Chapitre 2 NOTIONS SUR UNIX

Avertissement : UNIX est une marque déposée de Bell Laboratories.

1. GENERALITES

1.1 Historique

Vers 1965, Bell Laboratories, General Electric et le M.I.T. lancent le projet MULTICS qui vise à créer un système d'exploitation **multitâches et multi-utilisateurs en temps partagé**, implantable sur tout ordinateur, assurant une **portabilité** des applications. Les chefs de projets étaient Ken THOMPSON et Dennis RITCHIE de Bell Laboratories. Une première version écrite en PL/1 est proposée en 1970, avec deux originalités : un système de gestion de fichiers arborescent et un **shell** conçu comme processus utilisateur. Mais c'est l'échec, PL/1 n'étant pas adapté aux objectifs.

Après le retrait de Bell Laboratories, lassé d'avoir investi 7 millions de $ sans retour, Ken THOMPSON et Dennis RITCHIE continuent seuls et réécrivent MULTICS en Assembleur. Sur la machine hébergeant le premier MULTICS, Ken THOMPSON crée le langage B en 1971 et en 1973, Dennis RITCHIE et Brian KERNIGHAN créent le langage C issu d'améliorations de B, en vue d'une réécriture d'UNIX, nom qui a succédé à MULTICS (*Uniplexed Information and Computer Service*). Ce fut la première version (V6) commercialisée en 1974. Depuis, plus d'une vingtaine de versions d'UNIX ont vu le jour, sans compter les systèmes très proches. Les quatre versions principales sont BSD, SYSTEM V, POSIX (tentative de fédérer les deux familles précédentes) et LINUX proposé par Linus TORVALDS en 1991. Le code source, écrit pour l'essentiel en C, est largement accessible.

Le label UNIX 98 de l'Open Group prescrit les threads (tâches) standardisés (cf. chapitre 11), le temps réel, les 64 bits et les systèmes de fichiers étendus (cf. chapitre 7).

Quelles que soient les versions :
UNIX = le noyau + l'interpréteur de commandes + des utilitaires

UNIX est construit autour de deux notions essentielles :
- **fichiers** organisés en structure arborescente
- **processus** : un processus correspond à l'exécution d'un programme (système ou utilisateur) et à son environnement (descripteurs de fichiers, registres, compteur ordinal, …). La gestion de la mémoire centrale constitue un aspect important du contrôle des processus.

1.2 Les entrées-sorties

Dans UNIX, toute unité d'E/S est désignée par un fichier spécial (cf. ce que vous connaissez en C sur stdin, stdout, stderr). Il existe deux modes d'échange pour l'E/S :

- **bloc** : les mémoires de masse (disque, disquettes) échangent des blocs d'information avec la mémoire centrale en utilisant des tampons

- **caractère** : les terminaux, les imprimantes et les réseaux échangent des caractères avec la mémoire centrale, sans tampon.

On peut rediriger les E/S standard :

< pour redéfinir stdin

> pour redéfinir stdout

>> idem, mais les données sont ajoutées en fin de fichier

La redirection s'applique à la dernière commande (parenthèser pour une redirection portant sur un groupe de commandes. La redirection de **stderr** nécessite d'utiliser son descripteur de valeur 2 :

commande 2 > fichier

echo "bonjour" > &2 redirige le message sur la sortie standard d'erreurs

commande > test 2 > &1 redirige la sortie vers le fichier test et la sortie standard des erreurs vers la sortie standard

Pour rediriger stdin pendant toute la session d'exécution de *exec*, on utilisera :

exec < fichier. De même exec > fichier.

Il est possible de transmettre des arguments à *main* () d'un exécutable C lors du lancement de son exécution grâce aux deux arguments : **main (int argc, char **argv)** dans cet ordre où :

argc (arguments count) est le compteur d'arguments, nom du programme exécutable inclus

argv (arguments vector) est un vecteur de pointeurs sur chaînes de caractères qui contiennent les arguments de la ligne de commande, à raison de un par chaîne ; argv [0] pointe sur le nom de l'exécutable et argv [argc] vaut NULL.

Exemple : tp1.e toto titi

argc vaut 3, argv[0] pointe sur "tp1.e" ; argv[1] pointe sur "toto" ; argv[2] pointe sur "titi"

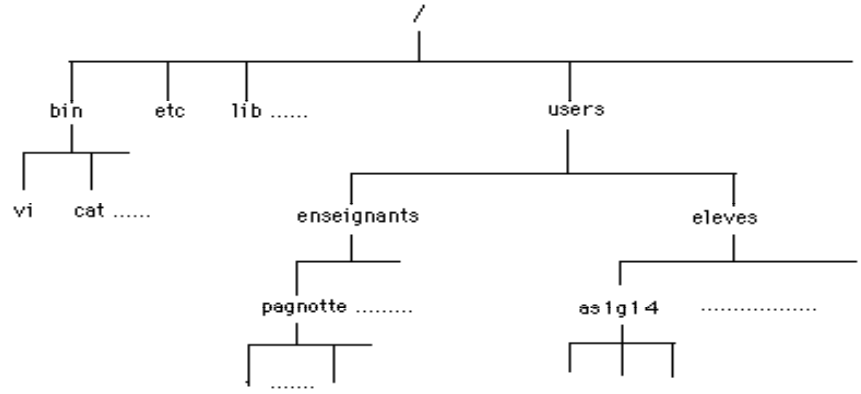
1.3 Utilitaires standards

UNIX dispose d'**utilitaires standards** nombreux parmi lesquels (voir annexe pour certains) :

- compilateurs (notamment C) avec son vérificateur **lint**, ses débogueurs **adb**, **sdb** et **dbx** et son archiveur de bibliothèques **ar**
- gestionnaire d'applications **make** et **sccs**
- outils de génération de compilateurs **lex** et **yacc**

éditeurs de texte / manipulateurs de documents **ed, sed, vi, emacs, nroff, troff**

2. LE SYSTEME DE GESTION DE FICHIERS (SGF)



Le S.G.F. gère une structure d'arbre :
- la racine est désignée par /
- les nœuds sont les répertoires non vides
- les feuilles sont les répertoires "vides" et les fichiers.

Les chemins sont décrits avec le séparateur /

Exemple : */users/eleves/m-dupont96* est une référence absolue (commence par /)
source/tp1.c est une référence relative (par rapport au répertoire courant), car ne commence pas par /

Un fichier est une suite d'octets, généralement non structurée.
Les noms de fichiers et répertoires sont limités à 14 caractères (/ est interdit et - ne peut être le premier caractère).

