GUIDE D'INTRODUCTION A L'AIX/6000

Numéro de document : zz01-0116-02

5 avril 1996

SEGI - ULG
Service Général d'Informatique
Campus Universitaire du Sart Tilman
Bâtiment B 26 - Parking 32
Sart Tilman - 4000 Liège
Tél. 041/66.49.04

Avant-propos

Du fait de son indépendance vis-à-vis des constructeurs d'ordinateurs et de sa portabilité, UNIX est en passe de devenir **le** système d'exploitation standard.

Les versions suivantes de UNIX jouent actuellement un rôle important sur le marché:

- UNIX SYSTEM V d'AT&T,
- Berkeley UNIX de l'université de Berkeley en Californie,

AT&T et Sun Microsystems ont entrepris une action pour résoudre les problèmes d'incompatibilité existant entre les versions de Berkeley et le System V avec pour objectif la création d'une version unique confondue. AT&T et Microsoft ont annoncé une stratégie identique pour XENIX. Ces projets démontrent clairement l'apparition de UNIX SYSTEM V comme standard de fait.

En parallèle, divers organismes et associations ont travaillé à l'élaboration de standards UNIX.

Les travaux de /usr/group, de X/Open, de l'ANSI, de l'ISO, de l'IEEE, avec le soutien d'AT&T pour UNIX SYSTEM V, convergent vers l'élaboration d'un standard UNIX portable commun connu sous le nom de POSIX (*Portable Operating System Standard for Computer Environment*).

L'AIX (*Advanced Interactive eXecutive*) quant à lui est un système d'exploitation IBM basé sur le UNIX SYSTEM V. Il intègre également certaines caractéristiques du UNIX de Berkeley (BSD, *Berkeley Software Distribution*). Il se conforme aux standards en vigueur de ISO, IEEE, FIPS (*Federal Information Processing Standard*), X/Open, OFS AES (*Open Software Foundation Application Environment Specification*), SVR4 (SYSTEM V Release 4, compromis entre SYSTEM V et BSD).

Le présent guide décrit l'AIX/6000 qui est installé sur les stations IBM RS/6000. Ces stations utilisent les processeurs RISC (*Reduced Instruction Set Computer*) dont le jeu d'instructions ne comprend que les instructions statistiquement utiles et qui peuvent s'exécuter en 1 cycle de base; les autres sont émulées par du code produit par les compilateurs. Ce guide s'adresse aux utilisateurs des machines IBM RS/6000 en général et plus particulièrement de l'IBM SP2 (*IBM 9076 Scalable POWERparallel System*) installé au SEGI. Le SP2 est une collection de processeurs RISC System/6000 connectés entre eux par un réseau local qui permet l'échange de données et la synchronisation des tâches. En plus d'un adaptateur ethernet, ce réseau comprend un *high-performance switch adapter*, HPS, qui offre une largeur de bande supérieure et un temps de latence réduit. En plus des possibilités de traitement habituel, cette machine offre la possibilité de calcul parallèle.

Ce guide est une introduction **pratique** à l'AIX/6000. Il doit permettre au débutant de se familiariser avec Unix et d'exploiter les possibilités de traitement offertes sur la gamme des machines RS/6000 gérées par le SEGI. L'utilisateur chevronné de Unix qui souhaite travailler sur les machines du SEGI devrait également trouver ici des informations utiles relatives aux particularités de l'AIX ou aux procédures et règles d'exploitations spéciales mises en oeuvre au SEGI.

Les ajouts et modifications par rapport à l'édition antérieure sont signalés par un trait vertical dans la marge.

Table des matières

1.0 Introduction

A l'ULg, la gestion journalière de chaque système AIX est prise en charge par un "Administrateur" dont la mission présente plusieurs volets:

- maintenance du système de base;
- gestion des comptes de travail;
- gestion de l'espace disque mis à la disposition des utilisateurs et prise en charge des opérations de "sauvegarde" des fichiers et répertoires implantés sur le système;
- premier niveau d'intervention pour tout problème d'exploitation que pourraient rencontrer les utilisateurs du système.

Le SEGI, quant à lui, assure cette même mission pour les machines installées dans ses locaux et intervient en "support" pour les machines qu'il gère "indirectement" comme celles installées à l'UNIPC.

Dans le cadre de sa mission de gestion du RESEAU, le SEGI sera également consulté pour tout problème d'intégration d'un de ces systèmes dans l'Interréseau de l'ULg.

Suit une liste d'agents du SEGI (fonction, nom, numéro téléphonique et adresse électronique) susceptibles de servir ou renseigner les utilisateurs ou administrateurs d'un environnement AIX/UNIX:

Fonction	Nom	Tél.	E-mail
Intégration de systèmes	Benedet	664901	benedet@segi.ulg.ac.be
Opérations et systèmes	Michel	664912	michel@segi.ulg.ac.be
Support Interréseau	Pirard	664932	pirard@segi.ulg.ac.be
Support systèmes AIX	Virgilii	664975	virgilii@segi.ulg.ac.be
Support connectivité UNIX	Minsoul	664977	minsoul@segi.ulg.ac.be
Support logiciels	Nihon	664978	nihon@segi.ulg.ac.be
Support opérations	Nissen	664929	control@segi.ulg.ac.be
Dispatching pour tout problème d'exploitation (réseau ULg inclus)	Debeck	664949	helpdesk@segi.ulg.ac.be

Tableau 1. Agents du SEGI

2.0 Structure de l'AIX

Les principaux composants logiciels qui constituent le système AIX (ou UNIX) sont les suivants:

- Le noyau : il accomplit les tâches de bas niveau en dialoguant directement avec le matériel. C'est le seul composant qui est dépendant du matériel; les autres éléments dialoguent avec le noyau. Ses principales fonctions sont les suivantes:
  - Gestion et sécurité des fichiers.
  - Services d'entrées/sorties.
  - Gestion de la mémoire.
  - Gestion et planification des processus.
  - Gestion des interruptions et erreurs.
  - Services de date et heure.
  - Comptabilité du système.
- Le shell : c'est un interpréteur de commandes. Il reçoit les commandes de l'utilisateur et organise leur exécution. Il comprend les fonctions essentielles suivantes :
  - Gestion du dialogue avec l'utilisateur.
  - Interpétation des commandes et appel des utilitaires appropriés pour leur exécution.
  - Contrôle de l'acheminement des entrées/sorties.
  - Gestion des communications entre programmes (par l'intermédiaire de tubes ou *pipes*).
  - Gestion de l'environnement de travail (par l'intermédiaire de variables).
  - Le shell est aussi un véritable langage de programmation qui permet de développer des procédures pour effectuer des opérations complexes.
- Les utilitaires : ce sont des programmes fournis avec le système et qui servent à diverses tâches telles que la copie de fichiers, l'édition de textes, ... Ces utilitaires sont les commandes du système. Les principales catégories d'utilitaires sont les suivantes :
  - Utilitaires de gestion des fichiers et des répertoires.
  - Utilitaires d'édition de documents, de programmes, ...
  - Utilitaires d'impression.
  - Utilitaires d'aide.
  - Utilitaires de programmation.
  - Utilitaires de sécurité.
  - Utilitaires de gestion et d'utilisation des réseaux.
  - Utilitaires d'administration du système.
- Interface des appels système : les utilitaires et programmes d'applications UNIX font appel aux services du noyau. Le mécanisme utilisé pour solliciter ces services est appelé *appel système*. Les appels système sont le seul moyen par lequel les applications et le noyau peuvent dialoguer directement.

Remarque : Dans les systèmes UNIX, les programmes développés par les utilisateurs se situent au même niveau que les utilitaires; ils s'utilisent comme des commandes.

3.0 Procédure de connexion et déconnexion

3.1 Login

Sur un terminal X, la procédure de chargement (*boot*) du terminal affiche la mire de la Figure 1.

```
DISPLAY = xst2:0                                        (1)
Enter host name for login (sp2s.ulg.ac.be):       ...(2)
sp2s.ulg.ac.be login: ...                               (3)
 ...'s Password : ...                                   (4)
```

Figure 1. Mire de login

- (1) Indique l'adresse du terminal (ici, dans une forme condensée; l'adresse complète est xst2.ulg.ac.be:0 - le 0 est le n° d'un système d'affichage ou *display*).
- Sur la ligne (2), on peut entrer l'adresse d'une station [1]. L'adresse indiquée entre parenthèses sera prise par défaut. C'est celle à partir de laquelle s'est effectuée la procédure de chargement (*boot*).
- Sur la ligne (3), on entre le *login name*.
- Sur la ligne (4), on entre le mot de passe.

A partir d'un autre terminal (en VM ou sur toute autre machine connectée au réseau Internet) on utilisera la commande **telnet** :

telnet adresse

qui fait apparaître le texte suivant sur l'écran ;

```
IBM AIX Version 3 for RISC System/6000
(C) Copyrights by IBM and by others 1982, 1990.
Login:
 ...'s Password:
```

Au terme de la procédure de login, le processus shell initial est lancé, les variables d'environnement sont initialisées à partir du fichier /etc/environment, le *profile* général /etc/profile et l'éventuel *profile* de l'utilisateur .profile sont exécutés et l'on se retrouve en session AIX avec l'indicatif du shell (généralement le $) sur l'écran.

Remarques :

1. La commande **telnet** peut aussi être utilisée dans une session AIX pour atteindre une autre machine.
2. Certaines stations gérées par le SEGI utilisent un gestionnaire d'écran appelé *xdm*. Dans ce cas, la procédure de démarrage est sensiblement différente. Pour plus de détails, voir «Démarrage controlé par xdm» à la page 76.

3.2 Logout

On quitte une session AIX soit

- en tapant la commande **logout**;
- en tapant ctrl-d (*End of file*) quand le processus shell initial est en attente d'une entrée au terminal - sauf si l'option suivante a été définie : **set -o ignoreeof** (voir la commande **set** au paragraphe «Commandes spéciales» à la page 33).

---

[1] Voir «Procédures d'exploitation» à la page 95. Si l'on entre une adresse autre que celle de la station de chargement, il faudra définir l'adresse suivante:

DISPLAY=adresse du terminal suivie de deux-points (:) et du n° de système d'affichage, généralement 0 (dans l'exemple ci-dessus : xst2.ulg.ac.be:0) et exporter la variable DISPLAY. Voir «Environnement» à la page 25 et «Processus» à la page 27.

4.0 File System

Un *file system* AIX est une structure hiérarchique de fichiers et de répertoires.

Les répertoires sont utilisés pour organiser les fichiers en groupes. Chaque répertoire peut contenir des fichiers et des sous-répertoires. Le répertoire qui contient tous les autres est le répertoire principal ou racine.

Un *file system* réside sur un volume logique qui peut se répartir sur un ou plusieurs volumes physiques. Plusieurs *file systems* peuvent être combinés dans une structure unique pour être vus comme un seul et nouveau *file system*.

En outre, le produit NFS (*Network File System*) permet de partager des *file systems* entre différentes machines. L'intérêt est qu'un utilisateur qui possède un compte sur différentes machines peut retrouver les mêmes fichiers sur chacune d'elles alors que ces fichiers ne sont physiquement présents que sur une seule de ces machines.

4.1 Fichiers

Un fichier est identifié par un nom qui peut comprendre jusque 255 caractères, lettres, chiffres, ... Le caractère / est interdit dans un nom de fichier. Les caractères suivants sont tolérés mais il est préférable de les éviter : \ " ' ; - { } ( ) < > ~ ! $ # @ & |

Le shell attribue aux caractères suivants, appelés métacaractères ou caractères génériques, un rôle particulier dans les noms de fichiers :

? représente un caractère quelconque;

[akz] représente l'un des caractères a, k ou z;

[x-z] représente l'un des caractères x, y ou z; on peut combiner les deux notations précédentes; exemple : [akx-z];

* représente un nombre quelconque de caractères quelconques.

Remarques :

1. L'utilisation de *patterns* avec caractères génériques n'est autorisée que pour désigner des fichiers existants.

2. Il existe des fichiers cachés. Ce sont des fichiers dont le nom commence par un point; on les appelle aussi *dot files*. Ils sont dits cachés car ils n'apparaissent normalement pas dans les listes fournies par les commandes de listage des répertoires (**li**, **ls** - sauf si on le demande explicitement au moyen d'une option). Ce sont en général des fichiers qui contiennent des commandes à exécuter automatiquement lors de la procédure de démarrage (fichier .profile ou fichiers de configuration de logiciels, par exemple). Pour que ces fichiers soient pris en considération par un *pattern* qui utilise des caractères génériques, le point initial doit être explicitement spécifié dans le *pattern*.

4.2 Répertoires

Le répertoire est le moyen d'organiser des fichiers en sous-ensembles distincts dans une structure hiérarchique. Un répertoire est un fichier spécial qui contient des références à des fichiers d'un même sous-ensemble. Ces fichiers peuvent être des fichiers proprement-dits ou des sous-répertoires. Par abus de langage, on utilise indifféremment le terme répertoire ou sous-répertoire pour désigner un sous-ensemble de fichiers aussi bien que le fichier spécial qui contient les références à ce sous-ensemble.

Comme un répertoire est un fichier, son nom se forme comme celui d'un fichier, sauf le répertoire principal qui s'appelle /.

La structure hiérarchique d'un *file system* est représentée dans la Figure 2.

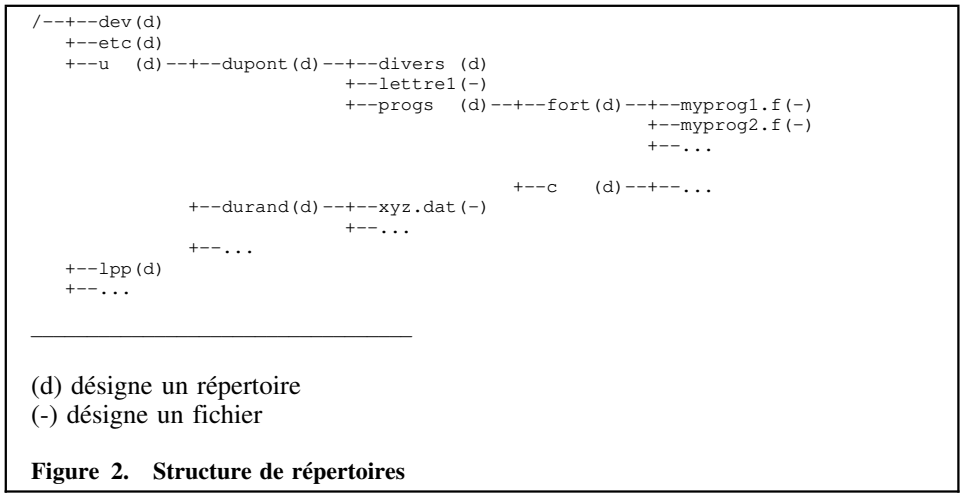

Figure 2. Structure de répertoires

Répertoires particuliers :

- *home directory* ou *login directory* ou répertoire principal de l'utilisateur : c'est le répertoire défini par l'administrateur du système pour chaque utilisateur. C'est dans ce répertoire que l'utilisateur peut créer des fichiers et de nouveaux répertoires.

- répertoire en cours ou de travail : il est possible de passer d'un répertoire à l'autre. Le répertoire en cours est celui dans lequel on travaille à un moment donné. Au terme de la procédure de démarrage, le répertoire en cours est le *home directory*.

4.3 Chemin d'accès

Le chemin d'accès d'un fichier est la liste des noms de répertoires qui mènent de la racine au fichier. Les noms de répertoires formant le chemin d'accès sont séparés par des barres obliques (/) et la racine est aussi désignée par une barre obliqe. Exemple:

```
/u/dupont/progs/fort
```

est le chemin d'accès qui mène, par exemple, au fichier myprog2.f dans la structure de la Figure 2 à la page 7. Pour désigner ce fichier sans ambiguïté on écrira

```
/u/dupont/progs/fort/myprog2.f
```

Le nom du chemin d'accès est donc aussi séparé du nom du fichier par une barre oblique. Un chemin d'accès peut se désigner de deux façons :

- par un nom absolu : liste de noms de répertoires qui définissent un chemin à partir de la racine (voir exemple précédent);
- par un nom relatif : liste de noms de répertoires qui définissent un chemin à partir du répertoire en cours; ce dernier se désigne par un point (.) et son répertoire-père par deux points consécutifs (..).

Exemples : Si le répertoire en cours est /u/dupont (voir Figure 2 à la page 7) :

1.

```
./progs/fort/myprog2.f
```

désigne le même fichier que précédemment.

2.

```
../durand/xyz.dat
```

désigne le fichier /u/durand/xyz.dat

3.

```
./lettre1
```

désigne le fichier /u/dupont/lettre1; de même que

```
lettre1
```

En effet, quand aucun chemin d'accès n'est spécifié pour désigner un fichier ou un répertoire, c'est le répertoire courant qui est pris en considération. Le "./" est sous-entendu dans les chemins d'accès qui ne commencent pas par "/".

4.4 Liens

Un fichier a quatre composants : un numéro (*i-number*, numéro unique par lequel le fichier est identifié au niveau du *file system*), un nom, un contenu et des informations administratives (adresse, taille, propriétaire, droits d'accès, type, dates de création, de dernier accès et de dernière modification). Les informations administratives sont mémorisées dans un emplacement spécial appelé *inode*. Le *inode* est identifié par le *i-number*.

Un répertoire est un fichier qui contient une table de correspondances entre des noms de fichiers et des *i-numbers*. Chaque entrée dans un répertoire est donc un **lien** entre un nom de fichier et un *i-number*.

Cette organisation permet d'attribuer plusieurs noms différents au même fichier ou d'introduire des références au même fichier dans des répertoires différents. C'est la commande **ln** qui permet d'effectuer de telles opérations (voir «Gestion de fichiers» à la page 10).

5.0 Principales commandes

5.1 Syntaxe

Pour la description des commandes, les conventions suivantes sont utilisées :

- Les noms de commande sont en caractères gras.
- Les éléments optionnels sont entre crochets.
- {élément1 | élément2} indique qu'il faut choisir un des deux éléments.
- Les abréviations suivantes sont utilisées :
  - dir : nom de répertoire.
  - file : nom de fichier.
  - dfile : nom de fichier éventuellement précédé de l'indication de son répertoire.
  - gfile : nom de fichier qui peut contenir les caractères génériques *, ?, [xyz] [x-z].
  - dgfile : nom de fichier avec caractères génériques éventuels et éventuellement précédé de l'indication du répertoire.
  - les mêmes abréviations au pluriel (dirs, files, dfiles, ...) désignent une liste de noms de répertoires ou fichiers séparés les uns des autres par un espace au moins.
  - stdin : *standard input file*.
  - stdout : *standard output file*.
  - stderr : *standard error file*.

Par défaut, stdin, stdout et stderr sont affectés au terminal (pour des détails, voir «Redirection des entrées sorties»).

La forme la plus répandue de commande est la suivante :

commande [options] [paramètres]

Les options sont précédées du signe **-** ou **+**. Plusieurs options d'une même commande peuvent généralement être regroupées; per exemple, -a -b -c peut s'écrire -abc.

On peut écrire plusieurs commandes sur la même ligne en les séparant par point-virgule (;). Une commande peut se prolonger sur plusieurs lignes à condition de terminer chaque ligne qui a une suite par une barre oblique inversée (\). Exemple :

```
commande1; commande2; début de la commande 3 \
suite de la commande 3
```

Remarques :

1. Dans ce guide, seules les principales options des commandes sont décrites. Une description complète de chaque commande peut être trouvée dans la documentation enregistrée (voir «Help» à la page 21) ou écrite (voir «Annexe A. Bibliographie» à la page 99).

2. Les combinaisons suivantes de touches ont une signification spéciale :
   - *ctrl-c* permet d'interrompre une commande;
   - *ctrl-d* est interprété comme fin de fichier par le shell aussi bien que par les commandes exécutées par le shell. Lorsque le processus shell initial est en attente d'une entrée au terminal, *ctrl-d* équivaut à **logout**[2] .

3. **UNIX est "case sensitive", il distingue les majuscules et les minuscules dans les noms de commandes, de fichiers, d'utilisateurs, dans les mots de passe, ...**

5.2 Redirection des entrées sorties

La plupart des commandes envoient leurs résultats vers un fichier appelé *standard output file* (en abrégé, stdout) et leurs messages d'erreur vers le *standard error file* (en abrégé, stderr). Certaines commandes lisent des données à partir d'un *standard input file* (en abrégé, stdin). Par défaut, les fichiers stdout, stderr, stdin sont affectés au terminal.

Ainsi, par exemple, la commande **cat**, sans aucun paramètre, lit des données sur stdin, le terminal, et les recopie sur stdout, le terminal, jusqu'à ce que la fin de fichier soit atteinte sur stdin. Au terminal la fin de fichier est simulée en appuyant sur ctrl-d.

Il est possible de modifier l'affectation des fichiers standards et donc de rediriger les entrées, sorties et messages d'erreur d'une commande telle que **cat** par exemple. Cela se fait au moyen des opérateurs de redirection suivants :

< file - affecte stdin à file;

> file - affecte stdout à file qui sera ouvert en *output mode*;

>> file - affecte stdout à file qui sera ouvert en *append mode*;

2> file - affecte stderr à file qui sera ouvert en *output mode*;

2>> file - affecte stderr à file qui sera ouvert en *append mode*;

---

[2] Sauf si l'option suivante a été définie : **set -o ignoreeof**. Voir la commande **set** au paragraphe «Commandes spéciales» à la page 33.

<< 'string' - définit stdin comme un fichier qui contient les lignes qui suivent directement la commande jusqu'à la rencontre d'une ligne qui ne contient que *string*.

Exemples :

1.

```
cat > newfile
```

lit des données au terminal et les écrit dans le fichier newfile jusqu'à ce que l'utilisateur tape ctrl-d.

2.

```
cat < oldfile
```

lit le fichier oldfile et l'écrit au terminal.

3.

```
cat < oldfile >> newfile 2> errfile
```

lit le fichier oldfile, l'écrit dans newfile à la suite du contenu initial et écrit éventuellement des messages d'erreur dans errfile.

4.

```
cat << 'End' > otherfile
123
421
675
End
```

crée le fichier otherfile, de 3 lignes, contenant respectivement 123, 421 et 675.

5.3 Filtres, tubes et pipelines

Un filtre est un programme qui lit des données sur stdin, les traite et produit les résultats sur stdout.

Le tube (*pipe*) est le moyen de connecter le stdout d'un programme avec le stdin d'un autre.

Un *pipeline* est la connexion de deux ou plusieurs programmes au moyen de tubes; c'est la combinaison de deux ou plusieurs filtres où la sortie standard de l'un devient l'entrée standard du suivant. Les différents programmes d'un *pipeline* s'exécutent simultanément.

L'opérateur de tube est la barre verticale (|).

Exemple :

```
who | sort
```

La commande **who** écrit la liste des utilisateurs actifs sur son stdout; l'opérateur | connecte le stdout de **who** au stdin de **sort**; **sort** lit son stdin, trie les lignes et écrit le résultat sur son stdout, ici, l'écran du terminal..

Remarque : La commande **tee** peut être utilisée dans un *pipeline* pour sauvegarder les résultats obtenus à l'une des étapes intermédiaires du *pipeline*. Exemple :

```
who | sort | tee save.data | lpr
```

La commande **tee** lit les données produites par **sort** et les copie dans le fichier save.data en même temps qu'elle les transmet à la commande d'impression **lpr**.

5.4 Gestion de fichiers

Afficher le contenu de fichiers :

more [options] [dgfiles]

more lit les fichiers indiqués ou, à défaut, stdin, affiche le contenu à l'écran et s'arrête quand celui-ci est rempli en affichant "more" au bas de l'écran. Pousser sur RETURN provoque l'affichage de la ligne suivante; poussez sur la barre d'espacement provoque l'affichage de la page d'écran suivante; taper b provoque l'affichage de la page d'écran précédente.

Lister les noms de fichiers :

ls [options] [dgfiles] [dirs]

ls donne les noms des fichiers et/ou le contenu des répertoires désignés sur le stdout. En l'absence de paramètre, la commande s'applique au répertoire en cours.

Principales options :

- -l : donne les caractéristiques de chaque fichier en plus de son nom.
- -a : donne aussi les noms des fichiers cachés.
- -t : les fichiers sont triés dans l'ordre décroissant des dates de dernière mise à jour (plutôt que dans l'ordre des noms).
- -R : donne aussi le contenu des sous-répertoires de tous niveaux (parcours récursif).

Exemple :

```
ls -Ral | more
```

Affiche, avec un arrêt à chaque page d'écran, le contenu du répertoire en cours et de ses sous-répertoires (option -R), fichiers cachés compris (option -a) et dans le format long (option -l).

Le résultat de la commande précédente se présente comme suit :

```
total 9584
drwxr-xr-x   3 nihon    staff       1536 25 sep 15:00 .
drwxr-xr-x  12 nihon    staff       1536 25 sep 14:17 ..
-rw-r--r--   1 nihon    staff       5365 25 sep 11:33 .index
-rw-r--r--   1 nihon    staff          0 27 nov 1991  fort.7
-rw-r--r--   1 nihon    staff          0 19 dec 1991  fort.9
drwxr-xr-x   2 nihon    staff        512 25 sep 15:01 fort.prog
-rw-r--r--   1 nihon    staff       6293 08 nov 1991  gantt.lst
-rw-r--r--   1 nihon    staff       3682 08 nov 1991  gantt.o
-rwxr-xr-x   1 nihon    staff      38787 08 nov 1991  gantt.out
 ...

./fort.prog:
total 104
drwxr-xr-x   2 nihon    staff        512 25 sep 15:01 .
drwxr-xr-x   3 nihon    staff       1536 25 sep 15:00 ..
-rw-r--r--   1 nihon    staff       1163 25 sep 15:01 gantt.f
-rw-r--r--   1 nihon    staff       3628 25 sep 15:01 mybitmap.f
-rw-r--r--   1 nihon    staff       1858 25 sep 15:01 mycgi.f
 ...
 (1)        (2) (3)      (4)         (5)  (6)    (7)   (8)
```

- total ... indique le nombre de blocs de 1K octets occupés par le répertoire mais ne tient pas compte des éventuels sous-répertoires.
- (1) drwxr-xr-x :
  - le premier caractère indique le type de fichier : d = répertoire; - = fichier ordinaire (voir la documentation pour les autres indicateurs).
  - les 9 caractères suivants représentent les droits d'accès; les 3 premiers concernent le propriétaire du fichier; les 3 suivants, le groupe auquel il appartient; les 3 derniers, les autres utilisateurs. Les droits d'accès sont représentés par des codes dont les principaux sont :

```
Pour un fichier     répertoire

  r     lecture     lecture du contenu
  w     écriture    création et suppression de fichiers
  x     exécution   autorisation d'en faire le répertoire courant
  -     aucun droit
```

  Pour la façon d'attribuer les droits d'accès, voir «Sécurité au niveau des fichiers» à la page 19.

- (2) nombre de liens de type *hard link* (voir ci-dessous la commande **ln**) vers le fichier.
- (3) *login name* du propriétaire du fichier.
- (4) nom du groupe auquel appartient le fichier.
- (5) taille du fichier en octets.
- (6) et (7) date et heure de dernière mise à jour
- (8) nom du fichier. Les fichiers dont le nom commence par un point sont des fichiers cachés.

Connaître le type des fichiers :

file dgfiles

file analyse les fichiers indiqués pour essayer de découvrir le type de données qu'ils contiennent et écrit les renseignements sur stdout. *dgfiles* indique les fichiers à analyser.

Exemple :

```
file *
```

Cette commande donne la liste des fichiers du répertoire en cours et leur type sous la forme suivante :

```
Mail:            directory
PostScript.ps:   ascii text
Power.dt:        ascii text
bin:             directory
calendar:        data or International Language text
from_nihon:      ascii text
gaix:            empty
mydoc:           troff, tbl, or eqn input text with garbage
mytest:          commands text
test1.c:         c program text
test1.lst:       ascii text
test1.out:       executable
```

Copier des fichiers :

cp [options] dfile1 dfile2 **cp** [options] dgfiles dir **cp** [options] dir1 dir2

cp copie le(s) fichier(s) désigné(s) par le premier paramètre vers une destination définie par le second paramètre.

Exemples :

1.

```
cp fich1.dat ../donnees
```

copie fich1.dat du répertoire en cours dans le fichier donnees du répertoire père.

2.

```
cp $HOME/*.f .
```

copie les fichiers *.f du répertoire principal de l'utilisateur actuel dans le répertoire en cours. $HOME (ou HOME) est une variable du shell (voir «Environnement» à la page 25) qui contient le chemin d'accès au répertoire principal de l'utilisateur.

3.

```
cp -r /u/dupont .
```

copie tous les fichiers et sous-répertoires (option -r) du répertoire /u/dupont dans le répertoire en cours.

Déplacer des fichiers :

mv [options] dfile1 dfile2
mv [options] dgfiles dir
mv [options] dirs1 dir2

mv déplace, en le(s) renommant éventuellement, le(s) le(s) fichier(s) ou répertoire(s) désigné(s) par le premier paramètre vers une destination définie par le second paramètre. Si source et destination sont dans le même répertoire, il y a simplement échange de noms.

Remarque : Sans l'option -i, mv remplace par le fichier destination tout fichier existant qui porterait le même nom.

Exemples :

1.

```
mv -i ../donnees ../stat.data
```

renomme ../donnees en ../stat.data

2.

```
mv -i ../prog1.f stat.f
```

déplace le fichier prog1.f du répertoire père vers le répertoire courant en le renommant stat.f.

3.

```
mv -i *.c *.out ./newrep
```

déplace tous les fichiers *.c et *.out du répertoire courant vers le répertoire ./newrep en conservant leurs noms.

Créer des liens :

La commande **ln** permet de créer deux types de liens :

- *hard link* : lien qui associe un nouveau nom de fichier à un *i-number* (revoir «Liens» à la page 8). Un tel lien ne peut s'appliquer qu'à un fichier, pas à un répertoire. De plus il ne peut être défini qu'à l'intérieur du *file system* auquel le fichier appartient, puisque le *i-number* est unique au niveau du *file system*.
- *soft link* ou *symbolic link* : lien qui associe un nouveau nom de fichier ou de répertoire respectivement à un nom de fichier ou de répertoire existant. Ce type de lien n'est pas soumis aux restrictions du type précédent. Toutefois, les liens symboliques doivent être manipulés avec précaution. En effet, un lien symbolique peut continuer à exister même quand le fichier original a été supprimé.

ln [options] dfile1 dfile2
ln [options] dgfiles dir

dfile1 désigne un nom de fichier existant; *dfile2* désigne un nouveau nom pour ce fichier; ces deux noms peuvent être dans le même répertoire ou dans des répertoires différents.

dgfiles désignent une série de noms de fichiers existants; *dir* un répertoire dans lequel ces noms seront introduits avec des références vers les fichiers correspondants.

Exemple : l'utilisateur durand veut créer, dans son répertoire en cours, un lien vers le fichier /progs/fort/myprog2.f de dupont :

```
ln /u/dupont/progs/fort/myprog2.f herprog.f
```

Le lien ainsi créé est un *hard link*. C'est l'option **-s** qui permet de créer un lien symbolique. Pour un lien symbolique, *dfile1* et *dfile2* peuvent être des noms de répertoires. Exemple : l'utilisateur dupont veut créer, dans son répertoire principal /u/dupont, un lien mypvm qui pointe directement vers son répertoire /u/dupont/pvm3/bin/RS6K :

```
ln -s /u/dupont/pvm3/bin/RS6K /u/dupont/mypvm
```

Supprimer des fichiers :

rm [options] dgfiles
rm [options] dir

rm supprime les liens, fichiers ou répertoires désignés. Un fichier n'est effectivement supprimé que lorsque l'on supprime son dernier *hard link*.

Exemple :

```
rm -r ./dupont
```

Efface le répertoire ./dupont et son contenu (liens et/ou fichiers) et sous-répertoires (option -r), du répertoire en cours.

Concaténer des fichiers :

cat [options] [dgfiles]

cat lit les dgfiles ou, à défaut, stdin et les recopie l'un après l'autre sur stdout.

Exemples :

1.

```
cat fich1.dat > ../donnees
```

copie fich1.dat du répertoire courant dans le fichier donnees du répertoire père.

2.

```
cat ../donnees
```

affiche le contenu de ../donnees au terminal.

3.

```
cat > fich2
```

copie les lignes entrées au terminal dans fich2 jusqu'à ce que l'utilisateur tape ctrl-d (eof).

4.

```
cat myfich[267] > fichtout
```

copie, parmi myfich2, myfich6 et myfich7, les fichiers qui existent en les concaténant dans fichtout.

Afficher le contenu de fichiers en hexadécimal ... :

od [options] [dgfiles]

od lit les dgfiles ou, à défaut, stdin et les recopie sur stdout

en octal si l'option -o est spécifiée (c'est l'option par défaut);
en hexadécimal si l'option -x est spécifiée;
en caractères si l'option -c est spécifiée.

Plusieurs options peuvent être combinées. Exemple :

```
od -cx fichtout
```

5.5 Gestion des répertoires

Lister les répertoires :

li [options] [dgfiles] [dirs]

li est comparable à ls (voir «Gestion de fichiers» à la page 10) sauf pour certaines options. **li** comprend notamment l'option -O qui permet de sélectionner le type des fichiers.

Exemple :

```
li -l -Od /u/dupont
```

affiche la liste des sous-répertoires (option -Od) de /u/dupont dans le format long (option -l).

Afficher le nom du répertoire en cours :

pwd

pwd écrit le nom du répertoire en cours sur stdout.

Changer de répertoire en cours :

cd [dir]

cd permet de changer le répertoire en cours. Si dir n'est pas spécifié, **cd** renvoie au *home directory*.

Créer un nouveau répertoire :

mkdir dir

mkdir permet de créer un nouveau répertoire. Exemples :

```
mkdir tests.prog
```

crée le répertoire tests.prog dans le répertoire en cours.

```
mkdir /u/dupont/progs/cobol
```

crée le répertoire cobol dans le répertoire /u/dupont/progs.

Supprimer un répertoire :

rmdir dir

rmdir supprime le répertoire indiqué. Celui-ci doit être vide.

5.6 Espace disque

Les commandes suivantes permettent de vérifier l'espace disque disponible et occupé.

du [options] [dgfiles] [dirs]

du indique, sur stdout, l'espace occupé par les fichiers et/ou répertoires indiqués. Cet espace est indiqué en blocs de 512 octets (ou 1024 si l'option -k est spécifiée). Option utile : -a pour obtenir aussi le renseignement pour chaque fichier ou répertoire individuellement. Exemple :

```
du -k
```

df [options] [FileSystems]

df écrit, sur stdout, des informations sur l'espace occupé par des *file systems*. Un *file system* peut être désigné soit par le nom du *device* (volume logique) sur lequel il réside, soit par le nom d'un fichier ou d'un répertoire qui en fait partie. Exemple :

```
df .
```

donne des renseignements sur l'espace occupé par le *file system* qui comprend le répertoire en cours. Si aucun *file system* n'est indiqué, ces renseignements sont donnés pour tous les *file systems* accessibles.

Le résultat de la commande précédente se présente comme suit :

```
Filesystem    Total KB    free %used   iused %iused Mounted on
/dev/hd1         94208    6784   92%    2982    12% /home
```

iused et %iused représentent le nombre et le pourcentage d'inodes utilisés.

