Download as PDF

TP n°2: Gestion des fichiers

Objectifs:

- Connaître la structure arborescente d'UNIX et les principaux répertoires
- Savoir indiquer le chemin relatif ou absolu d'un objet
- Savoir lire les attributs d'un objet (type, droits, propriétaire, etc.)
- Maîtriser les commandes élémentaires de gestion des fichiers et des répertoires

Éléments techniques abordés:

- *Les commandes*
  - `echo`, `tar`, `man`
  - `cat`, `more`, `less`, `cp`, `mv`, `rm`
  - `ls`, `cd`, `pwd`, `mkdir`, `rmdir`
  - `which`
  - `nano`, `vi`, `emacs`
- *Les répertoires*
  - `/`, `/bin`, `/sbin`, `/dev`, `/etc`
  - `/home`, `/lib`, `/mnt`, `/tmp`, `/user`, `/var`
  - `/opt`, `/proc`, `/root`
- *Les fichiers*
  - `/dev/null`

Le travail de base d'un utilisateur consiste principalement à exécuter des logiciels, et à gérer ses fichiers ainsi que ses répertoires. Nous allons nous intéresser dans ce TP à la méthode de gestion des fichiers et des répertoires sous UNIX, ainsi que les commandes associées. Les notions de droits ainsi que d'attributs sur les fichiers seront abordées dans le TP n°3: Gestion avancée des fichiers.

Notions sur les fichiers et les répertoires

Avant de présenter la gestion des fichier sous UNIX et la notion d'arborescence, rappelons quelques grands concepts sur les fichiers et les répertoires.

Principe

Une session de travail sur un ordinateur consiste à traiter des données à l'aide d'un logiciel. Deux composants sont souvent sollicités:

1. La mémoire centrale, pour le traitements des données;
2. le disque dur, pour la sauvegarde.

Les utilisateurs usent volontiers du terme « fichier » pour parler des données affichées à l'écran par leur logiciel applicatif et qui sont, à ce moment précis, dans la mémoire centrale. Il faut rappeler, pour lever toute ambiguïté, que les notions de fichier et de répertoire ne s'appliquent qu'aux données stockées sur un

support physique (disque dur; CD-ROM; clé USB, etc.).

Remarques:

1. Dans un ordinateur, tout traitement de données se déroule dans le processeur (CPU [https://fr.wikipedia.org/wiki/Processeur]) à partir des données contenues dans la mémoire centrale (mémoire vive). Le logiciel utilisé y est tout d'abord copié pour être exécuté. Les données sont ensuite chargées dans une partie de la mémoire dédiée à l'application. Aucun traitement ne se déroule sur le disque dure. Si la session s'arrête brutalement, toutes les modifications sont perdues.
2. Le disque dur de l'ordinateur est dédié à la conservation des données. La sauvegarde consiste à faire une copie des informations présentes en mémoire, sur le disque, avant de quitter le logiciel. Il est même recommandé d'en faire périodiquement durant la session.

L'organisation des données sur le disque dépend du système d'exploitation. Les systèmes actuels utilisent une arborescence de fichiers et de répertoires. Cette organisation est détaillée un peu plus loin dans ce TP.

Concept de fichier

Les utilisateurs ont généralement une vision erronée du fonctionnement d'un fichier. Nous présentons ici une version simplifiée de cette « structure de données », afin de proposer une vision « logique » de l'organisation des données. Cette représentation est différente de l'implantation physique sur disque.

Représentation logique

Un fichier est un espace disque contenant des données. Il est généralement crée au moment de la première sauvegarde. Le système UNIX ne propose aucune commande spécifique de création de fichier. Par analogie au modèle humain, le fichier est similaire à la feuille de papier sur laquelle sont stockées les informations, alors que la mémoire centrale est comparable au cerveau dans lequel sont traitées les données. En clair, avant d'écrire une phrase sur une feuille de papier, il faut y penser. Il y a donc une phase de traitement en mémoire (dans le cerveau) avant la phase de sauvegarde sur le disque (la feuille de papier).

Remarque: Le fichier est une « enveloppe logique » qui délimite une zone de stockage de données sur le disque dur. Il est caractérisé par un type (texte, binaire, exécutable, etc.) qui dépend de la nature des données qu'il contient, et par des *attributs* (comme les droits), qui permettent à chaque utilisateur d'en contrôler l'accès. Contrairement aux informations qui se trouvent en mémoire centrale et qui s'effacent quand l'ordinateur s'éteint, le fichier conserve les données de manière permanente.

Représentation physique

Les données du fichier sont réparties sur un ou plusieurs secteurs du disques dur. Chaque secteur stocke 512 octets. L'écriture ne se fait plus secteur par secteur, mais par *groupe de secteurs*, également appelé *cluster*. Les secteurs (ou groupe de secteurs) d'un même fichier ne se trouvent pas nécessairement les uns à la suite des autres : ils peuvent être dispersés sur le disque, auquel cas on parle de *fragmentation* de l'espace disque. Cette dispersion peut engendrer une perte de performance. des accès (la tête de lecture doit se déplacer beaucoup plus souvent pour accéder aux données). Le recours à un logiciel de défragmentation est généralement requis, afin de repositionner les données de manière contiguë pour chaque fichier, ce qui limite alors les déplacements de la tête de lecture.

Remarque:

1. Un fichier est représenté sur un disque par une suite de secteur (ou groupes de secteurs) qui peuvent être dispersés. Une table des secteurs qui cartographie l'ensemble du disque, secteur par secteur (ou cluster par cluster), permet de lire les informations contenus dans un fichier en retrouvant la suite des

secteurs qui le constituent.
2. La table des secteurs est appelée *table d'allocation des fichiers* ou *FAT* (*File Allocation Table*) sous Windows, et chaque unité de disque en possède une. Sa dimension limite la taille des partitions à 2 Go (gigaoctets) en FAT16 et à 2 To (téraoctets) en FAT32. UNIX utilise une table appelée *table d'inodes*. Chaque *système de fichiers* (ou *file system*) possède sa table. Elle référence chaque fichier par un numéro entier unique, appelé *inode*.

Créer un fichier

En reprenant la liste des commandes de manipulation de fichiers et de répertoires, on remarque qu'il n'existe pas de commande pour créer un fichier comme il en existe une pour créer un répertoire (mkdir).

Pour créer un fichier sous Unix, on est alors obligé de détourner le fonctionnement normal d'autres commandes:

	commande	remarque
1	touch fichier	méthode recommandée si fichier vide
2	nano fichier	puis quitter en sauvegardant /sinon
3	cat > fichier	
4	cp /dev/null fichier	
5	mv /dev/null fichier	
6	echo "" > fichier	

Editer un fichier

Sur les systèmes UNIX, on retrouve quasi-systématiquement les programmes Vi et Emacs; vous devez être familier des commandes de base de ces éditeurs. Sous Linux, on trouve également l'éditeur nano; c'est celui que nous vous recommandons d'utiliser au quotidien, et dans la suite de ces travaux pratiques. Pour l'utiliser:

```
nano fichier
```

Concept de répertoire

Le répertoire est la structure complémentaire des fichiers pour la gestion de l'espace disque. Il a un rôle organisationnel, et chaque répertoire définit un niveau hiérarchique de l'arborescence. Tout comme le fichier, il possède une représentation logique et une implantation physique.

Représentation logique

Un répertoire est un espace de rangement de fichiers, au même titre qu'un tiroir de bureau dans lequel sont rangées les feuilles contenant les informations. Un répertoire peut comprendre des fichiers ou d'autres répertoires. Cette organisation est comparable à celle d'un bureau dans lequel un tiroir principale contient des « sous-tiroirs », qui peuvent eux-même en renfermer d'autres.

Remarques:

1. Le répertoire est un espace de rangement de fichiers. Il peut contenir des fichiers et des sous-répertoires. Il dispose de droits, à travers lesquels l'utilisateur peut protéger son espace disque. Il correspond à une table de « sommaire », dans laquelle chaque entrée donne des informations sur un fichier ou sur une table.

2. Chaque utilisateur possède un *répertoire personnel* appelé répertoire *home* ou *home directory*.

Représentation physique

L'implantation physique sur le disque d'un répertoire est identique à celle d'un fichier. Le répertoire occupe un certain nombre de secteurs (ou clusters). Ce nombre est fixe dans l'environnement Windows, ce qui limite la quantité de fichiers par répertoire. Si un seul répertoire existe, le nombre total de fichiers sur le disque correspondant au nombre maximal de fichiers de ce répertoire. Sous UNIX, la gestion d'une hiérarchie de répertoires en cascade permet de dépasser cette contrainte.

L'organisation des répertoires sous UNIX est identique, mais la limitation est fixée par la table des inodes, qui précise le nombre de fichiers par « système de fichiers » ou « *file system*».

Nom d'un fichier ou d'un répertoire

Les commandes de gestion de l'espace disque copient, déplacent, détruisent, etc. (cf. Memento Unix - Manipuler les fichiers) des fichiers ou des répertoires. La syntaxe de ces commandes utilise comme argument le nom du fichier ou du répertoire en question (voire des caractères de substitution). Le choix du nom de l'objet doit respecter certaines conventions que nous présentons ci-dessous.

Règles de nommage

Les caractères autorisés.

Le système UNIX autorise tous les caractères pour nommer un fichier ou un répertoire. Si aucune restriction n'existe du point de vue du système, ce n'est pas le cas pour tous les environnements et en particulier pour l'interpréteur de commande ou terminal, qui attribue un sens précis à certain caractères, comme l'espace, l'étoile ou le point d'interrogation. **Il est fortement conseillé de rester prudent dans le recours à certains caractères**. D'autre part, le système UNIX s'appuie sur la table ASCII, qui différencie les majuscules et les minuscules.