A chaque instant, l'utilisateur accède à un *répertoire courant* dans cette arborescence. Il se déplace avec la commande **cd**.

2.1 Les i-nodes

A tout fichier est attaché un nœud d'index, ou **i-node** contenant des informations sur ce fichier. C'est une structure de quelques dizaines d'octets décrite dans **/usr/include/sys/inode.h** qui contient généralement les champs suivants :

- le type (fichier ordinaire, spécial, catalogue ,...),
- les protections,
- les dates de création, accès et mise à jour (en s. écoulées depuis le 1/1/1970),
- la taille du fichier en octets,
- le nombre de liens (**un lien** d'un fichier est un autre nom de ce fichier),
- les éléments d'identification du propriétaire et de son groupe,
- l'adresse physique d'implantation sur disque (cf. chapitre 7) sous forme de 13 adresses de blocs disques

Bon exemple : Braquelaire p. 370-378.

L'i-node ne contient pas de "nom de fichier". La désignation d'un fichier se fait par l'intermédiaire du répertoire dans lequel est stocké le numéro d'i-node (entier naturel sur 2 octets) et le "nom

relatif" sur 14 octets. Ainsi, tout fichier est référencé de manière unique par son ***numéro d'i-node*** , même si son "nom relatif" peut être non unique dans l'arborescence.

L'i-node de n° 0 n'est jamais utilisée. L'i-node de n° 1 permet la gestion des blocs défectueux. La racine de l'arborescence possède l'i-node de n° 2.

Dans chaque répertoire,'•' est un lien au répertoire lui-même (dans le répertoire, en regard de ce "nom", figure le n° d'i-node du répertoire).De même, '••' est un lien au répertoire père (dans le répertoire, en regard de ce nom, figure le n° d'i-node du répertoire père). Un répertoire UNIX n'est donc jamais totalement vide.

Les i-nodes des fichiers d'un disque sont implantés par numéro croissant au début du disque (*voir 2.4*), ce qui en facilite beaucoup l'accès. Les i-nodes des fichiers ouverts sont copiés en mémoire centrale dans la **table des i-nodes**.

2.2 Les différents types de fichiers

On distingue :
- les fichiers ordinaires : données, programmes, textes

- les fichiers spéciaux : ils correspondent à des ressources. Ils n'ont pas de taille. Les opérations de lecture/écriture sur ces fichiers activent les dispositifs physiques associés. On distingue :

  - **character devices :** fichiers spéciaux en mode caractère (par exemple : terminaux)
- **block devices :** fichiers spéciaux en mode bloc (par exemple : disques)
  un bloc comprend 512 octets, 1024 ou davantage.

- les fichiers répertoires contiennent des informations sur d'autres fichiers et permettent la structuration arborescente

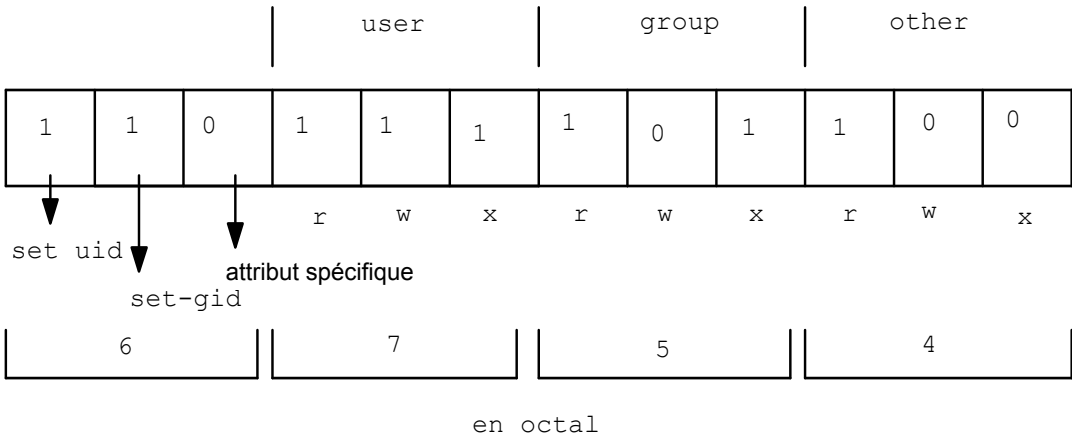
- les fichiers de liens symboliques : les fichiers qui sont des tubes nommés, les fichiers qui sont des prises réseaux.

2.3 Les protections des fichiers

A la création d'un compte d'**utilisateur**, l'administrateur affecte celui-ci à un **groupe**. Chaque groupe possède un nom. Un utilisateur peut changer de groupe par la commande **newgrp**. On peut donner des droits d'accès particuliers à certains fichiers pour les membres d'un même groupe.

Au total, il existe 3 types d'accès : propriétaire (**u** = user), groupe (**g** = group), autres (**o** = other). L'ensemble est désigné par **a** (all).

L'expression des droits nécessite 12 bits :



Pour un fichier, **r** = droit de lecture, **w** = droit d'écriture, **x** = droit d'exécuter (1 = oui, 0 = non)

Pour un répertoire,

r :on peut consulter la liste des fichiers qui y sont contenus
w : on peut créer ou effacer des fichiers à l'intérieur
x : on peut utiliser ce répertoire en argument d'une commande et s'y positionner

Ces 9 bits sont précédés par 3 autres bits pour compléter la description des protections :

- **le bit "set-uid" :** quand il vaut 1 pour un fichier exécutable, le processus d'exécution a les droits du propriétaire du fichier et non de l'utilisateur qui le lance

- **le bit "set-gid" :** même rôle, mais relativement au groupe

- **l'attribut spécifique :** outre la valeur "-" pour les fichiers ordinaires, il peut prendre la valeur :

d	fichier répertoire
c	fichier spécial en mode caractère (cas des terminaux)
b	fichier spécial en mode bloc (cas de lecteurs de disques ou de bandes)
s	bit de collage ou "sticky-bit" : il maintient le fichier en zone de recouvrement (swap) à partir de laquelle le chargement est plus rapide

La commande **ls** avec l'option **l** permet d'afficher les protections de fichiers. La commande **chmod** permet de modifier les protections d'un fichier.

Exemple : *d rwx rwx - - - 139 pagnotte profess 352 Nov 25 1999 tp*
tp est un répertoire (d)
Son propriétaire est *pagnotte*, du groupe *profess*
les protections rwx rwx - - - sont à interpréter selon les indications ci-dessus

La commande **ls - ias** permet l'affichage succinct des numéros d'i-nodes du répertoire.