5.7 Recherche de fichiers

find dirs critere action

find permet de sélectionner des fichiers ou répertoires et de leur appliquer une action. *dirs* désigne les répertoires qui seront parcourus récursivement (c'est-à-dire sous-répertoires compris) pour faire la recherche; *critere* définit le critère de sélection; *action* définit l'action à appliquer.

Exemple de critères :

-user 'propriétaire' : fichiers/répertoires qui appartiennent au propriétaire indiqué.

-group 'nom_de_groupe' : fichiers/répertoires qui appartiennent au groupe indiqué.

-name 'gfile' : fichiers/répertoires dont le nom est défini par gfile.

-newer 'dfile' : fichiers/répertoires qui ont été mis à jour plus récemment que le fichier indiqué.

-mtime n | +n | -n : fichiers/répertoires dont la date de mise à jour est séparée de la date actuelle par un nombre de jours égal (n), supérieur (+n) ou inférieur (-n) à n.

Un critère peut être précédé de l'opérateur logique de négation **!**. Il est possible de spécifier plusieurs critères en les combinant avec les opérateur logiques -a (AND) ou -o (OR).

Exemples d'action :

-print : écriture sur stdout.

-exec commande {} \; : exécution de la commande indiquée. La commande doit être terminée par \; {} représente le paramètre de la commande; il sera remplacé successivement par les noms des fichiers sélectionnés.

Exemples :

1.

```
find / -name 'disspla*' -print 2> /dev/null
```

recherche, dans le répertoire principal du *file system* et dans tous ses sous-répertoires, les fichiers dont le nom commence par displa, les affiche à l'écran (action -print) et redirige les éventuels messages d'erreur vers le *device* /dev/null. Ce dernier est un *device* qui ignore les données qui lui sont envoyées. Un message d'erreur est envoyé chaque fois que la commande **find** analyse un répertoire pour lequel l'utilisateur ne bénéficie pas du droit d'accès en exécution (x).

2.

```
find . ! -user 'dupont' -print
```

affiche la liste des fichiers du répertoire en cours et de ses sous-répertoires qui n'appartiennent pas à dupont.

3.

```
find . -name '*.o' -a -mtime +360 -exec rm {} \;
```

efface les fichiers, du répertoire en cours et de ses sous-répertoires, dont le nom se termine par .o et dont la dernière mise à jour remonte à plus de 360 jours.

5.8 Impression

5.8.1 Imprimantes disponibles et état des files d'attente

lpstat

lpstat donne, sur stdout, la liste des imprimantes disponibles (locales et *remotes*) ainsi que la liste des fichiers en attente ou en cours d'impression :

```
Queue   Dev   Status    Job Files              User         PP %   Blks  Cp Rnk
------- ----- --------- --- ------------------ ---------- ---- -- ----- --- ---
B26Aasc i4019 READY
B26Agl  i4019 READY
B26Apcl i4019 READY
B26Aps  i4019 RUNNING   403 bench.linpack      nihon         0 70   122   1   1
B26Bpcl i4039 READY
B26Bps  i4039 READY
B26Basc i4039 READY
B26Blan i4039 READY
bsh     bshde DOWN
ulis    ulisd READY
vmlocal 3800  READY
vmlocal local QSTATUS
apple   lp    READY
```

Les principaux renseignements fournis sont le nom de la file d'attente (queue), le *device* (Dev, par exemple i4019 pour une imprimante IBM 4019), l'état de le file d'attente (Status), le n° de job (Job), le nom du fichier (Files), le nom du propriétaire (User).

L'exemple précédent montre que l'imprimante i4019 est accessible via 4 files d'attente. Chacune de ces files d'attente a été définie pour accueillir des fichiers d'un type particulier :

- B26Aasc, file d'attente pour des fichiers ASCII;
- B26Agl, file d'attente pour des fichiers HPGL;
- B26Apcl, file d'attente pour des fichiers Laserjet II;
- B26Aps, file d'attente pour des fichiers PostScript.

Il faut remarquer que les noms des files d'attente d'impression disponibles sur un système sont fixées par l'administrateur de ce dernier. Il en est de même pour les règles d'exploitation des imprimantes. L'utilisation des imprimantes "visibles" sur un système impliquera donc bien souvent un contact préalable avec l'administrateur concerné.

5.8.2 Impression

Imprimer :

lpr [options] [dgfiles]

lpr lit les dgfiles (ou, à défaut, stdin jusqu'à ce que l'utilisateur tape ctrl-d) et les envoie dans une file d'attente d'imprimante.

Principales options :

- -P queue[:device] : indique la file d'attente et, éventuellement le *device*. Par défaut, file d'attente et *device* sont définis comme suit :
  - valeur de la variable d'environnement LPDEST si elle existe (voir «Environnement» à la page 25);
  - sinon, valeur de la variable d'environnement PRINTER si elle existe (voir «Environnement» à la page 25);
  - sinon file d'attente et *device* indiqués dans la première ligne de la sortie produite par la commande **lpstat** (voir «Imprimantes disponibles et état des files d'attente»).
- -m : pour recevoir un message électronique quand l'impression sera terminée, message consultable par **mail**.
- -#n : n est un nombre entier qui représente le nombre de copies demandées. Défaut : 1.

Exemple :

```
lpr -P B26Bps:i4039 mygraph.ps
```

imprime le fichier PostScript mygraph.ps sur l'imprimante i4039. Dans le cas présent, la commande suivante aurait le même effet

```
lpr -P B26Bps mygraph.ps
```

puisque, la file d'attente ps ne comprend qu'une imprimante, lp0 (voir sortie de la commande lpstat, «Imprimantes disponibles et état des files d'attente»).

Vérifier l'état de fichiers dans une file d'attente :

lpq [options] [username] [jobnumer]

lpq écrit sur stdout des renseignements sur l'état de fichiers dans une file d'attente d'imprimante.

Option utile : -P queue[:device] (mêmes signification et défaut que pour la commande **lpr**).

username et *jobnumber* permettent d'indiquer pour quels fichiers on désire obtenir ces renseignements. Ceux-ci sont comparables à ceux donnés par la commande **lpstat** (voir «Imprimantes disponibles et état des files d'attente» à la page 15).

Supprimer des fichiers dans une file d'attente :

lprm [options] [username] [jobnumer]

lprm permet de supprimer des fichiers dans une file d'attente d'imprimante.

Option utile : -P queue[:device] (mêmes signification et défaut que pour la commande **lpr**).

username et *jobnumber* permettent d'indiquer quels fichiers on désire supprimer.

Remarque : La commande **pr** peut être utilisée, en conjonction avec la commande **lpr**, pour préparer la mise en page d'un fichier avant de l'imprimer (voir «Annexe A. Bibliographie» à la page 99).

5.9 Archivage de fichiers

La commande **tar** permet d'archiver des fichiers, de consulter les archives et de récupérer des fichiers archivés.

Archiver des fichiers :

tar [options] {**-c**|**-r**|**-u**} **-f** archive [dgfiles] [dirs]

Les *dgfiles* et/ou le contenu des répertoires (*dirs*) indiqués sont archivés.

Options :

- -c : crée une nouvelle archive. Le cas échéant, celle qui existait est détruite.
- -r : ajoute les nouveaux fichiers à la fin de l'archive indiquée. Pas possible sur cassette.
- -u : ajoute les nouveaux fichiers à la fin de l'archive indiquée seulement s'ils n'y sont pas encore ou s'ils ont été modifiés depuis le dernier archivage. Pas possible sur cassette.
- -f archive : définit l'endroit où les fichiers doivent être archivés. *archive* peut être
  - un *device* tel que
    /dev/rfd*n* (où *n* est un nombre entier 0, 1, 2, 3 ...) pour désigner un lecteur de disquettes;
    /dev/rmt*n* (où *n* est un nombre entier 0, 1, 2, 3 ...) pour désigner un lecteur de cassettes;
  - ou un fichier classique.
- Autre option utile : -v, qui liste les noms des fichiers lorsqu'ils sont traités.

Exemples :

1. Archiver les fichiers myprog.*.c sur une cassette placée dans le lecteur 0 :

   ```
   tar -cvf /dev/rmt0 myprog.*.c
   ```

2. Regrouper tous les fichiers du répertoire ./chimie dans un seul fichier nommé chimie.tar (par exemple, pour le transférer en une seule opération via le réseau; dans ce cas, il pourrait être utile de compacter le fichier avant de l'envoyer - voir «Compactage de fichiers» à la page 17) :

   ```
   tar -cvf chimie.tar ./chimie
   ```

3. Ajouter le fichier chimie.news dans l'archive précédente :

   ```
   tar -rvf chimie.tar chimie.news
   ```

Consulter les archives :

tar -tf archive

Cette commande donne la liste des fichiers contenus dans l'archive indiquée.

Exemple : voir le contenu du fichier d'archive chimie.tar :

```
tar -tf chimie.tar
```

Récupérer des archives :

tar [options] **-xf** archive [dgfiles] [dirs]

Les *dgfiles* et/ou *dirs* indiqués sont recopiés sur disque à partir de l'archive.

Option utile : -v, pour obtenir la liste des noms de fichiers traités.

Exemple : récupérer le fichier myprog.integrale.c archivé sur une cassette placée dans le lecteur 1 :

```
tar -vxf /dev/rmt1 myprog.integrale.c
```

Remarque : Lorsque l'on place dans une archive le contenu d'un ou plusieurs répertoires, il est conseillé d'y faire référence à l'aide d'un chemin d'accès relatif plutôt qu'absolu. Exemple : utiliser

```
tar -cvf chimie.tar ./chimie
  ou
tar -cvf chimie.tar chimie
```

plutôt que

```
tar -cvf chimie.tar /u/dupont/travaux/chimie
```

En effet, dans le dernier exemple, il ne sera pas possible d'extraire des fichiers de l'archive chimie.tar si /u/dupont/travaux n'existe pas (ou plus). De plus, si /u/dupont/travaux/chimie existe, les fichiers extraits seront placés dans ce répertoire et pourraient écraser des fichiers existants de même nom. Par contre, le nom du fichier d'archive à créer peut être référencé de manière relative ou absolue sans influencer le désarchivage.

5.10 Compactage de fichiers

Les commandes **compress** et **pack** sont utilisées pour compacter des fichiers. Elles suppriment les fichiers dont les noms leur sont communiqués et les remplacent par les fichiers compactés. Les noms des fichiers compactés sont obtenus en ajoutant un suffixe aux noms des fichiers originaux : .Z pour **compress** et .z pour **pack**.

Les algorithmes de compactage utilisés par ces deux commandes sont différents. Pour décompacter des fichiers, il faut donc utiliser une commande qui correpond à celle qui a servi au compactage : **uncompress** pour **compress** et **unpack** pour **pack**.

La commande **compress** utilise l'algorithme *Lempel-Zev* modifié [3].

compress [options] dgfiles

Cette commande comprime les *dgfiles* indiqués et ajoute le suffixe .Z à leurs noms.

Options utiles :

- -v : écrit le taux de compression sur stdout.
- -f : force la compression même si un fichier de suffixe .Z existant doit être effacé.
- -c : les fichiers comprimés sont écrits sur stdout.

uncompress [options] dgfiles

Cette commande décomprime les *dgfiles* et supprime le suffixe .Z de leurs noms. Les options présentées ci-dessus pour la commande **compress** peuvent aussi être utilisées avec **uncompress**.

pack [options] dgfiles

Cette commande compacte les fichiers indiqués et ajoute le suffixe .z à leurs noms.

Options :

- - : donne des statistiques relatives au compactage.
- -f : force le compactage; par défaut, celui-ci n'est pas réalisé si le taux de réduction de la taille du fichier n'est pas suffisant.

unpack dgfiles

Cette commande décompacte les fichiers indiqués et supprime le suffixe .z de leurs noms.

Exemple : compacter le fichier chimie.tar :

```
pack - chimie.tar
```

Cette commande crée le fichier compacté chimie.tar.z et supprime le fichier chimie.tar.

5.11 Compression et archivage de fichiers

La commande **zip** permet de comprimer et archiver des fichiers. Elle est analogue à une combinaison des commandes **compress** et **tar**.

zip [options] [zipfile] [dgfiles]

- *zipfile* désigne le fichier d'archive. Si le nom du *zipfile* ne comprend pas de point, le suffixe *.zip* est automatiquement ajouté.
- *dgfiles* désigne les fichiers à comprimer et archiver.

Option utile : **-r** pour le parcours récursif de répertoires. Exemple : créer une archive compressée pvm3.zip du contenu du répertoire pvm3/bin/RS6K et de tous les sous-répertoires :

[3] Cf. Terry A. Welch, *Technique for High Performance Data Compression*, IEEE Computer, vol.17, no.6, june 1984, pp 8-19.

```
zip -r pvm3 pvm3/bin/RS6K
```

Cas particuliers :

- si **-@** est spécifié en lieu et place de *dgfiles*, la liste des fichiers à comprimer et archiver est lue sur stdin. Exemple :

  ```
  find . -name '*.[ch]' -print | zip sources -@
  ```

- si **-** est spécifié en lieu et place de *dgfiles*, c'est le contenu du stdin qui est pris comme fichier à comprimer et archiver.
- si **-** est spécifié en lieu et place du *zipfile*, l'archive est écrite sur stdout.

La commande **unzip** effectue l'opération inverse de **zip** :

unzip [options] zipfile [dgfiles]

dgfiles indique les membres de l'archives qui doivent être extraits et décompressés. Défaut : tous les membres. Exemples :

1. Voir le contenu de l'archive pvm3.zip :

   ```
   unzip -l pvm3.zip
   ```

2. Extraire et décomprimer les membres dont le nom se termine par paramat3.*.f de l'archive pvm3.zip :

   ```
   unzip pvm3.zip *.paramat3.*.f
   ```

   Les fichiers et sous-répertoires éventuels sont recréé dans le répertoire en cours.

Pour plus de détails, voir **man zip** et **man unzip**.

6.0 Sécurités

6.1 Sécurité au niveau du système

La sécurité d'accès est assurée par un système de *login names* et de mots de passe. Chaque utilisateur est identifié par un *login name* unique auquel est associé un mot de passe.

La liste des utilisateurs autorisés est stockée dans le fichier /etc/passwd qui contient, pour chaque utilisateur :

- le *login name*,
- le mot de passe codé,
- un numéro d'utilisateur,
- un numéro de groupe,
- le nom de personne,
- le nom du répertoire principal,
- le nom du programme à exécuter au terme de la procédure de démarrage (généralement le shell).

Un utilisateur peut appartenir à un ou plusieurs groupes. Les groupes sont définis dans le fichier /etc/group. Le groupe mentionné dans le fichier /etc/passwd est le *login group*.

L'utilisateur peut changer son mot de passe via la commande suivante :

passwd

Remarques :

1. La commande **who** écrit sur stdout la liste des *login names* des utilisateurs actifs.

2. La commande **who am i** écrit sur stdout le *login name* de l'utilisateur actuel.

3. Le logiciel NIS (*Network Information System*), anciennement appelé *Yellow Pages*, permet de gérer de façon centralisée des fichiers du système, notamment le fichier des mots de passe. Cela signifie qu'un utilisateur qui possède un compte sur plusieurs machines du même domaine NIS ne doit gérer qu'un seul mot de passe pour tous ces comptes.

6.2 Sécurité au niveau des fichiers

La sécurité au niveau des fichiers est assurée par un système de droits d'accès.

A chaque fichier sont associés 9 *flags* qui représentent les droits d'accès (revoir la sortie de la commande **ls**, «Gestion de fichiers» à la page 10); les 3 premiers représentent les droits d'accès du propriétaire du fichier; les 3 suivants, ceux du groupe auquel il appartient; les 3 derniers, ceux des autres utilisateurs. Les droits d'accès sont représentés par des codes dont les principaux sont :

```
Pour un fichier     répertoire

  r     lecture     lecture du contenu
  w     écriture    création et suppression de fichiers
  x     exécution   autorisation d'en faire le répertoire
                    courant
```

La commande suivante permet, au propriétaire, de changer les droits d'accès à ses fichiers :

chmod [options] [users] {+|-} perm [dgfiles] [dirs]

- *users* représente les utilisateurs pour lesquels on redéfinit les droits d'accès. On note :

  u pour le propriétaire (c'est le défaut),
  g pour le groupe,
  o pour les autres utilisateurs,
  a pour tous;

- *perm* représente les droits d'accès; ils se notent r, w, x, ...[4] comme indiqué précédemment; ils sont précédés du signe + si on les accorde; du signe - si on les retire;

- option utile : -R applique la commande **chmod** récursivement à tous les sous-répertoires des répertoires désignés.

Exemples :

```
chmod ug+x ./progs/fort/*.out
```

[4] Pour la liste complète des différents droits d'accès, voir **man chmod**.

donne le droit d'exécuter les fichiers ./progs/fort/*.out au propriétaire du fichier et aux membres du groupe auquel il appartient.

Rappel, la commande **ls -l** permet de voir les droits d'accès.

6.3 Sécurité globale

L'administrateur d'un système peut imposer des règles particulières de sécurité, notamment pour la structure des mots de passe ou l'accès à certaines ressources. Il sera donc consulté pour tout problème de ce type.

7.0 Help

L'AIX est fourni avec une documentation électronique que l'on peut consulter via les commandes **man** et **info**.

7.1 Commande man

man [options] command

man permet d'obtenir des informations sur différents articles tels que les commandes AIX, les *subroutines* et les fichiers du système.

Diverses options peuvent être utilisées pour obtenir ces informations sous une forme condensée ou détaillée, ou pour effectuer des recherches sur base de mots-clé.

Le contrôle de l'affichage de la documentation relative à une commande peut s'effectuer à l'aide des touches duivantes:

- *return* permet de passer à la ligne suivante;
- *space bar* permet de passer à la page suivante;
- *b* permet de passer à la page précédente;
- *ctrl-c* interrompt la commande.

Exemples:

1.

```
man man
```

pour obtenir la documentation relative à l'utilisation de la commande **man**.

2.

```
man -k passwd
```

pour afficher une ligne d'information pour chaque entrée contenant le mot-clé "passwd"

On peut aussi imprimer la documentation relative à une commande par

man command | lpr [options]

Pour les options de la commande **lpr**, voir «Impression» à la page 15.

7.2 Commande info

info

info donne un accès *online* à la totalité de la documentation fournie avec chaque système AIX et stockée sur disque ou CD-ROM.

La commande **info** s'appuye sur le logiciel InfoExplorer qui exploite pleinement l'environnement AIXwindows (même si son utilisation est possible à partir d'un terminal ASCII).

Par la navigation de type *hyper-texte* au travers de multiples menus, cette commande conversationnelle permet, entre autres:

- d'effectuer des recherches sur base de mots-clés;
- d'accéder à l'ensemble des brochures, des commandes et des tâches;
- de maintenir des *notes* et *bookmarks*;
- de maintenir un *history* des recherches effectuées;
- d'avoir accès à des cours *online*.

7.3 Divers

Des réponses à des questions fréquemment posées par des utilisateurs de UNIX sont reprises sur Internet dans le *Newsgroup* intitulé *comp.unix.questions*. On y trouve notamment un ensemble de fichiers *Unix - Frequently Asked Questions*.

8.0 Editeurs

Les éditeurs suivants sont fournis avec l'AIX :

- éditeurs ASCII : **ed** et **vi**;
- éditeur X-Window : **xedit**.

L'éditeur **emacs**, très apprécié des utilisateurs de UNIX, est également disponible.

Le SEGI recommande **xedit** pour ceux qui se contentent d'un éditeur simple et **emacs** pour ceux qui veulent un éditeur très complet.

Appel de l'éditeur **vi** :

vi dfile

L'éditeur **vi** est décrit dans GC23-2212, «Annexe A. Bibliographie» à la page 99. Voir aussi la commande **man vi**.

Appel de l'éditeur **xedit** :

xedit [dfile]

Appel de l'éditeur **emacs** :

emacs [dfile]

L'éditeur **emacs** est décrit dans CAME91, «Annexe A. Bibliographie» à la page 99. Voir aussi la commande **man emacs**.

Les principales fonctions de **emacs** ainsi que celles de **vi**, qui est disponible dans tous les systèmes UNIX, sont résumées dans le tableau suivant.

Principales fonctions des éditeurs emacs et vi

	Fonction	emacs	vi
Help	command-apropos	C-h a	n/a
	describe-bindings	C-h b	n/a
	describe-function	C-h f	n/a
	describe-key	C-h k	n/a
	describe variable	C-h v	n/a
	where-is	C-h w	n/a
	news	C-h C-n	n/a
Cursor movements	beginning-of-line	C-A	0
	end-of-line	C-E	$
	backward-character	C-B	h ou <BS>
	forward-character	C-F	1 ou <Space>
	down-line	C-N	j ou C-N
	up-line	C-P	k ou C-P
	Beginning of file	A-<	1G
	End of file	A->	G
	next screen	C-V	C-F
	previous screen	A-V	C-B
Insert delete text	open line	C-O	o
	delete-character	C-D	x
	backward-delete-character	<DEL>	X
	backward-delete-character	<BS>	X
	kill-line	C-K	d$
	kill to mark	C-W	d'c
	delete-blank-lines	C-X C-O	dd
Windows	split-window horizontally	C-X 2	n/a
	split-window vertically	C-X 5	n/a
	switch-cursor	C-X O	n/a
	delete-other-windows	C-X 1	n/a
	delete-this-windows	C-X 0	n/a

	Fonction	emacs	vi
Read write files	edit directory	C-X d	n/a
	insert-file	C-X i	:r
	find-file	C-X C-F	n/a
	save-file	C-X C-S	:w
	visit-file	C-X C-V	n/a
	write-file	C-X C-W	:w file
Query replace	Search-and-replace	A-%	:g/s1/s//s2/
	reverse-incremental-search	C-R	n/a
	incremental-search	C-S	n/a
	kill-buffer	C-X k	n/a
	reverse-string-search	C-A-R	?
	string-search	C-A-S	/
Marking	set-mark-here	C-@	mc
	exchange-point-and-mark	C-X C-X	n/a
	mark-paragraph	A-h	n/a
	mark-page	C-X C-P	n/a
	mark-entire-buffer	C-X h	n/a
Miscellaneous	shell-command	A-!	:!cmd
	abort	C-G	<ESC>
	undo-last-changes	C-_	:u
	exit	C-X C-C	:q
	indent-region	C-A-\	n>>
	capitalize-word	A-c	n/a
	lowercase-word	A-L	n/a
	downcase-region	C-X C-L	n/a
	uppercase-word	A-U	n/a
	upcase-region	C-X C-U	n/a
	copy-region	A-w	

Note: Les abréviations suivantes sont utilisées :
C- = ctrl-; A- = Alt-; <BS> = backspace key; n/a = not available

9.0 Environnement

L'environnement dans lequel on travaille est défini par une série de variables, dites variables d'environnement. Ces variables reçoivent une valeur au moment du démarrage du système à partir du fichier /etc/environment [5].

La commande suivante écrit sur stdout les valeurs des variables d'environnement

env

sous la forme suivante :

```
_=/bin/env
MANPATH=/usr/man:/usr/local/man
LANG=en_US
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/prime/%N
VISUAL=emacs
PATH=/bin:/usr/bin:/usr/local/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:.:/u/dupont/bin
XSTATION=xst2
COLUMNS=80
VDIPATH=/usr/lpp/vdi/drivers
WINDOWID=25165832
LOGNAME=dupont
MAIL=/usr/spool/mail/dupont
LOCPATH=/usr/lib/nls/loc
USER=dupont
DISPLAY=xst2:0
SHELL=/bin/ksh
ODMDIR=/etc/objrepos
HOME=/u/dupont
TERM=aixterm
MAILMSG=[YOU HAVE NEW MAIL]
PWD=/u/dupont
TZ=NFT-1
ENV=/etc/rc.ksh
DDEVICE=vdix
LINES=25
A__z=] LOGNAME
```

La commande **echo** permet d'écrire sur stdout la valeur d'une ou plusieurs variables particulières :

echo $var [...]

var désigne le nom de la variable; *$var* fait référence à sa valeur. Exemple :

```
echo $LOGNAME
```

donne dupont.

Quelques-unes des principales variables d'environnement sont décrites ci-dessous ;

- LOGNAME contient le *login name* de l'utilisateur.
- HOME contient le chemin d'accès au répertoire principal de l'utilisateur.
- PATH contient la liste des chemins d'accès aux répertoires dans lesquels le shell recherchera les commandes entrées par l'utilisateur. Les chemins sont séparés par deux-points (:). Rappelons que le point (.) désigne le répertoire en cours. Ici, il peut d'ailleurs être omis; un chemin vide signifie aussi le répertoire en cours (PATH=... :.: ... et PATH=... :: ... sont équivalents).
- LPDEST et PRINTER définissent la file d'attente et le *device* pris en considération par la commande **lpr** (voir «Impression» à la page 15).
- LANG définit le langage et le code-page utilisés. LANG peut avoir l'une des valeurs suivantes :

  ```
  LANG   Langage  Pays       Code-page
  Fr_BE  Français Belgique   PC850
  fr_BE  Français Belgique   ISO8859
  En_US  Anglais  Etats-Unis PC850
  en_US  Anglais  Etats-Unis ISO8859
  ```

  La valeur par défaut est en_US.

- VISUAL=emacs permet d'utiliser les touches fonctions suivantes :
  ctrl-p pour remonter dans la pile des commandes entrées au clavier.
  ctrl-n pour redescendre dans la pile des commandes entrées au clavier.
  ctrl-b pour déplacer le curseur vers la gauche dans le texte d'une commande.
  ctrl-f pour déplacer le curseur vers la droite dans le texte d'une commande.
  ctrl-a pour amener le curseur en début de commande.
  ctrl-e pour amener le curseur en fin de commande.
  ctrl-d pour effacer le caractère en cours.
  ctrl-u pour effacer la ligne en cours.

[5] Pour plus de détails sur la notion de variable, voir «Le shell» à la page 29.

10.0 Processus

10.1 Structure hiérarchique et héritage

Un processus est un programme en cours d'exécution.

Un processus peut en lancer un autre. Par exemple, le shell, qui est lancé par la procédure de *login*, est un processus qui attend que l'utilisateur entre une commande puis l'exécute; l'exécution d'une commande est un nouveau processus.

Chaque processus actif est identifié par un numéro unique appelé *process-id* (pid). La commande **ps** permet d'obtenir des renseignements sur les processus actifs ou sur un processus dont on donne le *pid* :

ps [pid]

Les renseignements produits par la commande **ps** se présentent comme suit :

```
  PID  TTY    TIME CMD
24389 pts/39  0:15 /usr/lpp/x3270/bin/x3270 vm1.ulg.ac.be
27597 pts/39  0:00 -ksh
56910 pts/39  0:00 ps
```

PID est le *process-id*; CMD est la commande qui a lancé le processus.

Les processus ont une structure hiérarchique analogue à celle des répertoires dans un *file system*: chaque processus a un seul père et peut avoir plusieurs fils. Dans cette structure, les processus fils peuvent hériter de tout ou partie de l'environnement créé par le père (c'est-à-dire connaître et utiliser certaines des variables créées par le père) mais l'inverse est impossible. C'est la commande **export** [6] qui permet à un processus père de transmettre son héritage à ses fils :

export variables

En fait, on appelle variable d'environnement une variable du shell exportée. Un processus fils hérite de toutes les variables d'environnement du processus père. Les variables non exportées sont internes à l'interpréteur de commandes, et leur valeur ne peut être consultée par ses processus fils.

Remarque : La commande **env**, qui a été présentée au chapitre «Environnement» à la page 25 comme donnant la liste des variables d'environnement, donne en fait la liste des variables qui ont fait l'objet de la commande **export**.

10.2 Traitement en arrière-plan

Normalement, quand un processus P1 lance un processus P2, P1 s'interrompt jusqu'à ce que P2 soit terminé. Il est cependant possible de lancer des processus concurrents : le symbole & placé à la suite d'une commande lance un processus en arrière-plan (*background*). Le processus en cours ne s'interrompt pas. Ceci est intéressant pour des opérations lourdes, comme par exemple la compilation et surtout le *link edit* d'un programme : on peut taper la commande et la terminer par &; elle lance le processus de compilation et *link edit* en arrière-plan et l'on peut continuer à travailler avec le shell.

Le & à la suite d'un *pipeline* s'applique à l'ensemble des programmes du *pipeline*.

Exemple :

```
find / -name 'disspla*' -print > disspla 2> /dev/null &
```

pendant que le processus lancé par la commande **find** s'exécute, l'interpréteur de commandes rend la main à l'utilisateur qui peut entrer de nouvelles commandes.

10.3 Principales commandes de contrôle des processus

sleep n

Cette commande se contente de patienter n secondes, puis se termine. Par conséquent, elle suspend l'exécution du processus qui l'a appelée pendant n secondes.

wait [pid]

wait attend que le processus *pid* ou, à défaut, tous les processus connus du processus shell en cours soient terminés.

---

[6] **export** est une commande spéciale des shells dérivés du Bourne shell dont le Korn shell (voir «Le shell» à la page 29). Elle n'est pas disponible dans tous les shells mais il existe une notion équivalente dans les autres shells.

kill [options] pid

kill interrompt définitivement les processus *pid*.

nice [-n priority] command **&**

nice lance un processus pour exécuter la commande *command* indiquée en lui attribuant une pritorité relative. *priority* est un nombre entier compris entre 0 et 20. La priorité la plus basse est 20; le défaut est 10. Cette commande n'a d'intérêt que si elle lance le processus en arrière-plan.

nohup command **&**

Le processus lancé par **nohup** continuera à s'exécuter même si la session est terminée (**logout**). Les données que *command* afficherait normalement à l'écran sont redirigées vers le fichier nohup.out dans le répertoire en cours.

at -t date
at -l [jobids]
at -r [jobids]

at avec l'option -t lit les commandes d'un *job* sur stdin et lancera l'exécution de ce job à la date indiquée même si, à ce moment, la session est terminée. Le format de la date est [[CC]YY]MMDDhhmm[.ss], où

- CC indique le siècle (par exemple 19);
- YY indique l'année (par exemple 92);
- MM indique le mois (01 ... 12);
- DD indique le jour (01 ... 31);
- hh indique l'heure (00 ... 23);
- mm indique les minutes (00 ... 59);
- ss indique les secondes (00 ... 59).

Exemple : lancer le programme myjob le 15 octobre de l'année en cours à 11h30 :

```
at -t 10151130 < myjob
```

On reçoit un message de la forme suivante :

```
Job dupont.729225100.a will be run at Thu Oct 15 oct 11:30:00 NFT 1992.
```

où dupont.729225100.a est le *jobid*.

La commande **at** renvoie le contenu de stdout et stderr par **mail** (voir «Courrier électronique» à la page 90) sauf si les sorties standards ont été redirigées dans le *job*, ou si le *job* ne produit aucun résultat sur stdout ou stderr.

at avec l'option -l permet d'obtenir la liste des travaux en attente. Exemple :

```
at -l
```

donne les renseignements suivants :

```
dupont.729225100.a Thu Oct 15 11:30:00 NFT 1992
dupont.719225220.a Fri Oct 16 09:47:00 NFT 1992
dupont.719245900.a Thu Oct 15 11:45:00 NFT 1992
```

On peut obtenir ces renseignements pour des travaux particuliers en indiquant les *jobids* dans la commande **at -l**.

at avec l'option -r permet d'annuler des travaux en attente. Exemple :

```
at -r dupont.719245900.a
```

11.0 Le shell

11.1 Définition

Le shell est la couche la plus externe du système d'exploitation; c'est une interface entre ce dernier et l'utilisateur. Son rôle est de gérer le dialogue entre l'utilisateur et le système, contrôler la gestion des fichiers, les entrées-sorties, l'environnement, les processus, ... En tant qu'interpréteur de commandes, il joue aussi le rôle d'un véritable langage de programmation. C'est cet aspect du shell qui est présenté dans ce chapitre.

Plusieurs shell sont disponibles en AIX :

- le Bourne shell, premier interpréteur de commandes UNIX, disponible dans tous les systèmes UNIX, invoqué par les commandes **bsh** et **sh**;
- le *restricted* shell, un sous-ensemble du Bourne shell, invoqué par la commande **Rsh**; c'est une version réduite du Bourne shell; son intérêt est de limiter les actions des utilisateurs pour des questions de sécurité par exemple;
- le C shell, invoqué par la commande **csh**; sa programmation s'apparente à celle du langage C;
- le *trusted* shell, une variante du C shell, invoqué par la commande **tsh**;
- le *remote* shell, invoqué par la commande **rsh** (voir «rlogin et rsh» à la page 89);
- le Korn shell, invoqué par la commande **ksh**.

Le Korn shell est le shell standard en AIX. Il reprend toutes les caractéristiques du Bourne shell de même que les principales caractéristiques du C shell. Son indicatif (*prompt character*) est le dollar ($). C'est le Korn shell qui est décrit dans le présent guide. Pour les autres, voir «Annexe A. Bibliographie» à la page 99.

En cours de session, tout utilisateur peut passer d'un shell à l'autre en tapant l'une des commandes précédentes. Pour revenir au shell précédent, on tape ctrl-d (fin de fichier) [7].

Un utilisateur peut aussi demander à l'administrateur du système de lui définir un *login shell* particulier. Le *login shell* est celui qui est lancé au terme de la procédure de démarrage. Il s'indique dans le fichier /etc/passwd (voir «Sécurité au niveau du système» à la page 19).

11.2 Variables du shell

La commande suivante permet de créer une variable shell :

```
var=valeur
```

Le nom d'une variable (*var*) peut comporter des lettres, des chiffres et le caractère souligné (_); le premier caractère doit être une lettre ou le caractère souligné. **Il ne peut y avoir d'espace autour du symbole d'affectation (=)**. Si la valeur de la variable contient un ou plusieurs des caractères spéciaux suivants

; & | < > ( ) $ \ ' " espace ` (*left quote*).

elle doit être écrite entre *quotes* simples ou doubles (' ou "). Les simples *quotes* protègent les doubles *quotes* et inversément. Exemple :

```
myvar="L'espace est un caractère spécial"
```

Pour faire référence à la valeur d'une variable, on utilise la notation **$var**. Exemple :

```
echo $myvar
```

affiche

```
L'espace est un caractère spécial
```

à l'écran.

La commande suivante

```
set
```

écrit sur stdout la liste de toutes les variables définies avec leurs valeurs.

11.3 Procédures shell

Une procédure shell (ou *shell script* ou *script file* ou simplement *script*) est une séquence de commandes mémorisées dans un fichier.

---

[7] Attention! ctrl-d au niveau du shell initial équivaut à **logout** sauf si l'option suivante a été définie : **set -o ignoreeof**.

Le shell est un programme qui, comme la plupart des programmes UNIX, lit ses données sur stdin, écrit sur stdout, autorise la redirection des entrées-sorties, accepte des options et paramètres. En particulier, il accepte comme paramètre le nom d'un *script file* et, dans ce cas, il en interprète et exécute les commandes.

