CHAPITRE 2 :

GESTION DE FICHIERS SOUS LINUX

Objectifs spécifiques

- Connaître la définition d'un système d'exploitation
- Connaître le rôle d'un système d'exploitation
- Connaître l'historique et les caractéristiques de UNIX et LINUX

Eléments de contenu

I. Introduction

II. Système de gestion de fichiers

III. Désignation des partitions sous Linux

IV. Système de fichier

V. Arborescence

VI. Chemins relatifs et chemins absolus

VII. Types de fichiers

VIII. Catégories de fichiers

IX. Les i-nodes de fichiers

X. Commande Shell de manipulation de fichiers

Volume Horaire :

Cours : 4 heures 30 mn

TD : 1 heure 30 mn

2.1 Introduction

Le volume des données traitées par les applications informatiques atteignant plusieurs Méga et Giga octets, ces données ne peuvent pas être stockées dans la mémoire centrale. On souhaite également disposer d'un stockage à long terme qui ne disparaisse pas lorsqu'on éteint la machine. Le principe consiste à stocker ces données dans des mémoires secondaires sous forme de fichiers, c'est-à-dire de suites de blocs (la plus petite unité que le périphérique de stockage est capable de gérer). Le contenu de ces blocs, simple suite de chiffres binaires, peut être interprété selon le format de fichier comme des caractères, des nombres entiers ou flottants, des codes d'opérations machines, des adresses mémoires, etc... L'échange entre les deux types de mémoires se fait ensuite par transfert de blocs.

L'objectif du système de fichier est de permettre l'accès au contenu du fichier (l'ouverture du fichier, sa recopie à un second emplacement ou sa suppression) à partir de son chemin d'accès, formé d'un nom précédé d'une liste de répertoires imbriqués.

2.2 Système de gestion de fichiers

Un fichier est une collection logique d'information. Un système de fichiers est une collection de fichiers. Une des fonctions d'un SE est de masquer les spécificités des disques et des autres périphériques d'entrées/sorties et d'offrir au programmeur un modèle de manipulation des fichiers agréable et indépendant du matériel utilisé. Les appels système permettent de créer des fichiers, de les supprimer, de lire et d'écrire dans un fichier. Il faut également ouvrir un fichier avant de l'utiliser, le fermer ultérieurement. Les fichiers sont regroupés en répertoires arborescents; ils sont accessibles en énonçant leur chemin d'accès (chemin d'accès absolu à partir de la racine ou bien chemin d'accès relatif dans le cadre du répertoire de travail courant). Le SE gère également la protection des fichiers.

2.3 Désignation des partitions sous Linux

Un disque dur est généralement organisé de la façon suivante :

Figure 1 : Organisation d'un disque dur

Le Master Boot Record (MBR) est situé dans les premiers secteurs du disque. Le mode de partitionnement MBR est le mode historique des ordinateurs de type IBM PC-compatible et le plus répandu. Dans ce modèle, une zone de 512 octets appelée le *Master Boot Record* (MBR) est réservée en début de disque pour contenir l'information relative à un maximum de 4 partitions.

Le MBR est constitué de 2 parties :

- ⇨ La table de partition ;
- ⇨ Le programme d'amorçage qui charge le noyau du système d'exploitation.

Dans un disque dur, plusieurs types de partitions peuvent coexister. On peut mentionner les partitions principales, étendues et logiques.

- ⇨ Les partitions principales :
  - Sont au maximum de 4
  - Accepte tout type de système de fichiers.
- ⇨ Les partitions étendues :
  - Sont destinées à contenir des partitions logiques et non au système de fichiers,
  - Nécessitent au moins une partition principale.
- ⇨ Les partitions logiques :
  - Sont contenues dans une partition étendue,
  - Accepte tout type de systèmes de fichiers.

Sous Linux, le pointeur spécial /dev permet l'accès aux disques.

Pour dénommer les disques, un format est adopté : **/dev/XXYZ** où

- ✈ XX désigne le type de bus :

- hd pour les périphériques de type IDE
- sd pour les périphériques de type SATA
  - ✈ Y désigne la lettre de périphérique :
- "a" pour le maître de la nappe IDE primaire
- "b" pour l'esclave de la nappe IDE primaire
- "c" pour le maître de la nappe IDE secondaire
- "d" pour l'esclave de la nappe IDE secondaire
- Z désigne le numéro de la partition

Exemples :

- /dev/hda1 : partition 1 sur le 1er disque IDE
- /dev/sdb2 : partition 2 sur le 2ème disque SATA

2.4 Systèmes de fichiers

Les données sont normalement présentées à l'utilisateur et aux programmes selon une organisation structurée, sous la forme de répertoires et de fichiers. Pour pouvoir stocker ces données structurées sur un périphérique, il faut utiliser un format qui les représente sous la forme d'une succession de blocs de données : c'est ce qu'on appelle un système de fichiers.

On dit qu'un système de fichiers est journalisé s'il prévient les coupures lors de la sauvegarde de fichiers sur disque. Au lieu d'écrire immédiatement sur le disque dur les données à l'endroit exact où elles devraient être enregistrées, le système de fichiers journalisé écrit les données dans une autre partie du disque dur et note les changements nécessaire dans un journal, et ensuite, en arrière-plan, il repasse chacune des entrées du journal et termine le travail commencé ; lorsque la tâche est accomplie, il raye la tâche de la liste.

Sous Windows, les systèmes de fichiers disponibles sont :

- ✈ **FAT** : Développé par Microsoft, ce système de fichiers se rencontre moins fréquemment aujourd'hui. Il reste néanmoins utilisé sur les disquettes 3½ formatées sous Windows et devrait être utilisé sous Linux si une disquette doit aussi être lue sous Windows. Il est aussi utilisé par plusieurs constructeurs comme système de fichiers pour cartes mémoires (*memory sticks*), car, bien documenté, ce système de fichiers reste le plus universellement utilisé et accessible.
- ✈ **FAT32** : Ce système de fichiers, aussi créé par Microsoft, est une évolution de son prédécesseur. Depuis ses versions 2000 SP4 et XP, Windows ne peut pas formater (ou bloque volontairement le formatage) une partition en FAT32 d'une taille supérieure à 32 Go. Cette limitation ne s'applique pas sous Linux, de même qu'avec des versions antérieures de Windows. Une partition FAT32 d'une taille supérieure à 32 Go déjà formatée pourra être lue par Windows, peu importe sa version.

- **NTFS :** Ce système de fichiers a aussi été développé par Microsoft, et il reste très peu documenté. L'écriture depuis Linux sur ce système de fichiers est stable à l'aide du pilote **ntfs-3g**.
- **extFAt :** Ce système de fichiers a aussi été développé par Microsoft. L'écriture depuis Linux sur ce système de fichiers est stable à l'aide du pilote **exfat-fuse**

Linux utilise également FAT, FAT32, NTFS et extFat ainsi que plusieurs autres systèmes de fichiers, nous citons :

- **ext2fs :** Extended File System est le système de fichiers natif de Linux. Les versions 1 et 2 de ce système ne disposent pas de la journalisation. Ext2 peut tout de même s'avérer utile sur des disquettes 3½ et sur les autres périphériques dont l'espace de stockage est restreint, car aucun espace ne doit être réservé à un journal.
- **ext3fs :** ext3 est essentiellement ext2 avec la gestion de la journalisation. Il est possible de passer une partition formatée en ext2 vers le système de fichiers ext3 (et *vice versa*) sans formatage.
- **ext4fs :** ext4 est considéré par ses propres concepteurs comme une solution intermédiaire en attendant le vrai système de nouvelle génération que sera Btrfs.
- **ReiserFS :** Développé par Hans Reiser et la société Namesys, ReiserFS est reconnu particulièrement pour bien gérer les fichiers de moins de 4 ko. Un avantage du ReiserFS, par rapport à ext3, est qu'il ne nécessite pas une hiérarchisation aussi poussée: il s'avère intéressant pour le stockage de plusieurs fichiers temporaires provenant d'Internet. Par contre, ReiserFS n'est pas recommandé pour les ordinateurs portables, car le disque dur tourne en permanence, ce qui consomme beaucoup d'énergie.

