COURS

UNIX.

1.INTRODUCTION.HISTORIQUE.CONNEXION.....4

1.1 Définition.....4
1.2 Historique.....5
1.3 Quelques commandes pour se faire la main.....7
1.4 Travaux pratiques.....7

2.LES FICHIERS SOUS UNIX.....9

2.1 La commande LS.....9
2.2 Les différentes sortes de fichier.....11
2.3 Structure de l'arborescence UNIX.....12
2.4 Les commandes pour se déplacer dans l'arborescence.....13
2.5 Création des répertoires.....15
2.6 Manipulation des fichiers.....15
2.7 Notion de table d'allocation et d'inode.....17
2.9 Exercices.....18

3 AUTRES COMMANDES DE BASE.....19

3.1 Commande de copie.....19
3.2 Commande de déplacement.....20
3.3 Création d'un lien.....21
3.4 La commande Find. 22
3.5 La commande grep.....23
3.5 Exercices.....24

4.LES DROITS SOUS UNIX.....25

4.1 Notion d'utilisateur.....25
4.2 Notion de groupe d'utilisateur.....27
4.3 Les protections.....30
4.4 Les commandes de modification des droits d'accès.....31
4.5 Travaux pratiques.....35

4.L'EDITEUR DE TEXTE VI.....36

4.1 Présentation.....36
4.2 Commandes pour lancer et sortir de VI.....37
4.3 Les commandes de déplacement.....37
4.4 Les commandes d'insertion.....40
4.5 Recherche de chaîne de caractères.....41
4.6 Annulation d'une commande.....41
4.7 Commandes diverses.....41
4.8 Utilisation du buffer p lors des copier et déplacer.....43
4.9 Insertion dans un fichier.....44
4.10 Sauvegarde d'une partie du fichier dans un autre.....44
4.9 Exercices.....45

5.ENVIRONNEMENT D'EXECUTION ET VARIABLES DU SHELL.....46

5.1 Les différents SHELL sous UNIX.....46
5.2 Les variables du shell.....47
5.3 Les process.....50
5.4 Options et commandes liées au process.....52
5.6 Quelques commandes de surveillance du système.....54

6.LE SHELL.....61

6.1 Définition des variables du shell.....61
6.2 Lecture de la valeur à saisir dans une variable.....63
6.3 Affectation du résultat d'une commande à une variable.....64
6.4 Les calculs.....65
6.5 Les opérateurs de comparaison.....66
6.6 Le passage de paramètres.....67
6.7 Exercices.....67

7. LES STRUCTURES DE PROGRAMMATION.....68

7.1 La commande TEST....68
7.2 La structure IF....72
7.3 EXERCICES....73
7.4 La boucle FOR....74
7.5 La boucle WHILE....76
7.6 La boucle UNTIL....78
7.7 La structure CASE....79
7.8 Exercices....80

8.LA RECURSIVITE....81

8.1 Exemple de programme récursif :....81
8.2 Exercice :....81

1.INTRODUCTION.HISTORIQUE.CONNEXION.

1.1 Définition.

UNIX est un système d'exploitation ayant comme caractéristiques d'être:

-Multi-tâches,

-Multi-utilisateurs,

-Temps partagé.

UNIX est composé d'un ensemble d'outils tel que langage de programmation (shell, C), un stack telecom de base (TCP-IP), des outils de production de documents et des outils de développement logiciel.

UNIX est un système d'exploitation paramétrable. L'executable ainsi que les paramètres d'UNIX constituent le noyau du système d'exploitation.

Plusieurs interpréteurs de commande ont été développés pour UNIX (Sh, KSH, CSH, BSH...).

A l'heure actuelle, il existe des versions ''Temps Réel'' d'UNIX.

1.2 Historique.

C'est en 1969 que commence l'épopée d'UNIX. Ken THOMPSON, père fondateur d'UNIX quitte le projet MULTICS (Projet de conception d'un système d'exploitation commun aux laboratoire BELL, à la General Electric et au Massachussetd Institute of Technologie).

Thompson, persévérant, en collaboration avec Richie, poursuit son objectif de réalisation de système d'exploitation pour « micro-processeur ». Les premiers travaux se font sur PDP7 de Digital Equipement. Faute de moyens importants, les premières versions sont rudimentaires : unix est d'abord mono-utilisateur, puis bi-utilisateur et est écrit en assembleur.

En 1971, unix est porté sur une machine plus puissante, le PDP11. Un traitement de texte et un éditeur de documentation sont développés.

En 1973, unix est réécrit en C.

En 1975, unix est commercialisé (version 6).

En 1979, unix qui ne fonctionne encore que sur PDP11, devient une référence en matière de système d'exploitation dans le monde scientifique.

Quelques années plus tard, plusieurs versions apparaissent:

-la version BERKELEY,

-les versions SYSTEM V d'ATT,

-la version XENIX de MICROSOFT sur micro-ordinateur.

Les ordinateurs équipés du système d'exploitation UNIX sont en général des mini-ordinateurs qui jouent le rôle de serveurs d'entreprise (hormis le cas de linux ou unix SCO sur micro-ordinateur PC). Plusieurs possibilités sont envisageables pour se connecter sur un serveur UNIX :

-connexion via réseau en utilisant le protocole TCP-IP via un PC,

-connexion « directe » par l'intermédiaire d'une ligne asynchrone,

-connexion depuis une autre machine UNIX.

Le système UNIX nous accueille par une banière qui nous donne des informations générales (le nom du système, la date et l'heure, des recommandations...).

Puis apparait login: qui est l'invite de connexion qui nous permet de saisir le nom d'utilisateur suivi de passwd: qui nous permet de saisir le mot de passe de l'utilisateur (s'il y en a un).

1.3 Quelques commandes pour se faire la main.

L'invite de commande (prompt) est $ pour un utilisateur et # pour un administrateur. Ce prompt peut être changé grâce à la variable PS1.

date pour connaître la date et l'heure.

who pour avoir la liste des utilisateurs connectés.

id pour connaître son identité.

tty pour connaître les caractéristiques de sa connexion.

uname permet de connaître le nom de sa machine.

pwd (print working directory) pour savoir dans quel répertoire on se situe.

ls pour lister les fichiers de ce répertoire.

contrôl-D ou exit permet de se déconnecter.

Suivant les systèmes UNIX, il existe entre 100 commandes (les commandes de base) et 600 commandes (pour AIX). Bien souvent une interface (graphique ou ascii) permet de faciliter la tâches des utilisateurs ou des administrateurs (smit ou smitty pour AIX, sysadm.sh pour unix SCO...).

Si on se trompe dans un nom de commande (faute d'orthographe, commande qui n'existe pas ou qui a été supprimée) le système répond : not found.

On peut arrêter le défilement de l'écran par control-S et le libérer par control-Q, mais sur un écran graphique il est plus facile d'utiliser la barre de déplacement dans la fenêtre.

La commande man qui veut dire manuel permet d'avoir de l'aide sur la signification des commandes ainsi que sur la syntaxe et la signification des options.

Exemple : man ls.

1.4 Travaux pratiques.

1) Utiliser la commande man pour trouver l'option de uname qui donne le nom de la machine.

2)Utiliser le commande cal pour lister le calendrier de ce mois.
Repérer la particularité de l'année 1752 (mois de septembre) grâce à la commande :

```
cal 1752
```

3) Utiliser la commande wc pour compter le nombre d'utilisateurs connectés :

```
who | wc -l
```

4) Modifier votre prompt pour y faire apparaître le répertoire dans lequel vous êtes :

```
PS1='$PWD $'
PS1='user1 $PWD :'
```

Suppléments :

5) Activation de l'historique pour le shell KSH. L'historique des commandes est validé automatiquement sous LINUX.
Pour activer l'historique des commande, on tape :

```
set -o vi
```

On récupère les anciennes commandes par les touches escape et moins (tiret) tapées simultannément.

2.LES FICHIERS SOUS UNIX.

2.1 La commande LS.

La commande ls est la commande qui permet d'obtenir le maximum d'informations sur les fichiers et répertoires.

L'option -l (informations étendues) donne les principales informations dont un administrateur a besoin.

Exemple :

ls -l

```
total    87112
dr-xr-xr-x       1 bin      bin             4528   15    aoû 1995  tmp
-rwxr-xr-x       1 root     sys            17218   06    jun 1995  2ltp_reorg
-rwx------       1 root     sys             5440   06    jun 1995  abttms
lrwxrwxrwx       1 root     adm               22   12    nov 1997  acctcom@ -> /usr/sbin/acct/acctcom
-r-xr-xr-x       1 bin      bin             3976   15    aoû 1995  acledit
lrwxrwxrwx       1 bin      bin               25   12    nov 1997  acl_edit@ -> /usr/lpp/dce/bin/acl_edit
-r-xr-xr-x       1 bin      bin             5110   09    avr 1995  aclget
-r-xr-xr-        1 bin      bin            10562   31    jul 1995  aclput
-r-xr-xr-x       1 bin      bin           100986   05    mar 1997  actdeq
-r-xr-xr-x       1 bin      bin           101026   05    mar 1997  actenq
-r-xr-xr-x       1 bin      bin           123562   05    mar 1997  acterd
-r-xr-xr-x       1 bin      bin              775   05    mar 1997  actlog
-r-xr-xr-x       1 bin      bin           100882   05    mar 1997  actlsq
-r-xr-xr-x       1 bin      bin           114738   15    aoû 1995  adb
-rwxr-xr-x       1 bin      bin            11026   15    aoû 1995  addbib
-r-xr-xr-x       1 bin      bin           103014   05    mar 1997  addent
-r-xr-xr-x       1 bin      bin            47286   14    aoû 1995  admin
-rwxr-xr-x       1 root     sys            21537    06    jun 1995  admlog
lrwxrwxrwx       1 root     system            17    12    nov 1997  aix2dos@ -> /usr/bin/charconv
-r-xr-xr-x       1 bin      bin            68876    15    aoû 1995  ali
-r-xr-xr-x      15 bin      bin             1042    10    aoû 1995  alias
-r-xr-xr-x       1 bin      bin            16266     04    aoû 1995  alog
-r-xr-xr-x       1 bin      bin           103350     05    mar 1997  amailcfg
-r-xr-xr-x       1 bin      bin            82108     17 aoû 1995  anno
-rwxr-xr-x       1 bin      bin             3599     0    jun 1995  apisestrace
-rwxr-xr-x       1 bin      bin             5119     01 jun 1995  apixaptrace
-r-xr-xr-x       1 bin      bin             4528     15    aoû 1995  apply
-r-xr-xr-x       1 bin      bin           134660     05    mar 1997  aprintcfg
-r-xr-xr-x       3 bin      bin            96640     21 aoû 1995  apropos
lrwxrwxrwx       1 bin      bin               15     12    nov 1997  ar@ -> /usr/ccs/bin/ar
lrwxrwxrwx       1 bin      bin               31     12    nov 1997  arraymgr@ -> /usr/lpp/ArrayGUIde/arraymgr.sh
lrwxrwxrwx       1 bin      bin               15     12 nov 1997  as@ -> /usr/ccs/bin/as
dr-xr-xr-x       1 bin      bin             4528     15 aoû 1995  repertoire
```

Les informations obtenues :

Le type de fichier (1 caractère) :

- pour un fichier ordinaire,

d pour un répertoire,

l pour un lien (autre nom pour un fichier ou un répertoire),

b pour un fichier spécial (accès périphérique) de type bloc,

c pour un fichier spécial (accès périphérique) de type caractère,

p pour un fichier pipe (fichier vide) qui sert à verrouiller l'accès à une ressource (ne peut être crée que par un programme en C).