On peut lancer l'interprétation d'un *script* de trois façons :

1. En invoquant explicitement le shell :

```
ksh [options] [p1=v1 p2=v2 ...] script [w1 w2 ...]
```

Options utiles :

-a : toutes les variables définies dans le *script* seront automatiquement exportées.
-e : arrête l'exécution du script dès qu'une commande donne un *exit status* différent de zéro.
-f : désactive le mécanisme de substitution des noms de fichiers (voir «Substitutions» à la page 32).
-n : vérifie la syntaxe du script mais ne l'exécute pas.
-o ignoreeof : empêche que le processus shell en cours ne soit arrêté par la rencontre d'une marque de fin de fichier. La commande **exit** devra être utilisée à cet effet.
-s : trie les paramètres positionnels lexicographiquement.
-u : traite les paramètres et variables non définis comme une erreur.
-v : écrit les commandes du script sur stdout au moment où elles sont lues.
-x : écrit les commandes du script sur stdout comme elles sont exécutées.

Ces options peuvent aussi être spécifiées dans le *script* au moyen de la commande **set** (voir «Commandes spéciales» à la page 33).

v1, v2, ..., w1, w2, ... sont les arguments transmis à la procédure. Ce sont des *strings* ou des variables précédées du symbole $ qui seront reçus dans des paramètres locaux à la procédure. Ces paramètres sont :

- des paramètres nommés, c'est-à-dire des identificateurs choisis par l'utilisateur (p1, p2, ...); ils reçoivent respectivement les valeurs v1, v2, ... Les éventuelles variables p1, p2, ... du shell en cours ne sont pas affectées.
- des paramètres de position : $1, $2, ...; ils reçoivent les valeurs w1, w2, ...

Les paramètres nommés peuvent avoir des attributs définis par la commande **typeset** (voir «Commandes spéciales» à la page 33).

Les variables spéciales suivantes concernent les paramètres positionnels :

$0 = nom du script

$* = tous les paramètres positionnels

$@ = tous les paramètres positionnels; équivalent à $* sauf quand écrit entre doubles *quotes* : "$@" n'est pas évalué

$# = nombre de paramètres positionnels

2. Sans référence explicite au shell; le script doit être exécutable (voir la commande **chmod** au paragraphe «Sécurité au niveau des fichiers» à la page 19) :

```
[p1=v1 p2=v2 ...] script [w1 w2 ...]
```

3. Dans les deux cas précédents, les opérations se déroulent comme suit :
   - le processus shell initial (1) est en attente d'une entrée au terminal;
   - on tape **ksh ... script ...** ou **... script ...**; d'où un second processus shell (2) est lancé pour exécuter le script.

   L'environnement de (1) peut être hérité par (2) si une commande **export** a été exécutée; mais l'environnement éventuellement créé par (2) ne modifie pas celui de (1). Si on veut que le **script** modifie l'environnement de (1), il faut qu'il soit exécuté par le shell initial (1). La commande suivante, notée point (.), permet de faire exécuter un *script* par le shell en cours :

```
. script
```

Un espace au moins doit séparer le point et le *script*. Les commandes du *script* seront exécutées exactement comme si elles étaient entrées au terminal. Il n'est donc pas possible, dans ce dernier cas, de transmettre des arguments au *script*.

Remarque : La première ligne d'un script peut contenir l'indication du shell pour lequel il est écrit :

```
#! /bin/ksh
```

Cette ligne n'est prise en considération que si le script est lancé sans référence explicite à un shell particulier. Si le type n'est pas indiqué, le script sera exécuté sous contrôle du shell en cours si c'est le Korn shell.

Un programme shell peut être composé d'un certain nombre d'éléments de base (des commentaires, des métacaractères, des *patterns*, des constantes, des variables, des tableaux, des expressions, des commandes spéciales, des instructions de contrôle, des instructions d'entrée-sortie et de reprise d'erreur). Ces éléments de base sont présentés dans les paragraphes qui suivent pour le Korn shell.

11.4 Commentaires

Tout string précédé de # jusqu'à la fin de ligne est un commentaire.

11.5 Métacaractères

Les caractères génériques (*, ?, [...] - voir «Fichiers» à la page 7) de même que les caractères spéciaux repris dans le Tableau 2 ont une signification spéciale pour le shell :

Métacaractère	Signification
#	le reste de la ligne est un commentaire
|	*pipe*, voir «Filtres, tubes et pipelines» à la page 10
&	*ampersand*, voir «Traitement en arrière-plan» à la page 27
;	séparateur de commandes, voir «Syntaxe» à la page 9
<	voir «Redirection des entrées sorties» à la page 9
>	idem
(	*left parenthesis*
)	*right parenthesis*
$	*dollar*, voir ci-dessous, dans le présent chapitre
`	*left quote*, voir «Substitutions» à la page 32
'	*right or single quote*, voir «Substitutions» à la page 32
"	*double quote*, voir «Substitutions» à la page 32
\	*backslash* : \n représente le caractère *newline* (code ASCII x'0a'); \t, le caractère *tab* (code ASCII x'09'). Voir aussi ci-dessous.

Tableau 2. Métacaractères du shell

Ces caractères peuvent perdre leur signification particulière s'ils sont entourés de *quotes* simples ou doubles ou s'ils sont précédés individuellement de la barre oblique inversée (\). Pour des détails, voir «Substitutions» à la page 32.

11.6 Patterns

Un *pattern* est un mot qui peut contenir un ou plusieurs caractères génériques.

Un *pattern* peut être utilisé

- pour générer des noms de fichiers; exemple :

```
ls -l *.f
```

- pour voir si un mot correspond ou non au *pattern*; exemple :

```
case $F in
  fich?) ...
```

(voir «Sélection de cas» à la page 35).

Quand un *pattern* est utilisé pour la génération de noms de fichiers, le point (.) initial ou le point qui suit directement un *slash* (/), ainsi que le *slash* lui-même, doivent être spécifiés dans le *pattern*.

Dans une liste, les *patterns* sont séparés les uns des autres par une barre verticale (|).

11.7 Constantes

String éventuellement entre *quotes* simples ou doubles. Les simples *quotes* protègent les doubles et inversément.

Constantes numériques (entières uniquement) : [b#]n
b est la base : 2 <= b <= 36 (défaut : 10)
n est un nombre dans la base indiquée

11.8 Variables

```
var=val # val peut être de type caractère ou numérique
let var=expN # expN représente une expression numérique
```

11.9 Tableaux

On peut définir des tableaux à une dimension :

```
set -A tab v0 v1 v2 ... v511 (511 = max)
```

et s'y référer par

```
${tab[$I]} : Ième élément de tab
${#tab[*]} = nb d'éléments de tab
```

11.10 Substitutions

Lors de l'interprétation d'un *script*, le shell analyse chaque *string* et lui applique un mécanisme de substitution qui comporte les quatre étapes suivantes :

1. Substitution paramétrique :

```
$var       = valeur de var
${var}     = valeur de var (les accolades peuvent être
             utiles pour éviter des confusions;
             exemple : ${substitution}nonsubstitution
${#var}    = longueur de var
${var-val} = $var si var est définie, val sinon
${var=val} = $var; si var était indéfinie, elle reçoit
             d'abord la valeur val
${var+val} = val si var est définie, rien sinon
${var?mes} = $var si var est définie, impression du
             message mes et exit sinon
${var#pat} = $var si le pattern pat ne correspond pas
             au début de $var, $var amputé de la partie
             la plus petite correspondant à pat sinon
${var##pat}= $var si le pattern pat ne correspond pas
             au début de $var, $var amputé de la partie
             la plus longue correspondant à pat sinon
${var%pat} = $var si le pattern pat ne correspond pas
             à la fin de $var, $var amputé de la partie
             la plus petite correspondant à pat sinon
${var%%pat}= $var si le pattern pat ne correspond pas
             à la fin de $var, $var amputé de la partie
             la plus longue correspondant à pat sinon
```

Exemples : sachant que la variable PWD contient le nom du répertoire en cours, chemin d'accès depuis la racine compris; par exemple, *PWD=/u/dupont/progs/fort* :

a.

```
echo ${PWD#*/}
```

donne *u/dupont/progs/fort* car la plus petite partie du début de PWD qui correspond au *pattern* */ est son premier caractère (/).

b.

```
echo ${PWD##*/}
```

donne *fort* car la plus longue partie du début de PWD qui correspond au *pattern* */ est */u/dupont/progs/*. Voir «Le fichier profile» à la page 38 pour une application pratique.

Remarque : Le mécanisme de substitution paramétrique n'est appliqué qu'une seule fois dans un *string*. Exemple :

```
Y=Hello
X='$Y'  # $Y est protégé contre toute substitution
        # par les simples quotes
echo $X # donne $Y et non Hello!
```

Pour que la substitution paramétrique soit appliquée récursivement, il faut utiliser la commande **eval** (voir «Commandes spéciales» à la page 33) qui évalue son argument et le transmet au shell comme une commande à exécuter. Avec l'exemple précédent,

```
eval echo $X # donne Hello
```

2. Substitution de commande :

   $(commande)=Résultat de la commande.
   `commande` =Résultat de la commande (ancienne forme;
   le caractère utilisé pour encadrer la commande est
   le *quote* inversé).

3. Interprétation des espaces : cette étape consiste à décomposer le *string* résultant des substitutions précédentes en **mots**. Les caractères reconnus comme séparateurs de mots sont définis dans la variable spéciale $IFS qui, par défaut, contient les 3 caractères suivants : *space*, *tab*, *newline*.

4. Génération de noms de fichiers : chaque mot résultant de l'étape précédente est analysé et remplacé par une liste de noms de fichiers s'il contient des caractères génériques.

Remarques :

1. Un *string* entouré de simples *quotes* est protégé contre toute substitution.

2. Un *string* entouré de doubles *quotes* est protégé contre l'interprétation des espaces et la génération de noms de fichiers; donc, seules les substitutions paramétriques et de commandes sont effectuées.

3. Dans les autres cas, les 4 étapes du mécanisme de substitution sont appliquées.

11.11 Expressions

11.11.1 Expressions numériques (expN)

Les expressions numériques entières peuvent notamment être utilisées dans l'instruction suivante :

let var=expN

Les opérateurs sont + - * / et % (reste). Ils suivent les règles habituelles de commutativité et d'associativité. Des parenthèses peuvent être utilisées. Il ne peut y avoir d'espaces dans une expN. Exemples :

```
let X=$1+($2*$3)/$4 # dans let, * perd sa signification
                    # de caractère générique
let totx=$totx+$x
```

11.11.2 Expressions logiques (expL)

11.11.2.1 Expressions logiques concernant les fichiers

Dans le tableau suivant, *f* représente un nom de fichier.

-a f : vrai si f existe **-d** f : vrai si f existe et est un répertoire **-f** f : vrai si f existe et est un fichier ordinaire **-s** f : vrai si f existe et n'est pas vide **-r** f : vrai si f existe et est accessible en lecture **-w** f : vrai si f existe et est accessible en écriture

11.11.2.2 Expressions logiques concernant les strings

Dans le tableau suivant, *s, s1, s2* représentent des *strings* :

-n s : vrai si s a une longueur > 0 **-z** s : vrai si s a une longueur = 0 s1 **=** s2 : vrai si s1=s2 s1 **!=** s2 : vrai si s1<>s2 s1 **<** s2 : vrai si s1 est avant s2 sur base du code ASCII s1 **>** s2 : vrai si s1 est après s2 sur base du code ASCII

11.11.2.3 Expressions logiques portant sur des expressions numériques

Dans le tableau suivant, *e1, e2* représentent des expressions numériques

e1 **-eq** e2 : vrai si e1=e2 e1 **-ne** e2 : vrai si e1<>e2 e1 **-lt** e2 : vrai si e1<e2 e1 **-gt** e2 : vrai si e1>e2 e1 **-le** e2 : vrai si e1<=e2 e1 **-ge** e2 : vrai si e1>=e2

11.11.2.4 Expressions logiques composées

Les expressions logiques peuvent être combinées au moyen des parenthèses et des opérateurs logiques suivants :

! : négation **&&** ou **-a** : et logique **||** ou **-o** : ou logique

11.11.2.5 Exit status

Toute commande retourne un *exit status* qui est 0 si la commande s'est exécutée normalement. En particulier, les commandes **test** expL et [expL] évaluent expL et retournent un *exit status* de zéro si expL est vrai. C'est la variable **$?** qui mémorise l'*exit status* de la dernière commande exécutée.

Remarque : Un espace au moins doit séparer les crochets ouverts ou fermés des expressions logiques [expL].

11.12 Commandes spéciales

La commande alias :

alias [options] [nom[=valeur]]

- alias nom=valeur définit *nom* comme alias de valeur.
- alias sans arguments donne la liste des alias existants sur stdout.
- Option utile : -x pour exporter l'alias, c'est-à-dire le transmettre aux processus descendants.

Exemple : alias dir='ls -l'

La commande eval :

eval argument

L'argument est évalué et la commande résultante est exécutée.

Remarque : L'argument ne peut contenir de *pipe* (|), d'*ampersand* (&) ni d'opérateurs de redirection des entrées-sorties.

La commande exit :

exit [n]

provoque la sortie du script avec *n* comme *exit status* ou, à défaut, l'*exit status* de la dernière commande exécutée.

La commande export :

export var[=valeur] ...

transmet les variables indiquées aux processus descendants.

La commande getopts :

getopts OptionString var [args]

permet de traiter les options dans le *style UNIX*, c'est-à-dire les options précédées des signes - ou +.

args représente des variables, des paramètres nommés par exemple. Si des *args* sont spécifiés, ce sont ces variables qui sont traitées; sinon, ce sont les paramètres positionnels ($1, $2, ...).

OptionString désigne les lettres des options autorisées. Une lettre suivie du caractère **:** désigne une option qui doit être suivie d'un *string*.

Chaque fois que la commande **getopts** est exécutée, elle place la lettre de l'option suivante dans la variable *var*, le numéro de l'option dans la variable OPTIND, l'éventuel *string* dans la variable OPTARG (: si le *string* est omis).

getopts retourne un *exit status* différent de zéro dès qu'il n'y a plus d'options. Voir exemple **testgetopts** au paragraphe «Répétition» à la page 36.

La commande set :

On a déjà vu que la commande **set** permet de définir un tableau (voir «Tableaux» à la page 31).

Elle permet également de définir des options pour le script en cours, options qui peuvent aussi être spécifiées lors de l'invocation du shell (voir «Procédures shell» à la page 29) :

set [options]

Un signe + devant une option désactive cette option. La variable $- contient la liste des options activées.

Elle permet encore d'affecter des valeurs aux paramètres positionnels.

Exemple 1 :

```
set UN DEUX TROIS
```

affecte UN à $1, DEUX à $2, TROIS à $3.

Exemple 2 :

```
set $(date)
```

affecte les différents mots qui résultent de la commande **date** respectivement à $1, $2, $3, ...

Sans options ni arguments, la commande **set** écrit sur stdout la liste des variables et paramètres nommés avec leur valeur.

La commande shift :

shift [n]

renomme les paramètres positionnels $n+1, $n+2, ... en $1, $2, ... et décrémente $# de n. La valeur de n par défaut est 1. n peut être une expN dont la valeur est comprise entre 1 et $#. Voir exemple **testshift** au paragraphe «Répétition» à la page 36.

La commande typeset :

typeset [options] [name[=val] ...]

name désigne un nom de paramètre ou de variable auquel la commande **typeset** peut affecter des attributs et une valeur *val*.

Options utiles :

-L[n] : *n* est un entier qui désigne la longueur de la variable; par défaut, cette longueur est égale à la longueur de la première valeur affectée. -L provoque un alignement à gauche.
-R[n] : provoque un alignement à droite.
-Z[n] : provoque un alignement à droite avec remplissage par des zéros.
-i[b] : définit le type de la variable comme *integer*; utile comme contrôle et pour la performance des opérations arithmétiques. *b* désigne la base du système de numération (défaut 10).
-l : convertit en minuscules.
-u : convertit en majuscules.
-r : définit la variable *read only*.
-x : exporte la variable.

11.13 Instructions de contrôle

11.13.1 Alternative

if $commandes_1^0$
then $commandes_1^1$
elif $commandes_2^0$
then $commandes_2^1$
...
else $commandes_3^1$
fi

Notes :

1. Dans toutes les listes de commandes, les commandes peuvent être séparées par le caractère *newline* ou point-virgule (;).

2. C'est l'*exit status* de la dernière commande de $commandes_n^0$ qui détermine si les $commandes_n^1$ doivent être exécutées ou non : un *exit status* égal à zéro est interprété comme vrai. En particulier, $commandes_n^0$ peut être de l'une des formes suivantes :

```
test expL
[ expL ]
```

3.

```
if commande1
  then commande2
fi
```

peut aussi s'écrire

```
commande1 && commande2
```

et

```
if ! commande1
  then commande2
fi
```

peut aussi s'écrire

```
commande1 || commande2
```

Exemple : le script suivant, appelé **dir**, exécute l'une des commandes **ls -l | more** ou **ls -l** selon que l'argument **-p** est spécifié ou non :

```
#! /bin/ksh
# dir [/p]
if [ $# -gt 0 ]
  then
    if [ $1 = /p -o $1 = /P ]
      then ls -l | more
      else echo Option invalide
    fi
  else ls -l
fi
```

11.13.2 Sélection de cas

case word **in**
$pat_1^1 \mid pat_1^2 \mid$ ...) $commandes_1$**;;**
$pat_2^1 \mid pat_2^2 \mid$ ...) $commandes_2$**;;**
...**;;**
...
esac

pat_n^1, pat_n^2, ... désignent des *patterns*; ceux-ci ne sont soumis qu'aux deux premières étapes du mécanismes de substitution, la substitution paramétrique et la substitution de commande (voir «Substitutions» à la page 32). Si *word* correspond à l'un des *patterns* pat_n^1,pat_n^2, ..., alors les $commandes_n$ sont exécutées.

Exemple : l'exemple suivant est une autre version du script **dir** :

```
#! /bin/ksh
# dir [/p]
if [ $# -gt 0 ]
  then
    case $1 in
      /p | /P) ls -l | more
      *) echo Options invalide;;
    esac
  else
    ls -l
fi
```

11.13.3 Répétition

Les deux instructions suivantes permettent de programmer des boucles :

for var **in** wordlist
  do commandes
done

Chacun des mots de la *wordlist* est successivement attribué à la variable var et les *commandes* sont chaque fois exécutées.

while $commandes_0$
  do $commandes_1$
done

La boucle est exécutée tant que l'*exit status* de la dernière commande de $commandes_0$ est nul.

Exemple 1 : l'exemple suivant est une généralisation du script **dir** qui peut être utilisé comme indiqué dans les commentaires :

```
#! /bin/ksh
# dir [dgfiles] [/p][/w][/t][/s][/d]
# /p : more
# /w : wide format
# /t : sorted on date
# /s : scan subdirectories
# /d : directories only
F=
P=
W=l
T=
S=
D=
for I in $*
  do
    case $I in
      /p | /P) P=' | more';;
      /w | /W) W=;;
      /t | /T) T=t;;
      /s | /S) S=R;;
      /d | /D) D=d;;
      /*) echo Option $I invalide;;
      *) F="$F $I"
    esac
done
if [ .$D = .d ]
  then
    SWITCHES="-$W"
    if [ $SWITCHES = - ]
      then SWITCHES=
    fi
    eval "li -Od -$W $F $P"
  else
    SWITCHES="-$W$T$S"
    if [ $SWITCHES = - ]
      then SWITCHES=
    fi
    eval "ls $SWITCHES $F $P"
fi
```

Exemple 2 : l'exemple suivant, nommé **testgetopts**, montre la façon de traiter les options et arguments *style UNIX* d'un script qui peut être invoqué comme indiqué dans les commentaires :

```
#! /bin/ksh
# testgetopts [-|+a] [-|+b string] [-|+c] [argument]
a=
b=
c=
arg=
while getopts ab:c OPT
  do
    case $OPT in
      a)      a=-;;
      +a)     a=+;;
      b | +b) b=$OPTARG;;
      c)      c=-;;
      +c)     c=+;;
    esac
done
if [ $# -ge $OPTIND ]
  then arg=$(eval echo '$'$OPTIND)
fi
 ...
```

Exemple 3 : le script suivant, nommé **testshift**, affiche la somme des paramètres positionnels qui lui sont fournis :

```
#! /bin/ksh
# testshift
typeset –i SOMME=0
while [ .$1 != . ]
  do let SOMME=$SOMME+$1
     shift
  done
echo $SOMME
```

11.14 Instructions d'entrée-sortie

echo [options] [arguments]

Ecrit ses *arguments* sur stdout. Option utile :

-n : annule le saut de ligne après écriture.

Exemple :

```
PS1=
echo –n Entrez donnée :
```

La variable PS1 contient l'indicatif du shell, $ par défaut. Ici, elle est initialisée avec le *string* vide. La commande **echo** affichera donc le message "Entrez donnée" non précédé de l'indicatif du shell et en maintenant le curseur sur la ligne en cours.

print [options] [arguments]

Ecrit ses *arguments* sur stdout. Options utiles :

-n : annule le saut de ligne après écriture.

-u n : n désigne le *file descriptor* du fichier d'output. Défaut : 1 (=stdout).

read [options] [var?prompt] [var...]

lit des données sur stdin, les décompose en mots sur base des séparateurs contenus dans la variable $IFS et affecte ces mots aux variables spécifiées. Si la première variable est suivie de *?prompt*, le string *prompt* est écrit sur stderr. Option utile :

-u n : n désigne le *file descriptor* du fichier d'input. Défaut : 0 (=stdin).

read retourne un *exit status* différent de 0 quand un caractère de fin de fichier (ctrl-d) est lu à l'écran.

Remarque : Pour utiliser l'option *-u n* des commandes **print** et **read**, il est nécessaire, au préalable, d'associer le *file descriptor* n à un fichier au moyen de la commande **exec** comme suit :

exec n**op** file

où **op** désigne un opérateur de redirection (voir «Redirection des entrées sorties» à la page 9).

Exemple : le script **testread** suivant lit le fichier myfile.data et reproduit son contenu à l'écran :

```
#! /bin/ksh
# testread
exec 3< myfile.data # open input file
while read –u3 REC
  do
    echo $REC
  done
```

11.15 Gestion des interruptions

Quand une commande se termine anormalement, elle donne un *exit status* différent de zéro, un message est généralement écrit sur stderr et l'exécution du *script* reprend à la commande suivante. Il est possible de définir une action différente de l'action standard en cas de fin anormale d'une commande au moyen de la commande **trap** [8]:

trap [commands] [ERR]

S'il y a plusieurs commandes, elles doivent être entre *quotes* et séparées par des points-virgules (;).

Exemple :

```
trap 'rm $tmp* > /dev/null; exit' ERR
```

Remarque : $? mémorise l'*exit status* de la dernière commande exécutée.

[8] Le message système est quand même envoyé sur stderr.

11.16 Le fichier profile

Le fichier *profile* est un *script* dont le nom est **.profile** et dont l'exécution est lancée automatiquement au terme de la procédure de *login*. C'est l'exemple le plus courant d'utilisation du shell comme langage de programmation.

Exemple de fichier *profile* :

```
#! /bin/ksh
# .profile

PATH=.:$HOME/bin:$PATH
export PATH           # To let PATH known to son shell

#export PS1='$PWD $ ' # To get current directory
                      # into the prompt
export PS1='${PWD##*/} $ ' # idem to get last component

if [ -s "$MAIL" ]     # Check that $MAIL exists
                      # and is not empty
then echo "$MAILMSG"  # $MAILMSG = [YOU HAVE NEW MAIL]
fi

if [ -n "$XSTATION" ]
then startx           # Init AIXWindows
fi
```

Remarques.

1. La procédure d'initialisation (*boot*) engendre un processus fils chaque fois qu'un utilisateur entre en session. Ce processus exécute un utilitaire (**getty**) qui affiche les messages d'invite à entrer en connexion puis lance l'utilitaire **login** (ou **rlogin** si l'on travaille en *remote*). Celui-ci vérifie nom d'usager et mot de passe, place l'utilisateur dans son répertoire principal et appelle le programme à exécuter après connexion, programme défini dans /etc/passwd, généralement le Korn shell. Le Korn shell initialise les variables d'environnement à partir de /etc/environment, exécute le profile système, /etc/profile, puis l'éventuel profile utilisateur.

2. Chaque fois qu'un nouvel utilisateur AIX est défini sur une des machines gérées directement ou indirectement par le SEGI, un fichier .profile est créé dans son répertoire principal à partir d'un canevas contenu dans /etc/security/.profile. L'utilisateur n'a plus, éventuellement, qu'a adapter ce canevas selon ses besoins. Ce canevas est le suivant :

```
PATH=$PATH:$HOME/bin
export PATH

if [ -n "$XSTATION" ]
then startx
fi

if [ -s "$MAIL" ]
then echo "$MAILMSG"
fi
```

12.0 Utilitaires

Ce chapitre présente un certain nombre d'utilitaires AIX qui n'ont pas encore été rencontrés.

Trois d'entre eux sont décrits en détails. Il s'agit de filtres; ils lisent des données sur stdin ou dans des fichiers spécifiés, les transforment éventuellement et écrivent les résultats sur stdout.

Les autres sont seulement cités. On peut trouver la documentation qui les concerne via la commande **man** (voir «Help» à la page 21).

Tous ces utilitaires peuvent bien entendu être appelés dans des procédures shell.

12.1 Utilitaires de la famille grep

Les trois utilitaires **grep**, **fgrep** et **egrep** se comportent de la même façon : ils lisent les lignes d'une série de fichiers (à défaut, stdin), les comparent avec un *pattern* et écrivent les lignes qui correspondent au *pattern* sur stdout. Nous présentons ci-dessous **egrep**, le plus récent de ces utilitaires.

egrep [options] [pattern] [dgfiles]

Le *pattern* est un *string* qui peut contenir des métacaractères comparables à ceux utilisés par le shell (voir «Patterns» à la page 31); toutefois, **certains métacaractères ont une signification différente pour le shell et pour les utilitaires de la famille grep**. Les *patterns* traités par **grep** et **fgrep** sont appelés *expressions régulières*; ceux traités par **egrep** sont des *expressions régulières étendues*. Le Tableau 3 donne la liste des métacaractères utilisés dans les expressions régulières étendues, dans l'ordre décroissant de priorité.

Pour éviter que des métacaractères soient interprétés par le shell (avant de l'être par **grep**), il faut écrire le *pattern* entre simples *quotes*.

pattern peut être une liste. Dans ce cas, chaque ligne des *dgfiles* est comparée successivement avec les différents éléments du *pattern*.

Métacaractère	Signification
\c	supprime l'éventuelle signification spéciale du caractère c
^	début de ligne
$	fin de ligne
.	caractère quelconque (équivalent à ? en shell)
[...]	classe de caractères comme en shell (exemple [efgh] ou [e-h])
[^...]	désigne les caractères non contenus dans la classe indiquée
r{n}	n occurences de r (*)
r{n,m}	entre n (défaut : 0) et m (défaut : infini) occurences de r (*)
r*	0 ou plusieurs occurences de r (équivalent à r{0,})
r+	1 ou plusieurs occurences de r (équivalent à r{1,}) (*)
r?	0 ou 1 occurence de r (équivalent à r{0,1}) (*)
$r_1 r_2$	r_1 suivi de r_2
$r_1 \mid r_2$	r_1 ou r_2 (*)
Remarque : Les expressions régulières étendues sont indiquées par (*). r désigne un caractère, une classe, ou une expression régulière étendue entre parenthèses.	

Tableau 3. Caractères spéciaux utilisés dans les expressions régulières et régulières étendues

Options utiles :

- -c : affiche seulement le nombre de correspondances.
- -f dfile : désigne un fichier de *patterns*. Chaque ligne correspond à un *pattern*. Une ligne vide correspond au *pattern* nul. La correspondance avec un *pattern* nul est toujours vérifiée.
- -h : quand plusieurs *dgfiles* sont indiqués, les noms des fichiers dans lesquels des correspondances sont trouvées sont donnés. -h supprime la production de ces noms de fichiers.
- -i : ignore la différence entre majuscules et minuscules lors des comparaisons.
- -l ou -y : donne seulement les noms des fichiers où des correspondances sont vérifiées.
- -n : donne les numéros des lignes où une correspondance est vérifiée.
- -v : reproduit toutes les lignes des *dgfiles* sauf celles qui correspondent au *pattern*.
- -w : fait les comparaisons par mots.

- -x : pour qu'il y ait correspondance, le *pattern* doit correspondre à l'entièreté d'une ligne.

Exit status : 0=des correspondances ont été trouvées. 1=aucune correspondance n'a été trouvée; 2=erreur de syntaxe ou fichier inaccessible.

Exemples :

1. Dans le répertoire ./divers, rechercher les fichiers dont le nom se termine par .data et qui contiennent le mot essai en minuscules ou majuscules :

```
egrep -il essai ./divers/*.data
```

2. Afficher les lignes du fichier myprog.f qui contiennent write et XYZ séparés par un nombre quelconque de caractères quelconques :

```
egrep -i 'write.*XYZ' myprog.f
```

3. Afficher les lignes du fichier test.data qui commencent par Début ou qui se terminent par Fin :

```
egrep '^Début|Fin$' test.data
```

4. Voir si Dupont et Durand sont en session :

```
who | egrep -i 'Durand
Dupont'
```

   La commande **who** donne sur stdout la liste des utilisateurs actifs. Dans cet exemple, le *pattern* est composé de deux lignes; le caractère *newline* en fait partie.

5. Recopier un fichier f1 dans un autre f2 en ignorant les lignes blanches et les lignes vides :

```
egrep -v '^ *$' f1 > f2
```

12.2 sed ("Stream editor")

L'utilitaire **sed** est un éditeur qui s'utilise comme un filtre, de façon non interactive. Il lit les lignes d'une série de fichiers (à défaut, stdin) et les écrit sur stdout après les avoir éventuellement transformées :

sed [options] [[-e]commands] [dgfiles]

Options utiles :

- -n : supprime l'écriture sur stdout.
- -f dfile : désigne un fichier de commandes où chaque ligne correspond à une commande.
- -e : annonce les commandes. Utile en cas d'ambiguïté.

Forme générale d'une commande sed :

```
[adresse]opération[argument]
```

adresse détermine la partie des *dgfiles* à laquelle s'applique l'opération. Si l'adresse n'est pas spécifiée, l'opération est appliquée à toutes les lignes des *dgfiles*. *adresse* peut être :

- des numéros de lignes de la forme l1[,l2]. Par défaut, l2=l1. La numérotation des lignes se fait à partir de 1 pour l'ensemble des *dgfiles*;
- $, pour désigner la dernière ligne;
- un *pattern* de la forme /pat1/. Dans ce cas, l'opération indiquée sera appliquée successivement à toutes les lignes qui correspondent au *pattern*;
- deux *patterns* de la forme /pat1/,/pat2/. Dans ce cas, l'opération indiquée sera appliquée à partir de la première ligne qui correspond à pat1 jusqu'à la rencontre d'une ligne qui correspond à pat2. Les *patterns* sont des expressions régulières strictes (voir Tableau 3 à la page 39 - les métacaractères notés (*) dans ce tableau n'ont pas de signification spéciale pour **sed**).

Les principales opérations que **sed** peut effectuer sont les suivantes :

a permet d'insérer une ou plusieurs lignes de texte après la ligne en cours. Chaque ligne du texte à insérer doit être terminée par une barre oblique inversée, sauf la dernière. Exemple :

```
sed '/ci-dessous/a\
1ère ligne à insérer\
2ème ligne à insérer\
3ème ligne à insérer' test.data > test2.data
```

les 3 lignes de texte à insérer seront insérées après chacune des lignes du fichier test.data contenant le mot 'ci-dessous'; le tout sera recopié dans le fichier test2.data.

i permet d'insérer une ou plusieurs lignes de texte avant la ligne en cours.

c permet de remplacer la ligne en cours par une ou plusieurs lignes de texte.

s/pattern/string/flags remplace *pattern* par *string* dans la ligne en cours. Les barres obliques peuvent être remplacées par tout autre caractère, pourvu qu'il ne fasse pas partie du *pattern* ni du *string*. *flags* peut contenir un ou plusieurs arguments :

- g : pour remplacer toutes les occurences du *pattern*;
- n : un nombre, pour remplacer la nème occurence seulement;
- p : pour écrire la ligne sur stdout si un remplacement a été fait (utile si l'option -n a été spécifiée).

d supprime la ligne en cours.

p écrit la ligne en cours sur stdout (utile si l'option -n a été spécifiée).

w dfile écrit la ligne en cours dans le fichier indiqué.

r dfile recopie le contenu du fichier indiqué sur stdout.

! si le code opération est précédé d'un point d'exclamation, l'opération est appliquée aux lignes qui ne correspondent pas à l'adresse indiquée.

q termine le traitement.

Remarques :

1. Plusieurs commandes, séparées par le caractère *newline*, peuvent être transmises à **sed**. Dans ce cas, l'ensemble des commandes est appliqué séquentiellement à chacune des lignes des *dgfiles*. Exemple :

```
sed -n '/^[0-9]/w fich1
/[0-9]$/!w fich2' test.data
```

   écrit les lignes de test.data qui commencent par un chiffre dans fich1 et celles qui ne se terminent pas par un chiffre dans fich2.

2. Dans la partie *string* de la commande de remplacement **s**, le caractère & peut être utilisé pour désigner la partie de la ligne qui correspond au *pattern*. Exemple :

```
sed 's/^......../&;/' test.data > test2.data
```

   introduit le caractère point-virgule (;) après le 8ème caractère de chacune des lignes du fichier test.data.

12.3 awk ("Report generator")

awk, souvent utilisé comme générateur de rapport, est un filtre programmable très puissant. Il utilise un système de notations qui s'inspire du langage C.

awk [options] ['program'] [dgfiles]

awk lit les lignes d'une série de fichiers (*dgfiles* - à défaut, stdin) et les écrit sur stdout après les avoir éventuellement transformées.

Options :

- -f dfile : désigne un fichier qui contient le programme awk.
- -F char : désigne les caractères utilisés comme séparateurs de champs (défaut : espace). Un seul séparateur peut être désigné comme suit : -F";". Pour en désigner plusieurs, on utilise la notation de classe : -F"[,;/]".
- -v variable=value : assigne une valeur à la variable indiquée; cette assignation se produit avant le début de l'exécution du programme.

Forme générale d'un programme awk :

un programme awk peut contenir trois sections (chacune d'elle est optionnelle) :

```
BEGIN { actions qui seront exécutées
        avant lecture de la première ligne
        des fichiers d'input
      }
commandes qui seront exécutées
pour chaque ligne des fichiers d'input
END   { actions qui seront exécutées
        après lecture de la dernière ligne
        des fichiers d'input
      }
```

Action :

une action est une série d'instructions séparées par le caractère point-virgule (;) ou *newline*. Les principales instructions disponibles sont les suivantes :

```
variable=expression
print expression1, expression2 ... [> fichier]
printf format, expression1, expression2 ... [> fichier]
if (condition) statement [else statement]
while (condition) statement
for (expression1; condition; expression2) statement
for (variable in array) statement
exit
```

Toutes ces instructions, sauf l'avant dernière, sont des instructions du langage C. Un exemple d'utilisation de l'instruction "for (variable in array") est donné en fin de paragraphe.

Expressions :

elles s'écrivent comme en C; elles utilisent les opérateurs du C :

opérateurs arithmétiques :

```
+, -, *, /, %, ++, --, +=, -=, *=, /=, %=
```

opérateurs relationnels :

```
==, !=, <, >, <=, >=
```

opérateur de concaténation : espace.

