Le système de fichier sous Linux

INDEXATION DU DOCUMENT

	TITRE :	*REFERENCE* :
	Le système de fichier sous Linux	

ACTION	*NOM*	*DATE*	*SIGNATURE*
RÉDIGÉ PAR	Jean-Noël Avila	8 janvier 2007	

SUIVI DU DOCUMENT

INDICE	DATE	MODIFICATIONS	NOM

Table des matières

Table des figures

Résumé

Tout ce que vous avez toujours voulu savoir sur les fichiers sans jamais oser le demander. Pour Unix, toute ressource (ou presque) est fichier. Et pour faire rentrer dans cette catégorie tout ce que l'on peut trouver sur un système d'exploitation, il a bien fallu différencier les fichiers par type. Mais, ce n'est pas tout : Linux est un système qui n'aime pas mettre ses fichiers n'importe où, et un nouveau standard, le FHS qui permettra une compatibilité entre systèmes est en train de naître.

1 Introduction

TBD

2 Des fichiers et des types

2.1 Le Virtual File System

Qu'est-ce donc que le Virtual File System ? Il s'agit tout simplement d'une couche d'abstraction du système de fichiers sous-jacent de manière à ce que les appels aux fonctions standards d'accès aux fichiers soient transparents aux applications. De cette manière, que les systèmes de fichiers sur votre disque dur soient de la FAT32 (système Microsoft), du ext2fs ou du HPFS, les programmes qui utilisent la fonction `open` accéderont aux fichiers de tous ces systèmes de la même manière.

L'abstraction ne se limite pas aux simples systèmes de fichiers des périphériques locaux, mais peut tout aussi bien rendre abstrait des systèmes appartenant à d'autres ordinateurs du réseau local, voire même montrer de la même manière tout à fait transparente des sites ftp distants. Mais le VFS va plus loin car l'abstraction peut aller jusqu'à mettre sous forme de système de fichiers des paramétrages du noyau.

Le système de fichiers étant organisé comme une arborescence unique de dossiers (il n'y a rien qui s'y oppose car tous les systèmes de fichiers sont virtualisés et peuvent coexister), la manière de rassembler les différents systèmes de fichier consiste à transformer l'accès à un dossier en accès à la racine de l'arborescence d'un autre système de fichiers. Pour plus de clarté, voici une exemple : généralement, les fichiers utiles au tout début du démarrage d'un système Linux sont rassemblés sur une même partition (partie d'un disque dur constituant un système de fichiers à part entière). Cette partition est généralement "montée" dans le répertoire `/boot`.

Exemple 2.1 Changement de système de fichier à la traversée d'un répertoire.

```
$ pwd
/
$ df .
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda2              3598412   2861252    737160  80% / ❶
$ ls
bin/   dev/  home/    lib/  opt/   root/  swap/  usr/ ❷
boot/  etc/  initrd/  mnt/  proc/  sbin/  tmp/   var/
$ cd boot ❸
$ df .
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda1               15522      8727      5994  60% /boot ❹
```

❶ Nous sommes sur le système de fichiers ("Filesystem" vaut `/dev/hda2`) de la racine globale (le champs "Mounted on" indique `/`) telle que les applications la voient.

❷ Sous cette racine, nous pouvons visualiser les répertoires qui ont tous l'air de même nature

❸ Lorsque nous changeons de répertoire, pour aller dans le répertoire `/boot`,

❹ de manière transparente, nous venons de changer le système de fichier que nous sommes en train de visualiser("Filesystem" vaut maintenant `/dev/hda1`) .

NOTE

On peut monter des systèmes de fichiers sur n'importe quel répertoire. A partir de ce moment, l'ancien contenu du répertoire n'est plus visible et on visualise à la place la racine du système de fichier monté, occultant sans effacer les fichiers initialement présents dans le répertoire.

Ce n'est généralement pas un comportement utile (fichiers inaccessibles, place perdue), et on veillera à monter un système de fichiers sur un répertoire vide. Ceci est néanmoins généralement mis en place pour des systèmes de fichiers qui peuvent être utilisés en backup ; par exemple, on l'utilise pour des systèmes NFS : `/usr/bin` est monté avec NFS par dessus un répertoire qui contient le minimum vital au cas où le montage par réseau ne fonctionne pas. Un autre exemple, mis en place depuis l'apparition de la gestion des devices par un module dans le noyau 2.4.x, consiste à monter sur le répertoire `/dev`originel le nouveau type de gestion des devices. De cette manière, même si la fonctionalité n'est pas activée, le système reste utilisable.