Les droits d'accès (9 caractères) que nous expliqueront dans le chapitre sur les droits.

Le nombre de liens (ou de noms différents pour un même fichier. Ou dans le cas d'un répertoire le nombre de sous-répertoires de ce répertoire.

Le nom du propriétaire.

Le nom d'un groupe d'utilisateurs (qui n'est pas forcement le groupe d'appartenance du propriétaire.

La taille du fichier en octets.

La date de création ou de dernière modification.

Le nom du fichier.

2.2 Les différentes sortes de fichier.

On a coutume de dire que sous UNIX tout est fichier. Cependant il existe plusieurs catégories de fichiers, de nature complètement différentes quant à leur utilisation.

A) Les répertoires.

Un répertoire contient des fichiers ou d'autres répertoires. Les répertoires sont organisés en arborescence. L'arborescence d'unix fait référence a plusieurs partitions disque avec système de fichiers. A chaque partition correspond une arborescence et les arborescences correspondant aux différents systèmes de fichiers sont raccrochées les unes aux autres par des répertoires qui joue un rôle particulier de rattachement et que l'on appelle point de montage.

B) Les fichiers ordinaires.

Les fichiers ordinaires sont une suite d'octets. Il n'y a pas de structure particulière associée aux fichiers sous UNIX. Ces fichiers sont soit des fichiers de données, des exécutables binaire, des procédures en langage de commande, du texte...

En ce qui concerne les fichiers de texte, le saut de ligne est matérialisé par le caractère 0A (line feed).

Il n'y a pas de caractère de fin de fichier. UNIX sait que la fin de fichier est atteinte car il a mémorisé le nombre d'octets du fichier.

C) Les fichiers spéciaux.

Les fichiers dits spéciaux servent de point d'accès aux périphériques de la machine.

Un fichier spécial est vide, dès lors la taille est remplacée par des informations (2 nombres) qui vont permettre de définir l'accès au périphérique. Le premier numéro que l'on appelle MAJOR permet le calcul de l'adresse du pilote de périphérique correspondant qui est contenu dans l 'exécutable UNIX. Le deuxième numéro que l'on appelle MINOR sert à calculer l'adresse du périphérique sur la carte contrôleur.

D) Les liens.

Les fichiers liens sont des noms supplémentaires pour des fichiers (ordinaires ou répertoires) existant par ailleurs. Ils jouent le rôle de pointeur pour les accès au contenu physique du fichier. Ils servent à raccourcir les chemins d'accès ou pour permettre d'utiliser des noms différents pour le même fichier.

E) Les fichiers pipe.

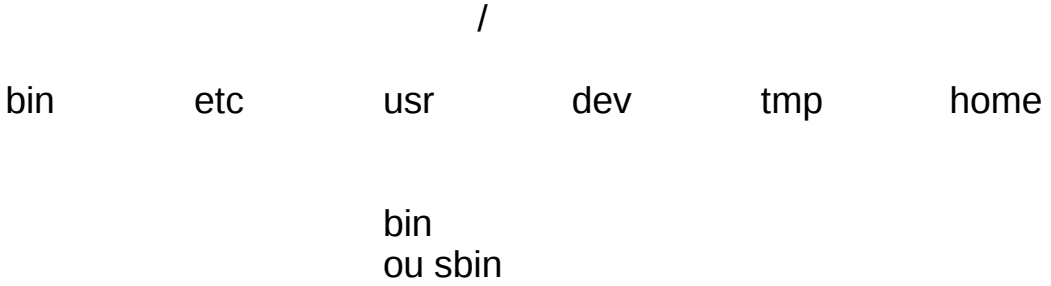
Les fichiers pipe sont vides. Ils servent à contrôler l'accès à une ressource (par exemple contrôle du démarrage du service des impressions).

2.3 Structure de l'arborescence UNIX.

Une fois installé un système UNIX comprend de nombreux fichiers et répertoires. L'arborescence commence par un sommet que l'on appelle racine (root) et que l'on note /.

L'arborescence est structurée, et les fichiers relatifs au système d'exploitation sont rangés dans plusieurs répertoires. Les noms des principaux répertoires du systèmes sont plus ou moins normalisés (en effet les noms changent un peu en ce qui concernent certains répertoires suivant le fournisseur).

Voici la structure simplifiée d'une arborescence UNIX:



Les répertoires bin et /usr/bin contiennent des commandes UNIX. bin est l'abréviation de binary.

Le répertoire /etc contient les fichiers d'administration et les procédures en langage de commandes du répertoire init.d qui sont lancées automatiquement au démarrage du système et que l'on appelle les ''run commands'' (abréviation rc telle que rc.tcpip, rc.nfs ...).

Le répertoire /usr contient les différentes parties du système d'exploitation tel que des bibliothèques partagées...

Le répertoire /dev contient les fichiers spéciaux qui sont classés par sous-répertoires en fonction du type de périphérique (disque, lignes asynchrones, cd-rom, télécommunications).

Le répertoire /tmp sert à recevoir les fichiers temporaires (car ils sont sujet à être détruits) qui voit être créés par le système lors de son utilisation (dump mémoire, fichiers créés suite à une anomalie). Ce répertoire ne doit contenir aucun fichier important car il peut être vidé à tout moment par l'administrateur. Ce répertoire (qui donne accès à un système de fichiers en général) doit toujours être suffisamment grand pour garantir le fonctionnement correct du système.

Le répertoire /user ou /u (sous AIX) sert pour créer les répertoires des différents utilisateurs.

2.4 Les commandes pour se déplacer dans l'arborescence.

La commande pour se déplacer dans l'arborescence est la commande cd (change directory). On travaille avec les noms complets ou absolus des fichiers (dans ce cas le nom du fichier est décrit depuis le sommet de l'arborescence et commence par /) ou relatif c'est à dire par rapport au répertoire dans lequel on se trouve (chemin relatif).

Syntaxe :

cd /chemin_complet
cd chemin_partiel

Exemple pour aller dans le répertoire de pierre :

cd /users/pierre
cd pierre si on se trouve déjà dans le répertoire /users.

Lorsque l'on travaille avec des noms de fichiers sous UNIX, on peut utiliser indifféremment des noms absolus ou relatifs. En général les noms

relatifs sont plus courts. Mais les noms relatifs sont plus pratiques car ils restent valables quel que soit l'endroit où l'on est positionné (il faut y penser pour les procédures shell).

La commande pwd (print working directory) permet de connaître le répertoire dans lequel on se trouve.

Lorsqu'un utilisateur se connecte sous UNIX, il ne se trouve pas positionné au hasard mais dans son répertoire d'accueil (qui se trouve en général sous /users et dans un sous-répertoire qui est le nom de l'utilisateur).

La commande cd non suivi d'un nom de répertoire permet de retourner dans son répertoire d'accueil.

Dans les noms relatif .. représente le nom du répertoire père du répertoire dans lequel on est.

Exemple :

Pierre s'il est dans son répertoire peut taper cd .. pour aller dans le répertoire /users

Il peut taper ../../etc pour aller dans le répertoire /etc

cd - permet à un utilisateur de retourner dans le répertoire dans lequel il était juste avant.

. représente le nom du répertoire courant. Pour lister les fichiers du répertoire ls -l est équivalent à ls -l .

2.5 Création des répertoires.

Les répertoires sont créés par la commande mkdir. mkdir crée automatiquement les fichiers-répertoires **.** et **..** On peut les voir grâce à la commande ls -a (l'option -a permet de voir les fichiers ou répertoires dont les noms commencent par un **.**).

Syntaxe :

mkdir /nom_complet
mkdir nom_relatif

Exemples :

mkdir /users/pierre/rep
cd /users/pierre/rep
mkdir bidon
cd bidon
ls -a

La commande rmdir permet de détruire un répertoire vide. Si le répertoire contient des fichiers on peut utiliser la commande rm -r pour détruire le répertoire et son contenu.

2.6 Manipulation des fichiers.

La longueur des noms de fichiers est variable suivant le système UNIX. La version UNIX système V a fixé cette limite à 14 caractères. Cette limite est largement dépassée dans les systèmes UNIX actuels avec lesquels des noms de plusieurs centaines de caractères peuvent être utilisés. Bien entendu plus le nom du fichier est long, plus ce nom est difficile à manipuler. Il n'y a pas de restriction en ce qui concerne les caractères qui peuvent être utilisés.

Toutefois certains caractères sont déconseillés compte tenu de leur signification particulière ($,* , ?).

$ signifie contenu de pour une variable.

* remplace n'importe quelle chaîne de caractères dans un nom de fichier.

? remplace un seul caractère dans un nom de fichier.

On peut utiliser des espaces dans les noms; dans ce cas il faudra protéger le nom du fichier par des guillemets car espace joue le rôle de séparateur dans les commandes.

Quelques commandes.

A) Les commandes cat et more permettent de visualiser le contenu d'un fichier ASCII. La commande cat ne permet pas d'arrêt entre les pages.

B) Le signe > permet de créer un fichier vide. Exemple : >fic

C) La commande cat permet également outre la visualisation de créer un petit fichier :

```
cat > fic
ligne1
ligne2...
control-D  (touches contrôle et D en début de ligne).
```

D) La commande file permet de connaître la nature d'un fichier (texte, ASCII, binaire, répertoire, fichier de commandes...).

Remarque : sur certains UNIX il existe en plus une commande pg qui permet de revenir en arrière de une ou plusieurs pages pendant la visualisation.

E) La commande rm permet de détruire un fichier. Attention la destruction d'un fichier sous UNIX est irrémédiable. Il n'y a pas de UNDO donc rm est une commande a manipuler avec précautions. On peut utiliser l'option -i qui permet d'obtenir une question de confirmation avant la destruction du fichier (répondre y pour yes si la question est en anglais et o pour oui si la question est en français, tout autre caractère est considéré comme non).

rm fic ou rm -i fic

L'option -r de la commande permet de détruire un répertoire non vide (c'est à dire avec son contenu ainsi que les sous-répertoires).
Les informations sont contenues dans l'inode du fichier.

