

1. Les systèmes de fichier.

Les fichiers de périphériques.

Sous Linux, il n'existe pas de lettres de lecteur mais une arborescence. Comme dans un lecteur sous Windows, les fichiers sont enregistrés dans une structure hiérarchisée de répertoires.

Nous retrouvons une seule racine qui s'appelle / à l'exception de Windows qui propose autant de racine que de disques logiques présents dans votre ordinateur.

L'accès à un périphérique quelconque, que se soit un disque dur, une carte réseau ou un port sériel, se fait par l'intermédiaire d'un répertoire présent dans la racine qui s'appelle dev.

Pour les disques:

Les disques durs IDE ont les noms suivants:

/dev/hda	Disque maître du contrôleur IDE primaire.
/dev/hdb	Disque esclave du contrôleur IDE primaire.
/dev/hdc	Disque maître du contrôleur IDE secondaire.
/dev/hdd	Disque esclave du contrôleur IDE secondaire.

Les disques durs SCSI sont désignés par les noms suivants:

/dev/sda, /dev/sdb, /dev/sdc...etc dans l'ordre des identificateurs, ID, SCSI.
L'abréviation SCSI correspond à SCSI disk.

Pour les partitions:

Les partitions sont numérotées. Les quatre partitions primaires possibles du disque /dev/hda s'appelleront donc /dev/hda1, /dev/hda2, /dev/hda3, /dev/hda4. Si on retrouve des lecteurs logiques dans une partition étendue, nous retrouverons les numéros 5, 6, 7 ... etc.

Pour les lecteurs de CDROM

Qu'ils soient en ATAPI/IDE ou en SCSI, les lecteurs de CDROM sont désignés comme les disques durs. Un lecteur CDROM IDE connecté comme esclave sur le second contrôleur IDE prendra comme nom /dev/hdd. Pour un lecteur de CDROM en SCSI,

nous aurons /dev/scd0, /dev/scd1... etc selon l'identificateur SCSI. S'il n'existe qu'un seul périphérique SCSI, le lecteur prendra toujours le nom /dev/scd0 (scd signifie SCSI CD). L'installation de la distribution SUSE crée automatiquement des liens dans le répertoire media sous le nom /dev/cdrom

On retrouvera les mêmes principes pour les périphériques suivants:

/dev/lptx pour les interfaces d'imprimantes. x prend une valeur numérique

/dev/ethx pour les cartes réseau. x prend une valeur numérique.

/dev/ttySx pour les interfaces sérieelles. x prend une valeur numérique.

Les principes des liens.

Un lien peut être vu comme un raccourci permettant de pouvoir accéder à un même fichier sous différents noms et dans différents répertoires sans devoir en créer une copie.

Pour ce faire, nous retrouvons sous Linux la notion de Inode (abréviation de Information Node). Un inode est un enregistrement dans un tableau spécial qui va décrire le fichier: son nom, le propriétaire du fichier, les droits d'accès, la taille et l'emplacement du fichier dans la partition.

Lorsque l'on crée un fichier dans un répertoire, un inode est créé et c'est un pointeur vers cet inode qui est alors créé dans votre répertoire.

On peut dès lors créer par la suite des liens menant sur le même inode. On va distinguer sous Linux des liens durs (hard links) et des liens symboliques (soft links).

Les liens durs.

La création d'un lien dur s'effectue grâce à la commande ln. La syntaxe est la suivante:

ln source cible.

exemple:

cd /home/ commande permettant de choisir /home/ comme répertoire courant

touch origine.txt commande créant un fichier vide appelé origine.txt.

ls -l commande permettant de lister le contenu du répertoire (1)

ln origine.txt liendur.txt

ls -l commande permettant de lister le contenu du répertoire (2)

(1) -rw-r--r-- 1 root root 0 Sep 18 22:41 origine.txt

(2) -rw-r--r-- 2 root root 0 Sep 18 22:41 liendur.txt

 -rw-r--r-- 2 root root 0 Sep 18 22:41 origine.txt

On peut remarquer la valeur numérique qui s'incrémente et qui indique le nombre de liens durs qui sont associés au même inode. La question que l'on peut se poser est de savoir ce qu'il risque d'arriver si on supprime le fichier original. La suppression peut s'effectuer par la commande suivante:

```
rm nom du fichier
```

```
ls -li          -i permet d' ajouter le numéro de l'inode dans la liste(1)
```

```
rm origine.txt commande supprimant le fichier origine.txt
```

```
ls -l          (2)
```

```
(1) 95827 -rw-r--r--  2 root  root      0 Sep 18 22:41 liendur.txt
```

```
    95827 -rw-r--r--  2 root  root      0 Sep 18 22:41 origine.txt
```

Les fichiers liendur.txt et origine.txt ont le même inode à savoir dans notre exemple 95827.

```
(2) -rw-r--r--  1 root  root      0 Sep 18 22:41 liendur.txt
```

On remarque que le fichier origine.txt a été supprimé mais pas liendur.txt. Le système d'exploitation se base sur le compteur d'inode pour savoir si le fichier doit être ou pas physiquement supprimé de la partition.

Pour bien se persuader que les noms pointent sur un même fichier physique, nous allons recréer un lien dur et modifier le contenu du fichier au travers d'un des liens et vérifier l'impact sur l'autre lien.

```
In liendur.txt liendur2.txt
```

```
ls -l > liendur.txt redirection de la sortie dans un fichier
```

```
ls -l          (1)
```

```
vi liendur2.txt vi est un éditeur de texte bien connu sous Unix. (2)
```

Pour sortir de l'éditeur vi, il suffit de taper :q <enter>

```
(1) -rw-r--r--  2 root  root    452 Sep 18 23:08 liendur.txt
```

```
    -rw-r--r--  2 root  root    452 Sep 18 23:08 liendur2.txt
```

Les deux noms correspondent à un fichier ayant une taille identique. Initialement liendur2.txt présentait une taille nulle!

```
(2)
```

```
total 21
```

```
drwxr-xr-x  4 root  root    192 Sep 18 23:08 .
```

```
drwxr-xr-x 19 root  root    432 Sep 18 21:24 ..
```

```
-rw-r--r--  1 root  root   18154 Sep 18 23:03 Linux.abw
```

```
drwxr-xr-x  2 root  root     80 Sep 18 18:38 Test
```

```
-rw-r--r--  2 root  root      0 Sep 18 23:08 liendur.txt
```

```
-rw-r--r--  2 root  root      0 Sep 18 23:08 liendur2.txt
```

```
drwxr-xr-x  5 willy  users    624 Sep 18 17:20 willy
```

Ces liens présentent une limitation qui découle de l'organisation des inodes: lorsque deux enregistrements de répertoire liés de cette manière possèdent le même numéro

d'inode, ce numéro étant géré séparément en fonction des partitions, une liaison par ln ne peut s'appliquer qu'à une source et une cible dans le même système de fichiers, autrement dit dans la même partition.

C'est pour contourner cette limitation qu'il existe les liens symboliques.

Les liens symboliques

Un lien symbolique peut être mis en rapport avec les raccourcis que l'on peut créer sous Windows. Un lien symbolique ne pointe pas vers un inode mais vers un fichier au travers d'un chemin. La création d'un tel lien s'effectue également par la commande ln au moyen de l'argument -s. ln -s source cible.

Reprenons notre exemple précédent dans lequel nous avons un fichier origine.txt et un lien dur. Nous décidons dans le même répertoire de créer un lien symbolique.

```
cd /home/  
touch origine.txt  
ln origine.txt liendur.txt  
ln -s origine.txt liensymb.txt      création du lien symbolique  
ls -l > origine.txt  
ls -l                               (1)
```

(1)

```
-rw-r--r--  2 root  root    535 Sep 18 23:31 liendur.txt  
lrwxrwxrwx  1 root  root     11 Sep 18 23:31 liensymb.txt -> origine.txt  
-rw-r--r--  2 root  root    535 Sep 18 23:31 origine.txt
```

On peut observer que le lien est indiqué clairement à la fois par la redirection mais également par un des drapeaux que l'on retrouve en début de ligne et qui indique un l.

Sous Linux, la suppression du fichier d'origine peut dans certains cas poser des problèmes mais dans d'autres pas. Dans le cas de notre fichier texte, l'ouverture ultérieure du fichier par son lien symbolique ne linux:/home/student # umount / home/student/floppy

```
linux:/home/student # ls -l /home/student/floppy  
total 1
```

```
drwxr-xr-x  2 root  root    48 Sep 21 19:11 .  
drwxr-xr-x  8 student users  696 Sep 21 19:37 ..
```

va naturellement pas restituer le contenu du fichier mais un nouveau fichier sera créé en correspondance avec le nom du fichier d'origine. Pour un exécutable, vous pouvez aisément comprendre que la tentative d'exécution d'un tel lien provoquera une erreur.

```
rm origine.txt  
vi liensymb.txtman
```

Les points de montage.

Tous les fichiers accessibles dans un système Unix sont arrangés dans une arborescence dont le sommet est noté '/'. Ces fichiers peuvent être répartis sur des plusieurs périphériques physiques différents. La commande **mount** va permettre d'attacher un système de fichier trouvé sur un périphérique à cette arborescence. Contrairement, la commande **umount** détachera le périphérique de cette arborescence.

Supposons que l'on ait inséré une disquette dans le lecteur et que l'on désire pouvoir en lire le contenu. Comme nous l'avons vu dans le paragraphe 1.1, le lecteur de disquette est accessible par une entrée dans le répertoire `dev` que l'on nomme `fdxxx`. `fd0` correspond au premier lecteur de disquette trouvé.

`fd0h720` correspond à une disquette 720Ko

`fd0h1440` correspond à une disquette 1.44Mo

L'accès direct à ce lien ne nous permet pas d'effectuer la lecture de la disquette qui y serait insérée. Pour que cela soit possible, il faut monter la disquette c-à-d faire en sorte que son système de fichier soit accessible par une des entrées de l'arborescence.

Syntaxe: **mount [-fnrsvw] [-t vfstype] [-o options] device dir**

-t vfstype L'argument `-t` est utilisé pour indiquer le système de fichier à utiliser. Les systèmes de fichiers supportés sont: `adfs`, `affs`, `autofs`, `coda`, `coherent`, `cramfs`, `devpts`, `efs`, `ext`, `ext2`, `ext3`, `hfs`, `hpfs`, `iso9660`, `jfs`, `minix`, `msdos`, `ncpfs`, `nfs`, `ntfs`, `proc`, `qnx4`, `reiserfs`, `romfs`, `smbfs`, `sysv`, `tmpfs`, `udf`, `ufs`, `umsdos`, `vfat`, `xenix`, `xfs`, `xiafs`.

Exemple:

Nous avons créé un répertoire `floppy` dans le dossier `/home/student/` et nous exécutons la commande suivante:

```
mount -t msdos /dev/fd0h1440 /home/student/floppy
cd /home/student/floppy
mkdir repfd
touch filefd.txt
ls -l (1)
cd ..
umount /home/student/floppy
```

(1)

```
drwxr-xr-x  3 root  root    7168 Sep 21 19:25 .
drwxr-xr-x  6 student users   648 Sep 21 19:11 ..
-rwxr-xr-x  1 root  root      0 Sep 21 19:25 filefd.txt
drwxr-xr-x  2 root  root    512 Sep 21 19:25 repfd
```

Le fichier `filefd.txt` et le répertoire `repfd` ont été créés sur la disquette.

Nous pouvons effectuer la même démarche pour un lecteur de CDROM ou pour une seconde partition non montée par le système d'exploitation. Nous avons créé un répertoire `windows` correspondant à la partition sur laquelle le système d'exploitation

windows est installée (en fat32) et un répertoire cdrom. La partition contenant le système d'exploitation windows est la première partition primaire située sur le disque dur installé en esclave sur le premier contrôleur IDE.

On pourra donc accéder au lien /dev/hdb1.

Le lecteur de cdrom est accessible par un lien symbolique créé dans le répertoire /dev/ qui est cdrom.

```
mount -t vfat /dev/hdb1 /home/student/windows
mount -t iso9660 /dev/cdrom /home/student/cdrom
```

Création d'un système de fichier.

