Introduction au système d'exploitation UNIX

Thomas LEDUC

Thomas.Leduc@masi.ibp.fr

http://quartz.dgs.jussieu.fr:8080/

Mars 1997

Contents

1 Présentation globale **4**
- 1.1 Définition . . . 4
- 1.2 Historique . . . 4
- 1.3 Vue générale du système . . . 5
- 1.4 Les grandes caractéristiques . . . 5
- 1.5 Quelques remarques préliminaires . . . 5

2 L'UNIX de base - premiers pas **6**
- 2.1 Procédure de connexion . . . 6
- 2.2 Fonctionnement d'un shell interactif . . . 6
- 2.3 Commandes élémentaires . . . 7
  - 2.3.1 Modifier le mot de passe . . . 7
  - 2.3.2 Date . . . 7
  - 2.3.3 Se convaincre du caractère multi-utilisateurs . . . 7
  - 2.3.4 Communiquer . . . 7
  - 2.3.5 Se protéger . . . 8
  - 2.3.6 Réaction du système face à une inconnue . . . 9
  - 2.3.7 En vrac . . . 9
- 2.4 L'aide en ligne . . . 9
- 2.5 Procédures de déconnexion . . . 10

3 Le système de gestion de fichiers **10**
- 3.1 Définition . . . 10
- 3.2 Navigation dans l'arborescence des fichiers . . . 10
- 3.3 La hiérarchie . . . 10
- 3.4 Les droits d'accès aux fichiers . . . 11
- 3.5 Modification des droits d'accès . . . 13
- 3.6 Les fichiers spéciaux . . . 13
- 3.7 Quelques fichiers particuliers de configuration . . . 14
- 3.8 Commandes de manipulation de fichiers . . . 15
  - 3.8.1 Manipulation de répertoires exclusivement . . . 16
  - 3.8.2 Travaux pratiques . . . 16

4 Les processus **16**
- 4.1 Généralités . . . 16
- 4.2 Gestion élémentaire des processus . . . 17
  - 4.2.1 La commande ps . . . 17
  - 4.2.2 Arrêter un processus . . . 17
- 4.3 Fichiers standards et redirection . . . 17
- 4.4 Enchaînement de processus . . . 19
- 4.5 Lancement en arrière plan . . . 19
- 4.6 Autres commandes . . . 19

5 Les principales commandes externes **20**

6 L'interface de commande - les shells **24**
6.1 Introduction 24
6.1.1 Sélection de l'interprèteur à l'appel du script 24
6.1.2 Choix de l'interprèteur 25
6.2 Généralités sur les langages de commande - les mécanismes 25
6.2.1 Les métacaractères 26
6.2.2 Le mécanisme de substitution 26
6.2.3 Le mécanisme d'expansion 27
6.2.4 Les délimiteurs de chaînes 27
6.2.5 Processus et contrôle des tâches 27
6.3 Un petit exemple illustrant les similitudes et différences entre le C -langage de haut niveau, le C-shell et le Korn-shell 28
6.4 Petit memento csh 29
6.5 Petit memento ksh 33

7 L'édition de texte **36**
7.1 Notion d'expression régulière 36
7.2 L'éditeur ed 37
7.3 Le filtre d'édition sed 37
7.4 L'éditeur vi 37

8 Les commandes orientées réseau **38**

9 Le langage awk **41**

10 Le langage Lex - construction d'analyseurs lexicaux **42**
10.1 Présentation 42
10.2 Premier exemple 42
10.3 Ré-écrivons la commande "wc" 43

1 Présentation globale

1.1 Définition

Unix est un système d'exploitation multi-tâches, multi-utilisateurs, avec mémoire partagée et mémoire virtuelle (le **swap**[1]) ainsi que de nombreuses facilités comme les services réseaux. On dit, au sens strict, que c'est un noyau d'OS en temps partagé, c'est-à-dire un programme gérant les ressources d'un système informatique. Il permet donc tout à la fois d'exécuter des programmes, de gérer des périphériques et de fournir un système de gestion de fichiers pour un stockage de longue durée, le tout de manière transparente pour l'utilisateur.
L'idée de créer un système multi-tâche est venu d'une simple question de rentabilité du processeur. L'inactivité du processeur coûte en effet chère sur les gros systèmes. Pour se convaincre de la nécessité du multi-tâche, considérons un simple problème de traitement de texte (comme par exemple l'écriture de ce support) : l'échelle de temps de la CPU est la nanoseconde, celle d'un "périphérique lent" tel qu'un utilisateur est la seconde... Une station de travail dont l'activité du processeur servirait uniquement à gérer une entrée-sortie (frappe du clavier) par seconde perdrait "infiniment" son temps. Il faut donc libérer le processeur dès qu'un traitement requiert une opération d'entrée-sortie.
Comme il n'y a par ailleurs pas de raison théorique pour que les tâches en cours appartiennent au même utilisateur, les programmes sont dits ré-entrants. L'exécutable n'est donc chargé qu'une fois et porte sur des données propre à chaque utilisateur appelant. Un même traitement de texte peut donc être utilisé par plusieurs utilisateurs de la même machine en donnant à chacun l'impression d'être "seul au monde" avec sa machine qui lui est entièrement dédiée.
Quand la demande est trop forte, il n'y a pas de blocage mais juste un ralentissement de l'activité... en théorie du moins !

Classification sur le partage de temps

Mono-tâche Le système n'effectue qu'une tâche à la fois (cas du MSDos ou du MacOs), il est parfois possible de permuter entre plusieurs tâches sans avoir à relire le programme sur le disque.

Multi-tâches Le système exécute plusieurs tâches "simultanément" (ou quasi simultanément par *time-sharing* si la machine est mono-processeur). Le *time-sharing* est une technique de partage du temps qui permet à chaque tâche de s'exécuter pendant un petit laps de temps ("Slice time"). On distingue les systèmes multi-tâches **préemptifs** des systèmes multi-tâches **non préemptifs**.
Dans un système multi-tâches **préemptif**, il existe un *scheduler* (programme spécial) qui alloue du temps à chaque tâche et les interrompt lorsque le temps qui leur est imparti est écoulé. Il y a une file d'attente des tâches avec une notion de priorité (cas d'UNIX ou de Windows 95).
Dans un système multi-tâches **non préemptif**, le scheduler n'existe pas et les programmes doivent-être conçus pour "rendre la main" de temps à autre (cas de Windows 3.*)

1.2 Historique

UNIX est né aux Bell Laboratories en 1969, on le doit à Ken Thompson. Ecrit d'abord en langage machine puis en assembleur avec John Kernighan, il a profité du développement du C (Dennis Ritchie) pour être réécrit dans un langage de plus haut niveau ⇒ plus grande portabilité ⇒ diffusion plus rapide.

[1] Attention : le swap est beaucoup plus lent que l'accés à la mémoire **RAM** !

Une V7 est commercialisée en 1978. Deux grandes tendances voient alors le jour : en milieu universitaire l'Université de Berkeley sort les versions UNIX BSD (Berkeley Software Distribution), en milieu industriel AT&T et les Bell Laboratories sortent les versions UNIX SYSTEM III et V.
le noyau UNIX d'AT&T ⇒ XENIX (Microsoft), UNIX SCO, VENIX...
MINIX, XINU, GNU (Free Software Fondation à objectifs de totale compatibilité et de gratuité) sont basés sur des noyaux UNIX non AT&T.
Multiplications de versions ⇒ tentatives de regroupement et de normalisation (définitions d'interfaces standard : XOPEN, POSIX).
IHM de base "frustre" ⇒ intégration dans l'environnement UNIX du système X-Window (projet Athena du MIT) qui permet de travailler en mode graphique ⇒ ce standard a permis de développer des bibliothèques telles que MOTIF, OPEN LOOK... puis des interfaces type Macintosh comme NeXTStep.

1.3 Vue générale du système

Le système UNIX est composé de :

- un noyau pour gérer la mémoire, les ES de bas niveau et l'enchaînement des tâches,
- d'interpréteurs de langage de commande (les shells), de commandes de manipulation de fichiers, de commande de gestion des processus, de commande de communication entre utilisateurs et systèmes,
- d'outils d'édition et manipulation de textes, d'outils généraux de développement d'application (compilateur, édition de liens, débogueurs, gestion de versions, analyseurs lexicaux et syntaxiques...)

1.4 Les grandes caractéristiques

- une hiérarchie (doublée d'une "génétique") des processus avec héritage d'un ensemble de caractéristiques d'un processus père à son processus fils,
- des appels système possible depuis les langages de haut niveau tel que le C,
- un aspect multi-tâches avec possibilité de lancer des processus en arrière-plan et de faire du multi-fenêtrage (⇒ simuler le parallèlisme),
- des shells permettant d'écrire des scripts et d'enrichir ainsi le noyau de nouvelles commandes (un shell permet d'analyser lexicalement, syntaxiquement et sémantiquement toute ligne de commande placée sur l'entrée standard puis de l'exécuter en faisant appel aux fonctions du noyau),
- un système de fichiers hiérarchisé,
- un mécanisme de redirection des entrée-sorties standard permettant de faire du filtrage (concept fondamental)...

1.5 Quelques remarques préliminaires

UNIX est case-sensitive
interruption "sauvage" par : CTRL-\, CTRL-|, CTRL-C ou CTRL-Z (suivi d'un kill %%)
le jeu de caractères usuel est l'ASCII - caractères accentués ISO-Latin 1
après un CTRL-S (interruption du flot vers la sortie standard), faire un CTRL-Q

interruption du flot d'entrée par un CTRL-D
système multi-utilisateur ⇒ notion de sécurité (un seul utilisateur par login, mot de passe personnel régulièrement modifié, droits d'accès...)

2 L'UNIX de base - premiers pas

2.1 Procédure de connexion

```
login: leduc
Password:
Login incorrect
login: leduc
Password:
Last login: Sun Jan  5 11:22:21 from cestac
SunOS Release 4.1.3 (GENERIC) #3: Mon Jul 27 16:43:54 PDT 1992
You have new mail.
cestac{leduc}1:
```

Recommandations : votre compte est personnel, ne laissez jamais une session en cours sans la "locker" si vous êtes en environnement graphique (xlock sous X-11), ne diffusez pas ni n'écrivez votre mot de passe...
Un mot de passe doit faire entre 6 et 8 caractères, utiliser les majuscules, les minuscules, les chiffres et tant que faire se peut d'autres symboles tels que : le slash, backslash, l'underscore, l'espace, la ponctuation... Il ne doit figurer dans aucun dictionnaire, ne doit pas être obtenu par combinaison d'éléments de votre état civil, de votre adresse... Il doit-être renouvelé fréquemment mais sans être oublié ! Attention à ne pas le diffuser sur l'Internet, aux programmes simulant des connexions erronées, aux BBS (où il serait visible pour l'opérateur)... Il faut un mot de passe pour chacune des machines ou vous avez un compte !
Remarque : à chaque connexion, il y a affichage possible de plusieurs messages tels que : le contenu de /etc/motd ("Message Of The Day") le fichier de bienvenu, la date, la présence ou non de nouveaux mails... avant l'écriture de l'invite (ou prompt) du shell.