2.5 Arborescence

Une arborescence est une organisation logique de fichiers sur un ou plusieurs systèmes de fichiers. Il s'agit d'une structure de données hiérarchique de type arbre. Le système qui gère les fichiers sous Linux est un peu déroutant au début, surtout quand on est habitué à celui de Windows. En effet, sous Windows, il y a en fait plusieurs racines.C:\est la racine du disque dur, E:\est la racine de votre lecteur CD (par exemple). Sous Linux, **il n'y a qu'une et une seule racine** : « / ». Il n'y a pas de lettre de lecteur car justement, Linux ne donne pas de nom aux lecteurs comme le fait Windows. Il dit juste « **La racine, c'est**/ ». Au lieu de séparer chaque disque dur, lecteur CD, lecteur de disquettes, lecteur de carte mémoire… Linux place en gros tout au même endroit sous une seule racine.

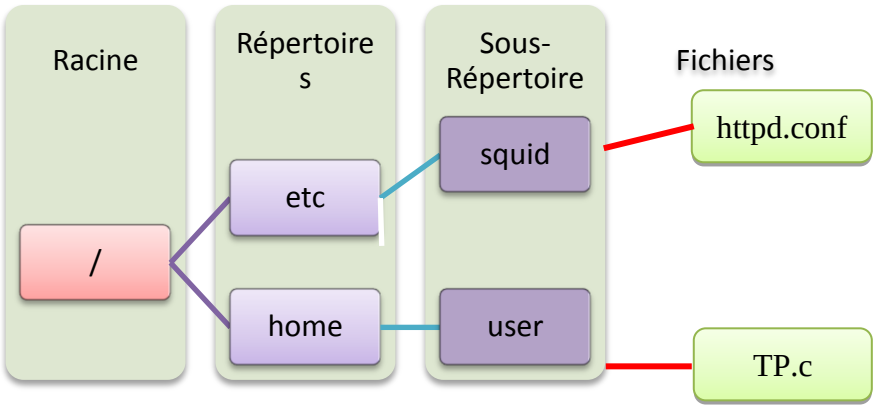

Figure 2 : Exemple d'arborescence

L'arborescence typique d'un système Linux est comme suit :

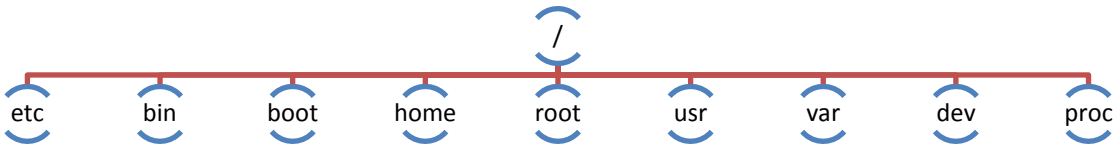

Figure 3 : Arborescence typique de Linux

- **etc :** répertoire contenant les fichiers de configuration. Ce répertoire contient des fichiers de données ainsi que des programmes réservés pour la maintenance du système. Ces fichiers conditionnent notamment le démarrage du système, les entrées en session, etc… Ce répertoire contient notamment le fichier ***passwd*** qui est un fichier texte contenant la liste des noms utilisateurs avec les mots de passe cryptés, leur identification (uid et gid), leurs répertoires initiaux et leurs Shells. On y retrouve également le fichier ***group*** qui est un fichier texte contenant la liste des groupes d'utilisateurs, leurs identificateurs et les listes d'utilisateurs par groupe.
- **bin** : répertoire contenant les principales commandes disponibles pour les utilisateurs ;
- **boot** : répertoire contenant les fichiers de démarrage du système contenant le noyau ;
- **home** : répertoire contenant les répertoires personnels des utilisateurs. Lorsque vous êtes connecté sous Unix, vous vous retrouvez chez vous à savoir votre home directory qui a été assigné par l'administrateur système (Ex : /home/imene). Tous les fichiers ou répertoires que vous allez créer, le seront sous votre home directory.
- **root** : répertoire personnel du super-utilisateur ;
- **usr** : Ce répertoire est réservé pour l'utilisation du système. Il est essentiellement utilisé comme répertoire racine pour de nombreux sous répertoires, notamment pour représenter les répertoires personnels des utilisateurs (home directories). Par exemple, l'administrateur du système réservera le plus souvent pour un utilisateur nommé « imene » un répertoire personnel

/usr/imene. De plus, des sous répertoires additionnels sont créés et réservés pour Unix. Ces sous répertoires sont généralement les suivants :

- ⇨ /usr/bin est un répertoire composé essentiellement de fichiers contenant le code exécutable des principales commandes sous linux. Si un de ces fichiers est supprimé ou change de nom, la commande correspondante ne sera plus reconnue.
- ⇨ /usr/include est un répertoire qui contient les fichiers référencés en tête (*.h) d'un programme source en langage C.
- ⇨ /usr/lib est un répertoire qui comprend généralement des bibliothèques ou fichiers de données supplémentaires à l'usage de linux.
- ⇨ /usr/spool est un répertoire contenant une variété de sous répertoires utiles pour y conserver les fichiers en attente d'impression.

- ✈ **var** : répertoire contenant les journaux systèmes. Il contient une hiérarchie de fichiers système qui évoluent durant le fonctionnement de la machine. Ex : /var/spool/ est le répertoire contenant les files d'attente pour les sous-systèmes du courrier, de l'impression,
- ✈ **dev** : point d'entrée vers les périphériques. Ce répertoire contient des fichiers spéciaux qui assurent le contrôle des accès aux différents dispositifs d'entrée/sortie. Ces fichiers sont indispensables et ne peuvent en aucun cas être détruits sous peine de rendre impossible l'accès aux unités correspondantes. Il est donc utilisé pour les périphériques, terminaux, etc. …
- ✈ **proc** : pseudo-système contenant des informations sur les processus en exécution

Remarques :

Sous le système Linux, plusieurs symboles sont utilisés pour désigner les répertoires. On cite pour cela :

- ✈ Le « . » : pour désigner le répertoire courant (la commande pwd affiche le nom du répertoire courant)
- ✈ Le « .. » : pour désigner le répertoire parent courant (la commande cd .. affiche le nom du répertoire parent)
- ✈ Le « ~ » : pour désigner le répertoire personnel de l'utilisateur (la commande cd ~ affiche le nom du répertoire personnel)

2.6 Chemins relatifs et chemins absolus

Sous Linux le disque est organisé en une structure hiérarchique, c'est-à-dire en une arborescence de répertoires. Chaque répertoire contient des noms de fichiers ou de sous répertoires qui contiennent eux-mêmes des noms de fichiers et de sous répertoires, etc. …

Pour retrouver un répertoire ou un fichier, il sera donc nécessaire d'énumérer la séquence de répertoires à traverser pour aboutir au répertoire ou au fichier souhaité. Cette séquence est appelée chemin d'accès ou "pathname".