L'option –f permet de forcer la suppression. Attention dans ce cas aucune question de confirmation n'est posée !

rm -rf /home/pierre

2.7 Notion de table d'allocation et d'inode.

Dans chaque système de fichiers, il existe une table d'allocation ou table d'inodes qui permet de recevoir les informations relatives aux fichiers ainsi que les pointeurs vers les données.

Le numéro d'inode d'un fichier est donné par la commande ls -i

Exemple : ls -i fic
534 fic

2.9 Exercices.

1) Dans quel répertoire êtes-vous connecté ?

2) Créer 2 fichiers non vides fic1 et fic2 (mettre quelques lignes de texte dedans).

3) Que fait la commande :

```
cat fic1 fic2 > fic3
```

4) Créer un répertoire et se positionner dessous.

5) Repérer le nom d'un fichier spécial et noter son numéro de major et de minor :

```
ls –l  /dev
```

6) Créer un sous-répertoire.
Se positionner dessous et repérer ce qu'il y a dedans par la commande
ls -a

3 AUTRES COMMANDES DE BASE.

3.1 Commande de copie.

La commande cp permet de dupliquer des fichiers.

Syntaxe de cette commande : cp source destination
Avec 2 variantes :

cp fichier1 fichier2

Cette syntaxe permet de copier un fichier dans un autre, dans le même répertoire ou dans un autre.

cp fichier1 fichier2 fichiern répertoire

Cette syntaxe permet de copier plusieurs fichiers à la fois. Dans ce cas où plusieurs fichiers sont copiés, le dernier argument doit être obligatoirement un nom de répertoire (différent de celui dans lesquels se trouvent les fichiers). On remarquera que la commande se comporte différemment que sous MS-DOS où l'on trouve une syntaxe de type :
copy *.doc *.old
On peut également utiliser des caractères joker sous UNIX :
cp fic* répertoire

Un fichier copié appartient à celui qui effectue la copie, et le nom du groupe du propriétaire est positionné.

3.2 Commande de déplacement.

La commande mv permet de déplacer ou de renommer un fichier. La commande mv se comporte comme la commande cp.

Syntaxe de cette commande : mv source destination
Avec 3 variantes :

mv ancien_nom nouveau_nom
mv fichier répertoire

Cette syntaxe permet de renommer un fichier, ou de le déplacer dans un autre répertoire.

mv fichier1 fichier2 fichiern répertoire

Dans ce cas le dernier argument de la commande est obligatoirement un nom de répertoire différent de celui dans lequel se trouvent les fichiers initialement.

Dernière syntaxe : on peut déplacer un répertoire (donc un bout d'arborescence) dans un autre.

mv repertoireA repertoireB
Dans ce cas repertoireA n'existe plus dans le répertoire courant après la commande. repertoireA et repertoireB doivent avoir le même père.

3.3 Création d'un lien.

La création d'un lien a pour but de donner un autre nom à un fichier ou un répertoire. Ceci permet de simplifier un chemin d'accès ou de permettre d'utilisation de plusieurs noms pour la même chose.

Il existe 2 sortes de liens :

A) les liens au sein d'un même système de fichiers (pas forcemment le même répertoire). Un seul numéro d'inode est consommé pour les différents noms pointant sur le même fichier. Il y a plusieurs noms pour un même contenu. Le contenu physique du fichier disparait quand tous les noms ont été supprimés.

Syntaxe : ln fichier nouveau_nom

B) les liens symboliques qui servent lorsque les noms de fichiers sont dans 2 systèmes de fichiers différents ou dans le cas des répertoires. Les liens symboliques sont identifiés par l comme caractère type de fichier (voir la commande ls -l). Chaque nom dans le cas d'un lien symbolique consomme un numéro d'inode.

Syntaxe : ln -s fichier nouveau_nom

Exemples :

ln fic1 fic1nombis
ln -s fic1 fic1nomter
ln -s fic1 /tmp/fic1
ln -s /usr/bin /sbin

Il convient d'utiliser avec précautions les liens (et le moins possible) car les liens sont sources de confusion, et peuvent être difficiles à gérer par la suite. Les liens ne permettent pas d'avoir des droits différents sur le fichier!

3.4 La commande Find.

La commande find permet d'effectuer des recherches dans l'arborescence d'UNIX parmi les systèmes de fichiers actifs c'est à dire ceux qui ont été rattachés à l'arborescence par la commande mount.

Find permet de rechercher :

des noms de fichiers,

les fichiers d'un utilisateur,

un fichier par son numéro d'inode.

Syntaxe :

find /repertoire_de_debut_de_recherche -option argument -print

Exemples :

Pour rechercher les fichiers hosts
find / -name hosts -print
On peut utiliser les caractères joker :
find / -name ''*log'' -print

Pour rechercher tous les fichiers d'un utilisateur :
find / -user pierre -print

Pour rechercher les fichiers qui correspondent au numéro d'inode 544 :
find / -inum 544 -print
Il peut y avoir un numéro d'inode 544 dans chaque système de fichiers. Plusieurs noms seront renvoyés pour le même inode dans le cas où des liens sont mis en oeuvre.

On peut exécuter une commande pour chaque occurrence trouvée :
find / -inum 544 -exec ls –l { } \;

3.5 La commande grep.

Cette commande permet de rechercher des mots dans un ou plusieurs fichiers. Elle peut aussi être utilisée derrière un | .

Exemples :

grep root /etc/passwd

ps –ef | grep inetd

grep mavariable *.java

Cette commande accepte des caractères joker qui s'appellent des expressions régulières. Attention la signification de ces caractères est différentes des caractères joker utilisés avec les noms de fichiers.

Signification des principaux caractères des expressions régulières :

$ matérialise la fin d'une ligne,

^ matérialise le début de la ligne,

. remplace n'importe quel caractère,

.* remplace n'importe quelle chaine de caractères,

[abcd] identifie une seule des 4 lettres parmi l'ensemble des lettres entre les crochets,

[^abcd] identifie une des lettres sauf a ou b ou c ou d,

\ annule la signification particulière d'un caractère joker.

Exemple : ls –l | grep '^d'

3.5 Exercices.

1) Copier les fichiers créés précédemment dans un sous-répertoire créé dans sous votre répertoire d'accueil.

2) Créer un lien normal. Que remarquez-vous?

3) Créer un lien symbolique. Que remarquez-vous?
Supprimer le premier nom du fichier?
Peut-on accéder au contenu en utilisant le lien symbolique?

4) Rechercher dans l'arborescence les fichiers passwd. Combien y en a-t'il? Sont-ils semblables ou a-t-on affaire à des fichiers différents?

5) Renommer votre fichier.

6) Rechercher le nom du fichier correspondant à l'inode 15333.
Existe-t'il ?

7) Copier /etc/passwd dans le répertoire courant. Que remarquez-vous?
Créer un lien dans un autre répertoire pour ce fichier.
Rechercher à nouveau les fichiers passwd. Combien y en aura-t-il maintenant?

8) Détruire quelques fichiers.

9) Peut-on copier un fichier dans lui-même?

4.LES DROITS SOUS UNIX.

4.1 Notion d'utilisateur.

Pour se connecter sur un système multi-utilisateurs tel que UNIX, il est nécessaire d'être identifié de la part du système.

La liste des utilisateurs est déclarée dans un fichier d'administration qui s'appelle /etc/passwd. Ce fichier contient des informations relatives aux utilisateurs. Chaque ligne de ce fichier permet de décrire un utilisateur et chaque ligne comporte 7 champs qui sont séparés par :.

La signification des 7 champs est la suivante :

En premier, le nom de l'utilisateur, c'est le nom qui doit être utilisé à la connexion.

Le deuxième champ est relatif au mot de passe. Ce champ contient suivant les systèmes UNIX soit le mot de passe crypté, soit un caractère de renvoi (!, x...) qui indique une référence à un autre fichier d'administration dont le nom dépend des système (/etc/shadow, /etc/security/passwd...).

Le troisième champ est le numéro de l'utilisateur. Le système travaille avec ce numéro pour identifier l'utilisateur (propriété des fichiers..).

Le quatrième champ mentionne un numéro de groupe d'utilisateurs. Les groupes d'utilisateurs sont décrits dans un autre fichier d'administration : /etc/group.

Le cinquième champ est un champ commentaire qui permet à l'administrateur du système de stocker quelques informations sur l'utilisateur (son implantation, son numéro de téléphone...).

Le sixième champ est le répertoire d'accueil de l'utilisateur, c'est à dire l'endroit où il se trouvera positionner lors d'une connexion sur le système. Ce répertoire doit exister et l'utilisateur devra posséder des droits sur celui-ci.

Le dernier champ est le programme qui sera lancé pour l'utilisateur lors après la connexion. C'est en général un interpréteur de commande : /bin/ksh ou /bin/csh.

Exemple de Fichier /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
privoxy:x:73:73::/etc/privoxy:
radvd:x:75:75:radvd user:/:/bin/false
pierre:x:500:500::/home/pierre:/bin/bash
romain:x:501:501::/home/romain:/bin/bash
oracle:x:600:600::/usr/oracle:/bin/bash
```

Exemple de Fichier /etc/shadow