Si vous désirez accéder à une partition en la montant dans un des répertoires de votre arborescence, il faut qu'au préalable vous y ayez créé un système de fichier. La commande que vous pouvez utiliser est **mkfs** (make filesystem).

Outre les systèmes de fichiers correspondant à d'autres systèmes d'exploitation dans les lesquels on retrouve entre autres vfat (fat32) et msdos (fat16), on retrouvera des systèmes de fichiers propres à Linux (Unix): Ext2, Ext3 ReiserFS, JFS, XFS.

Lorsque l'on évoque un système de fichiers, on retrouve couramment des termes tels que metadata et journal.

Metadata: Un système de fichier a une structure de données interne qui assure que les données sur le disque sont correctement organisées et accessibles. Essentiellement, c'est une donnée à propos de données. Chaque système de fichiers a sa propre structure de metadata, ce qui explique en partie pourquoi les systèmes de fichiers peuvent avoir des caractéristiques en performance sensiblement différentes. C'est un des aspects importants de maintenir les meta données intactes du fait qu'autrement l'entiereté des fichiers système pourraient être corrompus.

Journal: Dans le contexte des systèmes de fichiers, un journal signifie une structure contenant un fichier d'activité où le pilote assurant la gestion du système de fichier enregistre les modifications apportées aux méta données.

Un journal va naturellement réduire le temps de recouvrement d'un système Linux du fait que le pilote n'aura pas à réaliser une recherche des méta données corrompues mais ne devra tenir compte que des modifications apportées à celles ci en se basant sur le journal. Ext2 n'est pas un système de fichier journalisé; Ext3, Reiser FS, JFS et XFS le sont.

Voici les conclusions d'un article que vous pouvez trouver sur le site <http://bulmalug.net/body.phtml?nIdNoticia=1167&nIdPage=7>.

XFS et ReiserFS ont de très bonnes performances comparativement à ext2fs, pourtant largement testé et optimisé. Ext3 semble être un peu plus lent même si il se rapproche de ext2. XFS, ReiserFS et Ext3 ont démontré qu'ils sont d'excellents et de très fiables systèmes de fichiers. XFS est résolument performant pour les E/S de très gros fichiers, spécialement vis à vis de son adversaire le plus proche, ReiserFS. Pour le travail sur

de petits fichiers, typiquement entre 100 et 10 000 octets, ReiserFS a démontré les meilleurs résultats, lorsque les fichiers concernés n'étaient pas encore dans le cache (comme pendant le démarrage par exemple). Dans le cas où le système lit des fichiers qui sont déjà cachés en mémoire vive, la différence est en pratique négligeable pour tous les systèmes (ext2, Ext3, ReiserFS et XFS). D'autre part, JFS a obtenu les résultats les moins bons de tous les tests comparatifs, non seulement en termes de performance mais aussi à cause de problèmes stabilité. Pour ceux qui seraient désireux de migrer d'un système de fichier Ext2 non journalisé vers un système de fichier journalisé, Ext3 est tout à fait indiqué du fait de son incroyable facilité qui s'explique du fait que Ext3 s'appuie sur la même structure de méta données. Pour les autres systèmes, il faut sauvegarder et réinstaller 'from scratch'.

Nous retrouvons sous Linux des variantes à mkfs pour les formats précités sous la forme d'autres utilitaires dont les noms ne demandent que peu de commentaires: mkdosfs, mke2fs, mkfs.vfat, mkfs.xfs. Ces utilitaires intègrent plus de paramètres qui sont propres à chaque format. Nous attirons votre attention que cet utilitaire installe un système de fichiers mais qu'il ne procède pas au formatage de bas niveau. Le formatage de bas niveau s'effectue grâce à la commande fdformat.

Exemple:

```
fdformat /dev/fd0h1440
mkdosfs /dev/fd0h1140
mount /dev/fd0h1140 /home/student/floppy (1)
cd /home/student/floppy
ls -l (2)
mkdir test
touch file.txt
ls -l (3)
cd ..
umount /home/student/floppy
ls -l /home/student/floppy (4)
```

(1) Il est important que le répertoire floppy existe au préalable.

(2) Le contenu de la disquette est vide.

```
linux:/home/student/floppy # ls -l
```

```
total 8
drwxr-xr-x  2 root  root    7168 Jan  1  1970 .
drwxr-xr-x  8 student users   696 Sep 21 19:37 ..
```

(3) la disquette associée au répertoire floppy contient le répertoire créé ainsi que le fichier.

```
drwxr-xr-x  3 root  root    7168 Sep 22 20:15 .
drwxr-xr-x  8 student users   696 Sep 21 19:37 ..
-rwxr-xr-x  1 root  root      0 Sep 22 20:15 file.txt
drwxr-xr-x  2 root  root    512 Sep 22 20:15 test
```

```
(4)linux:/home/student # umount /home/student/floppy
```

```
linux:/home/student # ls -l /home/student/floppy
```

```
total 1
```

```
drwxr-xr-x  2 root  root    48 Sep 21 19:11 .
```

```
drwxr-xr-x  8 student users 696 Sep 21 19:37 ..
```

Étant donné que le lecteur a été démonté, on retrouve le contenu du répertoire floppy présent sur le disque dur. Celui ci était à l'origine vide.

2.Manipulation des fichiers.

Introduction.

Nous allons étudier dans un premier temps les différentes commandes nous permettant de nous déplacer dans l'arborescence du système et celles permettant de déplacer des fichiers, de les lister, de les filtrer, de les supprimer...

L'étude de ces commandes sera exhaustive mais nous allons analyser les différents moyens existents sous Linux d'obtenir de l'aide.

a) La commande **man** (manuel) suivie du nom (de commande dans certains cas) sur lequel on désire de l'aide.

Supposons que l'on désire de l'aide sur la commande ls:

man ls

L'affichage est alors en partie le suivant:

```
LS(1)           User Commands           LS(1)
```

```
NAME
```

```
ls - list directory contents
```

```
SYNOPSIS
```

```
ls [OPTION]... [FILE]...
```

```
DESCRIPTION...
```

Le début de la page d'aide nous donne le nom de la commande sur laquelle on désire de l'aide ainsi qu'un numéro entre parenthèses. Ce numéro correspond à une classification dont voici une explication:

- (1) commandes utilisateur
- (2) appels système
- (3) sous routines
- (4) drivers

- (5) formats de fichiers
- (6) jeux
- (7) divers
- (8) administration système
- (9) nouveau.

b) Les HowTo et mini HowTo concernant un sujet particulier. Ces aides sont généralement disponibles au format HTML, texte et pdf. Sous Linux, nous allons retrouver ces aides dans le répertoire suivant: `/usr/share/doc/howto/fr/`.

c) Les FAQ sur internet généralement sur le site correspondant à votre distribution.

La commande ls.

Cette commande permet d'obtenir la liste des fichiers et sous répertoires présents dans le répertoire courant ou dans un répertoire donné.

Parmi l'ensemble des options, nous nous limiterons aux suivantes: `-l -a -i -R`

L'option -a affiche les fichiers et les répertoires même ceux dont le nom commence par un `.` Sans cette option, ces derniers n'apparaissent pas à l'écran.

L'option -l permet d'afficher une série d'informations liées au fichier dont nous avons déjà parlé précédemment.

Exemple:

```
drwxr-xr-x  2 root  root    48 Sep 21 19:37 cdrom
```

Nous retrouverons dans l'ordre de gauche à droite:

- Type du lien ('-' pour un fichier 'd' pour un répertoire)
- Autorisations d'accès (aspect analysé lors d'un prochain chapitre)
- compteur de lien (voir paragraphe 1.2 traitant des liens)
- propriétaire du fichier
- groupe auquel le fichier appartient
- taille du fichier
- date de dernière modification du fichier.
- nom du fichier ou répertoire.

L'option -i permet d'afficher le numéro de l'Inode associé à cette entrée.

```
100320 drwxr-xr-x  2 root  root    48 Sep 21 19:11 floppy
```

L'option -R permet d'assurer une récursivité de la commande vis à vis du contenu des sous répertoires qui s'y trouvent.

```
cd /home/student
ls -lR
```

```
./windows:
```

```
total 1
```

```
drwxr-xr-x  2 root  root    48 Sep 21 19:37 .
```

```
drwxr-xr-x  8 student users 696 Sep 21 19:37 ..
```

L'ensemble des sous répertoires sont passés en revue, leur nom est affiché et leur contenu également.

La commande cp (copy).

Cette commande permet d'effectuer une copie d'un fichier. Elle se présente sous deux formes:

- 1) cp [OPTION]... SOURCE DEST
- 2) cp [OPTION]... SOURCE... DIRECTORY

Exemple:

Nous créons dans un premier temps quelques fichiers dans notre répertoire de travail courant en utilisant la commande touch.

```
(0) cd /home/student/  
(1) touch fichier1.txt  
(2) touch fichier2.txt  
(3) mkdir copie  
(4) ls  
(5) cp fichier1.txt copie/test1.txt  
(6) ls copie  
(7) cp copie/test1.txt .  
(8) ls tes*.*  
(9) cp fichier1.txt fichier2.txt copie
```

(0) cette commande permet de rendre le répertoire /home/student/ comme répertoire de travail courant.

(1) et (2) la commande touch permet de mettre à jour l'heure et la date de création d'un fichier. Si le fichier n'existe pas, le fichier est créé. Dans notre cas, l'objectif est de créer deux fichiers nommés fichier1.txt et fichier2.txt dans le répertoire courant.

(3) mkdir est l'abréviation de make directory. L'objectif est de créer un sous répertoire dans le repertoire courant auquel nous allons donner le nom copie.

(5) Cette commande utilise la première syntaxe de la commande ls à savoir de renseigner le nom du fichier source unique et le nom du fichier de destination unique. *cp fichier1.txt copie/test1.txt* permet de copier le fichier fichier1.txt du répertoire courant vers le fichier test1.txt situé dans le sous répertoire copie.

(6) la commande ls copie permet de lister le contenu du sous répertoire copie et provoque l'affichage suivant:

```
ns:/home/student # ls copie
```

```
. .. test1.txt
```

(7) la commande `cp copie/term [OPTION]... FILE...st1.txt .` permet d'effectuer la copie du fichier `test1.txt` se situant dans le sous répertoire `copie` vers le répertoire courant identifié par le point.

(8) cette commande permet de lister le contenu dans le répertoire courant des fichiers dont le nom commence par `tes` et portant toute extension. Elle provoque l'affichage suivant:

```
ns:/home/student # ls tes*.*
test1.txt
ns:/home/student #
```

(9) cette commande correspond à la deuxième syntaxe et permet de copier plusieurs fichiers d'une source vers une destination de répertoire unique en ayant les mêmes noms de fichiers.

Remarques :

- les noms des fichiers dans la commande `cp` peuvent contenir des expressions régulières dont la syntaxe ressemble à ce que l'on connaît sous `msdos`. Le caractère `'*'` peut être remplacé par un ou plusieurs caractères tandis que le caractère `'?'` peut être remplacé par un caractère unique.

Exemple:

```
cd /home/student
mkdir conf
cp /etc/n*.conf conf
ls
```

Cet exemple provoque la copie de tous les fichiers situés dans le répertoire `/etc` dont le nom commence par le caractère `n` et dont l'extension est `conf`. Le répertoire de destination complet est `/home/student/conf`. L'exécution de la commande `ls` donne l'affichage suivant:

```
ns:/home/student # mkdir conf
ns:/home/student # cp /etc/n*.conf conf
ns:/home/student # ls conf
. .. named.conf nscd.conf nsswitch.conf ntp.conf
ns:/home/student #
```

- La commande `cp` présente certaines options intéressantes
 - s qui permet de créer un lien symbolique au lieu d'une copie
 - r qui permet d'effectuer une copie récursive (prendre les fichiers sélectionnés dans les sous répertoires éventuels. (équivalent à la commande `xcopy` sous `msdos`))

La commande rm (remove)

Cette commande permet de supprimer un ou plusieurs fichiers dans un répertoire. La syntaxe est la suivante:

```
rm [OPTION]... FILE...
```

Exemple: cd /home/student/wilfart
 touch fichier.txt
 (1) ls -l
 (2) rm fichier.txt
 (3) ls -l

(1) l'exécution de la commande ls -l provoque l'affichage suivant:

```
ns:/home/student/wilfart # touch fichier.txt
ns:/home/student/wilfart # ls -l
total 2
drwxr-xr-x  2 root  root    80 Sep 30 21:23 .
drwxr-xr-x 34 student users 1656 Sep 30 21:20 ..
-rw-r--r--  1 root  root     0 Sep 30 21:23 fichier.txt
ns:/home/student/wilfart #
```

(2) la commande rm fichier.txt permet de supprimer le fichier fichier.txt du répertoire courant.