Sous Linux, un chemin d'accès se compose d'une suite de mots séparés par des /. Il faudra cependant faire la distinction entre les chemins d'accès absolus et relatifs.

- **Chemin absolu** : la référence absolue d'un objet (fichier ou répertoire) se fait par la liste des répertoires traversés pour atteindre cet objet, en commençant par la racine /. Ex: /home/imene/iset/cours référence le fichier cours qui est sous le répertoire iset lui-même sous imene qui lui est sous la racine ce qui est indiqué par le premier /.
- **Chemin relatif** : la référence relative d'un objet (fichier ou répertoire) se fait par la liste des répertoires traversés pour atteindre cet objet, relativement au répertoire courant de travail. Ex : TP/tp2.c indique qu'à partir du répertoire de travail, on doit trouver un sous répertoire TP contenant le fichier tp2.c

Tout chemin qui ne commence pas par un caractère / (slash) est interprété comme un chemin relatif au répertoire courant. On peut ainsi accéder aux fichiers du répertoire courant en donnant simplement leur nom.

2.7 Types de fichiers sous linux

Il existe deux grands types de fichiers sous Linux :

- **les fichiers classiques** : ce sont les fichiers que nous connaissons, ça comprend les fichiers texte (.txt,.doc,.odt…), les sons (.wav,.mp3,.ogg), mais aussi les programmes. Bref, tout ça, ce sont des fichiers retrouvables dans Windows ;
- **les fichiers spéciaux** : certains autres fichiers sont spéciaux car ils **représentent** quelque chose. Par exemple, le lecteur CD est un fichier pour Linux. Là où Windows fait la distinction entre ce qui est un fichier et ce qui ne l'est pas, Linux, lui, dit que **tout est un fichier**. C'est une conception très différente, un peu déroutante comme je vous l'ai dit, mais pas de panique, vous allez vous y faire.

2.8 Catégories de fichiers sous linux

La commande **ls –l** donne un tel résultat :

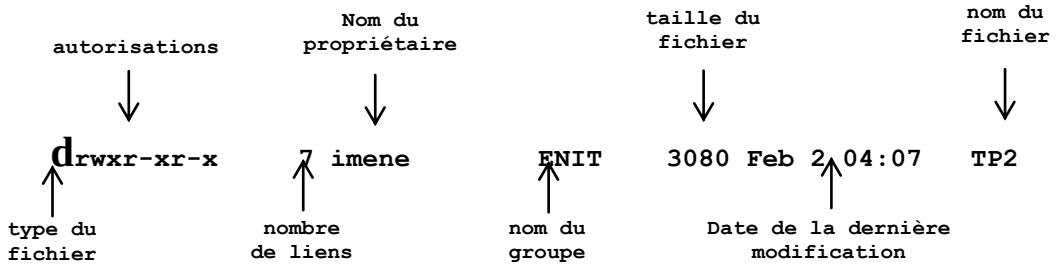

Figure 4 : Résultat de la commande ls -l

Il existe 4 catégories de fichiers :

- Les fichiers normaux (-) : comme les fichiers textes, courrier, sources de programmes(c, java…), scripts, configuration, exécutables, programmes en code binaire.
- Les fichiers répertoires (directories en Anglais) (d) : ce sont des fichiers conteneurs qui contiennent des références à d'autres fichiers. Ils permettent d'organiser les fichiers par catégories.
- Les fichiers spéciaux : situés sous /dev, ce sont les points d'accès préparés par le système aux périphériques. Le montage va réaliser une correspondance de ces fichiers spéciaux vers leur répertoire "point de montage". Par exemple, le fichier /dev/hda permet l'accès et le chargement du 1er disque IDE.
- Les fichiers liens symboliques (l) : ce sont des fichiers qui ne contiennent qu'une référence (un pointeur) à un autre fichier. Cela permet d'utiliser un même fichier sous plusieurs noms sans avoir à le dupliquer sur le disque.

2.9 Les I-nodes des fichiers

Un fichier possède plusieurs informations, telles que le nom, le contenu et d'autres informations comme les permissions et les dates des dernières modifications. Ces informations sont sauvegardées dans l'INODE du fichier (nœud d'index) avec d'autres données. Un i-node est une structure de quelques dizaines d'octets décrite dans /usr/include/sys/inode.h qui contient généralement les champs suivants :

- **le type du fichier** (fichier ordinaire, spécial, répertoire,...).
- **les droits d'accès**.
- UID : **le propriétaire** du fichier.
- GID : **le group** à le quel il appartient le propriétaire.
- **la date de dernière modification**, **la date du dernier accès**, **la date de création**.
- 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.

Remarques :

- Tout fichier possède son unique i-node.
- L'i-node contient la totalité des informations sur le fichier, sauf le nom.
- Les i-nodes sont tous de même taille.

2.10 Commandes shell de manipulations de fichiers

Une commande, dans le sens plus général, est un fichier exécutable.

2.10.1 La commande pwd (print working directory)

Lorsqu'un utilisateur se connecte à une station Unix, il est placé dans un répertoire personnel, en général situé dans le répertoire /home. La commande **pwd** affiche le chemin d'accès au répertoire courant.

2.10.2 La commande cd (change directory)

Cette commande permet de changer le répertoire courant. Sans argument cd permet de se déplacer vers le répertoire de connexion. **cd [chemin]**

Exemples

cd ~ et cd ramènent dans le répertoire de connexion.

cd . : ne change pas le répertoire courant.

cd .. : permet de se déplacer vers le répertoire parent.

cd / : il permet de se déplacer vers la racine.

2.10.3 La commande ls (list files)

ls [-options] chemin

Le chemin est un nom de fichier ou de répertoire. Si c'est un fichier, **ls** permet d'afficher sa description. Si c'est un répertoire, elle affiche son contenu. Sans arguments, le répertoire courant est traité. Les options de ls sont :

- -l : format long (types de fichier, droits, nombre de liens, propriétaire, groupe, taille en octets, date de la dernière modification, nom du fichier).
- -a : liste tous les fichiers (y compris les fichiers qui commencent par le point : fichiers cachés).
- -F : format court avec indication du type de fichier (ajoute * si exécutable, / si répertoire).
- -i : affiche les numéros d'Inode des fichiers.
- -R : récursif, génère la liste de tout les fichiers du sous-arbre tout entier.
- -r : trie en ordre inverse.
- -d : si l'argument est un répertoire, la commande affiche son nom.

Remarque

Certains caractères, appelés méta caractères, sont interprétés spécialement par le Shell avant de lancer la commande entrée par l'utilisateur.

Par exemple, si l'on entre ls *.c, le Shell remplace l'argument *.c par la liste des fichiers du répertoire courant dont le nom se termine par .c.

Les métacaractères permettent donc de spécifier facilement des ensembles de fichiers, sans avoir à rentrer tous leurs noms. Voici les plus utilisés :

- ⇨ * remplacé par n'importe quelle suite de caractères.
- ⇨ ? remplacé par un seul caractère quelconque.
- ⇨ [ ] remplacé par l'un des caractères mentionnés entre les crochets. On peut spécifier un intervalle avec - : [a-z] spécifie donc l'ensemble des lettres minuscules.

Exemple

```
ls *.[a-z].*
hel.l.o a.m.a
ls [hg]*
ger hello
```

2.10.4 La commande cat

La commande cat affihe les fichiers l'un après l'autre sur la sortie standard (écran). Si aucun argument n'est spécifié, lit sur l'entrée standard (clavier) jusqu'à rencontrer un caractère fin de fichier CTRL^D.