2.2 Fonctionnement d'un shell interactif

Il s'agit par exemple du *login-shell*, le processus interactif qui est lancé lors de la procédure de connexion d'un utilisateur. A l'apparition de l'invite le *prompt*) l'utilisateur rentre le texte de sa commande suivie d'une fin de ligne ("linefeed", CTRL-J) ou d'un retour charriot ("return", CTRL-M). L'interpréteur analyse alors la commande et l'évalue. Le tout s'opère selon l'algorithme ci-dessous :

```
Tant que VRAI faire
        si les commandes ne sont pas lues à partir d'un fichier[2]
                alors afficher le prompt
        Lire la commande[3]
        si c'est une commande interne
                alors l'exécuter
        sinon
                chercher un exécutable qui porte ce nom[4]
                si cet exécutable n'existe pas
```

[2] C'est-à-dire si l'entrée standard n'a pas été redirigée
[3] Chaîne de caractères se terminant par un retour charriot.
[4] Dans le répertoire courant, les répertoires du PATH...

```
                alors afficher ``Command not found.''
            sinon l'exécuter (par création d'un nouveau processus)
```

Pour faciliter la frappe des caractères de la ligne de commande, le terminal est paramétrable au moyen de la commande **stty**. Pour connaître les valeurs des *caractères de contrôle*[5] EOF, ERASE, INTR, QUIT, WERASE... au cours de la session, taper **stty -a**. Exemple : `stty erase ^H`

2.3 Commandes élémentaires

2.3.1 Modifier le mot de passe

```
$ passwd
Changing password for leduc on quartz.
Old password:
New password:
Retype new password:
$
```

ou, mieux encore, yppasswd...

2.3.2 Date

```
$ date
Sun Jan  5 17:14:44 WET 1997
$
```

2.3.3 Se convaincre du caractère multi-utilisateurs

```
$ who
leduc    console Jan  5 11:28
leduc    ttyp0   Jan  5 11:29
leduc    ttyp1   Jan  5 11:29
leduc    ttyp4   Jan  5 12:23   (:0.0)
$
$ w
  5:48pm  up 34 days,  7:23,  1 user,  load average: 1.00, 1.00, 1.00
User     tty       login@  idle   JCPU   PCPU  what
leduc    ttyp1    12:25pm            1         w
montagna ttyp4    Thu 2pm 3days                -
lamotte  ttyp7    18Dec9618days                -
lamotte  ttyp8    18Dec9618days                -
lamotte  ttyp9    18Dec9618days                -
$
```

2.3.4 Communiquer

- A soi-même ;-)

  ```
  $ echo bonjour Thomas
  bonjour Thomas
  $
  ```

[5] On aborde ici le cadre de la gestion des terminaux... Exemple : stty erase '^ h' intr '^ c'

- A un autre utilisateur (connecté) de la même machine (en temps réel)

  ```
  $ write leduc
  write: leduc logged in more than once ... writing to console
  Bonjour Thomas
  $
  ```

  Permet de lire sur la console (en environnement multi-fenêtré) ce qui suit :

  ```
  Message from leduc@quartz on ttyp1 at 17:19 ...
  Bonjour Thomas
  EOF
  ```

  Note : cette commande a plus d'intérêt si l'on écrit à quelqu'un d'autre...

  Un $*echo Bonjour Thomas > /dev/console* aurait eu le même effet !

- A un utilisateur distant connecté (en temps réel)

  ```
  quartz{leduc}1: talk leduc@achille.ibp.fr
  ```

  Permet de lire sur la machine achille ce qui suit :

  ```
  Message from Talk_Daemon@achille at 17:27 ...
  talk: connection requested by leduc@quartz.dgs.jussieu.fr.
  talk: respond with:  talk leduc@quartz.dgs.jussieu.fr
  ```

- A un utilisateur distant ou non, non connecté

  ```
  $ mail leduc@masi.ibp.fr
  Subject: Bonjour Thomas
  Voila c'est tout
  .
  EOT
  $
  ```

  Note : cette commande a plus d'intérêt si l'on écrit à quelqu'un d'autre...

2.3.5 Se protéger

Pour vous mettre aux abonnés absent (ie vous protéger d'un write ou talk intempestif) :

```
$
$ mesg
y
$ mesg n
$ mesg
is n
$ write leduc
write: You have write permission turned off
$
```

Une tentative de talk depuis une machine distante permet de lire ce qui suit :

```
quartz{leduc}1: talk leduc@achille.ibp.fr
[Ringing your party again]
[Ringing your party again]
[Your party is refusing messages]
quartz{leduc}2:
```

2.3.6 Réaction du système face à une inconnue

```
$ maCommandeAmoi
maCommandeAmoi: not found
$
```

2.3.7 En vrac

clear efface l'écran
whoami permet de retrouver son nom de login

2.4 L'aide en ligne

Elle est obtenu par la commande *man* : "man displays information from the reference manuals." Cette documentation retourne des renseignements sur la commande passée en argument en les regroupant dans des rubriques telles que :
NAME donne le nom de la commande ainsi qu'une brève description
SYNOPSIS résume les conditions d'utilisation de la commande (options, arguments...)
DESCRIPTION décrit les effets de la commande et de ses options
Mais il existe aussi d'autres rubriques possibles : **AVAILABILITY**, **OPTIONS** (décrit chacune des options), **FILES** (décrit les fichiers utilisés par la commande), **SEE ALSO** (consulter des commandes voisines), **BUGS** (les bogues connus)...
Le manuel, souvent placé dans le répertoire /usr/man (voir MANPATH), comporte 8 chapitres :

- 1. les commandes accessibles à l'utilisateur
- 2. l'interface UNIX - langage C
- 3. la bibliothèque C (stdio, math...)
- 4. Les carctéristiques des fichiers systèmes associés aux périphériques
- 5. Les formats des fichiers systèmes
- 6. Les jeux, les démonstrations
- 7. La bibliothèque de macro pour la manipulation de documents
- 8. Les commandes de maintenance du système

Pour plus d'informations, taper man man sous un shell UNIX.
Exemple : **man 2 exit** renvoie de la doc sur void _exit(int) à ne pas confondre avec **man exit** qui renvoie de la doc sur la commande exit du shell (man 1).

2.5 Procédures de déconnexion

Différentes selon les shells : une fin de fichier à partir du prompt CTRL-D (eof) ou bien exit ou logout (si le shell employé est de la famille du C-shell et paramétré de telle sorte que le CTRL-D soit inefficace pour sortir du shell). Exemple :

```
iris{leduc}1: ^D
Use "logout" to logout.
iris{leduc}1: logout
Connection closed.
login:
```

3 Le système de gestion de fichiers

3.1 Définition

Dans UNIX, tout (ou presque) est fichier : données, commandes, répertoires, périphériques (imprimante, disques physiques ou logiques[6], terminaux, écran...). Un fichier est une chaîne de caractères non structurée à laquelle est associée un bloc d'informations (*i-noeud* ou *index-node*) stocké dans une table.
Le i-noeud est une structure comprenant : un identifiant de propriétaire et de groupe, le type du fichier et les droits d'accès, la taille du fichier (en nombre de caractères), le nombre de liens physiques sur le fichier, 3 dates significatives (dernières lecture, modification du fichier et modification du noeud), l'adresse des blocs sur le disque (pour les fichiers physiques uniquement) et l'identification de la ressource associée pour les fichiers spéciaux. Le nom du fichier n'est pas inclus dans le i-noeud mais dans un répertoire qui est lui-même un fichier contenant des couples (index du fichier, nom du fichier).
Un répertoire, en tant que fichier, possède un i-noeud qui est lui-même référencé dans un répertoire, qui est un fichier... et ceci jusqu'à la racine. Un répertoire donné, en plus des liens qu'il posséde vers ses répertoires fils et fichiers (potentiels), possède un lien vers lui-même et un autre vers son père (potentiel). Ces liens sont nommés respectivement "." et "..".
Tout fichier (et donc a fortiori tout répertoire) peut donc être référencé de 2 manières : par rapport à la racine (référence absolue) en indiquant le chemin complet d'accés et par rapport au répertoire courant (référence relative).

3.2 Navigation dans l'arborescence des fichiers

pwd print working directory - affiche le répertoire courant
ls liste le contenu du répertoire courant
file tente de déterminer le type du fichier
cd changer de répertoire
$HOME répertoire personnel
$PWD ou **$cwd** répertoire courant

3.3 La hiérarchie

```
/ ---- bin (commandes essentielles sous forme executable)
    |
    -- dev (fichiers speciaux - peripheriques)
    |
```

[6] Partition logique des disques physiques.

```
-- etc (fichiers systemes - divers)
|    |
|    -- passwd (fichier d'identification des utilisateurs)
|    |
|    -- profile (fichier)
|
-- lib (bibliotheques du systeme - y compris cc)
|
-- tmp (fichiers temporaires des applications du systeme)
|
-- home (partie utilisateur - dynamique)
|    |
|    -- cestac
|         |
|         -- leduc
|              |
|              -- .login
|              |
|              -- .profile
|
|
-- usr (partie systeme - relativement statique)
     |
     -- bin (commandes externes aux shells et shells)
     |    |
     |    -- csh
     |    |
     |    -- sh
     |
     -- lib (bibliotheques du systeme - y compris cc)
     |
     -- local
     |    |
     |    -- bin (commandes externes aux shells propre au site)
     |    |
     |    -- man (manuel UNIX propre au site, a l'architecture)
     |
     -- man (manuel UNIX)
     |
     -- tmp (fichiers temporaires des utilisateurs)
```

3.4 Les droits d'accès aux fichiers

Sous UNIX tout utilisateur possède un numéro de compte (**uid**) et appartient à un groupe donné (**gid**). Pour un fichier donné, un utilisateur quelconque est donc soit propriétaire, soit membre du groupe propriétaire du fichier (mais pas propriétaire), soit autre utilisateur.

```
$ id
uid=1222(leduc) gid=100(cestac) groups=100(cestac)
$
```

Il existe un super-utilisateur qui est privilégié puisqu'il dispose de tous les droits.