```
root:$1$ùøòpÄu6µ$1L2Bk5AirLFmQOtFAkGlX.:12091:0:99999:7:::
bin:*:12091:0:99999:7:::
daemon:*:12091:0:99999:7:::
adm:*:12091:0:99999:7:::
lp:*:12091:0:99999:7:::
sync:*:12091:0:99999:7:::
shutdown:*:12091:0:99999:7:::
halt:*:12091:0:99999:7:::
mail:*:12091:0:99999:7:::
news:*:12091:0:99999:7:::
ident:!!:12091:0:99999:7:::
privoxy:!!:12091:0:99999:7:::
radvd:!!:12091:0:99999:7:::
pierre:$1$1ÎZ7ùÂzÎ$vx9zRY98WGaWhxc04W3hB0:12091:0:99999:7:::
romain:$1$gÒÁ42Éae$bFOine8ZMPU/SnL75wga31:12091:0:99999:7:::
oracle:$1$TDlsiEyc$a1GRPbNFYV8ALxEzFcKXM0:12131:0:99999:7:::
```

4.2 Notion de groupe d'utilisateur.

Pour des questions de facilités de gestion les utilisateurs peuvent être regroupés dans des groupes (un groupe principal et un/des groupe(s) secondaire(s)). Cette notion est purement organisationnel et l'affectation des utilisateurs au sein des groupes est gérée par l'administrateur du système.

Les différents groupes sont déclarés dans le fichier d'administration /etc/group. Chaque ligne permet de décrire un groupe et comprend 4 champs :

Le premier champ est le nom du groupe.

Le deuxième champ est inutilisé ou fait référence à un mot de passe qui permet à un utilisateur de changer de groupe par la commande newgrp (quand elle existe suivant les systèmes, dans ce cas l'utilisateur n'a pas la possibilité d'appartenir à des groupes secondaires). Le mot de passe qui sert à un contrôle de sécurité lorsque l'on change de groupe est positionné par la commande passgrp.

Le troisième champ est le numéro du groupe. Ce numéro est utilisé par le système pour les contrôles de droit.

Le quatrième champ est une liste d'utilisateurs.
Suivant les systèmes :

Soit une liste d'utilisateurs invités qui n'appartiennent pas augroupe mais qui ont le droit de venir dans le groupe pour bénéficier de privilèges d'accès supplémentaires.

Soit ce champ permet de matérialiser l'appartenance des utilisateurs à des groupes secondaires. Pour un groupe donné, on a la liste des utilisateurs qui « ont » ce groupe en tant que groupe secondaire.

Exemple de fichier /etc/group

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
tty:x:5:
disk:x:6:root
lp:x:7:daemon,lp
mem:x:8:
kmem:x:9:
wheel:x:10:root
mail:x:12:mail,postfix
news:x:13:news
mysql:x:27:
netdump:x:34:
ldap:x:55:
ident:x:98:
privoxy:x:73:
radvd:x:75:
pierre:x:500:
romain:x:501:
dba:x:600:
```

Il y a 2 cas de figure possibles :

L'utilisateur ne peut pas appartenir à des groupes secondaires, c'est le cas dans les UNIX pur et dur tel que définit à l'origine.

Sur les UNIX récents :
L'utilisateur peut appartenir à plusieurs groupes (secondaires). C'est le cas sous LINUX, AIX et sur d'autres systèmes.

4.3 Les protections

Comme sous tout système multi-utilisateurs, des mécanismes de protection des fichiers sont mis en œuvre.

La protection des fichiers est matérialisée par :

-la notion de propriétaire,

-la notion de groupe (ensemble d'utilisateurs),

-9 droits d'accès qui vont par groupe de 3 et que l'on visualise par la commande ls -l

Les droits sous UNIX sont de 3 sortes :

droit R (read) pour permettre la lecture d'un fichier ou du contenu d'un répertoire,

droit W (write) pour permettre l'écriture dans un fichier ou dans un répertoire,

droit X (execute) pour permettre l'exécution d'un programme ou d'un fichier shell en tapant juste son nom, ou droit de traverser un répertoire (droit de passage).

Il n'y a pas à proprement parler de notion d'héritage entre répertoire et sous-répertoires comme sous NOVELL par exemple.

Par contre pour mettre en œuvre la protection des fichiers on combine 2 niveaux de protection qui sont :

la protection du répertoire dans lequel se trouve le fichier,

la protection sur le fichier lui-même.

Les droits d'un fichier ou répertoire se trouvent dans la table d'allocation des fichiers qui s'appelle la table des inodes.

4.4 Les commandes de modification des droits d'accès.

Les droits d'accès sont positionnés sur les fichiers et répertoires vis à vis des utilisateurs du système.

La commande de changement de droits d'accès est la commande chmod. Seul le propriétaire d'un fichier ou root peut changer les droits d'accès.

Changement de droits d'accès en absolu.

Avec cette méthode on utilise une codification numérique avec 3 chiffres pour représenter les droits d'accès.

Poids binaire :	4	2	1	4	2	1	4	2	1
	R	W	X	R	W	X	R	W	X

Si le droit est positionné on prend le poids correspondant sinon 0.

	R	W	-	R	-	X	R	-	-
	1*4	1*2	0	1*4	0	1*1	1*4	0	0

syntaxe : chmod valeur fichier

Exemple : chmod 654 fic

Changement de droits d'accès en symbolique.

On utilise des codes mnémotechniques pour affecter les droits d'accès :

+ permet d'ajouter des droits,

- permet de retirer des droits,

= permet de remettre à zéro les droits

r représente le droit de lecture,

w représente le droit d'écriture,

x représente le droit d'exécution ou de passage,

u représente le propriétaire du fichier,

g représente le groupe positionné pour le fichier,

o représente les autres,

a représente u,g et o.

Syntaxe : chmod [ugoa] + [rwx] fichier

Exemple : chmod u+rwx fic
chmod u-x,o+r fic

Changement du groupe positionné sur un fichier.

La commande chgrp permet de modifier le groupe positionné sur un fichier. Cela permet aux utilisateurs d'un autre groupe de bénéficier des droits d'accès positionnés sur le groupe (deuxième série de droits). Les utilisateurs de l'ancien groupe bénéficient maintenant des droits des autres.

Syntaxe : chgrp nomdegroupe fichier

Exemple : chgrp staff fic

Changement de propriétaire d'un fichier.

Cette commande permet de changer le propriétaire d'un fichier. Sur certains systèmes seul root peut utiliser cette commande pour des raisons de sécurité.

Syntaxe : chown nomdutilisateur fichier

Exemple : chown root fic

Notion de bit suid et sgid.

Ce mécanisme qui est également un droit d'accès a pour but de fournir temporairement des privilèges supplémentaires à un utilisateur lors de l'utilisation d'une commande ou du lancement d'un exécutable.

Si l'option suid est positionné l'utilisateur prend l'identité du propriétaire du fichier pendant l'exécution.

Si l'option sgid est positionné l'utilisateur appartient au groupe positionné sur le fichier pendant l'exécution.

Exemple : la commande /bin/passwd utilise ce mécanisme pour permettre aux utilisateurs de changer leur mot de passe et d'écrire le mot de passe crypté dans le fichier des mots de passe.

4.5 Travaux pratiques.

1.Créer grâce à la commande cat un script shell qui écrit coucou sur l ‘écran, la date, la liste des utilisateurs connectés.

2. Essayer de lancer votre script shell.
Que constatez-vous ? Que manque-t’il ?

3. Rendre votre script exécutable grâce à la commande chmod.

4. Essayer de relancer votre script.
Le système a-t‘il trouvé votre fichier ? Pourquoi ?

5.Comment positionner les droits R-X—XR— sur un fichier en symbolique et en absolu.

5.Supprimer tous les droits de votre fichier de commande. Peut-on le lancer ? Pourquoi ?

6.Créer un fichier vide.

7.L’affecter à un groupe d’utilisateurs.

8.Essayer de l’affecter à un utilisateur. Est ce possible ?

9.Que fait la commande chmod uo= fic.

10.Copier le fichier /etc/passwd dans votre répertoire. Que remarquez-vous à propos des droits d’accès et de l’appartenance du fichier.

4.L'EDITEUR DE TEXTE VI.

4.1 Présentation.

VI est l'éditeur de texte d'UNIX, en tout cas celui qui est le plus souvent utiliser. Il est indispensable de savoir s'en servir pour administrer un système UNIX.

Remarque : il en existe d'autres tels que ed qui est un éditeur ligne, emacs...

VI est un éditeur pleine page. Son ergonomie peut sembler rébarbative car par défaut lorsque l'on rentre sous VI on est en mode commande et non en mode insertion. Il convient de noter que pour des raisons de compatibilité, il n'a pas évolué depuis la naissance d'UNIX.

Pour travailler avec VI il convient d'initialiser la variable TERM avec le type de son terminal (en général vt320 ou l'émulation du terminal). Ne pas oublier d'exporter la variable.

Pour travailler sous VI on utilise des commandes qui sont des touches ou des combinaisons de touches du clavier. Certaines de ces commandes commencent par : et sont tapées en bas de l'écran. La touche escape permet de passer en mode commande. La dernière ligne de l'écran sert à envoyer des messages à destination de l'utilisateur (erreurs, informations ou statistiques control G)

Il existe une version de VI qui s'appelle VIEW et qui permet de travailler en mode lecture.

4.2 Commandes pour lancer et sortir de VI.

VI est lancé par la commande vi fichier. Si le fichier n'existe pas il est créé sinon on est positionné en début de fichier.

Pour sortir de VI, il convient de taper :

ZZ ou :x ou :wq si on veut sauvegarder,

:q ou :q! pour ne pas sauvegarder.

Pour essayer créer un fichier de quelques lignes par exemple :

```
ls -l /bin > fic
```

et taper :

```
vi fic
```

4.3 Les commandes de déplacement.

On peut utiliser les touches de direction si on utilise une émulation supérieure ou égale à vt320. Il faut noter qu'à l'origine ces touches n'existaient pas sur les claviers UNIX.

Lorsque ces touches ne fonctionnent pas, on utilise les commandes :

l ou barre d'espace pour avancer,

h pour reculer,

j pour descendre ou enter,

k pour remonter ou - ,

^ pour aller en début de ligne,

$ pour aller en fin de ligne.

Commandes de déplacement par page :

touches controle et F pour avancer d'une page,

touches controle et B pour reculer d'une page,

touches controle et D pour avancer d'une demi-page,

touches controle et U pour reculer d'une demi-page,

M permet de se positionner au milieu de l'écran,

H permet de se positionner en haut de l'écran,

L permet de se positionner en bas de l'écran,

z. permet de positionner la ligne courante au milieu de l'écran,

z permet de positionner la ligne courante en haut de l'écran,

z- permet de positionner la ligne courante en bas de l'écran.

Autres déplacement :

w ou W permet de se déplacer sur le début du mot suivant,

b ou B permet de se positionner sur le début du mot courant,

e ou E permet de se positionner sur la fin du mot courant.

Vous pouvez aussi essayer ( ) { } pour vous déplacer sur les phrases mais en tenant compte des règles de ponctuation américaines.

Numérotation des lignes :

La directive :set nu permet de numéroter les lignes du fichier. Les numéros de lignes n'apparaissent qu'à l'écran et ne feront pas partie du fichier une fois sauvé.

:set nonu permet d'enlever cette numérotation.

On peut se déplacer sur un numéro de lignes par:

:N pour aller sur la ligne N,

:0 ou :1 pour aller en début de fichier,

:$ ou G pour aller en fin de fichier.

Certaines directives peuvent être placées dans le fichier .exrc de manière à être prise en compte de façon automatique lors de l'ouverture d'un fichier.

4.4 Les commandes d'insertion.

On rentre dans le mode insertion par une commande et on en sort par la touche escape.

Les commandes d'insertion sont :

a pour insérer du texte après le curseur,

i pour insérer du texte avant le curseur,

o permet d'insérer une ligne vide en dessous de la ligne courante,

O permet d'insérer une ligne vide en dessus de la ligne courante,

A permet d'insérer du texte en fin de ligne,

I permet d'insérer du texte en début de ligne.