2.4 SGF mono- ou multi-volumes

Un SGF UNIX peut être subdivisé en plusieurs disques logiques appelés "**filesystem**" ou "volume" (cas d'un seul disque partitionné, cas de plusieurs disques sur la même machine ou cas d'un système

de fichiers réparti sur plusieurs machines). Chaque *filesystem* comporte une arborescence de fichiers à partir de sa propre racine. L'un d'eux est appelé "**root filesystem**" ; il fédère les autres arbres. Cet "accrochage" est réalisé par l'administrateur du système grâce à la commande **/etc/mount** : on dit qu'une arborescence est *montée* sur le SGF.Un n° d'i-node peut alors ne plus être unique : il doit être obligatoirement associé au n° du volume.

Le système gère une table des volumes "montés" avec deux copies : l'une en mémoire centrale, l'autre sur disque (fichier */etc/mttab* sur UNIX System V).

L'ensemble des informations sur disque associées à un arbre est divisée en 4 zones :

- **zone 0** : pas utilisée par le SGF, mais par UNIX lui-même. C'est le **bootstrap**, contenant la procédure d'initialisation du système, de montage/démontage des volumes.

- **zone 1** : elle est appelée **superbloc**. Elle contient des informations sur le S.G.F. :
  - caractéristiques du SGF (nom du volume, nom et type du SGF, état du SGF pour savoir si le système a bien été arrêté normalement, date de mise à jour, taille en blocs du SGF, etc...)
  - caractéristiques des blocs libres (nombre, liste partielle, index vers le tableau des blocs libres, verrou d'accès aux blocs libres, etc...). Le superbloc est défini dans **/usr/include/sys/fs/s5param.h**
- **zone 2** : appelée **i-list** (liste d'index). Elle contient les i-nodes du volume.
  Pour augmenter la performance, le superbloc et la i-list sont copiés en mémoire et sauvés périodiquement sur disque.

- **zone 3** : contient les fichiers proprement dits.

Le *root filesytem* contient un certain nombre de répertoires standards :

/bin,/usr/bin utilitaires standards
/dev fichiers spéciaux pour périphériques (cf. ci-dessous)
/etc commandes, fichiers de maintenance/administration du système
/lib et /usr/lib bibliothèques standards
/unix le noyau du système
/usr/adm les fichiers de comptabilité
/tmp, /usr/tmp fichiers temporaires.
/usr/include fichiers en-têtes, d'extension .h
/usr/src les sources du système

L'adresse de sa racine figure "en dur" dans le noyau du système. Cette référence servira à construire la totalité de l'arbre des i-nodes, à contrôler les appels système de recherche, de création et de modification des i-nodes.

La *variable d'environnement* **HOME** mémorise le chemin du répertoire courant de l'utilisateur.

3. LES COMMANDES

Il existe de nombreuses commandes de base décrites dans la section 1 du Manuel de référence ou accessibles par la commande

man nom_de_commande

On trouvera ci-après les principales . Leur syntaxe n'est pas homogène. Elle est souvent de la forme :

nom [-options] [arguments]

Toute commande donne une **valeur de retour** nulle si le résultat est sans erreur ou vrai, non nulle sinon.

Un **tube** | permet de relier la sortie standard d'une commande à l'entrée standard de la suivante. Un tube retourne la valeur de retour de sa dernière commande
exemple :

```
ls | wc
ls | grep  pa | wc - l
#  on recherche les lignes du  répertoire contenant pa et  on affiche leur nombre)
who | tee  /tmp/g
# les informations transmises à la sortie standard le sont aussi au fichier cité, ici  /tmp/g
```

Une **liste** est une séquence de commandes ou de tubes séparés par des ; ou && ou || ou & et terminée par ; ou & Ainsi :

liste1 ; liste2 produit une exécution séquentielle des listes.

liste1 **&&** *liste2* : liste2 est exécutée si la valeur de retour de liste1 est VRAI (0)

liste1 || *liste2* : liste2 est exécutée si la valeur de retour de liste1 n'est pas VRAI (1)

Une liste entre des ( ) est exécutée dans un nouvel environnement, c'est à dire un processus shell fils créé à cet effet

Substitution de commande : **$(commande)**

```
$ echo la date est $(date)
la date est Mon Oct 14 21:32  MET 1999
```

Les alias : un alias est un surnom de commande

alias [-x] [nom[=valeur]]
-x : l'alias est exporté (réutilisable)
aucun **nom** : la liste des alias est visualisée
pas de **valeur** : la valeur de l'alias désigné est visualisée
Exemple : $ alias la="ls -ia"
Pour supprimer un alias, **unalias nom**

alias permet d'abréger des noms de commandes longs ou fréquemment utilisés

at **at** *18 mar 15 fic1* lance l'exécution de fic1 le 15 mars à 18 heures pour les utilisateurs répertoriés dans **/usr/lib/cron/at.allow** et non répertoriés dans **/usr/lib/cron/at.deny**

basename *chaîne [suffixe]*

basename */usr/include/sys/sgtty.h* affiche sgtty.h
basename */usr/include/sys/sgtty.h .h* affiche sgtty

cal *[mois] année* affiche le calendrier de l'*année*, ou seulement du *mois*.

cancel *ident_req* annule la requête d'impression *ident_req* obtenue par la commande **lp**

cat **cat** *fic1 fic2 > fic3* concatène fic1 à fic2 dans fic3 (*cat = catenate and print*)

cat *fic1* concatène fic1 à rien et redirige par défaut sur la sortie standard, donc affiche le contenu de fic1
cat > *fic2* concatène l'entrée standard à rien et la redirige sur fic3, donc saisit dans fic3 [ré]initialisé ce qui est frappé au clavier, jusqu'à la frappe de <EOF> (CTRL-D par défaut).

cd *dir* (*change directory*) change le répertoire courant pour *dir.* En l'absence d'argument spécifié,
dir = HOME

chgrp *group filenames* change le groupe des fichiers *filenames* pour *group*.
L'option **-R** effectue un changement récursif sur toute une arborescence de racine donnée

chmod *mode filename* affecte la protection *mode* à *filename* (cf. ci-dessus).
chmod *544 fic1* affecte la protection r-xr--r-- à *fic1*
chmod *g+x fic1* transforme cette protection en r-xr-xr--
chmod *a+w fic1* donne à tout le monde le droit d'écriture dans *fic1*
chmod *u-w fic1* retire ce droit au propriétaire.

chown *owner filenames* change le propriétaire des fichiers *filenames* pour *owner*.
L'option **-R** effectue un changement récursif sur toute une arborescence de racine donnée

cmp *filename1 filename2* donne le n° de l'octet et de la ligne où se produit la première différence entre *filename1* et *filename2*.