(3) l'exécution de la commande ls -l provoque l'affichage suivant:

```
ns:/home/student/wilfart # rm fichier.txt
ns:/home/student/wilfart # ls -l
total 2
drwxr-xr-x  2 root  root    48 Sep 30 21:26 .
drwxr-xr-x 34 student users 1656 Sep 30 21:20 ..
ns:/home/student/wilfart #
```

Remarques:

- les noms des fichiers dans la commande cp peuvent contenir des expressions régulières dont la syntaxe ressemble à ce que l'on connaît sous msdos. Le caractère '*' peut être remplacé par un ou plusieurs caractères tandis que le caractère '?' peut être remplacé par un caractère unique.

Exemple: cd /home/student/wilfart
 mkdir conf
 cp /etc/*.conf conf
 ls -l conf
 rm conf/n*.conf

```
ls -l conf
```

Cet exemple de commande rm supprime l'ensemble des fichiers du sous répertoire conf commençant par la lettre n et ayant l'extension conf.

- La commande rm présente certaines options intéressantes telles que:
 - r permet d'effectuer une suppression récursive
 - i permet l'affichage d'un message de demande de confirmation avant la suppression

La commande mv (move - rename)

Cette commande permet de renommer un fichier ou de déplacer un fichier vers un répertoire donné. La syntaxe de cette commande se présente sous les formes suivantes:

```
mv [OPTION]... SOURCE DEST
mv [OPTION]... SOURCE... DIRECTORY
```

Exemple:

```
cd /home/student/wilfart
mkdir conf
touch fichier.txt
ls -l
mv fichier.txt renomme.txt
ls -l
mv renomme.txt conf/test.txt
ls -lR
```

L'exécution de ls -lR provoque l'affichage suivant:

```
ns:/home/student/wilfart # ls -lR
.:
total 2
drwxr-xr-x  3 root  root    72 Sep 30 21:51 .
drwxr-xr-x 34 student users 1656 Sep 30 21:20 ..
drwxr-xr-x  2 root  root    80 Sep 30 21:51 conf

./conf:
total 0
drwxr-xr-x  2 root  root    80 Sep 30 21:51 .
drwxr-xr-x  3 root  root    72 Sep 30 21:51 ..
-rw-r--r--  1 root  root     0 Sep 30 21:50 renomme.txt
ns:/home/student/wilfart #
```

Le fichier a bien été dans une première étape renommé et ensuite déplacé vers le sous répertoire conf.

3.Manipulation des répertoires.

La commande mkdir (make directory).

Cette commande permet de créer un sous répertoire dans le répertoire courant. La syntaxe est la suivante: `mkdir [OPTION] DIRECTORY...`

Exemple: `cd /home/student/wilfart`
`mkdir SousRep`
`ls -l`

L'exécution des commandes de cet exemple provoque l'affichage suivant:

```
ns:/home/student/wilfart # mkdir SouRep
ns:/home/student/wilfart # ls -l
total 2
drwxr-xr-x  3 root  root    72 Sep 30 22:04 .
drwxr-xr-x 34 student users 1656 Sep 30 21:20 ..
drwxr-xr-x  2 root  root    48 Sep 30 22:04 SouRep
ns:/home/student/wilfart #
```

La commande rmdir (remove directory).

Cette commande permet de supprimer un répertoire pour autant que ce répertoire soit vide. La syntaxe est la suivante: `rmdir [OPTION]... DIRECTORY...`

Exemple: `cd /home/student/wilfart`
`mkdir SousRep`
`touch SousRep/fichier.txt`
`(1) rmdir SousRep`
`rm SousRep/*.*`
`rmdir Sousrep`

(1) L'exécution de cette commande nous renvoie une erreur du fait que le répertoire que l'on tente de supprimer n'est pas vide. Nous obtenons alors l'affichage suivant:

```
ns:/home/student/wilfart # rmdir SousRep
rmdir: `SousRep': Directory not empty
```

Pour pouvoir supprimer le répertoire, il faut l'en vider de ses fichiers avec la commande `rm SousRep/*.*` et ensuite réutiliser la commande `rmdir SousRep`.

Remarque: Il existe un moyen de supprimer un répertoire et son contenu en utilisant la commande suivante: `rm -R SousRep`

Les commandes cd (change working directory) et pwd

La commande cd permet de changer le répertoire de travail courant tandis que la commande pwd permet d'afficher le répertoire de travail courant (print working directory).

Exemple:

```
cd /home/student
pwd
cd ../wilfart
pwd
cd ../favier
```

L'exécution des commandes provoque l'affichage suivant:

```
ns:/home/student # cd /home/student
ns:/home/student # pwd
/home/student
ns:/home/student # cd ../wilfart
ns:/home/student/wilfart # pwd
/home/student/wilfart
ns:/home/student/wilfart # cd ../favier
ns:/home/student/favier # pwd
/home/student/favier
ns:/home/student/favier #
```

Recherche de fichiers.

Linux offre des commandes permettant de retrouver rapidement un fichier. La commande find associée éventuellement à un filtre de type grep permet cette recherche. Nous reprenons quelques exemples simples de recherche, la commande man find vous offrant une gamme d'options assez impressionnante.

- Recherche du fichier named.conf à partir du sommet de l'arborescence /
find / -name named.conf (sensible aux majuscules/minuscules)
- Même recherche mais non sensible aux majuscules/minuscules
find / -iname named.conf

Soit l'exercice suivant:

Nous devons trouver dans votre répertoire de travail l'ensemble des liens symboliques qui ne correspondent plus à aucun fichier ou répertoire physique existant et de les supprimer

Pour pouvoir tester notre commande, nous allons exécuter les commandes suivantes:

```
pwd
```

```
cd /home/student/  
mkdir fichiers  
touch ./fichiers/fichier1.txt  
touch ./fichiers/fichier2.txt  
touch fichier.txt  
ln -s ./fichiers/fichier1.txt ./fichier1.txt  
ln -s ./fichiers/fichier2.txt ./fichier2.txt  
ls -lR
```

Cette dernière commande permet de créer les différents liens symboliques. Nous allons maintenant effacer un des fichiers vers lequel un des liens pointe en exécutant la commande suivante:

```
rm ./fichiers/fichiers1.txt  
ls
```

Nous nous limiterons dans un premier temps à uniquement afficher ces liens de façon complète avec un format du type de la commande ls pour vérifier si nos options sont les bonnes

La commande fait apparaître sous une couleur différente le lien symbolique ne pointant plus sur un nom de fichier valide. Pour faire apparaître uniquement la liste de ces liens erronés, nous utiliserons la commande suivante:

```
find ./ -xtype l -ls
```

Reprenons chaque partie de la commande pour en comprendre le fonctionnement:

```
find ./ -xtype l
```

Nous retrouvons par la commande man find l'information suivante pour l'option -xtype:

`-xtype c`
Le même que pour l'option -type à moins que le fichier soit un lien symbolique. Pour les liens symboliques : si l'option -follow a été donnée, 'true' si le fichier est un lien vers un fichier de type c; si -follow a été donné, true si c est 'l'. En d'autres mots, pour les liens symboliques, -xtype permet de tester le type de fichier tandis que l'option -type de permet pas ce test .

Nous aurons donc à l'écran la liste des liens symboliques ne pointant plus sur un fichier existant. Si nous souhaitons supprimer les liens trouvés, nous pouvons utiliser l'option -exec qui permet d'exécuter une commande sur base de chaque fichier trouvé.

```
find ./ -xtype l -exec rm {} \;
```

Nous retrouvons par la commande man find l'information suivante pour l'option -xtype:

`-exec command ;`

true si l'état 0 est retourné. Tous les arguments qui suivent sont considérés comme faisant partie de la commande jusqu'à ce que le caractère ';' soit

rencontré. La chaîne {} est remplacée par le nom de fichier courant. Ces deux instructions doivent être suivies par un \.

Soit l'exercice suivant: Veuillez afficher à l'écran la liste des fichiers présents dans votre répertoire de travail dont la taille dépasse 400Ko.

```
find ./ -type f -size +400k
```

Notons l'utilisation du plus précédent le chiffre 400 pour indiquer que ce doit être supérieur à 400.

Soit l'exercice suivant: Veuillez afficher à l'écran la liste des fichiers qui appartiennent à un utilisateur donné.

```
find ./ -type f -user student
```

Vous pouvez également remplacer le nom de l'utilisateur par son numéro d'identification pouvant être obtenu par la commande id en utilisant la syntaxe suivante:

id -u student. Si cette commande renvoie 501, il suffira alors d'exécuter la commande find sous la forme find ./ -type f -user 501.

Soit l'exercice suivant: la commande grep permet d'effectuer la recherche d'un paterne dans un fichier. (voir la commande grep)

En utilisant la commande find couplée à la commande grep, veuillez donner la syntaxe permettant de rechercher l'ensemble des fichiers à partir de la racine ayant l'extension conf et contenant le mot blancquart.

```
find / -type f -name "*.conf" -exec grep -H blancquart {} \;
```

Les filtres:

Pour aborder l'utilisation des filtres, il est important de comprendre l'utilisation des 'pipes' (tubes). Un tube n'est rien d'autre qu'un mécanisme par lequel deux applications peuvent communiquer ensemble. Prenons la commande suivante:

```
ls -lR /* | more
```

La sortie standard de la commande ls est envoyée vers l'entrée standard de la commande more en passant par une mémoire tampon. Dans le shell (interpréteur de commande), deux processus fils sont en fait créés: l'un pour la commande `ls -lR /*` et l'autre pour la commande `more`, ces deux processus communiquant par un tube. Sans interrompre l'exécution de la deuxième commande, nous exécutons sous un autre shell la commande `ps -Al` et nous obtenons l'affichage suivant:

```

(1)  004 S   0 2298 2296 0 75 0 - 706 wait4 pts/1  00:00:00 bash
      006 R   0 2391 1465 1 76 0 - 22928 -   ?      00:00:04 kdeinit
      004 S   0 2393 2391 0 75 0 - 703 wait4 pts/2  00:00:00 bash
(2)  000 S   0 2421 2298 0 76 0 - 669 pipe_w pts/1  00:00:00 ls
(3)  000 S   0 2422 2298 0 76 0 - 440 schedu pts/1  00:00:00 more
      040 R   0 2424 2393 0 76 0 - 737 -   pts/2  00:00:00 ps

```

(1) Nous retrouvons le processus bash qui est l'interpréteur de commande. Il est exécuté en ayant comme identificateur de processus 2298.

(2) Le processus ls est exécuté sous l'interpréteur de commande bash. Nous retrouvons comme identificateur de processus fils 2421 lié au processus parent 2298.

(3) Le processus more est exécuté sous l'interpréteur de commande bash. Nous retrouvons comme identificateur de processus fils 2422 lié au processus parent 2298.

Soit l'exercice suivant: la commande wc (word count) permet de compter le nombre de byte, de mots ou de lignes dans un fichier. (voir la commande man wc)

En utilisant la commande find, veuillez compter le nombre de fichiers que l'on retrouve dans votre répertoire de travail.

```
find ./ -type f | wc -l
```

4.La structure du système de fichier.

Linux comprend dans son système de fichier certains répertoires standard. Les plus importants sont repris ci après:

/home C'est le répertoire contenant tous les répertoires de travail des différents utilisateurs. Votre répertoire de travail contient toutes vos données, fichiers de configuration personnels et tout ce que vous pourriez avoir accès. La majorité de ces données sont lisibles par d'autres utilisateurs mais ne peuvent être changés que par le propriétaire. C'est naturellement un paramètre par défaut qui peut être changé à tout moment.

/root C'est le répertoire de travail de l'administrateur root

/usr/ Le répertoire usr contient une majeure partie du système d'exploitation. On y retrouve /usr/bin, /usr/sbin, /usr/lib servant à stocker les exécutables et les bibliothèques dynamiques.

