INITIATION UNIX

David ROSSIGNOL (email : rossignol.d@linux-france.org)

28/04/99

Table des matières

Introduction

Copyright :

Pré-requis :

Connaissances de base du système d'exploitation DOS ; connaître l'organisation des données informatiques (notions de fichiers et répertoires).

Objectifs :

À la fin de ce cours, vous serez en mesure :

- ✓ d'utiliser les commandes UNIX les plus usuelles ;
- ✓ d'effectuer l'administration quotidienne de vos machines ;
- ✓ d'expliquer les dysfonctionnements de vos machines dans les Usenet.

Ne font pas partie de ces objectifs les points suivants :

- × la compréhension des mécanismes internes ;
- × les commandes d'administration impliquant ces mécanismes ;
- × les options dont l'utilité est limitée.

Conventions d'écriture :

Dans tout ce cours :
les paramètres précisés entre crochets `[]` sont optionnels ;
les `|` séparant les options, précisent qu'il faut choisir parmi l'une d'entre elles.

Partie cours 'théorique' (Présentation et annexes) :
Texte normal.
`Commandes telles qu'elles doivent être tapées au clavier.`
`Résultat obtenu à l'écran.`

Partie cours 'pratique' (Les commandes) :
Texte normal.
`Nom des commandes.`
`Résultat obtenu à l'écran et paramètres éventuels.`

Présentation du système UNIX.

Caractéristiques d'UNIX :

Un système d'exploitation est un ensemble de programmes qui coordonnent le fonctionnement des différents composants matériels et logiciels d'un système informatique.
UNIX est un système d'exploitation ayant les caractéristiques suivantes :

- **Multi-utilisateurs et Multitâches** : cela signifie que plusieurs utilisateurs peuvent accéder simultanément au système et exécuter un ou plusieurs programmes.
- **Temps partagé** : c'est-à-dire que les ressources du processeur et du système sont réparties entre les utilisateurs.
- **Système de fichiers hiérarchique** : plusieurs systèmes de fichiers peuvent être rattachés au système de fichiers principal ; chaque système de fichiers possède ses propres répertoires.
- **Entrées-Sorties intégrées au système de fichiers** : les périphériques sont représentés par des fichiers, ce qui rend le système indépendant du matériel et en assure la portabilité ; l'accès aux périphériques est donc identique à l'accès aux fichiers ordinaires.
- **Gestion de la mémoire virtuelle** : un mécanisme d'échange entre la RAM et le disque dur permet de pallier un manque de RAM et optimise le système.
- **Processus réentrants** : les processus exécutant le même programme utilisent une seule copie de celui-ci en RAM.
- **Interface utilisateur interactive (shell)** : elle est constituée d'un programme séparé du noyau permettant à l'utilisateur de choisir son environnement de travail. Elle intègre un langage de commandes très sophistiqué (scripts).

Le shell

Sous le nom shell (coquille), on désigne l'interface utilisateur qui, dans l'architecture d'UNIX, entoure le noyau (kernel, voir annexe 2).
Plus simplement, le shell est un programme qui permet à l'utilisateur de dialoguer avec le coeur du système (le noyau).
Il permet de réaliser plusieur tâches :

- il interprète les ordres tapés au clavier par l'utilisateur et permet ainsi l'exécution des programmes;
- il fournit à l'utilisateur un langage de programmation interprété;
- il permet de modifier l'environnement;
- il se charge de réaliser les redirections des entrées-sorties, les pipes et la substitution de type fichier (détaillés plus loin dans ce cours).

Les commandes

Le shell interprète la série de caractères tapés à chaque fois qu'un retour chariot (entrée) est reçu; en même temps, les caractères tapés sont affichés à l'écran.
Les premiers caractères tapés jusqu'au premier séparateur (espace ou tabulation) sont interprétés par le shell comme le nom de la commande à exécuter, les chaînes de caractères suivantes étant considérées comme les arguments.
La syntaxe d'une commande UNIX est donc :
commande ***[argument(s)]***
où :
commande est le nom d'une commande interne ou un programme exécutable,
et ***arguments*** sont des options reconnues par cette commande.

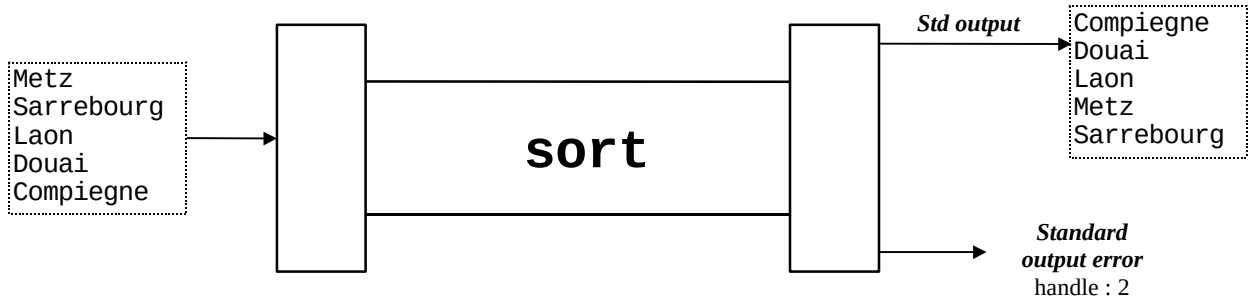
Les entrées-sorties et leur redirection.

Une commande lit normalement ses données d'entrée dans l' *entrée standard* et écrit sa sortie dans la *sortie standard* qui, par défaut, correspondent respectivement au clavier et à l'écran du terminal.
Les processus accèdent à ces périphériques comme s'ils accédaient à des fichiers normaux, par le biais de *handle* (descripteurs de fichiers). Voici donc un récapitulatif des périphériques, de leur association par défaut, ainsi que des descripteurs de fichiers :

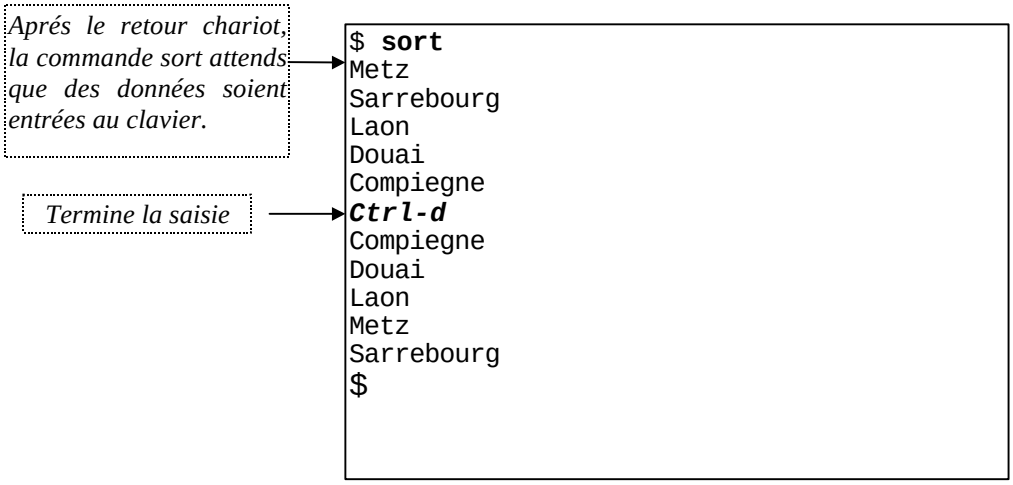
Périphérique	Association par défaut	Descripteur de fichier
Standard input buffer	Clavier	0
Standard output	Ecran	1
Standard output error	Ecran	2

Ils peuvent être matérialisés de la sorte :

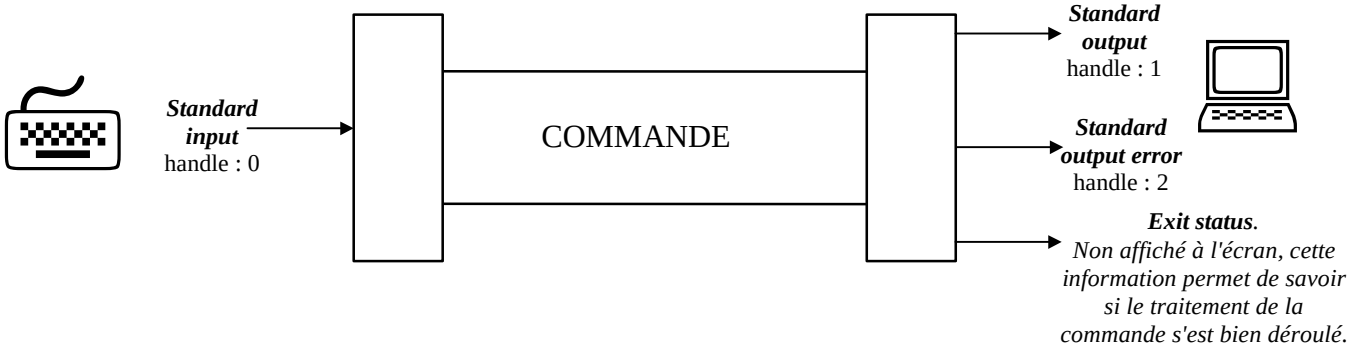
Voici un exemple avec la commande **`sort`** (commande qui trie les données d'entrée par ordre alphabétique) :



À l'écran, on observe :



S'il y avait eu une erreur à l'exécution de la commande, le message aurait été envoyé vers le *standard error output* qui par défaut, coïncide avec la *sortie standard* (l'écran).
En plus du résultat 'visible' des commandes, chacune d'entre elles renvoie au shell un code de sortie (*exit status*), qui est égal a zéro si la commande s'est terminée normalement, et prend une valeur différente si le processus a été interrompu.