cp **cp** *filename1 filename 2* copie *filename1* sur *filename2*
cp -r *directory1 directory2* copie récursivement *directory1* avec son sous-arbre dans *directory2*, en le créant s'il n'existe pas.
cp *filenames directory* copie les *filenames* dans *directory*.
-i demande confirmation
-s produit simplement un code de retour (0 : identiques, 2 : différents, 3 : erreur)

crontab Le processus système **cron**, créé à la génération du système, gère toutes les tâches périodiques, y compris de l'arrivée du courrier dans les boîtes aux lettres. La commande **crontab** permet à un utilisateur de rajouter, de lister ou de retirer un fichier de tâches périodiques (batch) :
crontab *filename* ajoute le *filename* de tâches périodiques
crontab *-l* liste le fichier de tâches périodiques de l'utilisateur
crontab *-r* retire le fichier de tâches périodiques de l'utilisateur

Un fichier de tâches périodiques comprend une ligne par tâche. Chaque ligne comprend 6 champs séparés par des blancs ou des TAB , * spécifiant n'importe quelle valeur :
les minutes (0-59) de l'heure de lancement,
l'heure (0-23) de la date de lancement,
le jour de lancement (ou les jours de lancement) dans le mois (0-31),
le mois de lancement
le jour de lancement dans la semaine (0-6, 0=dimanche),
le texte de la commande à lancer
*0 14 * * 1-5 fic1* exécutera *fic1* chaque semaine du lundi au samedi à 14 h.
*0 0 1,15 * 1 fic2* exécutera *fic2* chaque lundi, et chaque 1er et 15 de chaque mois à 0 h.

crypt *chaîne* crypte l'entrée standard avec la clé chaîne ou la décrypte

cut permet d'extraire des caractères ou des champs de chaque ligne d'un fichier.

ls|cut -c*1,4,7* extrait les caractères 1, 4 et 7 de la sortie de ls
ls|cut -c*1-3,8* extrait les caractères 1 à 3, et 8 de la sortie de ls
l'option **-f** permet de spécifier des champs au lieu de caractères
l'option **-d***c* permet de respécifier le délimiteur de champ
(TAB par défaut, nouvelle valeur *c*)

date affiche la date

du **du** affiche le nombre de blocs (512 ou 1024 octets) occupés par chaque sous-arbre du répertoire courant
du -a affiche cette information pour tout fichier, tout répertoire du répertoire courant.
du -s affiche seulement le total

echo *chaîne_de_caractères* affiche *chaîne_de _caractères* sur l'écran.
echo "Votre nom \c" le curseur ne va pas à la ligne

expr interprète les arguments pour des calculs numériques. Voir exemple plus loin

file *filename* affiche la nature présumée du contenu de *filename*

find *pathname_list expression* parcourt récursivement le sous-arbre dont le chemin d'accès est *pathname_list* en recherchant les fichiers satisfaisant *expression*.
find */ -name fic1 -print* recherche *fic1* dans toute l'arborescence.
find */ -user gr1605 -print* affiche la liste des fichiers appartenant à gr1605
find *source -perm 777 -type f -exec rm {}\;* recherche les fichiers ordinaires (*-type f*; b désignerait un fichier de type bloc, c un fichier de type caractère et d un répertoire) dans le sous-répertoire *source*, les efface (après -exec, on peut utiliser n'importe quelle commande en la faisant suivre de "\;"). Les accolades {} remplacent le nom du fichier trouvé

grep *[-civw] pattern filenames*
"globally find regular expressions and print". Les *pattern* sont des expressions régulières de **ed** ou **sed**. Les lignes satisfaisant à ces critères sont affichées. Voir exemple ci-dessus.
Les options :
-c (count) compte les lignes sans les afficher
-i considère les majuscules et minuscules de façon identique
-v (reverse) cherche les lignes ne correspondant pas au critère
-w (word) cherche *pattern* comme un mot isolé

head *[-n] filename* affiche les *n* premiers caractères de *filename* (10 par défaut)

id affiche le nom et le n° (UID) de l'utilisateur, son nom et son n° de groupe (GID)

ln *filename1 filename2* crée un nouveau lien de nom *filename2* pour le fichier *filename1*
Il n'y a ni copie du fichier, ni création d'un nouvel i-node, mais simplement augmentation du compteur de référence du fichier dans un répertoire, c'est à dire ajout d'un couple (n°, référence)

logname affiche l'identification de l'utilisateur (valeur de la variable d'environnement LOGNAME)

lp *filename* ajoute la requête d'impression de *filename* à la queue du spoule

lpstat affiche des informations sur l'imprimante et son spoule

ls *[-alR] filename* affiche le contenu du répertoire *filename*.

Sans argument, le répertoire courant est traité. Les principales options :
-a permet l'affichage des noms de fichiers commençant par .
-l format long. Sont affichés : nombre de liens, propriétaire, taille en octets, date de dernière modification, mode (d = répertoire, b = fichier bloc, c = fichier caractère, l = lien symbolique, p = tube nommé, - = fichier ordinaire)
-i affiche les numéros d'i-nodes
-R récursif, génère la liste du sous-arbre tout entier

mkdir *dir* crée un nouveau répertoire.
L'option **- m** *droits* crée un nouveau répertoire avec les *droits* spécifiés

more *filename* affiche le contenu du fichier texte *filename*, un écran complet à la fois. Si l'on frappe :

ESPACE une ligne supplémentaire est affichée
RETURN un nouvel écran est affiché
b l'écran précédent est affiché
h affiche l'aide (liste complète des options)
q on quitte more

mv **mv** *filename1 filename2* change le nom (lien) *filename1* en *filename2*.
mv *directory1 directory2* change le nom *directory1* en *directory2* si *directory2* n'existe pas déjà et si *directory1* et *directory2* ont le même père.
mv *filenames directory* crée un lien entre les *filenames* et le *directory*

paste *filename1 filename2* concatène les lignes de même numéro de *filename1* et de *filename2*

pwd (*print working directory*) affiche la référence absolue du répertoire courant

rm *[-ri] filenames* efface les liens des *filenames* dans le répertoire courant, si le droit d'écriture dans le répertoire existe.
Le contenu d'un fichier qui n'a plus de lien est perdu. Options :
-i interactif. Demande une confirmation pour chaque fichier.
-r récursif pour tout le sous-arbre, si l'argument est un répertoire

rmdir *dir* supprime le répertoire *dir* s'il est vide
-i avec confirmation

rpl *chaine 1 chaine 2 < fic1 > fic2*
remplace toutes les occurences de chaine1 par chaine2 dans fic1 et émet dans fic2
ex.: rpl " IT " "Italie" < films.cine > films.tele

sort *[options] [+n1 -n2] filename1 [-o filename2]*
trie, selon l'ordre lexicographique du code, les lignes de *filename1*, affiche le résultat ou le redirige sur *filename2*. Options :
-b on ignore les espaces de tête
-d seuls les chiffres, lettres et espaces sont significatifs dans les comparaisons,
-f majuscules et minuscules sont confondues,
-i les caractères dont le code ASCII est extérieur à l'intervalle [32,126] sont ignorés dans les comparaisons,
-n les débuts de lignes numériques sont triés numériquement,
-t*c* définit comme *c* le séparateur de champs au lieu de TAB