D'autre part, monter le même système de fichier à plusieurs endroits peut amener des effets de bord désagréables. Enfin, monter plusieurs systèmes de fichiers au même endroit est réalisable. C'est même utilisable (typiquement pour du NFS avec automount) mais seulement pour des utilisateurs avertis.

2.1.1 Le fichier `/etc/fstab`

Le montage de certains systèmes de fichiers cruciaux pour Linux peut être automatisé au démarrage. Cette automatisation est décrite par le fichier `/etc/fstab`. Voyons donc ce que contient ce fichier :

```
$ cat /etc/fstab
# /etc/fstab: static file system information.
# file system❶ mount pt❷ type❸  options❹                    dump pass
/dev/hda2      /        ext2    defaults,errors=remount-ro 0    1 ❺
/dev/hdb2      none     swap    sw                         0    0 ❻
proc           /proc    proc    defaults                   0    0 ❼
/dev/fd0       /floppy  auto    defaults,user,noauto       0    0
/dev/cdrom     /cdrom   iso9660 defaults,ro,user,noauto    0    0
/dev/hda1      /boot    ext2    rw                         0    2
/dev/hdd4      /zip     vfat    defaults,user,noauto       0    0
```

❶ Cette colonne indique les systèmes de fichiers que l'on va monter. Dessous sont listées les ressources de base qui contiennent des systèmes de fichiers. Les termes commençant par `/dev` sont des ressources réelles physiques (généralement le disque dur), mais on peut aussi bien monter d'autres types de ressources.

❷ Cette colonne indique où le système de fichier apparaîtra. Comme déjà dit, la ressource est montée sur un répertoire. En fait, on peut aussi monter une ressource sur un fichier mais c'est un cas rare (systèmes de gestion de base de données).

❸ C'est dans cette colonne que l'on va spécifier le type particulier du système de fichier. Le module spécifique du noyau prend en charge ce type de système de fichiers ; au delà de ce niveau, les accès sont transparents (aux limitations du système de fichier près).

❹ Ici sont passées les options de montage, ces options sont passées à la commande **mount** et peuvent être spécifiques au type de système de fichiers.

Le drapeau [**?**] indique si le système de fichier doit être sauvegardé par la commande **dump** et le drapeau [**?**] indique l'ordre de vérification des systèmes de fichiers au démarrage, le premier étant le 1. Voyons par le détail quelques lignes exemplaires de ce fichier :

❺ Montage de la première partition sur la racine. Cette partition est la première montée ; elle repose sur un système de fichiers ext2 qui est le standard du noyau Linux depuis 1993. En cas d'erreur, on demande le remontage en lecture seule (option `errors=remount-ro`).

❻ Ici, le montage est spécial : on ne monte pas effectivement un système de fichier mais une partition qui servira d'espace de pagination sur la disque dur.

❼ Encore un montage spécial : ce système de fichier n'est par réel mais sert à rendre compte du paramétrage interne du noyau. En listant le contenu de certains fichiers, on visualise les paramètres, tandis qu'en écrivant dans d'autres on change le fonctionnement du noyau. Enfin, ce répertoire liste les processus actifs sur le système en tant que répertoires, et les fichiers contenus dans chaque répertoire caractérisent le processus correspondant.

2.1.1.1 Références

[1] */usr/src/linux/Documentation/filesystems/vfs.txt*

[2] ***man** mount,**man** fstab,*

2.2 Les fichiers standards

2.3 Les fichiers Device

Les fichiers device sont généralement rassemblés dans le répertoire `/dev`. Ce n'est pas obligatoire mais c'est une convention standard. Ce sont des fichiers très spéciaux qui permettent au moyen du système de fichiers de travailler avec les périphériques réels ou virtuels de l'ordinateur. Par exemple, une carte son apparaît dans cette arborescence comme un fichier `/dev/audio`. On peut directement utiliser ce fichier pour jouer un son en déversant le contenu d'un fichier standard de type `*.wav`.

Exemple 2.2 Jouer un son directement sur le device /dev/audio

```
$ cat /scream1.wav > /dev/audio
```

NOTE
Il se peut que cet exemple ne fonctionne pas si vous utilisez un gestionnaire de fenêtres qui capture le périphérique `/dev/audio` (exemple : Gnome, KDE).