Le mécanisme de redirection des entrées-sorties, géré par le shell, permet de changer l'association par défaut de l'*entrée standard* et des *standards outputs*;

dans ce cas, on utilise des fichiers normaux en entrée ou en sortie. Ces redirections sont effectuées grâce aux signes suivants :

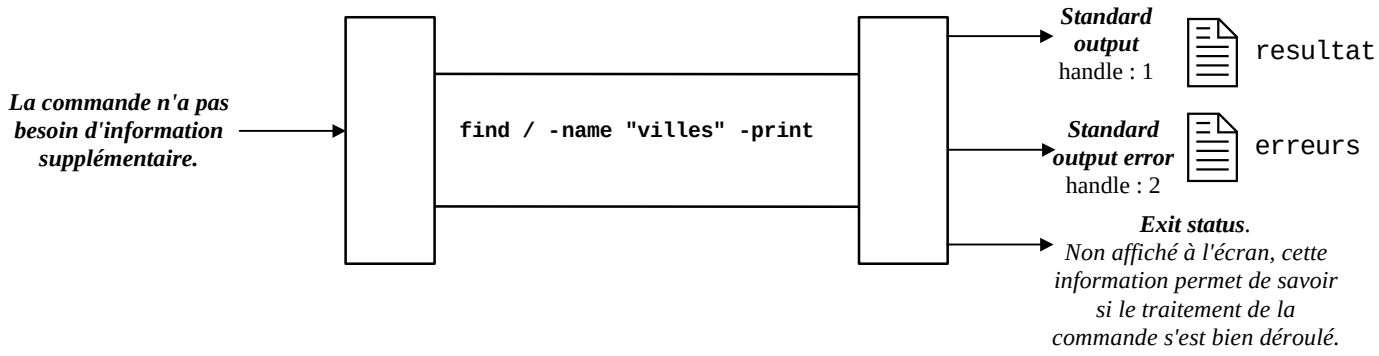
< nom_fic	prend comme entrée le fichier `nom_fic` ;
<< mot	prend comme entrée toutes les lignes tapées au clavier jusqu'à celle qui contient `mot.`
> fichier	envoie la sortie dans `fichier` (si `fichier` existe, il est écrasé).
>> fichier	rajoute la sortie à la fin du `fichier ;` on dit aussi concaténation de la sortie et de `fichier` (si `fichier` n'existe pas, il est créé).

Plus généralement, on peut rediriger n'importe quel fichier en entrée ou en sortie en utilisant le numéro descripteur de fichier (numéro unique attribué par le noyau chaque fois qu'un fichier est ouvert). Il est rappelé que le descripteur vaut 0 pour l'*entrée standard*, 1 pour la *sortie standard*, et 2 pour le *standard error output*. Pour rediriger un fichier de cette manière, il suffit de faire précéder le symbole de redirection par le numéro de descripteur (SANS espace de séparation).

Exemple :

```
$ find / -name "villes" -print 1> resultat 2> erreurs
$
```

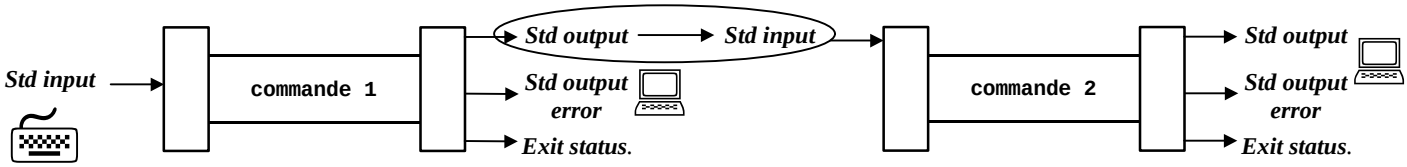
Dans ce cas, à l'exécution, on obtient le résultat suivant :



TP N°1

Les Pipes (tubes)

Le tube (ou pipe en anglais) est un mécanisme permettant la communication entre plusieurs commandes. Dans un pipe, la sortie de la première commande devient l'entrée de la seconde; le shell prend à sa charge la connexion de la sortie standard de la première commande sur l'entrée standard de la deuxième.



Exemple :

```
$ cat villes
Metz
Sarrebourg
Laon
Douai
Compiegne
```

Signe représentant le pipe

```
$ cat villes | sort
Compiegne
Douai
Laon
Metz
Sarrebourg
$
```

TP N°2

Les groupes de commandes

Il existe plusieurs méthodes pour enchaîner des commandes sur une même ligne :

- Exécution séquentielle :
  `cmd1; cmd2;...;cmdN`
- Exécution sous condition d'erreur :
  `cmd1 || cmd2 || ... || cmdN`
  si `cmd1` ne se termine pas correctement, alors `cmd2` est exécuté, et ainsi de suite.
- Exécution sous conditions de réussite :
  `cmd1 && cmd2 && ... && cmdN`
  si `cmd1` s'est bien déroulée, alors `cmd2` sera exécutée; et ainsi de suite

Les groupes de commandes peuvent remplacer les commandes simples dans les pipes; mais dans ce cas, il faudra entourer le groupe de commande avec des accolades **`{}`**; cela permet les combinaisons les plus variées; par exemple :

{ cmd1 && cmd2; cmd3 || cmd4; } | cmd5

ATTENTION: l'espace après **{** et le **;** avant **}** sont indispensables.

L'utilisation des parenthèses **()**est possible et a le même effet à la différence qu'elles invoquent un nouveau shell qui sera chargé de l'exécution des commandes.

TP N°3

Les substitutions de type fichiers

Il n'est pas rare que les arguments des commandes UNIX soient des noms de fichiers; à ce titre, les shell offrent la possibilité d'indiquer un groupe de fichiers dont les noms ont des similitudes. Cette possibilité est matérialisée par les métacaractères :

Métacaractère	Utilisation
?	indétermination d'un caractère
*	indétermination de **0** à **n** caractères
[xyz123...]	indétermination d'un caractère pris dans la liste
[b-t]	indétermination d'un caractère pris dans l'intervalle
[!xyz123...]	indétermination d'un caractère pris à l'extérieur de la liste
[!b-t]	indétermination d'un caractère pris à l'extérieur de l'intervalle

ATTENTION : différenciation des minuscules et MAJUSCULES.

TP N°3 bis

Les caractères spéciaux

Ce sont tous les caractères vus précédemment qui ont une signification particulière pour le shell. Il comprennent les métacaractères **?*[]**, les signes de redirection (**<** et **>**), ou encore les caractères **"${}'|`()** .

Un problème se pose lorsque l'on veut utiliser ces caractères sans qu'ils soient interprétés par le shell !!!

Dans ce cas, certains caractères nous permettent de signaler au shell cette intention.

Il s'agit :

- du caractère **** qui annule la fonction spéciale du caractère le suivant,
- des guillemets **"**, qui annulent tous les caractères sauf **$\`**,
- les simples quotes **'**, qui annulent l'ensemble des caractères spéciaux.

Ce dernier ayant une signification particulière pour le shell, si vous voulez l'utiliser, il faut le faire précéder d'un ****.

Exemple :

```
$ echo 10000>nombre
$
/* Cette commande va générer un fichier nombre contenant 10000 */
```

```
$ echo 10000\>nombre
10000>nombre
$

$ echo '10000\>nombre'
10000\>nombre
$

$ echo 'Aujourd'\''hui'
Aujourd'hui
$
```

Les variables d'environnement

Les variables sont des valeurs associées à des noms explicites; elles constituent l'environnement d'exécution des commandes.
On retrouve dans l'environnement standard du shell un certain nombre de variables dont il se sert à l'exécution. Les plus utilisées sont :

- **HOME** : variable initialisée au login et contenant le répertoire de connexion de l'utilisateur.
- **PATH** : variable contenant la liste des répertoires où effectuer une recherche de programmes entrés au clavier (à la différence avec le DOS, si vous ne précisez pas le répertoire courant (.) dans la variable, il n'y sera effectué aucune recherche).
- **PS1** : contient l'invite (équivalent à la variable PROMPT de DOS). Elle est généralement initialisée à '$'.
- **CDPATH** : précise les chemins de recherche de répertoire pour la commande cd (Change Directory).

Bien sûr, l'utilisateur peut définir ses propres variables, en respectant la syntaxe suivante :

```
$ nom_variable=valeur
    /* Sans espace avant ni après le signe égal */
```

Pour récupérer le contenu d'une variable, l'utilisateur utilise le caractère **$** suivi du nom de sa variable.
Exemple :

```
$ echo $LOGNAME
root
    /* Les noms de variables distinguent aussi les
    minuscules des MAJUSCULES */
```

Sous UNIX, on distingue deux types de variables, les variables locales, et les variables globales (ou exportées).
Une variable locale est spécifique au niveau du processus en cours et seul ce processus pourra l'exploiter, alors qu'une variable exportée sera disponible pour tous les processus fils créés.
ATTENTION : les valeurs modifiées par un processus fils n'affectera pas la valeur de la variable du père.
Par convention, on utilise les MAJUSCULES pour les variables globales et les minuscules pour les variables locales.

Procédure de connexion

Pour pouvoir utiliser la machine, il vous faut disposer d'un nom d'utilisateur et du mot de passe qui lui est associé. Ceci vient du fait que l'un des processus système lancé au démarrage de la machine, le `ttymon`, remplit cette fonction de contrôle. Ce processus suit l'algorithme suivant :

⇒ Demande de nom d'utilisateur : Invite `login :`
- ✓ *Après que vous ayez entré un nom d'utilisateur,*

⇒ Demande de mot de passe. Invite `password :`
- ✓ *Le mot de passe entré,*

⇒ Le processus vérifie la cohérence des informations avec ses fichiers systèmes (`/etc/passwd` et `/etc/shadow`) ; si les informations sont correctes, `ttymon` termine son travail en invoquant le shell qui vous est associé (`/bin/sh` ou `/bin/ksh` ou `/bin/bash`).

Votre shell charge aussitôt ses variables spécifiques (`LOGNAME`, `TTY`, `HOME`, ...). Puis il exécute les scripts de démarrage nommés `/etc/profile` et `$HOME/.profile` qui initialisent, entre autres, les variables spécifiques à l'utilisateur (certains shell exécutent d'autres fichiers `.bashrc` pour `/bin/bash`).

La gestion des fichiers et des systèmes de fichiers

Sous UNIX, toutes les données sont manipulées à l'image d'un fichier ordinaire. De ce fait, il existe plusieurs types de fichiers en fonction de leur utilisation.
Les différents types sont :

⇒ **les fichiers répertoires** dont le contenu fait l'association entre le numéro d'inode (numéro unique représentant un fichier sur le système de fichiers) et le nom du fichier ;

⇒ **les fichiers ordinaires** qui stockent les données et les programmes sans format particulier mais comme une suite d'octets ;

⇒ **les fichiers spéciaux de type blocs ou caractères** qui constituent une porte permettant de communiquer avec les périphériques de l'ordinateur (Exemple : le lecteur de disquettes = /dev/fd0) ;

⇒ **les pipes nommés et les liens symboliques** qui ne seront pas abordés dans ce cours.