Les options +n1 -n2 délimitent la clé : du champ n° n1 inclus au champ n° n2 exclu. La numérotation des champs commence à 0.

sort permet aussi le tri sur des portions de champs, sur plusieurs clés, ou des fusions de fichiers (voir Nebut p. 135-137)

split -*number infile outfile* fragmente le fichier *infile* en fichiers de *number* lignes (1000 par défaut), de noms *outfile*aa, *outfile*ab, etc... (*outfile* = x s'il n'est pas spécifié)

stty [-a] *options*

L'utilisation des terminaux fait appel à 10 caractères particuliers, désignés par un nom symbolique :
<NEWLINE> : dans l'utilisation normale bufferisée, le processeur d'E/S reçoit un caractère du clavier, envoie un écho à l'écran. Mais il n'envoie les caractères au processeur central qu'après réception de <NEWLINE>. Par défaut le caractère NL du code.
<ERASE> : effacement du caractère précédent. Par défaut :# ou le caractère BS du code ou CTRL-H
<KILL> : effacement de la ligne courante. Par défaut @ ou CTRL-U
<EOL> : marqueur de fin de ligne. Par défaut \0 ou CTRL-@
<EOF> : marqueur de fin de fichier s'il est en début de ligne. Par défaut le caractère EOT du code ou CTRL-D.
<STOP> : interrompt le flux de caractères vers la sortie. Par défaut le caractère DC3 du code ou CTRL-S.
<START> : relance le flux de caractères vers la sortie. Par défaut le caractère DC1 du code ou CTRL-Q.
<INTR> : émission du signal SIGINT. Par défaut le caractère DEL du code ou CTRL-C.
<QUIT> : émission du signal SIGQUIT. Par défaut CTRL-\ ou CTRL-Z

stty *caractère_de_contrôle c* donne la nouvelle valeur *c* au *caractère_de_contrôle* avec la syntaxe '^D' pour CTRL-D et '^?' pour DEL

stty -a affiche les options actuelles de la sortie standard.
stty sane rétablit les valeurs par défaut

tail *[options] filename* affiche la fin de *filename*.

tail -*14c fic1* affiche *fic1* à partir du 14ème caractère avant la fin
Au lieu de c, on peut utiliser l (ligne) ou b (bloc)
tail +*25l fic2* affiche la fin de fic2 à partir de la 25ème ligne

tee *[-a] filename* l'affichage de la sortie standard est en même temps dirigé sur *filename*
L'option **-a** signifie >>

test *exp* ou **[** *exp* **]** retourne 0 si la valeur de exp est vrai, 1 sinon. exp peut être :
-a<filename> : vrai si filename existe
-r<filename> : vrai si filename existe et peut être lu,
-w<filename> : vrai si filename existe et peut être écrit,
-x<filename> : vrai si filename existe et peut être exécuté,
-f<filename> : vrai si filename existe et est un fichier ordinaire,
-d<filename> : vrai si filename existe et est un catalogue,
-b<filename> : vrai si filename existe et est de type bloc,
-c<filename> : vrai si filename existe et est de type caractère,
-s<filename> : vrai si filename existe et est de taille non nulle,
-z<chaîne> : vrai si la chaîne est de longueur nulle,
-n<chaîne> : vrai si la chaîne est de longueur non nulle,
<chaîne1> = <chaîne2> : vrai si les deux chaînes sont égales,
<chaîne1> != <chaîne2> : vrai si les deux chaînes sont différentes,
<valeur1> -eq <valeur2> : vrai si les deux valeurs sont égales,

autres prédicats possibles : -ne (≠),-gt (>),-lt (<),-ge (≥), -le (≤)
On peut composer les conditions avec les opérateurs booléens **-a** ou && (ET), **-o** ou || (OU), **!** (NON)

tr *string1 string2* l'entrée standard est copiée sur la sortie standard, mais un caractère ayant une occurence dans *string1* est remplacé par le caractère de même rang dans *string2*. Avec l'option **-d**, les caractères en entrée, présents dans *string1*, sont supprimés en sortie.
tr *[a-z][A-Z] < fic1* affiche fic1 en remplaçant les minuscules par des majuscules.
tr *-d '\012' < fic1* affiche fic1 en supprimant (option **-d**) les caractères de code ASCII 12 en octal, donc 10 en décimal, donc en supprimant les sauts de ligne
tr *-s "a-z" "A-Z" < fic1* réduit toute chaîne de caractères identiques en un seul
tr *abc xyz < fic1* traduit a en x, b en y, c en z

tty affiche le nom du terminal associé à la sortie standard

wc *[-lwc] filename*
wc *fic1* affiche le nombre de lignes, mots, et caractères de fic1
wc *-lc fic1* affichera le nombre de lignes et de caractères de fic1
L'option **-w** correspond aux mots, **-l** aux lignes, **-c** aux caractères

whereis *filename* affiche le chemin d'accès à *filename*.

who affiche la liste des utilisateurs connectés

xd *filename* affiche *filename* en hexadécimal et "en clair".
ls|xd affichera les noms complets des fichiers du répertoire courant, y compris les caractères non affichables qui pourraient être présents dans les noms de fichiers

4. LE SHELL

Le shell (littéralement coquille autour du noyau d'UNIX) est l'interpréteur de commandes d'UNIX. Tout à la fois :
- il exécute en mode interactif les commandes émises par l'utilisateur,
- il propose un langage de programmation interprété permettant de générer de nouvelles commandes ou procédures cataloguées ("scripts shell"), C étant le langage le plus adapté pour construire les nouvelles commandes que le shell ne peut traduire.

Le shell ne fait pas partie du noyau d'UNIX et n'est pas résident en mémoire principale. Ainsi, on peut disposer facilement de plusieurs interpréteurs de commandes : Bourne-shell, C-shell, Korn-shell, ...