Fonctions :

un certain nombre de fonctions sont disponibles : fonctions arithmétiques (int, sqrt, log, cos, ...), fonctions de *string* (length, substr, index, ...).

Variables :

l'utilisateur peut définir des variables avec l'instruction "variable=expresion". Toute variable utilisée dans un programme awk est automatiquement initialisée avec la valeur *string* vide.

Il existe aussi un certain nombre de variables prédéfinies dont les principales sont données ci-dessous:

$0 : ligne en cours
$1 : 1er champ de la ligne en cours.
$2 : 2ème champ de la ligne en cours.
...
$NF : nombre de champs.
$FS : caractères utilisés comme séparateur de champ dans le fichier d'input. Peut être changé en cours de programme. Peut être une classe de caractères.
$RS : caractère utilisé comme séparateur de lignes dans le fichier d'input (défaut : *newline*).
$OFS : caractère utilisé comme séparateur de champ dans le fichier d'output.
$ORS : caractère utilisé comme séparateur de lignes dans le fichier d'output (défaut : *newline*).
$NR : n° de la ligne en cours (0 dans la section BEGIN).
$FILENAME : nom du fichier d'input en cours.
$FNR : n° de la ligne en cours à l'intérieur du fichier d'input en cours.

Tableau :

L'affectation d'une valeur à une variable indicée (T[3]=12 par exemple) crée un tableau. L'indice d'un tableau peut très bien être un *string*, ce qui permet de créer des tableaux associatifs (voir exemple en fin de paragraphe).

Commandes :

Les commandes que l'on écrit dans le corps du programme, celles qui sont exécutées pour chaque ligne des fichiers d'input, sont de la forme suivante :

```
pattern { action }
```

L'action est exécutée si la ligne correspond au *pattern*. Si le *pattern* n'est pas indiqué, l'action est toujours exécutée. L'action par défaut consiste à écrire la ligne sur stdout.

Trois types de *patterns* peuvent être utilisés avec **awk** :

- des expressions régulières étendues (voir Tableau 3 à la page 39) écrites entre des barres obliques (/); exemple :

  ```
  awk '/^Début|Fin$/' test.data
  ```

  affiche les lignes de test.data qui commencent par le mot Début ou qui se terminent par le mot Fin.

- des expressions relationnelles utilisant les opérateurs relationnels et le *tilde* : ~ (correspond à) et !~ (ne correspond pas à); exemples :

  ```
  awk '$3>40' test.data
  ```

  affiche les lignes de test.data dont le 3ème champ est supérieur à 40;

  ```
  awk '$2 ~ /xy[abc]z/' test.data
  ```

  affiche les lignes de test.data dont le 2ème champ correspond au *pattern* indiqué;

- des combinaisons de *patterns* pour

  - définir des adresses comme dans **sed** (voir «sed ("Stream editor")» à la page 40); exemple:

    ```
    awk '/un/,/deux/' test.data
    ```

    affiche la première ligne de test.data qui contient 'un' jusqu'à la prochaine ligne qui contient 'deux';

  - définir des expressions relationnelles composées au moyen des opérateurs logiques ! (non), && (et), || (ou); exemple :

    ```
    awk '$1=="test" && $2=123'
    ```

Exemples :

1. Appliquer la même commande à une série de fichiers.

   La commande **tar -xf file** permet de récupérer les fichiers archivés dans un fichier sur disque (voir «Archivage de fichiers» à la page 16). Cependant, dans l'option **-f**, un seul fichier peut être indiqué. L'exemple suivant montre comment on peut appliquer la commande **tar** à tous les fichiers archives d'un même répertoire, le répertoire en cours par exemple :

```
ls -l *.tar | awk '{print "tar -xf", $9}' > detar
chmod +x detar
detar
rm *.tar
```

La première commande crée le *script* detar dont chaque ligne contient **tar -xf** suivi d'un nom de fichier (9ème champ des lignes fournies par **ls -l *.tar**). La deuxième commande en fait un *script* exécutable. La troisième commande exécute le *script*. La dernière commande supprime les fichiers d'archives dont on a récupéré le contenu.

2. Utilisation de la fonction substr (*substring*) :

   le *script* suivant extrait d'un string S L caractères à partir de la position K et les écrit sur stdout :

```
#! /bin/ksh
# substr S K L
#
echo $1 | awk '{print substr($0,'$2','$3')}'
```

   Exemple d'utilisation :

```
substr 'Ceci est un exemple' 5 9
```

   donne

```
est un ex
```

3. Utilisation de tableaux associatifs :

   soit le fichier test.data :

```
03/01/93; Dupont   ;34000
05/01/93; Durand   ; 7800
05/01/93; Dupont   ; 5600
06/01/93; Van Wee  ;45000
12/01/93; Dupont   ;-5000
12/01/93; Van Wee  ;-4500
12/01/93; Durand   ;67500
14/01/93; Dupont   ;-5000
```

   et le programme awk, écrit dans le fichier test.awk :

```
BEGIN { FS=";" }
$3>0  { SC+=$3
        TC[$2]+=$3
        TD[$2]-=0 }
$3<0  { SD-=$3
        TD[$2]-=$3
        TC[$2]+=0 }
END   { printf "%10s%10s%10s%10s\n\n", " ", "Credit", "Debit", "Solde"
        for (N in TC)
           printf "%-10s%10d%10d%10d\n", N, TC[N], TD[N], TC[N]-TD[N]
        printf "\n%-10s%10d%10d%10d\n", " Totaux", SC, SD, SC-SD }
```

   La commande suivante

```
awk -f test.awk test.data
```

   donne

```
              Crédit     Débit     Solde

 Van Wee       45000      4500     40500
 Durand        75300         0     75300
 Dupont        39600     10000     29600

 Totaux       159900     14500    145400
```

12.4 Utilitaires divers

- Liste des utilisateurs actifs : **who**.
- Date et heure : **date**.
- Fractionnement de fichiers : **split** et **csplit**
- Comparaison de fichiers : **cmp, comm, sdiff, diff, diff3, dircmp, sum, wc**
- Conversion de données (ASCII-EBCDIC, majuscules-minuscules, ...) : **dd, tr**
- Tri : **sort, uniq, nl, tsort**
- Traitement de données tabulaires : **cut, paste, join**
- Cryptage : **makekey**
- Formatage de texte : **nroff, troff, tbl, eqn, cw, mm, deroff, spell, ptx** ...
- Outils de calculs : **dc** (postfixe), **bc** (infixe), **xcalc** (calculatrice).
- Calendrier et aide mémoire : **ca**l, **calendar**
- Analyse lexicale : **lex**.
- Gestion archives et bibliothèques : **ar**.

- Compilateur d'expressions régulières : **regcmp**.

La description de ces utilitaires peut être consultée au moyen de la commande **man** (voir «Help» à la page 21).

13.0 Perl

Perl est un interpéteur de commandes distribué par le GNU. Il est présenté comme un langage de manipulation de textes, de fichiers, de processus et de communications (communications entre processus de différentes machines via sockets). Ses principaux avantages sur le Shell sont les suivants :

1. Il regroupe des fonctions d'interpréteur de commandes comme shell et de langage de programmation classique comme C.

2. Il peut être utilisé avec plusieurs systèmes d'exploitation différents dont Unix. Il fournit ainsi un modèle de programmation portable à travers différentes architectures.

3. Il est *freeware*.

Perl peut être utilisé avantageusement en lieu et place du Shell et d'utilitaires tels que **sed** et **awk**. On peut lancer l'interprétation d'une procédure de commandes Perl (ou d'un *script* Perl) de deux façons :

1. En invoquant explicitement Perl :

perl [options] script [args]

Principales options :

-d : exécute le script sous contrôle d'un *debugger*. La commande **h** (*help*) du *debugger* permet d'afficher la liste de ses commandes.
-v : affiche le numéro de version.
-w : effectue différents contrôle; en particulier, affiche un message d'avertissement quand une variable scalaire non initialisée est utilisée.

args représente une liste d'arguments qui seront transmis au *script*.

2. Sans référence explicite à Perl :

script [args]

Le script doit être exécutable (voir la commande **chmod** au paragraphe «Sécurité au niveau des fichiers» à la page 19) et commencer par la ligne suivante :

```
#! /usr/local/bin/perl [options]
```

Un programme Perl peut être composé d'un certain nombre d'éléments dont les principaux sont présentés dans les paragraphes qui suivent.

13.1 Scalaires

Un scalaire est un nombre ou un *string*. Une variable scalaire se note *$nom*. Le nom doit commencer par une lettre ou _ et peut ensuite comporter un nombre quelconque de lettres ou chiffres. Exemples :

```
$N=-43.2+'0.1e-3';                      # 1
$Titre='Exemple';                       # 2
$X="$Titre : " . ' $N= ' . "$N \n";     # 3
$Date=`date`;                           # 4
chop($Y=`date`);                        # 5
```

Remarques :

1. Comme dans le shell, tout *string* précédé de # jusqu'à la fin de ligne est un commentaire. Toute instruction doit être terminée par **;**. Plusieurs instructions peuvent être placées sur une même ligne. Une instruction peut se prolonger sur plusieurs lignes.

2. Dans l'exemple 1, '0.1e-3' est un *string*. Perl effectue automatiquement les conversions nécessaires en fonction du contexte. Tous les opérateurs arithmétiques du langage C sont disponibles.

3. L'exemple 2 montre la façon d'initialiser une variable scalaire avec un *string*. Un *string* ne doit pas nécessairement être écrit entre *quotes*; il sera reconnu comme *string* s'il ne peut être confondu avec un autre élément de Perl.

4. L'exemple 3 illustre le mécanisme de substitution. Revoir «Substitutions» à la page 32. Comme en shell, un *string* entouré de simples *quotes* est protégé contre toute substitution. Par contre, la substitution a lieu à l'intérieur d'un string entouré de doubles *quotes*. Le point est l'opérateur de concaténation. La notation **\n** représente le caractère *newline*.

5. L'exemple 4 illustre la substitution de commande. Les *quotes* inversés encadrent la commande **date**. L'*output* de cette commande, caractère *newline* compris, est affecté à la variable $Date.

6. L'exemple 5 illustre l'emploi de l'une des nombreuses fonctions de Perl. La fonction **chop** enlève le dernier caractère du *string* qui lui est fourni comme argument et le renvoie comme résultat de la fonction; ce dernier caractère est ici le *newline*.

7. Toute variable non initialisée a la valeur nulle qui est interprétée comme " (*string* vide) ou 0 selon le contexte.

8. Toujours selon le contexte, *string* vide ou 0 peuvent être interprétés comme la valeur logique faux; toute autre valeur peut être interprétée comme vrai.

13.2 Opérateurs et fonctions

Perl dispose d'un nombre important d'opérateurs. Seuls les principaux sont décrits ci-dessous. Les opérateurs suivants du shell sont supportés : opérateurs arithmétiques («Expressions numériques (expN)» à la page 33), opérateurs relatifs aux fichiers («Expressions logiques concernant les fichiers» à la page 33) sauf **-a**, opérateurs logiques **!**, **&&**, **||** («Expressions logiques composées» à la page 33). Exemple :

```
if (-f 'f1.data') {print "ok \n";} else {print "no \n";}
```

Cet exemple illustre l'emploi de l'opérateur de fichier **-f** et montre en même temps la syntaxe de l'instruction **if** (voir «Instructions composées» à la page 48).

Les opérateurs de comparaisons se répartissent en deux classes selon qu'ils s'appliquent à des données numériques ou à des *strings* :

```
Numérique String
   ==       eq
   !=       ne
   >        gt
   >=       ge
   <        lt
   <=       le
```

La plupart des fonctions de Perl peuvent être utilisées comme fonctions ou comme opérateurs unaires. Exemples :

```
print STDOUT (1+2)*4, "\n"; # 1
print 4*(1+2), "\n";        # 2
print (1+2)*4, "\n";        # 3
```

1. Dans l'exemple 1, **print** est utilisé comme opérateur unaire. Son premier argument est un *filehandle* prédéfini qui désigne le *standard output file* stdout; l'argument suivant est une liste d'expressions dont les valeurs sont envoyées vers stdout.

2. Dans l'exemple 2, **print** est aussi utilisé comme opérateur unaire. Le *filehandle* n'est pas spécifié. Dans ce cas, STDOUT est pris par défaut.

3. Dans l'exemple 3, le mot *print* étant suivi d'une parenthèse ouverte, **print** est utilisé comme une fonction dont l'argument est 1+2. C'est donc la valeur 3 qui est envoyée vers stdout. Le résultat de la fonction **print**, 1 ou 0 selon que l'opération s'est déroulée correctement ou non, est ensuite multiplé par 4.

Les opérateurs/fonctions **exec** et **system** sont particulièrement utiles. Ils permettent de soumettre une ou plusieurs commandes au shell. Après avoir soumis les commandes au shell, **exec** arrête l'exécution du *script*; tandis que **system** attend que les commandes soient terminées avant de permettre au *scritp* de poursuivre son exécution. Exemple :

```
system 'ls -l';
```

13.3 Listes et tableaux

Une liste est un ensemble ordonné de scalaires. Exemple :

```
(45.2, 6..11, 'Hello', $V, $X+1)
```

.. est le *range operator*; 6..11 désigne la suite 6, 7, 8, 9, 10, 11.

Un tableau est une liste nommée. Les noms de tableau suivent les mêmes règles que les noms de variables scalaires sauf qu'ils commencent par @ ou %. @ désigne un tableau indexé par nombre; % désigne un tableau indexé par *string*, ou tableau associatif. Exemple :

```
@T=(45.2, 6..11, 'Hello', $V, $X+1);
%A=('L' ,'Lundi',
    'Ma','Mardi',
    'Me','Mercredi',
    'J' ,'Jeudi',
    'V' ,'Vendredi',
    'S' ,'Samedi',
    'D', 'Dimanche');
```

Remarques :

1. Par défaut, l'index du premier élément de @T est 0; ainsi, par exemple, $T[2] est égal à 7. Cet index peut être mis à 1 en affectant la valeur 1 à la variable spéciale **$[**; par défaut, elle a la valeur 0. Cette variable détermine aussi l'index du premier caractère d'un *string*. On notera l'usage des crochets pour indexer un tableau à index numérique.

2. La notation @T, selon qu'elle est évaluée dans un contexte scalaire ou dans un contexte tableau, donne le nombre d'éléments du tableau @T ou la liste des valeurs des éléments de T. Exemple :

   ```
   $N=@T; # Nombre d'éléments
   print "@T \n"; # Liste des éléments
   ```

3. Les deux instructions suivantes ont des effets un peu différents:

   ```
   print "@T \n";
   print @T, "\n";
   ```

   Elle donne toutes deux la liste des valeurs des éléments de @T. Dans le premier cas, deux éléments consécutifs sont séparés par un espace; dans le second cas non. L'insertion d'espaces a lieu à l'intérieur d'un *string* entouré de double *quotes*. C'est la variable spéciale $" qui détermine le caractère d'insertion; sa valeur par défaut est espace.

4. La variable spéciale $#T donne le numéro du dernier élément du tableau @T.

5. L'instruction suivante définit un tableau vide :

```
@X=();
```

   Ici, $#X= -1 (si $[=0, ce qui est le défaut).

6. La notation suivante permet de faire référence à un sous-tableau de @T :

```
@T[3, 5, 7..9]
```

7. Un tableau associatif tel que %A est une liste de couples de scalaires. Le premier élément de chaque couple joue le rôle de clé d'index. La clé d'index doit être unique. Par exemple, $A{'Me'} est égal à 'Mercredi'. On notera l'usage des accolades pour indexer un tableau associatif.

8. La fonction **keys** s'applique à un tableau associatif. Elle donne la liste des valeurs d'index. Exemple:

```
print keys(%A), "\n";
```

   donne L Ma Me J V S D.

9. Il existe un tableau associatif prédéfini, %ENV, qui contient les valeurs des variables d'environnement connues du processus Perl en cours. Exemple:

```
$ENV{'PATH'}
```

   désigne la valeur de la variable d'environnement PATH.

13.4 Correspondances et substitutions sur base de patterns

Les *patterns* de Perl sont des expressions régulières comparables a celles utilisées dans l'utilitaire **egrep** (revoir Tableau 3 à la page 39) mais plus étendues.

L'opérateur de correspondance (*match operator*) se note

```
[m]/pattern/[g][i]
```

Si **/** est utilisé comme délimiteur, **m** est optionnel. Tout autre caractère non alphanumérique peut être utilisé comme délimiteur; mais dans ce cas **m** est obligatoire. Les *modifiers* **g** et **i** ont la signification suivante :

- **g** : est le *global modifier*; s'il n'est pas spécifié, la recherche des correspondances s'arrête à la première;
- **i** : indique qu'il ne faut pas faire de distinction entre majuscules et minuscules.

Exemples :

```
if ('This is a string' =~ m/is/) {...}      # 1
while ('This is a string' =~ m/is/g) {...} # 2
@T=(...);
print grep(/$P/g, @T), "\n";                # 3
```

Commentaires :

1. =~ est le *match pattern binding operator*. !~ est le *not match pattern binding operator*.

2. Dans l'exemple 1, le résultat est vrai puisque le *pattern* **is** apparait dans le *string*.

3. Dans l'exemple 2, les instructions placées entre accolades seront exécutées deux fois puisque le *pattern* **is** apparait deux fois dans le *string* et que le modificateur **g** a été spécifié.

4. Dans l'exemple 3, tous les éléments de @T qui correspondent au *pattern* sont écrits sur stdout. La fonction **grep** évalue sont premier argument pour chacune des valeurs des arguments qui suivent et renvoie le tableau des valeurs pour lesquels l'évaluation a donné vrai.

L'opérateur de substitution se note

```
s/pattern/exp/[g][i]
```

Exemple :

```
$X='This is a string';
$X =~ s/ is / is not /;
print $X, "\n";
```

La commande **print** écrit *This is not a string* sur stdout.

Remarque : Par défaut, le *string* auquel s'applique le *pattern matching* est la variable spéciale $_, qui est aussi l'*input* par défaut comme on le verra plus loin (voir l'*input operator*, «Entrées-sorties» à la page 48). Ainsi, si $_='This is a string', les deux instructions suivantes sont équivalentes :

```
if ($_ =~ /is/) {...}
if (/is/) {...}
```

Il existe d'autres possibilités pour manipuler les *strings* au moyen de *patterns*. Elles ne sont pas décrites ici car les fonctions **index**, **substr** et **length** permettent de résoudre la plupart des problèmes qui peuvent se poser; d'autant plus que la fonction **substr**, comme beaucoup d'autres fonctions, peut être utilisée comme une *l-value*. Ces fonctions ont le mérite de nous éviter de devoir recourir à des notations souvent obscures. Exemple

: si le début du *string* $string contient le texte 'Subject:', on souhaite en extraire le texte qui suit:

```
$string =~ /^Subject: (.*)/ && ($substring=$1); # 1
if (substr($string,0,8) eq 'Subject:')
  {$substring=substr($string,9);}               # 2
```

Les deux exemples donnent le même résultat, le premier avec concision, le second avec plus de clarté.

13.5 Instructions composées

Les principales instructions composées sont les suivantes :

if (exp) block [**elseif**(exp) block ...] [**else** block]

while (exp) block

until (exp) block

for (exp1**;** exp2**;** exp3) block

foreach [var] (list) block

- Un block est une suite d'instructions entre accolades.
- Dans les boucles *while* et *until*, la condition de fin de boucle est évaluée avant la première itération; *until* inverse simplement le sens du test.
- Dans la boucle *foreach*, la variable *var* prend successivement les valeurs de la liste *list*; le block est exécuté pour chacune des valeurs de *var*. Si *var* n'est pas spécifié, la variable spéciale $_ est prise par défaut.

Exemples :

1. Les trois boucles suivantes sont équivalentes :

```
for ($I=0; $I<10; $I++) {print "$I ";} # 1

foreach $I (0..9) {print "$I ";}       # 2

$I=0;                                  # 3
while ($I<10)
  { print "$I ";
    $I=$I+1;
  }
```

2. Cet exemple affiche la liste des noms et valeurs des variables d'environnement triées dans l'ordre alphabétique des noms :

```
foreach (sort keys(%ENV)) {print "$_=$ENV{$_}\n";}
```

13.6 Entrées-sorties

Ouverture d'un fichier :

open(filehandle**,**filename)

- *filename* et *filehandle* sont des expressions de type *string* qui représentent respectivement le nom d'un fichier et une référence à utiliser dans les autres instructions d'accès à ce fichier.
- Si le *filename* est précédé de
  - **<** ou rien, le fichier est ouvert en mode *input*;
  - **>**, le fichier est ouvert en mode *output*;
  - **>>**, le fichier est ouvert en mode *append*;
  - **+<** ou **+>** le fichier est ouvert en mode *input-output*.
- Cas particuliers :
  - Si le *filename* est précédé de **|**, il est interprété comme une commande de *pipe* qui recevra les données envoyées vers le fichier (par la fonction **print** par exemple).
  - Si le *filename* est suivi de **|**, il est interprété comme une commande qui enverra son *output* vers le fichier quand celui-ci sera lu.
- La fonction **open** renvoie une valeur différente de 0 en cas de succès et la valeur indéfinie (interprétée comme faux) en cas d'échec.

Remarque : Les *filehandles* STDIN, STDOUT, STDERR sont prédéfinis; ils représentent respectivement stdin, stdout et stderr.

Fermeture d'un fichier :

close[(]filehandle[)]

Ecriture dans un fichier ouvert :

print[(][filehandle] [list][)]

- Le *filehandle* par défaut est STDOUT.
- Noter l'absence de ponctuation entre le *filehandle* et la *list*.

Lecture dans un fichier ouvert :

read(filehandle**,** var**,** length [**,** offset])

- Cette fonction essaye de lire *length* octets dans le fichier et de les affecter à la variable scalaire *var*.
- *offset* indique la position en nombre d'octets où la lecture doit commencer. Si *offset* n'est pas spécifié, la lecture commence à la position en cours.
- La fonction renvoie le nombre d'octets effectivement lus; 0 si la position en cours était la fin de fichier.

Exemple : recopier le fichier f1.dat à la suite de f2.dat :

```
open(F1,"f1.dat") || die "Can't open f1.dat: $!\n"; # 1
open(F2,">> f2.dat");
while (read(F1, $buffer, 500)) {print F2 $buffer;}
close F1;
close(F2);
```

La fonction **die** (voir ligne # 1 dans l'exemple précédent) écrit la liste de ses arguments sur stderr et arrête l'exécution du script. La variable spéciale $! donne le code ou le message d'erreur selon que le contexte est numérique ou *string*.

L'instruction suivante, qui utilise l'*input operator* **<>**, permet aussi de lire un fichier ouvert :

var**=<**filehandle**>**

- L'*input operator* reconnait le caractère *newline* comme une marque de fin de ligne.
- La partie du fichier qui est lue dépend du contexte : si la variable *var* est scalaire, la ligne en cours est lue; si la variable *var* est un tableau, tout le reste du fichier est lu, chaque ligne étant affectée à un élément du tableau.
- La valeur retournée par l'*input operator* comprend le caractère *newline*.
- Quand la fin de fichier est atteinte, l'*input operator* renvoie la valeur indéfinie qui est interprétée comme *faux*.
- Dans la condition de fin d'une boucle **while**, l'*input operator* utilisé sans "*var*=" affecte sa valeur à la variable spécial $_.

Exemples :

1. Afficher le contenu du fichier f1.data :

```
open(F1,"f1.data");
while(<F1>) {print $_;}
```

2. Affecter le résultat de la commande Unix **ls -l** au tableau @T :

```
open(INPUT_PIPE,"ls -l |");
@T=<INPUT_PIPE>;
```

3. Lire des données sur stdin. Exemple :

```
print 'Entrez la valeur de X : ';
$X=<STDIN>;
```

13.7 Arguments

Les arguments transmis à un *script* Perl sont récupérés dans le tableau prédéfini @ARGV. Exemple : si l'on appelle comme suit le *script* myscript,

```
myscript -a -b arg1 -c arg2
```

le tableau @ARGV contiendra 5 *strings* : '-a', '-b', 'arg1', '-c' et 'arg2'; $#ARGV sera égal à 4 (n° du dernier élément); @ARGV, interprété dans un contexte scalaire, contiendra 5 (nombre d'éléments).

Lorsque des noms de fichiers sont transmis comme arguments, le *filehandle* prédéfini ARGV permet de traiter le contenu de ces fichiers. Exemple : dans le *script* suivant, les arguments qui ne sont pas des noms de fichiers du répertoire en cours sont éliminés du tableau @ARGV; ensuite les fichiers dont le nom est resté dans @ARGV sont lus et écrits sur stdout :

```
#! /usr/local/bin/perl
$N=@ARGV;
while ($I<$N)
  { $A=$ARGV[$I];
    print $A;
    if (-T $A)
        { print " is a text file\n";
          $I+=1;
        }
    else
        { print " is not a text file\n";
          splice(@ARGV,$I,1); # Remove element number $I
          $N-=1;
        }
  }
if ($N > 0)
  { while (<ARGV>) # 1
      { print $_;
      }
  }
```

Remarques :

1. L'argument par défaut de l'*input operator* est ARGV. En # 1, on aurait donc pu écrire *while (<>)*

2. L'instruction # 1 permet de lire toutes les lignes de tous les fichiers spécifiés comme arguments. En l'absence d'arguments, <ARGV> lit stdin.

13.8 Options

Il y a plusieurs façons de traiter les options (ou *switches*) transmis à un *script* Perl. La façon la plus complète est celle du sous-programme getopts.pl de la librairie Perl. Pour pouvoir faire appel à un sous-programme de la librairie Perl, il faut placer la commande suivante dans le *script* :

```
require "getopts.pl";
```

L'instruction pour appeler le sous-programme est

```
&Getopts(string)
```

Le *string* passé comme argument à Getopts est la liste des options autorisées. Chaque option est représentée par un seul caractère. Si une option est suivie du caractère deux points (**:**), l'argument qui suit est pris comme valeur de cette option. A chaque option *x* est associée une variable $opt_x qui reçoit la valeur de l'argument correspondant ou 1 s'il s'agit d'une option sans argument. Pour une option non transmise, $opt_x a comme valeur *undefined*. Dans l'appel du *script*, les options doivent être précédées du signe **-** et être placées avant tout autre argument. Après avoir initilisé la variable $opt_x, le programme Getopts retire l'option x du tableau des arguments @ARGV.

Exemple : le *script* suivant

```
#! /usr/local/bin/perl
require 'getopts.pl';
print "ARGV=@ARGV \n";
&Getopts('abc:d:e:g:');
print "opt_a= $opt_a \n";
print "opt_b= $opt_b \n";
print "opt_c= $opt_c \n";
print "opt_d= $opt_d \n";
print "opt_e= $opt_e \n";
print "opt_f= $opt_f \n";
print "opt_g= $opt_g \n";
print "ARGV=@ARGV \n";
```

appelé comme suit :

```
myscript -a -c 2 -x -dHello -e -f other -g
```

donne

```
ARGV=-a -c 2 -x -dHello -e -f other -g
Unknown option: x
opt_a= 1
opt_b=
opt_c= 2
opt_d= Hello
opt_e= -f
opt_f=
opt_g=
ARGV=other -g
```

13.9 Sous-routines

Un *script* Perl peut comprendre des sous-routines. Une sous-routine se définit comme suit :

sub name block

Rappelons qu'un *block* est une suite d'instructions entre accolades. Une définition de sous-routine peut être placée n'importe où dans un *script*. Une sous-routine est appelée par l'opérateur **&** :

&name [(args)]

- Les arguments *args* transmis à une sous-routine sont récupérés dans le tableau spécial @_.
- Une sous-routine peut être utilisée comme une fonction en l'invoquant comme partie d'une expression.
- La valeur retournée par une sous-routine peut être de type scalaire ou tableau. Cette valeur est celle de l'expression *exp* spécifiée dans une instruction **return** de la forme suivante

return exp

ou, de la dernière expression évaluée à défaut d'instruction **return**.

Remarque : le tableau @_ contient des références aux arguments. La modification d'un élément de ce tableau modifie l'argument correspondant. La fonction **local** permet de définir des variables locales au block dans lequel elle est spécifiée. Dans une sous-routine, la fonction **local** peut être employée pour définir des variables qui joueront le rôle de paramètres vers lesquels le transfert des arguments s'effectue par valeur. Les variables p1, p2, p3 jouent ce rôle dans l'exemple suivant :

```
$v=1;
&MySub ('Hello', $v, 3+$v);

sub MySub
  { local($p1, $p2, $p3)=@_;
    ...
  }
```

13.10 Remarque finale

Bon nombre de possibilités n'ont pas été évoquées dans cette introduction à Perl. On peut citer principalement :

- Traitement de données tabulaires et *reporting*.
- Traitement de fichiers à enregistrements de longueur fixe.
- Récursion.
- Accès aux répertoires.
- Définition de *packages*. Un *package* est un ensemble de sous-routines qui se partagent la même table de symboles.

Pour une description complète de Perl, voir WALL91 «Annexe A. Bibliographie» à la page 99.

14.1 Fortran

14.1.1 Compilation

xlf [options] dgfiles

xlf compile les dgfiles, *link edit* les fichiers objets produits et crée un fichier exécutable.

Les *dgfiles* peuvent être :

- des fichiers de code source Fortran de nom postfixé par .f; ils seront pris en charge par le compilateur Fortran;
- des fichiers de code source Assembler de nom postfixé par .s; ils seront pris en charge par l'assembleur (**as**);
- des fichiers objets de nom postfixé par .o; ils seront pris en charge par le *linkage editor* (**ld**).

Le compilateur peut produire deux types de fichiers :

- un fichier objet par fichier source, de même nom que le source mais postfixé par .o;
- un fichier listing par fichier source, de même nom que le source mais postfixé par .lst.

Le *linkage editor* appelé par **xlf** produit un fichier exécutable dont le nom est, par défaut, a.out.

Les éventuels messages d'erreur sont envoyés sur stderr et dans les fichiers listings.

Les principales options sont énumérées dans le Tableau 4 et le Tableau 5 à la page 54. Ces options peuvent être spécifiées dans la commande **xlf** ou, pour les options du compilateur, dans une instruction @PROCESS placée dans le code source avant la première instruction d'une routine Fortran. Le format de l'instruction @PROCESS est le suivant :

```
@PROCESS option1 [, option2 ...]
```

Les options définies dans une instruction @PROCESS ne s'appliquent qu'à la routine qui suit directement.

Option commande	Option @PROCESS	Défaut	Description
-U	[NO]MIXED	NOMIXED	Avec MIXED, Fortran distingue les majuscules des minuscules.
-O	[NO]OPT	NOOPT	Optimisation.
-q charlen=n	CHARLEN(n)	CHARLEN(500)	Longueur maximum des constantes et variables de type caractère (1 <= n <= 32767)
-I dir			Chemin d'accès à un répertoire qui contient les fichiers qui font l'objet d'instructions INCLUDE. Utile pour les instructions INCLUDE qui ne précisent pas le chemin d'accès.
-C	[NO]CHECK	NOCHECK	Vérifie que les indices des tableaux et expressions de type caractère restent dans les limites définies.
-q [no]recur	[NO]RECUR	NORECUR	Autorise les appels récursifs de sous-programmes et fonctions.
-q [no]extchk	[NO]EXTCHK	NOEXTCHK	Vérifie la concordance des types dans les COMMONs et entre arguments et paramètres au niveau du *link edit*.
-g	[NO]DBG	NODBG	Permet d'utiliser le *debugger* **dbx**.
-q [no]source	[NO]SOURCE	NOSOURCE	Produit le listing source du compilateur.

Option commande	Option @PROCESS	Défaut	Description
-q [no]xref -q xref=full	[NO]XREF XREF=FULL	NOXREF	Produit une table des références aux identificateurs. Si FULL est spécifié, les identificateurs définis mais non utilisés seront aussi repris.
-q [no]attr -q attr=full	[NO]ATTR ATTR=FULL	NOATTR	Produit une table des attributs des identificateurs. Si FULL est spécifié, les identificateurs définis mais non utilisés seront aussi repris.
-q [no]listopt	[NO]LISTOPT	NOLISTOPT	Indique l'état des options dans le listing source.

Tableau 4. Principales options du compilateur

Option	Défaut	Description
-c	*Link edit*	-c indique que le *linkage editor* ne doit pas être appelé.
-o gfile	a.out	Indique le nom du fichier exécutable produit par le *linkage editor*.
-l key	Librairies du fichier de configuration /etc/xlf.cfg	Permet d'indiquer les librairies à utiliser. Les librairies ont des noms de la forme lib*key*.a.
-L dir	Répertoires standards.	Permet d'indiquer les répertoires contenant les librairies définies par -l key.

Tableau 5. Principales options du *linkage editor*

Exemple (les deux commandes suivantes sont équivalentes) :

```
xlf -q source -q listopt -C -g -o myprog.out myprog.f mysp1.f mysp2.o
xlf -q source,listopt    -Cg   -o myprog.out myprog.f mysp1.f mysp2.o
```

Ces commandes

- compilent les fichiers myprog.f et mysp1.f en insérant du code pour vérifier que les indices de tableaux et des expressions de type caractère restent dans les limites définies (option -C);
- *link-edit* le résultat de cette compilation avec le fichier objet mysp2.o;
- produisent un programme exécutable myprog.out (option -o) qui pourra être analysé par le *debugger* **dbx** (option -g), un listing de compilation (option -q source) avec l'état des différentes options (option -q listopt).

Remarque : La librairie BLAS (*Basic Linear Algebra Subroutines*) fait partie des librairies installées en même temps que le compilateur XL Fortran. On peut y accéder en spécifiant l'option **-l blas** dans la commande **xlf**. Les routines BLAS sont documentées dans le **man**; voir aussi «Annexe A. Bibliographie» à la page 99, IBM SC23-2205.

14.1.2 Entrées-sorties

Par défaut, toutes les unités d'entrées-sorties de Fortran sont préconnectées :

- 0 à stderr,
- 5 à stdin,
- 6 à stdout,
- n (pour n différent de 0, 5 et 6) à un fichier de nom fort.*n*.

Toute unité n, sauf l'unité 0, peut être réaffectée au moyen d'une instruction Fortran OPEN de la forme

```
OPEN (n, file='dfile' [,...])
```

14.1.3 Exécution

Pour exécuter un programme, si l'on a la permission d'exécution (voir «Sécurité au niveau des fichiers» à la page 19), il suffit de taper le nom du fichier exécutable. Exemple :

```
myprog.out 2> myprog.err
```

Dans cet exemple, la sortie stderr (messages d'erreur) est redirigée vers le fichier myprog.err.

14.1.4 Debugging

Exemple d'utilisation du *debugger* **dbx** (les réponses du système sont écrites en *italique*; les commandes de l'utilisateur sont en caractères **gras**) :

dbx myprog.out # Appel du *debugger*
dbx version 3.1 for AIX
Type 'help' for help
reading symbolic information ...
(dbx) **stop in mysp2** # Définit un *break point* au début de mysp2
[1] stop in mysp2
(dbx) **run** # Démarre l'exécution du programme
[2] stopped in mysp2 at line 3 in file "mypsp2.f"
(dbx) **stop at 45** # Définit un *break point* à la ligne 45 de mysp2
[2] stop at "mypsp2.f":45
(dbx) **skip** # Continue jusqu'au *break point* suivant
[2] stopped in mysp2 at line 45 in file "mypsp2.f"
(dbx) **print x, vect(5), tab** # Affiche le contenu des variables
...
(dbx) **quit** # Quitte le *debugger*

Remarque : Il existe une version X-Windows du *debugger* appelable par **xde**.