UNIX en tant que système d'exploitation sécurisé gère des droits d'accès sur les fichiers. Il existe trois niveaux de sécurité qui correspondent respectivement aux droits de l'utilisateur, du groupe, et des autres utilisateurs.
À chacun de ces niveaux, il est possible de déterminer les droits suivants :

- **La lecture (R ou 4)** ;
- **L'écriture (W ou 2)** ;
- **L'exécution (X ou 1)**

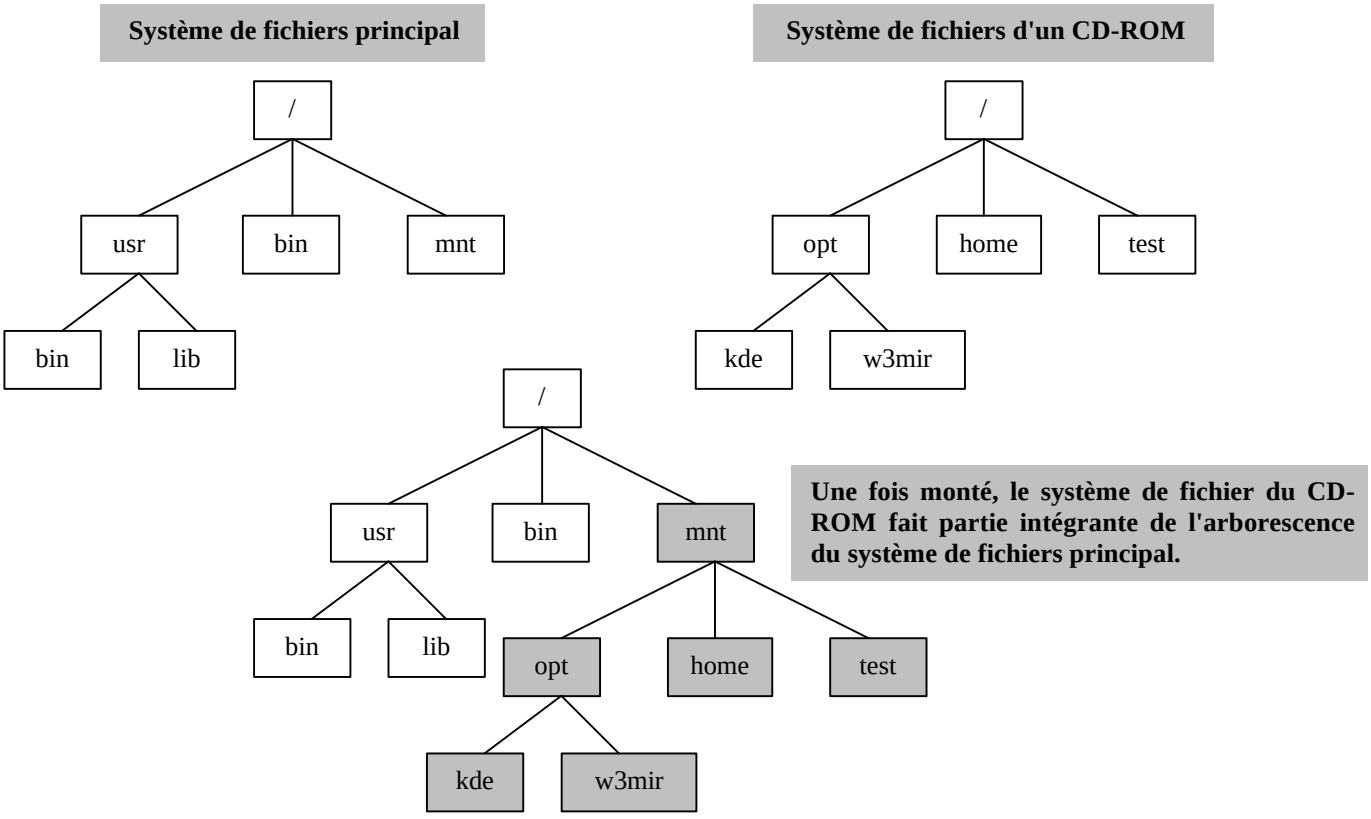
Leur interaction avec les différents fichiers est décrite ci dessous :

	Fichiers	Répertoires
Lecture	Autorise la lecture du fichier.	Permet de lister le contenu du répertoire.
Ecriture	Permet la modification du fichier.	Autorise la création et la suppression des fichiers du répertoire. **ATTENTION** : cette permission est valable quels que soient les droits des fichiers.
Exécution	Autorise l'exécution du fichier.	Permet de se positionner dans le répertoire.

Le système de fichiers est une structure logique qui permet de gérer l'espace disque. En effet, si l'on veut stocker des fichiers (ou répertoires) sur un disque, il faut préparer une ou plusieurs structures d'accueil. On retrouve ce procédé sous DOS lorsque vous créez X partitions sur un disque dur (avec `fdisk`) ; il faut les formater (avec `format`) avant de pouvoir les utiliser.
La différence avec le DOS, se situe dans la hiérarchisation de ces systèmes de fichiers :

Sous DOS, chaque système de fichiers porte un nom bien précis (Exemple : A:, C:, D:, …).

Sous UNIX, tous les systèmes de fichiers utilisés viennent se "rattacher" (on dit se monter) sur le système de fichiers principal (sur lequel on a booté), il y a hiérarchisation. Exemple :



Les commandes

man

Syntaxe :

man [section] commande
man -k mot_clé

Description :

man permet de rechercher une aide sur une commande ou un mot-clé. Il utilise la variable MANPATH pour effectuer la recherche des pages et la variable PAGER pour connaître le programme chargé de l'affichage.

Les pages man sont organisées en 8 sections standard comme suit :

1	= Commandes utilisateur
1M	= Commandes administrateur
2	= Appels systèmes C.
3	= Fonctions C.
4	= Format des fichiers système.
5	= Divers.
6	= Jeux.
7	= Fichiers spéciaux.
8	= Procédures de maintenance système.

Options courantes:

section	Numéro de la section qui contient la page.
-k	précise que la recherche s'effectue sur un mot-clé et non une commande.

Exemple :

```
$ man passwd

$ man 4 passwd

$ man -k passwd
passwd (1)          - define or change  login password  and password
                    attributes
default_passwd (4D)  - /etc/default/passwd file
passwd (4)          - password file
pwconv (1M)         - installs  and updates / etc/shadow  with information
                    from / etc/passwd
in.yppasswdd (1M)   - change  passwords for network user   accounts
yppasswd (1)        - change global user  password
yppasswdd (1M)      - change  passwords for network user   accounts
```

Remarques :

Les pages man sont toujours articulées autour des paragraphes suivants :

NOM
SYNTAXE
DESCRIPTION
OPTIONS
FICHIERS UTILISES
VOIR AUSSI

Sous Linux, la description des fichiers se trouve dans la section 5

Gestion et déplacement des arborescence

⇒ **`ls`** liste le contenu d'un répertoire.

⇒ **`cd`** changement de répertoire courant.

⇒ **`pwd`** affichage du répertoire courant.

⇒ **`mkdir`** création de répertoire.

⇒ **`rmdir`** destruction d'un répertoire.

ls

Syntaxe :

ls [options] [noms]

Description :

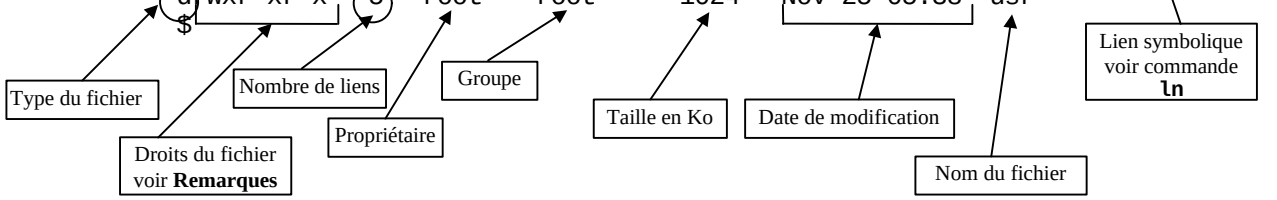
ls liste les répertoires et les fichiers précisés dans noms. Par défaut, la sortie est envoyée à l'écran par ordre alphabétique. Les options déterminent les informations à afficher et la présentation de l'affichage. Sans options, **ls** n'envoie que le nom des fichiers. Si noms n'est pas précisé, c'est le répertoire courant qui est listé.

Options courantes:

-R	Traitement récursif
-a	Tous les fichiers (y compris ceux qui commencent par un point)
-d	Affiche le nom des répertoires sans leur contenu
-l	Format long (avec beaucoup de détails)
nom	est le nom d'un fichier ou d'un répertoire (avec ou sans métacaractères)

Exemple :

```
$ ls -ld r* u*
lrwxrwxrwx   1   root    other        6   Nov 21  1997  rep1 -> /usr/rep1
lrwxrwxrwx   1   root    other        6   Nov 21  1997  rep2 -> /usr/rep2
lrwxrwxrwx   1   root    other        6   Nov 21  1997  rep3 -> /usr/rep3
-rwxr--r--   3   root    other  5423661   Apr  1  1997  unix
drwxr-xr-x   5   root    root      1024   Nov 25 05:38  usr
$
```



Remarques :

Au niveau du propriétaire, du groupe, et des autres, il est possible de déterminer un certain nombre de droits :

- **La lecture (R)** : pour un fichier, ce droit permet la lecture du fichier; alors que pour un répertoire, il autorise l'utilisateur à lister son contenu.
- **L'écriture (W)** : pour un fichier, il permet sa modification; alors que pour un répertoire, il permet la création et la suppression des fichiers du répertoire (**ATTENTION** : cette permission est valable quels que soient les droits des fichiers).
- **L'exécution (X)** : pour un fichier, il autorise son exécution; et pour un répertoire, il permet de se positionner dessous

cd

Syntaxe :

cd [répertoire]

Description :

La commande **cd** permet de changer le répertoire de travail. Si répertoire n'est pas précisé, alors le nouveau répertoire de travail sera le répertoire de connexion ($HOME).

Option :

répertoire représente le futur répertoire de travail.

Exemple :

```
$ cd /usr/lib/news/bin

$ cd $HOME/rep1
```

Remarque :

la commande **cd**, comme toutes les commandes utilisant des répertoires, permet de spécifier deux types de chemins :

- les chemins relatifs : ils sont relatifs au répertoire de travail, et utilisent notamment le répertoire '..' (répertoire père).

Exemple : **cd** ../rep1

- les chemins absolus : ils faut spécifier toute l'arborescence depuis la racine.
  Exemple : **cd** /home/rep1

TP N°4

pwd

Syntaxe :

pwd

Description :

La commande **pwd** permet d'afficher le répertoire de travail.