Dans tout shell d'UNIX, un **blanc** est un espace ou un caractère de tabulation. Un **séparateur** est un blanc ou un caractère NL. Un **mot** est une chaîne de caractères entre deux séparateurs. Tout ce qui est entre deux ' est considéré comme un seul mot.

4.1 Les variables shell

leur ***nom*** : une suite de caractères lettres, chiffres et _ commençant par une lettre ou _
ex.: a=paul chemin=/users/eleves/m-durand99

leur ***valeur*** : $a ou ${a} désigne la valeur de la variable a et ${a}c désigne la valeur de a suivie de c.

ex.: a=paul
b=chou
echo $a $b

On utilise trois ***caractères génériques*** :

* toute sous-chaîne, même vide,
? tout caractère,
[...] tous les caractères d'un intervalle.

Toute fin de ligne commençant par # est un commentaire

ex.: ls - l *.c # édite les noms de fichiers d'extension .c

ls -l [a-e]* # édite les noms de fichiers commençant par a à e.

métacaractères : < * ? | & , \ ont un sens spécial.

ex.: a="bijou * caillou "
b=chou ; c=caillou ; r="$a $b";echo $r

Précédés de \, les métacaractères perdent leur signification particulière

ex.: echo * ; echo \\ echo abc***d

les ***délimiteurs de chaînes*** :

dans une chaîne délimitée par des " , les caractères $, \, ', ` sont des caractères spéciaux.
dans une telle chaîne, un caractère " doit être précédé de \
dans une chaîne délimitée par des ' , tous les caractères perdent leur aspect spécial

types de variables : la commande **typeset [-filrux] [variable [= valeur]]** permet de typer une variable et/ou de lui affecter une valeur:

-f : variable est une fonction
-i : variable est de type entier
-l : majuscules transformées en minuscules
-r : variable accessible seulement en lecture
-u : minuscules transformées en majuscules
-x : variable exportée

Exemples :

```
$ typeset -r v=abc
$ echo $v
abc
$ typeset v=hier
/bin/ksh  v is read only
$ w=3+5
$ echo $w
3+5
$  typeset -i w
$ echo $w
8
$ typeset +r  v      (déprotection de v)
$ v=123
$ echo $v
123
```

règles de substitution : elles utilisent les { }

${#variable} : longueur de la variable

```
$ X=abc
$ print ${#X}
3
```

${variable :- mot} : valeur de *variable* si elle est définie et non nulle, sinon valeur de *mot*

```
$ X=abc
$ print ${X-:cde}
abc
$ unset X
$ print ${X:-cde}
cde
```

${variable := mot} : valeur de *variable* si elle est définie et non nulle, sinon *variable* prend la de *mot*

${variable :? mot} : valeur de *variable* si elle est définie et non nulle, sinon valeur de *mot* et sortie

${variable :+ mot} : valeur de mot si *variable* est définie et non nulle, sinon pas de substitution

```
$ Y=abc
$ print ${Y=+def}
def
$ unset Y
$ print ${Y:+def}
$ .....
```

${variable#pattern}** ou **${variable##pattern} : valeur de *variable* à laquelle on extrait en partant de la gauche la plus petite partie (ou la plus grande) correspondant au *pattern*

```
$ X=abcabcabc
$ print ${X#abc*}
abcabc
 $ print ${X##abc*}
$  (rien)
```

${variable % pattern}** ou **${variable %% pattern} : idem à ci-dessus, mais en partant de la droite

les **tableaux** de variables : en assignant à X [1] une valeur, la variable X est *transformée* en tableau

```
$ X=A
$ X[1]=B     (alors, X [0] vaut A )
```

On peut **affecter** ainsi :

variable[0]=valeur0 ; variable[1]=valeur1 ; .... ; variable[n]=valeurn
ou encore : ***set*** -A variable valeur0 valeur1 ..... valeurn
ou encore :
typeset variable[0]=valeur0 variable[1]=valeur1...variable[n]=valeurn

On peut **réaffecter** ainsi : ***set*** -A variable valeur0 valeur1 ..... valeurn
ou encore en ne donnant que certaines valeurs :

```
$ set -A  X  one  two  three  d  e  f
$ print ${X[*]}
```

```
one  two  three  d  e  f
$ set -A  X  a  b  c
$ print ${X[*]}
a  b  c  d  e  f
$ print ${X[1]}
b
$ print ${#X[*]}
6      (nombre d'éléments du tableau)
```

variables prédéfinies gérées automatiquement par le shell :

$# nombre de paramètres d'une commande, ceux-ci étant désignés par $1 à $9
($0 le nom de la commande elle-même).

$* la liste des paramètres $1 $2 ...

$$ le numéro du processus en cours (très utile dans la suite)

$! le n° du dernier processus lancé en arrière plan

$? le code de retour de la dernière commande exécutée

variables d'environnement prédéfinies :

HOME l'argument par défaut pour la commande cd, c'est à dire le chemin correspondant au sous-répertoire à l'ouverture de la session
LOGNAME le nom de l'utilisateur (sous SYSTEM V) ou ***USER*** (sous système BSD)
PATH la liste des répertoires à chercher pour exécuter une commande
PS1 le prompt principal
PS2 le prompt secondaire
IFS la liste des séparateurs de mots pour le shell (inter fields separators)
SHELL le shell courant
TERM le type de terminal
TZ le fuseau horaire (time zone)

Le fichier **.profile** permet de donner des valeurs non standards aux *variables d'environnement* et d'exécuter des commandes comme :

uname -a (nom de la machine)
umask (masque des autorisations d'accès au fichier par défaut)
ulimit (taille maximale des fichiers)
banner (affichage d'une bannière en grandes lettres).

La commande **set** permet d'afficher la valeur des *variables d'environnement*

Exemple d'un fichier *.profile* :

```
HOME=/home/$LOGNAME
PATH=$PATH:$HOME/mes_tp
PS1="OK ?"
banner  HELLO WORLD
export  HOME PATH PS1
```

4.2 Les shell scripts

L'interpréteur de commandes est un fichier exécutable se trouvant dans /bin. Le shell est donc lui-même une commande.

Ainsi, pour disposer d'une nouvelle procédure cataloguée ("***script shell***"), on saisit dans un fichier (par exemple ici *fic*) une liste de commandes avec un éditeur

Pour exécuter ce fichier, on a deux solutions:

sh fic [arg] ou mieux **chmod u+x fic**
fic [arg]

Le langage de commandes du shell dispose des structures classiques if, for, while, until.