14.2 C

Compilation

xlc [options] dgfiles

xlc (ou **cc** ou **c89**) précompile et compile les dgfiles, *link edit* les fichiers objets produits et crée un fichier exécutable.

Les *dgfiles* peuvent être :

- des fichiers de code source C de nom postfixé par .c; ils seront précompilés et pris en charge par le compilateur C;
- des fichiers précompilés de nom postfixé par .i; ils seront pris en charge par le compilateur C;
- des fichiers de code source Assembler de nom postfixé par .s; ils seront pris en charge par l'assembleur (**as**);
- des fichiers objets de nom postfixé par .o; ils seront pris en charge par le *linkage editor* (**ld**).

Le compilateur peut produire quatre types de fichiers :

- un fichier précompilé par fichier source .c, de même nom que le source mais postfixé par .i;
- un fichier objet par fichier source, de même nom que le source mais postfixé par .o;
- un fichier listing par fichier source, de même nom que le source mais postfixé par .lst;
- un *target file* par fichier source .c ou .i de même nom que le source mais postfixé par .u; ces fichiers peuvent être utiles dans la commande **make** (voir «Annexe A. Bibliographie» à la page 99).

Le *linkage editor* appelé par **xlc** produit un fichier exécutable dont le nom est, par défaut, a.out.

Les éventuels messages d'erreur sont envoyés sur stderr et dans les fichiers listings.

Les principales options de compilation sont énumérées dans le Tableau 6 à la page 56. Les options du *linkage editor* sont celles présentées dans le Tableau 5 à la page 54.

Les options de compilation peuvent être spécifiées dans un fichier de configuration xlc.cfg (voir «Annexe A. Bibliographie» à la page 99), dans la commande **xlc** ou dans le programme C lui-même au moyen de la directive **#pragma options** (voir «Annexe A. Bibliographie» à la page 99).

Exemple :

```
xlc -q source -q listopt -g -lm -o myprog.out myprog.c mysp1.c mysp2.o
```

Cette commande

- compile les fichiers myprog.c et mysp1.c;
- *link edit* le résultat de cette compilation avec le fichier objet mysp2.o et des modules de la librairie mathématique libm.a (option -lm) [9];
- produit un programme exécutable myprog.out (option -o) qui pourra être analysé par le *debugger* **dbx** (option -g), un listing de compilation (option -q source) avec l'état des différentes options (option -q listopt).

L'exécution et le *debugging* d'un programme C peuvent se faire comme indiqué aux paragraphes «Exécution» à la page 54 et «Debugging» à la page 54.

[9] La liste des *AIX Operating System Libraries* est donnée dans IBM SC23-2205. Voir «Annexe A. Bibliographie» à la page 99.

Option commande	Option #pragma	Défaut	Description
-qlanglvl=l	LANGlvl=l	l=ansi pour xlc et c89 l=extended pour cc	Niveau de langage : ansi, saa, saal2 ou extended
-O			Optimisation.
-I dir			Chemin d'accès à un répertoire qui contient les fichiers qui font l'objet de directives #INCLUDE. Utile pour les directives #INCLUDE qui ne précisent pas le chemin d'accès.
-g			Permet d'utiliser le *debugger* **dbx**.
-q [no]source	[NO]SOURCE	NOSOURCE	Produit le listing source du compilateur.
-q [no]xref -q xref=full	[NO]XREF XREF=FULL	NOXREF	Produit une table des références aux identificateurs. Si FULL est spécifié, les identificateurs définis mais non utilisés seront aussi repris.
-q [no]attr -q attr=full	[NO]ATTR ATTR=FULL	NOATTR	Produit une table des attributs des identificateurs. Si FULL est spécifié, les identificateurs définis mais non utilisés seront aussi repris.
-q [no]listopt	[NO]LISTOPT	NOLISTOPT	Indique l'état des options dans le listing source.

Tableau 6. Principales options du compilateur C

14.3 C++

xlC [options] dgfiles

xlC, avec C majuscule, est comparable à la commande **xlc** si ce n'est qu'elle appelle le compilateur C++. Les fichiers de code source C++ doivent être postfixés par .C (C majuscule) au lieu de .c. Les options du Tableau 6 sont aussi utilisables en C++.

14.4 Pascal

Compilation

xlp [options] dgfiles

xlp compile les dgfiles, *link edit* les fichiers objets produits et crée un fichier exécutable.

Les *dgfiles* peuvent être :

- des fichiers de code source Pascal de nom postfixé par .p; ils seront pris en charge par le compilateur Pascal;
- des fichiers de code source Assembler de nom postfixé par .s; ils seront pris en charge par l'assembleur (**as**);
- des fichiers objets de nom postfixé par .o; ils seront pris en charge par le *linkage editor* (**ld**).

Le compilateur peut produire deux types de fichiers :

- un fichier objet par fichier source, de même nom que le source mais postfixé par .o;
- un fichier listing par fichier source, de même nom que le source mais postfixé par .lst.

Le *linkage editor* appelé par **xlp** produit un fichier exécutable dont le nom est, par défaut, a.out.

Les éventuels messages d'erreur sont envoyés sur stderr et dans les fichiers listings.

Pour les options de compilation, voir IBM SC09-1326 («Annexe A. Bibliographie» à la page 99). Les options du *linkage editor* sont celles présentées dans le Tableau 5 à la page 54.

14.5 Utilitaires d'aide à la programmation

La description des utilitaires suivants peut être consultée au moyen de la commande **man** (voir «Help» à la page 21).

- Programmation en C : **lint**, **cf**, **cxref**, **make**, **cflow**.
- Gestion de modules objets : **ar, ld, nm, size, lorder, strip**.
- Assistance au développement de programmes : **yacc**.

15.0 Logiciels d'application disponibles en AIX/6000

15.1 ESSL

ESSL est la *Engineering and Scientific Subroutine Library* d'IBM.

Pour qu'un programme Fortran puisse appeler des routines de cette librairie, il suffit de spécifier l'option **-l essl** dans la commande **xlf** (voir «Fortran» à la page 53).

Exemple : le programme suivant appelle deux routines ESSL, une routine d'inversion de matrice (SGEICD) et une routine de produit matriciel (SGEMUL) :

```
      program   TESTESSL
c
      implicit  none
      integer   ND, NAUX
      parameter (ND=100, NAUX=33*ND)
      real      X(ND, ND), Y(ND, ND), XY(ND, ND), AUX(NAUX)
      real      R, DETX, DD(2)
      integer   M, N, I, J
c
      read (9,*) N
      if (N > ND) then
        write (6,*) ' Dimensions insuffisantes'
        stop
      endif
      do I=1,N
        read (9,*) (X(I,J), J=1,N)
        do J=1,N
          Y(I, J) = X(I, J)
        enddo
      enddo
      M=min(N,10)
      write (6,*) ' Donnees - X='
      call PUTIT (X, ND, ND, M, M)
c
      call SGEICD (Y, ND, N, 2, R, DD, AUX, NAUX)
      write (6,*) ' Inverse - Y='
      call PUTIT (Y, ND, ND, M, M)
      DETX = DD(1) * 10.** DD(2)
      write (6,*) ' Determinant - DETX=', DETX
c
      call SGEMUL (X, ND, 'N', Y, ND, 'N', XY, ND, N, N, N)
      write (6,*) ' Verification - XY='
      call PUTIT (XY, ND, ND, M, M)
c
      end
c-------------------------------------
      subroutine PUTIT (T, MT, NT, M, N)
      implicit none
      integer MT, NT, M, N, I, J
      real T(MT, NT)
      do I=1,M
        write(6,'(1X, 6G13.5)') (T(I, J), J=1,N)
      enddo
      return
      end
```

Pour compiler ce programme :

```
xlf -O -o testessl.out -l essl -q source testessl.f
```

Pour l'exécuter, créer un fichier fort.9 qui contient les données et taper **testessl.out**.

15.2 DISSPLA

DISSPLA est un logiciel graphique distribué par Computer Associates. Il se présente sous la forme d'un ensemble de routines réparties dans des librairies et appelables en Fortran ou en C.

Les *drivers* installés sont les suivants :

- X-Windows-112
- PostScript-All
- PostScript-AllC
- HP-Lasertjet2
- HP-LasertjetP
- Metafile DISP
- Metafile HPGL
- Metafile CGM-B
- Metafile CGM-C
- Metafile CGM-T

La documentation relative aux *drivers* DISSPLA se trouve dans le répertoire /usr/local/disspla11/doc. (Voir aussi la documentation écrite, «Annexe A. Bibliographie» à la page 99).

Les procédures shell suivantes sont disponibles :

. disspla

définit l'environnement nécessaire. Ne pas oublier de mettre un point suivi d'un espace en tête de la commande (voir la raison au paragraphe «Procédures shell» à la page 29).

dis77links [options] dgfiles

compile les *dgfiles*, les *link edit* et crée un fichier exécutable. Les options qui peuvent être spécifiées dans cette commande sont celles du Tableau 4 à la page 53 et du Tableau 5 à la page 54.

Exemple : le programme suivant utilise les *drivers* X-Windows et PostScript.

```
      program MYANY
C DISSPLA 11 - AIX RS/6000
C
C Initialize driver
      call ANYDEV(XAXIS,YAXIS)
C
```

```
C Call user graphic routine(s)
      call area2d(XAXIS,YAXIS)
      call MYPLOT1
      call endpl(0)
C
      call area2d(XAXIS,YAXIS)
      call MYPLOT2(XAXIS,YAXIS)
      call endpl(0)
C
C Sign off the device
      call donepl
C
      stop
      end


      subroutine ANYDEV (XAXIS, YAXIS)
C Needed for IOMGR
      dimension IBUF(16)
C Needed for ldev
      dimension IARGRY(10),ARGRY(10)
      data IARGRY /112, 0,0,0,0, 0, 2, 1, 0, 0/,NARG/10/
      equivalence (IARGRY,ARGRY)
      equivalence (ARGRY(2),XORIG),(ARGRY(3),YORIG)
      equivalence (ARGRY(4),XPAGE),(ARGRY(5),YPAGE)
C
      parameter (KOUNT=3)
      character *20 NAME(KOUNT) /
     x '1 X-Windows          '   ,
     x '2 PostScript         '   ,
     x '0 Exit               '   /
      data INUNIT/5/, IOUT/6/
C
C Dynamic allocation of system virtual memory
      call sysbuf
C
      write (IOUT,101)
101   format (' DISSPLA - List of devices'//)
      do I=1,KOUNT
        write (IOUT,103) NAME(I)
103     format (1x,a20)
      enddo
      write (IOUT,104)
104   format (' Your choice?')
100   continue
        read (INUNIT,*) IDEV
        if (IDEV .eq. 1) then
           XPAGE=9.
           YPAGE=6.
           XORIG=2.5
           YORIG=1.5
           call ldev ('XWINDOWS_112_GENERAL',IARGRY,NARG,ISTAT)
        elseif (IDEV .eq. 2) then
           XPAGE=7.99
           YPAGE=10.78
           WIDTH=0.0139
           IBUF(1)=5
           CALL IOMGR(IBUF,-102)
           call pscrpt (XPAGE, YPAGE, WIDTH)
        elseif (IDEV .eq. 0) then
           stop
        else
           go to 100
        endif
C
      call page (XPAGE, YPAGE)
      XAXIS = XPAGE / 1.4
      YAXIS = YPAGE / 1.4
C
      return
      end


      SUBROUTINE MYPLOT1
C                  DISSPLA 11
C CA-DISSPLA USER'S MANUAL, PART A, P 7-3, EXAMPLE 3
      DIMENSION X(100),Y(100),R(100)
C SET UP PLOT
      CALL HEADIN('GRAF WITH A GRIDS - DISSPLA 11 $',100,1.5,1)
      CALL XNAME('X AXIS WITH GRAF TYPES$',100)
      CALL YNAME('Y AXIS WITH GRAF TYPES$',100)
      CALL THKFRM(.01)
C (LEVEL 2 TO 3)
      CALL GRAF(0.,0.1,1.,-1.,0.4,1.0)
C CALL THE GRID OPTION
      CALL GRID(4,2)
C SET UP CURVE DATA VALUES
      PI=3.14159
      DO 40 J=1,100
        X(J)=FLOAT(J)/100.
        Y(J)=EXP(-3.*X(J))*SIN(X(J)*8.*PI)
        R(J)=EXP(-3.*X(J))*COS(X(J)*8.*PI)
40    CONTINUE
C DRAW CURVES
      CALL CURVE(X,Y,100,0)
      CALL CURVE(X,R,100,0)
C
      RETURN
      END


      SUBROUTINE MYPLOT2(XAXIS,YAXIS)
C                  DISSPLA 11
C CA-DISSPLA USER'S MANUAL, A-P 7-4, EXAMPLE 5
      DIMENSION X(100),Y(100),R(100)
C SET UP PLOT
      CALL SETCLR('RED')
      CALL HEADIN('LOG GRID USING YLOG - DISSPLA 11 $',100,1.25,1)
      CALL SETCLR('CYAN')
      CALL XNAME('X AXIS IS GRAF TYPES$',100)
      CALL YNAME('Y AXIS IS LOG TYPES$',100)
      CALL THKFRM(.02)
C (LEVEL 2 TO 3)
      XORIG=-1.5
      YORIG=0.01
      XSTEP=3./XAXIS
      YCYCLE=YAXIS/2.
      CALL YLOG(XORIG,XSTEP,YORIG,YCYCLE)
C NOTE USE OF GRID OPTION
      CALL SETCLR('BLUE')
      CALL GRID(2,5)
C SET UP CURVE DATA VALUES
      PI=3.14159
      DO 40 J=1,100
        X(J)=FLOAT(J)/100.
        Y(J)=EXP(-3.*X(J))*SIN(X(J)*8.*PI)
        R(J)=EXP(-3.*X(J))*COS(X(J)*8.*PI)
40    CONTINUE
C DRAW CURVES
      CALL SETCLR('GREEN')
      CALL CURVE(Y,X,100,0)
      CALL SETCLR('YELLOW')
      CALL CURVE(R,X,100,0)
C
      RETURN
      END
```

La marche à suivre pour compiler et exécuter ce programme est la suivante :

1. Définir l'environnement DISSPLA :

   ```
   . dissplа
   ```

2. Compiler et créer un fichier exécutable (on suppose que les routines qui composent le programme précédent se trouvent dans les fichiers myany.f, anydev.f, myplot1.f et myplot2.f) :

   ```
   dis77links -o myany.out myany.f anydev.f myplot1.f myplot2.f
   ```

3. Exécuter le programme :

   ```
   myany.out
   ```

4. Si le *driver* PostScript a été invoqué, un fichier std00001.dat a été créé. Ce fichier PostScript peut être imprimé comme indiqué au paragraphe «Impression» à la page 15.

Remarque : Pour choisir un *driver* à l'exécution, on peut aussi utiliser la routine **pdev** de Dissplа. Utilisée comme suit

```
CALL PDEV('',IRC)
```

cette routine permet de faire un choix parmi tous les *drivers* connus de Dissplа.

15.3 TeX et LaTeX

TeX est un système de traitement de texte. Il a été développé par Donald E. Knuth de l'Université de Stanford. Il est particulièrement apprécié pour la composition de documents scientifiques.

LaTeX, mis au point par Leslie Lamport, peut être considéré comme un complément à *TeX*. C'est un ensemble de macro-instructions qui font appel à des commandes de *TeX* et qui en facilitent l'utilisation.

Le présent paragraphe doit être considéré comme le *local guide* de *LaTeX*. Il se borne à décrire brièvement les commandes permettant d'utiliser *LaTeX* sur des machines gérées directement ou indirectement par le SEGI. Pour la description complète des systèmes *TeX* et *LaTeX*, voir «Annexe A. Bibliographie» à la page 99.

Un document *LaTeX* se compose avec un éditeur quelconque. Il comprend du texte et des commandes de mise en page. On supposera que le fichier d'entrée, c'est-à-dire le fichier qui contient le texte et les commandes *LaTeX* s'appelle *doc.tex*. (Le suffixe *.tex* est obligatoire). Lorqu'un tel fichier a été créé, il faut le *compiler* pour procéder à la mise en page avant de pouvoir le visualiser à l'écran ou l'imprimer.

15.3.1 Préalable

Avant toute autre opération, il est nécessaire de définir les variables d'environnement en exécutant la commande suivante :

. tex.env

Ne pas oublier de mettre un point suivi d'un espace avant la commande (voir la raison au paragraphe «Procédures shell» à la page 29).

15.3.2 Compiler

latex doc

Plusieurs fichiers peuvent être créés par la commande **latex** :

- doc.dvi : résultat de la compilation dans un format *device independent*.
- doc.log : *Transcript file*, contient des statistiques relatives au déroulement de la compilation et les éventuels messages d'erreur.
- doc.aux : fichier auxiliaire dans lequel sont enregistrées des informations qui seront réutilisées lors de compilations ultérieures.
- doc.toc : fichier auxiliaire utilisé pour créer une table des matières.
- doc.idx : fichier auxiliaire utilisé pour créer un index.
- doc.glo : fichier auxiliaire utilisé pour créer un glossaire.
- doc.bbl : fichier auxiliaire utilisé pour créer une liste de références bibliographiques.
- doc.lof : fichier auxiliaire utilisé pour créer une liste des figures.
- doc.lot : fichier auxiliaire utilisé pour créer une liste des tables.

Remarque : Il est parfois nécessaire de compiler le même document deux fois successivement; la première compilation enregistre les renseignements nécessaires aux diverses références (renvois, table des matières, ...) dans les fichiers auxiliaires; la seconde utilise ces renseignements pour introduire l'information désirée dans le texte.

15.3.3 Visualiser et imprimer

Visualiser le fichier *doc.dvi* :

xdvi [options] doc

Pour les options, voir la commande **man xdvi**.

Imprimer *doc.dvi* en PostScript :

dvips [options] doc[.dvi]

La commande **dvips** convertit le fichier *doc.dvi* en PosScript.

Principales options :

- -o dfile : définit la destination du fichier PostScript. Défaut : imprimante par défaut.
- -tlandscape : pour une impression en mode *paysage*. Défaut : mode *portrait*.

Les options par défaut sont définies dans un fichier *config.ps*. L'utilisateur peut définir ses propres options par défaut dans un fichier *.dvipsrc*. La forme de ce fichier est la même que celle du fichier *config.ps*.

Exemple : pour définir l'imprimante à utiliser, introduire la ligne suivante dans le fichier *.dvipsrc* :

```
o !lpr -Pqueue[:device]
```

Revoir «Impression» à la page 15.

Remarque : Si le nom du fichier, *doc*, contient des points, le suffixe *.dvi* doit être spécifié.

Visualiser le fichier PostScript *doc.ps* :

ghostview [options] doc.ps

L'utilitaire **ghostview** est parfois appelé **gv**.

15.3.4 Bibliographie

Pour constituer la bibliographie, deux techniques peuvent être utilisées : la première consiste à introduire la bibliographie dans le texte (voir LAMP86, «Annexe A. Bibliographie» à la page 99); la seconde consiste à utiliser une base de données bibliographiques.

Cette seconde technique exige l'emploi de la commande **bibtex** qui est décrite ci-dessous. Pour le format d'une base de données bibliographique, voir LAMP86.

On supposera que la base de données est constituée et s'appelle *bibli.bib*. La marche à suivre pour insérer la bibliographie dans le document *LaTeX* est la suivante :

- Insérer les commandes *bibliographystyle{...}* et *bibliography{bibli}* dans le document *LaTeX*.
- Compiler le document pour qu'une référence à *bibli.bib* soit insérée dans le fichier *doc.aux*.
- Appeler le programme *bibtex* au moyen de la commande suivante :

bibtex doc

  La commande **bibtex** crée les deux fichiers suivants :

  - doc.bbl : fichier auxiliaire des références bibliographiques;
  - doc.blg : fichier contenant les éventuels messages d'erreurs.

- Recompiler deux fois le document *doc.tex*.

15.3.5 Index

La démarche pour la création d'un index est la suivante :

- Placer des commandes *index{mot}* dans le fichier *doc.tex* pour les mots qui doivent être repris dans l'index.
- Lorsque la commande *makeindex* est placée dans le préambule du document, la compilation produit un fichier *doc.idx*.
- Ce fichier doit être traité par la commande **makeindex** avant de pouvoir être inclus dans le document pour produire un index :

makeindex < doc.idx > doc.idx.tex

  Cette commande exploite l'information contenue dans le fichier *doc.idx* pour créer un fichier *doc.idx.tex*

- Introduire la commande *input{doc.idx}*[10] dans le fichier *doc.tex* à l'endroit où l'on souhaite voir apparaître l'index.
- Recompiler le document.

Remarque : La commande **delatex** permet d'obtenir la liste de tous les mots d'un document *LaTeX*. Utilisée comme suit :

delatex doc.tex | sort -uf doc.words

elle donne, dans *doc.words*, la liste des mots de *doc.tex* triés dans l'ordre alphabétique, avec une seule occurence de chacun des mots. Ceci peut être utile pour repérer les mots qui devront figurer dans l'index.

15.4 Reduce

Reduce est un système de programmation algébrique appelable via la commande

reduce

Exemple d'instruction Reduce :

```
(x+y)**2;
```

[10] Dans la commande *input*, le suffixe *.tex* est implicite.

donne $x^2 + 2xy + y^2$.

L'instruction **quit;** permet de sortir de Reduce.

Ceux qui disposent de X-Windows (voir «AIX X-windows» à la page 75), ont intérêt à utiliser l'interface X via la commande

```
xr &
```

L'interface *gnuplot*, qui permet d'obtenir des représentations graphiques de courbes ou de surfaces, est également disponible via la commande plot de reduce. Exemple :

```
plot(x**2*sin x, x=(-pi .. pi));
```

Le *REDUCE User's Manual* est disponible en format LaTeX dans le fichier /usr/local/reduce/doc/reduce.tex.

15.5 Mathematica

Mathematica est un système de programmation conçu pour les matématiques. Il est distribué par Wolfram Research. Il est accessible via la commande

```
math
```

Exemple d'instruction Mathematica :

```
Solve[ a x + b == c, x]
```

donne $x \rightarrow (\frac{c-b}{a})$.

L'instruction **Quit** permet de sortir de Mathematica.

Ceux qui disposent de X-Windows (voir «AIX X-windows» à la page 75), ont intérêt à utiliser le *X Front End* via la commande

```
mathematica &
```

Mathematica offre aussi un système de visualisation graphique très riche.

15.6 Matlab

Matlab (*MATrix LABoratory*) est un logiciel de calcul numérique et de visualisation. Il intègre l'analyse numérique, le calcul matriciel, et le graphisme. Les options additionnelles suivantes sont disponibles au SEGI :

- *Simulink*, simulation de systèmes dynamiques;
- *Control System*;
- *Signal Processing*;
- *Image Processing*.

Matlab est accessible via la commande

```
matlab
```

L'instruction **quit** permet de sortir de Matlab.

16.0 Parallélisme

Comme on l'a dit précédemment, l'IBM SP2 est une collection de processeurs RISC System/6000 connectés entre eux par un réseau local qui permet l'échange de données et la synchronisation des tâches. En plus d'un adaptateur ethernet, ce réseau peut comprendre, en option, un *high-performance switch adapter*, HPS, qui offre une largeur de bande supérieure et un temps de latence réduit. Cet adaptateur est disponible au SEGI.

Dans un système à mémoire distribuée, comme le SP2, où la mémoire et l'*adress space* sont locaux à chaque noeud, le partage des données peut être réalisé par la technique du *message passing*. Un programme parallélisé appelle donc des routines d'une librairie de *message passing* (MPL). Pour le HPS, ces routines, à leur tour, appelle des routines du *Communication Subsystem*, CSS, qui traite les communications entre noeuds. Il existe deux implémentations de la librairie CSS qui utilisent des protocoles différents : Internet (*ip*) et *User Space* (*us*). Les routines CSS peuvent être *linkées* statiquement ou dynamiquement.

Un programme parallélisé est lancé à partir d'un noeud, le *home node* et un certain nombre de tâches peuvent s'exécuter sur d'autres noeuds, les *remote nodes*. Les noeuds nécessaires à l'exécution du programme peuvent être alloués de façon spécifique par l'utilisateur au moyen d'un *hostfile* ou implicitement par le *POWERparallel System Resource Manager*. Il y a également la posibilité de déterminer comment les ressources allouées doivent être utilisées : partager ou non les *node's CPU* et les *node's adapters* (voir «Partition» à la page 66). Une utilisation du HPS en mode *us* est limitée à un seul utilisateur à la fois mais n'empêche pas une utilisation partagée du HPS en mode *ip* par d'autres utilisateurs à condition que les *node's CPU* ne soient pas dédicacés.

Les systèmes de programmation parallèle connectent généralement les *standard I/O* de chaque *remote node* de sorte que les tâches parallèles puissent communiquer avec le *home node*. On peut ainsi utiliser les techniques familières de redirection des I/O, de *pipe*, etc.

Si les fichiers utilisés par un programme (exécutables, données) sont partagés via NFS [11], le *Partition Manager* charge une copie des exécutables sur chaque noeuds de la partition et les fichiers de données sont accessibles à chaque tâche. Sinon, il faut les copier sur chaque noeud.

L'administrateur du système peut diviser les noeuds en *pools* séparés pour les affecter à des tâches particulières. Au SEGI, c'est le *pool* 0 qui regroupe les noeuds utilisés pour le calcul parallèle. La commande suivante permet d'obtenir des informations sur les *pools* et sur les programmes en cours d'exécution :

jm_status options

Options :

- -P donne des informations sur les *pools* de noeuds.
- -p *pool* donne des informations sur un *pool* particulier.
- -j donne des informations sur les programmes en cours d'exécution.
- -v, en complément à l'une des options précédentes, donne des informations plus détaillées.

Les logiciels de *message passing* disponibles au SEGI sont

- *IBM Parallel Environment* (PE),
- *IBM PVMe* (PVMe),
- *Public Domain PVM*, (PVM).

Le PE permet d'utiliser les différents adaptateurs, HPS et ethernet; de même que les protocoles *ip* et *us* sur le HPS. Il permet aussi, pour le HPS en mode *ip*, de dédicacer ou de partager l'adaptateur de chaque noeud et de dédicacer ou partager les CPUs.

Le PVMe utilise systématiquement le HPS en mode *us* et donne la possibilité de dédicacer ou partager les CPUs.

Le PVM peut utiliser l'ethernet ou le HPS en mode *ip*. PVM et PVMe sont compatibles au niveau du source.

Les différentes possibilités sont résumées dans le tableau suivant :

Adaptateur	Protocole	Logiciel	Adaptateur dédicacé (D) partagé (P)	CPU dédicacé (D) partagé (P)
ethernet	ip	PE	D/P	D/P
		PVM	P	P
HPS	ip	PE	D/P	D/P
		PVM	P	P
	us	PE	D	D/P
		PVMe	D	D/P
Remarque : D/P = dédicacé ou partagé selon option spécifiée par l'utilisateur. L'option par défaut est dédicacé.				

Tableau 7. Résumé des différentes possibilités

[11] Au SEGI, les répertoires des utilisateurs ainsi qu'un certain nombre de répertoires du système sont partagés via NFS.

Pour illuster l'emploi des différents systèmes, on considérera une application écrite en Fortran, de modèle MPMD (*Multiple Program Multiple Data*), composée de deux fichiers sources : paramat.master.f et paramat.slave.f. Le programme *master* est conçu pour lancer un certain nombre de tâches *slaves* auxquelles il enverra des données et dont il recueillera les résultats.

16.1 Parallel Environment

16.1.1 Compilation

mpxlf [xlf_options] [**-ip** | **-us**] **-o** executable sources.f

xlf_option utile : **-g** pour pouvoir utiliser des outils tels que le *debugger*.

Les options *-ip* et *-us* permettent d'indiquer l'implémentation de la librairie CSS qui doit être *linkée*. Si aucune de ces options n'est spécifiée, la librairie CSS sera *linkée* dynamiquement au moment de l'exécution; c'est la variable d'environnement MP_EUILIB (ou l'option *euilib*) qui indiquera l'implémentation choisie (voir «Environnement»). Avec PE, il est paticulièrement utile de pouvoir *linker* les routines CSS dynamiquement puisque PE permet l'emploi des deux protocoles *ip* et *us*. Le programme peut ainsi choisir l'implémentation de la *CSS library* qui convient au moment de l'exécution.

16.1.2 Partition

La partition, ensemble des noeuds utilisés par le programme, peut être définie dans un *hostfile* dans le répertoire en cours. Le *hostfile* doit contenir un noeud par ligne et autant de noeuds que le programme comprend de tâches. Si le nombre de tâches est supérieure au nombre de noeuds physiques disponibles, certains noeuds doivent être répétés. Exemple pour un nombre de tâches nt=5 et pour une partition de 3 noeuds parsw3, parsw4, parsw5 :

```
hostfile : parsw3
           parsw4
           parsw5
           parsw3
           parsw4
```

Ainsi, les noeuds parsw3 et parsw4 seront chacun utilisés pour 2 des 5 tâches.

Remarques :

1. Le nombre de tâches ne peut pas être supérieur au nombre de noeuds quand on utilise le HPS en mode *us*.
2. Si on ne fait pas référence à un *hostfile* au lancement d'un programme PE, la partition sera définie par le *System Resource Manager* qui cherchera à allouer un nombre de noeuds égal au nombre de tâches.
3. Chaque spécification de noeud dans le *hostfile* peut être suivie des options suivantes:

   d|s u|m

   d, *node adapter dedicated*, un seul programme peut utiliser l'adaptateur du noeud. C'est toujours le cas pour le HPS en mode us.
   s, *node adapter shared*, plusieurs programmes peuvent se partager l'adaptateur du noeud.
   u, *CPU unique*, un seul programme peut utiliser le CPU.
   m, *CPU multiple*, plusieurs programme peuvent utiliser le CPU.

16.1.3 Environnement

L'environnement peut être défini,

- avant le lancement du programme, par des variables d'environnement;
- au lancement du programme, par des options.

A chaque variable d'environnement *MP_xxx=valeur* correspond une option *-xxx valeur* dans la commande **poe** (voir «Exécution» à la page 67). Les options doivent être spécifiées si l'environnement n'a pas été défini au moyen des variables d'environnement. Si des variables d'environnement ont été définies et si des options sont aussi spécifiées, ces dernières ont priorité.

Les principales variables d'environnement sont présentées dans le Tableau 8.

Variable	Valeur	Commentaire
MP_PROCS	nt	Nombre de tâches.
MP_RESD	**yes**	Pour que le *System Resource Manager* puisse être utilisé. Il est nécessaire pour le HPS.
MP_HOSTFILE	dfile	Nom du fichier *hostfile* qui définit la partition.
MP_INFOLEVEL	n	Nombre compris entre 0 et 6 qui définit le niveau de gravité des messages écrit sur stderr. Un niveau souvent utilisé est 2. Voir «Annexe A. Bibliographie» à la page 99.
MP_EUILIB	**ip**|**us**	Protocole de communication Internet (*ip*) ou *User Space* (*us*).

Variable	Valeur	Commentaire
MP_EUIDEVICE	**css0|en0**	Adaptateur HPS (css0) ou ethernet (en0).
MP_PGMMODEL	**spmd|mpmd**	Modèle *Single Program Multiple Data* (spmd) ou *Multiple Programs Multiple Data* (mpdm).
MP_CMDFILE	dfile	fichier contenant la liste des programmes exécutables.
MP_RMPOOL	**0**	*Pool* des noeuds réservés aux programmes parallèles. Au SEGI, 0.

Tableau 8. PE : Principales variables d'environnement.

16.1.4 Exécution

Le programme peut être lancé à partir d'un noeud du *pool* parallèle [12] via la commande suivante :

poe [dfile] [options]

dfile est le nom d'un programme exécutable.

1. Si ni *dfile* ni *cmdfile* n'est spécifié, un dialogue interactif s'engage au cours duquel le système affiche successivement tous les noeuds de la partition et où l'utilisateur indique le programme associé à chacun de ces noeuds.
2. Si *dfile* est spécifié et pas de *cmdfile*, le même programme exécutable est lancé sur tous les noeuds de la partition. Cette façon de procéder convient pour un programme SPMD.
3. Si *dfile* n'est pas spécifié mais bien un *cmdfile*, les exécutables du *cmdfile* sont envoyés sur les noeuds correspondants de la partition. Cette façon de procéder convient pour un programme MPMD.

Les principales options de la commande **poe** sont celles qui correspondent aux variables d'environnement évoquées au paragraphe «Environnement» à la page 66.

16.1.5 Exemple

1. Compilation :

```
mpxlf -O -o paramat.master.exe paramat.master.f
mpxlf -O -o paramat.slave.exe paramat.slave.f
```

Pour les *xlf_options*, voir Tableau 4 à la page 53.

2. Création du *cmdfile* suivant, nommé *paramat.exelist* :

```
/home/segi/nihon/mype/paramat.master.exe
/home/segi/nihon/mype/paramat.slave.exe
/home/segi/nihon/mype/paramat.slave.exe
/home/segi/nihon/mype/paramat.slave.exe
```

3. Exécution : lancer la commande suivante à partir d'un des noeuds du pool parallèle:

```
poe -procs 4 -resd yes -euilib ip -euidevice css0 \
    -rmpool 0 -pgmmodel mpmd -infolevel 2 \
    -cmdfile paramat.exelist
```

16.2 PVM

16.2.1 Préalables

Chaque utilisateur doit avoir les sous-répertoires pvm3/bin/RS6K dans son *home directory* et, dans le sous-répertoire pvm3, un lien défini comme suit :

```
ln -s /usr/local/pvm3/lib lib
```

Le *hostfile* et les exécutables seront placés dans pvm3/bin/RS6K. De plus il faudra définir les variables d'environnement suivantes :

```
export PVM_ROOT=$HOME/pvm3
export PVM_ARCH=RS6K
```

16.2.2 Partition

Les noeuds de la partition peuvent être définis soit de façon interactive lors du lancement de PVM (voir «Lancement de PVM» à la page 68) soit dans un *hostfile*. Par exemple :

[12] Le *home node* doit obligatoirement être un noeud du *pool* parallèle. On peut y accéder par exemple via la commande suivante : *rlogin pars*. Voir «Procédures d'exploitation» à la page 95.

```
hostfile : pari3  ou  parsw3
           pari4      parsw4
           pari5      parsw5
           ...        ...
```

Avec pari*x*, l'ethernet sera utilisé. Avec parsw*x*, c'est le HPS qui sera utilisé.

16.2.3 Compilation

xlf [xlf_options] **-L/usr/local/pvm3/lib/RS6K **
-lfpvm3 -lgpvm3 -lpvm3 -o executable sources.f

Pour les *xlf_options*, voir Tableau 4 à la page 53.

