**... **done**

```
    if test $# -ne 2; then
        echo (message d'erreur)
    else
        fic = /tmp/f$$ (n° de processus en cours)
        echo -n $1 > $fic
        taille = `cat $fic Á wc-c`
        while [ $taille -lt $2 ]; do
            echo -n $1 >> $fic
            taille `cat $fic Á wc-c`
        done
        ls -l $fic
        rm $fic
    fi
```

53 Série 5 . Synthèse

L'objectif de cette série est de réaliser une synthèse des séries précédentes.

1. Procédure qui vérifie si l'argument est un fichier ou un répertoire.
appel : **repchk <fichier>.**

2. Procédure de commande qui compte de 1 à 10.

3. Faire éditer un dicton pour chaque jour de la semaine. Le jour est entré en paramètre.

4. Procédure testant si le paramètre en entrée est un fichier. S'il est un répertoire, on s'y déplace, on le liste, on revient au répertoire père. Autrement, on édite un message d'erreur.

5. Procédure interactive lisant 2 noms de fichiers et comparant leur nombre de lignes.

6. Procédure acceptant en entrée le nom d'un utilisateur et vérifiant sa présence.

7. Procédure affichant en sortie le diagramme du sous-arbre courant représenté par la variable d'environnement **$HOME**, représentant le nom du répertoire de l'utilisateur.
Si l'on est dans **/usr** :

arbre bin
ou : **arbre../bin** affiche le sous-arbre bin

arbre/usr/bin : débute par **usr** de **root** et continue par **bin**

arbre : débute le répertoire courant.

Annexe 4 : Corrigés des Travaux Pratiques

54 Série 2 . Traitement des fichiers

1. Exercices sur find

a . **find /etc -name "*p" -print**
find /etc -name "*[ab]* " -print
find /etc -name "*.lock" -print

b . **find /etc -perm 700 -print**

c . **find ~ -atime 2 -print** ou encore **find $HOME -atime 2 -print**

d . **find ~ -name "*5*" -ok rm {} \;**

2. Exercices divers

a . **wc -l passwd**

b . **grep " ^.a " passwd | wc -l**

c. **cut -d: -f2 passwd**

d . **cut -d: -f2 passwd | grep 7**

e . **cut -d: -f2 passwd | grep 7 | wc -l**

f . **sed -n -e '/pguenel,/x168/p' passwd | wc -l**

g . **sed -n -e '1,10s/:/,/' -e 1,10p passwd**

55 Série 3. Traitement des fichiers

1. On crée 2 fichiers texte sous **vi**, **fic1.txt** et **fic2.txt.**

On trie les deux fichiers par ordre alphabétique. On utilise la commande **sort**, soit :

sort fic1.txt > fic1 (ou **sort -o fic1 fic1.txt**)

sort fic2.txt > fic2 (ou **sort -o fic2 fic2.txt**)

On utilise la commande **comm fic1 fic2** pour éditer les différences qui existent entre les deux fichiers. Cette commande place :

. Dans une première colonne les lignes de fichier1 qui ne sont pas dans fichier2.

. Dans une seconde les lignes de fichier2 qui ne sont pas dans fichier1,

. Dans une troisième les lignes qui sont dans les deux fichiers

Pour supprimer une colonne, il suffit de donner son numéro en paramètre. Pour avoir seulement les différences on tapera donc la commande :

comm -3 fic1 fic2 > dif12

On ne veut éditer que les 10 différences existant entre les deux fichiers. Il faut donc utiliser la commande **split**. Cette commande permet de découper le fichier spécifié en un certain nombre de sections de n lignes.

On tape donc la commande : **split -10 dif12 > 10dif12**

On peut aussi faire appel à la commande **head** : **head -10 dif12 > 10dif12**

L'ensemble de ces opérations peut être groupé un une seule commande :

comm -3 fic1 fic2 | head -10 > 10dif12

2. On veut maintenant éditer trois fichiers :

fic1-2, où l'on aura les lignes du seul **fic1**

fic12, où l'on aura les lignes communes aux deux fichiers

fic2-1, où l'on aura les lignes du seul **fic2**

On tape donc les commandes :

comm -13 fic1 fic2 > fic2-1

comm -12 fic1 fic2 > fic12

comm -23 fic1 fic2 > fic1-2

3. Par redirection, on copie le contenu de notre répertoire courant dans le fichier **fic3**. Soit la commande : **ls -al > fic3**

4. On veut séparer **fic3** en deux fichiers :

fic31 qui est la première zone, délimitée par le premier "**.**"

fic32 qui est la deuxième zone après le premier "**.**"

On utilise alors la commande **cut** de la façon suivante :

cut -f1 -d.fic3 > fic31

cut -f2 -d.fic3 > fic32

Avec la commande **paste**, on peut reconstituer le fichier **fic3** :

paste -d"." fic31 fic32 > fic33

Le "**:**" permettant de relier les deux zones à coller.