Option :

La commande **pwd** n'accepte pas d'option

Exemple :

```
$ pwd
/usr/lib/news/bin
$ cd ..;pwd
/usr/lib/news
$ cd bin
$ pwd
/usr/lib/news/bin
$ cd /bin
$ pwd
/bin
```

mkdir

Syntaxe :

mkdir [-p] nouveau_répertoire

Description :

La commande **mkdir** créé le répertoire spécifié sur la ligne de commande (nouveau_répertoire). Si l'un des répertoires intermédiaires n'existe pas, la commande retourne un code d'erreur (*exit status*) sans créer le répertoire (sauf si l'option -p est spécifiée).

Options courantes

-p	permet de créer tous les répertoires intermédiaires qui n'existeraient pas.
répertoire	représente le nom du répertoire à créer. C'est un argument obligatoire

Exemple :

```
$ mkdir /tmp/rep1
$ cd /tmp/rep1
$ mkdir rep11/rep111
mkdir: Cannot create directory " rep11/rep111": No   such file or directory
$ mkdir -p rep11/rep111
$
```

Remarque :

Pour pouvoir créer un répertoire, le répertoire d'origine doit avoir les droits en écriture positionnés.

TP N°5

rmdir

Syntaxe :

rmdir [-p] [-s] répertoire

Description :

La commande **rmdir** supprime le répertoire spécifié sur la ligne de commande (répertoire). Si il existe des fichiers ou des sous répertoires, la commande retournera un code d'erreur (*exit status*).

Options courantes

-p	permet de détruire tous les sous-répertoires vides.
-s	mode silencieux (aucun affichage).
répertoire	représente le nom du répertoire à détruire. C'est un argument obligatoire.

Exemple :

```
$ rmdir /tmp/rep1/rep11/rep111
$ cd /tmp
$ rmdir -p rep1/rep11
rmdir: rep1/rep11:  Whole path removed.
$ cd rep1
rep1: does not  exist
$
```

Remarque :

Pour pouvoir supprimer un répertoire, le répertoire père doit avoir les droits en écriture positionnés.

Voir aussi la commande **rm -r** pour supprimer des répertoires contenants des fichiers.

TP N°6

Gestion et manipulation de fichiers

- ⇒ **`cp`** copie de fichiers.
- ⇒ **`mv`** déplacement de fichiers.
- ⇒ **`rm`** destruction de fichiers.
- ⇒ **`cat`** visualisation et/ou concaténation de fichiers.
- ⇒ **`Pg | less`** visualisation d'un fichier texte page par page.
- ⇒ **`chmod`** change les droits d'un fichier/répertoire.
- ⇒ **`chown`** change le propriétaire d'un fichier/répertoire.
- ⇒ **`chgrp`** change le groupe propriétaire du fichier/répertoire.
- ⇒ **`find`** recherche de fichiers ou répertoires.
- ⇒ **`grep`** recherche d'une chaîne de caractères dans un fichier.
- ⇒ **`head/tail`** affiche le début/la fin d'un fichier.
- ⇒ **`ln`** crée un lien avec un fichier éxistant.
- ⇒ **`sort`** trie les lignes d'un fichier.
- ⇒ **`umask`** choix des permissions par défaut.
- ⇒ **`wc`** compte le nombre de mots/lignes/caractères d'un fichier.

cp

Syntaxe :

```
cp [-i] [-p] fichier1 fichier2
cp [-i] [-p] [-r] source1 [source2...] répertoire
```

Description :

La commande **cp** copie le contenu de `fichier1` dans `fichier2` ;
ou bien elle copie `source1` et `source2` (etc...) dans `répertoire` .

Options courantes

`-i`	mode interactif, demande la confirmation avant écrasement.
`-p`	conserve les dates du fichier source.
`-r`	copie récursive de répertoires.
`source`*X*	représente le nom des fichiers ou répertoires à copier.

Exemple :

```
$ cp /tmp/rep1/fic1 .
$ cp /tmp $HOME
$ cp -r rep1 rep2
$
```

Remarque :

Pour pouvoir copier un fichier/répertoire, vous devez avoir les droits suivants :

- droits de lecture du fichier à copier ;
- droits d'exécution sur le répertoire contenant le fichier à copier ;
- droits d'écriture sur le répertoire de destination.

mv

Syntaxe :

```
mv [-f] [-i] source1 [source2...] destination
```

Description :

La commande **mv** déplace les fichiers `fichier1, fichier2 etc...` dans `destination` .
Si `destination` est un fichier, alors **mv** a pour action de renommer `fichier1` en `destination` ; si `destination` est un répertoire, alors **mv** déplace `fichier1` dans ce répertoire.

Options courantes

`-i`	mode interactif, demande la confirmation avant écrasement.
`-f`	force la commande.
`sourceX`	représente le nom des fichiers ou répertoires a déplacer.
`destination`	représente le nom des fichiers ou répertoires de destination.

Exemple :

```
$ mv fic1 fic2
$
```

Remarque :

Pour pouvoir copier un fichier/répertoire, le répertoire cible doit avoir les droits en écriture positionnés, les droits en lecture sur le fichier source, et les droits d'accès dans le répertoire source.

rm

Syntaxe :

rm [-f] [-i] [-r] fichier1 [fichier2...]

Description :

La commande **rm** supprime les fichiers spécifiés sur la ligne de commande.

Si vous n'avez pas les droits d'écriture sur `fichier1`, alors **rm** vous demandera de confirmer votre action ; la réponse oui (y) détruira quand même le fichier (sous réserve d'avoir les droits d'écriture sur le répertoire).

Options courantes

`-i`	mode interactif, demande la confirmation avant chaque suppression.
`-f`	force la commande (aucune confirmation).
`-r`	récursif (détruit tous les sous répertoires. **ATTENTION**)

Exemple :

```
$ rm fic1
$
$ rm -i fic*
rm: File fic1.  Remove ( yes/no)? y
rm: File fic2.  Remove ( yes/no)? N
$
```

Remarque :

Pour pouvoir supprimer un fichier, le répertoire où se trouve le fichier doit avoir ses droits en écriture positionnés ; sauf dans le cas où le répertoire aurait les droit suivants :

```
drwxrwxrwt  1   user    user         6   Nov 21  1997  rep1
```

cat

Syntaxe :

cat [fichier...]

Description :

La commande **cat** visualise et/ou concatène les fichiers spécifiés sur la ligne de commande.
Par défaut, **cat** lit sur l'entrée standard et affiche le résultat sur la sortie standard.

Option :

La commande **cat** n'admet pas d'option.

Exemple :

```
$ cat villes
Sarrebourg
Douai
Couvron
Phalsbourg

$ cat villes pays
Sarrebourg
Douai
Couvron
Phalsbourg
France
Belgique
Italie
$
```

Remarque :

pg

Syntaxe :

pg [+numlig] [+/chaîne/] [fichier...]

Description :

La commande **pg** affiche à l'écran les fichiers spécifiés sur la ligne de commande.
Par défaut, **pg** lit sur l'entrée standard ce qui permet de l'associer à un pipe.

Options courantes

+numlig	spécifie le numéro de ligne où doit commencer l'affichage
+/chaîne/	commence l'affichage à la première ligne du fichier contenant chaine

Exemple :

```
$ pg villes
Sarrebourg
Douai
Couvron
Phalsbourg
(EOF):

$ pg +2 villes
Douai
Couvron
Phalsbourg
(EOF):

$ cat villes  pays|pg +/Couvron/ pays
Couvron
Phalsbourg
France
Belgique
Italie
(EOF):
```

Remarque :

Sous Linux, **pg** n'existe pas mais l'on pourra utiliser avantageusement **less** (voir la page man).

chmod

Syntaxe :

chmod [-R] mode nom [...]
chmod [-R] [ugoa]{+|-|=}[rwx] nom [...]

Description :

La commande **chmod** change les droit d'accès aux fichiers spécifiés sur la ligne de commande.
Deux manières d'attribuer les droits sont possibles. La première stipule les droits de manière numérique par un calcul des différentes valeurs associées aux droits; la seconde permet de spécifier ces mêmes droits de manière plus symbolique.

Options courantes

`-R`	récursif sur tous les fichiers et sous-répertoires contenus si `nom` est un répertoire
`mode`	c'est la combinaison des droits numériques (voir **Remarques**)
`ugoa`	caractère spécifiant le champ d'application des modifications : `u` représente l'utilisateur, `g` le groupe, `o` les autres, `a` regroupe tous ces derniers.
`+|-|=`	indique l'action à accomplir respectivement l'ajout du droit, son retrait, ou bien son affectation.
`rwx`	indique le droit proprement dit (`r` lecture, `w` écriture, `x` éxecution).

Exemple :

```
$ chmod 777 villes
$ chmod g-rw villes
$ chmod -x villes
```

rappel: r=4,w=2,x=1.
7=4+2+1 (r+w+x).
Les droits du fichier `villes` seront donc `rwxrwxrwx`

Le groupe propriétaire du fichier `villes` n'aura plus l'accès ni en lecture, ni en écriture.
`rwx--xrwx`

Remarque :

Plus personnes n'aura le droit d'exécution sur le fichier.
`rw----rw-`

propriétaire	**groupe**	**autres**
rwx	rwx	rwx
4	4	4
2	2	2
1	1	1

TP N°7

chown
chgrp

Syntaxe :

```
chown [-R] [-h] utilisateur nom [...]
chgrp [-R] [-h] groupe nom [...]
```

Description :

La commande **chown** change le propriétaire des fichiers spécifiés sur la ligne de commande.
La commande **chgrp** change le groupe des fichiers spécifiés sur la ligne de commande.

Options courantes

`-R`	récursif sur tous les fichiers et sous-répertoires contenus si `nom` est un répertoire
`-h`	traitement sur les liens symboliques
`nom`	exprime le nom d'un fichier ou d'un répertoire
`utilisateur`	représente soit le nom de l'utilisateur, soit son UID (User IDentification)
`groupe`	représente soit le nom du groupe, soit son GID (Group IDentification)

Exemple :

```
$ chown root villes
$ chown 102 villes
$ chgrp news villes
```

Remarque :

Seul le propriétaire des fichiers traités ou **root** ont le droit d'utiliser `chown` et `chgrp`.

TP N°8

find

Syntaxe :

find répertoire option1 [option2...]

Description :

La commande **find** permet de rechercher un fichier dans l'arborescence à partir du point spécifié.

Options courantes:

-name fich	recherche sur le nom fich
-size n	recherche sur la taille en blocs
-ctime n	recherche sur la date de création
-exec cmd {} \;	exécute la commande cmd sur les fichiers trouvés
-print	affiche le résultat de la recherche
!	négation du critère de recherche

Exemple :

```
$ find $HOME -name "vil*" -print
$ find . -print|cpio -ocvB /dev/streamer
$ find / -name "profile*" - exec pg {} \;
```

Représente le nom des fichiers trouvés.