Les devices décrits dans `/dev` ne sont pas forcément réels. En effet, la plupart des distributions fournissent par défaut 6 consoles texte virtuelles. Ces consoles, bien que n'ayant pas de pendants réels sont mises en place au niveau du noyau, et sont donc présentées dans le répertoire `/dev` comme des périphériques `ttyn`.

2.3.1 Devices caractères et devices blocs

Il faut bien séparer deux types distincts de périphériques. Les périphériques à accès caractères et les périphériques à accès bloc. Cette distinction vient de la différence qu'il peut y avoir entre un disque dur et un modem. Lors d'un accès à un disque dur, ce que l'on va écrire ou lire du périphérique, ce sont des blocs de données (512 octets par exemple) tandis qu'avec un modem, on peut écrire caractère par caractère. Cela signifie aussi que les périphériques à bloc implémentent des buffers pour la lecture et l'écriture tandis que les périphériques caractère n'ont pas de mémoire sur leurs entrées/sorties. Généralement, cette distinction est aussi une distinction entre des périphériques de mémoire de masse (CDROM, disque dur, disquette : mode bloc) et des périphériques de communication (liaison série, parallèle, carte réseau : mode caractère).

Cette différence importe beaucoup pour les développeurs de gestionnaires de périphériques car les modes d'accès à implémenter sont différents selon ce type. Le montage de fichier (Cf. Section 2.1.1) ne peut avoir lieu que sur des périphériques de type bloc (hors de question de monter une carte réseau comme système de fichier !). On peut voir le type de périphérique en effectuant un **ls** `-l` sur certains fichiers dans `/dev`. La première lettre apparaissant au début de la ligne indique b(loc) ou c(aractère).

Exemple 2.3 liste de périphériques bloc et caractère

```
$ ls -l /dev/hda1    # hda1 est un peripherique bloc
brw-------    1 root     root       3,   1 jan  1  1970 /dev/hda1
$ ls -l /dev/audio   # audio est un peripherique caractere
crw-------    1 root     root       14   1 jan  1  1970 /dev/audio
```

2.3.2 Devfs

Jusqu'aux versions 2.4.*n* du noyau, les fichiers périphériques devaient être créés à la main dans le répertoire `/dev`. Comme [**?**], on ne connaît pas les périphériques de la machine à installer, il était nécessaire de prévoir dans ce répertoire tous les fichiers correspondant à tous les périphériques existants. Le répertoire `/dev` était surpeuplé de fichiers dont la plupart était inutiles. Cela ne pose pas de problème en soi : si on essaie d'accéder à un fichier qui n'a pas de périphérique correspondant, le noyau nous avertit. Mais cela nuit à la clarté du système car on ne sait pas quels sont les fichiers réellement utiles dans le répertoire. De plus, cet encombrement a aussi un effet de ralentissement du système lors de la recherche d'un fichier périphérique dans le répertoire.

Pour pallier ces défauts, un nouveau module a été codé, qui permet de peupler le répertoire `/dev` en fonction de ce qui a été détecté par le noyau et paramétré auparavant. Le dossier `/dev` devient alors un point de montage pour un système de fichier particulier : le Devfs. Malheureusement, la structure du Devfs est différente de celle retenue au départ pour les fichiers de périphériques et les applications qui ont les références "en dur" aux fichiers device ne peuvent plus fonctionner dans le nouveau système. Pour conserver la compatibilité avec les versions des applications pendant la période de migration, les anciens fichiers périphériques sont créés comme des liens sur les périphériques de Devfs. Peu à peu, l'ancienne liste plate de fichier dans `/dev` disparaîtra au profit de la nouvelle arborescence qui classe les périphériques. De ce point de vue, la représentation de `/dev` se rapproche et propose une unification avec celle de `/proc`.

Ainsi, là où on avait pêle-mêle dans `/dev` les fichiers `audio0`, `dsp0`, `mixer0` qui correspondent en fait à divers fonctionnalités offertes par une même carte son, la nouvelle architecture classe toutes ces fonctions dans une répertoire conteneur `/dev/sound` qui liste les fichiers de périphérique `audio`, `dsp` et `mixer` spécifiques à cette carte.