5. On doit rechercher tous les fichiers portant l'extension **.txt** depuis la racine de notre compte et dans tous ses fils.

La commande à taper est donc :

find ../. -name '*.txt' -print > fic4

6. Nous allons ici rechercher quatre fichiers portant l'extension **.txt** ,depuis le répertoire courant et dans tous ses fils, et qui ont été sollicités durant les 8 derniers jours. La commande à taper est donc :

find . -name '*.txt' -atime -8 -print | split -4 > fic5

7. On veut trier le fichier **fic 3** sur le deuxième champ après **<espace>** ou <**tab**> :

sort + 1 -2 -o fic3a fic3

Tri sur les deux premiers caractères en ordre inverse :

sort + 1.1 -1.3 -o fic6 fic3 fic4

8. Recherchons les fichiers crées le 4 février (par exemple)

On utilise la commande **grep** qui permet de rechercher une chaîne de caractères dans un fichier. Etant donné que **fic3** contient la liste des fichiers avec leur date de création, on utilise ce fichier pour cette recherche. On met le résultat dans le fichier **fic4fev**. Soit :

grep '4fev' fic3 > fic4fev

9. On recopie le sommaire dans **fic7** grâce à la commande : **ls -al > fic7**

On concatène les fichiers **fic3** et **fic7** puis on les trie pour que les lignes identiques consécutives puissent être extraites par la commande **uniq** qui enlève les redondances d'un fichier :

cat fic3 fic7 | sort | uniq -d

10. La commande **wc** permet de compter mots, lignes et caractères. Si on veut compter dans **fic7** :

le nombre de mots, on tape: **wc -w fic7;**

le nombre de lignes, on tape: **wc -l fic7;**

le nombre de caractères, on tape: **wc -c fic7.**

11. Pour transformer les majuscules en minuscules dans le fichier **fic7**, on utilise la commande **tr** :

tr [A-Z] [a-z] <fic7 >fic8

Pour permutter majuscules et minuscules :

tr [a-zA-Z] [A-Za-z] <fic7 >fic9

56 Série 4 . Structures de contrôle, variables

1. Variables et commande test

Ex 1 :

On crée le fichier **zzz** avant de taper la séquence proposée. Par contre, le fichier **proc1** n'existe pas.

pwd : affiche le chemin d'accès au répertoire courant.

touch zzz : crée un fichier vide de nom **zzz.**

ls -l zzz : affiche les informations concernant le fichier **zzz**.

echo $? : renvoie un code fonction du résultat de la dernière commande exécutée, en l'occurence **0** puisqu'elle n'a pas provoqué d'erreur.

test -f zzz : teste si le fichier **zzz** est présent dans le répertoire courant mais ne renvoie aucun message.

echo $? : renvoie **0** : le test est valide, le fichier existe dans le répertoire courant.

ls -l proc1 : affiche à l'écran un message indiquant que le fichier n'existe pas.

test -f proc1 : teste si le fichier **proc1** est présent dans le répertoire courant, mais ne renvoie aucun message.

Ex2 :Le mécanisme de substitution

var="ls -l" : affecte la chaîne de caractères **ls -l** à la variable nommée **var**.

echo $var : renvoie la chaîne **ls -l**, la variable $**var** est substituée par sa valeur.

echo "je suis $var" : idem, la variable $var est substituée dans la chaine encadrée par **"** .

echo 'je suis $var' : renvoie **je suis $var**, pas de susbtitution entre deux **'** .

echo "\$var=$var"** : renvoie **$var=ls -l, **\$** est le caractère $ et non l'opérateur $ de substitution.

echo `$var` : renvoie le résultat de la commande **ls -l**, la chaîne encadrée de **`** est exécutée.

echo $HOME : affiche le chemeni ansolu de votre répertoire de connection.

echo $PATH : affiche la liste des chemein de recherche des exécutables.