Remarque :

Il est nécessaire de faire suivre l'option **-exec** par {} \;

Métacaractère	Utilisation
?	indétermination d'un caractère
*	indétermination de **0** à **n** caractères
[xyz123...]	indétermination d'un caractère pris dans la liste
[b-t]	indétermination d'un caractère pris dans l'intervalle
[!xyz123...]	indétermination d'un caractère pris à l'extérieur de la liste
[!b-t]	indétermination d'un caractère pris à l'extérieur de l'intervalle

TP N°9

grep

Syntaxe :

grep [-ilsfv] expression [fichier...]

Description :

La commande **grep** permet de rechercher `expression` dans `fichier` . Elle affiche les noms de fichiers ainsi que les lignes contenant `expression` .

Options courantes

`-i`	ne tient pas compte des minuscules et des MAJUSCULES
`-l`	n'affiche que le nom des fichiers (pas les lignes)
`-s`	pas de message d'erreur sur les fichiers inaccessibles
`-f fich`	spécifie un fichier contenant les expressions à rechercher
`-v`	affiche toutes les lignes, sauf celles qui contiennent l'expression
`expression`	chaîne de caractères (ou expression régulière non abordée dans ce cours)
`fichier`	nom des fichiers à traiter

Exemple :

```
$ grep Sarrebourg *
villes : Sarrebourg
$ grep -l Sarrebourg *
villes
```

Remarque :

head
tail

Syntaxe :

head [-n] [fichier...]
tail [-n|+n] [-f] [fichier]

Description :

La commande **head** affiche les n premières lignes d'un fichier, alors que **tail** affiche les dernières lignes d'un fichier.
Si n n'est pas précisé, il prend la valeur 10.

Options courantes

-n	nombre de lignes à afficher depuis le début/la fin de fichier
+n	affichage à partir de la ligne numéro n
-f	attente de nouvelles lignes (sortie par *Ctrl-c*)
fichier	nom des fichiers à traiter

Exemple :

```
$ head -2 villes
Sarrebourg
Douai
$ tail -2 villes
Couvron
Phalsbourg
$ tail +2 villes
Douai
Couvron
Phalsbourg
```

Remarque :

ln

Syntaxe :

```
ln [-s] fichier1 fichier2
ln [-s] fichier1 [fichier2...] répertoire
```

Description :

La commande **ln** permet de créer des entrées multiples dans l'arborescence d'un système de fichiers pour un même fichier physique.
Ce qui revient à dire que si l'on modifie un fichier, ses liens le sont aussi.
ln permet aussi de faire des liens dans des systèmes de fichiers différents par la méthode des liens symboliques (un peu comme les raccourcis de chez MS).
Si le dernier argument de la ligne de commande est un répertoire, **ln** crée des liens dans ce répertoire pour tous les fichiers pré-cités (`fichier1, fichier2, ...`).

Option :

`-s`	permet de faire un lien symbolique

Exemple :

```
$ ln villes villes5
$ ls villes*
-rwxr--r--  2  root  other  5423661   Apr  1  1997  villes
-rwxr--r--  2  root  other  5423661   Apr  1  1997  villes5
$ ln -s villes villes10
$ ls villes*
-rwxr--r--  2  root  other  5423661   Apr  1  1997  villes
-rwxr--r--  2  root  other  5423661   Apr  1  1997  villes5
lrwxr--r--  1  root  other  5423661   Apr  1  1997  villes10  -> villes
```

Remarque :

Les liens peuvent aussi concerner des répertoires (dans ce cas, ce seront toujours des liens symboliques).

TP N°10

sort

Syntaxe :

sort [-ufnr] [-o fic] [fichier...]

Description :

La commande **sort** trie les lignes des fichiers en arguments et affiche le résultat à l'écran. Le clavier est lu si `fichier` est omis.
Par défaut **sort** effectue un tri par ordre alphabétique; mais les options suivantes en modifient les critères.

Options courantes

`-u`	permet de n'afficher qu'une seule fois les lignes multiples
`-f`	ne différencie pas les minuscules et MAJUSCULES
`-n`	effectue un tri numérique
`-r`	ordre décroissant
`-o fic`	enregistre la sortie dans `fic`

Exemple :

```
$ sort villes
Couvron
Douai
Phalsbourg
Sarrebourg
$ sort -r villes
Sarrebourg
Phalsbourg
Douai
Couvron
```

Remarque :

TP N°11

umask

Syntaxe :

umask [???]

Description :

La commande **umask** permet de définir les droits affectés par défaut aux fichiers lors de leur création.
Si le masque ??? est omis, alors umask affiche le masque en cours.

Options courantes

???	Chaque ? représente une valeur entre 0 et 7 qui est le complément à 7 des droits à affecter aux fichiers. Si l'on veut avoir des fichiers avec 751 (rwxr-x--x) comme droits, il faudra définir comme masque 026 (on remarque que 7+**0** = 5+**2** = 1+**6** = 7).

Exemple :

```
$ umask 022
$ umask
022
```

Vos fichiers seront donc créés avec les droits 755

Remarque :

Il faut noter que les droits affectés à la création d'un fichier dépendent aussi de l'utilitaire qui les a créés; si vous avez un masque 000 et que vous créez un fichier avec vi, les droits effectifs de votre fichier sont 666 (rw-rw-rw-) car vi est un éditeur de texte et non de programmes shells.
À l'inverse, quel que soit le masque utilisé, le compilateur c cc (programme permettant de créer des fichiers programmes) positionnera toujours les droits d'exécution sur les fichiers qu'il crée.

TP N°12

WC

Syntaxe :

wc [-lwc] [fichier...]

Description :

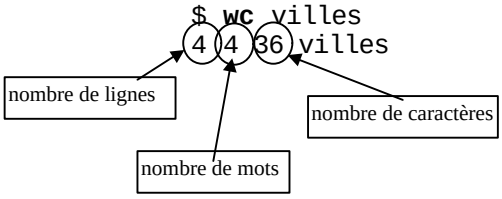
La commande **WC** compte le nombre de lignes, mots, ou caractères d'un fichier texte
Si aucun fichier n'est passé en paramètre, c'est l'entrée standard qui sera lue.
Si aucune option (-lwc) n'est précisée, alors **WC** compte le nombre de lignes, mots, **et** caractères du fichier.

Options courantes

`-l`	précise que c'est le nombre de lignes qui doit être compté
`-w`	précise que c'est le nombre de mots qui doit être compté
`-c`	précise que c'est le nombre de caractères qui doit être compté

Exemple :

```
$ wc -l villes
4 villes
$ wc villes
 4  4 36 villes
```



Remarque :

TP N°13

Archivage et restauration de données

⇒ **`cpio`**.

⇒ **`tar`**.

cpio

Syntaxe :

```
cpio -i[cvBdmut] [-E fic] <  fichier_archive
cpio -o[cvB] > fichier_archive
```

Description :

La commande **cpio** permet d'archiver les fichiers dont les noms sont reçus sur l'entrée standard et de restaurer les fichiers d'une archive.
Les archives peuvent être soit des fichiers normaux, soit des fichiers spéciaux de type blocs, ce qui permet de mettre les archives directement sur un support physique (streamer, DAT, disquette, ...).
Par sa manière de créer une archive, cpio est entièrement portable entre différents systèmes (UNIXWARE esim2, SINIX esim1, ...). Il doit par conséquent être votre outil privilégié pour l'archivage.
ATTENTION, si une archive a été créée avec des chemins absolus, il n'est pas possible de les restaurer ailleurs qu'à leur emplacement d'origine.

Options courantes

`-i`	restauration ou listage d'une archive existante
`-o`	création d'archive cpio
`-c`	ajout d'un en-tête ASCII (portabilité)
`-v`	mode bavard
`-B`	écrit des blocs de 5 Ko
`-d`	restauration de l'arborescence
`-m`	restauration des dates de modification des fichiers
`-u`	restauration inconditionnelle
`-t`	listage des fichiers de l'archive
`-E fic`	spécifie le fichier `fic` a restaurer

Exemple :

```
$ find . -print |  cpio -ocvBdmu >/ dev/streamer
fic1
fic2
rep1
rep1/fic11
rep2/fic12
$ cpio -icvBdmu < /dev/fd0
fic1
rep1
rep1/fic11
$ cpio -itv < / dev/dat
-rw-rw-rw-   1  root   other 192  Nov 30 10:36 1998,  fic1
drw-rw-rw-   1  root   other 331  Nov 30 10:36 1998,  rep1
-rwxr-xr-x   1  root   other 138  Apr 25 09:29 1997,  rep1/fic11
```

TP N°14

tar

Syntaxe :

```
tar c[vf] [fic_sortie] [fichier...]
tar x[vf] [fic_entree] [fichier...]
tar t[vf] [fic_entree] [fichier...]
```

Description :

La commande **tar** archive et restaure les fichiers entrés sur la ligne de commande.
ATTENTION, si une archive a été créée avec des chemins absolus, il n'est pas possible de la restaurer ailleurs qu'à son emplacement d'origine.

Options courantes

c	création d'un fichier d'archive
x	extraction de fichiers d'une archive
t	listage du contenu d'une archive
v	mode bavard
f	précise le fichier d'archive à utiliser

Exemple :

```
$ tar cvf /dev/fd0 *
a fic1 1 tape block
a rep1   1 tape block
a rep1/fic11 1tape block
$ tar tvf archive
-rw-r--r--102/100     52 Jul  8 11:26 1998   fic1
drw-r--r--102/100     34 Jul  8 11:26 1998   rep1
-rw-rw-rw-102/100     36 Nov 30 09:07 1998   rep1/fic11
$ tar xvf /dev/streamer
x fic1, 52 bytes, 1 tape block
x rep1, 34 bytes, 1 tape block
x rep1/fic11, 36 bytes, 1 tape block
```

Remarque :

TP N°15

Utilitaires réseau

⇒ **`ping`** vérification d'une connexion réseau.

⇒ **`telnet`** connexion au travers du réseau.

ping

Syntaxe :

ping correspondant [délai]

Description :

La commande **ping** envoie sur le réseau des paquets de réflexion. C'est à dire que le destinataire renvoie les paquets à l'émetteur.
Cette commande permet donc de vérifier une connexion réseau entre deux correspondants.

Options courantes

correspondant	nom du correspondant ou adresse IP
délai	délai d'attente entre l'émission d'un paquet et sa réponse

Exemple :

```
$ ping www.linux-france.com
www.linux-france.com  is alive
$ ping www.zindows-france.com
UX:ping: INFO: no  answer  from www.zindows-france.com
```

Remarque :

telnet

Syntaxe :

telnet [correspondant]

Description :

La commande **telnet** permet d'ouvrir une session sur une machine distante.
Si le correspondant n'est pas précisé sur la ligne de commande, telnet fonctionnera en mode interactif (prompt : telnet>) et le résumé des commandes s'obtient avec ? .