2.3.3 Références

[3] */usr/src/linux/Documentation/filesystems/devfs/README*

[4] *http://www.linuxdoc.org/LDP/khg/HyperNews/get/devices/basics.html*

2.4 Les fichiers Lien

La grande force des systèmes de fichier UNIX, lorsque le système de fichiers sous-jacent le permet est la possibilité de créer des liens. Le liens est un fichier qui se manipule de la même manière qu'un autre type de fichier, mais qui à la propriété d'être une référence à un fichier standard déja existant.

Quel intérêt ? Imaginons qu'un fichier de paramétrage soit utile à plusieurs applications simultanément, mais que les références à ce fichier soient inscrites en dur dans les applications et que de surcroit, les applications ne soient pas d'accord sur l'endroit

où elles sont sensées le trouver. La méthode la plus simple à priori consiste simplement à recopier le fichier à tous les endroits où une application est sensée le trouver. Simple en apparence car dès la première modification de ce fichier, on tombe dans une avalanche de changements manuels de toutes les copies, sans compter les possibilités d'oubli, d'erreur de recopie, etc.

Une solution bien plus élégante consiste à "factoriser" ce fichier : un fichier unique mais plusieurs images de ce fichier dans l'arborescence. Le *même* fichier apparaît à tous les endroits utiles ! Une modification d'une des images du fichier impacte directement toutes les autres car en fait, on n'a qu'un fichier unique.

Pour réaliser ce tour de passe-passe, il existe deux types de création de lien qui ont leurs avantages et leurs faiblesses : les liens durs (*hardlinks* dans la littérature anglaise) et les liens symboliques (*symlinks* ou *softlinks*). Les plus utilisés actuellement sont les liens symboliques car il permettent un plus grande souplesse. Pour créer un lien sur un fichier, on utilise la commande unique **ln** dont le comportement varie selon les paramètres qu'on lui passe. Il est possible de créer des liens sur tout type de fichiers : dossiers, fichiers standards, devices, etc.

2.4.1 Les liens durs

Les liens durs correspondent à la manière brute de réaliser un lien. La correspondance entre le nom du fichier et l'emplacement du fichier sur un medium de stockage est généralement enregistrée dans les données de structure du système de fichiers. En étendant ce mécanisme, il est possible de faire correspondre directement plusieurs noms de fichiers au même emplacement. Chaque nom du fichier est *réellement* le fichier. Il n'y a pas de priorité d'un nom par rapport à un autre.

Pour ne pas se perdre lors de la création et la destruction de noms associés à un fichier, la structure du système de fichier garde en mémoire le nombre de noms qui pointent sur la données. Lorsque l'on crée un nouveau lien dur sur un fichier, on incrémente ce compteur. Lorsque l'on détruit un des noms du fichier, on décrémente le compteur. Si ce compteur atteint 0, cela signifie qu'il n'y a plus de nom pointant sur le fichier et que l'on peut effectivement l'effacer. Le décompte du nombre de noms d'un fichier apparaît lorsque l'on fait la commande **ls** `-l` sur le fichier.

Exemple 2.4 Gestion de liens durs d'un fichier

```
$ touch monfichier.txt❶
$ ls -l monfichier.txt
-rw-rw-r--    ❷1 jnavila  jnavila  0 fÃ©v 20 21:18 monfichier.txt
$ ln monfichier.txt monfichier2.txt❸
$ ls -l monfichier*
-rw-rw-r--    ❹2 jnavila  jnavila  0 fÃ©v 20 21:18 monfichier2.txt
-rw-rw-r--    ❺2 jnavila  jnavila  0 fÃ©v 20 21:18 monfichier.txt
$ rm monfichier2.txt
$ ls -l monfichier*
-rw-rw-r--    ❻1 jnavila  jnavila  0 fÃ©v 20 21:18 monfichier2.txt
```

❶ Nous créons un fichier `monfichier.txt` vide

❷ Lorsque nous listons les caractéristiques du fichier, la deuxième colonne contient un chiffre qui indique le nombre de noms qui pointent vers le fichier, en l'occurence un seul pour notre fichier fraîchement créé.

❸ Cette commande permet de créer un deuxième nom de fichier `monfichier2.txt` à partir du fichier pointé par le nom `monfichier.txt`.

❹, ❺ Nous pouvons bien vérifier que `monfichier.txt` et `monfichier2.txt` apparaissent tous deux comme deux fichiers standards et pointent bien tout deux sur un fichier référencé par deux noms.