Remarques:

1. Pour nommer vos fichiers ou répertoires, limitez-vous aux caractères alphabétiques minuscules et majuscules (de a à z et de A à Z), aux chiffres (de 0 à 9), ainsi qu'au point (.) et au caractère de soulignement (_).
2. Un nom de fichier ou de répertoire contenant des majuscules est différent d'un nom équivalent écrit en minuscules; ainsi les fichiers « Essai » et « essai » sont deux fichiers distincts.
3. L'utilisation de caractères accentués peut provoquer des erreurs d'interprétation (voir le paragraphe sur Les caractères accentués).

On insère parfois des caractères de contrôle (combinaison de la touche CTRL avec une touche de caractère) dans le nom, ce qui sert de moyen de protection. Leur présence rend impossible la saisie du nom sur une ligne de commandes.

La notion d'extension.

Avec le système UNIX, les règle qui régissent l'attribution de noms sont les même pour les fichiers et les répertoires. L'ensemble des caractères et des chiffres sont utilisables dans les deux vas, y compris le caractère « . » (point). Un répertoire peut aussi tout à fait posséder une extension.

Remarque:
La forme du nom ne permet pas de différencier un fichier d'un répertoire. Seul l'affichage des attributs de l'objet par la commande `ls -l` indique son type. Si la ligne commence par `d` (pour *directory*), c'est un

répertoire; si elle commence par un tiret (-), c'est un fichier.

UNIX ne propose aucune gestion des extensions. L'utilisateur est libre de se servir du caractère « . » autant qu'il le désire, ce signe n'ayant aucun sens particulier, ni pour les fichiers ni pour les répertoires. Ainsi, le nom suivant est tout à fait correct: `stage.unix.2010`. Si le système UNIX n'exploite pas la notion d'extension, en revanche certains logiciels l'utilisent. C'est le cas du compilateur C, qui attribue l'extension « .c » aux programmes sources, et « .o » aux programmes objets.

Exemples de noms: Voici quelques noms valides de fichiers ou de répertoires:

- `Rep.2010`
- `trace.txt`
- `essai`
- `liste_noms.data.20100922`

Le dernier nom contient le caractère soulignement, « _ », qui remplace l'espace. Si l'espace est autorisé, il peut cependant poser quelques problèmes sur une ligne de commandes. Ce point est abordé au paragraphe les caractères spéciaux.

Astuce:
Le numéro `20100922` correspond à une notation date du 22/09/2010. C'est une astuce qui fait gagner du temps lorsque l'on cherche des fichier selon un critère de date. En effet, les programmeurs ont souvent besoin de conserver plusieurs versions d'un fichier, en fonction des modifications successives. L'usage est d'ajouter la date (et éventuellement l'heure) à la fin du nom de la copie conservée. Or, si on ajoute l'extension `20100922` à la place de `22.09.2010`, cela permet de faire coïncider le tri « alphabétique » (ou plutôt ASCII) avec le tri par date. La commande `ls` affichera les différentes version du fichier par dates croissantes.

Exercice 2.1 (Les bien nommés)

Pour tous les noms de fichiers ci-dessous indiquez s'ils sont bien nommés, i.e., s'ils respectent les conventions de nommage UNIX:

1. `Le-fichier-de-maurice.txt`
2. `TP_noté.xls`
3. `exo_12.txt`
4. `Le fichier de maurice.txt`
5. `Le_fichier_de_maurice.txt`
6. `Le_fichier_de_Chloé.txt`
7. `Le_fichier_de_Chloe.txt`

Les caractères joker.

Certains caractères jouent le rôle particulier pour l'interpréteur de commande (shell). Ces caractères, appelés *jokers* ou *caractère de substitution*, sont les suivants: `*`, `?`, `[ ]`. Dans le nom du fichier, ils remplacent un nombre variable de caractères selon le caractère joker employé.

L'étoile

L'étoile « * » remplace un nombre quelconque de caractères (y compris aucun). On s'en sert pour sélectionner une liste de fichier (ou de répertoires) sur la base d'une partie du nom. Il peut être placé n'importe où dans le nom. Voici quelques exemples de son utilisation avec la commande `ls`.

Cette première syntaxe donne la liste de tous les fichiers (ou répertoire):

```
$ ls
coursUNIX          cours_1              exo.C.2010           exo.UNIX.2010
exo_1              exo_14               exo_2                exo_3                    exo_4
```

La syntaxe suivante affiche le nom de tous les fichiers (ou répertoires) commençant par « cou » :

```
$ ls cou*
coursUNIX          cours_1
```

Remarque:
Contrairement à Windows, la syntaxe « *.* » n'indique pas « tous les fichiers », mais tous les fichiers et répertoires contenant « . » (point). Pour indiquer « tous les objets », y compris ceux ne possédant pas d'extension, il faut employer la syntaxe « * ». Ainsi, la commande suivante copie tous les fichiers du répertoire courant dans le répertoire `/tmp`:

```
$ cp * /tmp
```

Exercice 2.2 (L'étoile)

Donner la commande qui permet d'afficher tous les fichiers (ou répertoires) contenant:

1. la chaîne « UNI »;
2. la chaîne « .UNIX. »;
3. trois parties séparées par le caractère « . ».

Le point d'interrogation

Le point d'interrogation « ? » remplace un caractère et un seul. Il s'utilise également pour sélectionner une liste de noms de fichiers (ou de répertoires) sur la base d'une partie du nom. Il peut être placé n'importe où dans le nom.

Cette première syntaxe donne la liste de tous les fichiers (ou répertoires) dont le nom contient 5 caractères et commence par « exo_ »:

```
$ ls exo_?
exo_1          exo_2          exo_3          exo_4
```

Exercice 2.3 (Le point d'interrogation)

Donner la syntaxe de la commande qui affiche la liste de tous les fichiers (ou répertoires) dont le nom:

1. contient 6 caractères et commence par « exo_ »;
2. est en trois parties (séparés par le caractère « . ») et dont le dernier élément est constitué de 4 caractères.

Les crochets

La syntaxe « [ ] » est identique au point d'interrogation, mais la substitution ne s'effectue qu'avec l'un des caractères présentés entre les crochets:

- **[abc]** Le caractère de remplacement est le « a », le « b » ou le « c »;
- **[0-9]** Le caractère de remplacement est un chiffre compris entre 0 et 9;
- **[!abc]** Remplace un caractère différent de « a », « b » ou « c »;
- **[!0-9]** Remplace un caractère autre que les chiffres compris entre 0 et 9.

Exercice 2.4 (Les crochets)

Donner la syntaxe de la commande permettant d'afficher la liste de tous les fichiers (ou répertoires) dont le nom:

1. contient 5 caractères, commence par « exo_ » et se termine par « 1 », « 2 » ou « 3 »;
2. contient 5 caractères, commence par « exo_ » et se termine par un caractère différent de « 1 », « 2 » ou « 3 »;

Les caractères à éviter

Les caractères spéciaux.

Certains caractère sont interprétés par le shell, qui leur attribue un sens particulier. Si un nom contient l'un de ces caractères, l'objet devient difficile à utiliser en lligne de commandes, où il n'est pas considéré comme un caractère « neutre ».

Les caractères joker : *, ?, []

Il ne faut pas les inclure dans les noms de fichiers mais y recourir comme caractères de remplacement.

Les caractères hiérarchiques et d'options: /, -

Ces deux signes indiquent le chemin d'accès au fichier (« / ») et l'option de la commande (« - »). Éviter de vous en servir, même si le « - » ne pose véritablement pas de problème qu'en début de nom.

Le caractère espace

Le caractère espace est considéré par le shell comme un séparateur de mots. Un nom contenant un espace sera assimilé à deux noms différents. L'interface graphique permet de taper des noms de fichiers ou de répertoires contenant des espaces. Mais cette syntaxe rend difficile l'utilisation de ces noms en ligne de commandes. Dans k'exemple qui suit, un répertoire `Dossier UNIX 2010` vient d'être créé *via* l'interface graphique. La commande `ls` affiche correctement son nom:

```
$ ls
Dossier UNIX 2010     cours_1          exo.UNIX.2010        exo_14           exo_3
coursUNIX             exo.C.2010       exo_1                exo_2            exo_4
```

Par exemple, la commande de déplacement `cd` (*change directory*) appliquée à ce répertoire échoue. Seul le nom « Dossier » est interprété, et il n'existe aucun répertoire de ce nom.

```
$ cd Dossier UNIX 2010
bash: cd: Dossier: No such file or directory
```

Pour contourner cette difficulté, il faut laisser le shell récupérer le nom complet en employant le caractère joker « * »:

```
$ cd Dossier*
```

La commande `pwd` (*print working directory*) affiche le nom du répertoire courant et confirme le succès du déplacement :

```
$ pwd
/home/dupont/Dossier UNIX 2010
```

Cette syntaxe a ses limites. Si deux répertoires commencent par « Dossier », la commande cd échoue à nouveau. Le déplacement ne s'effectue que dans un seul répertoire à la fois ! Le nom du répertoire contenant des espaces peut aussi être présenté entre guillemets (cf. Déspécialisation des caractères spéciaux).

Déspécialisation des caractères spéciaux.