Options courantes

correspondant nom du correspondant ou adresse IP

Exemple :

```
$ telnet www.linux-france.com
Trying 192.124.13.42...
Connected to www.linux-france.com.
Escape character is '^]'.

Red Hat Linux release 5.1 (Manhattan)
Kernel 2.0.35 on an i586
login:
```

Remarque :

Commandes d'administration

⇒ **`id`** identification d'utilisateur et de groupe

⇒ **`ps`** liste et état des processus

⇒ **`passwd`** changement d'un mot de passe

⇒ **`kill`** émission de signal aux processus

⇒ **`who`** état des connexions au système

⇒ **`df`** état d'occupation des systèmes de fichiers

⇒ **`su`** changement d'identité

⇒ **`which`** localisation d'une commande ou alias .

id

Syntaxe :

id [utilisateur]

Description :

La commande **id** affiche des informations concernant le numéro d'utilisateur (UID) ainsi que sur les groupes d'appartenance (GID).
Si `utilisateur` est omis, `id` affiche les informations concernant l'utilisateur courant.

Options courantes

`utilisateur` nom d'un utilisateur connu du système

Exemple :

```
$ id
uid=102( rsl) gid=100( other) groups=101( ftp)
$ id root
uid=0( root) gid=3( sys) groups=0( root), 1( other), 2( bin), 3( sys),
4(adm), 5( uucp), 6(mail), 7( tty), 8(audit), 10( nuucp), 12( daemon),
23(cron), 25( dtadmin), 47( priv),9( lp)
```

Remarque :

ps

Syntaxe :

ps [-ef] [-t liste] [-u liste]

Description :

La commande **ps** affiche l'état des processus; si aucune option n'est donnée, ce sont les processus de la session active qui sont affichés.

Options courantes

`-e`	affiche tous les processus du système
`-f`	affiche les information au format long
`-t liste`	affiche les processus liés aux terminaux de la `liste`
`-u liste`	affiche les processus liés aux utilisateurs de la `liste`

Exemple :

```
$ ps
  PID   CLS   PRI   TTY   TIME  COMD
  6665  TS    70    pts/3 0:00  ksh
  9280  TS    59    pts/3 0:00  ps
$ ps -ef | head -5
UID   PID   PPID   CLS  PRI C  STIME   TTY TIME COMD
root  0     0      SYS  79  0  Nov 28  ?   0:14 sysproc
root  1     0      TS   70  0  Nov 28  ?   0:03 /sbin/init
root  1019  1      TS   85  0  Nov 28  ?   0:00 /usr/lib/saf/sac -t 300
root  88    1      TS   88  0  Nov 28  ?   0:00 /usr/lib/mousemgr
$ ps -u rsl
PID   CLS   PRI     TTY      TIME  COMD
964   TS    80      ?        0:00  srv_tab
958   TS    80      ?        0:00  xlemsup
961   TS    80      ?        0:00  srv_imp
6665  TS    70      pts/3    0:00  ksh
9323  TS    59      pts/3    0:00  ps
```

Remarque :

Sous Linux , il faut utiliser l'option **x** pour voir tout les processus.

passwd

Syntaxe :

passwd [utilisateur]

Description :

La commande **passwd** permet à l'utilisateur de modifier son mot de passe.
Si vous êtes `root`, il vous est alors possible de modifier le mot de passe des autres utilisateurs.

Options courantes

`utilisateur` nom d'un utilisateur du système

Exemple :

```
$ passwd
UX:passwd: INFO:  Changing  password for  rsl
Old password:
New password:
Re-enter new  password:
$
```

Remarque :

TP N°16

kill

Syntaxe :

kill [-sig] num_process

Description :

La commande **kill** envoie au processus portant le numéro num_process un signal (sig). Par défaut, c'est le signal 15 (TERM) qui est envoyé.

Options courantes

-sig signal valide à transmettre. Les plus courants sont :
15 (TERM) demande au processus de se terminer (proprement!!!)
9 (KILL) demande au processus de se terminer (inconditionnel)

num_process numéro d'un processus (PID)

Exemple :

```
$ ps
   PID   CLS   PRI   TTY   TIME  COMD
   6665  TS    70    pts/3 0:00  ksh
   9280  TS    59    pts/3 0:00  ps
$ kill -9 6665

SURPRISE !!!
```

Remarque :

Le numéro du processus (PID) peut être déterminé avec la commande **ps**
La commande kill ne vous permettant pas de tuer les tâches des autres utilisateurs (seul le compte root peut le faire).

who

Syntaxe :

who am i
who

Description :

La commande **who** affiche les utilisateurs connectés au système.
Elle permet aussi de vous informer sur votre connexion.

Options courantes

am i	'qui suis-je' en français; information sur votre connexion

Exemple :

```
$ who
root     tty0         Nov 30 10:41
rsl      ttyp0        Nov 30 15:28 (192.152.230.14)
rsl      pts/3        Nov 30 09:07 (192.152.230.15)
$ who am i
rsl      pts/3        Nov 30 09:07 (192.152.230.15)
```

Remarque :

ATTENTION, si vous avez utilisé la commande **su**, **who** ne vous donnera que les informations de votre connexion initiale.

df

Syntaxe :

df -k

Description :

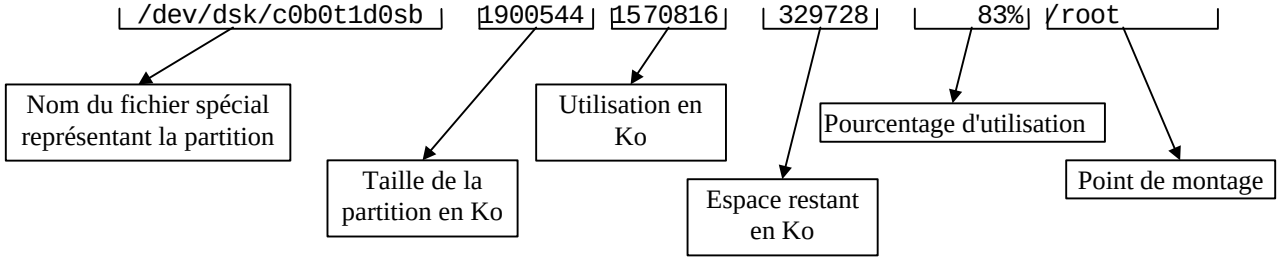
La commande **df** vous donne des informations sur l'état d'occupation des systèmes de fichiers.
Par défaut, df donne ces indications en blocs

Options courantes

-k donne les indications en Ko

Exemple :

```
$ df -k
filesystem             kbytes     used     avail  capacity  mounted on
/dev/root              500736   344072    156664       69%  /
/proc                       0        0         0        0%  /proc
/dev/dsk/c0b0t0d0sa     16384     4298     12086       26%  /stand
/dev/fd                     0        0         0        0%  /dev/fd
/dev/dsk/c0b0t1d0s3    200704    49800    150904       25%  /var
/dev/dsk/c0b0t0d0s3    300032    95896    204136       32%  /home
/dev/dsk/c0b0t0d0s4   1210368   251472    958896       21%  /home
/dev/dsk/c0b0t0d0sb   1900544  1700720    199824       89%  /rsl
/dev/dsk/c0b0t1d0s1   1935360   745184   1190176       39%  /ux
/dev/dsk/c0b0t1d0sb   1900544  1570816    329728       83%  /root
```



Remarque :

su

Syntaxe :

su - [utilisateur]

Description :

La commande **su** permet de changer en cours de session l'utilisateur courant.

Par défaut, si utilisateur n'est pas précisé, **su** essaie de vous connecter `root`

Options courantes

`-`	charge le profil de l'utilisateur (variables, ...)
`utilisateur`	nom d'un utilisateur connu du système

Exemple :

```
$ su
Password:
$ su - rsl
Password:
```

Remarque :

TP N°16

which

Syntaxe :

which [commande]

Description :

La commande **which** vous indique la commande qui sera exécutée si vous tapez commande .
which effectue une recherche dans le PATH.

Options courantes

commande	commande telle que vous la tapez au clavier

Exemple :

```
$ which ls
/usr/bin/ls
$ which startx
/usr/bin/X11/startx
```

Remarque :

Commandes orientées shell

⇒ **`echo`** affichage de texte sur la sortie standard

⇒ **`expr`** évaluation d'expressions numériques

⇒ **`test`** évaluation d'expressions diverses

⇒ **`clear`** efface l'écran

echo

Syntaxe :

echo [-n] message

Description :

La commande **echo** affiche sur la sortie standard les messages passés en paramètres (après leur interprétation par le shell).

Options courantes

-n	n'affiche pas de saut de ligne final

Exemple :

```
$ echo ceci est un petit message
ceci est un petit message
$ echo $PATH
/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/X11R6/bi
n:/home/rossignol_d/bin
$ echo \$PATH
$PATH
```

Remarque :

TP N°17

expr

Syntaxe :

expr exp1 { **+** | **-** | ***** | **/** | **%** } exp2

Description :

La commande **expr** évalue l'expression de la ligne de commande et retourne le résultat sur la sortie standard.

Options courantes

exp*X*	constante numérique ou variable du même type
+ | - | * | / | %	opérateur logique représentant respectivement l'addition, la soustraction, la multiplication (précédées d'un \ pour que le caractère * ne soit pas interprété par le shell), la division, le modulo (reste d'une division entière)

Exemple :

```
$ a=a+1
$ echo $a
a+1
$ a=0
$ a=`expr a + 1`
$ echo $a
1
$ echo `expr 23 % 6`
5
```

Remarque :

Il est indispensable de mettre un espace entre exp1 et l'opérateur, et entre l'opérateur et exp2.