4.3 Boucle for

```
for var  [in  mot1 [mot2 ...]]
            do  liste_de_commandes
            done
```

var prend les valeurs *mot1, mot2*, ... et exécute la liste de commandes pour chaque valeur

```
ex.:   for i do grep   $i   /users/eleves/m-durand99/telno
           done

# on range ce script dans le fichier TEL qu'on rend exécutable
# ici, la liste des valeurs est vide; par défaut,
# elle est prise égale à $*, la liste de paramètres de commande

$ TEL  paul  bernard
# affichera les lignes de /users/eleves/m-durand99/telno contenant paul  ou bernard
```

4.4 Boucle while

```
while liste_de_commandes
    do liste_de_commandes
    done
```

On boucle tant que la dernière commande de la liste renvoie 0 (vrai).

```
ex.:   while test $#  -ne 0            # tant qu'il y a des paramètres...
          do echo $1                   #...on affiche le premier
             shift                     # on décale les paramètres
          done
       # on range ce script dans le fichier ACTION qu'on rend exécutable

       ACTION  toto   titi        #  affiche toto titi
```

4.5 Boucle until

```
until  liste_de_commandes
            do liste_de_commandes
```

```
done
```

On boucle jusqu'à ce que la dernière commande de la liste renvoie 0 (vrai).

```
ex.:    until test -f  fic
         do sleep 50       #attendre 50 s.
          done
```

4.6 Choix if

```
if  liste_de_commandes
then  liste_de_commandes
        [elif  liste_de_commandes
           then  liste_de_commandes ]
           [else liste de commandes ]
fi
```

On exécute "then" si la dernière commande de la liste de "if" retourne 0 (vrai).

```
ex.:    if test -f  "$1"
           then echo fichier  $1 trouve
           else echo  pas de fichier de nom  $1
        fi
```

4.7 Sélection case

case vise à éviter les imbrications de if :

```
ex.:    # fichier ajout
         case $# in
           1) cat  >> $1;;
           2) cat  >> $2  <$1;;
          *) echo  "syntaxe : ajout [origine] destination";;
         esac

        ajout  fich      #ajoute en fin de fichier jusqu'à <CTRL-D>
        ajout  f1  f2    # f1 copié en fin de f2
```

On notera aussi les commandes **break** et **continue** (idem à C). C-Shell contient aussi **foreach**.

4.8 Sélection select

select est une commande intermédiaire entre case et for, utile pour la gestion des menus

```
ex.:    # fichier stest
        select i in Choix-A  Choix-B  Choix-C
        do
            if  [$i  = Choix-[A-C]]
            then
                print "Vous avez choisi  $REPLY : $i"
            else
```

```
            print "$REPLY : mauvais choix"
            continue
        fi
    done

    $ stest
    1) Choix-A
    2) Choix-B
    3) Choix-C
    #?  5
    5 : mauvais choix
    #?  3
    Vous avez choisi : 3
    #?  <RETURN>              # menu réaffiché
    .............
```

4.9 Utilisation du résultat d'une commande

On peut utiliser comme paramètre, ou affecter à une variable, la sortie standard d'une commande placée entre ` `

```
exemples  1 :   for i in `ls`
                    do echo "      "$i
                    done

          2 :   c=`pwd`    # voir commande pwd
                    ls $c      # affiche le catalogue de travail

          3 :   set  `date` ; echo $3 $2 $6
                    # set range dans les variables prédéfinies $1, $2, etc....
                    # il s'affiche jour, mois année
```

4.10 Autres commandes

read

```
echo "Votre nom ? \c"
read name
echo "Bonjour  $name, ca va ?"

print "this is a play again" | read X  Y  Z
print $X
this
print $Y
is
print  $Z
a play again
```

exec réalise des redirections d'E/S

```
exec 1 > std.out      # redirige la sortie standard vers le fichier std.out

exec < fic            # redirige l'entrée standard sur le fichier fic
.......
exec < /dev/tty       # rétablit l'entrée standard
```

```
exec 5<>fic.out
# le fichier fic.out est ouvert en lecture-écriture avec le descripteur 5
print  -u5  "Tout va bien"
cat  < &5                # lecture à partir du fichier de descripteur 5
exec 5 < &-              # fermeture du fichier de descripteur 5
```

let let "expression arithmétique" ou ((expression arithmétique))

exemples :

```
((X=-7))
((Y=-X+2))
echo $Y
```

toutes les opérations arithmétiques de C sont permises avec let

true retourne 0

false retourne 1

5. LE NOYAU D'UNIX

Le noyau est responsable de presque toutes les tâches de base qui sont exécutées par des procédures système :

- **initialisation du système** ,
- **gestion des ressources :** temps, mémoire primaire,

- **gestion des fichiers :** création, utilisation, disparition, gestion de la mémoire secondaire,

- **gestion de la mémoire :** gestion de la segmentation et de la pagination.
- **gestion des processus :** contrôle de la création, de la terminaison, de la synchronisation, du partage de temps (ordonnancement), de la communication entre processus,

- **gestion des E/S :** sélection des pilotes (drivers) pour contrôler l'E/S en fonction du n° majeur et du n° mineur du fichier spécial sélectionné,

- **gestion des communications :** réseaux locaux, X25, etc...

Pour assurer ces activités, le noyau dispose d'un certain nombre de tables :
- table des processus décrivant l'état des processus,
- table des fichiers ouverts,
- table des i-nodes en mémoire,
- pour chaque processus, une table des descripteurs de fichiers,
- table des volumes montés,
- table des textes (pour les exécutables partageables)

Le noyau réside en mémoire principale tant que le système est en fonctionnement.

BIBLIOGRAPHIE

S.BOURNE , Le système UNIX, Interéditions 1985

J.P. BRAQUELAIRE, Méthodologie de la programmation en Langage C, Masson, 1993
A.B. FONTAINE, Ph. HAMMES, UNIX Système V, Masson, 1993
B. KERNIGHAN et R. PIKE , L'environnement de programmation UNIX, Interéditions, 1985
J.L. NEBUT , UNIX pour l'utilisateur, Technip, 1990
J.M. RIFFLET , La programmation sous UNIX, Mac Graw-Hill, 1992
A. TANENBAUM, Systèmes d'exploitation, Prentice Hall, 1999

ANNEXE : QUELQUES UTILITAIRES

1. ar

ar permet de construire et de mettre à jour des bibliothèques utilisées par l'éditeur de liens **ld**.
On peut ainsi rassembler plusieurs modules objets en un seul fichier d'archive.