Il est possible, pour un fichier donné de faire 3 opérations : lecture, écriture, exécution (**rwx**). Les droits d'accès d'un fichier (répertoire) conditionnent l'usage qui peut en être fait par les diverses catégories d'utilisateurs. Ces droits sont stockés sur 3 groupes de 3 bits (ou 3 octets). Il y a donc 8 possibilités d'accès pour chacune des catégories d'utilisateurs :

droits d'accès	valeur binaire	valeur octale
- - -	000	0
- -x	001	1
-w-	010	2
-wx	011	3
r- -	100	4
r-x	101	5
rw-	110	6
rwx	111	7

$$\underbrace{r\ w\ x}_{user}\ \underbrace{r\ w\ x}_{group}\ \underbrace{r\ w\ x}_{other}$$

Exemple :

```
$ pwd
/home/cestac/leduc/TEX/BIDON/
$ ls -al
drwxr-sr-x  2 leduc        1024 Jan  4 10:37 .
drwxr-sr-x 14 leduc         512 Jan  3 18:10 ..
-rwxr--r--  1 leduc       18621 Nov 28 10:41 unFichier
lrwxrwxrwx  1 leduc          21 Oct 17 14:55 unLien -> ../TEX/uneRef
drwxr-sr-x  2 leduc         512 Jan  5 19:10 unRepertoire
...
```

Remarques :
le droit d'exécution pour un répertoire signifie qu'il peut-être "traversé" (on peut donc consulter éventuellement ses répertoires fils si ceux-ci l'autorisent). Attention au droit en écriture ($\Rightarrow$ droit de modifier les informations) sur un répertoire !
le droit d'exécution pour un simple fichier signifie que la frappe de son nom sous un shell est interprétée comme le nom d'une commande externe qui est aussitôt déclenchée.

umask et valeur par défaut L'**umask** est un octal qui, lorsque l'on considère son "complémentaire" (en octal) aux valeurs 777 (pour les répertoires) et 666 (pour les fichiers), permet de déterminer les droits d'accès accordés par défaut à un fichier UNIX. Exemple :

```
$
$ umask
0022
$ mkdir unRepertoire
$ touch unFichier
$ ls -l
total ...
-rw-r--r--  1 leduc           0 Jan  5 19:49 unFichier
drwxr-xr-x  2 leduc         512 Jan  5 19:49 unRepertoire
$
```

3.5 Modification des droits d'accès

Commande chmod (ponctuelle) Il y a deux façons de procéder :

chmod [valeur octale] fichier
chmod [ugo+/-rwx] fichier

Commande umask (globale)

Commande chown souvent inaccessible au simple utilisateur.

Commande chgrp souvent inaccessible au simple utilisateur.

3.6 Les fichiers spéciaux

Il y a plusieurs catégories de fichiers au nombre desquelles on compte : les fichiers "ordinaires" contenant des programmes (binaires, sources) ou des données (ASCII), les répertoires et les fichiers spéciaux (localisés dans /etc/dev ou /dev selon le système). Ces derniers sont soit des disques ou des ES (ils transitent par taille de 512 ou 1024 caractères par les caches du système d'IO), soit des terminaux (pour lesquels les ES sont réalisées caractère par caractère).

```
$ ls -l /dev
...
crw--w----  1 leduc      0,   0 Jan  5 20:39 console
crw-rw-rw-  1 root      11,   0 May  2  1994 des
crw-r-----  1 root       7,   0 May  2  1994 drum
crw-rw----  1 root      41,   0 May  2  1994 dump
crw-r-----  1 root       3,  11 May  2  1994 eeprom
crw-rw-rw-  1 root      22,   0 May  2  1994 fb
brw-rw-rw-  2 root      16,   2 May  2  1994 fd0
brw-rw-rw-  1 root      16,   0 May  2  1994 fd0a
brw-rw-rw-  1 root      16,   1 May  2  1994 fd0b
brw-rw-rw-  2 root      16,   2 May  2  1994 fd0c
...
srwxrwxrwx  1 root            0 Dec 18 15:59 printer
...
brw-r-----  1 root       7,   0 May  2  1994 sd0a
brw-r-----  1 root       7,   1 May  2  1994 sd0b
brw-r-----  1 root       7,   2 May  2  1994 sd0c
brw-r-----  1 root       7,   3 May  2  1994 sd0d
brw-r-----  1 root       7,   4 May  2  1994 sd0e
brw-r-----  1 root       7,   5 May  2  1994 sd0f
brw-r-----  1 root       7,   6 May  2  1994 sd0g
brw-r-----  1 root       7,   7 May  2  1994 sd0h
...
crw-rw-rw-  1 root       2,   0 Jan  5 21:48 tty
$
```

On remarque que les disques et bandes sont de type b (pour bloc) et que le reste est de type c (pour caractère).

```
$
$ /etc/mount   <--> indique la correspondance peripheriques/repertoires
/dev/sd0a on / type 4.2 (rw)
/dev/sd0g on /usr type 4.2 (rw)
/dev/sd0h on /home type 4.2 (rw)
$ df
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/sd0a               9931    4740    4198    53%    /
/dev/sd0g             204228  166073   17733    90%    /usr
/dev/sd0h            1566018 1271870  137547    90%    /home
$
```

/dev/console correspond à la console et, par ailleurs, /dev/tty est le terminal de connexion :

```
$ echo bonjour > /dev/tty
bonjour
$
```

Pour supprimer le flot de la sortie standard, il faut le rediriger vers /dev/null :

```
$
$ time date
Sun Jan  5 21:53:08 WET 1997
        0.0 real         0.0 user         0.0 sys
$ time date > /dev/null
        0.0 real         0.0 user         0.0 sys
$
```

/dev/fd0c correspond au périphérique lecteur de disquette qui peut alors être considéré exactement comme un répertoire de l'arborescence.

```
$ mountpc
/dev/fd0c mounted on /pcfs
$
```

Les descripteurs de fichiers Toute ES est une lecture ou une écriture de fichier (car tout est fichier sous UNIX). Avant toute ES il faut donc ouvrir (ou créer parfois dans le cadre de l'écriture) un fichier. Si le système nous y autorise alors il renvoie un entier appelé le descripteur de fichier. Par convention et pour faciliter le travail avec le terminal : au lancement par un shell, un programme hérite de 3 fichiers ouverts de descripteurs respectifs 0 (stdin - entrée standard), 1 (stdout - sortie standard) et 2 (stderr - sortie d'erreur standard). Si les ES sont redirigées vers des fichiers ou des tubes, le shell modifie l'assignation par défaut des descripteurs.

3.7 Quelques fichiers particuliers de configuration

.login Fichier particulier de commandes placé dans $HOME (le répertoire privé de chaque utilisateur) qui est évalué à chaque login-C-shell après le fichier personnel $HOME/.cshrc.

.logout Ce fichier est évalué à l'interruption du login-C-shell (il est de bon ton d'y inscrire un clear) à chaque fin de session.

.plan, .project Sont deux fichiers de renseignements propre à chaque utilisateur qui peuvent-être consultés à distance au moyen de la commande finger (attention : accorder droits en lecture sur ces fichiers à tous !). $HOME/.project doit tenir sur une ligne...

.cshrc Fichier particulier propre à chaque utilisateur évalué à chaque lancement d'un C-shell (ou Shell dérivé) et avant le fichier .login lors d'un login-shell.

.profile Fichier particulier propre à chaque utilisateur évalué à chaque lancement d'un Korn shell (ou Shell dérivé) et après le fichier /etc/profile lors d'un login-Korn-shell.

.history Fichier d'historique des commandes sous C shell.

.sh_history Fichier d'historique des commandes sous Korn shell.

3.8 Commandes de manipulation de fichiers

- od -option fichier

  Cette commande liste le contenu d'un fichier en mode : octal (option b, les octets sont visualisés en octal), ASCII (option c, les octets sont visualisés en ASCII), décimal (option d, les mots de 2 octets sont visualisés en décimal), octal (option o, les mots de 2 octets sont visualisés en octal), hexadécimal (option x, les mots de 2 octets sont visualisés en hexadécimal).

- cat -option fichier

  Cette commande liste le contenu d'un fichier. Avec l'option n ou l'option b, elle affiche le numéro à chaque début de ligne. Attention : $catf1f2 > f2$ commence par détruire f2 avant de le lire !

- more fichier

  Cette commande liste le contenu d'un fichier page par page. Le retour charriot permet d'avancer ligne par ligne dans le listing, la barre d'espace permet d'avancer page par page et la lettre q permet de terminer la commande.

- pg -NbLignes +NumPremiereLigne +/RE fichier

  Cette commande liste le contenu d'un fichier page par page sur un écran de "NbLignes" lignes (23 par défaut). En commençant ce listing à partir de la ligne numéro "NumPremiere-Ligne", ou à partir de la première ligne "matchant" l'expression régulière "RE" passée en argument.

- touch -option fichier

  Cette commande permet de modifier les dates d'accès en lecture et/ou en écriture d'un fichier. L'option c permet de ne pas créer le fichier s'il n'existe pas déjà.

- ln -s fichierReel lienSymbolique

  Permet de créer un lien symbolique sur un fichier donné en argument.

- cp -option fichierAncien fichierNouveau

  Cette commande permet de copier le contenu du fichier "fichierAncien" dans le fichier "fichierNouveau". L'option -i propose de confirmer toute opération, tandis que l'option f force l'exécution de la commande. Si "fichierAncien" et "fichierNouveau" sont deux répertoires, l'option r (ou R) permet d'effectuer récursivement cette opération.

- mv -option fichierAncien fichierNouveau

  Cette commande permet de déplacer (c'est-à-dire aussi renommer) le fichier "fichierAncien". Si le fichier "fichierNouveau" existe déjà, il est détruit lors de cette opération. L'option -i propose de confirmer toute opération, tandis que l'option f force l'exécution de la commande.

- rm -option fichier Cette commande permet de détruire le fichier spécifié. Avec l'option i, il y a demande de confirmation ; avec l'option f, on force l'exécution de cette commande ; l'option r permet de détruire récursivement le contenu d'un fichier de type répertoire.

3.8.1 Manipulation de répertoires exclusivement

- mkdir -option répertoire

  Cette commande permet de créer un nouveau répertoire. Avec l'option p elle autorise la création de tous les répertoires intermédiaires spécifiés en argument (s'ils n'existent pas).

- rmdir -option répertoire

  cf commande rm.

3.8.2 Travaux pratiques

Création d'une arborescence sous $HOME avec des répertoires src, bin, tmp, doc... Personnalisation des fichiers de configuration.