TP N°18

test

Syntaxe :

```
test [expression]
[ expression ]
```

Description :

La commande **test** évalue `expression` et si sa valeur est vraie, retourne un code de sortie zéro

Options courantes

`ch1 = ch2`	chaines de caractères `ch1` et `ch2` identiques
`ch1 != ch2`	chaines de caractères `ch1` et `ch2` différentes
`nb1 -eq nb2`	nombres `nb1` et `nb2` égaux
`nb1 -ne nb2`	nombres `nb1` et `nb2` différents
`nb1 -gt nb2`	nombre `nb1` supérieur à `nb2`
`nb1 -ge nb2`	nombre `nb1` supérieur ou egal à `nb2`
`nb1 -lt nb2`	nombre `nb1` inférieur à `nb2`
`nb1 -le nb2`	nombre `nb1` inférieur ou egal à `nb2`
`-r fic`	vrai si `fic` existe et est lisible
`-w fic`	vrai si `fic` existe et est accessible en écriture
`-x fic`	vrai si `fic` existe et est exécutable
`-f fic`	vrai si `fic` existe et est un fichier ordinaire
`-d fic`	vrai si `fic` existe et est un fichier répertoire
`-a`	opérateur logique ET
`-o`	opérateur logique OU

Exemple :

```
$ test $USERNAME =  rsl
$ echo $?
1
$ [ -d /tmp ]
$ echo $?
0
```

Affichage du code de sortie (*exit status*).
1= FAUX
0=VRAI

Remarque :

cette commande est le plus souvent associée à des instructions de controle (if, while, ...)dans les scripts shell.

TP N°19

clear

Syntaxe :

clear

Description :

La commande **clear** efface l'écran du terminal actif.

Options courantes

clear n'accepte pas d'option

Exemple :

```
$ clear
```

Remarque :

Certains systèmes permettent d'effacer un autre terminal que le sien (**clear** tty12).

ANNEXE 1 :

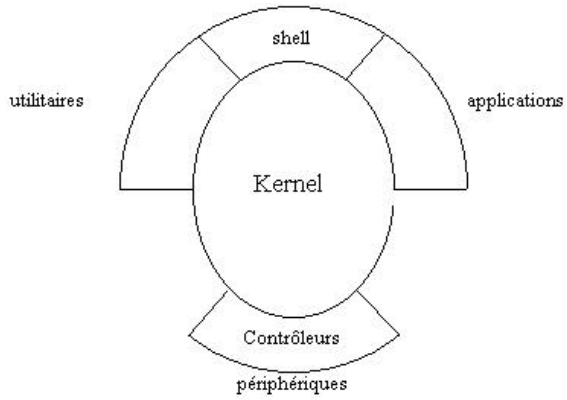
Présentation complémentaire du système UNIX.

Afin d'approfondir plus en détail le fonctionnement interne d'UNIX vous trouverez dans cette partie du cours un complément d'information de la partie théorique.

Architecture d'UNIX :

Afin d'assurer une certaine stabilité au système, UNIX a été conçu autour d'une architecture en couche. Il y a trois couches fondamentales qui forment le système :

- Le noyau (Kernel) ;
- La couche de gestion des périphériques ;
- La couche de processus utilisateurs.



Notions de processus :

Un processus est n'importe quel traitement, appelant un ou plusieurs programmes et produisant un résultat. Une seule copie d'un programme en mémoire peut être utilisée par plusieurs processus (***réentrance***) ; lorsque des utilisateurs créent des processus qui font appel aux mêmes programmes, la distinction entre processus est faite par leur image, qui par définition est l'ensemble des éléments nécessaires à l'exécution d'un processus à un instant donné (état de la mémoire, des registres, des fichiers, d'un processus).
Un processus peut se dupliquer par le biais de l'appel système `fork` ; le processus créé est alors appelé 'fils', et son image diffère du 'père' uniquement par son numéro d'identification (*PID*).
Les processus dits 'système' sont enclenchés par le noyau.

Chaque processus utilisateur dialogue avec l'extérieur par le biais de trois fichiers particuliers qui sont ouverts en permanence :

- l'entrée standard (standard input, handle 0) ;
- la sortie standard (standard output, handle 1) ;
- la sortie d'erreur standard (standard error output, handle 2) ;

par défaut, ces fichiers sont liés au terminal, et représentent le clavier (input) et l'écran (output). Il est possible de rediriger ces fichiers standards vers des fichiers différents en utilisant les sigles de redirection '<'et '<<' représentant l'entrée du processus, et '>' et '>>' représentant la sortie du processus (voir **Le shell/Redirection des entrées-sorties** pour plus de détails)

Un processus peut prendre six états différents :

- **en cours**(d'éxécution par le processeur) --> O(n the processeur)
- **actif** (prêt à l'exécution) ->R(unnable);
- **endormi** (en attente) ->S(leeping);
- **invalide** (en attente de mémoire) ->X (SXBRK);
- **créé** (état transitoire, processus en cours de création) ->I(dle);
- **zombie** (l'exécution du processus est terminée, mais ses éléments restent visibles) ->Z(ombie);
- **arrêt** (processus mis en attente par l'utilisateur)->T(raced).

La liste de tous les processus avec leur états peut être affichée avec la commande **`ps -el`**.
Lorsqu'un processus est actif, il a le contrôle du processeur de l'unité centrale ; lorsqu'il change d'état, le système utilise un mécanisme de ' swapping' et écrit l'image du processus sur le disque dur (dans une zone réservée !).

Communication et synchronisation entre processus :

Nous venons de voir que les entrées et sorties d'un processus pouvaient êtres détournées vers des fichiers ; mais puisque plusieurs processus peuvent s'exécuter en même temps, il pourrait être utile de rediriger ces sorties vers l'entrée d'autres processus, ou plus généralement d'échanger des données entre processus. Les mécanismes suivants sont mis en place à cet effet :

- **Les Pipes (tubes)**
  Avec l'établissement d'un pipe entre deux processus, la sortie du premier devient l'entrée du second. Le symbole '|' sert à établir ce pipe.
  Exemple : **`ls | pg`**
- **Les piles FIFO (First In First Out) ou pipes nommés**
  Ils agissent comme les pipes ordinaires, la différence étant qu'ils ont une entrée dans un répertoire et peuvent être utilisés par des processus indépendants.
- **Les événements**
  Le système annonce au processus actif l'arrivée d'un événement ; cela peut être utilisé soit pour déclencher (arrêter) un processus, soit pour synchroniser l'exécution de plusieurs processus. L'utilisateur peut introduire des événements à l'aide de la commande kill ; certaines touches de contrôle sont normalement associées à des événements.
  (Ex : **Ctrl-C** → interruption)

Le noyau :

Le noyau gère les tâches de base du système :

- l'initialisation;
- la gestion des processus système;
- la gestion des processus utilisateurs;
- la gestion du processeur et de la RAM;
- la gestion des systèmes de fichiers.

Le partage du temps processeur entre les processus actifs est géré par le processus système appelé *'scheduler'* et il est basé sur l'horloge de l'ordinateur. À des intervalles réguliers (de l'ordre d'un centième de seconde), le processeur abandonne l'exécution du processus courant et passe à l'exécution du premier processus de plus haute priorité en attente.
Pour accomplir les tâches qui lui sont propres, le noyau accède à un groupe de tables, dites système, qui concernent les processus actifs, les fichiers ouverts, le super bloc (voir *gestion des fichiers*), les buffers d'entrée-sortie, etc.
À chaque intervalle, un examen de la table des processus permet au noyau de déterminer lequel doit être exécuté; les données propres à chaque processus actifs (environnement de variables, état des registres, ...) sont présentes dans une zone mémoire réservée du noyau. Avant l'exécution du processus sélectionné, le noyau passe la main au *'swapper'*, le processus système qui gère les ressources mémoire et procède si nécessaire à des échanges entre la RAM et le disque dur. Ensuite, le processus est enclenché (état **Actif**).

La gestion des fichiers :

L'exploitation de la mémoire de masse est réalisée par des structures appelées systèmes de fichiers (*file systems*), qui occupent un espace physique du support exprimé en blocs de 1024 octets.
Exemple : un disque de 100 Mo, abritant un système de fichiers qui l'occupe entièrement, est constitué d'à peu près 100 000 blocs.
Les deux premiers blocs sont réservés par le système. Le premier bloc est vide sauf s'il s'agit du système de fichiers principal (bootable), qui dans ce cas contient un programme appelé '*bootstrap*' (il est chargé en mémoire à l'initialisation).
Le second bloc, appelé 'super bloc', contient les informations significatives concernant le système de fichiers; un de ses rôles étant à tout moment de répertorier l'occupation des blocs du système de fichier, le noyau doit le modifier à chaque modification ou création de fichier.
Pour des raisons d'optimisation, le noyau n'écrit pas sur le disque mais modifie une image du super bloc chargée en mémoire vive qui, à certains moments dans la vie du système (et notamment lors de l'arrêt de la machine), est réécrite sur le disque. Dans le cas d'un arrêt brutal du système, au démarrage suivant, le super bloc ne correspond plus à l'occupation effective des blocs et l'utilitaire fsck (File System ChecK) s'avère nécessaire pour reconstituer le super bloc (ATTENTION : cette procédure ne fonctionne pas toujours !!!).

Structure d'un file system (1 bloc = 1024 octets)	
N° DE BLOC	**CONTENU**
0	'*bootstrap*' pour une partition principale, sinon vide.
1	Super bloc
2 à N	Table des inodes (16 inodes par blocs).
N+1 à M	Adresses d'indirection.
De M+1 à 95% du FS	Zone de données (Fichiers - espace libre).
Les 5 % restants	Zone réservée par le système en cas de saturation de la zone de données
Les adresses d'indirection sont utilisées pour la localisation des fichiers de taille supérieure à 10 blocs. N et M correspondent à des nombres calculés par la machine lors de la création du FS en fonction du nombre d'inodes que vous avez réservés. Pour créer un FS, il vous faut utiliser la commande mkfs (MaKe File System).	

Structure du super bloc	
Champ	**Contenu**
s_isize	taille en blocs de la table des inodes
s_fsize	taille en blocs du file system
s_nfree	liste des blocs libres
s_free	adresse du premier bloc libre
s_ninode	liste des inodes libres
s_inode	numéro du premier inode libre
s_flock	flag de verrouillage pendant la mise à jour des 4 champs ci-dessus
s_ilock	flag de verrouillage pendant la mise à jour de la table des inodes
s_fmod	flag de modification du super bloc
s_ronly	flag de lecture seule
s_time	date de la dernière modification du super bloc
s_dinfo	information sur le device (sur lequel se trouve le FS)
s_tfree	nombre de blocs libres
s_tinode	nombre d'inodes libres
s_fname	nom du file system
s_fpack	nom 'pack' du file system
s_fill	zone d'ajustement
s_state	état du file system
s_magic	nombre magique du file system
s_type	type du file system

Dans le super bloc, le noyau lit les informations concernant la table des inodes et par son biais, accède aux fichiers.
Un inode est une structure de 64 octets contenant 10 champs qui décrivent les propriétés d'un fichier, y compris les moyens pour y accéder.
Structure d'un inode :

- type de fichier
- nombre de liens
- UID (User Identification) numéro d'utilisateur du propriétaire
- GID (Group Identification) numéro du groupe propriétaire
- taille du fichier en octets
- adresses des blocs de données (qui contiennent le fichier)
- droits du fichier
- date du dernier accès
- date de dernière modification
- date de création

La table des inodes regroupe l'un après l'autre autant d'inodes que de fichiers contenus dans le FS. Le numéro d'inode d'un fichier correspond à son rang dans la table des inodes.