Tous les caractères spéciaux sont interprétés par le shell, qui leur attribue un sens précis. Il est toutefois possible de neutraliser leur interprétation par la notation « '...' ». Tous les caractères compris entre les deux apostrophes sont « déspécialisés ». L'exemple suivant utilise la comme de ls et le caractère * pour affichier la liste de tous les fichiers commençant par « Messa ». Un seul fichier, dont le com est « Message » apparaît:

```
$ ls Messa*
Message
```

Le recours aux apostrophes annule l'interprétation du caractère « * ». Le fichier recherché porte le nom « Messa ». Il n'est pas trouvé:

```
$ ls 'Messa*'
ls: Messa*: No such file or directory
```

Tous les caractères compris entre les deux guillemets sont déspécialisés, sauf les caractères « \ » et « $ ». Dans l'exemple qui suit, la commande cd échoue à cause des espaces présents dans le nom du répertoire:

```
cd Dossier UNIX 2010
cd: Too many arguments.
```

L'utilisation des guillemets neutralise les espaces, qui ne sont plus assimilés à des séparateurs de mots:

```
$ cd "Dossier UNIX 2010"
$ pwd
/home/dupont/Dossier UNIX 2010
```

On peut aussi remplacer les guillemets de l'exemple précédent par le signe « \ »:

```
$ cd Dossier\ UNIX\ 2010
```

Cette syntaxe s'applique aussi aux caractères « \ », « ' » et « " ». Les exemples suivants utilisent la commande echo qui affiche, à l'écran, le texte donné en argument:

```
$ echo \\
\
$ echo \'
'
$ echo \"
"
```

Les caractères accentués.

Le code ASCII des caractères accentués n'est pas universel. Selon l'ordinateur, ou l'environnement graphique, leur numéro varie en fonction des tables de codages de caractères (ASCII étendu IBM, ASCII étendu Apple, ISO Latin1, ANSI 1252). Le libellé des noms de fichiers contenant des accents changera donc selon le contexte.

Ce problème apparaît lors d'échanges de fichiers entre deux ordinateurs utilisant des tables de codage distincts. Le caractères accentués est interprété différemment et il est remplacé par un autre portant le même numéro (le caractère « é » sous Windows et Linux, dans les tables ANSI 1252 et ISO Latin1, porte le numéro

233, qui correspond au « È » dans la table Apple). Par ailleurs, dans les environnements UNIX et Linux, il est possible d'ouvrir des fenêtres avec des polices de caractères différentes. Une police est une représentation graphique des caractères définis dans une table de codage. Si ces polices font appel à des tables différentes, alors le caractère accentué s'affichera correctement dans la fenêtre qui utilise la « bonne » police, et sera erroné dans l'autre.

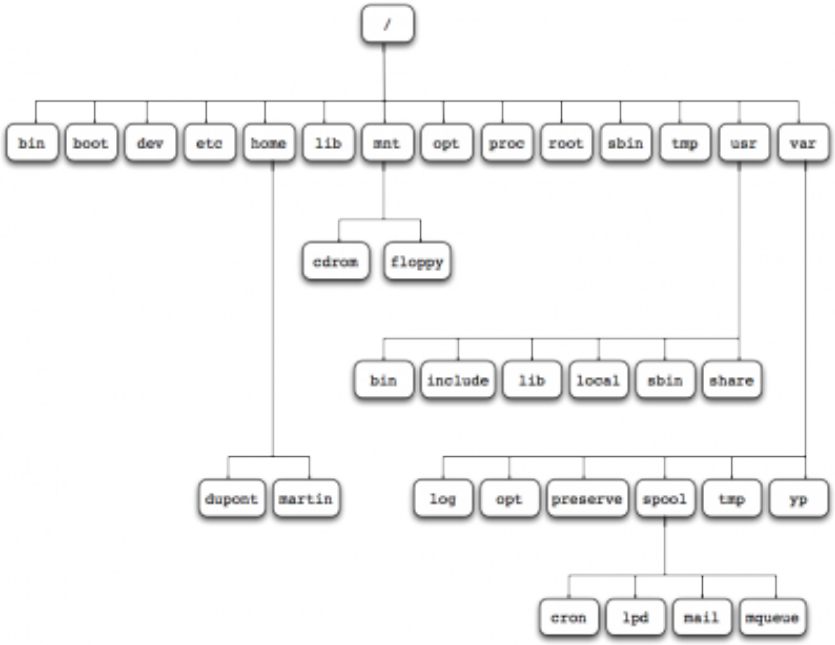
Évitez les caractères accentués dans les noms de fichiers sauf si c'est indispensable: préférez leurs homologues sans accent.

La hiérarchie

L'espace dique est géré en *hiérarchie arborescente*. Cette organisation en cascade de répertoires et de sous-répertoires, commence par une répertoire *racine..* Chaque répertoire peut contenir des fichiers. Contrairement à Windows, UNIX n'attribue aucun nom (ou lettre telle que C, D, E, etc.) aux unités de disques. La hiérarchie UNIX est déployée sur l'ensemble des disques et ne possède qu'une racine notée « / ».

Présentation de l'arborescence

Le système de fichier UNIX est illustré à la figure ci-dessous.



L'arborescence présenté ici n'est pas universelle. Elle correspond à une synthèse des hiérarchies rencontrées sur les systèmes UNIX. Les différences tiennent plus à la forme qu'au fond. Les noms des répertoires changent, mais leur rôle reste identique. D'autre part, les systèmes ont progressivement évolué pour respecter des normes telles que POSIX. Du fait de cette évolution, les systèmes font cohabiter les anciens noms de répertoires avec les nouveaux. Nous présentons ci-dessous quelques variantes d'arborescences (le rôle des répertoires indiqués est précisé plus loin).

Remarque: Sous Windows, les répertoires sont habituellement appelés des *dossiers*.

Particularités liées à la distribution

Le système Linux définit `/root` comme répertoire personnel de l'administrateur `root`. Ce n'est pas le cas de la majorité des systèmes (comme AIX), qui utilisent la racine, « / ».

Le répertoire `/boot` existe sous Linux mais pas sous AIX. Il contient l'image du noyau du système Linux.

Sous AIX, l'information équivalente est rangée dans le répertoire /unix et /usr/lib/root.

Compatibilité

Sous le système AIX, le répertoire /bin pointe sur /usr/bin, grâce à un lien symbolique. Ces deux nom désignent le même répertoire. Cette astuce permet au système de rester compatible avec les logiciels s'appuyant sur /bin ou /usr/bin. Le répertoire /u, répertoire des utilisateurs dans les versions antérieures d'AIX, pointe sur le répertoire /home.

Attention: L'arborescence des fichiers et des répertoires peut varier d'une version d'UNIX à une autre. Les grands répertoires sont toujours présents, mais leur nom et chemin d'accès peut changer. Ces différences sont relativement transparente pour l'utilisateur mais pas pour l'administrateur, qui doit s'adapter à la nouvelle hiérarchie lorsqu'il gère un nouveau système UNIX.

Le tableau ci-dessous présente la fonction des principaux répertoires du système UNIX.

Répertoire	Sous-répertoires	Contenu
/bin	/usr/bin	Commandes de base du systèmes
	/usr/local/bin	Commandes supplémentaires ajoutées par l'administrateur
/sbin	/usr/sbin	Commandes d'administration
/boot		Contient les fichiers du noyau Linux
/dev		Contient les fichiers particuliers aux périphériques
/etc	/etc.rc.d	Contient les fichiers et répertoires de configuration du système
	/etc/init.d	Sous-répertoire de démarrage des services sous Linux
/home		Contient tous les répertoires personnels des utilisateurs
/lib	/usr/lib	Contient les bibliothèques de sous-programmes utilisées pour les développements
/mnt		Contient les répertoires de périphériques amovibles
	/mnt/cdrom	Sous-répertoire d'accès au CD-ROM
	/mnt/floppy	Sous-répertoire d'accès au lecteur de disquette
	/mnt/usb	Sous-répertoire d'accès à la clé USB
	/mnt/nt	Sous-répertoire d'accès à la partition Windows sous Linux.
/opt		Contient des progiciels
/proc		Répertoire dédié aux processus
/root		Répertoire personnel de l'administrateur Linux
/tmp		Répertoire contenant tous les fichiers temporaires du système. Ce répertoire doit être nettoyer régulièrement.
/usr		Principale répertoire du système UNIX et des sous-systèmes.
	/usr/include	Sous-répertoire des fichiers d'en-tête
	/usr/share/man	Sous-répertoire des manuels UNIX
	/usr/X11R6	Sous-répertoire contenant les fichiers liés au protocole X11
	/usr/local	Sous-répertoire contenant les logiciels installés par l'administrateur
/var		Répertoire contenant la partie « variable » du système
	/var/log	Sous-répertoire contenant les traces d'activité du système (Linux). On le trouve également dans d'autres systèmes UNIX.
	/var/spool	Sous-répertoire contenant les files d'attente
	/var/mail	Sous-répertoire contenant les boîtes aux lettres des utilisateurs
	/var/spool/lpd	Sous-répertoire de gestion de files d'impression

	/var/spool/mqueue	Sous-répertoire d'envoi des courriers
	/var/yp	Sous-répertoire de gestion des NIS

Les espaces de travail

Chaque utilisateur possède plusieurs espaces de travail, qui rentre dans deux catégories: les espaces personnels que chacun administre à sa guise, et les espaces de travail communs partagés par tous.

Le répertoire personnel, ou home directory

Chaque utilisateur possède un *répertoire personnel*, également appelé *home directory*. Le nom de ce répertoire correspond généralement au login (identifiant) de l'utilisateur, et il est localisé sous `/home`. L'utilisateur `dupont` possède le répertoire : `/home/dupont`.

Attention: Cette règle n'est pas systématique. Ainsi, le répertoire personnel de l'administrateur *root* est généralement la racine, « `/` », ou bien « `/root` » pour le système Linux. De plus, dans le cas d'un système qui gère un grand nombre d'utilisateurs (plusieurs milliers) cette règle devra être modifiée. Elle engendre un trop grand nombre de sous-répertoires directement sous `/home`, ce qui devient vite pénalisant lorsqu'il faut lister son contenu. L'administrateur devra définir une autre politique, en prévoyant par exemple l'ajout d'un répertoire intermédiaire, dont le nom peut être choisi arbitrairement, ou en relation avec le groupe de l'utilisateur. Ainsi, le répertoire personnel de `dupont` qui appartient au groupe `l1` pourrait être : `/home/l1/dupont`. Cette politique présente l'avantage de rassembler les utilisateurs d'un même groupe sur un espace disque commun.