4 Les processus

4.1 Généralités

Un programme produit par compilation et édition de lien est un fichier (binaire, régulier, sur disque) de nom générique a.out. Un processus correspond à un instant donné à l'état d'avancement d'un tel programme ⇒ un processus est une entité dynamique correspondant à l'exécution d'un programme (ou de programmes différents du fait de l'utilisation de mécanismes de recouvrement).
Un processus c'est donc un ensemble constitué de l'exécutable (suite d'instructions), de l'ensemble des données manipulées par celui-ci et d'un *contexte d'exécution* (= ensemble des valeurs des registres des processeurs correspondant à l'état d'avancement de l'exécution, pile d'exécution, liens avec les utilisateurs et le système d'IO...). Cet ensemble est le *bloc de contrôle du processus.*
Toutes les activités dans le système (purement système ou sur demande précise d'un utilisateur) sont exécutées dans le contexte d'un processus. Le rôle du système est de permettre à tous les processus de progresser dans leur exécution. Le module du noyau ordonanceur (*scheduler*) alloue par *time-sharing* un *slice time* à chaque processus. De plus dans le cas multi-processeurs il répartit les processus sur les divers processeurs. Deux principes de base pour son fonctionnement : association d'une priorité dynamique (recalculée fréquemment) à chaque processus et stratégie du tourniquet (*round robin*) pour arbitrer les conflits dans le cas de priorités égales. Objectif : assurer à chaque processus sa part d'utilisation de la puissance processante de la machine. ⇒ processus = unité élémentaire manipulée par le scheduler.
Distinction entre :

- les processus systèmes ou *daemons*

  qui ne sont sous contrôle d'aucun terminal et ont le super-U comme propriétaire. Rôle : assurer les services généraux accessibles à tous (y compris les utilisateurs distants si réseau).

Leur répertoire de travail est souvent la racine afin de permettre le démontage de disques logiques sans avoir à les interrompre. Ils sont souvent crées au lancement du système...

init ce processus a 1 pour numéro, c'est le processus père des processus shells crées par tout utilisateur se connectant depuis un terminal alphanumérique classique.

inetd superviseur des services réseaux utilisant l'IP. C'est un super serveur qui prend en charge les requêtes distantes telles que demande connexion, exécution de commande à distance, copie de fichier à distance... il crée alors sur demande un sous processus.

cron lancement à des dates spécifiées de commandes particulières

- les processus utilisateurs

  sont souvent sous le contrôle d'un terminal. Il y a par exemple le login-shell, l'exécution de scripts, de binaires, l'appel à la primitive de création de processus *fork*...

Un processus UNIX possède un certain nombre de caractéristiques, certaines sont dynamiques (évoluent avec le temps). Dans la "génétique" des processus, tout processus fils hérite des propriétés du processus père et tout processus fils (hormis le processus 0 *swapper*) est généré par un *fork*[7] à la demande d'un autre processus. Les caractéristiques d'un processus sont : son identifiant (PID), l'identifiant du père (PPID), son état (STAT : R actif, S endormi), son terminal d'attachement s'il existe (TTY), sa priorité (CP, PRI, NI), son répertoire de travail, le temps cumulé d'exécution (TIME), le nom du fichier de commande correspondant au processus (COMMAND)...

4.2 Gestion élémentaire des processus

4.2.1 La commande ps

4.2.2 Arrêter un processus

Le zombi C'est un processus n'utilisant plus de ressources CPU mais encore une entrée dans la table des processus (s'assurer de sa bonne élimination pour ne pas engorger cette table). Ce processus est terminé mais son père n'a pas connaissance de sa terminaison.

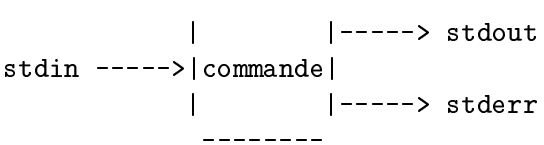
La procédure Emission d'un intr (CTRL C) ou d'un quit[8] (CTRL \) à partir d'un terminal. Utilisation de la commande **kill -ID_SIG NUMERO_PROCESSUS**.

4.3 Fichiers standards et redirection

Idée d'UNIX : les applications développées doivent pouvoir être réutilisées et composées entre-elles au moyen de mécanismes accessibles facilement du shell (⇒ pas de retouche des codes des applications).
⇒ une application est une boîte noire qui, par conventions, lit ses données sur le fichier logique appelé **entrée standard du pcs** (stdin) et écrit ses résultats sur un fichier logique appelé **sortie standard du pcs** (stdout). En mode de fonctionnement interactif, ces 2 fichiers logiques sont associés au terminal de l'utilisateur. En interne, ces 2 fichiers correspondent au 2 premiers **descripteurs** (indices) dans la tables des fichiers manipulés par le processus. Ils ont respectivement 0 et 1 pour valeur.Un pcs hérite à sa création des descripteurs de son père (⇒ tout pcs lancé depuis un shell interactif l'est aussi).
L'association d'un tel fichier logique à un fichier physique peut-être modifié par le phénomène de redirection. Cependant, il existe un autre "canal" de sortie pour afficher des diagnostiques,

[7] ou plutôt par exécution de l'appel système fork...
[8] Le quit génère un fichier core dans le répertoire de lancement, limitation en csh par : limit coredumpsize 0.

traces ou erreurs appelé **sortie erreur standard du pcs** (std error output) de descripteur égal à 2. La sortie est dite multiplexée. stderr en mode interactif est également associée au terminal de l'utilisateur.



Par convention, un pcs peut désigner son terminal de ctrl par la référence /dev/tty.

Redirection de stdin *commande* < *ficref* lancée depuis un shell interactif signifie que stdin est redirigée sur le fichier "ficref" et non plus associée au terminal. Il faut bien sûr que ficref existe et soit lisible par l'utilisateur. Exemple :

```
$ cat f
un
deux
trois
$ wc -l < f
  3
$ wc -l f
  3 f
$
```

Dans le premier cas, on lit le nombre de lignes sur stdin et dans le deuxième cas, on lit le fichier passé en paramètre. Premier cas : avant de charger /bin/wc, stdin est redirigée et wc est lancée sans paramêtre ; "< f" est analysé et la commande est leurrée (elle a l'impression de travailler sur stdin et non sur un fichier du disque ⇒ pas de nom de fichier en stdout). Dans le second cas, le fichier est connu puisqu'il est en paramêtre et ce nom est affiché sur demande de la commande wc.

Redirection de stdout *commande* > *ficref* cré ou écrase (sauf shell particulier) le fichier "ficref" avec le contenu de la sortie standard. Par contre, *commande* >> *ficref* se contente d'ajouter en fin de ficref le contenu de stdout.

Redirection de stderr Prouvons-nous l'existence de stderr :

```
$ ls f1 f2
f1 not found
f2
$ ls f1 f2 > f
f1 not found
$ cat f
f2
$
```

Petit exemple de redirection de stderr en Bourne-Shell :

```
$
$ ls f1 f2
f1 not found
f2
$ ls f1 f2 2> ferr
f2
$ cat ferr
f1 not found
$ ls f1 f2 2> ferr > f
$ cat f
f2
$ cat ferr
f1 not found
$
```

4.4 Enchaînement de processus

Sans communication entre eux Séquentiellement et indépendamment : ";"
Séquentiellement et conditionnellement : "&&" et "||"

Avec communication sommaire Utilisons les mécanismes de redirection et l'enchaînement séquentiel des tâches, comme dans l'exemple ci-dessous :

$ ps -ef > /tmp/bidon ; wc -l < /tmp/bidon ; rm /tmp/bidon

Avec communication évoluée par tubes ou "pipes" Il existe un mécanisme permettant de lancer un certain nombre de processus de manière concurrente (ou encore "parallèle") et de les faire communiquer par un tube (pipe). Le système gère alors la synchronisation de l'ensemble dans la mesure où il bloque le processus lecteur du tube si celui-ci est vide en attendant qu'il se remplisse d'une part, et, d'autre part, il tempère l'écriture du processus écrivain dans le tube en fonction de la vitesse de lecture du processus lecteur. Ainsi la commande **ps | wc -l** crée simultanément deux processus concurrents (au sens où ils partagent les mêmes ressources du système) et un tube. Les résultats de **ps** y sont écrits (sortie standard redirigée vers l'entrée du tube) et lus par la commande **wc -l** (entrée standard redirigée sur la sortie du tube).

4.5 Lancement en arrière plan

Deux moyens : "&" en fin de ligne de commande et combinaison de "CTRL Z" suivi de la commande *bg* à partir d'un terminal.
Attention : job control non supporté par le Bourne-shell (pas de commande jobs).

4.6 Autres commandes

at, batch, kill, nohup, time

nohup

nice [-increment] commande lance "commande" avec une propriété nice incrémentée de la valeur de "increment" (comprise entre 1 et 19 et valant 10 par défaut) ⇒ diminution de la priorité du processus. Le super-U peut utiliser un argument négatif sous la forme nice --15 commande (plus prioritaire).

at

```
$ at now + 1 minutes
at> date > fichier
at> <EOT>
job 4372 at Tue Jan  7 21:49:00 1997
$
```

Types d'unités possibles : minutes, hours, days, weeks ou months.
Heures symboliques utilisables : noon, midnight et now (avec un incrément), today et tomorrow.
Si stdout et stderr ne sont pas redirigées, alors par défaut tout est envoyé sur la boîte aux lettres de l'utilisateur.

5 Les principales commandes externes

Calcul d'expressions arithmétiques : bc

```
$ bc
2^3
8
sqrt(16)
4
scale(12.345)
3 --> nombre de decimales
a=3.1
b=5.7
a/b
0
scale=4
a/b
.5438
for(i=0;i<14;i++) s+=i ; s
91
(13*14)/2
91.0000
obase=8
12
14
obase=10
ibase=2
111
7
quit
$
```

Pour avoir la bibliothèque mathématique de calcul en grande précision, lancer la commande bc -l.

Calcul d'expressions arithmétiques : expr

```
$ expr 13 \* 3
39
```

```
$ echo $? <-- demande d'affichage de l'exit-status de la commande
0
$ x=5
$ x=‘expr $x \* 3‘
$ echo $x
15
```

Recherche d'une chaîne dans un fichier : grep Permet de sélectionner dans un fichier les lignes contenant un motif correspondant à l'expression régulière donnée en argument. Les options sont c (afficher uniquement le nombre total de lignes satisfaisant l'expression), i (pas de discussion majuscule-minuscule), l (seuls les noms des fichiers contenant des lignes avec l'expression sont écrits), n (précise les numéros de lignes), s (pas de message indiquant l'impossibilité d'ouvrir un fichier), v (les lignes ne contenant pas le message sont affichées).
exemple: $ls - al|grep“ \wedge d''$ liste tous les répertoires du répertoire courant.