Exemple

cat fich.txt : cette commande permet d'afficher sur l'écran le contenu du fichier fich.txt
cat fich1 fich2 : cette commande permet d'afficher sur écran le contenu des deux fichiers fich1 et fich2 (Concaténer deux fichiers).

2.10.5 La commande more

La commande **more [fichier...]** permet d'afficher le contenu d'un fichier page à page.

2.10.6 La commande touch (créer des fichiers vides)

Créer les fichiers f1.txt et f2.txt sous votre répertoire de travail en utilisant la commande suivante :

```
touch f1.txt f2.txt
```

Donner la commande qui permet de vérifier que les fichiers sont créés. Comment on peut vérifier que les fichiers sont vides ?

2.10.7 La commande file (type de fichier)

Il est possible sous Unix de connaître aussi le type de fichier sur lequel on travaille. Tous les fichiers ont un entête permettant de déterminer leur type (répertoire, exécutable, texte ASCII, programme C, document...). La commande **file** permet de visualiser le type du fichier en question.

file nom_de_fichier

Exemple:

file toto.c

toto.c: c program text

2.10.8 La commande mv (move)

Cette commande permet de déplacer un fichier **mv [–i] source destination**

Si « destination » est un répertoire, la commande **mv** déplace le fichier de nom source vers « destination ».

Si « destination » est un nom de fichier, la commande **mv** renomme le fichier de nom source.

L'option **mv -i** permet de demander confirmation en cas d'écrasement de la destination.

2.10.9 La commande ln (link)

Cette commande permet de créer des liens sur des fichiers. Les liens sont des fichiers spéciaux permettant d'associer plusieurs noms (liens) à un seul et même fichier. Ce dispositif permet d'avoir plusieurs instances d'un même fichier en plusieurs endroits de l'arborescence sans nécessiter de copie, ce qui permet notamment d'assurer un maximum de cohérence et d'économiser de l'espace disque. On distingue deux types de liens :

- **Les liens symboliques** représentant des pointeurs virtuels (raccourcis) vers des fichiers réels. En cas de suppression du lien symbolique le fichier pointé n'est pas supprimé. Les liens symboliques sont créés à l'aide de la commande ln -s selon la syntaxe suivante :

  ln -s nom-du-fichier-reel nom-du-lien-symbolique

ln –s fichier1 /titi/lien

ls –i fichier1 /titi/lien

→ 6341 fichier1 4230 /work/liensym

ls -l /titi/lien

4230 **l**rwxrwxrwx 1 ……. /titi/liensym → fichier1

Si on supprime le fichier de nom fichier1 : **rm fichier1**

Puis on fait **ls -i fichier1 /titi/lien** on aura :

fichier1 not found

4230 /titi/lien

Si on fait cat /titi/lien on aura :

cat : il n y a pas un fichier ou un répertoire de ce type.

- **Les liens physiques** (ou liens durs) représentent un nom alternatif pour un fichier c'est la création de deux ou plusieurs noms vers une inode unique au moyen de la commande ln. Ainsi, lorsqu'un fichier possède deux liens physiques, la suppression de l'un ou l'autre de ces liens

n'entraîne pas la suppression du fichier. Plus exactement, tant qu'il subsiste au minimum un lien physique, le fichier n'est pas effacé mais lorsque l'ensemble des liens physiques d'un même fichier est supprimé, le fichier est aussi supprimé. Il faut noter toutefois qu'il n'est possible de créer des liens physiques qu'au sein d'un seul et même système de fichiers. Les liens physiques sont créés à l'aide de la commande ln selon la syntaxe suivante :

ln nom-du-fichier-reel nom-du-lien-physique

Lors de la création d'un lien physique 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.

ls -li fichier1
6341 -rw-rw-r-- 1 …… fichier1
ln fichier1 /titi/liendur
$ls –li fichier1 /titi/ liendur
6341 -rw-rw-r-- 2 …… fichier1
6341 -rw-rw-r-- 2 …… /titi/ liendur

Les deux fichiers partent vers le même numéro d'index (numéro d'inode), ce qui veut dire qu'il y a un seul espace disque associé aux deux noms.

Si on déplae le fihier1 :

mv fichier1 fichier2

puis on fait :

ls -li fichier2

on aura :

6341 -rw-rw-r-- 2 …… fichier2

Si on supprime fichier2 :

rm fichier2

puis on fait :

ls -li fichier2 /titi/liendur

On aura :

fichier2 not found
6341 -rw-rw-r-- 1 …… /titi/liendur

2.10.10 La commande cp

cp [-options] source destination

Si « destination » est un répertoire, la commande cp copie le ou les fichier(s) source vers « destination ». Si « destination » est un nom de fichier, la commande cp renomme le fichier de nom source.

Si on effectue une copie d'un fichier sur un fichier qui existe déjà, celui-ci sera écrasé et remplacé par le nouveau fichier, certaines options de **cp** peuvent nous éviter de pareilles situations. On site dans ce qui suit quelques options :

- cp -i : avertit l'utilisateur de l'existence d'un fichier du même nom et lui demande s'il peut ou non remplacer son contenu.
- cp -b : permet également de s'assurer que la copie n'écrase pas un fichier existant : le fichier écrasé est sauvegardé, seul le nom du fichier d'origine est modifié et cp ajoute une tilde (~) à la fin du nom du fichier.
- cp -p : permet lors de la copie de préserver toutes les informations concernant le fichier comme le propriétaire, le groupe, la date de création.
- cp -r : permet de copier de manière récursive un répertoire et ses sous-répertoires.

2.10.11 La commande rm (remove file)

La commande rm est utilisée pour supprimer un fichier : **rm [-options] fichier(s)**

Ses principales options sont :

- rm -i : permet de demander à l'utilisateur s'il souhaite vraiment supprimer le ou les fichiers en question.
- rm -r : agit de façon récursive, c'est à dire détruit aussi les répertoires (pleins ou vide) et leurs sous-répertoires.
- rm -f : permet de supprimer les fichiers protégés en écriture et répertoires sans que le prompt demande une confirmation de suppression**.**

2.10.12 La commande mkdir (make directory)

Cette commande permet de créer un répertoire **mkdir [-options] répertoire**

Le chemin peut être :

- relatif (par exemple mkdir ../iset)
- absolu (par exemple mkdir /home/imene/iset /cours)

Une option de mkdir qui est souvent utile est mkdir –p elle permet de créer une suite de répertoires.

Exemple

```
mkdir –p cours/tp/rsi21
#crée le sous-répertoire rsi21 qui se trouve dans le sous-répertoire tp qui se trouve dans le répertoire #cours.
```

2.10.13 La commande rmdir (remove directory)

Cette commande permet de supprimer un répertoire vide **rmdir [-options] répertoire(s)**

L'options -i permet de demander à l'utilisateur s'il souhaite vraiment supprimer le ou les répertoires en question.

2.10.14 Droits d'accès et la commande chmod

Parmi les informations contenues dans l'inode de tout élément de l'arborescence, on trouve les droits d'accès. La protection d'un élément repose sur trois droits d'accès qui contrôlent les trois opérations de lecture (droit r pour read), d'écriture (droit w pour write) et d'exécution (droit x pour execute). Ces trois droits (rwx) sont appliqués à trois catégories d'utilisateurs qui sont le propriétaire (user : u) de l'élément, son groupe (group : g) et les autres utilisateurs (others : o).