L'espace de travail temporaire

L'ensemble des utilisateurs disposent d'un second espace de travail, `/tmp`. Tous les utilisateurs du système ont le droit d'y créer des répertoires et d'y copier des fichiers. Contrairement au répertoire personnel, `/tmp` est dédié au stockage temporaire et l'administrateur peut détruire sont contenu à tout moment. Cet espace est généralement très grand, ce qui rend possible le traitement de gros fichiers, alors que le répertoire personnel est souvent limité par quotas (cf. TP n°3: Les quotas de disque).

N'utilisez le répertoire `/tmp` que comme espace temporaire.
Les administrateurs nettoient souvent ce répertoire, parfois chaque nuit. Il est également d'usage qu'il soit effacé à chaque redémarrage de la machine

La boîte aux lettres

La boîte aux lettres n'est pas un espace de travail mais une boîte de réception des *courriels* (*courriers électroniques*). Elle est personnelle, et chaque utilisateur peut modifier son contenu. Il s'agit en réalité d'un fichier texte, dont le nom est souvent celui de l'identifiant de l'utilisateur. Ce fichier se situe dans le répertoire `/var/spool/mail` ou `/var/mail` selon les systèmes. Ainsi, la boîte aux lettres de l'utilisateur `dupont` sera le fichier `/var/spool/mail/dupont`.

Les chemins

Pour copier, déplacer ou détruire des fichiers ou des répertoires, il faut connaître:

- Les commandes de gestion des fichiers et des répertoires;
- Le chemin d'accès à l'objet (fichier ou répertoire).

Une commande agit sur des objets dont le chemin d'accès doit être précisé. Nous décrivons ci-dessous la syntaxe des *chemins absolus* et *relatifs*.

Le chemin absolu

Le chemin absolu (ou syntaxe absolue) également appelé « *absolute pathname* », est constitué par la suite des noms de répertoires qui représentent l'itinéraire qui permet d'atteindre l'objet à partir de la racine. Il est terminé par le nom de l'objet. Le caractère hiérarchique « / » sépare les noms spécifiés dans le chemin. Voici quelques exemples de syntaxe absolue:

Répertoire personnel de dupont	/home/dupont
FIchier exo1.c se trouvent dans le répertoire de dupont	/home/dupont/exo1.c
Répertoire temporaire du système	/tmp

la commande suivante (cp) copie, dans le répertoire /tmp, le fichier exo1.c, qui se trouve dans le répertoire de dupont. Elle le copie sous le nom exercice1.c:

```
$ cp /home/dupont/exo1.c /tmp/exercice1.c
```

Le fichier à copier (exo1.c) et le fichier cible (exercice1.c), sont écrits en absolu. Une lecture « littérale » de cette commande serait:

« Copier le fichier qui se trouve, à partir de la racine, dans le répertoire home, puis dans le répertoire dupont, et dont le nom est exo1.c, vers le fichier cibke qui se trouve, à partir de la racine, dans le répertoire tmp, et que l'on nommera exercice1.c».

Cette syntaxe n'est pas dépendante du répertoire de travail. L'utilisateur dupont peut tout à fait la taper à partir d'un autre répertoire que /home/dupont ou /tmp.

Remarque: L'utilisation de la syntaxe absolue dans une ligne de commandes rend son résultat indépendant du répertoire de travail. Quel qu'il soit, la commande s'exécutera correctement dès lors que le chemin des arguments est correct.

Le chemin relatif

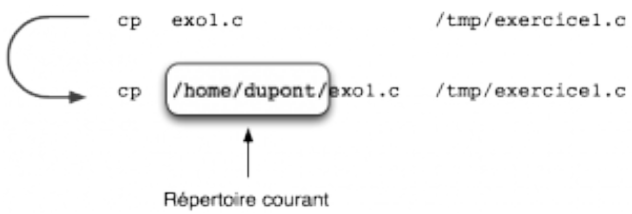
L'inconvénient principale de la syntaxe absolue est sa longueur, puisqu'il faut indiquer tout le chemin à partir de la racine. Le chemin relatif (ou syntaxe relative), lui, précise l'itinéraire « à partir » du répertoire de travail. La longueur des lignes de commandes est réduite car on utilise des chemin plus courts pour nommer les fichiers ou les répertoires. Si l'utilisateur dupont se trouve dans son répertoire personnel (/home/dupont), la syntaxe précédente se réécrit:

```
$ cp exo1.c /tmp/exercice1.c
```

Une lecture « litérale » de cette commande donne:

« Copier le fichier dont le nom est exo1.c (localisé dans le répertoire de travail) vers le fichier cible qui se trouve, à partir de la racine, dans le répertoire /tmp et que l'on nommera exercice1.c ».

Dans cet exemple, le fichier à copier (exo1.c) est indiqué en syntaxe relative alors que le fichier exercice1.c est indiqué en syntaxe absolue. Cette notation n'est qu'une facilité syntaxique proposée à l'utilisateur car le système « reconstruit » le chemin absolu du fichier donnée en relatif. Pour cela, il complète le nom en ajoutant le chemin du *répertoire courant* à son début (cf. figure ci-dessous).



Attention: Contrairement à la syntaxe absolue, la syntaxe relative est dépendante du répertoire de travail. Elle peut échouer si l'utilisateur se ne se trouve pas dans le « bon » répertoire, même avec une syntaxe correcte. Si l'utilisateur `dupont` table la commande précédente à parti du répertoire `/home`, qui ne contient pas le fichier `exo1.c`, le système va « reconstituer » une syntaxe absolue erronée:

```
$ cp /home/exo1.c /tmp/exercice1.c
```

Ce qui provoque une erreur car le fichier `/home/exo1.c` n'existe pas !

Remarque: Pour savoir si la syntaxe est absolue ou relative, le système UNIX applique la règle suivante: dès que le chemin commande par le caractère « / », il est en absolu, sinon c'est un chemin relatif.

L'ajout intempestif du caractère « / » au début du nom d'un fichier transforme une syntaxe relative en syntaxe absolue. Dans la ligne de commande suivante, le système considére que le fichier `exo1.c` est donnée en absolu:

```
$ cp /exo1.c /tmp/exercice1.c
```

Le système n'ajoute pas le nom du répertoire de travail. Si aucun fichier `exo1.c` n'existe sous la racine, cette commande échoue.

Les notations « . » et « .. »

Tous les répertoire contiennent deux « sous-répertoires » particuliers : le répertoire « . » et le répertoire « .. ».

La notation « . » désigne le répertoire de travail, et « .. » son père. Ces deux notations simplifient la syntaxe en évitant d'écrire des noms relatifs ou absolus. Ces deux « sous-répertoire » sont présents dès la création du répertoire qui les contient. Seule la racine déroge à cette règle et ne possède aucun répertoire père « .. ».

La notation « .. ». Cette notation est souvent utilisée avec la commande `cd` (*change directory*), qui effectue des déplacements dans la hiérarchie. La syntaxe suivante accomplit un déplacement dans le répertoire « du dessus »Û, ce qui fait remonter d'un niveau dans l'arborescence.

```
$ cd ..
```

Ces autres commandes réalisent deux déplacements successifs:

```
$ cd ..
$ cd ..
```

Elles peuvent être réécrites en une seule ligne qui fait remonter de deux niveaux:

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

La notation « . ». Cette notation n'est pas utilisée avec la commande `cd` (ce déboucherait sur un déplacement dans le répertoire où se trouve déjà l'utilisateur), mais avec des commandes comme la copie. La ligne suivante copie le fichier `essai` présent dans le répertoire `/tmp` vers le répertoire de travail (répertoire

dans lequel se trouve l'utilisateur au moment où il tape la commande):

```
$ cp /tmp/essai .
```

Cette notation trouve aussi son utilité lors de l'exécution d'un programme situé dans le répertoire de travail. Si l'environnement de travail est mal configuré (en particulier la variable PATH), la simple saisie du nom de l'exécutable ne suffit pas. Le systèmes indique qu'il ne trouve pas cette commande:

```
$ exo1
bash: exo1: command not found
```

Ce problème est résolu en indiquant le chemin absolu du fichier exécutable:

```
$ /home/dupont/exo1
Hello world
```

Si l'utilisateur se trouve dans le répertoire où se situe l'exécutable, la syntaxe devient:

```
$ ./exo1
Hello world
```

La notation « ~ ». Les interpréteurs de commandes csh, ksh, bash et d'une manière générale, les shells POSIX, proposent un raccourci pour atteindre le *répertoire de login* d'un utilisateur. Cette notation utilise le caractère, « ~ », suivi du nom de login. La syntaxe ~dupontremplace ainsi /home/dupont. La commande suivante affiche le contenu du répertoire personne de l'utilisateur *dupont*:

```
$ ls ~dupont
```

Attention: Cette notation ne s'applique qu'aux répertoires de login utilisateurs. Ainsi, ~tmp n'a aucun sens, sauf s'existait un utilisateur *tmp* !

En bash et lsh, le tilde seul indique le répertoire de login de l'utilisateur qui tape la commande.

```
$ cd /tmp
$ pwd
/tmp
$ cd ~
$ pwd
/home/dupont
```

Le title « ~ » suivi du signe « − », indique le répertoire précédent dans l'ordre des déplacements. En se déplaçant deux fois avec cette syntaxe, on revient au répertoire initial.

```
$ cd /tmp
$ cd /home/dupont
$ pwd
/home/dupont
$ cd ~-
$ pwd
/tmp
$ cd ~-
/home/dupont
```

Cette syntaxe est équivalente à « cd -. ». Enfin, la syntaxe « ~+ » indique le chemin absolu du répertoire courant.