Impression des premières lignes d'un fichier : head *head -n fichier1 fichier2...* affiche sur stdout les n permières lignes des fichiers passés en argument.

Impression des dernières lignes d'un fichier : tail *tail -n fichier* affiche sur stdout les n dernières lignes du fichier passé en argument.

Juxtaposition de lignes de plusieurs fichiers : paste Le nombre de fichiers en argument doit-être de 12 au maximum.

```
$ cat > f1
a1
b1
c1
$ cat > f2
d2
e2
f2
$ paste -d"\t" f1 f2
a1      d2
b1      e2
c1      f2
$
```

Extraction de portions de lignes dans un fichier : cut *cut -cN-M f1* permet d'extraire la portion de ligne allant du caractère N au caractère M au sein de f (pour toutes les lignes).
cut -fN-M -dC f1 permet d'extraire la portion de ligne allant du champ N au champ M au sein de f (pour toutes les lignes), le séparateur de champ étant C.

```
$ cat f
a cd ef
b cd fg
c cd hi
$ cut -f1-2 -d' ' f
a cd
b cd
```

```
c cd
$ cut -c1-3 f
a c
b c
c c
$
```

Recherche d'un fichier dans une arborescence : find Cette commande parcourt recursivement l'arborescence des fichiers pour rechercher des fichiers. Options :
-name f1 vrai si l'argument f1 est égal au nom du fichier
-type t vrai si le type du fichier est t (d pour directory, f pour fichier ordinaire, b pour fichier spécial bloc, c pour fichier spécial caractère, l pour lien symbolique...)
-user u vrai si le propriétaire est u
-group g vrai si le groupe propriétaire est g
-size s vrai si le fichier est de taille $s * 512$ octets
-atime n vrai si le fichier a été visité dans les n derniers jours
-mtime n vrai si le fichier a été modifié dans les n derniers jours
-exec cmd vrai si l'exécution de la commande cmd renvoie 0. La fin de la commande doit être marquée par un anti-slash suivi d'un point-virgule et le nom du fichier courant est représenté par {}.
-print toujours vrai affiche le nom du fichier en cours s'il satisfait les conditions.
-ok cmd identique a -exec sauf que la commande est écrite sur la sortie standard puis l'entrée standard est lue et la cmd n'est exécutée que si la réponse est y.
-prune éviter le parcours récursif de l'arborescence
Combinaison de ces options :

!e	negation d'un élément
e -o e	OU entre deux éléments
e e	ET (juxtaposition) entre deux éléments
()	ensemble d'éléments et d'opérateurs (attention les parenthèses doivent-être neutralisées pour ne pas être interprétées par le shell)

```
$ find . \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
$ find $HOME -type f -exec grep UNIX {} \; -print
```

Les deux commandes qui précèdent permettent respectivement de "faire le ménage" des vieux binaires dans le répertoire courant et tous ses sous répertoires d'une part et de rechercher récursivement à partir du "home-directory" tous les fichiers qui contiennent la chaîne de caractères "UNIX" d'autre part.

Comparaison de fichiers : cmp *cmp -option f1 f2* compare deux contenus de fichiers et indique la position de première différence. Si l'option l est choisie, toutes les différences sont signalées et si l'option s est choisie, la commande ne renvoie rien à l'écran mais juste un "code de retour" (0 pour fichiers identiques, 1 pour fichiers différents et 2 en cas d'erreur).

```
$
$ cat f1
a
b
c
$ cat f2
```

```
b
c
d
$ cmp f1 f2
f1 f2 differ: char 1, line 1
$ cmp -l f1 f2
     1 141 142
     3 142 143
     5 143 144
$ cmp -s f1 f2
$ echo $?
1
$
```

Comparaison de fichiers : comm *comm -option f1 f2* affiche sur trois colonnes le résultat de la comparaison de f1 et f2. En première colonne, il y a les valeurs spécifiques à f1, en deuxième colonne, les valeurs spécifiques à f2 et en dernière colonne les valeurs communes. En option, on indique les colonnes occultées, ainsi comm -12 n'imprime que les points communs.

Tri de fichier : sort *sort -options +position f1* trie f1 en ordre croissant (sauf si options vaut r) selon l'ordre lexicographique à partir du position-ème champ (premier par défaut). L'option f indique que les majuscules et les minuscules sont identiques et l'option i ignore tous les codes ASCII extérieurs à l'intervalle 32-126. Enfin l'option u élimine les lignes redondantes.

```
$
$ cat f
aa 123 5
ab 120 6
cd 129 2
aa 111 1
$ sort f
aa 111 1
aa 123 5
ab 120 6
cd 129 2
$ sort +1 f
aa 111 1
ab 120 6
aa 123 5
cd 129 2
$ sort +2 f
aa 111 1
cd 129 2
aa 123 5
ab 120 6
$
```

Spliter un fichier : commande split

```
split -b 64k fichier.uu x_
split -4000 fichier.uu x_
```

Commande wc

Archivage de fichiers : tar options cvf, tvf et xvf.

Commande tr *tr ch1 ch2* cette instruction permet de copier stdin sur stdout en effectuant une traduction de chaque caractère.

```
$ ls | tr '[a-z]' '[A-Z]'
```

Copie après conversion : dd *dd if=fichEntree of=fichSortie [conv=lcase | ucase]* transforme toutes les lettres du fichier d'entrée en minuscules ou majuscules.

Formatage de texte : fmt *fmt -23 montexte.txt* permet d'afficher sur la sortie standard le contenu du fichier montexte.txt placé sur 23 colonnes.

Modification d'identité : su

Informations systèmes : date, du (espace mémoire alloué), df (état des disques logiques)

Gestion du terminal : clear, stty

Gestions des impressions papier : lpr, lpq, lprm, lpstat

6 L'interface de commande - les shells

6.1 Introduction

Les langages d'interprétation de commandes du système UNIX sont appelés les *shells*. Un shell c'est un langage de commande mais aussi un langage de programmation. Chaque shell comporte un certain nombre d'instructions propres appelées aussi *commandes internes* (à ne surtout pas confondre avec les commandes externes). L'exécution d'une commande interne implique donc qu'il n'y a pas création d'un nouveau processus UNIX.
En tant que langage de commande (mode interactif, conversationnel), un shell constitue l'interface utilisateur-système. Il dispose alors de fonctionnalités facilitant le travail des utilisateurs : historique, complétion...
En tant que langage de programmation, un shell permet d'écrire des *scripts* (ou *batchs*). Ces (petits) programmes permettent de regrouper des fonctions complexes dans des fonctions plus simples à appeler ou d'automatiser des opérations compliquées ou devant se reproduire à heure fixe (cf crontab) par exemple. Attention, un script est relatif au shell dans lequel il a été écrit !

6.1.1 Sélection de l'interprèteur à l'appel du script

Appelons αsh notre shell et monScript le script à interpréter. On peut alors lancer le script de diverses manières, en voici quelques unes :

- /bin/αsh $HOME/SCRIPTS/monScript
- source $HOME/SCRIPTS/monScript
- cat $HOME/SCRIPTS/monScript | /bin/αsh (à manipuler avec précaution),

- /bin/αsh < $HOME/SCRIPTS/monScript (à manipuler avec précaution),

si, l'on prend soin de préciser en première ligne du fichier monScript l'instruction suivante :

#!/bin/αsh

on peut alors encore procéder de la manière suivante :

- chmod u+x $HOME/SCRIPTS/monScript ; $HOME/SCRIPTS/monScript
- chmod u+x $HOME/SCRIPTS/monScript ; exec $HOME/SCRIPTS/monScript

6.1.2 Choix de l'interprèteur

Deux grandes familles d'interprèteurs : celle des Bourne-shells dérivés du langage de commande originel d'UNIX et celle des C-shells dérivés du langage de commande des distributions Berkeley d'Unix. Attention, le choix d'un interprèteur est une question de goût... mais on a pu faire les constatations suivantes :

- **sh** (Bourne-shell) est très portable et adapté aux petits scripts mais tout à fait déconseillé pour les grands scripts ou les scripts à "tendance arithmétique" (cf bc). Il serait peu fiable.
- **ksh** (Korn shell) relativement déconseillé à l'utilisation interactive, cette extension est cependant (normalement) compatible avec sh et suffisamment normalisée. En plus, contrairement à sh, elle supporte les tableaux et les valeurs numériques.
- **csh** (C shell) à manipuler avec précaution, car il a subit de trop nombreuses mises à jour et aurait quelques contradictions dans sa grammaire. Il faut lui préférer tcsh.
- **tcsh** (Turbo C shell) est très recommandé pour une utilisation interactive grâce à ses facilités à la complétion et à l'édition.

Mais on recense aussi rsh (Restricted Shell), jsh (Job Shell), rksh (Restricted Korn-Shell), bash (Bourne-Again-Shell)...
Enfin, à la limite de cette catégorie, il existe **perl** (un C-interprété) très rapide, qui offre des possibilités étendues en matière de sécurité.
Remarque : les shells sont souvent plus portables et plus rapides à écrire qu'un programme destiné à être compilé (en C par exemple) mais ils sont souvent beaucoup plus lents à l'exécution du fait de l'appel aux commandes externes. Il faut donc trouver un compromis.

6.2 Généralités sur les langages de commande - les mécanismes

A la connection, un processus correspondant à un interprète shell (αsh par exemple) est lancé. Il s'agit du *login-shell*. Un tel processus (visible avec la commande ps sous le nom $-\alpha sh$) lance plusieurs commandes lues dans des fichiers particuliers (/etc/passwd en particulier[9] qui permet d'identifier l'interpréteur de commande à utiliser, mais aussi $HOME/.login...) avant de passer en arrière plan.
Si votre login-shell est αsh et que vous voulez passer sous βsh, il faut taper la commande suivante :

chsh $LOGNAME /bin/$\beta sh$

Exemple :

[9] Sur un système UNIX classique.

```
achille{leduc}1: echo $SHELL
/bin/tcsh
achille{leduc}2: chsh $LOGNAME /bin/sh
Changing login shell for leduc on achille.
achille{leduc}3: echo $SHELL
/bin/tcsh --> le shell courant reste identique
achille{leduc}4: rsh achille
Last login: ...
SunOS Release 4.1.4 (GENERIC) ...
$ echo $SHELL
/bin/sh --> voila le nouveau shell
$
```

ou demander à l'administrateur système de modifier le fichier des identifications.
La liste des shells disponibles sur votre plate-forme est placée dans le fichier /etc/shells.