/bin et /sbin contiennent des programmes liés au système d'exploitation qui sont nécessaires dans la phase d'amorçage. Les autres programmes se retrouvent dans les répertoires /usr/bin et /usr/sbin

/etc ce répertoire contient une bonne partie des fichiers de configuration système.

Parmi ces fichiers nous retrouveront:

- les fichiers de configuration réseau tels que host.conf, networks, hosts, resolv.conf, xinetd.conf
- les fichiers concernant les utilisateurs tels que passwd, group
- le fichier inittab qui est l'équivalent du fichier autoexec.bat sous dos.

/boot (parfois /stand ou /unix) C'est le répertoire contenant tout le kernel du système d'exploitation

/lib ou /usr/lib Ces répertoires contiennent les bibliothèques dynamiques. Ce sont les équivalents des DLL sous Windows. Il s'agit de bibliothèques standard utilisables par plusieurs applications mais qui ne sont chargées qu'une seule fois en mémoire permettant un gain d'espace mémoire et de performance.

/var/ Contient des fichiers dont le contenu peut varier assez régulièrement. Nous retrouverons notamment :

- /var/log permettant l'accès aux journaux d'événements
- /var/spool correspond à la queue d'impression
- /var/named contient les fichiers de configuration du serveur DNS.

5. Les droits d'accès aux fichiers et répertoires.

présentation.

Nous avons vu qu'il était possible par la commande ls d'afficher les attributs complets d'un fichier ou d'un répertoire. Exemple:

```
-rw-r--r--  1 root  root    5330 Sep 23 10:58 xinetd.conf
drwxr-xr-x  2 root  root     48 Oct  8 2002 xinetd.d
lrwxrwxrwx  1 root  root      6 Sep 18 16:54 rc.d -> init.d
```

Le premier bloc comprend 10 caractères:

- Le premier caractère correspond au type de l'entrée: - pour un fichier, d pour un répertoire et l pour un lien symbolique
- Les trois caractères suivants pouvant être dans l'ordre rwx correspondent aux droits qu'a le propriétaire sur le fichier
- Les trois caractères suivants pouvant être dans l'ordre rwx correspondent aux droits qu'a le groupe propriétaire sur le fichier
- Les trois caractères finaux pouvant être dans l'ordre rwx correspondent aux droits pour 'le reste du monde'.

Les attributs de permissions standard sont:

La permission de lecture (r). Si un fichier est lisible, on pourra alors le visualiser et aussi la copier, l'imprimer... Si c'est un répertoire, on pourra lister son contenu.

La permission d'écriture (w). Si on peut écrire dans un fichier, alors on pourra l'éditer pour modifier son contenu, le supprimer, le renommer. Si c'est un répertoire, alors on pourra y placer des fichiers ou des sous répertoires.

La permission d'exécution (x). Si un fichier a cet attribut, alors on pourra l'exécuter. Si c'est un répertoire, cela signifie qu'il est traversable c-à-d que même si on ne peut pas lister son contenu, on peut accéder directement à des fichiers ou à des sous répertoires qui s'y trouvent.

Les attributs système sont:

On peut retrouver les attributs (s), (g), (t) et (l) à la place de l'attribut x pour certains types de fichiers.

- **setuid** (s). Si un exécutable possède cet attribut, celui-ci s'exécutera sous l'identité du propriétaire du fichier. Cette possibilité est intéressante lorsque un programme doit accéder à des données que le seul droit de l'utilisateur ne permet pas d'accéder. En s'exécutant sous les droits du propriétaire de l'exécutable, celui-ci pourra accéder sans problème à ces données sans devoir pour cela modifier les droits de l'utilisateur. Ceci n'est valable que pour les programmes binaires et pas les scripts du shell abordés ultérieurement.
- **setgid** (s). Si un exécutable possède cet attribut pour les droits de groupe, celui-ci s'exécutera sous l'identité du groupe propriétaire du fichier. Ceci n'est valable que pour les programmes binaires et pas les scripts du shell abordés ultérieurement.
- **sticky** (t). Cet attribut peut s'appliquer à des exécutables et à des répertoires. Pour un exécutable, cet attribut permet de forcer son maintien en mémoire même lorsque le programme s'arrête. Cette présence en mémoire vive permet une exécution ultérieure plus rapide puisque aucun chargement ne doit plus être effectué. Si un répertoire possède cet attribut, alors la suppression ou le renommage des fichiers présents dans ce répertoire sont soumis aux limitations suivantes: seuls
 - le propriétaire,
 - le propriétaire du répertoire,
 - l'administrateurpeuvent supprimer ou renommer le fichier.

Exemple d'utilisation du sticky bit: la modification de son mot de passe par la commande `passwd`. Si vous avez un doute sur l'emplacement exacte de cet exécutable, il suffit de lancer la commande `find / -name passwd`. Cette commande donne l'affichage suivant:

```
ns:~ # find / -name passwd
/etc/pam.d/passwd
/etc/passwd
/usr/bin/passwd
```

L'exécutable `passwd` se trouve dans le répertoire `/usr/bin` et possède les attributs suivants obtenus par la commande `ls /usr/bin/passwd -l`.

```
ns:/usr/bin # ls passwd -l
-rwsr-xr-x 1 root shadow 68680 Sep 24 2002 passwd
```

Le fichier qui contient les mots de passe de façon crypté se trouve dans le répertoire /

etc/. Voici les attributs de ce fichier.

```
ns:/usr/bin # ls /etc/passwd -l
```

```
-rw-r--r-- 1 root root 950 Sep 23 12:21 /etc/passwd
```

Le fichier est la propriété de root et n'est accessible par les 'autres' qu'en lecture.

Comment faire pour qu'un utilisateur qui ne serait pas le root pourrait faire pour modifier son mot de passe. Lorsque nous regardons les attributs de l'exécutable passwd, nous retrouvons l'attribut (s) qui signifie que ce programme s'exécutera sous les droits de root et de ce fait, le contenu du fichier passwd correspondant pourra être modifié.

Il existe deux façons de pouvoir modifier les droits associés à un fichier ou un répertoire: la méthode octale et la méthode symbolique par la commande chmod.

La commande chmod.

Le format du mode symbolique est le suivant [ugoa...][[+|=][rwxXstugo...][...][,....]. Des opérations symboliques multiples peuvent être données en les séparant par une virgule.

La combinaison des lettres 'ugoa' permet de contrôler quels sont les utilisateurs pour lesquels s'appliquent les droits d'accès au fichier:

pour le propriétaire (u pour user), pour le groupe propriétaire (g pour group), les autres (o pour others) et tout le monde (a pour all). Si aucune de ces lettres n'est utilisée, c'est comme si un 'a' était utilisé (sans la modification de umask).

L'opérateur '+' permet d'ajouter la permission sélectionnée tandis que '-' permettra d'enlever cette permission. L'opérateur '=' permet d'attribuer uniquement ces permissions.

Les permissions sont les suivantes: lecture (r), écriture (w), exécution (ou accès pour les répertoires) (x), setuid (s), sticky (t).

La méthode octale repose sur l'attribution d'un nombre distinct à chaque attribut. Pour les attributs normaux nous obtenons:

lecture (r) correspond à 4,

écriture (w) correspond à 2,

exécution (x) correspond à 1,

pour le droit accordé au propriétaire, ce nombre est à multiplier par 100,

pour le droit accordé au groupe propriétaire, ce nombre est à multiplier par 10,

pour le droit accordé aux autres, ce nombre est à multiplier par 1.

Pour les attributs système, nous obtenons:

setuid (s) correspond à 4

setgid (s) correspond à 2

sticky (t) correspond à 1

Ce nombre est à multiplier par 1000.

Exemple:

Si l'on considère accorder les droits suivant rw-rw-rw- pour un fichier appelé data.txt. Si nous utilisons la commande symbolique, nous devons utiliser la commande:

```
touch data.txt
ls -l data.txt (1)
chmod a=rw data.txt
ls -l (2)
```

(1) cette commande donne l'affichage suivant:

```
drwxr-xr-x  4 root  root    120 Oct 13 23:41 .
drwxr-xr-x 37 student users  1760 Oct 7 11:50 ..
-rw-r--r--  1 root  root     0 Oct 13 23:41 data.txt
```

(2) cette commande donne l'affichage suivant:

```
drwxr-xr-x  4 root  root    120 Oct 13 23:41 .
drwxr-xr-x 37 student users  1760 Oct 7 11:50 ..
-rw-rw-rw-  1 root  root     0 Oct 13 23:41 data.txt
```

Si nous désirons rechanger les droits en utilisant la méthode octale, voici la valeur que nous devons introduire dans la commande:

```
rwx rwx rwx
421 421 421
```

La valeur correspond donc à $4*100+4*10+4=444$. Nous pouvons alors introduire la commande suivante: `chmod 444 data.txt`.

La commande `umask` permet de modifier les droits par défaut affectés aux nouveaux fichiers ou répertoires. L'aspect un peu rébarbatif de cette commande réside dans le fait que Linux accorde les droits maximum aux fichiers et répertoires et que la valeur associée à `umask` permet d'enlever certains de ces droits. La commande `umask` sans paramètre donne la valeur courante du masque.

Exemple:

```
cd /home/student/wilfart
umask (1)
mkdir temp
cd temp
ls -l (2)
```

La question est donc de savoir quels sont les droits effectifs qu'aura notre répertoire `temp` nouvellement créé. En partant du principe que Linux, hors masque, devrait accorder les droits maximaux, nous devrions obtenir `777` ou `rw-rw-rwx`.

(1) La commande `umask` nous retourne `022` c-à-d l'équivalent de `---w--w-`. Si nous retirons ces attributs, nous devrions obtenir le droit effectif `rw-r-xr-x`.

(2) La commande `ls -l` donne l'affichage suivant:

```
drwxr-xr-x  2 root  root    48 Oct 14 00:00 .
drwxr-xr-x  5 root  root   144 Oct 14 00:00 ..
```

Les commandes `chown` et `chgrp`.

chown permet de modifier le propriétaire et/ou le groupe propriétaire d'un ou plusieurs fichiers. Le futur propriétaire est défini par son numéro d'utilisateur ou son nom définis dans le fichier /etc/passwd. (1)

La modification du propriétaire ne peut se faire que par le propriétaire actuel ou par l'utilisateur numéro 0 (root).

Si les attributs setuid et setgid sont en place sur le fichier, la commande chown les annule.

chgrp permet de modifier le groupe propriétaire d'un fichier. Le groupe peut être défini par son numéro ou par son nom définis dans le fichier /etc/group (2).

La modification du groupe propriétaire ne peut se faire que par le propriétaire ou l'utilisateur ayant le numéro 0 (root).

Si les attributs setuid et setgid sont en place sur le fichier, la commande chgrp les annule.

(1) Si nous éditons le contenu du fichier /etc/passwd, nous obtenons un affichage comprenant des lignes ayant une structure analogue aux lignes suivantes:

```
willy:x:500:100:Emmanuel Wilfart:/home/willy:/bin/bash
student:x:501:0:LaboInfo:/home/student:/bin/bash
```

Nom de login:Mot de passe:Numéro utilisateur:Numéro de groupe: Nom d'utilisateur:répertoire de base:interpréteur de commande. Considérons la première ligne de l'exemple:

Nom de login: willy.

Mot de passe: il est crypté dans le fichier /etc/shadow et est remplacé dans le fichier /etc/passwd par le caractère 'x'.

*Numéro de l'utilisateur:*500

Numéro du groupe: 100

Nom d'utilisateur: Emmanuel Wilfart

Répertoire racine: /home/willy

Interpréteur de commande:/bin/bash

(2)Si nous éditons le contenu du fichier /etc/group, nous obtenons un affichage comprenant des lignes ayant une structure analogue aux lignes suivantes:

```
man:x:62:student
```

```
users:x:100:
```

```
video:x:33:willy,student
```

Nom de groupe: mot de passe:Numéro de groupe:liste d'utilisateurs

Considérons la première ligne de l'exemple:

Nom du groupe: man

Mot de passe: nous pouvons protéger l'accès à un groupe par un mot de passe. Le changement de groupe peut s'effectuer par la commande *sg*. Les mots de passe sont cryptés et placés dans le fichier *gshadow*. De ce fait, le mot de passe dans le fichier *group* est remplacé par le caractère 'x'.