Une insertion se termine par la touche escape. Si on ne se rappelle plus dans quel mode on est un escape nous replace en mode commande. La directive :set showmode permet l'ajout d'un pavé en bas à droite de l'écran permettant la visualisation du passage en mode insertion.

Une ligne se termine par line feed (0A, caractère non affichable) et peut avoir n'importe quelle longueur. Il n'y a pas de marque de fin de fichier sous UNIX.

4.5 Recherche de chaîne de caractères.

La commande / qui nous place en bas de l'écran permet de rechercher en avant la première occurrence d'un mot.

? fait la même chose pour une recherche en arrière.

n ou // enter permet de répéter la recherche dans le même sens qu'avant et N dans le sens inverse.

Il faut faire attention de bien se positionner au début du fichier pour effectuer la recherche car certains vi « oublie « le début du fichier si on commence la recherche au milieu.

4.6 Annulation d'une commande.

On peut annuler la dernière commande effectuée par la commande undo notée u. Il n'y a qu'un seul niveau de undo. Mais U permet d'annuler toutes les modifications faites sur une ligne.

4.7 Commandes diverses.

La commande . permet de répéter la dernière commande, ce qui évite de la retaper et d'aller plus vite.

Les touches contrôle et G frappées simultanement donne des informations statistiques sur le fichier : nombre de ligne, position dans le fichier.

La directive :! permet de lancer une commande du shell UNIX sans quitter la session VI. Une fois la commande terminée, il suffit de taper enter pour revenir à l'édition du fichier.

Exemple : ls -l

On peut passer temporairement sous un shell pour taper plusieurs commandes par :!ksh , on revient à l'édition par Control-D.

Les directives j ou J permettent de concaténer des lignes.

Exemple avec :

AB
CD
EF
GH

:1,4j donne ABCDEFGH

:3j donne AB
CD
EFGH

:j2 donne ABCDEF
GH

4.8 Utilisation du buffer p lors des copier et déplacer.

Sous VI, on peut copier du texte ou détruire du texte. Ce qui est détruit ou copié va dans un buffer, le contenu de ce buffer est récupéré par la commande p.

Les commandes de destruction sont :

dd pour détruire une ligne,

on peut détruire plusieurs lignes : exemple 5dd détruit 5 lignes,

x détruit le caractère sur lequel on est placé,

on peut détruire plusieurs caractères : 10x en détruit 10.

Commande de copie :

yy (yank-yank) permet de copier une ligne,

on copie plusieurs lignes en faisant précéder yy par le nombre de lignes à copier : 3yy copie 3 lignes dans le buffer p, pour les rappeler il faut taper la commande p.

Autres façons de copier ou déplacer :

la directive :12,17t 23 permet de copier le contenu des lignes de la ligne 12 à 17 après la ligne 23;

la directive :12,17m23 permet de déplacer les lignes 12 à 17 après la ligne 23.

4.9 Insertion dans un fichier.

La directive :r fic permet d'insérer le contenu du fichier fic dans le fichier courant à l'endroit où était positionné le curseur.

La directive :r! commande permet d'insérer le résultat de la commande dans le fichier courant à l'endroit du curseur.

4.10 Sauvegarde d'une partie du fichier dans un autre.

Il n'existe pas de sauvegarde automatique sous vi, c'est pourquoi de temps en temps il convient de sauver les modifications avec la directive :w .

Pour sauver une partie du fichier dans un autre on utilise aussi la directice w par exemple :

:10,20w ficbis sauve les lignes comprises entre la dixième et la vingtième ligne dans le fichier ficbis.

Pour ajouter à la fin du fichier ficbis on tape >> après w :
:10,20w>>ficbis

4.9 Exercices.

1) Créer un fichier contenant la liste des commandes unix.

2) Numéroter les lignes de ce fichier.

3) Compter le nombre de page.

4) Supprimer les lignes de votre fichier relatives aux commandes commençant par b.

5) Supprimer la ligne de la commande ls.

6) Annuler cette opération.

7) Supprimer la date sur la ligne de la commande mv et écrire le nom de la commande en majuscule.

8) Dire s'il existe une commande lscfg, si elle n'existe pas dupliquer une ligne pour la créer.

9) Mettre les 10 dernières lignes en début de fichier.

10) Sauver vos modifications sans sortir de vi par :w

11) Dire quelle est la taille du fichier sans sortir de l'éditeur.

12) Remplacer la commande cat par la commande type, pour se faire placer-vous sur la première lettre du mot cat et essayer la commande cw (change word).

13) Placez-vous sur une autre ligne et essayer la commande **.** ;que se passe-t-il?

14) Enlever les numéros de lignes.

15) Essayer de sortir sans sauvegarder par :q ; que constatez-vous?

16) Insérez la date en début de fichier et sauvez votre fichier sous un autre nom.

5.ENVIRONNEMENT D'EXECUTION ET VARIABLES DU SHELL.

5.1 Les différents SHELL sous UNIX.

Définition : Le shell est l'interpréteur de commandes sous UNIX. Lorsque l'on travaille dans l'environnement du shell, un prompt permet à l'utilisateur de taper des commandes. Le prompt est en général $ pour un utilisateur et # pour l'administrateur.

On peut paramétrer la chaine de caractère affichée en guise de prompt par l'intermédiaire de la variable PS1.

Exemple : PS1='machine $PWD #'

Certaines commandes complexes contiennent des sous-commandes ou des commandes internes à la commande, dans ce cas l'invite de commande est > qui est défini par la variable PS2.

Exemple : FTP>
La touche d'interruption est la combinaison de la touche contrôle et de la touche C.

Fonctionnement du shell :

Affichage du prompt $

Lecture de la commande et analyse de la syntaxe.

Création d'un processus (fork) qui reprend l'environnement c'est à dire les variables du process de l'interpréteur de commande.

Attente de la fin de l'execution de la commande.

Affichage du prompt qui correspond au prompt du shell initial.

Remarque : on peut empiler plusieurs shell les uns sur les autres. On revient au shell précèdent par la combinaison de la touche contrôle et de la touche D (séquence de touches qui permet aussi la déconnexion).

5.2 Les variables du shell.

```
BASH=/bin/bash
BASH_ENV=/root/.bashrc
COLORS=/etc/DIR_COLORS.xterm
COLORTERM=gnome-terminal
COLUMNS=80
DISPLAY=:0
EUID=0
LINES=24
LOGNAME=root
LS_COLORS='no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd=
40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.
com=00;32:*.btm=00;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31MACHTY
PE=i686-pc-linux-gnu
MAIL=/var/spool/mail/root
MAILCHECK=60
OLDPWD=/root
ORACLE_SID=CNAM
ORACLE_HOME=/usr/oracle/9.1.6
PATH=/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/
sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin
PIPESTATUS=([0]="0")
PPID=1325
PS1='[\u@\h \W]\$ '
PS2='> '
PWD=/etc
S=/etc/rc.d/init.d
SESSION_MANAGER=local/localhost.localdomain:/tmp/.ICE-unix/1210
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-
comments:monitor
SHLVL=2
SSH_AGENT_PID=1273
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SSH_AUTH_SOCK=/tmp/ssh-XXw36FUJ/agent.1210
SUPPORTED=en_US.UTF-8:en_US:en:fr_FR.UTF-8:fr_FR:fr
TERM=xterm
UID=0
USER=root
USERNAME=root
WINDOWID=23068813
XAUTHORITY=/root/.Xauthority
XMODIFIERS=@im=none
_=set
i=/etc/profile.d/which-2.sh
langfile=/root/.i18n
mc ()
{
    mkdir -p $HOME/.mc/tmp 2>/dev/null;
    chmod 700 $HOME/.mc/tmp;
    MC=$HOME/.mc/tmp/mc-$$;
    /usr/bin/mc -P "$@" >"$MC";
    cd "`cat $MC`";
    /bin/rm -f "$MC";
    unset MC
}
```

Un shell constitue un environnement qui est caractérisé par un ensemble de variables et par des privilèges.
La liste des variables définies est donnée par la commande set.

Exemple :

Une partie de ces variables et une partie seulement seront transmises aux programmes ou commandes qui seront appelés par le shell.

Les variables qui sont transmises sont dites exportées. La liste des variables exportées est donnée par la commande env ou export.

Les variables données par la commande env sont un sous-ensemble des variables données par la commande set.

Pour définir une variable on tape le nom de la variable suivi du signe égal sans laisser d'espace et de la valeur de la variable.
Dans le cas d'une chaîne de caractères la chaîne doit être protégée par des ‘ ou des ‘’ .

Exemple : var=5
CHAINE=’’ voiture de luxe ‘’

Les variables peuvent être typées de type entier ou table à une dimension. La définition du typage se fait par la commande typeset.

5.3 Les process.

Les process ou processus sont l'ensemble des programmes qui sont en cours d'éxécution sur la machine.

Un seul de l'ensemble des process utilise le CPU à un instant donnée. UNIX fonctionne en temps partagé. L'unité de partage de temps CPU est le centième de seconde (que l'on appelle tic).

La liste des process est donné par la commande ps -ef.

Exemple :

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 09:05 ?        00:00:04 init
root         2     1  0 09:05 ?        00:00:00 [keventd]
root         3     1  0 09:05 ?        00:00:00 [kapmd]
root         4     1  0 09:05 ?        00:00:00 [ksoftirqd_CPU0]
root         5     1  0 09:05 ?        00:00:00 [kswapd]
root         6     1  0 09:05 ?        00:00:00 [bdflush]
root         7     1  0 09:05 ?        00:00:00 [kupdated]
root         8     1  0 09:05 ?        00:00:00 [mdrecoveryd]
root        14     1  0 09:05 ?        00:00:00 [kjournald]
root        69     1  0 09:05 ?        00:00:00 [khubd]
root       288     1  0 09:05 ?        00:00:00 [kjournald]
root       289     1  0 09:05 ?        00:00:00 [kjournald]
root       290     1  0 09:05 ?        00:00:00 [kjournald]
root       291     1  0 09:05 ?        00:00:00 [kjournald]
root       637     1  0 09:06 ?        00:00:00 syslogd -m 0
root      1294   810  0 09:40 ?        00:00:00 fam
root      1305     1  0 09:40 ?        00:00:00 gnome-panel --sm-client-id defau
root      1307     1  0 09:40 ?        00:00:00 nautilus --no-default-window --s
root      1309     1  0 09:40 ?        00:00:00 magicdev --sm-client-id default4
root      1313     1  0 09:40 ?        00:00:00 pam-panel-icon --sm-client-id de
root      1315     1  0 09:40 ?        00:00:00 /usr/bin/python /usr/bin/rhn-app
root      1316  1313  0 09:40 ?        00:00:00 /sbin/pam_timestamp_check -d roo
root      1325     1  0 09:40 ?        00:00:01 gnome-terminal
root      1326  1325  0 09:40 pts/0    00:00:00 bash
root      1452  1326  0 10:05 pts/0    00:00:00 top
root      1455  1325  0 10:05 pts/1    00:00:00 bash
root      1491  1455  0 10:06 pts/1    00:00:00 ps –ef
```