Le premier champ constituant un inode est le type du fichier ; il existe trois types de fichiers en UNIX :

- Les fichiers répertoires;
- les fichiers ordinaires;
- les fichiers spéciaux (trois types possibles : type caractère, type bloc, et pipes nommés).

Les répertoires accessibles en lecture, permettent à l'utilisateur de cataloguer les fichiers contenus. Un répertoire est un fichier spécial regroupant des enregistrements de 16 octets, ou chaque structure contient le numéro d' inode (2 octets), et le nom du fichier (14 octets). À la création d'un répertoire, deux structures représentant le répertoire courant et le répertoire 'père', sont intégrés au fichier. Le système d'exploitation se charge de mettre à jour leur contenu tout au long de l'exploitation.
Exemple d'un fichier répertoire :

Numéro d'inode	Nom du fichier
24	.
23	..
26	fic1
27	fic2
35	rep1

Les fichiers ordinaires stockent les programmes et les données; le système n'impose aucun format particulier aux fichiers et les traite comme des séquences d'octets.
Les fichiers spéciaux sont des contrôleurs de périphériques qui permettent de gérer les entrées-sorties de façon transparente; en effet, un processus utilisateur peut indifféremment envoyer ou recevoir des données vers/depuis un fichier ou un périphérique. Les fichiers spéciaux correspondant aux périphériques sont stockés dans le répertoire /dev (devices) du système de fichiers.

Les contrôleurs de périphériques

La gestion des Entrées-sorties n'est pas accomplie par le noyau, mais par la couche logicielle, afin d'assurer l'une des prérogatives d'UNIX : l'indépendance vis à vis du matériel utilisé. Chaque périphérique rattaché au système est associé à un fichier possédant un numéro d' inode; dans l'inode correspondant sont contenus le 'major' et le 'minor device number' ainsi que le type bloc, ou caractère du périphérique.
L'accès à un de ces fichiers appelés contrôleurs est faite au départ comme dans le cas d'un fichier ordinaire : du numéro d' inode, on accède au noeud d'index associé et aux valeur contenues.
Après avoir reconnu qu'il s'agit d'un fichier spécial, le système analyse les 'devices numbers' et détermine l'adresse mémoire des sous-programmes à exécuter pour réaliser la liaison avec le périphérique en question.
Il existe deux types de contrôleurs, selon que les opérations d'entrées-sorties sont réalisées en mode caractère ou en mode bloc. Dans le premier cas, le contrôleur doit se charger de la gestion du transfert, qui s'effectue sans usage de tampons. Dans le second cas, en mode bloc, les données transitent par des

buffers de 1 Ko et le système se charge de gérer la mise en séquence des phases de transfert.
Les contrôleurs en mode bloc sont plus faciles à écrire, mais sont plus lents à exécuter ; ils sont pourtant indispensables pour tout périphérique destiné au traitement des données sous forme de fichier, puisque UNIX gère les file systems par blocs.

Les droits d'accès aux fichiers et répertoires

Pour être admis à l'exploitation du système, il faut que vous possédiez un **compte** qui se compose d'un nom d'utilisateur et d'un mot de passe.
Unix possède différents niveaux de sécurité qui sont :

- le propriétaire du fichier ou répertoire (un répertoire étant un fichier particulier),
- le groupe propriétaire du fichier,
- et le reste des utilisateurs.

Par défaut, lors de la création d'un fichier, son propriétaire est la personne qui l'a créé, et le groupe propriétaire est le groupe principal du créateur.
À un nom d'utilisateur (tout comme à un nom de groupe), est associé un numéro.
Exemple : l'utilisateur ***root***, a pour numéro d'utilisateur (UID) ***0***.
À chaque niveau de sécurité, il est possible de déterminer un certain nombre d'autorisations :

- **La lecture (R ou 4)** : pour un fichier, ce droit permet la lecture du fichier; alors que pour un répertoire, il autorise l'utilisateur à lister son contenu.
- **L'écriture (W ou 2)** : pour un fichier, il permet sa modification; alors que pour un répertoire, il permet la création et la suppression des fichiers du répertoire (**ATTENTION** : cette permission est valable quels que soient les droits des fichiers).
- **L'exécution (X ou 1)** : pour un fichier, il autorise son exécution ; et pour un répertoire, il permet de se positionner dessous

ANNEXE 2 :

Travaux pratiques

.

TP 1 :

En utilisant la commande `cat`, créer un fichier appelé 'Villes' et contenant les lignes suivantes :

```
Metz
Sarrebourg
Laon
```

Terminer la saisie par *'Ctrl-d'*

```
$ cat > Villes
Metz
Sarrebourg
Laon
'Ctrl-d'
```

Ajouter les lignes suivantes :

```
Douai
Compiegne
```

Terminer la saisie par *'Ctrl-d'*

```
$ cat >> Villes
Douai
Compiegne
'Ctrl-d'
```

Créer un fichier appelé 'liste' contenant la liste des fichiers du répertoire et de ses sous-répertoires.

NB : la commande permettant de lister le contenu du répertoire est **ls**

```
$ ls > liste
```

TP 2 :

Afficher la liste des fichiers présents dans le répertoire dans l'ordre alphabétique.

```
$ ls | sort
```

TP 3 :

Taper une ligne de commande qui affiche le message `'Le fichier est bien présent sur le disque dur'` seulement lorsqu'un fichier `'fic1'` est présent dans le répertoire courant.

NB : commandes à utiliser : **ls**, **echo**

```
$ ls fic1 && echo Le fichier...
```

TP 3 bis:

Afficher tous les fichiers contenants la lettre `p`

```
$ ls *p*
```

Afficher les fichiers commençants par un `l`

```
$ ls l*
```

TP 4 :

Aller dans le répertoire de connexion de votre voisin en utilisant les chemins absolus.

```
$ cd /home/stageX
```

Retourner dans votre répertoire de connexion.

```
$ cd
```

Aller dans le répertoire d'un autre stagiaire en utilisant les chemins relatifs.

```
$ cd ../stageX
```

TP 5 :

Après être retourné sous votre répertoire de connexion, créer les répertoires *'rep1'* ***et*** *'rep3'*.

```
$ cd
$ mkdir rep1 rep3
```

Aller sous le répertoire de connexion de votre voisin et créer le répertoire *'rep2'*

```
$ cd ../stageX
$ mkdir rep2
Erreur due au manque de droits.
```

TP 6 :

De retour sous votre répertoire de connexion, supprimer le répertoire *'rep3'*.

```
$ cd
$ rmdir rep3
```

Essayer de supprimer le répertoire *'rep1'* ***de votre*** voisin.

```
$ rmdir ../stageX/rep1
Erreur due au manque de droits sur le répertoire ../stageX même si les droits de rep1 vous permettent d'y ajouter des fichiers ( ls -l)
```

TP 7 :

Attribuer à votre répertoire de connexion les droits suivants: rwxrwxr-x

```
$ chmod 775 .
```

Refaire l'alinéa 2 du TP 6.

Attribuer au fichier *'liste'* ***les droits suivants*** r--r--r--

```
$ chmod 444 liste
```

Aller détruire le fichier *'liste'* ***de votre voisin***.

```
$ rm ../stageX/liste
Cela fonctionne grace au droits du répertoire ../stageX
```

TP 8 :

Dans votre répertoire, recréer le fichier `'liste'` auquel vous attribuerez les droits suivants `444`

```
$ ls > liste
$ chmod 444 liste
```

Changer le propriétaire du fichier

Nouveau propriétaire : stage(X-1).
Exemple : stage9 donnera à stage8

```
$ chown stageX liste
```

Ajouter les droits en écriture pour tout le monde sur ce fichier.

```
$ chmod +w liste
Erreur car nous ne sommes plus propriétaires
```

TP 9 :

Rechercher à partir de /home tous les fichiers contenants au moins un chiffre dans leur nom et les afficher.

NB : les erreurs seront enregistrées dans le fichier ERR_find.

```
$ find /home -name "*[0-9]*" -print 2>ERR_find
```

TP 10 :

Créer un lien du fichier `'Villes'` dans le répertoire de votre voisin.

Vous l'appelerez `'Cite'`

```
$ ln Villes ../stageX/Cite
```

Détruire votre fichier `'Villes'`.

```
$ rm Villes
```

Aller voir l'état du lien précédemment créé dans le répertoire de votre voisin

```
$ ls ../stageX/Cite
Erreur car le fichier physique n'existe plus.
```

TP 11 :

Créer un fichier `'mat'` contenant les lignes suivantes :

```
disquette
clavier
souris
ecran
disquette
ecran
claviers
```

```
$ cat > mat
disquette
clavier
souris
ecran
disquette
ecran
claviers
```

Créer un fichier `'tri'` contenant un seul exemplaire des lignes de `'mat'` triées dans l'ordre alphabétique inverse.

```
$ sort -ur -o tri mat
```

TP 12 :

On veut que les fichiers créés à compter de maintenant soient accessibles uniquement à leur propriétaire `rw-------`.

```
$ umask 077
ou
$ umask 177
```

TP 13 :

Compter le nombre d'erreurs de la recherche `find` ***du TP 9.***

```
$ wc -l ERR_find
```

Compter le nombre de fichiers présents dans votre répertoire.

```
$ ls | wc -l
```

TP 14 et 15 :

Créer un fichier `'arch'` archive de votre répertoire.

Le lister

Supprimer les fichiers et répertoires de votre répertoire de connexion (sauf `'arch'`).

```
$ rm -ri
ou
$ rm -r [!a]*
```

Restaurer `'arch'`

TP 16 :

changez votre mot de passe.

```
$ passwd
```

TP 17 :

Connectez-vous *site* ***avec l'environnement de démarrage, sans ouvrir de nouvelle session*** *Host Presenter.*

taper ***who am i***

taper ***id***

```
$ su - site
```

TP 18 :

Calculer le résultat de la multiplication de 8745 et de 4512 et stocker le résultat dans la variable *multip*.

```
$ multip=`expr 8745 \* 4512`
$ echo $multip
```

TP 19 :

Tester si votre terminal est de type *lanwp5*.

Tester si votre nom d'utilisateur est *site*.

```
$ test $TERM = lanwp5
$ echo $?

$ [ $LOGNAME = site ]
$ echo $?
```

ANNEXE 3 :

Index

A

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W