Exercice 2.5 (Relatif et absolu)

Dans le tableau ci-dessous, mettez en correspondance la syntaxe relative et la syntaxe absolue. Le répertoire de travail supposé est: `/home/dupont`.

Chemin relatif	Chemin absolu
`list.txt`	
	`/home/dupont/exo`
`Rep/trace`	
	`/home/dupont`
`..`	
	`/home/martin/prog.c`
`../../tmp`	

Exercice 2.6 (L'arborescence)

À l'aide des commandes `cd` et `ls` donner l'arborescence de votre machine UNIX à une profondeur de 2 sous-repertoires en partant de la racine « / ». Si le nombre de fichiers ou de répertoires est supérieur à 20 indiquez uniquement le nombre de fichiers ou de sous-répertoires.

Les commandes de base

Les commandes UNIX permettent, notamment, de gérer des fichiers et des répertoires, d'afficher des informations sur l'activité du serveur, de gérer les processus, d'administrer les services réseaux, etc. Elles sont nombreuses et leurs domaines d'applications sont variés. Elles possèdent, néanmoins, une syntaxe commune:

1. Toutes les commandes systèmes sont écrites en *minuscules*. Le système UNIX différencie les majuscules et les minuscules. Cette caractéristique, déjà présentée dans le cas des noms de fichiers et de répertoires, s'applique également aux commandes. La saisie d'une commande en majuscules provoque une erreur. Le système Mac OS X, qui s'appuie sur UNIX, est une des rares exceptions à cette règle.
2. Elles utilisent une syntaxe basée sur des *options* et des *arguments*.

Syntaxe d'une ligne de commandes

L'interface historique du système UNIX est le mode « ligne de commandes ». Les administrateur privilégient toujours la saisie de commandes au détriments de l'interface graphique. Dans cette partie, nous allons nous intéresser à la syntaxe générale d'une ligne de commandes , ainsi que les notions d'options et d'argument.

Forme générale

Une ligne de commandes est constituée par:

- une *commande* ;
- une liste *d'options* facultatives généralement précédées pat e caractère « − » ;
- une liste *d'arguments* facultatifs sur lesquels porte la commande.

Elle ressemble généralement à ceci:

```
cmd -option1 -option2 ... argument1 argument2 ...
```

Où `cmd` est la commande, `option1`, `option2`, etc., la liste des options, et `argument1`, `argument2`,

etc., la liste des arguments. Les différents éléments sont séparés entre eux par le caractère espace. Voici quatre exemples présentant des nombres d'options et d'arguments différents:

```
$ ls                            # pas d'option, pas d'argument
$ ls -l                         # une option, pas d'argument
$ ls /tmp                       # pas d'option, un argument
$ ls -a -l /tmp /home /var      # deux options et trois arguments
```

Attention: Il ne faut pas oublier l'espace entre la commande, les options et les arguments. Un oubli provoquerait une erreur, comme ci-dessous:

```
$ ls-l
bash: ls-l command not found
```

Ici, le système cherche la commande `ls-l` qu'il ne trouve pas.

Les options

L'option modifie le fonctionnement de la commande. Par exemple, la commande `ls`, sans option, liste le contenu d'un répertoire. Avec l'option `-l`, elle fournit plud d'informations, sur chaque objet, et avec l'option `-R`, elle se poursuit récursivement dans les sous-répertoires.

Les options sont généralement définies par une lettre (`l`, `R`, `a`, `i`, etc.) dont la signification varie selon la commande: l'option `-l` de la commande `wc` affiche ainsi le nombre de « lignes », alors que celle de la commande `ls` précise que l'information doit être plus « longue ». Les options sont chacune précédées par un tiret « - », ou bien regroupées derrière:

```
$ ls -a -i -l -R /tmp
```

La ligne précédente peut s'écrire:

```
$ ls -ailR /tmp
```

Certaines commandes acceptent les options sans préfixe « - », comme la commande `tar`, qui gère les fichiers d'« archive »:

```
$ tar cvf /tmp/rep.tar .
```

Par ailleurs, certaines options peuvent recevoir des arguments. Dans cet exemple, la commande `tar` (*tape archiver*) utilise les options `c` (*create*), `v` (*verbose*) et `f` (*filename*). L'option `f` utilise l'argument `/tmp/rep.tar`. L'argument de la commande est « `.` » (le répertoire courant). Cette commande crée un fichier archive, `/tmp/rep.tar`, qui contient toute la hiérarchie à partir du répertoire de travail.

Le système Linux autorise des options possédant un libellé plus long. Elles sont alors précédées par « - », et ont généralement une version courte associée:

```
$ ls --all
```

est équivalent à:

```
$ ls -a
```

L'option longue `--help` permet d'obtenir de l'aide sur l'utilisation de la commande.

Les arguments

Les arguments sont les noms des fichiers ou des répertoires traités par la commande. Chacun d'eux peut être spécifié en absolue ou en relatif. Leur type (fichier ou répertoire) est imposé par la commande. Ainsi, avec la *copie de fichiers* (cp), il est obligatoire que les deux arguments fournis soient des fichiers. Il est parfois possible de fournir le nom d'un répertoire à la place du fichier attendu (ou l'inverse). Si le répertoires existe, la commande s'adapte et « comprend » l'astuce syntaxique. Dans la cas contraire, elle considère que tous les éléments sont du type imposé, ce qui peut provoquer une erreur de traitement, comme dans l'exemple ci-dessous.

La première commande copie le fichier essai1 vers le fichier essai2, qui se troubve dans /tmp. Si essai2 n'existe pas, il est créé. Sinon, son contenu est écrasé.

```
$ cp essai1 /tmp/essai2
```

La deuxième commande copie le fichier essai1 dans /tmp. /tmp existe, et c'est un répertoire. La commande « comprend » donc qu'il faut copier le fichier essai1 dans le répertoire /tmp, en conservant son nom d'origine.

```
$ cp essai1 /tmp
```

La troisième commande copie le fichier essai1 dans /temp. /temp n'existe pas, il sera créé en tant que fichier. IL s'agit là d'une faute de frappe qui provoque une erreur d'interprétation du type de l'argument /temp.

```
$ cp essai1 /temp
```

Attention: La nature des arguments est définie par la commande. L'utilisateur doit veiller à fournir des noms de fichiers aux commandes qui gèrent des fichiers, et des noms de répertoires aux commandes qui gèrent des répertoires. Dans le cas contraire, l'objet doit exister afin d'obliger la commande à s'adapter.

L'ensemble des commandes

Le tableau ci-dessous présent les commandes usuelles de gestion de fichiers qu'il est nécessaire de connaître. Leur syntaxe et leurs options particulières sont étudiées par la suite.

Action	Fichier	Répertoire
Afficher le contenu	cat ou more	ls
Copier	cp	cp -R ou cp -r
Changer de nom	mv	mv
Se déplacer		cd
Créer		mkdir
Détruire	rm	rmdir ou rm -r
Déplacer	mv	mv
Affichier le répertoire courant		pwd

Ce tableau amène quelques commentaires:

- *Il n'existe pas de commandes de création de fichier*. Le rôle d'un fichier est de conserver, sur disque, des informations qui sont traitées en mémoire. Il est créé, lors de la sauvegarde des données, par le logiciel applicatif. Le système UNIX ne prévoit aucune commande spécifique à la création de fichier. Cependant, il

est possible d'obtenir ce résultat en détournant certaines commandes de leur rôle initial. Par exemple, les deux syntaxes suivantes créent le fichier vide `liste`:

```
$ cp /dev/null liste
$ touch liste
```

- *Certaines actions utilisent la même commande pour les fichiers et pour les répertoires, à l'option près* (« `-r` »). Les commande `cp` et `rm` possèdent une option récursive (`-r`) qui poursuit l'action dans les sous-répertoires. Elle étend le traitement des fichiers, aux répertoires.
- *Il y a une différence entre déplacer un objet et « se » déplacer dans un objet*. La commande `mv` (*move*) déplace un fichier ou un répertoire à un autre endroit de l'arborescence. Par analogie avec une personne travaillant à son bureau, cette action correspond au déplacement d'une feuille de papier (fichier) ou d'un classeur (répertoire) dans un autre tiroir (répertoire) du bureau. Cette personne ne se déplace pas pour effectuer cette opération. En revanche, la commande `cd` permet de se déplacer ou de naviguer dans le répertoire de travail. Si l'on reprend notre analogie, la personne se lève de sa chaise pour aller à un autre bureau. Cette commande n'a aucun sens pour un fichier et ne modifie pas l'arborescence.

La gestion des fichiers

Nous allons nous intéresser dans cette partie aux commandes indiquées dans le tableau précédent et présentons pour chacune d'elles les options les plus importantes. La liste complète des options est obtenue par la commande `man`, qui affiche le manuel de la commande donnée en argument. La syntaxe suivante fait apparaître le manuel de la commande `ls`:

```
$ man ls
```

Afficher le contenu d'un fichier

Pour afficher le contenu d'un fichier, deux commandes sont généralement utilisées:

1. la commande `cat`;
2. la commande `more`.