```
  F S UID        PID  PPID  C PRI  NI ADDR    SZ WCHAN  STIME TTY          TIME CMD
100 S root         1     0  0  75   0    -   316 schedu 09:05 ?        00:00:04 init
040 S root         2     1  0  75   0    -     0 contex 09:05 ?        00:00:00 [keventd]
040 S root         3     1  0  75   0    -     0 schedu 09:05 ?        00:00:00 [kapmd]
040 S root         4     1  0  94  19    -     0 ksofti 09:05 ?        00:00:00
[ksoftirqd_CPU0]
040 S root         5     1  0  75   0    -     0 schedu 09:05 ?        00:00:00 [kswapd]
040 S root         6     1  0  85   0    -     0 bdflus 09:05 ?        00:00:00 [bdflush]
040 S root         7     1  0  75   0    -     0 schedu 09:05 ?        00:00:00 [kupdated]
040 S root         8     1  0  85   0    -     0 md_thr 09:05 ?        00:00:00 [mdrecoveryd]
000 S root      1325     1  0  75   0    -  4636 schedu 09:40 ?        00:00:01 gnome-terminal
000 S root      1326  1325  0  75   0    -  1093 wait4  09:40 pts/0    00:00:00 bash
000 S root      1452  1326  0  75   0    -   944 schedu 10:05 pts/0    00:00:00 top
000 S root      1455  1325  0  75   0    -  1093 wait4  10:05 pts/1    00:00:00 bash
000 R root      1498  1455  0  76   0    -   757 -      10:07 pts/1    00:00:00 ps -elf
```

Résultat de la commande TOP :

```
10:05am  up  1:00,  1 user,  load average: 0,00, 0,01, 0,00
72 processes: 71 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:  3,3% user,  1,1% system,  0,0% nice, 95,4% idle
Mem:   254904K av,  145084K used,  109820K free,0K shrd,   16868K buff
Swap:  522072K av,  0K used,  522072K free  65644K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 1093 root       6 -10 80196  14M  4804 S <   1,9  5,6   0:02 X
 1325 root      15   0  9476 9472  7160 S     1,7  3,7   0:00 gnome-
terminal
 1288 root      15   0  6548 6548  5352 S     0,1  2,5   0:00 metacity
 1305 root      15   0 10792  10M  8284 S     0,1  4,2   0:00 gnome-panel
    1 root      15   0   460  460   408 S     0,0  0,1   0:04 init
    2 root      15   0     0    0     0 SW    0,0  0,0   0:00 keventd
    3 root      15   0     0    0     0 SW    0,0  0,0   0:00 kapmd
    4 root      34  19     0    0     0 SWN   0,0  0,0   0:00
ksoftirqd_CPU0
    5 root      15   0     0    0     0 SW    0,0  0,0   0:00 kswapd
    6 root      25   0     0    0     0 SW    0,0  0,0   0:00 bdflush
    7 root      15   0     0    0     0 SW    0,0  0,0   0:00 kupdated
    8 root      25   0     0    0     0 SW    0,0  0,0   0:00 mdrecoveryd
   14 root      15   0     0    0     0 SW    0,0  0,0   0:00 kjournald
```

Les process sont organisés sous la forme d’une arborescence de process. Un process donne naissance à un ou plusieurs process fils.

Le père de tous les process d’un utilisateur est le process du shell de connexion de l’utilisateur.

On peut envoyer un signal à un process grace à la commande kill.

La commande kill -15 permet d’arrêter normalement un process.

La commande kill -9 permet de tuer un process.

La commande fuser permet de tuer tous les process liés à un périphérique. Elle évite d’avoir à s’intéresser aux numéros de process.

Exemple : fuser -k /dev/tty5

Une procédure shell lors de son exécution donne naissance à un process qui constitue une duplication à l’identique du shell courant (mécanisme de fork). Ce shell est empilé sur le shell précèdent. Pour lancer une procédure sans utiliser ce mécanisme de duplication il faut utiliser la commande .

Exemple : . .bash_profile

Pour lancer une procédure, il faut la rendre exécutable par la commande
chmod +x procedure
ou bien taper
bash procedure

On peut lancer une procédure en mode trace grâce à la commande:
set -x qui permet l'impression de chaque commande avant execution avec remplacement des variables par leurs valeurs.
set -v fait la même chose sans remplacer les variables par leurs valeurs.

On peut arrêter ce mécanisme par la commande set -

Ces options peuvent être passées lors du lancement du shell.

Exemple : ksh -x procedure

On peut mettre des commentaire dans un shell car au caractère # en début de ligne.

5.4 Options et commandes liées au process.

On peut programmer la touche break sur certains systèmes grâce à la commande trap.

Exemple : trap 'ls -l' 2
trap ' ' 2
trap 2

2 représente le signal 2 qui est le signal d'interruption. Le signal 1 indique le décrochement du terminal. Les signaux sont envoyés au noyau.

On lance un process en arrière plan grace à la commande &, ce qui permet de reprendre la main avant la fin du process.

Exemple : xclock &

Dans ce cas, il peut être utile d'envoyer un message de fin à l'utilisateur:
(procedure ; echo FINI) &

Si l'on ne veut pas arrêter les process lancés en arrière plan lors de la deconnexion il faut faire précéder le nom de la procédure par la commande nohup. Dans ce cas les affichages écran sont redirigés dans le fichier nohup.out.

Exemple : nohup procedure &
exit

La commande wait permet d'attendre la fin d'un process. Cette commande sert à la synchronisation inter-process.

La commande sleep permet d'attendre.

Exemple : sleep 30 permet une temporisation de 30 secondes.

5.6 Quelques commandes de surveillance du système.

Ces commandes doivent être lancées lors de la période la plus chargée néanmoins ils permettent des prises de mesures à n'importe quel moment.

Le bilan UNIX permet de récupérer les informations qui concernent l'activité de la base. Les commandes doivent être lancées par root.
La majeure partie des informations peut être obtenue avec la commande sar qui est une commande standard que l'on trouvera sur tous les systèmes UNIX.

Script SHELL verifu:
sar -u 3 3
sar -w 3 3
sar -a 3 3
sar -b 3 3
sar -c 3 3
sar -q 3 3
sar -r 3 3
sar -v 3 3
sar -y 3 3
vmstat
iostat
iostat -xtc

Exemple d'éxecution de verifu avec commentaires:

sar -u
L'option -u est la plus significative pour une lecture instantanée de la charge d'un système:

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/16/97

18:07:20   %usr   %sys   %wio   %idle
18:07:23     73     27      0       0
18:07:26     74     26      0       0
18:07:29     91      9      0       0

Average      79     21      0       0
```

%usr représente le pourcentage de temps passé en mode utilisateur pour les process du système par rapport au temps total CPU;

%sys représente le pourcentage de temps passé en mode système(mode noyau) pour les process du système par rapport au temps total CPU;

%wait représente le pourcentage de temps passé en attente d'entrées/sorties (dans la plupart des cas ce sont des accès disques), une valeur différente de 0 indique que l'on est pénalisé par les accès périphériques et donc que l'on peut optimiser ces accès en installant des périphériques plus performants (avec un meilleur débit de transfert d'informations);

%idle indique le pourcentage de temps resté libre par rapport au temps total. Ce compteur nous indique la marge de manoeuvre dont on dispose.

Le compteur le plus important est %idle. Le pourcentage de temps libre doit toujours être supérieur à zéro car une valeur nulle indique une surcharge de la machine. Il doit rester au minimum 20 à 30% de temps libre en MOYENNE (en dehors des pointes ponctuelles de surcharge: démarrage du serveur graphique...).

Si le temps libre est de 0 ou d'une valeur estimée insuffisante qui se traduira par de mauvaises performances car le système est écroulé, (même principe que pour les réseaux de type éthernet), il convient :

-d'identifier les process les plus gourmands par la commande ps -ef et de noter les valeurs de temps CPU cumulées les plus fortes (certains process peuvent avoir été mal conçus, phénomène de bouclage, problèmes systèmes liés au process), d'analyser l'aspect normal ou pas de la valeur;

-Si la surcharge persiste, la seule solution est de migrer vers un modèle de système plus puissant.

sar -w

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/01/97

16:51:48 swpin/s bswin/s swpot/s bswot/s pswch/s
16:51:51    0.00     0.0    0.00     0.0      139
16:51:54    0.00     0.0    0.00     0.0      129
16:51:57    0.00     0.0    0.00     0.0      122

Average     0.00     0.0    0.00     0.0      130
```

Le compteur pswch indique le nombre de changements de contexte par seconde c'est à dire le nombre de fois où un process passe la main à un autre process. Les changements de contexte sont gérés par le système, on ne peut pas intervenir sur le mécanisme. Pour des raisons d'efficacité les process doivent conserver la main le plus longtemps possible, cela est lié directement à la puissance machine : la valeur pswch ne doit pas dépasser 300 changements de contexte par seconde.
Swpin et swpot représente le nombre de pages swappées par seconde, des valeurs différentes de 0 indiquent que la mémoire est trop petite. En fait les systèmes unix ne doivent pas swapper car ce mécanisme est très pénalisant pour le système à cause d'une perte de temps importante due à des accès supplémentaires sur disque
.
Solution : rajouter de la mémoire si les valeurs sont différentes de 0.

sar -a

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/01/97

16:51:57  iget/s namei/s dirbk/s
16:52:00       0      10       0
16:52:03       1       3       1
16:52:06       0       3       0

Average        0       5       0
```

iget représente le nombre de demande d'accès fichiers.
namei représente le nombre d'appels à la routine de recherche d'un répertoire. On peut diminuer cette valeur en travaillant avec des noms de fichiers relatifs et en se plaçant dans les répertoires adéquats.
dirbk représente le nombre de blocs lus au niveau des répertoires.

sar -b

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/01/97

16:52:06 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
16:52:09       0       0     100       0       0     100       0       0
16:52:12       0       0     100       0       0     100       0       0
16:52:15       0       8     100       0       0     100       0       0