16.2.4 Lancement de PVM

A partir d'un noeud du pool parallèle, le PVM peut être lancé de deux façons :

1. En interactif :

$HOME/pvm3/lib/pvm

On accède ainsi à la *console* du PVM qui permet d'entrer des commandes. La liste des commandes peut être obtenue en tapant **help**. Notamment, la commande **add** permet de définir la partition (dans ce cas le *hostfile* n'est pas utilisé). Exemple :

```
add parsw3
add parsw4
add parsw5
 ...
```

2. En arrière-plan :

$HOME/pvm3/lib/pvmd hostfile **&**

Pour arrêter PVM, démarrer la console si elle ne l'est pas et entrer la commande **halt**.

16.2.5 Exécution

Quand le PVM est lancé, le programme *master* peut être exécuté. Si le PVM a été lancé en interactif, on peut soit fermer la console PVM (commande **quit**) et lancer le programme, soit ouvrir une autre fenêtre pour y lancer le programme.

16.2.6 Remarque

PVM est accompagné d'un *debugger* interactif nommé xpvm. Quand le PVM est démarré, xpvm peut être appelé comme suit :

```
export DISPLAY=adresse du système d'affichage
export XPVM_ROOT=/usr/local2/xpvm
export TCL_LIBRARY=$XPVM_ROOT/src/tcl
export TK_LIBRARY=$XPVM_ROOT/src/tcl
$XPVM_ROOT/src/$PVM_ARCH/xpvm  &
```

xpvm est une application X qui ouvre une fenêtre comportant un menu dont les options *Tasks* - *Spawn* permettent de démarrer le programme *master*.

Remarque : Les opérateurs de redirections sont ignorés en xpvm.

16.2.7 Exemple

1. Compilation

```
xlf -O -L/usr/local/pvm3/lib/RS6K -lfpvm3 -lgpvm3 -lpvm3 \
    -o paramat3.master.exe paramat3.master.f
xlf -O -L/usr/local/pvm3/lib/RS6K -lfpvm3 -lgpvm3 -lpvm3 \
    -o paramat3.slave.exe paramat3.slave.f
```

2. Lancement de PVM après avoir créé le *hostfile* qui convient :

```
export PVM_ROOT=$HOME/pvm3
export PVM_ARCH=RS6K
$HOME/pvm3/lib/pvmd hostfile &
```

3. Exécution :

```
paramat.master.exe > paramat.results
```

16.3 PVMe

16.3.1 Préalable

Définir la variable d'environnement PVMEPATH qui indique le répertoire dans lequel se trouveront les exécutables. Par exemple :

```
export PVMEPATH=$HOME/mypvme
```

La partition peut être définie comme pour le PVM (voir «Partition» à la page 67) ou implicitement par le *Resource Manager* (voir «Lancement de PVMe» à la page 69). Pour rappel, PVMe utilise systématiquement le HPF en mode *us*.

16.3.2 Compilation

xlf [xlf_options]
**-L/usr/lpp/pvm3/lib -lpvm3 -lfpvm3 **
**-bI:/usr/lib/pvm3e.exp **
**-L/usr/lpp/ssp/css/libus -lcss **
**-bI:/lib/syscalls.exp -bnso **
-o executable sources.f

Pour les *xlf_options*, voir Tableau 4 à la page 53.

16.3.3 Lancement de PVMe

PVMe doit être lancé à partir d'un noued du pool parallèle qui sera le *home node*. Comme PVM, il peut être lancé de deux façons :

1. En interactif

/usr/lpp/pvm3/pvm -p 0 [**-share cpu**] [n|hostfile]

On accède ainsi à la console PVMe qui accepte des commandes analogues à celles de PVM.

L'option *-share cpu* permet de partager les CPUs. Par défaut, ils sont dédicacés. *n* désigne le nombre de noeuds que devraient allouer le *Resource Manager*. *hostfile* donne la liste des noeuds à allouer dans la partition. Si ni *n* ni *hostfile* n'est spécifié, le *Resource Manager* cherche à allouer un noeud. Les commandes disponibles permettent éventuellement de redéfinir la partition.

2. En arrière-plan :

/usr/lpp/pvm3/pvmd3e -p 0 [**-share cpu**] n|hostfile **&**

n représente le nombre de noeuds de la partition. Si *n* est spécifié, la partition sera définie implicitement par le *Resource Manager*.

On arrête PVMe de la même façon que PVM. Voir «Lancement de PVM» à la page 68.

16.3.4 Exécution

A partir du *home node*, l'exécution du programme se fait comme en PVM (voir «Exécution» à la page 68).

16.3.5 Exemple

1. Compilation

```
xlf -O -L/usr/lpp/pvm3/lib -lpvm3 -lfpvm3 \
     -bI:/usr/lib/pvm3e.exp \
     -L/usr/lpp/ssp/css/libus -lcss \
     -bI:/lib/syscalls.exp -bnso \
     -o paramat.master.exe paramat.master.f
xlf -O -L/usr/lpp/pvm3/lib -lpvm3 -lfpvm3 \
     -bI:/usr/lib/pvm3e.exp \
     -L/usr/lpp/ssp/css/libus -lcss \
     -bI:/lib/syscalls.exp -bnso \
     -o paramat.slave.exe paramat.slave.f
```

2. Lancement de PVMe à partir d'un noeud du pool parallèle :

```
export PVMEPATH=$HOME/mypvme
/usr/lpp/pvm3/pvmd3e -p 0 -share cpu 4 &
```

3. Exécution :

```
paramat.master.exe > paramat.results
```

16.4 Remarque

Pour plus de détails, notamment sur les divers outils qui accompagnent les systèmes de programmation parallèle (*debuggers*, ...), voir «Annexe A. Bibliographie» à la page 99.

17.0 Batch processing

LoadLeveler permet de soumettre un job sur une ou plusieurs machines / noeuds. Un *job* est constitué de *job steps*. Chaque *job step* comprend un exécutable. L'exécution d'un *job step* peut être dépendante de celle du précédent. Un *job* est décrit par un *job command file*, en abrégé JCF. Les *jobs* peuvent être sériels ou parallèles. Les *jobs* parallèles sont décomposés en tâches qui peuvent s'exécuter simultanément sur plusieurs machines / noeuds.

Les étapes du *batch processing* sont les suivantes :

1. Créer un JCF.
2. Soumettre le job.
3. Suivre l'exécution du job.

17.1 Job command file (JCF)

Un JCF peut comprendre :

1. des *keyword statements* de la forme suivante

```
# @ keyword [= valeur(s)]
```

   0, 1 ou plusieurs espaces peuvent séparer les différents éléments du *statement*. Les *keywords* peuvent être écrits en majuscules ou minuscules ou avec un mélange des deux. Un *slach* inversé en fin de ligne indique que le *keyword statement* continue sur la ligne suivante.

2. des commentaires. Un commentaire est une ligne qui commence par **#** et qui n'est pas un *keyword statement*.

keyword statements et commentaires sont traités comme commentaires par le shell.

On peut constuire un JCF à l'aide d'un éditeur quelconque ou via un GUI (*Graphical User Interface*) spécifique au *LoadLeveler* que l'on appelle par la commande

xloadl &

Le GUI **xloadl**, utilisable sous X-Windows, est tout à fait convivial. Il n'est pas décrit dans le présent guide. Les principaux *keywords* sont présentés dans le tableau suivant :

Keyword	Valeur	Commentaire
job_name	Nom du job	Nom : toute combinaison de lettres, chiffres et *underscore* (_). Ignoré au dela du 1er step. Même avec job_name, le job reçoit un nom de la forme jobid.stepid affecté par le système au moment de la soumission.
notification	**always** **start** **error** **complete** **never**	Envoi d'un message à l'utilisateur quand le job... ... commence, produit une erreur et se termine. ... commence. ... produit une erreur. ... se termine Pas d'envoi de message.
notify_user	email	Adresse électronique à laquelle le message de notification doit être envoyé.
class		Voir Tableau 11 à la page 95.
step_name	Nom du step	Nom : toute combinaison de lettres, chiffres, *underscore* (_) et point (.).
shell (*)	dgfile	Désignation du shell à utiliser par le job step. Défaut: celui indiqué dans le fichier /etc/password.
environment (*)	**COPY_ALL** **$**var**;**... **!**var**;**... var**=**valeur**;**...	Permet de reprendre tout ou partie des variables d'environnement du shell en cours au moment du lancement du job ou de définir de nouvelles variables. Copie les variables d'environnement en cours. Copie les variables indiquées. Utilisé avec COPY_ALL, exclut la variable var. Crée la variable var. Exemple : environment COPY_ALL; \ !VAR1; !VAR2; VAR3=Test
initialdir (*)	dir	Répertoire de travail initial. Les noms de fichiers ne commençant par / sont relatifs à ce répertoire. Défaut: répertoire en cours lors du lancement du job.

Keyword	Valeur	Commentaire
executable (*)	dgfile	Nom du programme exécutable. En l'absence de ce *keyword*, LoadLeveler traite le JCF comme un *script*.
arguments	args	Liste des arguments à passer à l'exécutable.
input (*)	dgfile	Nom du fichier utilisé par l'exécutable comme stdin. Défaut: /dev/null
output (*)	dgfile	Nom du fichier utilisé par l'exécutable comme stdout. Défaut: /dev/null
error (*)	dgfile	Nom du fichier utilisé par l'exécutable comme stderr. Défaut: /dev/null
dependency	expL	Le job step n'est exécuté que si l'expression logique expL a la valeur vrai. expL est de la forme : step_name operator value Exemple : dependency = (setp1 == 0) && (step2 > 0)
hold (*)	**user** **system** **usersys**	Permet de différer l'exécution d'un step jusqu'à ce que la commande llhold -r soit exécutée ... ... par le propriétaire du job ou l'administrateur du système. ... par l'administrateur du système. ... par le propriétaire du job et l'administrateur du système.
startdate	MM/DD/YY [hh:mm[:ss]]	Permet de préciser la date et l'heure à laquelle le job doit s'exécuter. Défaut : date et heure de la soumission.
job_cpu_limit	[[h:]m:]s[.f]	CPU maximum en heures (h), minutes (m), secondes (s) et fractions de secondes (f) qu'un job step peut consommer. Pour un programme parallèle, la limite s'applique à chacun des noeuds. Défaut: limite de la classe.
image_size (*)	n	Nombre de k-bytes nécessaire pour l'exécution du step. Défaut: taille de l'exécutable. La taille de l'exécutable ne suffit pas toujours; c'est par exemple le cas pour un programme qui lie dynamiquement des librairies. LoadLeveler essaie de "dispatcher" le step vers un noeud qui dispose des ressources demandées.
queue		Place une copie du job step en file d'attente.
Remarque : (*) : Valeur héritée par les job steps suivants.		

Tableau 9. LoadLeveler - Principaux keywords

Pour les programmes parallélisés :

Keyword	Valeur	Commentaire
job_type	**parallel** **pvm3**	pour PE et PVMe pour PVM domaine public Défaut : **serial**.
max_processors	n	Nombre maximum de noeuds requis par un programme parallélisé. Défaut : 1.
min_processors	n	Nombre minimum de noeuds requis par un programme parallélisé. Défaut : 1.
parallel_path	dir	Pour PVM : indique où se trouvent les exécutables
requirements (*)	**(pool==0)** **(Adapter=="hps_ip")** **(Adapter=="hps_user")** **(Adapter=="ethernet")**	Pool des noeuds réservés aux programmes parallélisés. Pour utiliser le HPF avec protocole Internet. Pour utiliser le HPF avec protocole User Space. Pour utiliser l'adaptateur Ethernet. Exemple : requirements = ((pool == 0) \ && (Adapter == "hps_ip"))
Remarque : (*) : Valeur héritée par les job steps suivants.		

Tableau 10. LoadLeveler - Keywords additionnels pour programmes parallélisés

Variables spéciales du *LoadLeveler* :

Les variable spéciales suivantes sont initialisées lors de la soumission d'un job. Elles peuvent être utiilisées dans un JCF :

- $(Machine) : contient le nom de la machine à laquelle le job est soumis;
- $(Jobid) : contient l'identification du job.
- $(Stepid) : contient l'identification du job step.

Exemples :

1. Job avec 2 steps dépendants

```
# Compilation et link edit d'un programme Fortran
# et exécution si pas d'erreur à l'étape précédente
```

```
#
# @ job_name = Exemple1
# @ notification = always
# @ notify_user = nihon@segi.ulg.ac.be
# @ class = sers
# @ initialdir = /home/segi/nihon/myll
# @ output = $(Jobid).$(Stepid).out
# @ error = $(Jobid).$(Stepid).err
#
# @ step_name = Step1
# @ executable = /usr/bin/xlf
# @ arguments = -qsource,listopt -o myprog.exe myprog.f
# @ input = /dev/null
# @ queue
#
# @ dependency = (Step1 == 0)
# @ step_name = Step2
# @ executable = myprog.exe
# @ input = exemple.data
# @ queue
```

Remarque : Dans cet exemple, on utilise les variables spéciales $(Jobid) et $(Stepid) pour les définitions des fichiers *output* et *error*. Ces définitions ne sont données qu'une fois en début de job et sont propagées dans les différents steps.

2. Jobs parallélisés

```
# Exécution d'un programme "Parallel Environment"
#
# @ job_name = Exemple2
# @ notification = always
# @ notify_user = nihon@segi.ulg.ac.be
# @ class = pars
# @ job_type = parallel
# @ min_processors = 4
# @ max_processors = 4
# @ requirements =  ((Adapter == "hps_ip") &&  (Pool == 0))
#
# @ initialdir = /home/segi/nihon/mype
# @ executable = /usr/bin/poe
# @ arguments = -procs 4          \
                -resd yes         \
                -euilib ip        \
                -euidevice css0   \
                -rmpool 0         \
                -pgmmodel mpmd    \
                -infolevel 2      \
                -cmdfile paramat.exelist
# @ output = paramat.results
# @ error = paramat.err
# @ queue
```

```
# Exécution d'un programme "PVMe"
#
# @ job_name = Exemple3
# @ notification = always
# @ notify_user = nihon@segi.ulg.ac.be
# @ class = pars
# @ job_type = parallel
# @ min_processors = 4
# @ max_processors = 4
# @ requirements =  ((Adapter == "hps_user") &&  (Pool == 0))
#
# @ initialdir = /home/segi/nihon/mypvme
# @ executable = /dev/null
# @ output = paramat.results
# @ error = paramat.err
# @ queue
echo "*** Starting ***"
export PVMETATH=/home/segi/nihon/mypvme
/usr/lpp/pvm3/pvmd3e  -p 0 4  \
    -exec paramat3.master.exe
echo "*** Ending ***"
```

```
# Exécution d'un programme "PVM"
#
# @ job_name = Exemple4
# @ notification = always
# @ notify_user = nihon@segi.ulg.ac.be
# @ class = pars
# @ job_type = pvm3
# @ min_processors = 4
# @ max_processors = 4
# @ requirements =  ((Adapter == "hps_ip") &&  (Pool == 0))
#
# @ initialdir = /home/segi/nihon/pvm3/bin/RS6K
# @ executable = paramat.master.exe
# @ output = paramat.results
# @ error = paramat.err
# @ parallel_path = /home/segi/nihon/pvm3/bin/RS6K
# @ queue
```

17.2 Soumettre un job

llsubmit [options] {dfile | -}

dfile est le nom d'un JCF. **-** indique que le JCF doit être lu sur stdin. **llsubmit** produit un message de la forme

submit: The job *jobid* has been submitted

où *jobid* est un identificateur de la forme *machine.job*. *machine* est le nom de la machine à laquelle le job est soumis. *job* est l'identificateur du job.

En outre, chaque *job step* d'un job reçoit aussi un identificateur. L'identificateur complet d'un step est de la forme *machine.job.step*.

17.3 Suivre l'exécution d'un job

Connaître l'état d'un ou plusieurs job steps :

llq [options] [jobids]

Si *jobids* n'est pas spécifié, la commande **llq** donne l'état de tous les jobs pris en chage par le *LoadLeveler*. La commande **llq** donne, pour chaque job step, le jobid, le propriétaire, la date et l'heure de soumission, l'état (ST), la priorité relative (PRI), la classe

et, éventuellement, le noeud sur lequel le job step s'exécute. L'état est indiqué par les abréviations suivantes :

R : *running*, le job step est en cours d'exécution.
I : *idle*, le job step est dans un état anormal d'attente. Réexécuter la commande **llq** avec l'option **-s** et le *jobid* permet d'obtenir des informations supplémentaires relatives à cet état anormal.
P : *pending*, le job step a été "dispatché" vers un noeud mais n'est pas encore en cours d'exécution.
C : *completed*, le job step est terminé mais pas le job auquel il appartient.
X : *removed*, le job step a été arrêté mais pas le job auquel il appartient.
S : *system hold*, le job step a fait l'objet d'un *system hold*.
H : *user hold*, le job step a fait l'objet d'un *user hold*.

Option utile : *-l* permet d'obtenir un *long listing* qui donne plus de détails sur l'état des jobs.

Retenir ou libérer un job :

llhold [options] jobids

Option utile : **-r**, abréviation de *release*, permet de supprimer l'état hold du job indiqué. Exemples : la première commande met les job steps sp2e1.222.0 et sp2e1.222.1 en état d'attente, la seconde libère sp2e1.222.0 :

```
llhold sp2e1.222.0 sp2e1.222.1
llhold -r sp2e1.222.0
```

Arrêter l'exécution de jobs :

llcancel [options] jobids

Connaître l'état des noeuds :

llstatus [options] [hosts]

hosts est la liste des adresses internet des noeuds. Si *hosts* n'est pas spécifié, **llstatus** donne l'état de tous les noeuds. Les principaux renseignements donnés pour chaque noeuds sont

Name, son adresse.
InQ, nombre de job steps en file d'attente.
Run, nombre de job steps en cours d'exécution.

Option utile : **-l** donne un *long listing*.

Les fonctions de ces différentes commandes sont toutes supportées par le GUI **xloadl**.

18.0 AIX X-windows

L'AIX X-windows est l'implémentation sous AIX de l'interface utilisateur graphique X-windows développée au MIT en 1984. Plusieurs versions de X-windows (également appelé X) ont été développées, la plus récente est X11R5.

L'AIX X-windows est également fourni avec Motif créé par l'*Open Software Foundation*. Motif peut être considéré comme un ensemble de modules logiciels qui se greffent sur X afin d'en augmenter les fonctionalités[13].

X est un système de fenêtrage graphique qui permet d'utiliser plusieurs programmes simultanément (chacun dans une fenêtre séparée) sur un même écran et d'émuler des terminaux standards dans certaines fenêtres pour servir de support aux applications et commandes non X.

X s'utilise typiquement à partir d'une station de travail équipée d'un écran graphique ou à partir d'un terminal graphique spécial appelé terminal X[14].

18.1 Aspects distribués de X

X est conçu pour supporter les environnements réseaux en reposant sur le concept client/serveur.

Le serveur (appelé serveur X) est le programme qui s'exécute sur la machine qui possède un ou plusieurs écrans, il peut s'agir d'une station de travail ou d'un terminal X. Il gère l'affichage, le clavier et la souris.

Le client est le programme d'application X exécuté sur la machine locale ou sur une autre machine. Il envoie des requêtes au serveur et reçoit des informations en échange.

La communication entre client et serveur se fait suivant le protocole X qui repose lui-même sur TCP/IP.

Un terminal X exécute seulement la partie serveur de X-windows. Les clients X qui affichent des informations à l'écran du terminal X doivent donc s'exécuter en *remote*.

La Figure 3 illustre les aspects distribués de X.
Avec la configuration décrite par la Figure 3, il est possible par exemple pour l'utilisateur du terminal X d'exécuter des applications X sur la station RS/6000 et d'afficher les résultats sur l'écran de son terminal. De plus, le protocole X étant standard, il est possible pour ce même utilisateur d'exécuter simultanément d'autres clients X sur une deuxième machine, par exemple la station Sun, et d'afficher le tout sur son propre écran. Nous verrons plus loin comment en pratique on réalise ce type d'utilisation.

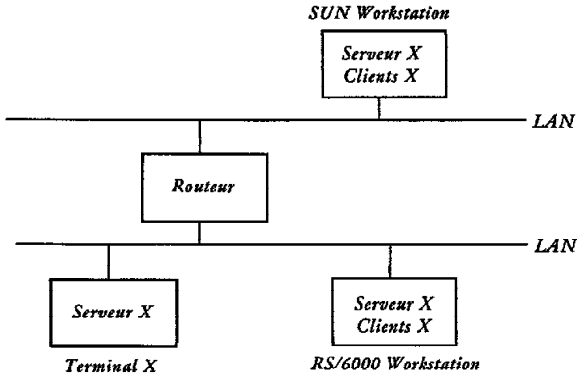

Figure 3. Illustration des aspects distribués de X

18.2 Survol de l'architecture de X

Le modèle simplifié de la Figure 4 à la page 76 montre comment les différents composants de X interagissent entre eux.
La librairie Xlib est telle qu'il existe presque une correspondance biunivoque entre les messages du protocole X et les fonctions qu'elle contient. Xlib est une librairie de relativement bas niveau car les messages du protocole X correspondent à des opérations relativement élémentaires. Par conséquent, développer une application X en se basant directement sur Xlib est relativement complexe et laborieux.

Afin de simplifier la tâche des programmeurs d'applications, des *toolkits* ont été développés. Un *toolkit* est une libraire d'objets (*widgets*) prédéfinis (tels que boutons, barres de défilement, boîtes de dialogue, menus, ...) que l'on peut utiliser pour construire une application. Certains *toolkits*, tels que le *X Toolkit*[15] et le *Toolkit* de *Motif* sont basés sur la librairie intermédiaire *Xt Intrinsics*, et de ce fait auront certaines propriétés en commun (voir «Les options de la ligne de commande» à la page 78 et «Personnaliser

---

[13] Motif n'est pas le seul système de ce type; il existe d'autres systèmes concurrents tel que Open Look par exemple.

[14] Il existe également des implémentations de X pour des micro-ordinateurs tel que PC ou MAC.

[15] La librairie d'objets du *X Toolkit* est en réalité l'*Athena Widget Set (Xaw)*.

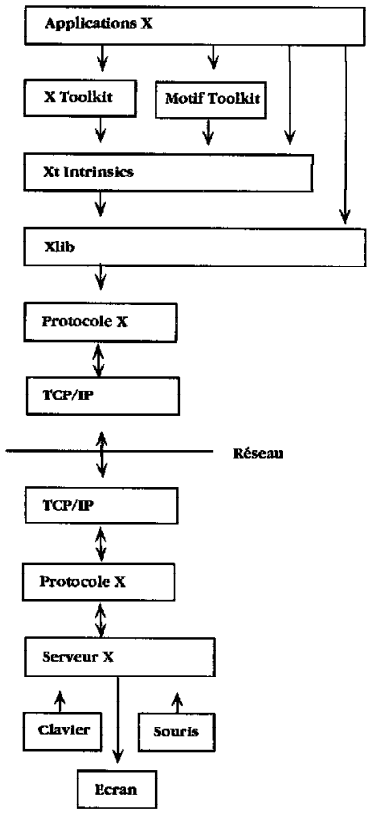

Figure 4. Composants du système X

le fonctionnement des applications X» à la page 81). En plus de procurer une augmentation substantielle de la productivité des programmeurs, les *toolkits* permettent d'uniformiser le "look" des applications. Ainsi tous les clients X basés sur un même *toolkit* ont le même "look and feel".

18.3 Démarrer X

Pour utiliser X-windows il faut réaliser les trois opérations suivantes:

- Démarrer le serveur X sur sa station.
- Démarrer au moins un émulateur de terminal comme *xterm* ou *aixterm*[16].
- Démarrer le gestionnaire de fenêtres (*Window Manager*)[17]

En fonction de la configuration de X sur votre système, ces étapes peuvent être réalisées automatiquement de différentes façons.

18.3.1 Démarrage controlé par xdm

Le gestionnaire de *display*, *xdm*, permet à l'utilisateur d'entrer en session à partir d'une fenêtre X. La session est directement démarrée dans l'environnement X. Une fenêtre présente un menu dans lequel l'utilisateur peut sélectionner la station sur laquelle il désire entrer en session. (S'il n'y a qu'une station autorisée, cette première fenêtre n'apparait pas.) Une seconde fenêtre lui permettra ensuite d'entrer son *login name* et son mot de passe.

xdm offre une sécurité accrue au niveau du serveur X : il empêche son utilisation par des clients X qui s'exécutent chez d'autres utilisateurs. Un utilisateur peut toutefois autoriser l'utilisation de son serveur X par un autre utilisateur, ou plusieurs, au moyen des commandes **xauth** ou **xhost** (voir **man**). **xhost** permet de définir des autorisations ou interdictions par station; **xauth** permet de définir des autorisations ou interdictions par utilisateur. Quand les deux commandes sont utilisées, ce sont les définitions de **xhost** qui l'emportent. Une autorisation doit notamment être définie quand un utilisateur entre en session sur une station via *xdm* puis, au moyen d'une commande comme **telnet**, entre en session sur une autre station. Exemple

```
xhost [+|-] [hostname]
```

donne (+) ou supprime (-) l'autorisation à la machine *hostname* d'accéder au serveur X de l'utilisateur en cours. Sans paramètre, la commande écrit la liste des autorisations déjà données sur stdout.

Certaines stations gérées par le SEGI sont configurées pour utiliser *xdm*. Pour celles qui ne le sont pas, c'est l'approche décrite dans le paragraphe suivant qui doit être utilisée.

---

[16] *aixterm* est un émulateur spécifique à l'AIX alors que *xterm* est un client X standard.

[17] Bien qu'il soit possible d'utiliser X sans gestionnaire de fenêtres, les fonctionalités offertes par X sont alors très limitées.

18.3.2 Démarrage de X au moyen de la commande startx

La commande **startx**, qui peut être utilisée manuellement, permet de démarrer le serveur X et d'exécuter le *shell script* $HOME/.xinitrc ou, à défaut, /usr/lpp/X11/defaults/xinitrc. Ce fichier, dont une version par défaut a dû être placée dans votre répertoire principal par votre administrateur système, permet de configurer votre serveur X ainsi que d'exécuter initialement un certain nombre de clients X comme par exemple un émulateur *aixterm*, l'horloge *xclock* et le gestionnaire de fenêtre *mwm* de *Motif*.

Pour effectuer un démarrage automatique de X, il suffit alors par exemple de placer dans son fichier .profile les lignes suivantes :

```
TERMTYPE=$(tty | grep hft)
 if [ -n "$TERMTYPE" -o -n "$XSTATION" ]
 then
        if [ -f /usr/lpp/X11/bin/startx ]
        then
                /usr/lpp/X11/bin/startx
        fi
 fi
```

qui lancent l'exécution de *startx* à condition que vous vous trouviez sur une station supportant X. Normalement, ces quelques lignes ont été incluses dans le fichier .profile que votre administrateur système a créé pour vous lors de l'initialisation de votre compte.

L'utilisateur peut évidemment modifier le fichier $HOME/.xinitrc afin de personnaliser certains aspects de l'environnement X qu'il rencontre dès son entrée en session.

18.4 Travailler dans l'environnement X

Le gestionnaire de fenêtres est le client X qui remplit les fonctions importantes suivantes:

- Changement de fenêtre active.
- Déplacement des fenêtres.
- Modification de la taille des fenêtres.
- Réduction des fenêtres à des icônes.
- Développement des icônes en fenêtres.
- Modification de l'ordre d'empilement des fenêtres.
- Fermeture et ouverture d'une fenêtre.
- Définition du "look and feel" du contour des fenêtres.

Toutes ces fonctions ont été concentrées dans le gestionnaire de fenêtres afin que les autres clients X n'aient pas à gérer ces problèmes. Ces fonctionnalités sont rendues accessibles aux utilisateurs notamment par la manipulation, au moyen de la souris, des boutons composant le contour des fenêtres.

L'AIX X-windows fournit deux gestionnaires de fenêtres:

- *mwm (Motif Window Manager)* qui est le gestionnaire de Motif.
- *twm (Tab Window Manager)* qui est le gestionnaire standard de X.

Dans la suite on ne fera plus référence qu'à *mwm*, qui est le gestionnaire généralement utilisé avec l'AIX X-windows.

18.5 Quelques clients X

Parmi les clients X standards qui sont fournis avec X-Windows on peut citer quelques applications utiles:

- Emulateur de terminal

  aixterm: émulateur propre à l'AIX.

  xterm: émulateur standard X

- Accessoires de bureau

  xbiff: notification de courrier entrant.

  xclock, oclock: horloges.

  xcalc: calculatrice.

  xman: accès aux pages du manuel online.

- Gestion des *fonts*

  xlsfonts: liste les fonts disponibles

  xfd: affiche les caractères associés à un font.

  xfontsel: permet d'afficher et de sélectionner un font.

- Utilitaires graphiques

  bitmap: éditeur bitmap.

  xmag: agrandir une zone de l'écran.

- Applications d'impression

xwd: copie une fenêtre dans un fichier.

xpr: convertit un fichier créé par xwd en PostScript ou dans un autre format approprié pour l'impression.

xwud: affiche à l'écran le contenu d'un fichier créé avec xwd.

- Accès aux caractéristiques des clients X

  xlsclients: donne la liste des clients connectés à un serveur X.

  xdpyinfo: liste les caractéristiques générales du *display*.

  xwininfo: donne les caractéristiques de la fenêtre sélectionnée.

  xprop: donne les propriétés associées à une fenêtre.

- Caractéristiques du clavier et du *display*.

  xset: permet de modifier certaines caractéristiques du clavier et du display, telles que: volume de l'alarme, vitesse du curseur, ...

  xmodmap: permet d'associer aux touches du clavier et aux boutons de la souris certaines fonctions.

18.6 Les options de la ligne de commande

La plupart des clients X possèdent un grand nombre d'options qui peuvent être spécifiées lors du démarrage de l'application. En plus d'options spécifiques au programme exécuté, toutes les applications construites à l'aide du X Toolkit (ou tout autre toolkit basé sur la librairie Xt Intrinsics, tel que le Toolkit de Motif), accepte certaines options standards.

Nous allons passer en revue les options standards (appelées également *X Toolkit options*) les plus fréquemment utilisées.

- *-bg* ou *-background* : couleur en arrière plan dans la fenêtre.
- *-display* : spécifie le serveur X utilisé pour l'affichage.
- *-fn* ou *-font* : *font* utilisé pour le texte.
- *-fg* ou *-foreground* : couleur en avant plan pour le texte et les graphiques.
- *-geometry* : position et dimensions de la fenêtre.
- *-iconic* : démarre le client X sous forme d'icône.
- *-name* : spécifie un nom pour l'application (voir «L'option -name» à la page 83).
- *-title* : donne un titre à la fenêtre (affiché dans la barre du titre).
- *-xrm* : permet de spécifier une ressource sur la ligne de commande (voir «L'option -xrm» à la page 82).

18.6.1 L'option -display

La syntaxe de l'option *-display* est:

```
-display [host]:server[.screen]
```

host est le nom ou l'adresse de la machine locale sur laquelle se trouve le serveur X. *server* représente le numéro du serveur et *screen*, le numéro d'écran. Dans ce contexte, *server* représente un système d'affichage physique (*display*) contrôlé par un serveur X. Un *display* peut être composé de plusieurs écrans mais seulement d'un seul clavier et d'une seule souris. Si un seul *display* existe, ce qui est le cas de la plupart des machines, il est numéroté 0; si une machine a plusieurs *displays*, chacun se voit attribuer un numéro (en commençant par 0). De façon similaire, si un *display* est composé de plusieurs écrans (partageant un clavier et une souris), chaque écran se voit assigner un numéro (en commençant par 0). La façon dont l'option *-display* est employée sera illustrée au paragraphe «Exécuter un client X sur une machine distante» à la page 80.

18.6.2 L'option -geometry

L'option *-geometry* s'emploie comme suit :

```
-geometry geometry
```

le paramètre *geometry* a quatre composants numériques, deux spécifiant la dimension de la fenêtre et deux spécifiant sa position. La syntaxe de l'argument est :

width**x**height±xoff±yoff

width et *height* spécifient respectivement la largeur et la hauteur de la fenêtre. Ils sont exprimés en pixels sauf pour les émulateurs comme *aixterm* et *xterm* où ils s'expriment naturellement en caractères.

xoff et *yoff* s'interprètent de la façon suivante:

- *+xoff* : distance en pixels du bord gauche de la fenêtre au bord gauche de l'écran.
- *+yoff* : distance en pixels du bord supérieur de la fenêtre au bord supérieur de l'écran.
- *-xoff* : distance en pixels du bord droit de la fenêtre au bord droit de l'écran.
- *-yoff* : distance en pixels du bord inférieur de la fenêtre au bord inférieur de l'écran.

Exemple:

```
xclock -geometry  -10+10 &
```

place une horloge de taille par défaut à 10 pixels du bord supérieur et du bord droit de l'écran.

18.6.3 La spécification des couleurs

Des options telles que *-bg* et *-fg* admettent comme paramètre une couleur. La spécification d'une couleur comme paramètre de ces options est réalisée soit en donnant le nom d'une couleur prédéfinie dans le fichier système /usr/lib/X11/rgb.txt soit en donnant le code RGB hexadécimal de la couleur souhaitée.

Exemple:

```
aixterm -bg lightblue -fg yellow &
```

permet d'ouvrir une nouvelle fenêtre dont le fond est en bleu clair et le texte en jaune.

Le fichier rgb.txt contient une table associant des noms à des couleurs. Chaque couleur est définie dans ce fichier au moyen de son code RGB. Le code RGB contient 3 bytes. Le premier byte représente l'intensité du rouge, le deuxième celle du vert et le troisième celle du bleu. Chaque couleur étant ainsi représentée par un mélange pondéré des trois couleurs rouge, bleu et vert. Par exemple, le noir est codé par 0 0 0, le blanc par 255 255 255, le rouge par 255 0 0, le vert par 0 255 0 et le bleu par 0 0 255.

Pour obtenir la liste des noms de couleurs prédéfinis on peut utiliser la commande **showrgb | more**, qui affiche le contenu du ficheir rgb.txt.

18.6.4 Spécification des fonts

La plupart des applications X permettent à l'utilisateur de spécifier le *font* à utiliser pour l'affichage du texte dans les fenêtres, menus et boutons. La spécification des *fonts* sous X est très souple mais de ce fait relativement complexe. Certains clients X sophistiqués possèdent une interface conviviale permettant de faire facilement le choix d'un *font*. Malheureusement ce n'est pas le cas des clients X standards. L'utilisateur est donc contraint pour sélectionner un *font* adéquat de connaître tout au moins certains aspects de la nomenclature des *fonts* sous X.

18.6.4.1 Structure des noms de fonts

Sous X, le nom de chaque *font* spécifie toutes les caractéristiques du *font* auquel il correspond. Exemple :

```
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
```

Donnons la signification des différents composants de ce nom de font:

- *adobe*: nom du fabricant.
- *courier*: nom de la famille à laquelle appartient le *font*.
- *bold*: intensité des caractères: *bold, medium, demibold*.
- *o*: style des caractères: *o (oblique), r (upright), i (italic), ri (reverse italic), ro (reverse oblique)*.
- *normal*: largeur des caractères: *normal, condensed, semicondensed, narrow, double width, block, ...*
- *10*: taille des caractères en *pixels*
- *100*: taille des caractères en dixième de points
- *75*: résolution horizontale en *dpi*: 75 ou 100 *dpi*.
- *75*: résolution verticale en *dp*i: 75 ou 100 *dpi*.
- *m*: espacement des caractères: *m* ou *c (monospaced), p (proportionally spaced)*
- *60*: largeur moyenne des caractères en dixièmes de *pixels*.
- *iso8859-1*: ensemble de caractères.