La commande « cat »

Le terme `cat` est une abréviation de *concatenate*. Cette commande affiche sur l'écran le contenu des fichiers texte donnés en arguments. Si les fichiers sont binaires, i.e., des exécutables, chaque octet est interprété comme un caractère. Dans notre premier exemple, le contenu du fichier `list.txt` est affiché:

```
$ cat list.txt
Pierre, Paul, Jacques
```

L'exemple suivant affihce le contenu de trois fichiers. Chaque affichage est concaténé avec le suivant sans séparation:

```
$ cat list.txt list2010 list2009
Pierre, Paul, Jacques
Dupont, Martin, Kaczma
Linux, AIX, Window, MacOsX
```

Si la commande `cat` est employée sans argument, elle affiche le contenu du fichier « entrée standard » qui pointe sur le clavier. Chaque mot tapé (ci-après le texte « saisie ») est affiché en écho. Pour terminer cette commande, il faut saisir le caractère de fin de fichier, CTRL+D (on appuie sur les tauches « contrôle » et « d

»).

```
$ cat
saisie
saisie
^D
```

Dans la syntaxe suivante, la commande `cat` n'a toujours pas d'argument, ce qui provoque l'écho de la frappe. La syntaxe `> liste` redirige l'affichage dans le fichier `liste`. Cette astuce permet de saisir un fichier directement à partir du clavier, sans passer par un éditeur de texte.

```
$ cat > list.txt
nouveau contenu
^D
$cat list.txt
nouveau contenu
```

Les options de la commande `cat` sont très peu utilisées.

Exercice 2.7 (Les courses)

1. Créer trois fichiers `fruits`, `legumes` et `viandes` ayant respectivement pour contenu “pomme, poire, prune”, “citrouille, courgette, cornichon”, et “magret, poulet, boeuf”.
2. Créer un fichier « `liste` » à partir des fichiers précédents contenant les trois listes ci-dessus.
3. Afficher le contenu du fichier `liste`.

La commande « more »

La commande `more` affiche le contenu d'un fichier *page à page*, alors que la commande `cat` affiche la totalité en une seule fois. L'utilisateur peut contrôler le défilement avec les touches suivantes:

- La touche ENTRÉE fait progresser l'affichage *ligne par ligne*.
- La bare d'espacement fait progresser l'affichage *page par page*.
- La lettre « q » arrête l'affichage.

```
$ more liste12
List12
List12
...
List12 34%
```

Cette commande est également utilisée, à la suite d'une autre commande, pour en contrôler l'affichage. Le caractère d'enchaînement est le « | » (prononcer « païpe », pour « pipe » en anglais). La syntaxe précédente peut se récrire:

```
$ cat list12 | more
```

L'exemple suivant montre comment afficher, page par page, le texte retournée pat la commande `who`.

```
$ who | more
```

Les options de la commande `more` sont très peu utilisées.

Exercice 2.8 (Page à page)

Placer vous dans votre répertoire personnel et remonter d'un niveau dans l'arborescence puis afficher page à page la liste des répertoires utilisateurs. Combien y a t'il de répertoires utilisateur ?

Copier, renommer, déplacer et détruire un fichier

Nous allons étudier les commandes `cp`, `mv`, et `rm`.

La commande « cp »

Le terme `cp` est l'abréviation de *copy*. Cette commande copie le contenu du premier fichier dans le second. Le premier fichier doit exister. Si le second n'existe pas, il est créé. Sinon, son contenu est écrasé.

```
$ cp list1 list2
```

Si le deuxième argument est un nom de répertoire existant, la commande réagit différemment. Elle copie le premier fichier dans le répertoire indiqué en conservant le même nom.

```
$ cp list1 /tmp
```

Cette commande équivaut à:

```
$ cp list1 /tmp/list1
```

Les options importantes de la commande `cp` sont :

- `-p` Cette lettre est l'abréviation de *permit*. Elle copie également les attributs du fichier d'origine. Le fichier cible possède le même propriétaire, les mêmes droits et les mêmes dates de modifications.
- `-r` ou `-R` Ces deux options copient *récursivement* les répertoires et leurs fichiers. Nous les verrons en détail avec les commandes de gestion des répertoires.

La ligne de commandes suivante copie `liste.txt` dans le répertoire `/tmp` sous le nom `liste.auve`, en conservant les attributs:

```
$ cp -p liste.txt /tmp/liste.sauve
```

La commande « mv »

Le terme `mv` est l'abréviation de *move*. Cette commande renomme un fichier ou le déplace dans la hiérarchie, selon le chemin indiqué pour les deux fichiers donnée en arguments. S'ils sont dans le même répertoire, la commande renomme le premier avec ke nom du second:

```
$ mv liste.txt liste.txt.sauve
$ mv /home/dupont/exo1.c /home/dupont/exercice.c
```

Si les deux fichiers se trouvent dans des répertoires différents, la commande déplace le premier dans le second:

```
$ mv /hime/dupont/liste.txt /tmp/liste.txt.sauve
```

Cette commande se comporte comme `cp` si le second argument est un répertoire existant. Le fichier initial est déplacé dans le répertoire cible, il conserve son nom. La commande précédente se récrit:

```
$ mv /home/dupont/liste.txt.sauve /tmp
```

Remarque: Il n'y a aucun déplacement de données quand le fichier d'origine et le fichier cible sont dans le même système de fichier, mais juste un déplacement de la référence du nom dans le nouveau répertoire. Si le fichier cible est dans un système de fichier différent, il y a déplacement des données. Il se fait en deux temps: la commande copie d'arbord le fichier, puis le détruit.

La commande « rm »

Le terme `rm` est l'abréviation de *remove*. Cette commande détruit **définitivement** les fichiers donnés en arguments. Il n'y a pas de commande « unremove » permettant d'annuler l'opération. L'exemple suivant détruit les trois fichiers `liste1`, `liste2` et `liste3` localisés dans des répertoires différents:

```
$ rm ./liste1 /tmp/liste2 /home/dupont/liste3
```

Les options importantes de la commande `rm` sont:

- `-i` Cette option *interroge* l'utilisateur et lui demande confirmation de la destruction. La réponse est « y » pour « yes » ou « o » pour oui si la langue sélectionnées et le français et « n » pour « no » (non).

```
$ rm -i essai
remove essai ? y
```

- `-f` Cette option *force* la destruction des fichiers protégés en écriture. L'exemple suivant montre que la commande sans l'option demande la confirmation. Avec l'option `-f`, la destruction est immédiate.

```
$ rm prog
rm : detruire un fichier protege en ecriture 'prog' ?
$ rm -f prog
```

- `-r` ou `-R` Ces deux options détruisent les répertoires et leurs fichiers. Nous les verrons en détail avec les commandes de gestion des répertoires.

Astuce: La syntaxe `rm -i *` permet de détruire les fichiers dont le nom contient des caractères qui perturbent l'interprétation de la ligne de commande, comme l'espace ou des caractères de contrôle (cf. Les caractères spéciaux). Le caractère jocker, « * », liste tous les fichiers, et l'option `-i` demande confirmation avant chaque destruction. Il suffit de répondre « n » (non) à tout, sauf à la question qui porte sur le fichier à détruire. Pour arrêter la commande quand la destruction du fichier en question est effectuée, il suffit d'appuyer sur CTRL+C.

Exercice 2.9 (Manipuler un fichier)

1. Créez un fichier `fichier` contenant une ou deux phrases.
2. Visualisez le contenu de `fichier` sans l'éditer.
3. Quelle est la taille de `fichier` en octets ?
4. Faites une copie de `fichier` appelée double.
5. Comparez leurs tailles.
6. Renommez double en introduction.
7. Quelle différence y a-t-il entre

```
$ mv double introduction
```

et

```
$ cp double introduction
```

La gestion des répertoires

Les répertoires permettent d'organiser la hiérarchie en la structurant, et en définissant différents niveaux de l'arborescence. Apprendre à gérer les répertoires, c'est apprendre à organiser son espace disque.

Afficher le contenu d'un répertoire

Pour afficher le contenu d'un répertoire, on se sert de la commande `ls`. Cette commande est l'abréviation de *list*. Elle liste le contenu des répertoires donnés en arguments et affiche le nom de tous les objets (fichiers, sous-répertoires, liens, etc.) qu'ils contiennent.

```
$ ls /tmp
cours_tmp                       TMP_23412                       Traces
```

Si aucun argument n'est précisé, la commande liste le contenu du répertoires de travail:

```
$ ls
CoursUNIX                       cours_1                       exo.C.2010                       exo.UNIX.2010
```

Si l'argument n'est pas un répertoire mais un fichier, son nom est simplement affiché. On a souvent recours à cette syntaxe pour confirmer l'existence d'un fichier.

```
$ ls cours_1
cours_1
```

Utilisée avec des arguments contenant le signe « * », la commande `ls` permet d'afficher la liste de tous les objets (fichier ou répertoires) dont le nom correspond au critère énoncé. La syntaxe suivant affiche tous les objets se trouvant dans le répertoire `/tmp` et dont le nom commence par « T ».

```
$ ls /tmp/T*
/tmp/TMP_23412
/tmp/Trace:
dump.log                      liste.20100111
```

L'affichage montre que le répertoire `/tmp` contient un fichier `/tmp/TMP_23412` est un répertoire `Traces`, dont le contenu est listé. SI cette denière syntaxe permet de différencier les fichiers des répertoires, ce n'est pas le cas avec la syntaxe plus courante.

La commande `ls` dispose de nombreuses options qui fournissent plus de renseignements sur ses arguments. Les principales sont détaillées dans la suite.