La notion de groupe correspond simplement à un ensemble d'utilisateurs auxquels on peut attribuer les mêmes droits d'accès aux fichiers. Ainsi, chaque utilisateur appartient à un groupe et son identité qui sert pour les contrôles d'accès est définie par un numéro d'utilisateur (UID : User IDentification) et un numéro de groupe (GID : Group IDentification).

Tout système a besoin de comptes spécialisés, avec des droits importants, pour des tâches d'administration. Sous UNIX, il existe un compte root dit compte de super-utilisateur, caractérisé par son numéro d'utilisateur (l'UID zéro) qui lui confère les pleins pouvoirs.

La signification des droits diffère selon le type de l'élément : répertoire ou fichier

Code	Répertoires	Fichiers
r : lecture	Explorer	Voir le contenu
w : écriture	Ajouter ou supprimer des fichiers	Modifier le contenu
x : exécution	Autorise l'accès au répertoire	Exécuter

La commande chmod possède deux formes : numérique et symbolique.

- **Forme symbolique**

chmod [ugoa] [+|-|=] [rwx] fichier

La syntaxe symbolique précise au moyen d'une simple lettre la catégorie des utilisateurs concernés par la nouvelle protection :

- u pour user, propriétaire du fichier,
- g pour group, les utilisateurs du même groupe que le propriétaire,
- pour others pour les autres utilisateurs,
- a pour all, pour les 3 catégories des utilisateurs. Cette option est utile quand une même protection doit être définie vis-à-vis des 3 catégories des utilisateurs.

La syntaxe symbolique utilise l'opérateur + pour ajouter et - pour enlever ou = pour réassigner un droit d'accès. Lorsque l'opérateur = est utilisé, les types d'accès énumérés sont autorisés et les autres non mentionnés sont supprimés.

La syntaxe symbolique exige après l'opérateur que soient énumérées des lettres précisant les types d'accès à autoriser ou interdire. Le débutant se contentera d'utiliser r pour la lecture, w pour l'écriture et x pour l'exécution.

Exemples

- chmod g+w titi permet aux utilisateurs du groupe du propriétaire du fichier titi d'accéder en plus à l'écriture.
- chmod a=r toto autorise uniquement l'ensemble des utilisateurs à accéder au fichier toto en lecture seule. Si l'accès en écriture était autorisé, ce droit est supprimé.
- chmod ug-w tata interdit dorénavant au propriétaire et son groupe de tenter une modification du fichier tata.

- **Forme numérique**

chmod mode fichier

La commande chmod peut être utilisée avec un masque composé de 3 chiffres en octal. Les bits de ce masque autorisent les types d'accès correspondants avec la même relation entre poids des bits et types d'accès.

- Le bit de poids **4** autorise les accès en lecture,
- le bit de poids 2 autorise les accès en écriture
- le bit de poids 1 autorise les accès en exécution.

Exemple :

```
chmod 750 toto
```

permet, pour le fichier toto, d'autoriser n'importe quel type d'accès de la part du propriétaire, de limiter les accès des utilisateurs du même groupe que le propriétaire en lecture et exécution, d'interdire tout type d'accès émanant des autres utilisateurs.

Exemples

chmod 644 titi	permet d'allouer les droits d'accès suivants : -rw-r—r—
chmod 622 tata	permet d'allouer les droits d'accès suivants : -rw—w-w-

2.10.15 La commande umask

La commande **umask** permet de définir un masque de protection des fichiers lors de leur création. Cette commande se trouve dans le fichier **.bashrc** et elle peut être exécutée à tout moment. Le masque défini doit être codé en base 8.

Quand vous créer un fichier, par exemple avec la commande **touch**, ce fichier par défaut possède certains droits. Ce sont **666** pour un fichier (**-rw-rw-rw-**) et **777** pour un répertoire (**-rwxrwxrwx**), ce sont les

droits maximum. Vous pouvez faire en sorte de changer ces paramètres par défaut. La commande **umask** est là pour ça.

- **Pour un fichier** :

Si vous tapez **umask 022**, vous partez des droits maximum **666** et vous retranchez **022**, on obtient donc **644**, par défaut les fichiers auront comme droit**644** (**-rw-r-r--**).

Si vous tapez **umask 244**, vous partez des droits maximum **666** et vous retranchez **244**, on obtient donc **422**, par défaut les fichiers auront comme droit**422** (**-rw--w--w-**).

- **Pour un répertoire** :

Si vous tapez **umask 022**, vous partez des droits maximum **777** et vous retranchez **022**, on obtient donc **755**, par défaut les fichiers auront comme droit**644** (**-rwxr-xr-x**).

Si vous tapez **umask 244**, vous partez des droits maximum **777** et vous retranchez **244**, on obtient donc **533**, par défaut les fichiers auront comme droit**422** (**-rwx-wx-wx**).

umask n'est utilisatable que si on est propriétaire du fichier.

Calcul de : $(\mathbf{0666})_8$ **AND NOT** $(0022)_8$ pour les fichiers

$(0022)_8 = (000\ 010\ 010)_2$
NOT $(0022)_8 = (111\ 101\ 101)_2$
$(0666)_8$ **AND NOT** $(0022)_8 = (110\ 100\ 100)_2$
$(0666)_8$ **AND NOT** $(0022)_8 = (0644)_8$

Calcul de : $(\mathbf{0777})_8$ **AND NOT** $(0022)_8 = (0755)_8$ pour les répertoires

2.10.16 La commande chown

Change le propriétaire des fichiers spécifiés en argument. Seuls le propriétaire et l'administrateur peuvent effectuer cette opération.

chown login fichiers

Exemples

```
ls -l TP.doc
-rw-r--r--  1  imene  enit  432  Feb 22 12:42  TP.doc
chown  sghaier  TP.doc
ls –l TP.doc
```

```
-rw-r--r--  1  sghaier  enit  432  Feb 22 12:42  TP.doc
```

2.10.17 La commande chogrp

Cette commande permet d'attribuer les fichiers donnés en argument à un nouveau groupe d'utilisateurs.

chgrp groupe fichiers

Exemples

```
ls –l tp.doc
-rw-r--r--  1  imene  enit  432  Feb 22 12:42  tp.doc
chgrp  iset  tp.doc
ls –l tp.doc
-rw-r--r--  1  imene  iset  432  Feb 22 12:42  tp.doc
```

2.10.18 La commande find

La commande **find** parcourt les répertoires et les sous-répertoires de manière récursive à la recherche de fichiers. **find** doit disposer de 3 indications :

- à partir de quel répertoire la recherche doit-elle commencer ? (obligatoire)
- quels sont les critères de recherche à mettre en œuvre ? (non obligatoire)
- que doit-il se passer si un fichier répond à ce(s) critère(s) ? (obligatoire)

Syntaxe

find répertoire(s) critère_de_sélection option(s)

Si on n'indique pas un critère de recherche, l'ensemble des fichiers placés sous le répertoire de départ sera considéré comme répondant à la recherche et sera donc sélectionné. Le *critère_de_sélection* permet d'extraire une partie de ces fichiers à partir de l'ensemble.
La commande find a de nombreuses options de sélection qui permettent de rechercher les fichiers qui vérifient certains critères. En voici quelques unes :