Remarque :

La taille du *font* tel qu'on le voit à l'écran, dépend non seulement de sa taille exprimée en point mais également de la résolution de l'écran (exprimée en *dpi*). Ainsi de nombreux *fonts* sont fournis à la fois en version 75 dpi et 100 dpi.

18.6.4.2 Choisir et spécifier un font

Les *fonts* fournis avec X se trouvent dans les trois répertoires suivants: /usr/lib/X11/fonts, /usr/lib/X11/fonts/75dpi et /usr/lib/X11/fonts/100dpi. Les *fonts* sont contenus dans ces répertoires sous forme de fichiers possédant l'extension snf.Z

Lorsque l'utilisateur spécifie un nom de *font*, X recherche ce *font* en suivant un chemin prédéfini et sélectionne le premier *font* rencontré qui satisfait la spécification. L'utilisateur peut modifier ce chemin avec la commande **xset** et l'option **fp**. Ainsi

```
xset fp= /usr/lib/X11/fonts,/usr/lib/X11/fonts/100dpi,\
/usr/lib/X11/fonts/75dpi
```

spécifie l'ordre de consultation des trois répertoires. Pour rétablir le chemin par défaut:

```
xset fp default
```

Pour obtenir la liste dans l'ordre de tous les *fonts* accessibles en fonction du chemin de recherche courant, on peut utiliser

```
xlsfonts
```

Pour sélectionner et visualiser un font on peut utiliser le client

```
xfontsel.
```

Pour visualiser un font à partir de son nom, on utilise:

```
xfd -font fontname.
```

18.6.4.3 Fichiers fonts.dir

Dans chaque répertoire faisant partie du chemin de recherche des *fonts*, en plus des fichiers associés aux différents *fonts*, on rencontre un fichier fonts.dir. Ce fichier réalise l'association entre les noms complets de *fonts* et les noms de fichiers utilisés par le serveur X pour représenter les *fonts*.

18.6.4.4 Alias pour noms de fonts

Chaque répertoire du chemin de recherche des *fonts* peut contenir un fichier d'*alias* fonts.alias. Un fichier fonts.alias permet d'associer des alias à des noms de *fonts*. Ces *alias* peuvent ensuite être utilisés en lieu et place des noms de *fonts*, afin de simplifier la spécification des *fonts* par l'utilisateur. Exemple :

```
6x10        -misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
```

Si le nom d'un fichier fonts.alias commence par FILE_NAMES_ALIASES, alors cela signifie que les noms des fichiers de *fonts* (contenus dans le même répertoire), sans l'extension.snf, peuvent être utilisés comme *alias*. Par exemple Rom14.iso1, bien que ne figurant dans aucun fichier fonts.alias, est un *alias* valide car dans le répertoire /usr/lib/X11/fonts on trouve le fichier de *font* Rom14.iso1.snf.Z et un fichier fonts.alias qui commence bien par FILE_NAMES_ALIASES. Pour déterminer quelles sont les caractéristiques du font référencé par Rom14.iso1, on peut consulter le fichier fonts.dir et on trouve alors:

```
Rom14.iso1.snf.Z       -ibm--medium-r-medium--20-14-100-100-c-90-iso8859-1
```

18.6.4.5 Synthèse de la spécification des fonts

Lorsque l'utilisateur souhaite utiliser un *font* particulier, il peut spécifier ce *font* de différentes façons:

- en donnant le nom complet du *font*
- en donnat un nom dans lequel les caractéristiques inconnues ou sans intérêt on été remplacées par un astérisque '*'. Remarquons que si plusieurs *fonts* satisfont cette spécification partielle, le premier de ces *fonts* trouvé dans le chemin de recherche sera sélectionné, ce qui correspondra peut-être à un *font* ne possédant pas les propriétés requises!

  Exemple: si le font est spécifié sur la ligne de commande on pourrait avoir

  ```
  -font '*courier-bold-r*140*'
  ```

  L'usage des *quotes* est nécessaire pour éviter l'interprétation par le shell des astérisques de la ligne de commande.

- - en donnant un *alias* de *font*.

18.7 Exécuter un client X sur une machine distante

L'utilisateur peut exécuter des applications sur une station distante comme il le fait sur sa propre machine. Cela n'est évidemment possible que si l'utilisateur peut entrer en session sur le système distant. L'utilisateur peut donc partager son écran entre plusieurs fenêtres associées à des applications X qui peuvent s'exécuter sur des machines différentes.

Pour exécuter un client X sur une machine *remote* on suit la procédure :

- autoriser l'accès à son serveur X par les clients X du système distant au moyen de la commande **xhost**.
- entrer en session sur le système distant.
- lancer l'exécution des applications X désirées sur le système distant. Dans tous les cas, il faut indiquer aux clients X qu'ils doivent se connecter au serveur X de la machine locale en utilisant l'option standard *-display* ou en affectant à la variable d'environnement DISPLAY de la machine distante la même valeur que celle du paramètre de l'option *-display*.

Exemple:

L'utilisateur qui travaille sur le terminal X xst6.segi.ulg.ac.be et qui est en session sur la machine aix1.segi.ulg.ac.be souhaite exécuter également des clients X sur la machine *remote* gw.unipc.ulg.ac.be.

- Il détermine quels sont les machines autorisées à accéder à son serveur X :

  ```
  $ xhost
  access control enabled (only following hosts are allowed)
  aix1.segi.ulg.ac.be
  ```

- Il faut donc introduire gw.unipc.ulg.ac.be dans cette liste de machines :

```
$ xhost gw.unipc.ulg.ac.be
gw.unipc.ulg.ac.be being added to access control list
```

- Ensuite, entrée en session au moyen de la commande **telnet** ou **rlogin**, ou bien usage de **rsh** ou **rexec** pour l'exécution d'une commande à distance. La ligne de commande utilisée pour exécuter par exemple le client X *aixterm* sera :

  ```
  aixterm –title gw.unipc –display xst6.segi.ulg.ac.be:0
  ```

  qui démarre donc sur gw.unipc.ulg.ac.be le programme *aixterm* mais qui ouvre la fenêtre de cette application sur le terminal X local. L'option *-title* permet à l'utilisateur de savoir où s'exécute l'application.

18.8 Personnaliser le fonctionnement des applications X

Il est possible pour l'utilisateur de modifier les caractéristiques de pratiquement tous les clients X. Ainsi on peut spécifier : la taille et la position de sa fenêtre, les couleurs utilisées, le *font*, l'utilisation éventuelle d'une barre de défilement (*scrollbar*), ...

Le comportement des applications UNIX traditionnelles est modifiable par les options de la ligne de commande. Nous avons vu (voir «Les options de la ligne de commande» à la page 78) que les applications X permettent aussi de spécifier des options au niveau de la ligne de commande. Cependant toutes les caractéristiques des clients X ne sont pas modifiables par cette voie et même si elles l'étaient cela conduirait à des lignes de commandes fastideuses à taper.

X offre une alternative aux options de la ligne de commande. Pratiquement toutes les propriétés des clients X sont spécifiées par des variables appelées ressources. Pour changer le mode d'exécution de l'application il suffit d'altérer les valeurs de certaines de ces variables. Il est à noter que toutes les options X ont leur équivalent sous forme de ressource. La réciproque est évidemment fausse puisqu'il y a de nombreuses ressources qui ne correspondent à aucune option de la ligne de commande.

18.8.1 Syntaxe des noms de ressources

Pour comprendre la syntaxe des noms de ressources, il est bon de rappeler la structure logicielle des applications X. Tout client X qui a été conçu pour utiliser le *X Toolkit* (ou tout autre *Toolkit* basé sur la librairie *Xt Intrinsic*, tel que le *Toolkit* de Motif) résulte de l'assemblage d'un certains nombre d'objets standards (*widgets*) tels que menus, boutons de commande, boîtes de dialogue, barres de défilement, ...

La structure d'un client X, en terme d'encapsulation d'objets, est une hiérarchie qui peut contenir un certain nombre de niveaux, un *widget* composant l'application peut en contenir un autre, qui lui même peut en contenir un autre, ... Par exemple une boîte de dialogue peut contenir un bouton, qui lui même contient du texte.

Il existe un parallèlisme entre la syntaxe des noms de ressources et la structure hiérarchique des objets qui composent l'application. La syntaxe générale de la définition d'une ressource est :

```
object.subobject[.subobject...].attribute: value
```

où :

object désigne le nom du programme;
subobjects: correspondent aux différents niveaux de la hiérarchie qui composent la structure de l'application, tels que fenêtres, menus, barres de défilement, ...
attribute désigne une caractéristique du dernier *subobjet* spécifié (par exemple un bouton de commande), telle que la couleur du *background* ou le *font* du texte affiché.
value désigne la valeur à affecter à *attribute*, c'est-à-dire un texte, une couleur, un *font* ou une autre caractéristique.

Le type de la valeur à spécifier est souvent facilement déductible du nom de la ressource ou de la description de la ressource donnée dans la documentation *online*.

Remarque : Lorsqu'une ressource est spécifiée dans un fichier de configuration et qu'elle comporte une erreur de syntaxe, le système ignore simplement l'existence de la ressource sans générer un message d'erreur.

18.8.2 Couplage étroit et couplage lâche.

La notion de couplage fait référence à la façon dont les composants de la spécification d'une ressource sont liés entre eux. Il existe deux types de couplage:

- Etroit, représenté par un point (.).
- Lâche, représenté par un astérisque (*).

Sans entrer dans des détails superflus, signalons que l'utilisateur a TOUJOURS intérêt à utiliser le couplage lâche plutôt que le couplage étroit. En effet, l'usage correct du couplage étroit requiert une connaissance parfaite de la structure hiérarchique des objets qui composent l'application, cette restriction ne s'appliquant pas au couplage lâche.

Exemple:

Par défaut le client *aixterm* n'utilise pas une barre de défilement. Pour en obtenir une l'utilisateur peut introduire dans son fichier .Xdefaults une spécification de ressource adéquate.

La définition suivante, utilisant le couplage étroit, n'est pas correcte car la structure hiérarchique n'a pas été respectée:

```
aixterm.scrollBar : true
```

La spécification correcte utilisant le couplage étroit est plutôt:

```
aixterm.vt100.scrollbar : true
```

qui est consistante avec la structure de l'application. Pour l'utilisateur, il est préférable d'avoir recours au couplage lâche qui lui permet d'ignorer la structure exacte du client X et d'omettre de spécifier certains niveaux de la hiérarchie. Ainsi au moyen du couplage lâche, on peut employer:

```
aixterm*scrollBar: true
```

18.8.3 Instances et classes

Chaque composant de la spécification d'une ressource appartient à une classe. Plusieurs composants différents peuvent appartenir à une même classe. Par exemple, dans le cas de *aixterm*, la couleur du texte (*foreground*), la couleur du curseur de la souris et la couleur du curseur de texte sont tous des instances de la classe *Foreground*. Grâce à la notion de classe, il est possible de définir la valeur de ces trois attributs avec une seule spécification. Ainsi, pour rendre la couleur du texte, et des deux curseurs bleu foncée, on peut procéder de deux manières:

```
aixterm*foreground: darkblue
aixterm*cursorColor: darkblue
aixterm*pointerColor: darkblue
```

ou

```
aixterm*Foreground: darkblue
```

Les noms de classe commencent toujours par une majuscule, alors que les noms d'instances commencent par une minuscule. Cependant, si le nom de l'instance est composé alors le second mot débute par une majuscule (exemple: cursorColor).

La notion de classe permet, par exemple, de spécifier pour un client X particulier que tous les boutons d'une boîte de dialogue soient bleus mais qu'un de ces boutons soit rouge. Par exemple pour l'application hypothétique *xclient*, on peut avoir les définitions suivantes:

```
xclient*buttonbox*Buttons*foreground: blue
xclient*buttonbox*delete*foreground: red
```

où *Buttons* est la classe de tous les boutons de la boîte *buttonbox* et *delete* est une instance particulière (c'est-à-dire ici un bouton particulier) de cette classe. Les noms de classes permettent ainsi de spécifier la valeur par défaut de toutes les instances de la classe. Les noms d'instances sont utilisés pour faire exception à la règle générale définie au moyen des noms de classes. L'usage des noms de classes est souple et puissant : il est par exemple possible de spécifier la couleur *foreground* par défaut de tous les clients X:

```
*Foreground: blue
```

Le manuel *online* (accessible par la commande man) de chaque client fournit les noms de classes et d'instances pour chaque ressource que l'utilisateur peut spécifier.

18.8.4 Règles de priorité

Même dans un seul fichier de ressources, comme par exemple le fichier $HOME/.Xdefaults défini par l'utilisateur, il peut exister de nombreux conflits entre les spécifications. Nous avons déjà vu un cas de conflit entre une définition faite au moyen d'un nom de classe et une définition portant sur la même ressource mais réalisée au moyen d'un nom d'instance. On a alors expliqué que c'était la spécification utilisant le nom d'instance qui prévalait sur la définition plus générale basée sur le nom de classe.

Il existe d'autres règles de priorité pour résoudre les différents conflits qui peuvent survenir. La philosophie générale est que plus une définition est précise (donc moins générale) et plus elle a un dégré de priorité élevé. Voici l'ensemble des règles qu'il convient d'appliquer pour résoudre les conflits:

1. Le couplage étroit a priorité sur le couplage lâche. Ainsi

   ```
   aixterm.vt100.scrollBar: false
   ```

   a priorité sur

   ```
   aixterm*scrollBar: true
   ```

2. Les instances ont priorité sur les classes. Par exemple **scrollBar* a priorité sur **Scrollbar*.

3. Un nom de classe ou d'instance spécifié aura priorité sur un nom omis. Par exemple, *xterm*scrollbar* est plus précis que **scrollbar*.

18.8.5 Modification des ressources par l'utilisateur

18.8.5.1 Le fichier .Xdefaults

L'utilisateur peut créer ses propres spécifications de ressources dans le fichier .Xdefaults de son *home directory*. Ces ressources seront prises en compte par le gestionnaire de ressources à condition que la base de données du serveur X n'ait pas été initialisée par la commande **xrdb** (voir «Spécifier les ressources avec xrdb» à la page 83). De plus, ces ressources ne seront appliquées qu'aux clients locaux; les clients connectés au serveur X local mais s'exécutant sur d'autres machines ne seront pas affectés par ces spécifications de ressources.

18.8.5.2 L'option -xrm

L'option *-xrm* est une option standard X. Cette option permet de définir au niveau de la ligne de commande n'importe quelle ressource telle que celles que l'on rencontre dans les fichiers de ressources. La spécification de la ressource doit être placée entre *quotes*. Exemple:

```
aixterm –xrm 'aixterm*Foreground: blue' &
```

L'option *-xrm* spécifie uniquement une ressource pour l'instance courante de l'application. Etant donné que la plupart des clients X possèdent des options correspondant aux différentes intances de variables, l'option *-xrm* sera surtout utilisée pour spécifier une classe au niveau de la ligne de commande.

18.8.5.3 L'option -name

L'option *-name* permet d'affecter un nom à une instance d'un client X. Le nom de l'instance d'un client détermine la façon dont les ressources sont interprétées. Par exemple si le fichier .Xdefaults contient:

```
Aixterm*Font:            8x13
smallaixterm*Font:       6x10
smallaixterm*Geometry:   80x10
bigaixterm*Font:         9x15
bigaixterm*Geometry:     80x55
```

alors la commande

```
aixterm &
```

va créer une fenêtre avec les ressources par défaut, tandis que:

```
aixterm –name bigaixterm &
```

créera une fenêtre comportant 55 lignes de 80 caractères et utilisant le font 9x15. La commande:

```
aixterm –name smallaixterm &
```

créera une petite fenêtre de 10 lignes de 80 caractères et employant le *font* 6x10.

18.8.5.4 Spécifier les ressources avec xrdb

Pour l'utilisateur qui exécute des clients X sur différentes machines, l'usage du fichier $HOME/.Xdefaults pose problème car cela oblige l'utilisateur à maintenir une copie de ce fichier sur chaque machine qu'il utilise. Comme palliatif à cette gestion encombrante des multiples copies du fichier .Xdefaults, X offre à l'utilisateur la possibilité de charger dans le serveur X de sa station les ressources qu'il a définies. Les ressources ainsi stockées dans la base de données du serveur X sont accessibles à tout client X quelle que soit la machine sur laquelle il s'exécute.

Bien que la commande **xrdb** puisse être invoquée de façon interactive, elle est généralement placée dans le fichier $HOME/.xinitrc afin d'initialiser les ressources dès l'entrée en session.

Syntaxe:

xrdb [options] [dfile]

Exemples:

1. **xrdb -help** : affiche un résumé des options disponibles.
2. Si **xrdb** est utilisé sans argument, il lit les ressources sur stdin et efface le contenu précédent de la base de données.
3. **xrdb .Xresources** ou **xrdb -load .Xresources** charge dans la base de données, tout en effaçant le précédent contenu, les ressources définies dans le fichier .Xresources.
4. **xrdb -merge .Xresources**: ajoute à la base de données les définitions contenues dans le fichier .Xresources.
5. **xrdb -query**: affiche le contenu de la base de données.
6. **xrdb -edit $HOME/.Xresources**: sauve la base de données dans le fichier .Xresources du *home directory*.

Remarque : Toute modification apportée à la base de données n'a aucun effet sur les programmes en cours d'exécution, seuls les programmes dont l'exécution commencent après la modification sont affectés.

18.8.6 Synthèse des différents moyens de spécifier les ressources

Lorsqu'un client est exécuté, les sources suivantes de définitions de ressources sont consultées dans cet ordre:

1. Le fichier système avec le même nom que le client contenu dans le répertoire /usr/lib/X11/app-defaults.
2. Les fichiers du répertoire spécifié par la variable XAPPLRESDIR ou, si la variable n'a pas de valeur, du *home directory*, dont le nom est *Class*, où *Class* est le nom de classe du client.
3. La base de données du serveur X chargée par la commande **xrdb**; ces ressources sont accessibles quelle que soit la machine sur laquelle le client s'exécute. Si au-

cune ressource n'est chargée dans la base de données au moyen de **xrdb**, alors le gestionnaire de ressources utilise les ressources du fichier $HOME/.Xdefaults; ces ressources ne sont accessibles que par les clients locaux.

4. Les ressources de n'importe quel fichier spécifié par la variable XENVIRONMENT seront utilisées. Si cette variable n'est pas définie, le gestionaire de ressources cherche le fichier $HOME/.Xdefaults-hostname où *hostname* est le nom de la machine sur laquelle le client tourne. Cette méthode est utilisée pour définir des ressources spécifiques à chaque machine.

5. Toute ressource spécifiée sur la ligne de commande avec l'option *-xrm* sera utilisée pour cette instance du programme.

Toutes les ressources contenues dans ces différentes sources sont fusionnées dans l'ordre indiqué; les conflits éventuels sont résolus par les règles de priorité citées dans la section "Règles de priorité".

Finalement, si l'utilisateur a invoqué le client en spécifiant des options (autres que *-xrm*) sur la ligne de commande, ces valeurs prévaudront sur les ressources avec lesquelles elles rentrent en conflit quelle que soit leur origine.

18.9 Personnaliser le comportement de mwm (Motif Window Manager)

Le gestionnaire de fenêtres *mwm* de *Motif* offre une grande flexibilité, pratiquement toutes ses caractéristiques sont modifiables par l'utilisateur. Ainsi, il est possible de changer l'apparence du contour des fenêtres, des icônes, des menus. On peut modifier les fonctions disponibles dans les *Root Menu* et *Window Menu* et la façon dont les icônes s'organisent à l'écran. Il est possible également de créer de nouveaux menus dans la *Root Window*.

La personnalisation de *mwm* est contrôlée de deux façons :

- Par le fichier $HOME/.mwmrc.
- Par les ressources de *mwm* spécifiées au moyen d'une des méthodes présentées au paragraphe «Personnaliser le fonctionnement des applications X» à la page 81.

Des dizaines de caractéristiques de *mwm* sont modifiables par les utilisateurs, nous ne survolerons que les plus utiles. L'utilisateur intéressé par plus de détails peut consulter les pages du manuel accessibles par les commandes **man** ou **info**.

18.9.1 Réinitialiser mwm après un changement de configuration

Avant de montrer comment il est possible de changer la configuration de mwm, voyons comment rendre une nouvelle configuration active. En effet, si l'on édite le fichier .mwmrc ou bien un des fichiers de ressources tel que .Xdefaults, *mwm* ne tiendra pas compte des changements immédiatement. Si l'on stocke les ressources dans la base de données du serveur X, il ne faut pas oublier de recharger la base de données avec les nouvelles valeurs. Ensuite pour rendre les modifications effectives, il faut redémarrer *mwm* au moyen de l'option *restart* du menu racine (*Root Menu*).

18.9.2 Créer son propre fichier .mwmrc

Le comportement par défaut de *mwm* est régi en grande partie par le fichier système /usr/lpp/X11/lib/X11/system.mwmrc qui établit le contenu des *Root Menu* et *Window Menu* et quelles combinaisons de touches et boutons sont utilisées pour gérer les fenêtres. Pour modifier le comportement de *mwm*, l'utilisateur peut éditer un copie de ce fichier dans son *home directory*. La version contenue dans son *home directory* aura comme nom: .mwmrc.

Le fichier system.mwmrc comprend trois sections:

- Spécifications des menus.
- Associations de fonctions à des touches ou séquences de touches.
- Associations de fonctions aux boutons (ou combinaisons bouton/touche) de la souris.

La notion de fonction fait ici référence aux fonctions prédéfinies du gestionnaire de fenêtre. Le nom de chaque fonction prédéfinie commence par "f.". Le fichier system.mwmrc contient plusieurs associations *action/fonction* où *action* est: sélection d'une option dans un menu, frappe d'une touche, clic sur un bouton de la souris ou une combinaison des deux derniers.

La signification de la plupart des *fonctions* se déduit aisément de leur nom. Par exemple, *f.resize* change la taille de la fenêtre, *f.move* déplace la fenêtre et *f.minimize* réduit la fenêtre à une icône. D'autres fonctions ont un nom moins significatif, par exemple *f.post_wmenu* qui affiche le *Window Menu*.

Chaque fonction est décrite dans les pages du manuel de *mwm*.

Chaque section a la syntaxe suivante:

```
Section_Type Section_Title
{
definitions
}
```

Les valeurs possibles pour *Section_Type* sont *Menu*, *Keys* et *Buttons*. *Section_Title* est laissé à la discrétion de l'utilisateur.

Dans la suite nous allons voir comment modifier un menu et comment en créer un nouveau.

18.9.3 Spécifications des menus

La section concernant les spécifications des menus définit le contenu du menu racine (*Root Menu*) et du menu fenêtre (*Window Menu*). Les options de ces menus sont associées à des fonctions prédéfinies du gestionnaire de fenêtre. La syntaxe de la spécification d'un menu est :

```
Menu menu_name
{
menu items defined
}
```

Par exemple le *Root Menu* est défini dans system.mwmrc par:

```
Menu RootMenu
{
    "Root Menu"         f.title
    no-label            f.separator
    "New Window"        f.exec "aixterm &"
    "Shuffle Up"        f.circle_up
    "Shuffle Down"      f.circle_down
    "Refresh"           f.refresh
    no-label            f.separator
    "Restart..."        f.restart
     "Quit"             f.quit_mwm
}
```

La syntaxe de la spécification du menu racine est donc très simple, chaque option du menu est définie par une ligne ayant le format:

```
"label" function
```

Chacune de ces lignes permet d'ajouter dans le menu une option dont le nom est *label* et d'exécuter la fonction associée lorsque cette option est sélectionnée au moyen de la souris.

18.9.4 Exemple de modification du menu racine

Nous allons modifier le menu racine de façon à y inclure un nouvelle option qui permet d'invoquer un sous-menu, ensuite nous définirons ce sous-menu. Ce sous-menu que nous appelons *Utilities*, permettra d'exécuter des utilitaires tels que **xcalc**, **xman** et **xmag**.

La fonction qui permet d'invoquer un menu est *f.menu*. Après modification du fichier system.mwmrc que nous avons copié dans notre *home directory* et renommé .mwmrc, la section concernant le Root Menu est devenue:

```
Menu RootMenu
{
    "Root Menu"         f.title
    no-label            f.separator
    "New Window"        f.exec "aixterm &"
    "Utilities"         f.menu          UtilitiesMenu
    "Shuffle Up"        f.circle_up
    "Shuffle Down"      f.circle_down
    "Refresh"           f.refresh
    no-label            f.separator
    "Restart..."        f.restart
     "Quit"             f.quit_mwm
}
```

Il faut ensuite définir le menu de nom *UtilitiesMenu*; cela est accompli par la spécification suivante:

```
Menu UtilitiesMenu
{
    "Utilities Menu"     f.title
    "Calculator"         f.exec "xcalc &"
    "Manpage Browser"    f.exec "xman &"
    "Magnify"            f.exec "xmag &"
}
```

La fonction *f.exec* permet d'exécuter un programme lorsque l'option associée est sélectionnée.

18.9.5 Modifier les ressources de mwm

Le gestionnaire de fenêtre de *Motif* possède de très nombreuses ressources modifiables par l'utilisateur. Nous renvoyons l'utilisateur aux manuels *online* pour une information complète concernant ces ressources. *mwm* étant un client X, la modification de ses ressources s'opère comme pour tout autre client X et par conséquent les méthodes vues dans la section "Personnalisation des applications X" sont applicables.

mwm possède trois catégories de ressources:

1. Ressources contrôlant l'apparence des composants de *mwm*. Les composants de *mwm* sont représentés par: les contours des fenêtres (*window frame*), les icônes, les menus et les boîtes de dialogues de confirmation.

2. Ressources spécifiques à *mwm*. Ces ressources spécifient les caractéristiques de *mwm* en tant que client X. On peut citer par exemple la politique de sélection de la fenêtre active.

3. Ressources spécifiques à un client. Ces ressources peuvent être utilisées pour modifier le comportement d'un client particulier.

Exemples de ressources associées aux composants de *mwm* :

Pour modifier la couleur *background* du contour des fenêtres de tous les clients

```
Mwm*client*background: lightblue.
```

Pour modifier la couleur *background* de tous les composants de *mwm*:

```
Mwm*background: lightgrey.
```

Exemples de ressources spécifiques à mwm

Une des ressources les plus intéressantes permet de modifier la politique de sélection de la fenêtre active. Une fenêtre est dite active lorsque les actions entreprises par l'utilisateur (par exemple les frappes au clavier) affecte le client associé à cette fenêtre. Par défaut, la sélection de la fenêtre active s'effectue en plaçant le curseur de la souris dans la fenêtre désirée et en "cliquant" dessus. Ce qui correspond à la spécification:

```
Mwm*keyboardFocusPolicy: explicit
```

Si l'on souhaite que le changement de fenêtre active s'effectue comme précédemment mais sans avoir besoin de cliquer sur le bouton de la souris, alors on utilise la définition suivante:

```
Mwm*keyboardFocusPolicy: pointer
```

Exemples de ressources spécifiques à un client

Dans l'exemple précédent, on a vu comment changer la politique de sélection de la fenêtre active. Lorsque l'on utilise l'approche *pointer*, la fenêtre active change automatiquement en fonction de la position de la souris. Cependant dans ce cas, la fenêtre active n'est pas automatiquement placée au sommet de la pile; pour qu'il en soit ainsi, il faut employer:

```
Mwm*focusAutoRaise: true
```

Bien que *focusAutoRaise* est utilisable pour affecter un client particulier, ici elle est utilisée pour tous les clients.

Pour modifier l'icône associée à l'application *xgopher*, on utilise la spécification suivante:

```
Mwm*xgopher*iconImage: filename
```

où *filename* est le nom du fichier qui contient l'image *bitmap* de l'icône.

18.9.6 Utiliser une boîte d'icônes

mwm offre la possibilité de rassembler toutes les icônes dans une fenêtre dédiée à la manipulation de ces icônes. Pour définir cette boîte d'icônes:

```
Mwm*useIconBox:True
```

Pour faire en sorte que cette fenêtre se place à l'extrême droite de l'écran, possède une largeur de 1 icône, une hauteur de 7 icônes et commence à 178 pixels du bord supérieur de l'écran:

```
Mwm*iconPlacement: top right
Mwm*iconBoxGeometry:1x7-0+178
```

19.0 Réseaux

Le SEGI développe depuis quelques années une infrastructure d'interconnexion des réseaux locaux appelée Interréseau ULgNet. Cet interréseau déborde largement du cadre AIX, puisqu'il concerne tous les systèmes Unix aussi bien que le VM, les PC et les MacIntosh. Pour plus d'informations, et notamment pour les commandes ou services qui ne seraient pas repris dans le présent guide, voir le "Guide d'Introduction à l'Interréseau ULg", SEGI, zz01-0117.

Les services réseaux présentés dans ce chapitre concernent la possibilité d'exécuter une session sur une machine à partir d'une autre, le courrier électronique, le transfert de fichiers, et la distribution d'informations.

19.1 Adresses Internet des stations

Nous donnons ci-dessous les adresses de quelques unes des machines connectées au réseau Internet et gérées directement par le SEGI.

```
Machine         Adresse symbolique       Adresse Internet

Superordinateur vm1.ulg.ac.be            139.165.32.1
RS/6000 320     aix1.segi.ulg.ac.be      139.165.32.13
SP2             sp2e1.segi.ulg.ac.be     139.165.25.11
```

Structure des adresses symboliques :

- ulg.ac.be : réseau ULg
- segi : zônes (un ou plusieurs sous-réseaux).
- vm1, aix1, sp2e1 : noms des stations.

Structures des adresses Internet :

- 139.165 : adresse du réseau ULg (Unique au monde)
- 32, 25 : adresses des sous-réseaux (attribuées par le SEGI).
- 1, 11, 13 : adresses des stations (attribuées par le gestionnaire du sous-réseau).

La correspondance entre adresses Internet et adresses symboliques est faite au niveau de *name servers*[18].

Dans la suite de ce chapitre, les termes *host*, *hostname*, *remote host*, *adresse d'une machine*, lorqu'ils sont utilisés en tant qu'identificateur d'une machine particulière, doivent être considérés comme synonymes; ils désignent une adresse symbolique ou Internet.

Une adresse de courrier électronique (*email*) est composée d'un *username* suivi du caractère @ et d'un nom de *domaine* qui, dans les cas particuliers présentés dans ce guide, se réduit toujours à une adresse symbolique.

Exemple :

```
dupont@aix1.segi.ulg.ac.be
```

19.2 Session à distance ("Remote Login")

19.2.1 Telnet, tn, tn3270

La commande **telnet**, qui peut aussi être invoquée par **tn** ou **tn3270**, permet à partir d'une machine locale d'entrer en session sur une machine distante. Le comportement exact de la commande **telnet** dépendra du nom utilisé pour l'appeler.

Syntaxe:

telnet | **tn** | **tn3270** [options] [Hostname]

Hostname peut être une adresse Internet ou une adresse symbolique. (Voir «Annexe A. Bibliographie» à la page 99, IBM GG24-3376). La commande **telnet** constitue une interface au protocole TELNET. Cette commande peut être utilisée en mode commande ou en mode input.

19.2.1.1 Mode commande

Si la commande **telnet** est entrée sans argument, alors le mode commande est activé et le *prompt* correspondant au nom utilisé apparait :

```
telnet> , tn> ou tn3270>
```

Il est possible de passer du mode input au mode commande en tapant *Ctrl-]* pour **telnet**, *Ctrl-T* pour **tn** ou *Ctrl-C* pour **tn3270**. En mode commande, l'utilisateur peut entrer une sous-commande afin d'altérer les caractéristiques de la communication entre les deux machines.

[18] Pour des détails, voir «Annexe A. Bibliographie» à la page 99, COME91.

19.2.1.2 Mode input

Lorsque la commande **telnet** est entrée avec des arguments, une connexion entre les deux machines est ouverte et on passe en mode input. Le mode input sera soit orienté caractère soit orienté ligne en fonction des caractéristiques de la machine distante.

- Mode orienté caractère : chaque caractère tapé est directement envoyé à la machine distante, qui va le traiter et le renvoyer pour être affiché sur le terminal de la machine locale.
- Mode orienté ligne : chaque caractère est affiché directement, les caractères sont envoyés seulement après avoir complété une ligne.

19.2.1.3 Négociation du type de terminal

telnet ou **tn** : négocie le type de terminal avec la machine distante et affecte à la variable TERM la valeur résultant de la négociation. Les types d'émulations possibles sont : **3270** (terminal IBM-3270), **vt100** (terminal DEC vt100) ou **none** (pas de terminal particulier émulé). Contrairement à la plupart des systèmes UNIX, l'AIX fournit une commande **telnet** capable d'émuler un terminal 3270 tout en supportant les caractères accentués.

Remarque:

```
telnet –e TerminalType
  ou
tn –e TerminalType
```

permet de choisir explicitement le type de terminal souhaité. Les valeurs possibles pour *TerminalType* sont: **3270**, **vt100** ou **none**. Il est également possible de faire ce choix en assignant la valeur souhaitée à la variable EMULATE.

tn3270 : pas de négociation, l'émulation est réalisée en mode 3270.

19.2.1.4 Configuration du clavier en émulation 3270

telnet ou **tn** : le fichier système */etc/3270keys.hft* spécifie la configuration par défault du clavier ainsi que des couleurs. Afin de personnaliser la configuration de son clavier, l'utilisateur peut créer le fichier *$HOME/.3270keys* en recopiant le fichier système et en le modifiant selon ses besoins.

tn3270 : les fichiers système */etc/map3270* et */etc/tn3270.keys* définissent la configuration par défault du clavier.

19.2.1.5 Sous-commandes

Voici une liste des sous-commandes principales qui peuvent être entrées en mode commande:

? [subcommand] : affiche le message d'aide pour la sous-commande spécifiée. Si aucune sous-commande n'est spécifiée, alors la liste de toutes les sous-commandes est fournie.

close : ferme la connexion TELNET et revient en mode commande.

display [Argument] : affiche tous les paramètres de la connexion TELNET si aucun argument n'est spécifié, sinon affiche la valeur du paramètre donné.

emulate TerminalType : spécifie le type de terminal émulé. Valeurs possibles:

- **vt100** : terminal DEC vt100
- **3270** : terminal IBM-3270
- **none** : pas de type particulier émulé.

mode Type : spécifie le type de mode input. Valeurs possibles:

- **line** : mode orienté ligne.
- **char** : mode orienté caractère.

open Host : ouvre une connexion avec la machine éloignée *Host*.

quit : ferme la connexion TELNET et quitte la commande **telnet**.

19.2.1.6 Exemple

```
telnet vm1.ulg.ac.be
```

permet d'entrer en session sur le superordinateur du SEGI, la négociation aboutit automatiquement à une émulation en mode 3270.

19.2.2 x3270

La programme **x3270** permet d'entrer en session sur un système IBM 370/390 en réalisant une émulation 3270 à partir d'un terminal X.