Numéro de groupe: 62

Liste des utilisateurs: student

Exemple:

Soit l'utilisateur willy dont le répertoire de travail courant est /home/willy. Sous son identité, nous allons exécuter les commandes suivantes:

```
touch data.txt
ls -l (1)
su student
rm data.txt (2)
ls -l > data.txt (3)
```

(1) l'exécution de cette commande provoque l'affichage suivant:

```
-rw-r--r--  1 willy  users      0 Oct 14 21:39 data.txt
```

Le propriétaire du fichier correspond bien à son créateur qui est willy, ce dernier appartenant au groupe users. Nous pouvons remarquer que le propriétaire a tous les droits tandis que les membres du groupe users et le reste du monde n'ont que la propriété de lecture seule.

(2) Le fichier n'est normalement accessible qu'en lecture par l'utilisateur student et de ce fait, nous obtenons le message d'avertissement suivant:

```
student@ns:/home/willy> rm data.txt
rm: remove write-protected regular empty file `data.txt'? y
rm: cannot remove `data.txt': Permission denied
```

(3) Le fichier n'est accessible qu'en lecture seule et de ce fait, toute tentative d'écriture sera infructueuse.

```
student@ns:/home/willy> ls -l > data.txt
bash: data.txt: Permission denied
```

Pour éviter ce problème, nous pouvons donner l'appartenance du fichier à l'utilisateur student ou accorder plus de droits pour le reste du monde. Prenons la première option et exécutons les commandes suivantes:

```
su root (1)
chown student data.txt
ls -l (2)
su student
ls -l > data.txt
ls -l (3)
```

(1) bien que autorisé précédemment si nous étions propriétaire du fichier, le changement de propriété ne peut plus se faire que sous root.

(2) Cette commande provoque l'affichage suivant:

```
-rw-r--r--  1 student users      0 Oct 14 21:39 data.txt
```

Nous remarquons que le propriétaire du fichier est maintenant student et non plus willy. Ce dernier a donc vu ses droits diminuer.

(3) La commande précédente a été acceptée puisque student est maintenant propriétaire du fichier et l'affichage indique l'augmentation de taille du fichier.

```
-rw-r--r--  1 student users    207 Oct 14 22:03 data.txt
```

6. Les processus.

Les propriétés d'un processus.

Lorsque l'on désire exécuter un processus, le système d'exploitation le charge en mémoire et va maintenir au cours de son exécution un certain nombre d'informations. Nous pouvons retrouver certaines de ces infos en utilisant la commande d'affichage des processus en cours: ps.

ps -ef donne en partie l'affichage suivant:

```
UID    PID  PPID  C  STIME TTY   TIME   CMD
root    1      0    0  21:40 ?    00:00:05  init
```

Nous retrouvons les colonnes suivantes:

PID: le numéro du processus spécifique (Process Identification)
PPID: le numéro du processus parent spécifique (Parent Processus Identification)
UID: le numéro de l'utilisateur (User Identification)
C: le temps CPU (en pourcentage) utilisé par le processus
STIME: l'heure ou le processus a été démarré (ou le jour)
TTY: le terminal de contrôle du processus ou ? si le processus n'est pas lié à un terminal
TIME: le temps CPU réellement utilisé, en seconde, depuis le démarrage du processus.
CMD: le programme exécuté.

Nous pouvons également utiliser la commande ps avec d'autres arguments provoquant un affichage au format BSD. La commande ps -aux provoque l'affichage suivant:

```
USER    PID %CPU %MEM  VSZ  RSS TTY   STAT START  TIME COMMAND
root     1   0.1  0.0   448  236 ?     S    21:40   0:05  init
```

Nous retrouverons les colonnes suivantes:

PID: le numéro du processus spécifique (Process Identification)
%CPU: le temps CPU utilisé par le processus
%MEM: le pourcentage de mémoire utilisé par la processus
SIZE: la taille de la mémoire virtuelle allouée
RSS: la taille de la mémoire RAM utilisée par le processus
TTY: le terminal de contrôle du processus ou ? si le processus n'est pas lié à un terminal
STAT: le status du processus en cours.
COMMAND: le programme exécuté.

Il est important dans la gestion des processus de se fixer clairement les idées sur ce que l'on entend par processus parent et état d'un processus

□ Processus parent:

Chaque processus peut démarrer lui même un autre processus. Pour s'en convaincre, il suffit de penser à l'interpréteur de commande (ex bash)qui sera à même d'exécuter toute commande. Cette donnée est importante du fait que dans certains cas, il est intéressant si l'on arrête un processus d'arrêter automatiquement tous les processus enfants. Il y a une exception à cette règle puisqu'il faut bien un premier processus qui est à l'origine des autres: c'est un pseudo processus qui porte très souvent le numéro 0 et il crée lui même un processus portant le numéro 1 qui porte le nom init. Il fournit la base permettant d'afficher à l'écran le login et d'utiliser le shell

□ Etat d'un processus:

Un processus peut prendre 3 états: l'état Runnable correspondant à un programme en attente d'exécution, l'état Running si il est en cours d'exécution et l'état sleeping si le processus est en attente. Du fait que nous ayons dans nos configurations de laboratoire un seul processeur, celui n'est jamais capable que d'exécuter une seule tâche à la fois.

Les traitements en tâche de fond.

Lorsque nous décidons d'exécuter une commande de longue haleine en mode terminal, il est souvent intéressant de basculer cette tâche en tâche de fond afin de pouvoir exécuter d'autres commandes. Inversément, si certaines commandes attendent à un moment donné l'intervention de l'utilisateur, il faudra refaire passer la tâche en avant plan. Ce basculement arrière plan/avant plan s'effectue par les commandes fg (foreground) et bg (background).

-1- Pour passer une tâche de l'avant plan vers l'arrière plan:

Prenons l'exemple suivant:

```
ls -lR / > liste 2>/dev/null
Appuyer simultanément sur CTRL Z (1)
jobs (2)
bg %1 (3)
jobs (4)
```

(1) cette séquence de touche permet de suspendre la commande en cours. Nous pouvons retrouver l'ensemble des séquences de touche acceptées en tapant la commande stty -a.

(2) cette commande permet d'afficher le liste des jobs suspendus. Nous retrouverons l'affichage suivant:

```
[1]+ Stopped ls $LS_OPTIONS -lR / >liste 2>/dev/null
```

La commande est associée à un numéro de tâche. Comme commande est la seule, elle possède le numéro 1.

Si nous désirons faire passer la tâche 1 en tâche de fond c-à-d s'exécutant en tâche de

fond au lieu d'être suspendue, nous devons utiliser la commande `bg %` suivie du numéro de tâche renvoyé par la commande `jobs`.

(3) La commande provoque l'affichage suivant:

```
[1]+  Running                  ls $LS_OPTIONS -IR / >liste 2>/dev/null &
```

Lorsque une tâche se termine alors qu'elle se trouve en arrière plan, un message est envoyé vers l'utilisateur pour l'en avertir. Voici le type de message que nous recevons:

```
[1]+  Exit 1                    ls $LS_OPTIONS -IR / >liste 2>/dev/null
```

Nous pouvons éviter cette série d'opération si nous souhaitons envoyer immédiatement une commande en tâche de fond. Une des particularité du shell `bash` est d'accepter la syntaxe suivante:

```
ls -IR / > liste 2>/dev/null &
```

Le caractère `&` permet de basculer immédiatement la commande en tâche de fond.

-2- Pour passer une tâche de l'arrière plan vers l'avant plan:

Prenons l'exemple suivant:

```
ls -IR / > liste 2>/dev/null & (1)
```

```
jobs (2)
```

```
fg %1 (3)
```

(1) Lors de cette commande, nous obtenons l'affichage suivant:

```
ns:~ # ls -IR / >liste 2>/dev/null &
```

```
[1] 2966
```

Nous retrouvons le numéro du processus 2966 ainsi que le numéro de la tâche [1].

(2) Cette commande provoque l'affichage suivant:

```
ns:~ # jobs
```

```
[1]+  Running                  ls $LS_OPTIONS -IR / >liste 2>/dev/null &
```

```
(3) ns:~ # fg %1
```

```
ls $LS_OPTIONS -IR / >liste 2>/dev/null
```

Etant donné que les processus sont en tâche de fond, il est parfois intéressant de pouvoir être informé lorsque toutes ces tâches sont terminées si nous désirions pouvoir ne fusse que fermer notre terminal.

Il existe pour cela la commande `wait`. Cette commande peut accepter un seul argument si l'on désire n'attendre la fin que d'un seul processus. La syntaxe peut être alors `wait %1`

-3- A savoir pour les tâches de fond:

a) Le processus ne doit pas prendre de saisie au clavier

b) Le processus ne doit pas retourner de résultats à l'écran ou au terminal sinon les sorties de ce processus risquent d'entrer en conflit avec celles du shell ou d'une autre commande...

c) Les commandes ont un parent: le shell car c'est lui qui les a lancées. Si nous quittons le shell, nous mettons fin également aux tâches de fond.

Envoi de messages aux processus.

Exécuter une commande ou un programme est très facile sous le shell. L'arrêter est aussi simple si l'on connaît la façon d'envoyer des messages vers les processus que l'on désire arrêter.

Si un processus est en foreground, il sera très simple de l'arrêter en appuyant simultanément sur les touches CTRL C. Pour les processus en tâches de fond, nous pouvons utiliser la commande kill dont voici la syntaxe: `kill [-s signal | -p] [-a] [--] pid ...`

signal correspond au signal que l'on désire envoyer

pid correspond au numéro d'identification du processus auquel ce signal est destiné. Pour se faire une idée des signaux que l'on peut envoyer, il suffit d'exécuter la commande `kill -l`.

Voici parmi une liste de plus de 60 signaux les principaux à retenir:

SIGHUP
SIGKILL
SIGSEGV
SIGTERM
SIGPWR
SIGSTOP
SIGCONT
SIGXCPU

SIGHUP Ce signal est envoyé par le processus parent à tous ses enfants lorsqu'il termine son activité. Ce signal est aussi envoyé à un processus pour signifier qu'il doit relire son (ses) fichier(s) de configuration.

SIGINIT C'est le signal envoyé pour interrompre un processus. Il est notamment envoyé par la combinaison de touches CTRL+C

SIGKILL Ce signal ne peut être ignoré d'aucun processus. C'est la façon de tuer un processus qui ne répond plus à aucun autre signal.

SIGTERM Ce signal permet de demander qu'un processus se termine. C'est le signal par défaut envoyé par la commande `kill`.

SIGSTOP Processus stoppé.

SIGCONT Processus continué.

Pour bien comprendre l'utilité de ces messages, nous allons exécuter les commandes suivantes:

```
ls -lR / > liste 2>/dev/null & (1)
```

Cette ligne provoque l'affichage suivant:

```
ns:~ # ls -lR / >liste 2>/dev/null &
```

```
[1] 3095
```

Nous obtenons donc l'information que ce processus est exécuté en tâche de fond et qu'il obtient le numéro d'identification de processus 3095.

Grâce à ce numéro, nous pouvons exécuter les commandes suivantes:

```
kill -SIGSTOP 3095 (1)
```

```
jobs (2)
```

(1) permet de stopper le processus

(2) donne l'affichage suivant:

```
ns:~ # jobs
```

```
[1]+ Stopped ls $LS_OPTIONS -IR / >liste 2>/dev/null
```

Cette commande est intéressante lorsque un traitement n'est pas terminé mais que l'on désire récupérer la puissance de calcul du processus pour exécuter un autre processus urgent tout en se laissant la liberté de reprendre l'exécution du processus stoppé par le signal SIGCONT.