Tableau: Critères de sélection de la commande find

Critère	Signification
-name motif	Recherche par nom de fichier. Le motif peut utiliser les caractères génériques du Shell ?, *, [ ]
-type type	Recherche par type de fichier. **f** pour le fichier ordinaire, **d** pour les répertoires, **b** pour les fichiers spéciaux en mode bloc, **c** pour les fichiers

	spéciaux en mode caractère, **l** pour les liens symboliques.
-user utilisateur	Recherche par propriétaire. Le propriétaire des fichiers trouvés doit être l'utilisateur référencé.
-group groupe	Recherche par groupe. Les propriétaires des fichiers trouvés doivent être des membres du groupe référencé.
-size [+/-]n	Recherche par taille de fichier. La taille du fichier est exprimée en blocs (512 octets). Si après la taille j'ajoute le caractère c, la taille sera exprimée en octets. Exemple : -size 50c
-atime [+/-]n	Recherche par date du dernier accès.
-mtime [+/-]n	Recherche par date de la dernière modification.
-ctime [+/-]n	Recherche par date de création.
-perm mode	Recherche par autorisations d'accès. Recherche les fichiers dont le mode de permission est égal au mode référencé.
-perm –mode	Recherche par autorisations d'accès. Recherche les fichiers qui ont au moins les autorisations décrite le mode.
-links [+/-]n	Recherche par nombre de liens pour les fichiers.
-inum [+/-]n	Recherche par numéro d'index pour les fichiers.
-newer nomfichier	Recherche des fichiers plus récents que le fichier passé en référence (nomfichier).

La notation [+/-]n indique un entier ayant les 3 formes suivantes:

- +n : indique que toute valeur supérieure à n est valide.
- n : indique que la valeur doit être exactement égale à n.
- -n : indique que toute valeur inférieure à n est valide.

On peut combiner les critères avec des opérateurs logiques :

- **critère1 critère2** ou **critère1 -a critère2** correspond au **et logique**,
- **!critère** non logique,
- **\ (critère1 -o critère2\)** ou logique,

La commande **find** est <u>récursive</u>, c'est à dire où que vous tapiez, il va aller scruter dans les répertoires, et les sous répertoires qu'il contient, et ainsi de suite.

L'option –print

La commande **find** doit être utilisée avec l'option **-print**. Sans l'utilisation de cette option, même en cas de réussite dans la recherche, **find** n'affiche rien à la sortie standard (l'écran, plus précisément le shell). La commande **find** est récursive, c'est à dire où que vous tapiez, elle va aller scruter dans les répertoires, et les sous répertoires qu'il contient, et ainsi de suite.

Exemple :

```
$find -print
.
./fic1
./fic2
./rep1
./rep1/fich2
./rep1/document
```

Cette commande a permis d'afficher toute la liste des fichiers du répertoire courant de façon récursive. C'est la plus simple ligne de commande utilisant **find**.

L'option –exec

L'option **–exec** permet d'exécuter une commande sur les fichiers qui vérifient les critères de sélection.

```
-exec commande {} \;
```

Exemples :

La commande suivante permet de rechercher les fichiers ordinaires de l'arborescence du répertoire courant qui ont une taille égale à 0 et supprimer chacun des fichiers trouvés.

```
$find . –type f -size 0 -exec rm {} \;
```

La commande suivante permet de rechercher les répertoires de l'arborescence de mon répertoire de connexion tel que le propriétaire est l'utilisateur imene, et appliquer à tout les répertoires trouvés un permission = 700.

```
$find ~ –type d -user imene -exec chmod 700 {} \;
```

La commande suivante permet de rechercher les fichiers ordinaires de l'arborescence de mon répertoire courant dont le nom se termine par .tmp, et copier chaque fichier trouvé dans le répertoire /tmp.

```
$find . –type f -name "*.tmp" -exec cp {} /tmp \;
```

L'option –ok

A la différence de l'option –exec, l'option **–ok** effectue la même chose mais en demandant confirmation de l'utilisateur avant l'exécution de la commande.
-ok commande {} \;

Exemples

La commande suivante recherche dans le répertoire courant les fichiers dont le nom commence par p et se termine .c

```
find . –name "p*.c" -print
```

La commande suivante recherche dans le répertoire courant les fichiers ordinaires ayant un numéro d'index > 13230 et affiche chaque fichier trouvé avec son numéro d'index.

```
find . –type f –inum +13230 -exec ls –i {} \;
```

La commande suivante affiche les fichiers de l'arborescence du répertoire /home qui ont été modifié dans les dix derniers jours.

```
find /home -mtime -10 -print
```

La commande suivante affiche les renseignements détaillés sur les fichiers de l'arborescence du répertoire /users qui appartiennent à toto et qui ont été modifié dans les dix derniers jours.

```
find /users –user toto –mtime -10 –exec ls –ld {} \;
```

La commande suivante affiche les renseignements détaillés sur les fichiers de l'arborescence du répertoire de connexion de l'utilisateur imene et qui appartiennent à imene.

```
find /home/imene ! –user imene –exec ls –ld {} \;
```

La commande suivante affiche les renseignements détaillés sur les fichiers ordinaires l'arborescence du répertoire courant qui ont au moins la permission 400, c'est-à-dire au moins le propriétaire a le droit de lecture du fichier.

```
find    .  –type f –perm -400 –exec ls –ld {} \;
```

La commande suivante affiche les fichiers de l'arborescence du répertoire courant et qui se terminent par .c ou par .java.

```
find    .  \(-name "*.c" –o –name "*.java"\) –print
```

La commande suivante supprime après confirmation par l'utilisateur les fichiers ordinaires l'arborescence du répertoire courant et qui sont vides.

```
find    .  –type f –size  0 –ok rm {} \;
```

2.10.19 La commande grep

La commande **grep** permet de rechercher une chaîne de caractères dans un fichier. Les options sont les suivantes :

- **-v** affiche les lignes ne contenant pas la chaîne
- **-c** compte le nombre de lignes contenant la chaîne
- **-n** chaque ligne contenant la chaîne est numérotée
- **-x l**igne correspondant exactement à la chaîne
- **-l** affiche le nom des fichiers qui contiennent la chaîne

Remarque

On a vu auparavant ce qu'étaient les méta caractères. Les expressions régulières sont aussi des suites de caractères permettant de faire des sélections. Elles fonctionnent avec certaines commandes comme **grep**. Les différentes expressions régulières sont :

- ^ début de ligne
- **.** un caractère quelconque
- **$** fin de ligne
- **x*** zéro ou plus d'occurrences du caractère **x**
- **x+** une ou plus occurrences du caractère **x**
- **x?** une occurrence unique du caractère **x**
- **[...]** plage de caractères permis
- **[^...]** plage de caractères interdits

- **\ {n\}** pour définir le nombre de répétition **n** du caractère placé devant

Exemples :

- l'expression **grep [a-z][a-z] *** cherche les lignes contenant au minimum un caractère en minuscule. **[a-z]** caractère permis, **[a-z]*** recherche d'occurrence des lettres permises. L'expression **^[0-9]\ {4\}$** a pour signification, du début à la fin du fichier **$**, recherche les nombres **[0-9]** de 4 chiffres **\ {4\}**.
- Exemple avec le fichier **carnet-adresse** :

  olivier:29:0298333242:Brest
  marcel:13:0466342233:Gardagnes
  myriam:30:0434214452:Nimes
  yvonne:92:013344433:Palaiseau

  si on tape grep ^[a-] carnet-adresse
  On va obtenir tous les lignes commençant par les caractères compris entre a et d. Dans notre exemple, on aura :
  marcel:13:0466342233:Gardagnes
  myriam:30:0434214452:Nimes
  la commande grep Brest carnet-adresse Permet d'obtenir les lignes contenant la chaîne de caractère Brest, soit : olivier:29:0298333242:Brest

2.11 Les Filtres

Un filtre est un programme qui sait écrire et lire des données par les canaux standard d'entrée et de sortie (lit les données du clavier et retourne un résultat à l'écran). Il peut aussi agir sur les données d'un fichier donné en paramètre.