Syntaxe : **ar** *clé lib liste_ref* avec :

lib nom du fichier archive (la bibliothèque),
clé **d** supprimer la liste de l'archive
r si l'un des modules de la liste est dans l'archive, il sera remplacé; sinon, il sera créé
q ajouter de nouveaux modules à la fin de l'archive, sans examiner l'existant
t ou **p** lister le contenu de l'archive
x extraire des modules de l'archive
liste_ref liste de fichiers objets

2. awk

awk (du nom de ses créateurs, Aho, Weinberger et Kernighan), est un micro-langage de programmation, accessible par le shell d'UNIX, inspiré de C et interprété.

appel : awk 'programme' *nom_fichier* **ou** awk -f *refprog nom_fichier*
où ***refprog*** est le fichier contenant le programme
nom_fichier est le fichier de données pour le programme ou une liste de fichiers

Un programme est de la forme :

```
BEGIN {.....}
{ .....}
END { .....}
```

Les sections BEGIN et END sont facultatives.

Dans le corps du programme (section 2), on peut utiliser la syntaxe complexe de *grep* pour exprimer des expressions ("motifs").

identificateurs : comme en C, mais pas de déclarations. Les types flottant et chaîne de caractères sont implicites.

Des variables sont prédéfinies :
$0 : l'enregistrement courant du fichier
$1, $2, ... : ses champs
FS : séparateur de champs (SP et HT par défaut)
RS : séparateur d'enregistrements (LF par défaut)
OFS : séparateur de champs en sortie (SP par défaut)
ORS : séparateur d'enregistrements en sortie (LF par défaut)
NF : nombre de champs de $0
NR : numéro de l'enregistrement courant $0

opérateurs : + - * / % ++ - < > <= >= == != ! && ||

fonctions : *length* (ch) retourne la longueur de la chaîne *ch*
int (exp) convertit *exp* en entier par troncature
index (ch1,ch2) retourne la position de la première occurence de *ch2* dans *ch1* ou 0 s'il n'y en a pas
split (ch,t,c) éclate *ch* dans *t[1], t[2], ...* avec *c* comme séparateur de zones.
substr (ch,m,n) retourne la sous-chaîne de *ch* commençant en position *m* et de longueur au-plus *n*

instructions :
if else comme en C
while comme en C
for comme en C
for (<var> in <tableau>) <instruction>
break continue comme en C
exit passage à la section END, ou abandon si on s'y trouve
next passage à l'enregistrement suivant

On dispose aussi de la fonction *printf* de C et de *print* (cf. exemple).

On peut passer la valeur d'une variable du shell à une variable locale d'un programme awk:
read reponse
awk ..... x=$reponse ....

en supposant que *x=$reponse* est en tête de la liste ***nom_fichier*** décrite plus haut, précédée de -v

Un commentaire commence par # et finit à la fin de la ligne.

3. make

make est un outil puissant pour gérer les projets. Il permet la maintenance des fichiers objets et des fichiers exécutables en lançant les seules compilations indispensables après des mises à jour.

make utilise un fichier appelé *makefile* par défaut qui contient une description des liens de dépendance entre les fichiers et les actions à réaliser.
Exemple :

```
prog:fic1.o fic2.o
        cc -o prog fic1.o fic2.o
fic1.o:fic1.c menu.h
        cc -c fic1.c
fic2.o:fic2.c
        cc -c fic2.c
```

Il peut y avoir plusieurs points d'entrée (tels que *prog*) dans *makefile*.
Les lignes indentées sont précedées de TAB.

Syntaxe : **make** *prog*
make *-f file prog* si *file* est utilisé au lieu de *makefile*

4. sccs

sccs permet le contrôle des fichiers source et de leurs différentes versions, notamment extraire une copie d'une version à partir d'un fichier historique, verrouiller une version contre tout changement pour la protéger, éditer les différences entre les versions, etc... (Voir Nebut, pages 223-229).

5. sed (et ed)

sed est un ancien éditeur de texte non interactif reprenant la plupart des commandes du très ancien éditeur ligne **ed** (qui n'a plus d'intérêt pris isolément). La syntaxe d'appel est :

sed -f fc fa > fb
 # crée un nouveau fichier fb à partir d'un fichier fa en prenant les directives dans le fichier de commandes fc

ed offre essentiellement les fonctionnalités suivantes :

- impression :

p imprime (envoie dans stdout) tout le fichier courant
2 p imprime la ligne n° 2
2,5 p imprime les lignes n° 2 à 5
8,$ p imprime de la ligne n° 8 à la fin de fichier
- désigne la ligne précédente, . la ligne courante et + la ligne suivante

- remplacement :

La commande **s/text_anc/texte_nouveau/g** remplace les occurences de **texte_anc** par **texte_nouveau** dans tout (à cause de **g**) le fichier courant.
Le point représente "tout caractère". L'astérisque désigne 0 ou n occurences.
L'accent circonflexe désigne le début de ligne. $ désigne la fin de ligne, [...] désigne un intervalle et [^...] l'intervalle complémentaire, * désigne une répétition de caractères.
Exemples :

```
1,$s/./,/g          remplace tous les caractères par des virgules
1,$s/\./,/g         remplace tous les points par des virgules
s/x  *y/x  y/g      remplace tous les blancs entre x et y par un seul blanc
1,$s/^[0-9]*//g     supprime tous chiffres en début de ligne
v/^[0-9]*/d         détruit toutes les lignes ne commençant pas (v) par un nombre
g/eau/s/&/vin/      change dans tout le fichier les lignes contenant eau en vin
/sanglots/          recherche la 1ère occurence de sanglots vers l'avant du fichier
s/longs/&  longs/   remplace dans cette occurence sanglots par sanglots longs
s/.*/(&)/g          enferme entre parenthèses toute ligne du fichier
g?Jean?             recherche toutes les lignes contenant Jean vers le haut du fichier
sed –n  " /PAGNOTTE/p " fichier
sed  " s/PAGNOTTE/Pagnotte " fic > res
```

Le critère de recherche **^[^Uu]r..\.** désigne une ligne ne comportant ni (^) U, ni u en début de ligne, mais incluant un r suivi de 2 caractères quelconques et d'un point.

- destruction :

2d détruit la ligne 2
2,5d détruit les lignes 2 à 5
.,/123/d détruit toutes les lignes entre la ligne courante et la 1ère ligne contenant 123

- mouvement :

2,5 t 19 insère une copie des lignes 2 à 5 à partir de la ligne 19
2,5 m 19 déplace les lignes 2 à 5 pour les insérer à partir de la ligne 19
0r fic insère le fichier fic en début du fichier courant