L'option « -l ». Cette option fournit une information « long ». Les attributs des fichiers et des répertoires sont affichés. L'affichage commence par une ligne qui indique la taille totale des fichiers de la liste, en blocs de 512 octets.

```
$ ls -l /tmp
total 16
drwx------   3   pellier   enseignant   102   11   jan   19:10   cours_tmp
-rw-r--r--   1   dupont    licence1      15   11   jan   23:16   TMP_23412
drwxr-xr-x   4   dupont    licence1     136   11   jan   23:05   Traces
...
```

La figure ci-dessous détaille les informations affichées par l'option `-i`:

Type	Droits	Nombre de références	Propriétaire	Groupe Propriétaire	Taille en octets	Date et heure de modification	Nom
d	rwx-----	3	pellier	enseignant	102	11 jan 19:10	cours_tmp
-	rw-r--r--	1	dupont	licence1	15	11 jan 23:16	TMP_23412
d	rwxr-xr-x	4	dupont	licence1	136	11 jan 23:05	Traces

Voici l'interprétation des différentes colonnes:

- Type : cette colonne précise le type de l'objet. La liste complète est présentée dans le tableau ci-dessous:

Codage	Type de l'objet
-	Fichier
d	Répertoire
l	Lien symbolique
b	Périphérique « bloc » (disques)
c	Périphérique « caractère » (terminal)
s	Canal de communication, « socket » (réseaux)
p	Pile FIFO (système)

Dans notre exemple `Traces` et `cours_tmp` sont des répertoires et `TMP_23412` est un fichier.

- Droits: ce groupe est constitué de 3 triplet (`rwx r-x r-x`). Il indique les droits pour:
  - le propritétaire (premier triplet, `rwx`);
  - le groupe propriétaire (second triplet, `r-x`);
  - les autres utilisateurs, qui ne sont ni le propriétaire ni des membres du groupe (troisième triplet, `r-x`).

Les lettres « r », « w » et « x » correspond à *read*, *write* et *execute*. Le « - » représente l'absence de droit. Ainsi l'utilisateur `dupont` a les droits de lecture et d'écriture sur le fichier `TMP_23412`; les membres du groupe `licence1` ont juste le droit de le lire, de même que les autres utilisateurs. Nous reviendrons sur la gestion des droits dans le prochain TP.

- Nombre de références: nombre de fois où l'objet est désigné par un nom. La valeur par défaut pour un fichier est 1. Le fichier est référencé par son nom. Cette valeur est augmentée par le nombre de liens physiques qui le pointent. La valeur par défaut pour un répertoire est 2. Il est référencé par son nom et par le sous-répertoire « . ». Cette valeur est augmentée par le nombre d'éléments qu'il contient. Le répertoire `Traces` présenté dans l'exemple contient deux fichiers.
- Propriétaire: nom de login du propriétaire. Si celui-ci n'existe plus (utilisateur dont le compte a été détruit), son nom est remplacé par son UID (*user ID*).
- Groupe propriétaire: nom du groupe propriétaire. Si celui-ci n'existe plus (groupe détruit) son nom est replacé par son (GID) (*group ID*).
- Taille: elle est indiquée en octets
- Date et heure de modification: si la date est antérieure ou postérieure de 6 mois à la date actuelle, l'année remplace l'heure.
- Nom.

Remarque: Les attributs d'un fichier ne contiennent pas le nom de login et le nom du groupe, mais l'UID de son propriétaire ey le GID de son groupe. À l'affichage de ces informations, l'UID est remplacé par le nom de login trouvé dans le fichier `/etc/passwd` et le GID par le nom du groupe trouvé dans le fichier `/etc/group`. Si aucune correspondance n'est trouvée, les numéro s'affichent.

L'option « -a ». Cette option affiche tous les objets, y compris ceux dont le nom commence par le caractère

« . » (point) et qui ne sont pas listés sans cette option. Ils sont considérés comme « cachés ». Nombre de fichiers de « démarrage » on un nom qui commence par « . ».

```
$ ls -a /tmp
.          ..          cours_tmps          TMP_23412          Traces
```

L'option « -i ». Cette option affiche le numéro *inode* devant le nom du fichier ou du répertoire. Chaque objet est repéré dans le système de fichier (*file system*) par un numéro unique, son inode. Cette option est surtout utile aux administrateurs. Elle permet de détecter les liens physiques.

```
$ ls -i /tmp
757502  cours_tmp          757698  TMP_23412          757689 Traces
```

L'option « -R ». Cette option parcourt les sous-répertoires récurcivement et affiche leur contenu.

```
ls -R /tmp
cours_tmp          TMP_23412         Traces
/tmp/cours_tmp:
plan.txt
/tmp/Traces:
dump.log          liste20100111
```

D'autre options, moins importantes que les précédentes, peuvent présenter une certaine utilité. En voici quatre exemples:

1. **« -b »** Il arrive parfois que des commandes erronées produisent des fichiers dont le nom contient des caractères non visibles. Cette option les affiche sous la forme de leur code ASCII précédé de « \ ».
2. **« -d »** Les répertoires sont présentés comme les fichiers, sans que leur contenu soit affiché.
3. **« -s »** La taille du fichier est affichée sous la forme du nombre de blocs de 512 ou 1024 octets occupés dans le système de fichiers.
4. **« -F »** Le nom de chaque objet affiché est suivi par un caractère qui indique son type. Le caractère « \ » précise que c'est un répertoire, « * » définit un fichier exécutable, « @ » suit le nom d'un lien symbolique et le « | » définit une pile FIFO (*First In First Out*). Dans l'exemple ci-dessous, `R2` est un répertoire, `prog` un exécutable, `prog.c` un fichier de texte et `prog2` un lien:

```
$ ls -F
prog* prog2@ prog.c R2/
```

La commande suivante affiche le contenu du répertoire `/tmp` avec les options `-ail`:

```
$ ls -ail /tmp
total 16
757456 drwxrwxrwt 6 root    root       204 11  jan 23:16 .
  8322 drwxr-xr-t 5 root    root       170 11  jan 23:07 ..
757502 drwx------ 3 pellier enseignant 102 11  jan 19:10 cours_tmp
757698 -rw-r--r-- 1 dupont  licence1   15  11  jan 23:16 TMP_23412
757689 drwxr-xr-x 4 dupont  licence1   136 11  jan 23:05 Traces
...
```

L'option « -i » affiche la première colonne de chiffres (les inodes), l'option « -a » affiche les répertoires « . » et « .. », l'option « -l » affiche les attributs (droits, nombre de références, propriétaire, groupe propriétaire, date et heure).

Copier, renommer et déplacer un répertoire

Nous allons étudier les commandes `cp -R` et `mv`.

La commande « cp -r » ou « cp -R »

La commande `cp`, avec les options `-R` et `-r`, copie récursivement les répertoires et leur contenu. Le premier répertoire doit exister. Si le second n'existe pas, il est crée.

```
$ cp -R Rep1 Rep2
```

Si le deuxième argument est un répertoire existant, la commande copie le premier répertoire dans le second en conservant le même nom.

```
$ cp -R Rep1 /tmp
```

équivaut à:

```
$ cp -R Rep1 /tmp/Rep1
```

Les options `-R` et `-r` traitent les fichiers spéciaux (liens symboliques, piles, etc.) différemment. Le comportement de la commande varie selon les systèmes UNIX.

L'option `-R` copie le répertoire indiqué et conserve les fichiers spéciaux à l'identique, dans le répertoire cible. La commande suivante affiche le contenu du répertoire `Rep1`: le lien symbolique `liste2.txt` (caractère « 1 » en première colonne) pointe vers `liste.txt` - un lien symbolique est équivalent à un « raccourci » vers un fichier.

```
$ ls -l Rep1
total 16
-rwr------ 1 dupont licence1 15 23 Jan 15:54 liste.txt
lrwxr-x--- 1 dupont licence1  9 23 Jan 15:54 liste2.txt -> liste.txt
```

la commande `ls` confirme que l'option `-R` a conservé les liens, dans le répertoire cible.

```
$ ls -l Rep2
total 16
-rwr------ 1 dupont licence1 15 23 Jan 15:55 liste.txt
lrwxr-x--- 1 dupont licence1  9 23 Jan 15:55 liste2.txt -> liste.txt
```

Avec l'option « -r » les fichiers spéciaux sont remplacés par des fichiers ordinaires. Le répertoire cible contient une copie du fichier à la place du lien symbolique.

```
$ cp -r Rep1 Rep3
```

la commande `ls` confirme que l'option `-r` remplace le lien symbolique par un fichier ordinaire:

```
$ ls -l Rep3
total 16
-rwr------ 1 dupont licence1 15 23 Jan 16:45 liste.txt
lrwxr-x--- 1 dupont licence1 15 23 Jan 16:45 liste2.txt
```

Attention: Avec les options `-R` et `-r`, la commande `cp` peut copier toute une hiérarchie. L'utilisateur doit veiller à éviter la saturation de l'espace disque en utilisant les *file systems* ou les « quotas de disques». L'exemple suivant montre qu'une commande mal contrôlée peut saturer totalement le disque dur. Cette syntaxe provoque une copie récursive infini, en copiant le répertoire « père » dans le répertoire de travail:

```
$ cp -r .. .
```

Le répertoire père contient, comme sous-répertoire, le répertoire de travail actuel et les autres répertoires situés au même niveau. Tous ces « sous-répertoires » sont copiés dans le répertoire de travail qui se remplit

au fur et à mesure, ouis qui se copie dans lui-même. Cette dernière action reproduit le mécanisme précédent, et ainsi de suite, sans jamais se terminer. Cette commande s'arrête quand le disque est plein !

La commande « mv »

La commande mv se comporte de manière identique avec les répertoires et les fichiers. Il a changement de nom si les deux répertoires se trouvent au même endroit, et il y a déplacement s'ils sont à des endroits différents dans la hiérarchie. Cela peut prendre du temps si le répertoire contient une arborescence importante.

Notre premier exemple renomme Rep1 en Rep2:

```
$ mv Rep1 Rep2
```

Attention: Pour que Rep2 soit renommé, le répertoire Rep2 ne doit pas exister. S'il existe, la commande déplace Rep1 dans Rep2 devient un sous-répertoire de Rep2.

Le deuxième exemple déplace Rep1 dans /tmp en changeant son nom:

```
$ mv /home/dupont/Rep1 /tmp/Rep2
```

Se déplacer dans un répertoire

Les deux commandes que nous allons considérer ici sont les commandes pwd et cd.