2.11.1 Recherche dans les fichiers : la commande grep

La commande **grep** lit les données, soit par le canal d'entrée standard, soit directement dans un fichier. Cette commande affiche sur l'écran les lignes des fichiers qui contiennent une chaîne de caractères correspondante à l'expression régulière spécifiée dans la ligne de commande par le paramètre *modèle_de_critére*. Autrement dit, le *modèle_de_critère* définit les lignes à trouver. Le modèle peut contenir des caractères spéciaux définis dans le second tableau.

Syntaxe

grep [options] modèle_de_critère [fichier1…]

Options

Tableau: Options de la commande grep

-v	Affiche seulement les lignes qui ne contiennent pas la chaîne.
-c	La commande ne retourne que le nombre de lignes contenant la chaîne mais sans afficher ces lignes.
-i	La recherche ignore la différence entre les majuscules et minuscules.
-n	Affiche chaque ligne trouvée avec son numéro de ligne.

Expressions régulières

Les expressions régulières sont utilisées par plusieurs commandes en particulier la commande grep. Un caractère ordinaire se désigne par lui-même mais l'expression régulière peut aussi contenir des caractères spéciaux.

Tableau : Spécification de certaines expressions régulières

Caractère	Signification
[…]	Plage de caractères permis à cet emplacement.
[^…]	Les caractères mentionnés sont interdits à cet emplacement, tous les autres, non placés entre les crochets étant permis.
.	Désigne n'importe quel caractère.
*****	Le * est un signe de répétition. Il agit sur le caractère placé devant l'étoile. Exemple : x* désigne 0 ou plusieurs caractères x successives.
+	Le + est un signe de répétition. Il agit sur le caractère placé devant. Exemple : x+ désigne 1 ou plusieurs caractères x successives.
^	Quand le signe ^ est placé au début du *modèle_de_critère*, *grep* recherche les lignes qui commencent par les chaînes de caractères décrites par le *modèle_de_critère*.
$	Placé en fin du modèle de critère, il désigne une fin de ligne.
****	Suivi d'un autre caractère, il désigne ce caractère. Il permet d'enlever le sens spécial des caractères * . \ [ ] $.

^ab : désigne la chaîne ab placée en début de ligne.

[0-9] : désigne un chiffre quelconque.

[adh-lA-Z] : désigne la lettre a ou la lettre d ou une lettre minuscule entre h et l, ou une lettre majuscule.

[^0-9] : désigne un caractère qui n'est pas un chiffre.

^\. : désigne une ligne qui commence par un point.

Exercice

On considère le fichier **Fleurs** :

```
cat Fleurs
Rose
Jasmin
Lilles
Orchidées
Marguerites
Camomilles
```

a. Rechercher dans le fichier Fleurs les lignes qui commencent par une lettre entre L et O .

b. Rechercher dans le fichier Fleurs les lignes qui commencent par une voyelle.

c. Rechercher dans le fichier Fleurs les lignes qui commencent par une lettre non voyelle .

d. Rechercher dans le fichier Fleurs les lignes qui commencent par un caractère quelconque et le second caractère est le caractère o.

e. Rechercher dans le fichier Fleurs les lignes qui se terminent par « se ».

f. Rechercher dans le fichier Fleurs les lignes qui ne commencent pas par M (l'option -v).

g. Rechercher les lignes qui ne commencent pas par M et afficher leur nombre seulement.

Solution

a.

```
grep ^[O-L] Fleurs
Lilles
Orchidées
Marguerites
```

b.

```
grep ^[aeiou] Fleurs
Orchidées
```

c.

```
grep ^[^aeiou] Fleurs
Rose
Jasmin
Lilles
Marguerites
Camomilles
```

d.

```
grep ^.o Fleurs
Rose
```

e.

```
grep se$ Fleurs
Rose
```

f.

```
grep -v ^p Fleurs
Rose
Jasmin
Lille
Orchidées
Camomille
```

g.

```
grep -v -c ^M Fleurs
1
```

2.11.2 Tri dans un fichier (sort)

```
sort [options] [+Pos1] [-Pos2] [Fichier…]
```

La commande **sort** effectue un tri des lignes d'un ensemble de fichiers (l'entrée standard par défaut). Les lignes sont triées selon la valeur d'une clé de tri formé d'une ou plusieurs champs extraits de chaque ligne. Par défaut, le tri se fait sur les lignes selon l'ordre lexicographique et selon l'ordre croisant.

+pos1 –pos2 permet d'indiquer le début et la fin du colonne de tri. Il indique que la colonne de tri commence au champ numéro pos1+1 et se termine a champ numéro pos2+1 (non compris).
Exemple : +2 -3 désigne le champ numéro 3. +2 -4 désigne les champs 3 et 4.

Si –Pos2 est omis, le tri porte de +Pos1 jusqu'à la fin de la ligne.
L'indication de plusieurs critères de tri est parfaitement possible. Le second critère ne sera mis en œuvre que si le premier critère ne suffit pas à trier les lignes.

Tableau : Options de la commande sort

Option	Signification
-n	Effectuer un tri numérique et non lexicographique sur les clés. On peut aussi accoler l'option n derrière pos1 si on ne veut trier numériquement que certains champs.
-f	Ignorer les différences entre les majuscules et minuscules. Toutes les minuscules sont converties en majuscules et le tri n'est effectué qu'après.
-r	Inversion de l'ordre de tri, donc tri est par ordre décroissant.
-tc	Le caractère « c » après l'option –t est définit comme caractère de séparation entre les champs d'une ligne.

Exercice :
On suppose que le fichier **Agenda** utilise le séparateur tabulation entre les champs.

cat Agenda

Zitouni Sarra	2090	Ariana	71345098
Skouri Mouna	2080	Tunis	71527860
Chebbi Moez	1060	Mahdia	73903561
Azzouna Ahmed	2100	Nabeul	72209189

a. Triez le fichier Agenda d'après les prénoms (le second champ).
b. Triez le fichier Agenda d'après les codes postaux
c. On veut trier le fichier **Agenda** selon le prénom (2 éme champ) et en cas d'égalité on effectue le tri de la colonne de l'adresse (4 éme champ).
d. On veut effectuer le tri du fichier /etc/passwd sur le numéro de l'utilisateur. Le séparateur utilisé dans /etc/passwd est le deux points « : ».

Solution :

a.

```
sort +1 -2 Agenda
```

Azzouna Ahmed	2100	Nabeul	72209189
Skouri Mouna	2080	Tunis	71527860
Chebbi Moez	1060	Mahdia	73903561
Zitouni Sarra	2090	Ariana	71345098

b.

```
sort -n +2 -3 Agenda
```

Chebbi Moez	1060	Mahdia	73903561
Skouri Mouna	2080	Tunis	71527860
Zitouni Sarra	2090	Ariana	71345098
Azzouna Ahmed	2100	Nabeul	72209189

c.

```
sort +1 -2 +3n -4n Agenda
```

d.

```
sort –t: +2n -3n /etc/passwd
```

Le tri ne se fait pas correctement si on oublie l'option n à la suite de +2. Ceci parce qu'on veut effectuer un tri d'un champ numérique.

2.11.3 Compter les caractères, les mots, les lignes (wc)

```
wc [options] [fichiers]
```

Cette commande affiche le nombre de lignes, de mots, de caractères et le nom de chaque fichier. Si des fichiers ne sont pas spécifiés, l'entrée standard est prise par défaut.