Le SEGI a installé deux versions différentes de ce programme; une version commerciale développée par IBM et une version *freeware* La version commerciale est installée sur les machines gw.unipc.ulg.ac.be et candice1.montefiore.ulg.ac.be. La version *freeware* est installée sur toutes les autres machines gérées par le SEGI.

19.2.2.1 Version commerciale IBM

La commande suivante permet d'entrer en session sur un système IBM 370/390 en réalisant une émulation 3270 graphique. Syntaxe:

x3270 [options] Hostname

x3270 est une application X-Windows qui réalise une émulation 3270 entre une *workstation* supportant X-Windows et un système 370/390. Si l'option **-graphics** est spécifiée lors de son appel, **x3270** fournit une émulation de type 3179-G supportant le graphisme GDDM. Le support de caractères APL2 peut être obtenu en utilisant l'option **-apl**.

x3270 étant un client X, les options standards communes aux applications X sont également utilisables par **x3270**.

Exemples

1.

```
x3270 vm1.ulg.ac.be
```

crée une session sur le superordinateur du SEGI.

2.

```
x3270 -graphics -title Super-SEGI -fg blue vm1.ulg.ac.be
```

crée une session graphique 3179-G sur le superordinateur du SEGI. La fenêtre aura le titre Super-SEGI et la couleur du texte et des graphiques sera bleue. Les options **-title** et **-fg** sont des options standards X.

19.2.2.2 Version freeware

Cette version est similaire à la version commerciale IBM sauf qu'elle ne supporte pas l'émulation graphique.

19.2.3 rlogin et rsh

La commande **rlogin** permet d'entrer en session sur une autre machine UNIX.

Syntaxe:

rlogin [options] RemoteHost

Alors que la commande **telnet** permet à partir d'un système AIX de se connecter sur la plupart des systèmes [19], la commande **rlogin** permet seulement de se connecter à un autre système UNIX [20]. La commande **telnet** est donc plus générale. Cependant la commande **rlogin** offre certains avantages par rapport à **telnet** :

- **rlogin**, à l'exception des délais de transmissions à travers le réseau, crée une session distante pratiquement transparente. Cela est possible car :
  - **rlogin** exporte une partie de l'environnement de l'utilisateur sur le système distant, tel que par exemple le type de terminal (variable TERM).
  - **rlogin** comprend certaines fonctions de contrôle du terminal, comme les caractères de contrôle de flux (typiquement Ctrl-S et Ctrl-Q).
- **rlogin** offre la possibilité à l'utilisateur de contrôler l'accès à son compte en autorisant la session à distance en se basant uniquement sur le nom de la machine distante et sur le nom de son utilisateur. Ainsi, il est possible, pour un utilisateur qui a un *login name* X sur une machine MX et un *login name* Y sur une autre machine MY, d'entrer en session sur MY à partir de MX sans entrer ni son *login name* ni son *password* sur MY. Pour y arriver l'utilisateur aura dû créer sur MY le fichier **$HOME/.rhosts** contenant :

  ```
  "Nom ou adresse de MX"    "login name X"
  ```

  Pour plus de détails, voir **man rhosts**. Pour des raisons de sécurité, les fichiers **.rhosts** doivent avoir rw------- comme droits d'accès.

La commande **rsh**, une variante de **rlogin**, permet d'invoquer un shell sur la machine UNIX distante et de lui passer les arguments de la ligne de commande en vue de leur exécution et cela sans passer par un processus de *login*.

Syntaxe:

rsh RemoteHost [options] [command]

Pour pouvoir exécuter une commande sur une machine distante au moyen de **rsh**, il faut absolument y être autorisé par l'existance sur la machine distante d'un fichier **.rhosts** adéquat.

Grâce au protocole RLOGIN qui comprend les notions UNIX de *standard input, standard output* et *standard error*, il est possible par exemple d'exécuter, sur la station aix1.segi.ulg.ac.be, la commande

```
rsh gw.unipc.ulg.ac.be ps > filename
```

qui provoque la redirection de la sortie standard de la commande **ps**, exécutée sur gw.unipc.ulg.ac.be, vers le fichier *filename* de la machine aix1.segi.ulg.ac.be.

---

[19] En effet, pratiquement tous les serveurs de session de l'Internet comprennent le protocole TELNET et supportent l'émulation vt100 ou 3270.

[20] Le protocole *rlogin*, issu du UNIX BSD 4, n'est en général supporté que par des systèmes UNIX.

19.3 Courrier électronique

19.3.1 La commande mail

La commande **mail** permet de gérer le courrier électronique.

Chaque utilisateur dispose d'une boîte aux lettres système (BS) et d'une boîte aux lettres personnelle (BP). Quand un message est envoyé par la commande **mail**, il est mémorisé dans la BS du destinataire. Quand un utilisateur lit le courrier de sa BS, toujours au moyen de la commande **mail**, il a la possibilité de le détruire ou de le sauvegarder dans un fichier particulier; s'il ne le fait pas, le courrier est automatiquement transféré dans la BP.

La BS et la BP sont localisées grâce aux variables d'environnement suivantes (voir «Environnement» à la page 25) :

- la BS est le fichier désigné par $MAIL;
- la BP est le fichier désigné par $HOME/mbox.

La variable MAILCHECK permet d'indiquer l'intervalle de temps, en secondes, au terme duquel la commande **mail** vérifie automatiquement qu'un nouveau message est arrivé dans la BS. La variable MAILMSG contient le texte du message qui avertira si une telle arrivée se produit.

Syntaxe de la commande **mail** :

mail [options] [users]

Les options et paramètres utiles dépendent de l'action entreprise :

1. Envoyer un message :

mail [-v] [-s 'sujet'] users

mail lit le message sur stdin et l'envoie aux *users* indiqués. Un *user* peut être désigné par une adresse de la forme *user[@nom de domaine*[21]*]* ou par un *alias* (voir ci-dessous la façon de définir des *alias*). L'adresse Internet, qui peut aussi se mettre sous la forme symbolique, n'est nécessaire que pour un destinataire non local. Les destinataires locaux sont ceux que l'on retrouve dans le fichier /etc/passwd.

L'option -v permet d'obtenir à l'écran les détails relatifs à la transmission du message. L'option -s permet d'indiquer le sujet du message.

Exemples :

a.

```
mail durand@segi.ulg.ac.be
Hello Durand,
ceci est un message
ctrl-d
```

b.

```
mail -v -s 'Note à Henri et Louis' henri louis < mynote
```

Dans le premier cas, le message est entré à l'écran. Dans le second, il provient du fichier mynote.

2. Consulter la BS

mail [-u login_name]

L'option -u permet de consulter la BS d'un autre utilisateur à condition que l'on bénéficie des droits d'accès appropriés. On se retrouve dans un contexte propre à la commande **mail** dont l'indicatif (*prompt character*) est **&**. On a accès aux sous-commandes de **mail** qui permettent de lire les messages, de les supprimer, de les sauvegarder, etc. On peut obtenir la documentation relatives à ces sous-commandes en tapant **?** ou **help**. On quitte le contexte **mail** en tapant **quit** ou **q**.

3. Consulter la BP

mail -f

On se retrouve dans le même contexte que ci-dessus.

L'utilisateur a la possibilité de mémoriser dans un fichier des sous-commandes de **mail** qui seront exécutées chaque fois que cette commande sera invoquée. Le fichier doit s'appeler **.mailrc** et résider dans le *home directory*. L'exemple suivant montre la façon de définir des alias dans le fichier .mailrc :

```
alias liste1 durand@segi.ulg.ac.be  \
             henri@gw.unipc.ulg.ac.be    \
             louis@gw.unipc.ulg.ac.be
alias jean   dupont@vm1.ulg.ac.be
```

[21] Rappelons qu'une adresse symbolique est un cas particulier de nom de domaine.

19.3.2 Pine

Bien que la commande **mail** de l'AIX possède de nombreuses fonctionnalités, l'usage de cette commande est trop souvent jugé par la plupart des utilisateurs comme peu commode, voir désagréable. Afin de pallier les inconvénients de la commande **mail**, le SEGI recommande l'utilisation d'un autre logiciel de courrier électronique : Pine.

Pine est un logiciel *freeware* développé par l'Université de Washington aux USA (http://www.cac.washington.edu/pine). Pine, bien que complexe, a été conçu dans le but principal de permettre aux utilisateurs débutants d'accéder rapidement aux bénéfices de la messagerie électronique.

Pine est installé sur tous les systèmes AIX gérés par le SEGI.

19.3.2.1 Caractéristiques principales

- Pine, grâce à son orientation plein écran et ses systèmes de menus, offre à l'utilisateur un degré de convivialité nettement supérieur à celui de la commande **mail**.
- Pine supporte le standard oficiel MIME et permet ainsi d'échanger avec tout correspondant supportant également ce standard des documents de tout type. MIME permet également le bon traitement des caractères accentués.
- Pine est également *NEWS reader* et permet donc de consulter les *newsgroup* de *NetNews (USENET)*.
- Pine supporte le protocole IMAP. Ce protocole permet aux utilisateurs sophistiqués de manipuler à partir d'une même station de travail plusieurs boîtes aux lettres résidant sur des machines différentes.

19.3.2.2 Bref aperçu de Pine

Pour démarrer Pine, il suffit d'entrer la commande **pine**. Au démarrage, Pine présente à l'utilisateur le menu principal à partir duquel il est possible de:

- Consulter l'aide *Online*.
- Composer et envoyer un message.
- Consulter la boîte aux lettres courante.
- Manipuler et consulter les différentes boîtes aux lettres ou dossiers existants ainsi que les *newsgroups*.
- Manipuler l'annuaire personnel.
- Personnaliser et configurer Pine.
- Quitter Pine.

En plus de l'aide générale disponible à partir du menu principal, une aide contextuelle relativement complète est également disponible dans les différents sous-menus.

La plupart des caractéristiques de Pine sont personnalisables. Par exemple, par défaut la composition d'un nouveau message fait appel à l'éditeur Pico qui accompagne Pine; l'utilisateur qui le souhaite peut évidemment configurer Pine pour remplacer Pico par Emacs.

Pour de plus amples informations, nous renvoyons le lecteur à la documentation *Online*.

19.4 Transfert de fichiers

19.4.1 Transfert de fichiers avec ftp

La commande **ftp** permet d'effectuer des transferts de fichiers entre une machine locale (client FTP) et une machine distante (serveur FTP) :

ftp [options] [Hostname]

Cette commande constitue une interface au protocole FTP (*File Transfer Protocol*). FTP est le protocole de transfert de fichiers utilisé dans l'Internet et a pour but de régir les interactions entre un client et un serveur FTP.

Le protocole FTP permet le transfert de fichiers entre machines possédant des systèmes d'exploitation différents et possédant de ce fait des systèmes de fichiers de structures différentes. Malgré sa grande flexibilité, FTP ne tente pas de préserver les attributs des fichiers transférés (tel que mode de protection, dernière date de modification, ...) qui sont spécifiques à un système de fichiers particuliers. De plus, FTP fait très peu d'hypothèses quant à la structure d'un sytème de fichiers et par conséquent ne fournit aucune possibilité pour effectuer une copie récursive d'une partie d'un système hiérarchique de fichiers[22].

19.4.1.1 Sécurité

Un transfert de fichier entre un client et un serveur n'est possible que si l'utilisateur y est autorisé. Afin de vérifier le droit d'accès à ses fichiers, le serveur demande à l'utilisateur de s'identifier par le biais d'une procédure de *login* similaire à une entrée en session classique. Ainsi, si l'utilisateur exécute la commande **ftp** en spécifiant l'adresse d'une machine (*Hostname*), le client essaye de se connecter au serveur. Si la connexion réussit, le serveur demande alors à l'utilisateur d'entrer son *username* ainsi que son *password*. Si la procédure de *login* réussit la session FTP est alors ouverte et l'utilisateur est invité par le *prompt* **ftp>** à entrer des sous-commandes FTP.

---

[22] Pour copier récursivement une partie d'un système de fichiers d'un système AIX vers un autre système AIX tout en préservant les attributs des fichiers, on peut utiliser la commande **rcp**.

Remarque : L'utilisateur peut automatiser la procédure de *login* en créant dans son *home directory* un fichier **.netrc**. Pour la création et l'utilisation de ce fichier, l'utilisateur est invité à consulter la documentation AIX accessible via la commande **info**.

19.4.1.2 Sous-commandes

Si l'utilisateur exécute la commande **ftp** sans spécifier de *Hostname*, alors le *prompt* **ftp>** apparait immédiatement et attend que l'utilisateur entre une sous-commande. Pour se connecter à un serveur, on peut exécuter la sous-commandes **open**. Lorsque la connexion est réalisée, l'utilisateur est invité à compléter la procédure de *login* en donnant son *username* et son *password*.

Lorsqu'apparaît le *prompt* **ftp>** l'utilisateur peut entrer des sous-commandes pour par exemple lister le contenu d'un répertoire du serveur FTP, changer le répertoire de travail sur le serveur, transférer plusieurs fichiers en une seule opération, fermer la connexion ftp, ...

Il existe un grand nombre de sous-commandes. Dans la suite nous ne passerons en revue que les sous-commandes les plus fréquemment utilisées :

help ou ? [subcommand] : affiche un message d'aide pour la sous-commande spécifiée. Si aucune sous-commande n'est spécifiée alors la liste de toutes les sous-commandes est affichée.

cd RemoteDirectory : change le répertoire de travail du serveur FTP.

cdup : change le répertoire de travail du serveur FTP en le remplaçant par son répertoire père.

pwd : affiche le répertoire de travail courant sur le serveur FTP

open Hostname : établit une connexion avec le serveur FTP spécifié par Hostname.

close : ferme la connexion ftp sans sortir de la commande ftp.

quit : quitte la commande ftp.

dir [RemoteDirectory] [LocalFile] : écrit le contenu du répertoire distant spécifié dans le fichier local indiqué. Si le répertoire n'est pas spécifié, c'est le répertoire courant qui est utilisé. Si le fichier local n'est pas spécifié, le contenu du répertoire est affiché sur le terminal local.

get RemoteFile [LocalFile] : transfère le fichier *RemoteFile* de la machine distante sur la machine locale en le renommant *LocalFile*. Si le nom du fichier local n'est pas spécifié, le nom du fichier distant sera utilisé après avoir éventuellement été altéré par le mode courant. Ce mode est défini par les sous-commandes **case**, **ntrans** et **nmap**. Le transfert de fichier est réalisé en utilisant le mode de transfert défini par les commandes **type**, **form**, **mode** et **struct**.

put LocalFile [RemoteFile] : sauve le fichier local sur la machine distante. Si *RemoteFile* n'est pas spécifié alors *LocalFile* est utilisé pour nommer le fichier sur la machine distante. Ce nom sera éventuellement altéré par les sous-commandes **case**, **ntrans** et **nmap**. Le transfert est réalisé en utilisant le mode défini par les commandes **type**, **form**, **mode** et **struct**.

mget RemoteFiles : réalise l'expansion du nom *RemoteFiles* des fichiers de la machine distante et effectue le transfert de ces fichiers vers la machine locale. Les noms sont éventuellement altérés par le mode défini par les sous-commandes **case**, **ntrans** et **nmap**. Le mode de transfert utilisé est spécifié par les commandes **type**, **form**, **mode** et **struct**.

type [ascii | binary | ebcdic | image | local M | tenex] : définit le mode de transfert. Si aucun argument n'est spécifié, alors le mode courant est affiché. Le mode par défaut est **ascii**. Le mode à utiliser pour les fichiers comprimés est **binary**.

Remarques :

1. L'interpréteur de sous-commandes offre des facilités telles que les *macros* afin de simplifier les tâches répétitives.

2. Pour terminer une session FTP pendant le transfert d'un fichier utiliser la séquence Ctrl-C.

Pour un exemple de session FTP voir «Sessions FTP anonymes».

Remarque : Attention! Les *unformatted files* créés en VS Fortran sous VM et en XL Fortran sous AIX/6000 ne sont pas compatibles.

19.4.2 Sessions FTP anonymes

Un serveur *FTP* est dit anonyme, lorsqu'il permet à tout client FTP d'accéder en lecture (et même parfois en écriture) à certains de ses fichiers. Pour ouvrir une session FTP anonyme, le client doit s'annoncer en entrant *anonymous* comme *username* et un mot de passe suggéré par le serveur (généralement l'adresse de courrier électronique de l'utilisateur).

La plupart des fichiers que l'on rencontre dans les serveurs FTP anonymes ont des noms du type *filename.tar.Z*. L'extension *.tar* indique que le fichier est archivé (voir «Archivage de fichiers» à la page 16) et l'extension *.Z* signale que le fichier est comprimé (voir «Compactage de fichiers» à la page 17). Pour traiter de tels fichiers il est conseillé:

- d'utiliser le mode de transfert binaire, accessible au moyen de la sous-commande **binary** ou **type binary** de la commande **ftp**.
- une fois le fichier rapatrié, le décomprimer avec **uncompress** *filename.tar.Z*
- après décompression, le "désarchiver" au moyen de la commande

tar -xf *filename.tar*

Voici un exemple de session ftp avec le serveur ftp anonyme du SEGI :

```
$ ftp ftp.ulg.ac.be
Connected to aix1.segi.ulg.ac.be.
220 aix1.segi.ulg.ac.be FTP server
(Version 4.1 Sat Nov 23 12:52:09 CST 1991) ready.
Name (ftp.ulg.ac.be:minsoul): anonymous
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> cd /pub/unix
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 Opening data connexion for /bin/ls.
total 13704
-rw-r--r--   1 system    182410 Dec 03 09:50 agrep-2.04.tar.Z
-rw-r--r--   1 system     65775 Jun 04 1992  dig.2.0.tar.Z
-rw-r--r--   1 system    992429 Oct 14 11:40 elm2.4.tar.Z
-rw-r--r--   1 system    777708 Jun 04 1992  gawk-2.13.2.tar.Z
-rw-r--r--   1 system     37290 Mar 05 14:47 ph.tar.Z
drwxr-xr-x   2 system       512 Mar 08 13:50 popper
-rw-r--r--   1 system      7922 Jan 22 09:05 sudo.tar.Z
-rw-r--r--   1 system    332451 Jan 27 10:52 tn3270.tar.Z
-rw-r--r--   1 system    123623 May 25 1992  unzip41.tar.Z
-rw-r--r--   1 system    986917 Feb 04 10:26 xarchie-1.3.tar.Z
-rw-r--r--   1 system    115875 Dec 03 09:01 xgopher.1.1a.tar.Z
-rw-r--r--   1 system   1061635 Feb 16 10:40 xntp3.tar.Z
-rw-r--r--   1 system    167503 Mar 05 15:49 xph.tar.Z
-rw-r--r--   1 system   1943999 Dec 23 14:35 xv-2.21.tar.Z
-rw-r--r--   1 system    186943 May 25 1992  zip10ex.tar.Z
226 Transfer complete.
ftp> type binary
200 Type set to I.
ftp> get ph.tar.Z
200 PORT command successful.
150 Opening data connexion for ph.tar.Z (37290 bytes).
226 Transfer complete.
37290 bytes received in 0.1532 seconds (237.7 Kbytes/s)
ftp> quit
221 Goodbye.
```

Remarquer l'usage de la sous-commande **type binary** pour passer en mode binaire avant d'effectuer le transfert.

19.5 Accès aux systèmes d'informations distribués de l'Internet

19.5.1 Accès aux serveurs gopher

Pour AIX, le SEGI met à la disposition des utilisateurs les clients gopher suivants :

- **xgopher** : client *gopher* sous X-windows.
- **gopher** : client *gopher* pour terminal ASCII.

19.5.2 Accès au World Wide Web

Netscape, le client W3 le plus apprécié, est installé sur les systèmes AIX du SEGI. Il s'agit d'une application X-Windows que l'on peut démarrer en entrant la commande **netscape**.

19.6 Service d'annuaire électronique (Phonebook)

L'annuaire électronique du personnel ULg, qui est consultable via les services gopher et W3, peut aussi être directement interrogé au moyen de deux clients spécifiques installés sur les systèmes AIX du SEGI :

ph : commande orientée caractères.

xph : commande X-Windows.

20.0 Procédures d'exploitation

20.1 Disponibilité des ordinateurs du SEGI

On distingue trois périodes (A,B,C) durant lesquelles les ordinateurs du SEGI sont accessibles aux utilisateurs. Elles se caractérisent par la nature des travaux BATCH ou INTERACTIFS qui y sont autorisés (cfr. «Classes de travaux "batch"» et «Travail interactif»). Ces périodes sont:

- A : en permanence (sauf maintenance);
- B : en exploitation de type "bâtiment fermé", c'est-à-dire chaque nuit de 22 à 6 heures ainsi que les samedis, dimanches et jours fériés;
- C : selon planification avec le contrôle des opérations.

Pour raison de maintenance, l'exploitation des ordinateurs est interrompue chaque vendredi de 19 à 21 heures. Les entretiens exeptionnels (installations de matériel ou modifications système plus importantes) seront planifiées le week-end. Tous les arrêts d'exploitation sont préalablement annoncés aux utilisateurs par Email.

De plus, durant chaque période d'entretien, le message d'accueil qui est affiché dans la procédure de "login" prévient les utilisateurs qu'ils travaillent dans un environnement "dangereux" et qu'il y a lieu de clôturer leur session. Rappelons en effet que, durant ces périodes de "maintenance", les sessions peuvent être interrompues à tout instant sans mise en garde particulière de l'opérateur, avec les conséquences qui peuvent en résulter comme la perte d'un fichier ou d'un job.

20.2 Travail interactif

Tous les ordinateurs du SEGI sont accessibles en permanence (sauf maintenance); il s'agit de la période A définie ci-avant.

En ce qui concerne les machines pouvant être équipées d'un grand nombre de processeurs (cas de l'environnement SP2 installé au SEGI), l'accès à un service interactif ne s'effectuera pas sur base des adresses Internet des différents processeurs mais bien par l'intermédiaire de noms génériques derrière lesquels se cachent des processeurs banalisés. C'est ainsi que l'accès SP2 en mode interactif s'effectuera sur les noms suivants:

sp2s (.ulg.ac.be) Cette classe caractérise des sessions de type "Small" réservées aux compilations, au traitement de texte ou à la mise au point des programmes utilisateurs réalisés en mode "série" (un seul processeur disponible).

sp2m (.ulg.ac.be) Cette classe caractérise des sessions de type "Medium" réservées à l'exploitation interactive des programmes utilisateurs ainsi qu'à celle des logiciels "gros consommateurs" de ressources comme MATHEMATICA, MATLAB, REDUCE, ... Il s'agit d'une classe de type "série" (un seul processeur disponible).

sp2par (.ulg.ac.be) Cette classe est destinée aux travaux interactifs de type parallèle qui exploitent les environnements PE, PVM et PVMe (cfr. «Parallélisme» à la page 65). Les travaux qui dédicacent le CPU n'y sont pas autorisés.

Il faut préciser qu'il est de la volonté du SEGI de favoriser au maximum le "flow batch" réservé à l'exploitation d'applications mises au point en mode interactif. Au fil du temps, des limites sur les ressources disponibles en mode interactif seront mises en service ou adaptées pour tendre vers cet objectif. Nous insistons donc pour que les utilisateurs fassent l'effort d'exploiter le système "batch" qui s'est vu affecter plus de ressources que le système "interactif".

20.3 Classes de travaux "batch"

Différentes classes de travaux "batch" sont disponibles. Le tableau ci-après en donne les caractéristiques techniques et précise les périodes d'exploitation (cfr. point 1) durant lesquelles l'exécution est autorisée. Les programmes parallèles qui dédicacent le CPU ne sont autorisés que dans la classe parl.

Classe	Nb max de jobs simultanés	Nb max de processeurs en //	CPU max (1)	Période
sers	12	1	10'	A
serm	3	1	illimité	A
pars	4	4	10'/3'	A
parm	8	8	4h/30'	B
parl	2	8	illimité/illimité	C
Remarque : (1)Pour les jobs parallèles, deux limites de temps CPU sont indiquées: la première porte sur la consommation globale du job, tous processeurs confondus; la seconde indique la limite maximale autorisée sur chacun des processeurs utilisés par le job concerné.				

Tableau 11. Classes de travaux batch

20.4 Procédures de back-up

Un sauvetage hebdomadaire des répertoires des utilisateurs est effectué par le SEGI. Précisons qu'il s'agit d'une opération de "dump" physique mal adaptée à la récupération de fichiers pris individuellement mais bien à celle de "file systems" complets. Ces copies de sécurité sont inaccessibles à l'utilisateur et utilisée par le SEGI en cas de destruction d'un disque magnétique.

20.5 Conservation des fichiers à durée de vie limitée

Il s'agit ici de fichiers conservés sur disques magnétiques dans des espaces de travail mis à la disposition de l'ensemble de la communauté et malheureusement LIMITES en capacité. Sont concernés les fichiers de type TMP qui, en principe, ont une durée de vie limitée à celle du processus qui les crée, ainsi que le courrier électronique non encore transféré dans la "mailbox" de l'utilisateur et conservé dans un "spool" du système.

Les fichiers de type Email sont conservés

- 15 jours pour les comptes étudiants
- 40 jours pour les comptes scientifiques et administratifs

Les fichiers de type TMP non libérés par les utilisateurs sont conservés 24 heures par le système.

21.0 Aspects administratifs

21.1 Principes

Pour accéder aux ressources offertes par les différentes stations AIX disponibles au SEGI, un candidat utilisateur doit disposer d'un compte de travail, défini pratiquement par un "user name", faisant partie d'un "groupe" AIX (entité proche du concept de service universitaire utilisé dans le domaine mainframe) .

Les différentes machines AIX du SEGI sont regroupées dans un domaine (dit domaine NIS): il s'agit d'une fonctionnalité qui permet le partage des utilisateurs et de leurs fichiers entre les différentes machines du domaine.

Une machine principale constitue le serveur-NIS et toutes les autres en sont des "clients". L'avantage est qu'un utilisateur dispose d'UN seul compte (ouvert sur le serveur NIS) qui est connu de toutes les machines (et utilisable sur celles qui lui sont assignées par le SEGI); de même, sa "home directory" ne se trouve physiquement que sur une machine (en général le serveur) mais est accessible, sauf restriction délibérée, depuis toutes les autres .

Il faut bien distinguer les deux notions de groupe AIX et groupe NIS :

- un groupe AIX est un ensemble de comptes-users pour lequel on peut spécifier globalement des autorisations d'accès aux fichiers des membres du groupe;
- un "net-group NIS" est un ensemble de comptes-users et de netgroups auquel on attribue un jeu d'autorisations ou d'interdictions d'accès aux différentes machines d'un domaine NIS .

21.2 Ouverture de comptes

Toute ouverture ou modification d'un compte implique une procédure administrative préalable, portant sur deux aspects:

- Création / modification d'un GROUPE AIX
- Création / modification d'un COMPTE AIX

Contact : J.M. Petit (ou Contrôle des opérations)

En pratique, chaque service universitaire souhaitant utiliser une machine AIX doit d'abord se voir attribuer un nom de groupe AIX:

- forme : 8 caractères : le numéro de service traditionnel au SEGI (généralement 4 chiffres) suivi d'un nom mnémonique de 4 lettres au choix du demandeur;
- utilité: ce nom identifiera le "filesystem" contenant les "home directories" des utilisateurs du groupe; y sera associée une taille maximale d'espace-disque disponible pour le groupe (nb de blocs de 4Mb), ainsi que la définition éventuelle d'"export".

Le service demandera l'ouverture d'un ou plusieurs comptes utilisateurs (ou "user names"):

- forme : 8 caractères minuscules au choix du user; autant que possible, préférer la formule "initiale du prénom + nom de famille";
- password : on demandera de respecter les règles de sécurité suivantes :
  - 6 à 8 caractères
  - 1 chiffre au moins dans le corps du password
  - pas plus de 2 caractères identiques
  - pw différent des 4 derniers utilisés
  - renouvellement du pw tous les mois

Le SEGI conviendra avec les utilisateurs, en fonction des besoins de ceux-ci, des autorisations à associer aux groupes et comptes en matière d'accessibilité aux différentes machines.

21.3 Contrôle des consommations

Dans un premier temps, le SEGI ne mettra pas en place de processus de comptabilisation des consommations sur les systèmes AIX, ni donc de procédure de demande de crédits-calcul (comme il en existe pour le "mainframe" ES/9000). Il se limitera à contrôler "de l'extérieur" la charge des machines.

Néanmoins, il est vivement conseillé aux utilisateurs de se faire eux-mêmes une idéé de la charge qu'ils induisent; ils disposent à cet effet d'une commande **time** qui, suivie du nom de la commande à exécuter, fournit en fin d'exécution 3 informations de temps:

- real: "elapsed time", c'est-à-dire le temps "clock" écoulé entre le lancement de la commande et la fin de son exécution;
- sys: temps CPU réel en minutes-secondes de la partie du code de programme exécutée par le noyau (appels système);
- usr: temps CPU réel en minutes-secondes du reste du code.

La somme des temps usr et sys constitue le temps CPU total imputable aux utilisateurs. Ceux-ci se réfèreront aux "procédures d'exploitation" (cfr le chapitre sur la question) qui fixent les conditions et limites d'utilisation des systèmes.

Annexe A. Bibliographie

A.1 IBM

GC23-2377 Getting Started: Using RISC System/6000.

GC23-2378 Getting Started: Managing RISC System/6000.

GC23-2202 AIX for RISC System/6000, General Concepts and Procedures.

GC23-2212 AIX for RISC System/6000, Editing Concepts and Procedures.

GC23-2376 AIX for RISC System/6000, Commands Reference, volume 1.

GC23-2366 AIX for RISC System/6000, Commands Reference, volume 2.

GC23-2367 AIX for RISC System/6000, Commands Reference, volume 3.

GC23-2393 AIX for RISC System/6000, Commands Reference, volume 4.

SC09-1611 XL Fortran for AIX, Language Reference.

SC09-1610 AIX XL Fortran Compiler/6000 User's Guide.

SC09-1327 AIX XL Pascal Compiler/6000 Language Reference.

SC09-1326 AIX XL Pascal Compiler/6000 User's Guide.

SC09-1354 AIX for RISC System/6000, XL C Language Reference.

SC09-1259 AIX for RISC System/6000, XL C User's Guide.

SC23-2205 AIX for RISC System/6000, General Programming Concepts.

SC09-1470 AIX XL C++ Compiler/6000, Language Reference.

SC09-1471 AIX XL C++ Compiler/6000, Class Library Guide.

SC09-1472 AIX XL C++ Compiler/6000, User's Guide.

SC09-1538 AIX XL C++ Compiler/6000, Source Code Browser User's Guide.

SC09-1705 AIX for RISC System/6000, Optimization and Tuning Guide for Fortran, C and C++.

SH26-7226. IBM LoalLeveler User's Guide.

GC23-2203 AIX for RISC System/6000, Communication Concepts and Procedures.

GG24-3376 TCP/IP Tutorial and Technical Overview.

GG24-3676 Mainframe and Workstation NIC Sotware Compatibility.

SC23-0184 Engineering and Scientific Subroutine Library, Guide and Reference, volumes 1, 2, 3.

SH26-7228. IBM AIX Parallel Environment, Parallel Programming Subroutine Reference.

SH26-7230 IBM AIX Parallel Environment, Operation and Use.

SH26-7231 IBM AIX Parallel Environment, Installation, Administration and Diagnosis.

SH23-0019 IBM AIX PVMe User's Guide and Subroutine Reference.

A.2 SEGI

zz01-0117 Guide d'Introduction à l'Interréseau ULg, SEGI.

A.3 Divers

ABRA92 Paul W. ABRAHAMS, Bruce R. LARSON, *UNIX for the Impatient*, Addison-Wesley, 1992.

BORC89 Francis BORCEUX, *LaTeX, La Perfection dans le Traitement de Texte*, Editions Ciaco, Bruxelles, 1989.

BOUR83 S. R. BOURNE, *The UNIX System*, Addison-Wesley, 1983.

CAME91 Debra CAMERON and Bill ROSENBLATT, *Learning GNU Emacs*, O'Reilly & Associates, 1991.

COME91 Douglas E. COMER, *Internetworking with TCP/IP*, Prentice-Hall, 1991.

GILT83 Henry McGILTON, Rachel MORGAN, *Introducing the UNIX System*, McGraw-Hill, 1983.

GROF89 James GROFF et Paul WEINBERG, *UNIX, Une Approche Conceptuelle*, InterEditions, Paris, 1989.

HEAR93 Anthony C. HEARN, *REDUCE User's Manual*, Konrad-Zuse-Zentrum, Belin, 1993.

KERN84 Brian W. KERNIGHAN, Rob PIKE, *The UNIX Programming Environment*, Prentice-Hall, Inc., 1984.

KERN87 Brian W. KERNIGHAN, Denis M. RICHTIE, *Le Langage C*, Masson, 1987.

KOCH90 Stephen G. KOCHAN and Patrick H. WOOD, *UNIX Shell Programming*, Prentice-Hall, Inc., 1990. Masson, 1987.

KNUT92 Donald E. KNUTH, *The TeXbook*, Addison-Wesley, 1992.

LAMP86 Leslie LAMPORT, *LaTeX, A Document Preparation System, User's Guide and Reference Manual*, Addison-Wesley, 1986.

NEUN93 Winfried NEUN, *REDUCE User's Guide for Unix Systems*, Konrad-Zuse-Zentrum, Belin, 1993.

ORAM91 Andrew ORAM and Steve TALBOTT, *Managing Project with make*, O'Reilly and Associates, Inc., 1991.

QUER90 Valerie QUERCIA, Tim O'REILLY, *X Window System, User's Guide, OSF/Motif Edition*, O'Reilly and Associates, Inc.

SERO92 Raymond SEROUL, *Le Petit Livre de TeX*, InterEditions, 1992.

WALL91 Larry WALL and Randal L. Schwartz, *Programming Perl*, O'Reilly & Associates, Inc., 1991.

WOLF91 Stephen WOLFRAM, *Mathematica*, Addison-Wesley, Inc., 1991.

--------- Mathematica User'Guide for Unix Systems, Wolfram Research.

--------- Mathematica User'Guide for the X Front End, Wolfram Research.

--------- *Matlab Reference Guide*.

--------- *Matlab, Building a Graphical User Interface*.

--------- *Matlab Release Notes*.

--------- *Matlab, New Features Guide*.

--------- *Matlab User's Guide for Unix Workstations*.

--------- *Matlab, External Interface Guide for Unix Workstations*.

--------- *Simulink User's Guide for the X Window System* (Matlab).

--------- *Simulink Release Notes* (Matlab).

--------- *Control System Toolbox for use with Matlab*.

--------- *Signal Processing Toolbox for use with Matlab*.

--------- *Image Processing Toolbox for use with Matlab*.

--------- CA-DISSPLA User's Manual version 11.0, volumes 1 and 2, Computer Associates.

--------- CA-DISSPLA Pocket Guide version 11.0, Computer Associates.

--------- CA-DISSPLA Installation Guide release 11.0, IBM RS/6000 AIX, Computer Associates.

--------- CA DEVICE DRIVERS Installation Guide release 1.0, IBM RS/6000 AIX, Computer Associates.

--------- *PVM 3 User's Guide and Reference Manual*, ORNL/TM-12187.

--------- *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press.

C

D

T

V