La commande « pwd »

La commande pwd est l'abréviation de *print working directory*. Elle affiche le nom du répertoire de travail (ou répertoire courant).

```
$ pwd
/tmp
```

Elle facilite la navigation dans l'arborescence du système de fichier en permettant de savoir exactement où l'on se trouve. Il est possible de configurer l'invite de commande (le « prompt ») pour afficher le répertoire de travail en permanence; cela permet de limiter l'usage de cette commande mais peut finir par nuire à la lisibilité de la console (notamment lorsque l'on se trouve dans les profondeurs de l'arborescence).

La commande « cd »

Le terme cd est l'abréviation de *change directory*. Cette commande déplace l'utilisateur dans un nouveau répertoire de travail. Par exemple, la commande suivante effectue un déplacement dans le répertoire /tmp:

```
$ cd /tmp
```

Le déplacement dans la hiérarchie a une incidence sur la syntaxe relative car le répertoire de travail change. Utilisée sans argument, la commande cd permet de revenir au répertoire de login, comme dans l'exemple suivant:

Le répertoire courant est /tmp.

```
$ pwd
/tmp
```

La commande cd n'utilise aucun argument.

```
$ cd
```

Après cette saisie, on affiche le répertoire de travail : l'utilisateur se retrouve dans son répertoire personnel.

```
$ pwd
/home/dupont
```

Remarque: Utilisée sans argument, la commande cd réalise un déplacement dans le répertoire indiqué par la variable d'environnement HOME. Cette variable contient par défaut le nom du répertoire de login.

Durant une session de travail, il est courant de se déplacer d'un répertoire à un autre et de revenir immédiatement au précédent. La syntaxe suivante permet de revenir au répertoire précédent sans en préciser le nom:

```
$ cd -
```

Deux déplacements successifs avec cette syntaxe font revenir au répertoire de départ.

Créer et détruire un répertoire

Nous allons nous intéressé ici au commande mkdir, rmdir et rm -r.

La commande « mkdir »

La commande « mkdir » est l'abréviation de *make directory*. Cette commande crée un répertoire vide, qui contient systématiquement les répertoires « . » et « .. ». Les principales options sont les suivantes:

- **-m** Cette option indique les droits à positionner lors de la création du répertoire. Les droits sur les fichiers et les répertoires seront abordés dans le TP n°3: Gestion des fichiers avancées.

```
$ mkdir -m "a=rx" /tmp/Doc
```

- **-p** Quand un répertoire est créer, son répertoire père doit exister. Cette option permet de lever cette contrainte en créant tous les répertoires intermédiaires qui manquent. Dans l'exemple qui suit, les répertoires R1, R2 et R3 n'existe pas. La première syntaxe échoue, le message précise que R1 et R2 ne sont pas trouvés:

```
$ mkdir /tmp/R1/R2/R3
mkdir: /tmp/R1/R2: No such file or directory
```

L'option -p crée la hiérarchie manquante, c'est à dire R1 et R2, avant R3:

```
$ mkdir -p /tmp/R1/R2/R3
```

Exercice 2.10 (Manipuler une arborescence)

- Dans votre répertoire de travail, créer l'arborescence suivante, en n'utilisant que des chemins relatifs et vérifier que l'arborescence que vous avez créée est conforme en utilisant la commande « tree ».

```
.
```

```
 +-- rep1
    +-- fich11
    +-- fich12
    +-- rep2
     |  +-- fich21
     |  +-- fich22
    +-- rep3
        +-- fich31
        +-- fich32
```

- Déplacer toute l'arborescence `rep3` dans `rep2` et vérifier le déplacement à bien eu lieu.
- Supprimer tout sauf `rep1`, `fich11` et `fich12`.
- Renommer `rep1` en `rep1.sauve`

La commande « rmdir »

La commande « rmdir » est l'abréviation de *remove directory*. Cette commande détruit un répertoire vide. Elle échoue si le répertoire contient des fichiers ou des sous-répertoires. Dans l'exemple ci-dessous, la destruction de `R2` provoque une erreur car le répertoire n'est pas vide, il contient `R3`:

```
$ rmdir R2
rmdir: /tmp/R2: Directory not empty
```

La destruction de `R3` et obtenu par:

```
$ rmdir /tmp/R1/R2/R3
```

La principale option, `-p`, permet de détruire tous les sous-répertoires indiqués dans le chemin, en commençant par le dernier, qui est le plus profond sans la hiérarchie, puis en remontant vers le premier. De proche en proche, chaque répertoire est détruit s'il ne contient plus aucun objet. La commande suivante détruit successivement les répertoires `R2`, `r1` puis `/tmp`. La desctrution de `/tmp` échoue car l'utilisateur *dupont* ne possède pas les droits suffisants:

```
$ rmdir -p /tmp/R1/R2
rmdir: '/tmp': Permission denied
```

Remarque: Il n'est pas possible de détruire le répertoire de travail, même vide, car l'utilisateur s'y trouve. Ainsi, la commande suivante ci-dessous échouera systématiquement:

```
$ rmdir .
```

La commande « rm -r »

Nous avons déjà étudié cette commande. Les options `-R` et `-r` sont identiques. Elles permettent de détruire un répertoire avec tout son contenu.

Attention: La destruction récursive est dangereuse. Elle détruit toute la hiérarchie (fichiers et répertoires) contenue dans le répertoire donné en argument, puis le répertoire lui-même. Il faut l'utiliser avec prudence et vérifier la syntaxe avant de valider la commande.

Dans l'exemple donné, le répertoire `R2` contient in fichier `essai` et un répertoire `R3`. La commande `rmdir` échoue après la destruction de `R3`, car le répertoire `R2` contient toujours le fichier `essai` et n'est donc pas vide:

```
$ rmdir -p R1/R2/R3
rmdir: R1/R2: Directory no empty
```

La commande `rm -r` détruit toute la hiérarchie, quel que soit sont contenu.

```
$ rm -r R1
```

Afin de protéger les utilisateus contre une destruction involontaire et potentiellement catastrophique, les administrateurs mettent souvent en place un *alias* « rm » qui se fait passer pour la commande. Lorsque l'utilisateur tape la syntaxe `rm`, il utilise sans le savoir l'alias qui appelle la commande `rm` avec l'option `-i`, ce qui oblige à confirmer chaque destruction:

```
$ rm -r R1
remove R1 ?
```

Si l'utilisateur est certain de vouloir tout détruire, la confirmation de chaque destruction devient très vite fastidieuse et pénible. Pour contourner cette interrogation il suffit d'ajouter l'option `-f`

```
$ rm -rf R1
```

De même qu'avec la commande `rmdir`, il n'est pas possible de détruire le répertoire de travail. Il faut d'abord en sortir par une commande de déplacement (`cd ..`) avant de pouvoir utiliser la commande de destruction.

Exercice 2.11 (Manipuler un répertoire)

Cette exercice est la suite de l'exercice 2.10:

1. Créez un répertoire `essai/`.
2. Déplacez introduction dans `essai/`.
3. Faites une copie de `fichier` appelée `copie`, et placez-la également dans `essai/`.
4. Affichez une liste de ce que contient `essai/`.
5. Essayez de détruire `essai/`. Que se passe-t-il ? Que faut-il faire pour détruire un répertoire ?
6. Détruisez tout ce que contient `essai/`.
7. Détruisez `essai/`.

Exercice 2.12 (Questions de révision)

1. Quelle est la différence entre un nom de fichier et un nom de répertoire ?
2. Pourquoi le nom d'un fichier qui contient des espaces peut-il poser problème ?
3. Qu'est ce que le chemin absolu d'un fichier ?
4. Quel est le problème lié à la syntaxe relative qui n'existe pas pour la syntaxe absolue ?
5. Comment peut-on exécuter un programme si celui-ci ne se lance pas avec son nom seul ?
6. Que se passe-t-il si la commande `ls-l` est saisie sans l'espace séparateur ?
7. Comment faire pour détruire un fichier dont le nom est « liste du personnel ».
8. Comment faire pour détruire un fichier dont le nom contient des caractères « bizarres » ?
9. Quelle est la différence entre la commande `rmdir` et la commande `rm -r` ?
10. Comment faire la différence entre un changement de nom et un déplacement quand on utilise la commande `mv` ?
11. Quel est le rôle de l'option `-p` de la commande `mkdir` ?
12. Quelle est la différence entre les options `-R` et `-r` de la commande `cp` ?

Exercice 2.13 (Synthèse)

Ces questions se suivent et constituent une manipulation de type « session de travail ».

1. Créer un fichier `essai` contenant le texte « essai de texte » saisi au clavier.
2. Afficher le numéro d'inode de ce fichier.
3. Afficher le nom du répertoire de travail.
4. Afficher le contenu du répertoire de travail. L'affichage fera apparaître les fichiers ou répertoires « cachés », les droits et le propriétaire de chaque fichier ou répertoire.
5. Déplacer vous dans le répertoire `/tmp`.
6. Créer dans `/tmp` un sous-répertoire `Sauve` contenant un répertoire `2010`, en une seule ligne de commandes.
7. Retournez dans votre répertoire personnel.
8. Déplacez le fichier `essai` créer à la question 1 vers le répertoire `2010`. Le répertoire `2010` est contenu dans le répertoire `Sauve`, qui se trouve lui-même dans `/tmp`.
9. Déplacer le répertoire `Sauve` et tout son contenu dans votre répertoire personnel.
10. Détruisez le répertoire `Sauve`.
11. Afficher les attributs de votre répertoire personnel.
12. Afficher les attributs de tous les fichiers et répertoires se trouvant à tous les niveaux de votre répertoire personnel.