6.2.1 Les métacaractères

A ne pas confondre avec les caractères de contrôle du terminal, ils sont généralement supportés par tous les interpréteurs. On recense ainsi :

- le commentaire de ligne, caractère #,
- la protection ou déspécialisation, caractère \ qui, placé devant un métacaractère, permet de lui supprimer tout sens particulier et qui est ignoré ailleurs,
- les délimiteurs de chaîne, caractères ",' ou ‘,
- le séparateur (espace),
- la redirection (permettant d'associer fichiers physiques et fichiers logiques),
- la gestion des processus associés aux commandes externes du script, caractères & (arrière-plan) , | (concurrents), ";"(séquentiels), && et || (conditionnels),
- la substitution, caractère $,
- l'expansion de chaînes, caractères ~, *, ?, [,] et -,
- l'historique, caractère ! en C-shell,

6.2.2 Le mécanisme de substitution

L'historique Accessible en C-shell par le caractère "!".

Les alias En C-shell : alias ll 'ls -l'
En Korn shell : alias ll='ls -l'

Le caractère tilde ñomDeLogin nous envoie sur le home-directory de l'utilisateur "nomDeLogin" s'il existe.

Les variables Une variable = chaîne de caractères alphanumériques commençant par une lettre (underscore est une lettre).IL existe plusieurs types de variables : variables définies par l'utilisateur, variables d'environnements (exportables)...
Type majeur : chaîne de caractères.
Parfois, selon le shell, le type numérique et les tableaux sont autorisés.
Pour substituer une valeur à la variable, il faut utiliser le $. Ainsi $maVar ou ${maVar} renvoient dans la plupart des shells la valeur de maVar.

6.2.3 Le mécanisme d'expansion

?	un caractère quelconque
*	une chaîne quelconque (vide ou non) de caractères
[	début de définition d'un ensemble
[!	début de définition du complément d'un ensemble
]	fin de définition d'un ensemble ou de son complément
–	marque d'intervalle dans l'ensemble

6.2.4 Les délimiteurs de chaînes

- le caractère '

  tous les caractères placés entre deux simples quotes sont protégés (à l'exception de la simple quote elle-même),

- le caractère "

  tous les caractères placés entre deux simples quotes sont protégés, à l'exception des caractères $ (substitution de variable), \ (déspécialisation), ' (quote) et `(anti quote, évaluation des commandes), une telle chaîne peut contenir le caractère " en le faisant précéder par \.

- le caractère `

  une chaîne délimitée par l'anti quote est interprétée comme une commande (après réalisation des substitutions des valeurs aux variables) et le résultat de la commande lui est substitué. Ainsi : **echo "je m'appelle $LOGNAME"**, **echo je m\'appelle $LOGNAME**, **echo je m\'appelle `whoami`** et **echo "je m'appelle `whoami`"** retournent le même résultat.

6.2.5 Processus et contrôle des tâches

Lancements Les interprètes shells proposent divers moyens différents pour lancer une commande (soient cmd1 et cmd2 deux commandes) :

- **cmd1** permet un lancement synchrone de la commande, le processus shell attend la réalisation des diverses substitutions puis la fin de l'exécution de celle-ci pour continuer l'analyse,
- **(cmd1)** permet un lancement de la commande au sein d'un sous-processus shell,
- **cmd1 &** permet un lancement asynchrone de la commande (en arrière-plan). Le processus shell reprend la main ;
- **cmd1 ; cmd2** exécute séquentiellement cmd1 puis cmd2
- **cmd1 && cmd2** exécute cmd1 puis, s'il n'y a pas d'erreur, exécute cmd2,
- **cmd1 || cmd2** exécute cmd1 puis, s'il y a eu erreur, exécute cmd2,

Redirections Soit cmd une commande et fic un nom de fichier.

- **cmd** > **fic** lance la commande et redirige la sortie standard vers le fichier de référence (pour forcer cette commande en csh, il faut faire **cmd** >! **fic**),
- **cmd** < **fic** lance la commande en redirigeant l'entrée standard sur le fichier de référence),
- **cmd** >> **fic** lance la commande et redirige la sortie standard vers la fin du fichier de référence (pour forcer cette commande en csh, c'est-à-dire créer le fichier s'il n'existe pas, il faut faire **cmd** >>! **fic**),

pour mémoire, on recense aussi les redirections suivantes : <<, <<! , > &, < &...

Communications Il existe un mécanisme permettant de lancer un certain nombre de processus de manière concurrente (ou encore "parallèle") et de les faire communiquer par un tube (pipe). Le système gère alors la synchronisation de l'ensemble dans la mesure où il bloque le processus lecteur du tube si celui-ci est vide en attendant qu'il se remplisse d'une part, et, d'autre part, il tempère l'écriture du processus écrivain dans le tube en fonction de la vitesse de lecture du processus lecteur. Ainsi la commande **ps** | **wc -l** crée simultanément deux processus concurrents (au sens où ils partagent les mêmes ressources du système) et un tube. Les résultats de **ps** y sont écrits (sortie standard redirigée vers l'entrée du tube) et lus par la commande **wc -l** (entrée standard redirigée sur la sortie du tube).

6.3 Un petit exemple illustrant les similitudes et différences entre le C - langage de haut niveau, le C-shell et le Korn-shell

```
#include<stdio.h>
#define PI 3.14159

main() {
float rayon,circonf;
printf("Entrez le rayon du cercle :\t"); scanf("%g",&rayon);
while (rayon >= 0) {
  circonf=2*PI*rayon;
  printf("La circonference du cercle correspondant est %g\n",circonf);
  printf("Entrez un nouveau rayon de cercle :\t"); scanf("%g",&rayon);}
}
```

```
#!/bin/csh
set pi=3
echo "Entrez le rayon d'un cercle :"
set r=$<
while($r >= 0)
@ c=2 * $pi * $r
echo "circonference associee : $c"
echo "Entrez un nouveau rayon :"
set r=$<
end
```

```
#!/usr/bin/ksh
# Voici un script en Korn-shell
PI=3
```

```
PS3="Quel numero de commande selectionnez-vous ? "
# PS3 est le prompt associe a la commande select
select choix in Terminer Calculer
do
        echo Commande selectionnee : $REPLY ou encore "$choix"
        if [ "$choix" = Terminer ]
        then echo A bientot...
             exit 0
        elif [ "$choix" = Calculer ]
        then echo Calcul de la circonference du cercle
                echo Entrez le rayon :
                read rayon
                typeset -i circonf=2*PI*rayon
                # circonf doit etre interpretee comme un entier
                echo La circonference vaut : "$circonf"
        else echo Commande non identifiee
        fi
done
```

Attention : veillez a bien respecter les espacements !

6.4 Petit memento csh

Manipulation des variables Les commandes de manipulation des variables sont les suivantes :

set var	permet de définir une variable (sa valeur est la chaîne vide)
set var = valeur	affecte à la variable locale la chaîne valeur
set var = $<	affecte à la variable locale la chaîne entrée au terminal
setenv var [valeur]	permet de définir une variable d'environnement (mécanisme d'exportation)
@ var = expression	évalue une expression numérique et affecte sa valeur à la variable
unset var	supprime la définition de la variable locale
unsetenv var	supprime la définition de la variable d'environnement
printenv ou env	liste les valeurs des variables d'environnement
set ou @	liste les valeurs des variables locales
$?var	renvoie 1 si la variable a été définie et 0 sinon

Manipulation des variables de type tableau

```
% set tab=(a b c)
% echo $?tab
1
% echo $#tab
3
% echo $tab[1]
a
% echo $tab[4]
tab: Subscript out of range.
% echo $tab
a b c
```

```
% echo $tab[2-3]
b c
```

Manipulation des variables et expressions arithmétiques attention à bien respecter les espaces...

```
% set a=3
% set b=3
% @ c= $a * $b ; echo $c
9
%
```

Manipulation des arguments de la ligne de commande

```
% cat exemple
#!/usr/bin/csh
echo "nombre de parametres : $#argv ou encore : ${#argv}"
echo "liste des parametres : $argv[*] ou encore : $* ou encore : $argv[1-$#argv]"
echo "nom de la commande : $0"
echo "premier parametre : $argv[1] ou encore : $1"
echo "second parametre : $argv[2] ou encore : $2"
echo "numero de processus shell de la commande : $$"
% exemple a b c d
nombre de parametres : 4 ou encore : 4
liste des parametres : a b c d ou encore : a b c d ou encore : a b c d
nom de la commande : exemple
premier parametre : a ou encore : a
second parametre : b ou encore : b
numero de processus shell de la commande : 9951
%
% cat lister
#!/usr/bin/csh
if (($#argv == 3) && ($1 <= $2)) then
        head -$2 $3 | tail +$1
else echo "Usage : lister nl1 nl2 fichier (avec nl1 <= nl2)"
endif

%
```

Les variables pré-définies Elles servent à paramétrer l'environnement de chaque utilisateur et peuvent permettre de modifier le comportement de l'interpréteur de commande en certaines circonstances...

set filec	activation du mécanisme de complétion des noms de fichier
set ignoreeof	un EOF ne pourra pas nous faire sortir du login-csh
set history=40	activation du mécanisme d'historique (40 commandes retenues)
set prompt="`hostname`{`whoami`}\!: "	modification du prompt (le ! désigne le numéro de la commande courante)
status	code de retour du dernier processus

Les alias - ou la redéfinition d'une chaîne de caractères par un mnémonique Les trois alias suivants sont équivalents :

alias cd 'cd \!:1;echo $cwd'
alias cd 'cd \!:1-1;echo $cwd'
alias cd 'cd \!*;echo $cwd' (liste des paramètres de l'alias)
alias cd 'cd \!$;echo $cwd' (premier des paramètres)
alias cd 'cd \!^;echo $cwd' (dernier des paramètres)

Manipulation des variables de l'historique Pour avoir l'historique des dernières commandes, taper "history".