Average        0       3     100       0       0     100       0       0
```

sar -b représente le nombre de blocs lus (bread) et écrits (bwrit) par seconde ainsi que l'activité du cache.

sar -c

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/01/97

16:52:15 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
16:52:19     401      14       7    0.00    0.00    4707     528
16:52:22     388      15       7    0.00    0.00    3407     533
16:52:25     356      10       6    0.00    0.00    2118     567

Average      381      13       7    0.00    0.00    3404     543
```

sar -c détermine le nombre d'appels systèmes utilisés, le nombre de lectures et d'écritures. Le rapport du nombre d'appels systèmes pour les lectures (sread/s) + le nombre d'appels systèmes pour les écritures (swrit/s) par rapport au nombre d'appels systèmes total (scall/s) doit être élevé. Une valeur inférieur à 50% (en utilisation normale et significative) peut traduire une programmation non optimisée (ici 21/381) comportant beaucoup d'appels systèmes pas efficaces.

rchar/s et wchar/s permettent de calculer le nombre moyens de caractères lus ou écrits par appel système.

sar -q

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/01/97

16:52:25 runq-sz %runocc swpq-sz %swpocc
16:52:28                1.0      99
16:52:31                1.0     100
16:52:34                1.0     100

Average                 1.0      99
```

swpq est la file d'attente du swap, traduit le nombre de process prêts à reprendre la main.

runocc traduit l'activité du système (file des process en attente, si 0 le système ne fait absolument rien, ici 1% donc faible activité).

sar -r

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/01/97

16:52:34 freemem freeswap
16:52:37     188      222429
16:52:40     174      223838
16:52:43     172      223822

Average      178      223361
```

sar -r donne le nombre de pages libres :
- -en mémoire (freemem),
- -dans la zone de swap(freeswap).

sar -v

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/01/97

16:52:43  proc-sz    ov  inod-sz    ov  file-sz    ov   lock-sz
16:52:46  133/1258    0 3202/3202    0  810/810     0    0/0
16:52:49  133/1258    0 3202/3202    0  810/810     0    0/0
16:52:52  133/1258    0 3202/3202    0  810/810     0    0/0
```

Dans le même esprit que la commande précédente l'option -v présente l'état des zones mémoire dédiées à l'accès fichiers (tables des fichiers).

Le premier compteur indique le nombre d'entrées utilisées par rapport au nombre d 'entrées totales (deuxième compteur).

sar -u

```
SunOS idefix 5.4 Generic_101945-34 sun4m    07/01/97

16:52:53 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
16:52:56      57       0       0       0       0       0
16:52:59      72       0       0       0       0       0
16:53:02       0       0       1       0       0       0

Average       43       0       0       0       0       0
```

sar -u indique l'activité des files d'attente tty.
outch/s représente l'activité de la file d'attente en sortie (output queue), ce qui est affiché sur les terminaux.

rawch/s représente le nombre de caractères récupérés par seconde (clavier...) c'est la raw (caractère) queue.
canch/s représente l'activité de la file d'attente canonique qui reçoit les caractères utiles en provenance de la raw queue après élimination des caractères parasites (fautes de frappe...)

Les 3 derniers compteurs représente le nombre d'interruptions:
- -rcvin : interruptions reçues des tty,
- -xmtin : interruptions liées au retour chariot,
- -mdmin : interruptions modem.

vmstat:

```
 procs     memory          page            disk          faults          cpu
 r b w    swap  free      re   mf   pi  po  fr   de   sr  s0  s3 -- --   in   sy    cs    us sy    id
 0 0 1     704  1408      1    25   4   10  12  432   3   6   0  0  0    75 2177  194    31  19    50
```

Commande de synthèse qui donne classé par thème (mémoire,page,disk,CPU):

-le nombre de process en attente dans la run-queue (r), le nombre de process en attente de ressources (b), et le nombre de process swappés mais exécutable (w);

-le nombre de pages libres en mémoire (swap et free);

-le nombre de pages swappées avec les compteurs po (nombre de pages allant dans la partition de swap par seconde) et pi (nombre de pages rapatriées en provenance de la zone de swap par seconde);

-le pourcentage temps libre (id), temps système(sy), temps utilisateur(us).

Cette commande n'est pas disponible sur certains systèmes.

iostat:

```
Linux 2.4.18-14 (localhost.localdomain)  20.03.2003

cpu-moy:  %user   %nice    %sys   %idle
           0,57    0,00    0,29   99,14

Device:            tps    Blk_lus/s Blk_écrits/s    Blk_lus Blk_écrits
dev2-0            0,01         0,45         0,03       1704         94
dev3-0            3,95        37,53        23,57     140668      88344
```

Iostat fournit l'activité des périphériques (tty et disques).
Cette commande permet de surveiller la charge des accès disques sur les différents contrôleurs de disques et permet de prendre des décisions de réimplantations des données pour un équilibrage de charge.

6.LE SHELL.

Le shell est un langage de programmation puissant. Outre les commandes UNIX, on bénéficie des structures algorithmes des langages de programmation (boucles, structure conditionnelle...). On peut définir des variables et passer des paramètres lors d'un appel de procédure shell.

Il existe des différences entre les différents shell (sh, ksh, csh, bsh ...), donc il convient de faire attention aux problèmes d'incompatibilités et aux différences lorsque l'on passe d'un shell à un autre. Le mieux étant d'utiliser toujours le même, de préférence ksh car c'est celui qui offre le plus de possibilités.

Dans cette partie sur le shell, les mécanismes et exemples sont tirés du shell ksh.

6.1 Définition des variables du shell.

Ksh permet de déclarer des variables de type chaînes de caractères ou entiers ou tableaux à une dimension de chaînes de caractères ou d'entiers.

La définition d'une variable se fait grâce au signe égal qui signifie l'affectation. Une variable n'est pas déclarée avant l'affectation.

Il ne faut pas laisser d'espace avant et après le signe égal car un = avec un espace avant et après signifie le test d'égalité. (En shell affectation et égalité ont le même symbole). Le contenu de la variable peut être encadré par ‘ ou ‘’ .

Le typage entier permet une vérification du contenu de la variable et il faut penser à utiliser cette possibilité pour les calcul, mais ce n'est pas obligatoire.

Les commandes echo $nom_de_variable ou print $nom_de_variable affiche le contenu de la variable nom_de_variable.

Syntaxe et exemples :

var=definition
var1= ‘’definition de variable’’
var2=’definition de variable’
var3=CAMION
var4= ‘’VOITURE DE COURSE’’
var5=VOITURE DE COURSE donne en fait var5=VOITURE

typeset -i nb1=5

typeset -i donne la liste des variables entières.

tab1[0]=element1
tab1[1]=element2

Pour un tableau les indices commencent à 0 comme en langage C

echo ${tab1[1]}
Il peut être prudent d’utiliser des { et } pour délimiter de façon certaine le nom de la variable.

En fait tab1[0] est équivalent à tab1 ; c’est à dire que l’on n’a pas besoin de préciser l’indice pour le premier élément.

Integer tab2
tab2[3]=5
tab2[4]=six
donne en ksh : six bad number.
Attention le shell de LINUX bash ne dit rien !

La commande unset permet d’effacer une variable : unset var1

L’option -r de la commande typeset permet de définir une variable de type lecture seule, c’est à dire une constante.

typeset -r nom_de_variable

typeset -r var4
var4 ne peut plus être modifié après.
var4=BUS
ksh : var4 : is read only

typeset -r donne la liste des variables en lecture seule.

echo ${#tab1[*]} compte le nombre d'éléments du tableau.

print ${tab1[*]} affiche tous les éléments du tableau.

echo ${#var2} affiche la taille de var2

6.2 Lecture de la valeur à saisir dans une variable.

La commande read permet de récupérer une valeur ou une chaîne de caractère pour initialiser une variable. Dans un shell cette commande fait souvent suite à une question pour permettre la récupération de la réponse. On peut aussi s'en servir de point d'arrêt.

Syntaxe : read nom_de_variable

Exemples :

```
echo ‘’tapez une valeur : ‘’
read var
echo ‘’la valeur tapée est $var ‘’

echo ‘’ voici un menu ! ‘’
echo ‘’ choix 1 :taper A ‘’
echo ‘’ choix 2 :taper B ‘’
echo ‘’ choix 3 :taper C ‘’
echo
read choix
echo ‘’ vous avez choisi : $choix ‘’
```

6.3 Affectation du résultat d'une commande à une variable.

Un mécanisme du shell permet d'alimenter une variable avec le résultat d'une commande.

Syntaxes :

nom_de_variable=`commande`

ou

nom_de_variable=$(commande)

Exemples :

jour=`date`

fichiers=$(ls)

Il est important sous UNIX de bien faire la différence entre les divers signes de ponctuation ; à savoir :

` ponctuation utilisée pour l'affectation du résultat d'une commande à une variable,

‘ pour l'affectation et les caractères particuliers ne sont pas interprétés,

’’ pour l'affectation et les caractères tels que $ sont interprétés.

Remarque : \ annule la signification particulière d'un caractère.

6.4 Les calculs.

Le shell nous permet de faire des calculs avec les opérations arithmétiques de base : l'addition, la soustraction, la multiplication, la division et le modulo (reste de la division).

Il y a 3 syntaxes pour exprimer ces calculs :

A) avec ((et))

((a=b+c))

On remarque que dans ce cas on n'utilise pas $ pour récupérer le contenu des variables b et c.
On utilise ((et)) pour ne pas confondre une opération avec l'opérateur de groupement (et).

Exemples :

```
typeset -i a=2 b=3 c=4
((a=b+c))
echo $a
7

((a=a+3))
echo $a
10
```

B) avec l'opérateur let.

Let ''a=b+c''

On remarque que dans ce cas on n'utilise pas $ pour récupérer le contenu des variables b et c.

Exemples :

```
typeset -i a=2 b=3 c=4
let ''a=b+c''
echo $a
7

let ''a=a+3''
```

```
echo $a
10
```

C) Avec le résultat de la commande expr.

```
a=`expr $b + $c`
```

On utilise une commande UNIX et le mécanisme d'affectation du résultat d'une commande à une variable.
Les opérateurs sont : + - - * / %
Pour la multiplication \ devant * permet d'annuler la signification joker du caractère * qui remplace n'importe quelle chaine de caractère.

```
typeset -i a=2 b=3 c=4
a=`expr $b+ $c`
echo $a
7
a=`expr $a+3`
echo $a
10
```

6.5 Les opérateurs de comparaison.

On peut comparer 2 valeurs à l'aide des opérateurs de comparaison usuelles à savoir :

>supérieur
<inférieur
>= supérieur ou égal
<= inférieur ou égal
== égalité stricte
!= différence

Le résultat vaut 0 pour vrai et 1 pour faux et alimente le code de retour des commandes qui est une variable système dont le contenu est visualisé par $?