2. Alternative **if**... **then**... **else**... **fi**

Ex 1 :
```
if test -f $1; then
    echo "$1 existe"
else
    echo "$1 n'existe pas"
fi
```

On sauvegarde ce fichier sous le nom **td321,** puis on autorise son exécution par :

chmod 700 td321 ou encore : **chmod u+x td321**

Cette procédure teste la présence du fichier passé en paramètre (**$1**) et affiche son nom en précisant s'il existe ou non. On peut donc tester ce fichier de commande en reprenant les exemples du premier exercice, soit :

La commande **td321 zzz** renvoie le message : **zzz n'existe pas**

La commande **td321 proc1** renvoie le message : **proc1 n'existe pas**

Ex2 :
```
if test -f $1 -o -d $1; then
    echo "le fichier ou le repertoire $1 existe"
else
    echo "le fichier ou le rep $1 n'existe pas"
fi
```

On sauvegarde ce fichier sous le nom **td322** puis on autorise son exécution.

Cette procédure teste la présence comme fichier ou répertoire du nom passé en paramètre (**$1**) et affiche son nom en précisant s'il existe ou non comme fichier ou répertoire sans faire la distinction. Pour tester cette procédure, on passe en paramètres un répertoire existant et un n'existant pas, on crée alors le sous répertoire **rep** :

td322 rep renvoie le message : **le fichier ou le répertoire rep existe**

td322 proc1 renvoie le message : **le fichier ou le répertoire proc1 n'existe pas**

Ex3 :
```
if test -f $1; then
    echo "le fichier $1 existe"
    if test ! -s $1; then
        rm $1
    else
        echo "le fichier n'est pas vide"
    fi
else
    if test -d $1; then
        echo "le repertoire $1 existe"
        # ls $1 | wc -w > n.fic
        a= `ls $1 | wc -w`
        if test $a -eq 0; then
            echo "le rep $1 est vide (supprimé)"
            rmdir $1
        else
            echo "le repertoire $1 est non vide"
```

```
        fi
    fi
else
    echo "le fichier ou le repertoire $1 n'existe pas"
fi
```

On sauvegarde ce fichier sous le nom **td323**, puis on autorise son exécution. Ce fichier :

. Recherche le nom de fichier **$1** (ligne 1) dans le répertoire courant et l'efface s'il est vide (ligne 4).

. S'il n'existe pas comme fichier, la procédure recherche **$1** comme sous répertoire courant (ligne 8).

. Si ce sous répertoire existe, elle met dans **a** le nombre de fichiers qu'il contient (ligne 11).

. Si le répertoire est vide, il est effacé (ligne 14), sinon on affiche un message **"repertoire non vide"**

. Si **$1** n'existe ni comme fichier, ni comme répertoire, on affiche le message suivant : **"$1 n'existe pas"**

3. Structure répétitive **for**... **in**... **do**...**done**...

```
case $# in
2)  if test -d $1; then
        for i in .o .c .p; do
            if test -f $1/$2$i; then
                echo "$1/$2$i existe"
            else
                echo "$1/$2$i n'existe pas"
            fi
        done
    else
        echo "le repertoire $1 n'existe pas"
    fi;;
*)  echo "il faut deux arguments";;
esac
```

On sauvegarde ce fichier sous le nom **td33** puis on autorise son exécution. Ce fichier réalise :

. Si la procédure est exécutée suivie d'un nombre d'arguments différent de 2, elle renvoie le message : "**il faut deux arguments**" (ligne 11) et rend la main. Le choix est fait par le **case $#**.

. Si le nombre d'arguments est 2, la procédure teste si le premier (**$1**) est un répertoire (ligne 2), sinon renvoie le message : **"le répertoire $1 n'existe pas**".

. Si le répertoire existe, alors la boucle en ligne 3 permet de tester l'existence du fichier nommé **$2** suivit de l'extention **$i** où la variable **i** prend les valeurs : **".o" ".c" ".p"** et affiche les messages d'existence ou de non existence (lignes 5 et 6).