Si nous démarrons un processus à partir de l'interpréteur de commande, il est intéressant que le processus enfant puisse survivre à l'arrêt du processus parent. Pour ce, il suffit que le processus enfant ne réponde pas au signal SIGHUP envoyé par le parent lorsqu'il s'arrête. Cette opération peut être réalisée par la commande nohup sous la syntaxe suivante:

```
nohup ls -IR / > liste 2>/dev/null & (1)
```

Pour mettre en évidence cette possibilité, nous allons exécuter la commande à partir d'une fenêtre terminal et ensuite fermer la fenêtre. Sous une nouvelle fenêtre terminal nous allons exécuter la commande ps -aux

(1) provoque l'affichage:

```
ns:~ # nohup ls -IR / >liste 2>/dev/null &
```

```
[1] 3308
```

La fenêtre est fermée et réouverte et ensuite nous exécutons ps -aux. Nous obtenons alors la partie d'affichage suivant:

```
root 3308 14.5 0.5 2548 1296 ? DN 20:43 0:14 ls -IR /
```

Priorité des processus.

Chaque processus qui est exécuté se voit doté d'une priorité. Qu'un processus tourne en tâche de fond (asynchrone) ou au premier plan (synchrone), sa priorité initiale sera la même. Chaque utilisateur peut choisir de faire tourner une tâche sur une priorité moindre en ayant les conséquences suivantes:

la tâche prendra plus de temps pour terminer son activité,

les autres tâches pourront disposer de plus de ressources processeur pour leur exécution.

La commande nice permet de rabaisser la priorité d'un processus. Il s'agit en fait d'ajuster la priorité lors de l'exécution de la commande.

Exemple:

```
ls -IR / > liste 2>/dev/null &
```

```
ps -l (1)
```

(1) l'affichage de cette commande donne

```
004 R 0 3476 3343 81 84 0 - 647 - pts/3 00:00:02 ls
```

Nous retrouvons une priorité de 0 pour cette commande ls.

Exécutons maintenant les commandes suivantes:

```
nice -n 19 ls -lR / > liste 2>/dev/null &  
ps -l (2)
```

(2) l'affichage de cette commande donne

```
004 R 0 3480 3343 77 99 19 - 647 - pts/3 00:00:03 ls
```

La commande nice accepte des valeurs comprises entre 19 (plus basse priorité) et -20 (priorité plus élevée). Si l'on désire modifier la priorité d'un processus en cours d'exécution, il est possible d'utiliser la commande renice.

Exemple:

```
ls -lR / > liste 2>/dev/null &  
ps -l (1)  
renice +10 3509 (2)  
ps -l (3)
```

Nous obtenons les différents affichages suivants:

```
(1) 004 R 0 3535 3343 85 84 0 - 647 - pts/3 00:00:01 ls
```

```
(2) 3535: old priority 0, new priority 10
```

```
(3) 004 R 0 3535 3343 83 95 10 - 644 - pts/3 00:00:07 ls
```

Processus d'amorçage du système d'exploitation.

Reprenons les étapes importantes lors de l'amorçage de votre ordinateur. Lors du reset d'un micro processeur, celui ci se branche à une adresse correspondant à un emplacement du BIOS qui exécutera le POST (Power On Self Test). Une fois cette opération terminée, le BIOS recherche un périphérique amorçable. Supposons que cette recherche aboutisse sur votre disque dur, le BIOS y recherchera le MBR (Master Boot Record) pour y lire les informations concernant les différentes partitions présentes sur le disque ainsi que la partition qui est active c-à-d étant amorçable. Le BIOS se branche alors dans cette partition active sur le Boot Record pour y charger le gestionnaire d'amorçage de votre système d'exploitation (ce gestionnaire tient sur un seul secteur 512ko et comprend une gestion rudimentaire des systèmes de fichiers gérés par le système d'exploitation pour assurer le chargement du noyau présent sur votre disque.

Après le chargement du noyau, le processus init est exécuté. Celui ci va monter le système de fichier root sous / et accéder au fichier /etc/inittab pour déterminer dans quel niveau d'exécution le système Linux doit être démarré. Tous les scripts de

démarrage sont exécutés dans les différents niveaux et ceux ci se trouvent dans le répertoire /etc/init.d

Voici une partie du contenu du fichier inittab:

id:5:initdefault:

```
# First script to be executed, if not booting in emergency (-b) mode
si::bootwait:/etc/init.d/boot
```

```
# /etc/init.d/rc takes care of runlevel handling
#
# runlevel 0 is System halt (Do not use this for initdefault!)
# runlevel 1 is Single user mode
# runlevel 2 is Local multiuser without remote network (e.g. NFS)
# runlevel 3 is Full multiuser with network
# runlevel 4 is Not used
# runlevel 5 is Full multiuser with network and xdm
# runlevel 6 is System reboot (Do not use this for initdefault!)
#
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
#l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

Nous retrouvons dans ce fichier des informations utiles sur les différents niveaux d'exécution:

```
runlevel 0 est l'arrêt système (Do not use this for initdefault!)
runlevel 1 iest le mode simple utilisateur
runlevel 2 est le mode local multiutilisateur sans connexion par le réseau
runlevel 3 est le mode complet multiutilisateur avec gestion réseau
runlevel 4 n'est pas utilisé
runlevel 5 est le mode complet multiutilisateur avec gestion réseau et mode graphique
runlevel 6 est le redémarrage (Do not use this for initdefault!)
```

id:5:initdefault: reprend le niveau sur lequel le système d'exploitation doit être démarré.

Comme renseigné précédemment, nous retrouvons des scripts associés à chaque niveau. Ces scripts figurent dans des répertoires déterminés:

```
niveau 0: /etc/init.d/rc0.d
niveau 1: /etc/init.d/rc1.d
niveau 2: /etc/init.d/rc2.d
niveau 3: /etc/init.d/rc3.d ...
```

Dans ces répertoires, nous retrouvons des scripts avec la particularité que les noms commencent par la letter K ou S suivi d'un nombre de deux chiffres et ensuite d'une

chaîne de caractères.

Lors de l'amorçage et de l'accès au niveau souhaité, ce sont les scripts commençant par la lettre S (start) qui seront exécutés suivant un ordre correspondant au nombre de deux chiffres: S10 est exécuté avant S15 ...

Lors de l'arrêt de votre ordinateur, ce sont les scripts commençant par la lettre K (kill) qui seront exécutés suivant le même ordre.

En fait chaque script S est appelé avec l'argument start et chaque script K est appelé avec l'argument stop.

Exemple:

Soit le service permettant la gestion des queues d'impression lpd. Nous retrouvons ce service dans le répertoire /etc/init.d.

```
ns:/etc/init.d # ls -l lpd
-rwxr--r-- 1 root  root    3860 Sep 18 2002 lpd
```

Si nous parcourons l'ensemble des répertoire rc#.d, nous retrouvons les scripts suivants:

```
ns:/etc/init.d/rc2.d # ls -l *lpd
lrwxrwxrwx 1 root  root    6 Nov  4 20:11 K08lpd -> ../lpd
lrwxrwxrwx 1 root  root    6 Nov  4 20:11 S13lpd -> ../lpd
```

Les deux scripts sont en fait des liens symboliques pointant sur le même fichier qui est lpd.

Les modules

Les modules sont un des éléments constitutifs importants de Linux. Initialement le noyau Linux était un noyau monolithique dans lequel on retrouvait tous les éléments permettant l'accès aux structures de données et aux routines. Si l'on désirait que le système d'exploitation prenne en charge la gestion du réseau, il fallait, lors de la compilation du noyau intégrer la gestion de la carte réseau. Ensuite l'introduction d'une carte d'extension supplémentaire dans le système nécessitait la création d'un nouveau noyau. Depuis, l'architecture de Linux a évolué et l'exploitation de modules extérieurs a été prévue à partir du noyau 2.0.x. Il s'agit souvent de pilotes qui sont chargés dynamiquement pendant l'exécution du noyau et qui peuvent être également être supprimés de la mémoire.

On comprend aisément les avantages et les inconvénients d'un tel mécanisme:

1- Un noyau modulaire se lance plus vite, la plupart des pilotes n'étant chargés que lorsque nous en avons besoin.

2- Nous pouvons envisager d'utiliser le même noyau sur des ordinateurs différents. Pensons à des disques amovibles permettant d'amorcer le même système d'exploitation sur des ordinateurs différents.

3- La modification de la version, le changement des paramètres d'un pilote ne nécessite pas de recompiler le noyau et de ce fait est réalisé plus rapidement.

4- Les inconvénients sont une consommation mémoire plus grande ainsi que la vitesse d'accès aux pilotes moins élevée dans les pilotes dynamiques.

L'exploitation de modules se réalise par les instructions **insmod** et **rmmod** si on désire effectuer cette opération manuellement. Pour que ces modules soient chargés automatiquement lors du démarrage de l'ordinateur, nous pouvons utiliser les fichiers **modules.conf** (ou **conf.modules**) situé dans le répertoire **/etc/**. Ce fichier enregistre la liste de tous les modules qui doivent être chargés ainsi que l'alias par lequel les périphériques associés devront être accédés.

Prenons par exemple la gestion de la carte réseau présente sur notre ordinateur: nous retrouvons dans le fichier **modules.conf** l'entrée suivante:

```
alias eth0 3c59x
```

Nous pouvons pour certains modules renseigner les paramètres matériels avec lesquels travailler notamment en ce qui concerne les numéros d'interruption, les ports d'entrée/sortie etc...:

```
# options 3c505      io=0x300 irq=10
```

L'ensemble des pilotes se trouve dans le répertoire **/lib/modules**. On y retrouve des sous répertoires associés aux différents noyaux compilés dans lesquels se trouve un sous répertoire **kernel** et ensuite dans la distribution **suse**:

```
.  block  evms  ieee1394  media  parport  sound
..  bluetooth  hotplug  input  message  pcmcia  telephony
acpi  cdrom  i2c  isdn  mtd  pnp  usb
atm  char  ide  md  net  scsi  video
```

Si nous nous plaçons dans le répertoire **net**, nous retrouvons le pilote **3c59x.o**.

Nous retrouvons un deuxième fichier important qui est le fichier **modules.dep**. Ce fichier contient les dépendances entre les modules.

Exemple d'entrée dans le fichier:

```
/lib/modules/2.4.19-4GB/kernel/drivers/net/3c509.o: /lib/modules/2.4.19-4GB/kernel/drivers/pnp/isa-pnp.o
```

3c509 est probablement une carte réseau de type **isa**. Ce pilote **3c509.o** nécessite la présence du pilote **isa-pnp.o** permettant la gestion du bus **isa**.

Vous pouvez à tout moment lister les pilotes chargés en mémoire en utilisant la commande **lsmod**. Cette commande provoque en partie l'affichage suivant:

Module	Size	Used by	Not tainted
<i>vfat</i>	9620	0 (autoclean)	
<i>fat</i>		30584 0 (autoclean)	[<i>vfat</i>]
<i>isa-pnp</i>	29664	0 (unused)	
<i>usbcore</i>	56768	1 [<i>usb-uhci</i>]	
<i>3c59x</i>	27088	1	

<i>ide-scsi</i>	7920	0
<i>reiserfs</i>	179536	1

Le mot autoclean de la deuxième colonne indique que ce module a été chargé par le démon **kerneld** et qu'il sera supprimé automatiquement de la mémoire s'il n'a pas été utilisé pendant un certain temps.

7.L'interpréteur de commande (shell).

Introduction

L'interpréteur de commande est l'interface entre l'utilisateur et le Linux. C'est lui qui permet de demander l'exécution d'un processus, d'une commande ou d'un script, ce dernier aspect étant possible via un langage de programmation propre à l'interpréteur utilisé.

Nous retrouvons plusieurs interpréteurs pouvant être regroupés en deux grandes familles: CShell (csh et tcsh) et sh (bash, ksh, zsh et ash). Nombre de ces interpréteurs héritent du premier interpréteur de commande sous Unix développé par Bourne Stephen dans les années 70. Nous retrouvons sous Linux le bash (Bourne Again Shell) qui est le plus connu et le plus répandu du fait peut être qu'il est l'interpréteur par défaut offert lors de l'installation du système d'exploitation.

Nous aborderons dans cette partie du cours le shell comme langage de programmation à part entière. Il met notamment en oeuvre:

- 1- des variables pour le stockage de données
- 2- des opérateurs
- 3- des structures pour contrôler le bon déroulement du programme
- 4- des fonctions

Les variables du shell.

Nous pouvons distinguer les variables créées par l'utilisateur et celles existant par défaut et attachées au système d'exploitation que l'on appelle variables d'environnement, certaines étant accessibles en lecture/écriture, d'autres en lecture seule.

Les variables d'environnement (encore appelées variables systèmes) importantes sont: PATH, SHELL, HOME, USER, DISPLAY. Pour en obtenir leur valeur, il suffit de les préfixer du signe \$. Si on désire afficher le contenu de la variable PATH, nous pouvons utiliser la commande echo sous la forme suivante:

```
ns:~ # echo $PATH
```

```
/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/java/jre/bin
```

PATH La liste des répertoires dans lesquels l'interpréteur de commande doit chercher

les fichiers exécutables.

HOME Contient le chemin absolu du répertoire de l'utilisateur.

SHELL Contient le nom de l'interpréteur de commande utilisé.

USER Contient le nom de l'utilisateur.

DISPLAY indique sur quel écran s'ouvre les fenêtres.

~ est le répertoire de travail courant (ne nécessite pas le signe \$ lors de la lecture).

Les variables utilisateur.

La création d'une variable s'effectue de la façon suivante:

```
ns:~ # a=/lib/modules
```

```
ns:~ # echo $a
```

```
/lib/modules
```

```
ns:~ # cd $a
```

```
ns:/lib/modules #
```

Nous avons dans ce code créé une variable a en même temps que son affectation.

Nous pourrions créer une variable vide en utilisant la syntaxe suivante:

```
ns:/lib/modules # b=
```

Cela permet à cette variable d'exister sans nécessairement disposer d'un contenu valide.

Une variable peut être supprimée en utilisant la commande **unset**. La commande

readonly est la seule protection contre la suppression accidentelle de variables.

Nous pouvons forcer une variable à être de type numérique en utilisant la commande **declare** sous la syntaxe suivante:

```
declare [-afFirtx] [-p] [name[=value]]
```

-i permet de renseigner que la variable contiendra uniquement un nombre entier. Les autres arguments sont documentés dans les man pages. Nous retrouverons dans un paragraphe ultérieur l'utilisation de l'argument f pour la déclaration d'une fonction du shell.

Le quoting, double quoting et backquoting.

Le quoting et le double quoting permettent d'entourer les arguments qui contiennent des caractères spéciaux de sorte qu'ils ne soient pas interprétés à l'exception des caractères \$ et \ pour le double quoting.

Exemple:

```
ns:~ # a='à tous'
```

```
ns:~ # b="bonjour $a"
```

```
ns:~ # c='bonjour $a'  
ns:~ # echo $b  
bonjour à tous  
ns:~ # echo $c  
bonjour $a
```

Le backquoting permet d'assigner à une variable le résultat d'une commande.

Exemple:

```
ns:~ # a=1  
ns:~ # b=2  
ns:~ # c=`expr $a + $b`  
ns:~ # echo $c  
3  
ns:~ #
```

Si nous déclarons les variables sous formes d'entiers grace à la commande declare, la syntaxe s'en trouve simplifiée:

```
ns:~ # a=20  
ns:~ # a=a+10  
ns:~ # echo $a  
30  
ns:~ #
```

`commande` est aussi équivalent à \$(commande).

Les différentes commandes.

La commande expr

La commande expr évalue une expression qui lui est passée sous forme de trois paramètres. Le résultat de cette expression est retournée par le biais du canal de sortie standard. La syntaxe est la suivante:

```
expr paramètre1 opérateur paramètre2
```

Nous pouvons utiliser les opérateurs suivants: + - * / et %, ces opérateurs ayant le même rôle que dans le langage C et uniquement valables pour des opérandes de type numérique.

Exemple:

```
ns:~ # a=10  
ns:~ # b=`expr $a + 15`  
ns:~ # echo $b
```

25

`ns:~ #`

La commande `expr` permet également de comparer les valeurs numériques de deux paramètres. Le résultat est 1 si la comparaison est vraie d'un point de vue logique et 0 dans le cas contraire. Les opérateurs suivants sont envisageables: `=`, `!=`, `<`, `>`, `<=`, `>=`.

Exemple:

```
ns:~ # a=10
```

```
ns:~ # b=20
```

```
ns:~ # e=$(expr $a \< $b)
```

```
ns:~ #
```

Nous retrouvons le caractère back slash précédent le signe `<`. Etant donné que le signe `<` est un caractère particulier interprété par le shell comme étant un opérateur de redirection comme nous le verrons dans un paragraphe ultérieur, nous devons lui indiquer de ne pas le considérer comme tel.

La commande `test`.

Cette commande sert à vérifier la réalisation d'une condition pouvant porter sur un fichier, sur une chaîne de caractères ou un test numérique. La syntaxe générale de cette commande est la suivante:

```
test EXPRESSION
```

```
test OPTION
```

Les options de test sur les fichiers sont les suivantes:

- e Le fichier existe-t-il?
- f S'agit il d'un fichier normal?
- d Le nom du fichier représente t il un répertoire?
- c Le nom du fichier correspond t il à un fichier de périphérique orienté caractère?
- b Le nom du fichier correspond t il à un fichier de périphérique orienté bloc?
- p Le fichier existe et est un tube nommé (Named PIPE ou fichier FIFO)
- r Le fichier existe et l'on dispose d'un droit en lecture sur celui ci
- w Le fichier existe et l'on dispose d'un droit en écriture sur celui ci
- x Le fichier existe et l'on dispose d'un droit en exécution sur celui ci
- s Le fichier existe et est non vide (taille supérieure à 0)
- u Le fichier existe et a son bit Set-UID positionné
- k Le fichier existe et son sticky bit est positionné
- L Le fichier existe et correspond à un lien symbolique

La commande renvoie une valeur 0 (vrai) ou 1 (faux) suivant l'évaluation de l'expression. Aucun résultat n'est envoyé à l'écran lorsque l'on utilise cette commande. La variable `$?` sera alors utilisée.

Exemple:

```
cd /home/student/wilfart
ls -l (1)
test -e data.txt (2)
echo $? (3)
test -e absent.txt
echo $?
```

Commentaires:

(1) cette commande nous permet de trouver dans notre répertoire un fichier existant avec lequel nous pourrions utiliser la commande test.

(2) L'affichage correspondant au contenu de la variable \$? est

```
ns:/home/student/wilfart # echo $?
```

```
0
```

Si nous choisissons un nom de fichier absent de notre répertoire courant lors de l'exécution de la commande test, nous aurons alors le contenu suivant pour la variable \$?

```
ns:/home/student/wilfart # echo $?
```

```
1
```

La commande test peut être remplacée par la syntaxe suivante: [exp]. Si nous reprenons notre exemple précédent, nous pouvons en modifier le contenu sous la forme suivante:

```
cd /home/student/wilfart
```

```
ls -l
[-e data.txt ]
echo $?
[-e absent.txt]
echo $?
```

Les options de test sur les chaînes de caractères sont les suivantes:

- n La longueur de la chaîne n'est pas nulle
- z La longueur de la chaîne est nulle

Exemple:

```
a="bonjour à tous"
[-z "$a" ] (1)
echo $? (2)
```

Commentaires:

(1) L'utilisation des doubles guillemets s'explique par le fait qu'en réalité, la commande est exécutée sous la forme [-z "bonjour à tous"]. Si vous omettez donc les doubles guillemets, les caractères blancs seront interprétés comme des séparateurs pour les arguments et la commande test renverra donc une erreur.

(2) Comme la chaîne de caractères n'a pas une longueur nulle, la commande doit renvoyer une valeur fausse correspondant à la valeur 1.

Nous pouvons également, pour la gestion des chaînes de caractères, utiliser les opérateurs = et != permettant d'effectuer une comparaison entre deux chaînes. La syntaxe est alors la suivante:

```
[ string1 = string2]  
[string1 != string2]
```

Pour les tests numériques, les options sont les suivantes:

- eq égal (equal)
- ne différent (not equal)
- lt strictement plus petit (little)
- gt strictement plus grand (greater)
- le plus petit ou égal (little or equal)
- ge plus grand ou égal (greater or equal)

Exemple:

```
a=10  
[ $a -eq 10 ]  
echo $?
```

Combinaisons de tests.

Les diverses formes de tests peuvent être combinés au moyen d'options supplémentaires.

- ! Négation logique du test
- a Combinaison de tests par un ET logique (and)
- o Combinaison de tests par un OU logique (or)

Les caractères spéciaux && et ||

Les caractères && et || permettent de chaîner des commandes dont l'exécution de la commande suivante dans le chaînage dépendra du résultat de l'exécution de la commande précédente.

- && la commande suivante est exécuté si la commande précédente a renvoyé une valeur vraie
- || la commande suivante est exécuté si la commande précédente a renvoyé une valeur fausse

Exemple:

```
cd /home/sudent/wilfart
mkdir temp
[ -d temp ] && echo "temp est un répertoire"
[ -f temp ] || echo "temp n'est pas un fichier"
```

Le regroupement des commandes.

On peut utiliser les parenthèses pour regrouper des commandes.

Par exemple: (ls -al ; ps -al ; who)

L'effet des parenthèses est de créer un sous shell qui exécute les commandes séparées de points virgules. Un résultat similaire est obtenu si on remplace les parenthèses par des accolades aux différences suivantes:

L'accolade de fermeture doit être placée en première position d'une ligne ou tout de suite après un point virgule. Pour le traitement des commandes, aucun sous shell ne sera créé mais le shell actif sera utilisé.

Pour bien mettre en évidence la création du sous shell, nous donnerons l'exemple suivant:

```
(ls / -R ; ps -el) > fichier.txt &
ps -el (1)
```

Commentaires:

Une partie de l'affichage provoqué par la commande ls sera le suivant:

```
004 S  0 2067 2065 0 75 0 - 704 wait4 pts/1 00:00:00 bash
002 S  0 2073 2067 0 76 0 - 703 wait4 pts/1 00:00:00 bash
004 R  0 2074 2073 43 76 0 - 638 - pts/1 00:00:01 ls
040 R  0 2075 2067 0 76 0 - 741 - pts/1 00:00:00 ps
```

La commande ls en cours d'exécution correspond au numéro de processus 2074 démarré du processus 2073 étant notre interpréteur de commande bash. Celui ci est un fait un sous shell du processus portant le numéro 2067.

Nous pouvons effectuer la même chose en remplaçant les parenthèses par des accolades en en tirant les conclusions qui s'imposent.

Les structures

Introduction.

Le langage de script s'inspire fortement du langage C et de ce fait, un grand nombre des structures que nous évoquerons s'en inspirent. Nous nous limiterons donc à en donner la syntaxe en supposant connu leur rôle.

La structure if

Syntaxe: if expr1 then body1 [elseif expr2 then body2 elseif] ...[else bodyN] fi

La structure case

Syntaxe: case string in patList body;; patList body;; ...esac

La boucle for

Syntaxe: for nom [in liste] do body done

La structure select

Syntaxe: select variable [in liste] do body done. Cette commande permet la création de menus simples. La liste est affichée à l'écran, chaque mot de la liste étant précédé d'un numéro d'ordre. Si l'utilisateur tape au clavier le numéro correspondant à un des choix dans le menu, le mot est alors placé dans la variable.

La boucle while et until

Syntaxe: while condition do body done

La commande break

Syntaxe: break [n]

Cette commande permet d'interrompre prématurément l'exécution d'une boucle (for, while, until). La valeur n indique le nombre de niveau qu'il faut remonter.

La commande continue

Syntaxe: continue [n]

Cette commande permet de passer à l'itération suivante dans une boucle (for, while, until). Si n est indiquée, il indique le nombre de passage dans la boucle.

Les commandes return et exit

Syntaxe: return [n]

La fonction (ou le script) shell en cours d'exécution se termine en renvoyant la valeur n. Si cette valeur est homise, la valeur retournée sera celle renvoyée par la dernière commande exécutée dans le corps de cette fonction (ou du script).

Le globbing.

Le globbing revient à pouvoir utiliser des patterns particuliers.

pattern signification

* remplace n'importe quelle chaîne de caractères

? remplace n'importe quel caractère
[...] est remplacé par un des caractères entre crochets
^ signifie caractère différent de

Exemples:

ls *.c permet de lister l'ensemble des fichiers ayant comme extension c.

Les redirections.

Avant qu'une commande ne soit exécutée, son entrée standard et sa sortie standard peuvent être redirigés en utilisant une notation spéciale interprétée par le shell. Les opérateurs de redirection peuvent précéder une commande, se trouver n'importe où dans la commande ou la suivre, les redirections étant exécutées dans l'ordre dans lesquelles elles apparaissent de gauche à droite.

< dirige l'entrée standard

> dirige la sortie standard

Nous pouvons placer un nombre entier avant ou après l'opérateur. ce nombre correspond à un flot standard d'entrée/sortie:

/dev/stdin File descriptor 0 is duplicated.

/dev/stdout File descriptor 1 is duplicated.

/dev/stderr File descriptor 2 is duplicated.

Exemple:

```
ls > dirlist 2>&1
```

La commande ls envoie sa sortie standard (stdout) vers le fichier dirlist tandis que l'affichage d'éventuelles erreurs est envoyé sur la sortie standard (sortie étant déjà redirigée au préalable vers un fichier) et de ce fait tout sera redirigé dans le fichier.