!!	permet d'obtenir la dernière commande
!nbre	permet d'obtenir la commande numéro "nbre"
!chaine	permet d'obtenir la dernière commande commençant par "chaine"
!-nbre	permet d'obtenir la commande exécutée il y a "nbre" commandes
!nbre:s/chaine1/chaine2/	substitue chaine1 à chaine2 dans la "nbre-ième" commande
!!chaine	concatène "chaine" à la dernière commande

Instructions internes du C-shell décrivons à présent rapidement la syntaxe de ce langage de commande.

```
etiquette:
goto etiquette

if (expr) cmd

if (expr) then
bloc de cmd
else bloc de cmd
endif

if (expr) then
bloc de cmd
else if (expression) then
bloc de cmd
else if ...
endif

switch (valeur)
```

```
case valeur:
bloc de cmd
breaksw
...
default:
bloc de cmd
endsw

repeat entier cmd

foreach var (liste)
bloc de cmd
end

while (expr)
bloc de cmd
end

break, continue : instructions de sortie et de reprise de boucle
onintr etiquette : branchement automatique sur reception d'un signal

dirs, pushd, popd
```

Des exemples de scripts C-shell

```
% cat find_en_profondeur
#!/usr/bin/csh -f
# l'option -f permet de lancer le csh en mode fast (pas de lecture
# du .cshrc)
if ($#argv == 2) then
        find $1 -type f -exec grep $2 {} \; -print
else echo "Usage : find_en_profondeur repertoire chaine"
endif
%

% cat ~/SCRIPTS/destroy
#!/bin/csh
# ------------------------------------------------------------
# Ce script permet de detruire tous les processus rattaches
# a la chaine de caractere passee en argument ($1) de la
# commande.
# ------------------------------------------------------------
ps -ax | grep $1 | egrep -v 'grep|'$0'' | awk '{print $1}' | xargs kill -9
%

% cat ~/SCRIPTS/fmajuscule
#!/bin/csh
foreach fic ($*)
mv $fic `echo $fic|dd conv=ucase`
end
%
```

```
% cat ~/SCRIPTS/vitesse
#!/bin/csh
# ------------------------------------------------------------
# ce script sert a evaluer la vitesse de chargement d'un
# fichier par ftp...
# ------------------------------------------------------------
if ($#argv > 0) then
# ########################
# Commandes peu precises :
#       set taille1 = `du -s $1 | cut -f1 ; sleep 10`
#       set taille2 = `du -s $1 | cut -f1`
# ########################
        set taille1 = `ls -l $1 | awk '{print $4}' ; sleep 5`
        set taille2 = `ls -l $1 | awk '{print $4}'`
        @ delta=($taille2 - $taille1) / 5000
        @ taille=$taille2 / 1000
        echo "Taille actuelle : $taille Ko"
        echo "Vitesse de transfert : $delta Ko"
else echo "USAGE : vitesse {file}"
endif
%
```

6.5 Petit memento ksh

Manipulation des variables Les commandes de manipulation des variables sont les suivantes :

read var	affecte à la variable "var" la valeur lue sur l'entrée standard
var=valeur	affectation de valeur à une variable locale
var=valeur export	affectation de valeur à une variable globale
typeset [±filrux] var=val	définition et/ou typage de variable (f pour fonction, i pour entier, l pour transformer les majuscules en minuscules, r pour ne rendre la variable accessible qu'en lecture, u pour transformer les minuscules en majuscules et x pour exporter la variable)
unset var	supprime la définition de la variable
unset -f fct	supprime la définition de la fonction

Manipulation des variables de type tableau

```
$ set -A tab aa bb cc dd
$ set | grep tab
tab[0]=aa
tab[1]=bb
tab[2]=cc
tab[3]=dd
$ echo $tab ou ${tab}
aa ou aa
$ echo ${#tab}
2
$ echo ${#tab[*]}
4
```

```
$ echo ${tab[*]}
aa bb cc dd
$ echo ${tab[0]}, ${tab[1]}, ${tab[2]}, ${tab[3]}
aa, bb, cc, dd
$
```

Manipulation des variables et expressions arithmétiques

```
$ valeur=4
$ typeset -i valeurTriple=3*valeur
$ echo $valeurTriple
12
$ let "2>3"
$ echo $?
1
$ let "2<3"
$ echo $?
0
$
```

Manipulation des arguments de la ligne de commande

```
$ cat ex1.ksh
echo "nombre de parametres : $#"
echo "liste des parametres : $* ou encore : $@"
echo "nom de la commande : $0 ou ${0}"
echo "premier parametre : $1 ou ${1}"
echo "second parametre : $2 ou ${2}"
echo "numero de processus shell de la commande : $$"
$ ex1.ksh a b c d
nombre de parametres : 4
liste des parametres : a b c d ou encore : a b c d
nom de la commande : ex1.ksh ou ex1.ksh
premier parametre : a ou a
second parametre : b ou b
numero de processus shell de la commande : 28434
$
```

Les variables pré-définies Elles servent à paramétrer l'environnement de chaque utilisateur et peuvent permettre de modifier le comportement de l'interpréteur de commande en certaines circonstances...

HISTFILE	nom du fichier utilisé par l'historique
HISTSIZE	nombre de lignes de l'historique
MAIL	nom de la boite aux lettres contrôlée
PS1	prompt principal du ksh (PS1="`hostname`{`whoami`}\!: ")
SECONDS	nombre de secondes écoulées depuis le lancement du ksh
PPID	numéro de processus du père
RANDOM	nombre aléatoire
$	numéro du processus courant
?	code de retour du dernier processus
_ (underscore)	dernier mot de la dernière commande

Les alias - ou la redéfinition d'une chaîne de caractères par un mnémonique Il est possible de redéfinir des abréviations ou surnoms de commandes :

alias	listing complet des alias
alias nom=valeur	définition d'un alias
alias -x nom=valeur	l'alias est exporté
unalias nom	suppression de la définition d'un alias

Manipulation des variables de l'historique Première des choses à faire :

$EDITOR = vi\ export$ ou $EDITOR = emacs\ export$ pour se retrouver dans le mode conversationnel de votre choix...

Instructions internes du Korn-shell

```
if cmd1
then cmd2
else cmd3
fi

if cmd1
then cmd2
elif cmd3
then cmd4
elif cmd5
...
fi

case choix in
motif1 ) cmd1 ;;
motif2 ) cmd2 ;;
...
* ) cmdn ;;
esac

for var [in choix1 choix2 ...]
do cmd
done

select var [in choix1 choix2 ...]
```

```
do cmd
done

while cmd1
cmd2
done

until cmd1
cmd2
done

break

Traitement des exceptions :
trap 'rm $HOME/tmp/*;exit' 2 3
videra le repertoire ~/tmp a la reception des caracteres intr ou quit
du terminal (le exit est necessaire pour terminer le processus apres
ce nettoyage)
```

Les fonctions du Korn-shell

```
function nomDeFonction { cmd; }

nomDeFonction() { cmd; }

avec retour possible d'une valeur entiere par la commande :
return entier
```

7 L'édition de texte

7.1 Notion d'expression régulière

Il s'agit d'un mécanisme qui permet de décrire des ensembles de mots. Les utilitaires standards du monde UNIX (ed, sed, vi, grep...) l'utilisent également.

Les caractères spéciaux

[	début de définition d'un ensemble de caractères (une "classe lexicale")
.	un caractère quelconque exceptée la fin de ligne
$*$	indicateur d'itérations d'un nombre quelconque de fois
$+$	Une occurence au moins de l'expression régulière qui précède
?	Une occurence au plus de l'expression régulière qui précède
|	permet d'envisager des alternatives
()	regroupement d'expression régulières pour en former de plus vastes
]	fin de définition d'un ensemble
$-$	marque d'intervalle dans l'ensemble
$\wedge$	signifie un début de ligne s'il est en début d'expression et un complémentaire d'ensemble quand il suit immédiatement un [
$	signifie une fin de ligne s'il est en fin d'expression
\	placé devant un caractère spécial lui retire son sens particulier

Explication par l'exemple $[a-zA-Z0-9]$ est une expression régulière qui représente l'ensemble de tous les caractères alphanumériques (équivalente à [: *alnum* :]).
L'expression régulière $[\wedge a-zA-Z0-9]$ correspond à l'ensemble complémentaire du précédent.
$[0-9]$ est une expression régulière qui représente l'ensemble de tous les chiffres décimaux (équivalente à [: *digit* :]).
$-?(([0-9]+)|([0-9]*\.[0-9]+)([eE][-+]?[0-9]+)?)$ est une expression régulière représentant ("matchant") l'ensemble des nombres flottants en arithmétique machine.
$\wedge[0-9]\backslash\{2\backslash\}[a-z]*$[: *punct* :]$ est une expression régulière qui représente l'ensemble des lignes commençant par deux chiffres suivis de caractères de l'alphabet en minuscule et se terminant par un caractère de ponctuation.

7.2 L'éditeur ed

C'est l'ancêtre des éditeurs du système UNIX. Il est interactif et en mode ligne... raison pour laquelle il n'est que très rarement utilisé.

7.3 Le filtre d'édition sed

La commande *sed [-n] [-e textecmd] [-f ficcmd] f1 f2...* permet de recopier les fichiers passés en argument (ou stdin par défaut) sur stdout en les traitant en fonction du texte de requête passé "textecmd". Il est aussi possible de cumuler le texte de requêtes "textecmd". L'option -n supprime la sortie vers stdout. Un texte de requête contient une requête par ligne... il peut y en avoir de plusieurs types, la plus "utile" est celle de subsitution : *s/expression-regulière/remplacement/*
Exemple : ps -ef | sed -e s/root/racine/

7.4 L'éditeur vi

Très puissant, standard, plein-écran... mais peu ergonomique ! Il possède trois modes : déplacement, commande, insertion.
Avant de le lancer, il faut d'abord préciser au système quel type de terminal est employé[10] (exemple en ksh : TERM=vt100 vi nomDuFichier ou bien TERM=vt100 export ou encore TERM=vt100 ; export $TERM...)

Variables d'état permettent de modifier le comportement de l'éditeur.

:set all	permet de visualiser l'ensemble des variables d'état à un instant donnè
:set ai,:set noai	indentation automatique ou non des lignes à l'insertion
:set nu,:set nonu	numéros des lignes ou non
:set showmode	matérialisation du mode insertion
:ab,:unab	:ab bjr bonjour pour abréger la frappe de textes

Commandes générales

:w	sauver le tampon
:q	quitter vi
:e	éditer un nouveau fichier
:f nom	enregistrer sous le nom "nom"
:r nom	insère le fichier "nom" après la ligne courante

CTRL G permet d'obtenir des informations et CTRL L permet de rafraîchir l'écran. ZZ sauve et sort de vi.
Déplacements

[10] Pour le faire de façon permanente, ce genre d'instruction est à écrire dans les fichiers d'initalisation de session.

0	place en début de ligne courante
$	place en fin de ligne courante
n|	place en n-ième colonne
h	caractère vers la gauche
l	caractère vers la droite
j	ligne précédente
k	ligne suivante
w	mot suivant
M	première colonne milieu de fenêtre
G	fin de fichier

Passage en mode insertion

a	insère après le curseur
A	insère en fin de ligne
i	insère sous le curseur
I	insère avant le premier caractère (autre que l'espace) de la ligne
o	insère après la ligne courante
O	insère avant la ligne courante
r	remplace le caractère sous le curseur
R	remplace le caractère sous le curseur et les suivants

En mode insertion, ESC permet de passer en mode commande, CTRL W supprime le mot sous le curseur...