4. Structure répétitive **while**... **do**...**done**...

```
if test $# -ne 2 ; then
    echo "il faut deux arguments"
else
    fic=/tmp/f$$
    echo -n $1 > $fic
    taille=`cat $fic | wc -c`
    while [ $taille -lt $2 ]; do
        echo -n $1 >> $fic
        taille=`cat $fic Á wc -c`
```

```
    done
    ls -l $fic
    cat $fic
    rm $fic
fi
```

On sauvegarde ce fichier sous le nom **td34** puis on autorise son exécution. Ce fichier réalise :

. Le test en ligne 1 vérifie que le nombre d'arguments de la ligne de commande est différent de 2, dans ce cas le message en ligne 2 apparaît.

. Sinon, la variable **fic** prend la valeur **/tmp/f$$**, où **$$** fournit le numéro du processus en cours (ligne 3).

. En ligne 4, on recopie le premier argument dans le fichier déclaré par **fic**, sans retour chariot.

. La variable **taille** prend la taille du fichier **fic** en nombre de caractères.

. La boucle **while** va ajouter le premier argument **$1** dans le fichier **fic** jusqu'à ce que sa taille soit supérieure à la valeur spécifiée dans **$2**, la taille étant remise à jour en ligne 9.

. La syntaxe **[$taille -lt $2];** est équivalente à : **test $taille -lt $2**

. Ensuite, la commande **ls -l $fic** affiche les informations complètes sur le fichier **$fic**, **cat $fic** affiche le contenu du fichier **$fic**, **rm $fic** l'efface..

57 Série 5 . Synthèse

1.

```
# procedure verifie si l'argument est un fichier ou un
# repertoire
# appel : repchk <fichier>

if test -f "$1"; then
    cat $1
elif test -d "$1" ; then
    (cd $1 ; ls -l; cd..)
else
    echo $1 "ni fichier ni répertoire"
fi
```

2.

```
cnt=1
until [$cnt -gt 10]; do
    echo $cnt
    cnt = `expr $cnt +1`
done
```

3.

```
# appel dicton <jour>
case $1 in
lundi)              echo "C'est ravioli";;
mardi)              echo "Bon courage";;
mercredi)           echo "Entre deux chaises";;
 jeudi)             echo "Rien de neuf";;
vendredi)           echo "Ouf ! C'est $1 ";;
```

```
    samedi | dimanche)  echo "Repos";;
    *)                  echo"Jour inconnu"
    esac
```

4.

```
    # procedure : fich_check
    # appel : fichk <nom>
    # attention (if .. fi) état du shell

    if test -f "$1"; then
        exit 2;
    elif test -d "$1"; then
        exit 3;
    else
        exit 4
    fi

    case $? in
    2)  cat $1; echo "fichier $1";;
    3)  cd $1; ls -l;cd ..;;
    4)  echo "ni fichier, ni repertoire"
    esac
```

5.

```
# procedure compare nombre de lignes de deux fichiers
 # appel : licomp <fich1> <fich2>

     if test `cat "$1"  | wc -l` -gt ` cat "$2" | wc -l` ; then
          echo $1 contient plus de lignes que $2
    else
        echo $2 contient plus de lignes que $1
    fi
```

6.

```
    # estula - vérifie la présence d'un utilisateur

    if [ "$#" -ne 1 ]; then
        echo "\n\restula : nombre d'arguments incorrect"
        echo "\rSyntaxe : estula <Utilisateur>\r"
    else
        user="$1"
        if who | grep "^$user " > /dev/null; then
            echo "$user est connecté"
        else
            echo "$user n'est pas connecté"
        fi
    fi
```

7.

```
# @(#) Arbre - Diagramme des répertoires et fichiers
```

```
    if [ "$#" -gt 1 ]; then
        echo "arbre   : nombre d'arguments incorrect"      >&2
        echo "Syntaxe : arbre [dir]"                       >&2
        exit 2
    fi
    if [ "$#" -eq 1 ]; then
        if [ ! -d $1 ]; then
            echo "$0: $1 pas de répertoire !" >&2
            echo "Syntaxe : arbre [dir]"      >&2
            exit 2
        fi
    fi

find ${1:-.} -print | sort | sed -e "1p" -e "1d" -e "s|[^/]*/|    /|g" -e "s|^ */|/|" -e "s|/\([^/]*\)$|\1|"
```