Options

```
-l      : nombre de lignes seulement
-w      : nombre de mots seulement
-c      : nombre de caractères seulement
```

Exercice

a. Comptez le nombre de fichiers de /bin

```
ls -a  /bin | wc –l
```

b. Comptez les lignes, mots et caractères de tous les fichiers du répertoire courant dont le nom commence par b.

```
wc  -lwc  b*
```

2.11.4 Conversion de caractères (tr)

```
tr  chaîne1 chaîne2
```

La commande **tr** transforme les caractères provenant de l'entrée standard ainsi (les caractères contenus dans la première chaîne sont transformés en les caractères correspondants dans la seconde chaîne et les autres caractères sont conservés tels quels. Le tout est envoyé vers la sortie standard.

Exercice

1. Ecrivez une commande qui permet de convertir le texte lu à partir du clavier (entrée standard).

```
tr "aeiou" "AEIOU"
bonjour
Ctrl^D
bOnjOUr
```

2. Ecrivez une commande qui permet de convertir le texte du fichier /etc/passwd de façon à changer chaque minuscule en une majuscule.

```
cat /etc/passwd | tr  "a-z"  "A-Z"
```

2.11.5 Découpage de fichiers (cut)

La commande **grep** sélectionne des lignes individuelles dans un fichier ou dans le flux de données. On parle dans ce cas de sélection horizontale. La commande **cut** sert à couper des zones dans chaque ligne, donc à effectuer une sélection verticale. **cut** sélectionne soit des

colonnes bien précises, que vous souhaitez afficher, soit elle montre des champs séparés les uns des autres par des caractères de séparation définis par l'utilisateur.

Sélection de colonnes

cut -cliste [fichiers…..]

Cette commande découpe un ou plusieurs fichiers (entrée standard par défaut) en sélectionnant des intervalles de caractères indiqués à l'aide de l'option -c. Le résultat est envoyé à la sortie standard.

Le paramètre liste est une liste de nombres séparés par des virgules avec éventuellement des tirets pour indiquer un intervalle. Le découpage se fait en se basant sur les numéros de caractères. Cette liste indique quels seront les caractères qui seront conservés sur chaque ligne.

- une colonne individuelle (par exemple –c5)
- une plage de colonnes (par exemple -c3-10 ou –c8-)
- une liste de chiffres séparés par des virgules (par exemple –c3,7,9)
- une combinaison des trois formes précédentes (par exemple -c1-3,7,20-)

Exemples

1, 3, 8 : désigne les caractères dans les positions 1, 3 et 8.

2-5, 10 : désigne les caractères dans les positions entre 2 et 5 et la position 10.

-5, 8 : désigne les caractères situés dans les positions de 1 à 5 et la positions 8.

3- : désigne les caractères de la position n°3 jusqu'à la fin de la ligne.

Exercice :

Soit le fichier Mois suivant

cat Mois

```
Mois
Janvier
Février
Mars
```

a. Ecrivez une commande qui permet d'afficher le premier caractère du fichier Mois.

```
cut –c1  Mois
M
J
F
M
```

b. Ecrivez une commande qui permet d'afficher les 3 premiers caractères du fichier Mois.

```
cut –c1-3 Mois
Moi
Jan
Fév
Mar
```

Sélection de champs

```
cut   [-dx]  -fliste   [fichier…]
```

Dans ce cas, le découpage se fait sur des champs indiqués par l'option –f, délimités par un délimiteur indiqué par l'option -d.
Le séparateur par défaut est la marque de tabulation. Pour fixer un autre caractère délimiteur de champs, on utilise l'option –d, en plaçant le caractère de séparation derrière l'option –d.

Exercice :

On considère le fichier **adresses** qui contient des champs séparés par deux points comme suit :

```
cat Adresses
Mahmoudi:Houssem:9000:Gabes
Belguessem:Ghassen:5090:Gafsa
Mediouni:Maha:3080:Beja
```

a. Ecrivez une commande qui permet de sélectionner le 3ème champ du fichier adresses

```
cut   -d:   -f3
9000
```

```
5090
3080
```

b. Ecrivez une commande qui permet d'afficher le 2ème et 4ème champ du fichier adresses

```
cut   -d:   -f2,4
Houssem Gabes
Ghassen Gafsa
Maha Beja
```

2.11.6 Comparaison du contenu de 2 fichiers

2.11.6.1 La commande diff

La commande **diff** permet d'indiquer quelles sont les lignes qu'on doit changer dans le premier fichier pour qu'il sera identique au second fichier.

```
diff  fichier1  fichier2
```

2.11.6.2 Commande cmp

```
cmp [-s] fichier1 fichier2
```

La commande **cmp** affiche le premier octet différent dans les 2 fichiers.
L'option –s permet de retourner un code de retour. En effet, elle retourne 0 si les deux fichiers sont identiques, la valeur 1 s'ils sont différents et 2 s'il a eu une erreur lors de l'exécution de la commande.

Exemple

```
cmp fich1 fich2
fich1 fich2 differ: char 160 line 5
```

2.11.7 Extraire le début ou la fin d'un fichier

2.11.7.1 Début du fichier (head)

```
head  [-n]  [fichiers]
```

La commande **head** extrait les n premières lignes de chacun des fichiers (entrée standard par défaut). Par défaut la valeur de n est égale à 10.

2.11.7.2 Fin du fichier (tail)

```
tail [+/-position] [fichiers]
```

La commande **tail** permet de copier la fin du fichier sur la sortie standard à partir de la position désignée. La première ligne affichée sur la sortie standard est indiquée par la position qui peut être +n (la n^{ième} ligne du fichier) et qui peut être aussi –n (la n^{ième} ligne à partir de la fin du fichier).

- *position* : un nombre qui définit le nombre de lignes à présenter.
- *+position* : recopie à partir de la position donnée en partant du début.
- *-position* : recopie à partir de la position donnée en partant de la fin.

Exercice :

Soit le fichier Fruits suivant

```
cat Fruits
pomme
poire
orange
pamplemousse
fraise
banane
```

a. Afficher les lignes du fichier Fleurs à partir de la deuxième ligne.

```
tail +2 Fruits
poire
orange
pamplemousse
fraise
banane
```

b. Afficher les 3 dernières lignes du fichier.

```
tail -3 Fleurs
```

```
pamplemousse
fraise
banane
```

2.11.8 Suppression des doublons (uniq)

La commande **uniq** n'affiche qu'un seul exemplaire des lignes de l'entrée standard ou du fichier passé en paramètre. Si plusieurs lignes consécutives sont identiques, une seule est envoyée vers la sortie standard. Cette commande ne considère pas deux lignes identiques non consécutives, il faut faire un tri préalable en utilisant la commande **sort** si on veut les prendre en compte.

Exercice :

Soit le fichier prénoms suivant

```
cat prénoms
mohamed
imene
Ali
Anis
olfa
safa
ahmad
safa
hajer
```

a. Ecrivez une commande qui affiche les lignes du fichiers "prénoms" tout en ignorant les doubloons

```
uniq prénoms
mohamed
imene
Ali
Anis
olfa
safa
ahmad
```

b. Triez le fichier selon l'ordre alphabétique

```
sort prénoms
ahmad
Ali
Anis
imene
mohamed
olfa
safa
safa
```

c. Triez le fichier et affichez le en ignorant les doubloons

```
sort prénoms | uniq
ahmad
Ali
Anis
imene
mohamed
olfa
safa
```