```
ls 2>&1 > dirlist
```

La commande ls envoie l'affichage de ses erreurs vers la sortie standard stdout tandis que le résultat de la commande sera envoyé vers le fichier dirlist.

Dans les deux cas, le contenu du fichier sera écrasé. Si l'on désire effectuer une opération d'ajout au contenu du fichier déjà existant, il suffit d'utiliser l'opérateur >>

Les scripts.

Comme nous avons pu le constater, l'utilisation du langage de l'interpréteur de commande va parfois nous amener à des syntaxes d'écriture assez complexe. Il est intéressant de pouvoir rassembler ces commandes au sein d'un fichier pour lequel on pourra demander à l'interpréteur de commande l'exécution des différentes commandes

qui s'y trouvent sans devoir donc les retaper à chaque utilisation. Ces scripts vont être très utiles pour les tâches administratives comme les exemples et exercices qui vont suivre vont nous le montrer.

Tout comme pour les fonctions, nous pouvons juger utile d'envoyer vers le script des arguments. Ceux ci peuvent être récupérés dans le script en utilisant la syntaxe suivante:

\$# permet de récupérer le nombre d'arguments
\$0 permet de récupérer le nom de la commande
\$1 permet de récupérer le premier argument
\$2 permet de récupérer le deuxième argument.

\$? permet de récupérer le retour de la dernière commande.

Exercice 1: utilisation de la commande TEST

a) Ecrivez un script qui dit si le paramètre passé est :
- soit un fichier
- soit un répertoire
- soit que rien n'existe en association avec ce nom
Ceci dans le répertoire courant.

b) Ecrivez un script comptant le nombre de fichiers présents dans le répertoire renseigné comme paramètre. Si le répertoire n'est pas renseigné, c'est le répertoire courant qui est choisi.

Solution exercice 1a:

```
#!/bin/bash
# script permettant de tester si le paramètre
# correspond à un fichier, un répertoire ou à rien

# la ligne de commande permet de tester si un argument a été passé.
if [ $# = 0 ] ; then
    echo "Vous n'avez pas entré de paramètres"
    echo "Le mode d'utilisation du script est $0 Nom"
    exit 0
fi
#permet de tester si $1 correspond à un fichier
if [ -f $1 ] ; then
    echo "$1 est un fichier"
    exit 0
fi
#permet de tester si $1 correspond à un répertoire
if [ -d $1 ] ; then
    echo "$1 est un répertoire"
```

```
    exit 0
fi
echo "$1 n'est ni un fichier ni un répertoire"
```

Solution exercice 1b:

```
#!/bin/bash
# script permettant compter le nombre de fichiers dans un répertoire
compte=0
# la ligne de commande permet de tester si un argument a été passé.
if [ $# = 0 ] ; then chemin="."
else chemin=$1
fi
#permet de tester si le répertoire existe
if ! [ -d $chemin ]; then
    echo "$chemin repertoire inconnu"
    exit 0
fi
#boucle fo permettant de parcourir toutes les entrées
# * permet de prendre en considération les fichiers n'ayant pas d'extension
for i in $chemin/*; do
    if [ -f $i ]; then
        compte=`expr $compte + 1`
    fi
done
echo "Nombre de fichiers: $compte"
```

Un script intègre souvent l'utilisation de commandes externes permettant notamment la manipulation des chaînes de caractères (grep, wc, tail, awk, gawk, head..). Nous allons donner quelques exemples d'utilisation de ces commande. Nous nous référerons aux pages de manuels de ces commandes pour des informations complémentaires.

Exercice 2: utilisation de la commande gawk.

- reprenre l'exercice précédent et compter le nombre de fichiers présents dans le répertoire courant en utilisant ce que renvoie la commande ls -al.
- En utilisant la commande df, veuillez lors de l'exécution du script n'afficher que les partitions des disques durs dont l'espace occupé dépasse 90% de la capacité totale.

Reprenons cette commande et analysons l'affichage qu'elle provoque:

```
drwxr-xrwx  5 root  root    256 Nov 18 20:21 .
drwxr-xr-x 39 student users  1888 Nov 18 20:01 ..
-rw-r--r--  1 root  root      0 Nov 12 10:54 data.txt
-rwxr--r--  1 root  root    632 Nov 18 20:17 ex1a
-rwxr--r--  1 root  root    572 Nov 18 20:33 ex1b
-rw-r--r--  1 root  root 1822996 Nov 12 11:15 fichier.txt
drwxr-xr-x  2 root  root     80 Oct  7 11:27 fichiers
-rw-r--r--  1 root  root 1813151 Nov 11 14:40 liste.txt
```

```
drwxr-xr-x  2 root  root    48 Nov 11 12:52 temp
drwxr-xr-x  2 root  root    48 Oct  1 12:02 test
```

Comme nous l'avons déjà vu dans la commande ls, le type de fichier est repris en début de chaque ligne: d pour répertoire et - lorsqu'il s'agit d'un fichier. Il serait donc intéressant de pouvoir reprendre chacune des lignes et en déterminer le premier caractère. La commande gawk permet ce travail.

Solution de l'exercice 2a:

La solution envisagée est de créer un canal (pipe) entre la commande ls -al et la commande gawk.

gawk scinde les données d'entrée en enregistrements et les enregistrements en champs. Un enregistrement est une chaîne d'entrée délimitée par défaut par un retour chariot tandis que le champ est une chaîne délimitée par défaut par un espace dans un enregistrement. On peut grâce à la variable intégrée FS définir le caractère utilisé comme séparateur de champ et la variable intégrée RS définir le caractère utilisé comme séparateur d'enregistrements.

Nous accepterons dans notre exemple les valeurs par défaut.

Les champs d'un enregistrement sont référencés par \$1, \$2,..., \$NF (dernier champ) Si nous désirons accéder aux attributs, nous utiliserons \$1.

Si nous utilisons le script simplifié suivant, nous obtiendrons l'affichage uniquement des attributs de tous les fichiers.

```
#!/bin/bash
```

```
ls -al | gawk '{ print $1 }'
echo "Fin du script"
```

Nous devons maintenant pouvoir définir un critère de sélection pour ne reprendre que les lignes commençant par le caractère '-'. Dans la commande gawk, le critère est inséré entre les chaînes BEGIN et END avec la syntaxe suivante:

```
gawk 'BEGIN[instructions] critères END{instructions}'
```

Un critère peut être une expression régulière, une expression ayant une valeur chaîne de caractères, une expression arithmétique ou une combinaison des expressions précédentes.

Si nous désirons que le critère soit la présence du caractère '-' en début du champ 1, nous devons utiliser la syntaxe suivante:

```
#!/bin/bash
ls -al | gawk 'BEGIN{} $1~/^-/{print $1} END{}'
echo "Fin du script"
```

Dans la deuxième ligne, nous allons décomposer la syntaxe correspondant au critère:

\$1: du fait que ce soit le contenu du premier champ que l'on doit considérer.

~: opérateur de test de correspondance avec une expression régulière et une commande. \$1 doit correspondre à /^[^]/ pour que la commande puisse être exécutée (dans notre cas, il s'agit de print)

/.../: permet de définir une expression régulière

^[^]: expression régulière signifiant que le champ doit commencer avec le caractère '-'

Nous allons maintenant retrouver à l'affichage uniquement le premier champ de chaque ligne pour autant qu'il commence par la lettre '-'. Il ne reste donc plus qu'à compter le nombre de ligne.

```
#!/bin/bash
```

```
# script permettant compter le nombre de fichiers dans un répertoire
```

```
ls -al | gawk 'BEGIN{compte=0} $1~/^[^]/{compte=compte+1} END{print compte}'
```

```
echo "Fin du script"
```

Solution de l'exercice 2b:

Lorsque nous exécutons la commande df, nous obtenons l'affichage suivant:

```
/dev/hda7      8257120 1734244 6522876 22% /
```

```
/dev/hda5      23270   5184   16885 24% /boot
```

```
shmfs         128064    0 128064 0% /dev/shm
```

Les champs qui nous intéressent sont le premier qui permettra de ne retenir que les disques durs et le cinquième qui comprend la taille déjà exprimée en %. Nous accepterons pour le premier champ, ceux correspondant à /dev/hdxx et /dev/sdxx (pour rappel les disques durs ide et les disques durs scsi).

Essayons dans un première étape n'afficher que les premiers champs n'étant associés qu'à des disques durs. Nous ferons également usage des expressions régulières. Nous retrouvons la syntaxe suivante:

```
#!/bin/bash
```

```
df | gawk 'BEGIN{} $1~/dev\[hs\]d\[a-z\][0-9]/{print $1} END{'
```

```
echo "Fin du script"
```

Nous retrouvons une syntaxe identique à l'exercice précédent. Analysons l'expression régulière avec laquelle nous travaillons dans cet exemple ci:

`Vdev\[hs\]d\[a-z\][0-9]` Les caractères '\', tout comme dans le langage C, permettent de faire suivre un caractère spécial que la commande gawk ne doit pas essayer

d'interpréter. Nous retrouverons donc comme début de chaîne **/dev/**. Ce début de

chaîne peut être suivi du caractère 'h' ou 's' suivi du caractère d, ensuite d'un caractère pouvant aller de 'a' jusque 'z' et finalement d'un chiffre pouvant aller de 0 jusque 9.

L'affichage de ce script est donc le suivant:

```
/dev/hda7
```

```
/dev/hda5
```

```
Fin du script
```

Nous pouvons maintenant essayer d'extraire le pourcentage que nous récupérons dans le cinquième champ. Pour pouvoir en tester le contenu, nous devons convertir la valeur en entier sans au préalable avoir oublié de supprimer le signe '%'.

La commande `gawk` intègre une collection très riche de fonctions intégrées permettant notamment la gestion des chaînes de caractères. Nous retiendrons deux de ces fonctions pouvant apporter un intérêt dans notre exercice: **substr** permettant d'extraire une chaîne de caractères d'une autre et la fonction **strtonum** permettant de convertir une chaîne de caractères en un entier et **length** renvoyant la longueur d'une chaîne de caractère. Pour la conversion de notre cinquième champ, nous devrions retrouver la syntaxe suivante:

```
strtonum(substr($5,0,length($5)-1))
```

La syntaxe complète de notre script sera alors:

```
#!/bin/bash
df | gawk 'BEGIN{} $1~/devV[hs]d[a-z][0-9]/{
q=strtonum(substr($5,0,length($5)-1));
if (q>23) print} END{}'
echo "Fin du script"
```

Nous pourrions imaginer envoyer un email pour signaler ce problème. Pour les besoins du test, nous avons diminué le pourcentage à 23.

Exercice 3:

Soit un script devant parcourir l'ensemble des fichiers du répertoire courant et convertir le nom des fichiers en minuscules. Nous pouvons utiliser la commande **mv** pour renommer le fichier tandis que nous devons utiliser une autre commande pour convertir le nom du fichier en minuscule. Nous utiliserons la commande `agwk` pour assurer la conversion de la chaîne de majuscules en minuscules.

Voici le code du script:

```
#!/bin/bash
for i in *; do
    if [ -f $i ]; then
        j=`echo $i | gawk '{print tolower($1)}'`
        `mv $i $j`
    fi
done
```

Commentons quelque peu la quatrième ligne de code:

```
j=`echo $i | gawk '{print tolower($1)}'`
```