Suppression (mode commande)

x	efface le caractère sous le curseur
dw	efface le mot courant
dd	efface la ligne courante
d$	efface du curseur jusqu'en fin de ligne

Récupération

u	annule la dernière modification de ligne
U	rétablit la ligne courante dans son état
Y	copier la ligne dans le buffer
p	insérer le contenu du buffer après le caractère courant
P	insérer le contenu du buffer avant le caractère courant

8 Les commandes orientées réseau

Connexion permanente (cas d'un réseau local de type Ethernet[11] - 10 Mb), connexion temporaire (par téléphone et modem). Services offerts par une telle interconnexion de machines : transfert d'information (mail, talk, ftp...), partage de ressources ("remote login", "remote shell" pour lancement d'exécutable à distance...). Cette utilisation peut-être transparente à l'utilisateur en cas d'utilisation d'un système de fichiers répartis comme **NFS**, d'un spouleur d'impression répartie (protocole LPD).

[11] Le "bradcast" est possible avec Ethernet.

Les 7 couches de la norme OSI

Modèle théorique OSI	UNIX-Internet
couche applicative	rlogin, rcp, telnet, ftp...
couche de codage	XDR
couche session	RPC
couche transport	TCP-UDP
couche réseau (routage)	IP
liaison point à point	
couche physique	ETHERNET - X25,TRANSPAC

communiquer ⇒ mécanisme d'adressage et de redirection, IP définit les règles en ce domaine. IP associe une adresse logique[12] à chaque réseau, puis à chaque sous réseau du réseau... et enfin à chaque machine. Actuellement codage sur 32 bits. Exemple :

```
$ nslookup
Default Server:  ibp.ibp.fr
Address:  132.227.60.30

> iris
Server:  ibp.ibp.fr
Address:  132.227.60.30

Name:    iris.ibp.fr
Address:  132.227.61.18

> achille
Server:  ibp.ibp.fr
Address:  132.227.60.30

Name:    achille.ibp.fr
Address:  132.227.61.15

> cestac
Server:  ibp.ibp.fr
Address:  132.227.60.30

Name:    cestac.ibp.fr
Address:  132.227.61.12

>
$
```

Pour passer d'un sous réseau à un autre, il est nécessaire de passer par une succession de *gateway* (passerelles appartenant à au moins 2 réseaux). Il existe des tables de routage sur chaque système ce qui permet d'atteindre la passerelle du réseau local où se trouve notre machine. Le message doit-être fragmenté s'il est de trop grosse taille mais le protocole IP se charge de reconstituer le message de départ.
Sur ce protocole sont montées deux classes de transport correspondant aux deux protocoles UDP (User Datagram Protocol) et TCP (Transmission Control Protocol). L'en-tête des trames permet de distinguer le type de classe de protocole... il y a multiplexage au niveau de la couche IP.

[12] A ne pas confondre avec l'adresse physique type Ethernet par exemple...

Un service la couche application est identifié par le protocole de la couche transport dont il utilise les services et par un numéro de *port*.
Quelques services proposés : telnet (23,TCP), ftp (21,TCP), Trivial FTP (69,UDP), SMTP (Simple Mail Transfert Protocol,25,TCP), finger(79,TCP), rlogin (spécifique UNIX), rcp (spécifique UNIX), rsh (spécifique UNIX), talk (spécifique UNIX,517,UDP)...

Les domaines et espaces de nommage Le niveau 1 : .fr, .gov, .mil, .com... Le niveau 2 : .ibp, .jussieu...

Fichiers de configuration

```
$ m /etc/services
...
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp
...
# Projet emulateur ATM, DD 04/94
emulATM1        2100/tcp
emulATM2        2101/tcp
emulATM3        2102/tcp
emulATM4        2103/tcp
emulATM5        2104/tcp
...
$ m/etc/hosts
#------------------------------------------------------------------------
# Reseau  Ethernet  IBP  (TCP-IP)         1 reseau principal:  61
#           8.12.92                      18 sous reseaux    : 314
#    domaine  ibp.fr                                          375
#------------------------------------------------------------------------
...
# Reseau principal  ibp-net
#--------------------------
...
#  Sous reseau olympe-net         masi   (Recherche et Enseignement)
...
132.227.61.12   cestac          #sun
132.227.61.15   achille         #
132.227.61.18   iris            #sun
...
132.227.61.71   Gai-luron       #imprimante TCP/IP HP laserjet 4+
...
```

Il y a aussi le fichier /etc/networks.

Commande telnet

Commande	Effet
help	aide
open	open iris.ibp.fr demande de connexion sur iris
close	fermeture de connexion
quit	fermeture de connexion, fin de telnet
status	affichage des caractéristiques du telnet

Commande ftp

Commande	Effet
help	aide
ascii	transfert en ASCII (plus "rapide")
binary	transfert en binaire (plus prudent, par défaut)
open	open iris.ibp.fr demande de connexion sur iris
close,disconnect	fermeture de connexion
bye,quit	fermeture de connexion, fin de ftp
glob	(dés-)activation mécanisme d'expansion
status	affichage des caractéristiques du ftp
!cmd	exécute cmd en local (telle que ls)
put	*put reflocale refdistante*
mput	*mput reflocale...*
get	*get refdistante reflocale*
mget	*mget refdistante...*

Autres commandes ypwich donne le nom du serveur NIS s'il existe (un service NIS permet de disposer d'un même environnement sur un domaine type NFS ⇒ utiliser yppasswd).
uname : informations sur la machine locale
rwho, rlogin, rsh[13] (Bourne shell), rcp...

```
$ rsh iris.ibp.fr hostname > fic
$ more fic
iris
$ rm fic
$ rsh iris.ibp.fr hostname \> fic
$ more fic
fic: No such file or directory
```

Le fichier fic est placé en local dans le premier cas et sur la machine distante dans le second.

9 Le langage awk

Le fichier de données

```
$ cat bd
INF tom 12
INF tim 13
DRH tum 14
INF tam 01
$
```

[13] Pensez à bien configurer votre fichier $HOME/.rhosts !

Le code source du filtre

```
$ cat filtre.awk
BEGIN {
masomme=0;
}
$1=="INF" {masomme+=$3}
END {
printf("Somme des INF : %d\n",masomme);
}
$
```

La ligne de commande : requête sur la base de données

```
$ awk -f filtre.awk bd
Somme des INF : 26
$
```

Une liste de commandes "identiques" dans leur effet

cut -f1 -d' ' bd et *awk '{print $1}' bd*

ont toutes deux pour effet d'afficher sur la sortie standard la première colonne du fichier bd,

grep tom bd et *awk '/tom/ {print $0}' bd*

ont toutes deux pour effet d'afficher sur la sortie standard la (ou les) ligne contenant l'expression régulière passée en argument.

10 Le langage Lex - construction d'analyseurs lexicaux

10.1 Présentation

Lex et Yacc sont deux outils d'aide a la réalisation de programmes effectuant des transformations sur des entrées structurées. L'éventail de leurs applications possibles va du simple programme de recherche de motifs dans un texte jusqu'à l'écriture d'un compilateur...
L'analyse lexicale permet de produire des "tokens" (éléments significatifs du texte en entrée), c'est la tâche de Lex. L'analyse syntaxique est l'étape qui permet de relier ces "tokens" au moyen d'une grammaire, c'est la tâche de l'utilitaire Yacc.

10.2 Premier exemple

Contenu du premier fichier "monExemple1" :

```
%{
/* Voici la "Partie de Definition" (1ere partie)
   Comprise entre les deux "pourcents-crochets", elle
   va etre recopiee telle-quelle dans l'en-tete du code
   C genere par Lex.
   ATTENTION : respecter les tabulations pour eviter des
   bogues difficiles a retrouver.
   Cette partie permet d'introduire des header-files...
*/
```

```
%}

%% /* Partie des Regles : Forme Action */

[\t\n ] /* ignorer les blancs */ ;
[0-9]+ {printf("Nombre entier positif\n");}
-[0-9]+ {printf("Nombre entier negatif\n");}
[a-z]* {printf("Mot en minuscule\n");}
[A-Z]* {printf("Mot en majuscule\n");}

%% /* Partie des procedures utilisateur */

/* Cette partie sera recopiee a la suite du code C genere
   par Lex.
*/
main()
{
   printf("Ok pour le depart...\n");
   yylex();
}
```

Pour l'exécuter, procédons comme suit :

```
$ lex monExemple1
$ cc lex.yy.c -ll
$ a.out
Ok pour le depart...
12
Nombre entier positif
...
$
```

10.3 Ré-écrivons la commande "wc"

Contenu du fichier "monExemple2" :

```
%{
int nbCar = 0;
int nbMots = 0;
int nbLignes = 0;
/*
   Ce qui suit, place dans la partie des definitions
   correspond a des "substitutions"... Lex fournit
   en effet un mecanisme simple de substitution pour
   definir plus facilement des formes longues ou
   complexes.

   Dans la partie regles, Lex va substituer toute forme
   entre accolades par l'expression reguliere donnee
   dans la partie des definitions.

   Lex essaye toujours de satisfaire le critere (la forme)
```

```
   le plus long possible. Lorsqu'une chaine de caracteres
   verifie deux regles differentes, la premiere dans la
   donnee des specifications ci-dessous l'emporte.
*/
%}

mot [^ \t\n]+
fdl \n
%%
{mot} { nbMots++; nbCar += yyleng; }
{fdl} { nbCar++; nbLignes++; }
.     { nbCar++; }
%%
main()
{
    yylex();
    printf("%d %d %d\n",nbLignes,nbMots,nbCar);
}
```

Pour l'exécuter, procédons comme suit :

```
$
$ lex monExemple2
$ cc lex.yy.c -ll
$ ls | wc
      25      25     284
$ ls | a.out
25 25 284
$
```

References

[1] Brian Kerninghan & Rob Pike. *L'environnement de programmation UNIX*. Informatique Intelligence Artificielle. InterEditions, 1986.

[2] Steve Bourne. *Le système UNIX*. Informatique Intelligence Artificielle. InterEditions, 1985.

[3] Jean-Marie Rifflet. *La programmation sous UNIX - 3ème ed.* EDISCIENCE International, 1993.

[4] Marc Schaefer. Courte information sur le système unix et les rséaux téléinformatiques. schaefer@alphanet.ch, http://liawww.epfl.ch/ schaefer/unixguide.html, février 1995.

[5] Philippe Mailly & Yves Maurin. Initiation aux services offerts par les réseaux informatiques internationaux - partie 1. UPMC - UFR SNV, 1995.