```
((a<b))
echo $ ?
1
let   ''a<b ''
```

6.6 Le passage de paramètres.

Une procédure en shell accepte le passage de paramètres. Les paramètres sont précisés lors du lancement de la commande et alimentent de façon automatique des variables systèmes de l'environnement.

La variable système $1 contient le premier paramètre.
La variable système $2 contient le deuxième paramètre.
La variable système $3 contient le troisième paramètre et ainsi de suite...

La variable système $0 contient le nom de la procédure, ce qui permet de tester le programme actuel lors d'une exécution dans une suite de programmes.

En outre la variable système $# contient le nombre de paramètres et la variable $* contient la liste de tous les paramètres.

$$ contient le numéro de process ou PID de la procédure lors de son exécution, numéro que l'on ne peut pas connaître d'avance.

Les paramètres peuvent être décalés grâce à la commande shift. Dès lors le contenu de $1 disparaît et est remplacé par le contenu de $2, le contenu de $2 est remplacé par le contenu de $3 et ainsi de suite...

Il convient de remarquer que l'on ne peut rien mettre directement dans ces variables par affectation.
Outre l'appel de la procédure, la commande set `commande` ou set $ (commande) permet d'affecter les paramètres positionnels. Ce mécanisme alimente les paramètres avec le résultat de la commande.

6.7 Exercices.

1) Ecrire une procédure shell qui permette de réaliser l'addition de deux nombres.

2) Ecrire un procédure interactive qui permette à un utilisateur de copier un fichier dans un répertoire de son choix.

7. Les structures de programmation.

7.1 La commande TEST

La commande test sert à :

récupérer la caractéristique d'un fichier (évaluation des droits d'accès,

évaluer une expression arithmétique,

comparer 2 chaînes de caractères.

Le résultat de la commande alimente la variable $?. Cette commande est essentiellement utilisée avec les structure de contrôle (if, boucle...).

$? contient le résultat de la dernière commande :

0 si la dernière commande s'est bien passée ou si le test est vrai,

un code de retour différent de 0 sinon.
Une valeur différente de 0 (en général 1) signifie également faux.

Cette commande admet 2 syntaxes :

test expression

ou

[expression]
On notera qu'il faut absolument laisser un espace après le crochet ouvrant et avant le crochet fermant.

A) Tests sur les fichiers :

test -f fic vérifie que le fichier ordinaire fic existe.

test -d rep vérifie que le fichier répertoire rep existe.

test -w fic vrai si le fichier possède les droits d'écriture.

test -x fic vrai si le fichier possède le droit d'exécution.

test -s fic vrai si le fichier existe et n'est pas vide.

test fica -ef ficbvrai si les 2 fichiers sont liés.

test fica -nt ficbvrai si fica plus récent que ficb.

test fica -ot ficb vrai si fica plus ancien que ficb.

B) Tests sur les expressions arithmétiques :

Les opérateurs utilisables sont :

-eq pour égal.

-ne pour différent.

-gt pour greater than (plus grand que).

-lt pour less than (plus petit que).

-ge pour greater or equal (supérieur ou égal).

-le pour less ou equal (inférieur ou égal).

Syntaxes :

test élément1 opérateur élément2

[élément1 opérateur élément2]

Exemples :

```
test   5   -eq   4

var=6
test  6   -ge   $var
echo $ ?
0

var1=7
var2=4
[   $var1   -lt   $var2   ]
echo $ ?
1
```

C) Tests sur les chaines de caractères.

Sur les chaînes de caractères, il existe que les opérateurs :

d'équivalence noté =

et de différence noté !=

-z pour tester si une chaîne est vide

-n pour tester qu'une chaîne n'est pas vide.

Exemples :

```
[  chaînea   =   chaîneb  ]

test  $chaînea  =   $chaîneb

test  '' $chaînea ''  =  ''$chaîneb ''

test  $SHELL   !=   /bin/ksh

test -n  ''$a ''
echo $ ?
0
```

Groupement de conditions.

Plusieurs conditions peuvent être testées en utilisant les opérateurs de regroupement :

-a pour et

-o pour ou

! pour la négation

\(et \) pour préciser l'ordre des conditions.

Exemple :

[\(-d fica -o -w fica \) -a -f ficb]

renvoie vrai si fica est un répertoire autorisé en écriture et que le fichier ordinaire ficb existe.

7.2 La structure IF.

La structure if permet de réaliser un test conditionnel : si le résultat est vrai on exécute une série d'instructions, si le résultat est faux on exécute une autre série d'instructions.
Le else n'est pas obligatoire si il n'y a qu'une série d'instructions à réaliser dans le cas vrai

Syntaxe :

```
if   commande
then  instruction1
       instruction2
       instruction3
else  instruction4
       instruction5
fi
```

On peut imbriquer plusieurs if. Dans ce cas un fi peut-être commun aux deux structures if mais cela n'est pas conseillé pour des questions de lisibilité.

```
If  commande1
then  instruction1
else if  commande2
        then instruction2
        else instruction3
        fi
fi
```

```
If  commande1
then  instruction1
elif  commande2
        then instruction2
fi
```

Exemple :

La commande peut être une commande de test ou n'importe quelle commande.

```
If  test  -d  $fic
then echo  '' $fic est un répertoire ''
else  echo  ''$fic est un fichier ordinaire. ''
fi
```

7.3 EXERCICES.

1) Améliorer la procédure de la question 2 du chapitre précédent en testant l'existence du fichier et du répertoire.

2) Ecrire une procédure qui décrive les caractéristiques du fichier passé en argument.

Les boucles.

Les boucles permettent un traitement itératif, c'est à dire d'exécuter plusieurs fois la même série d'instructions. Les instructions qui seront exécutées plusieurs fois sont dans un corps de boucle. Des mots clés identifient le début et la fin du corps de boucle.

7.4 La boucle FOR

La boucle for permet d'exécuter la même série d'instructions pour une liste de valeurs. Cette liste de valeurs est précisée dans la commande for ou on prends la liste des paramètres positionnels.

Syntaxes :

En précisant une liste de valeurs :

```
for nom_de_variable in valeur1 valeur2 valeur3
do
corps de boucle
done
```

En utilisant les paramètres positionnels :

```
for nom_de_variable in valeur1 valeur2 valeur3
do
corps de boucle
done
```

Exemples :

```
var=10
for i in 1 2 3 4 $var
do
echo $i
done
```

```
1
2
3
.....10
```

procédure contient :

```
for i
do
echo $i
done
```

procédure 2 4 6

```
2
4
6
```

7.5 La boucle WHILE

La boucle WHILE exécute la série d'instructions entre les mots clés do et done tant que la condition est vrai. La série d'instructions n'est jamais exécutée si la condition n'a jamais été vrai.

Syntaxe :

```
while condition
do instruction1
    instruction2
    instruction3
done
```

Exemples :

```
cpt=0
while  test   $cpt  -lt  10
do   echo $cpt
     ((cpt=cpt+1))
     done
```

Affichage du compteur cpt de 0 à 9 inclus.

```
while true
do  clear
     date
    sleep  60
done
```

Voici une boucle qui affiche l'heure toutes les minutes (pour les auditeurs qui s'ennuient).

```
while  ls
do  echo  « fin de la liste des fichiers »
done
```

Remarque : on peut utiliser n'importe quelle commande en guise de condition.

On peut sortir d'un niveau de boucle par la commande break. Pour sortir de plusieurs niveaux de boucles imbriquées break est suivi d'un nombre :

break 3 permet de sortir de 3 niveaux de boucle.

La commande continue permet de passer à l'indice ou l'occurrence suivante et d'abandonner le corps de boucle à l'endroit où l'instruction continue est rencontrée.

Attention , les commandes break et continue fonctionnent que pour les boucles et sont inopérantes pour les structure if et case.

Exemple :

```
while true
do
echo « taper un nombre : »
read nb
if  test  « $nb »  -eq  « 0 »
then break
fi
done
```

7.6 La boucle UNTIL

La boucle UNTIL fonctionne de la même façon que la boucle WHILE, mais la série d'instructions est exécutée tant que la condition est fausse. Si la condition a toujours été vrai, la série d'instructions n'aura jamais été exécutée.

Syntaxe :

```
until condition
do instruction1
    instruction2
    instruction3
done
```

Exemples :

```
cpt=0
until  test   $cpt  -ge  10
do   echo $cpt
     ((cpt=cpt+1))
     done
```

On affiche le compteur cpt de 0 à 9 aussi.

7.7 La structure CASE

La commande CASE permet d'exécuter une série d'instructions suivant la valeur ou une plage de valeurs d'une variable. Ceci constitue un test à choix multiples. On teste le contenu de la variable en tant que chaîne de caractères alphabétiques ou numériques. Les caractères joker du shell peuvent être utilisés pour évaluer la correspondance : * , ? [liste] et l'opérateur | qui dans ce cas signifie ou (en non un pipe). On n'exécute qu'une seule série d'instructions (la première qui est vrai) même si plusieurs des tests sont vrais.

Syntaxe :

case $variable **in**
possibilité1**)** instruction1
instruction2 **;;**
possibilité2 **)** instruction3 **;;**
possibilité3 **)** instruction4
instruction5
instruction6 **;;**
possibilité4 **)** instruction1
instruction2**;;**
* **)** instruction7**;;**
esac

Exemple :

```
cpt=3
case $cpt in
1) echo « il y a 1 dans le compteur » ; ;
2) echo « il y a 2 dans le compteur » ; ;
3) echo « il y a 3 dans le compteur » ; ;
*) echo « il y a autre chose que un, deux ou trois dans le compteur. » ; ;
esac
```

7.8 Exercices.

1) Ecrire une procédure qui affiche les caractéristiques des fichiers du répertoire un par un (utiliser une structure de boucle et la commande test).

2) Ecrire une procédure qui effectue la multiplication de 2 chiffres en utilisant une série d'additions.

3) Dupliquer le fichier dont le nom est passé en premier paramètre en autant de fois que précisé dans le deuxième paramètre. (Faire un test supplémentaire car le fichier ne doit jamais être dupliqué plus de 5 fois).

4)Ecrire un script shell qui permettra à l'utilisateur d'essayer de deviner un nombre aléatoire.
La variable système RANDOM fournit un nombre aléatoire qui change sans arrêt.

8.LA RECURSIVITE.

Le shell est un langage de programmation qui autorise la récursivité, c'est à dire qu'un programme peut s'appeler lui-même.

Mais attention la récursivité est un mécanisme gourmand et on notera qu'un process est crée à chaque appel.

8.1 Exemple de programme récursif :

Cette procédure décrit l'arborescence en listant tous les répertoires à partir du répertoire passé en paramètre.

```
if [ -d $1 ]
then echo  ''$1 est un répertoire''
        for k in $1/*
        do $0 $k
        done
fi
```

8.2 Exercice :

Ecrire la procédure qui calcule de façon récursive la somme de n nombres.

somme de 5 = 5 + 4 + 3 + 2 + 1

ou de façon récursive

somme de 5 = 5 + somme de 4
somme de 4 = 4 + somme de 3 ...