❻ Lorsque nous éliminons un des noms qui pointent sur le fichier, nous pouvons vérifier que le second continue de pointer dessus, mais que le compteur de liens a été décrémenté.

Le dossier `/usr/lib` est très souvent repiqué grâce aux liens durs sur d'autres repertoires.

Les limitations du liens durs apparaissent lorsque l'on veut créer des liens à travers plusieurs systèmes de fichiers. Pour pouvoir référencer le même fichier, les noms de fichiers des liens durs doivent nécessairement appartenir au même système de fichiers. On ne peut pas créer dans un répertoire correspondant à un système de fichier un lien dur pointant sur un fichier contenu dans un autre système de fichier. Pour ces cas, la seule option reste l'utilisation de lien symbolique.

2.4.2 Les liens symboliques

Les liens symboliques reposent sur une organisation hierarchisée des liens : un lien créé sur un fichier ne modifie pas ce dernier. Le liens créé n'est qu'un pointeur vers le nom du fichier réel. L'accès au fichier lien crée alors un résolution du nom du fichier pointé pour finalement accéder au fichier. Il est alors possible que le nom de fichier initialement pointé n'existe plus au moment de l'emploi du lien, ce qui provoque immanquablement une erreur de type "Aucun fichier ou répertoire de ce type".

Ce concept doit bien être différencié de celui des raccourcis Windows™. Ceux-ci ne sont que des fichiers particuliers que seul l'explorateur est capable de résoudre. Dans le cas de liens symboliques, tant que le fichier pointé existe, l'accès au fichier lien est *totalement* transparent pour toute application qui y accède. C'est à dire qu'en modifiant le fichier lien, on modifie bien le fichier pointé.

Exemple 2.5 Gestion de liens symboliques d'un fichier

```
$ echo toto > monfichier.txt  ❶
$ echo titi > monfichier2.txt ❷
$ ln -s monfichier.txt monfichier3.txt ❸
$ ls -l monfichier*
-rw-rw-r--    1 jnavila  jnavila    5 fÃ©v 20 22:45 monfichier2.txt ❹
lrwxrwxrwx    1 jnavila  jnavila   14 fÃ©v 20 22:46 monfichier3.txt -> monfichier.txt
-rw-rw-r--    1 jnavila  jnavila    5 fÃ©v 20 22:45 monfichier.txt
$ cat monfichier3.txt
toto ❺
$ rm monfichier.txt
$ ls -l monfichier*
-rw-rw-r--    1 jnavila  jnavila    5 fÃ©v 20 22:45 monfichier2.txt ❻
lrwxrwxrwx    1 jnavila  jnavila   14 fÃ©v 20 22:46 monfichier3.txt -> monfichier.txt
$ cat monfichier3.txt
cat: monfichier3.txt: Aucun fichier ou rÃ©pertoire de ce type ❼
$ mv monfichier2.txt monfichier.txt
$ cat monfichier3.txt
titi ❽
```

❶ Nous créons deux fichiers standards `monfichier.txt` et `monfichier2.txt` au contenu différent.

❸ Puis nous créons un liens symbolique sur le fichier `monfichier.txt` qui se nomme `monfichier3.txt` au moyen de la commande **ln** avec le paramètre particulier `-s` pour spécifier que l'on veut que le lien est symbolique.

❹ Lorsque nous listons les fichiers créés, `monfichier3.txt` apparaît comme un pointeur (le premier caractère de la ligne est un [**?**]) sur le nom `monfichier.txt`,

❺ et nous pouvons visualiser le contenu de ce fichier aussi normalement que si c'était un fichier standard ; comme attendu, nous trouvons le contenu de `monfichier.txt`.

❻ Pour corser la démonstration, nous éliminons sans ménagement le fichier `monfichier.txt`. Le système n'émet aucun avertissement. Lorsque nous listons les fichiers restants, ceux-ci n'ont pas été modifiés. Mais `monfichier3.txt` pointe toujours sur un fichier fantome qui n'existe plus...

❼ Et lorsque nous cherchons à visualiser le contenu de ce fichier, le système nous renvoie une fin de non-recevoir. Le fichier `monfichier3.txt` ne correspond plus à une entité existante et peut-être considéré comme inexistant.

❽ Pour corriger ce problème, nous renommons `monfichier2.txt` en `monfichier.txt`. Le pointeur redevient valide... Par contre, son contenu est maintenant celui de `monfichier2.txt` !

2.4.3 Récapitulatif

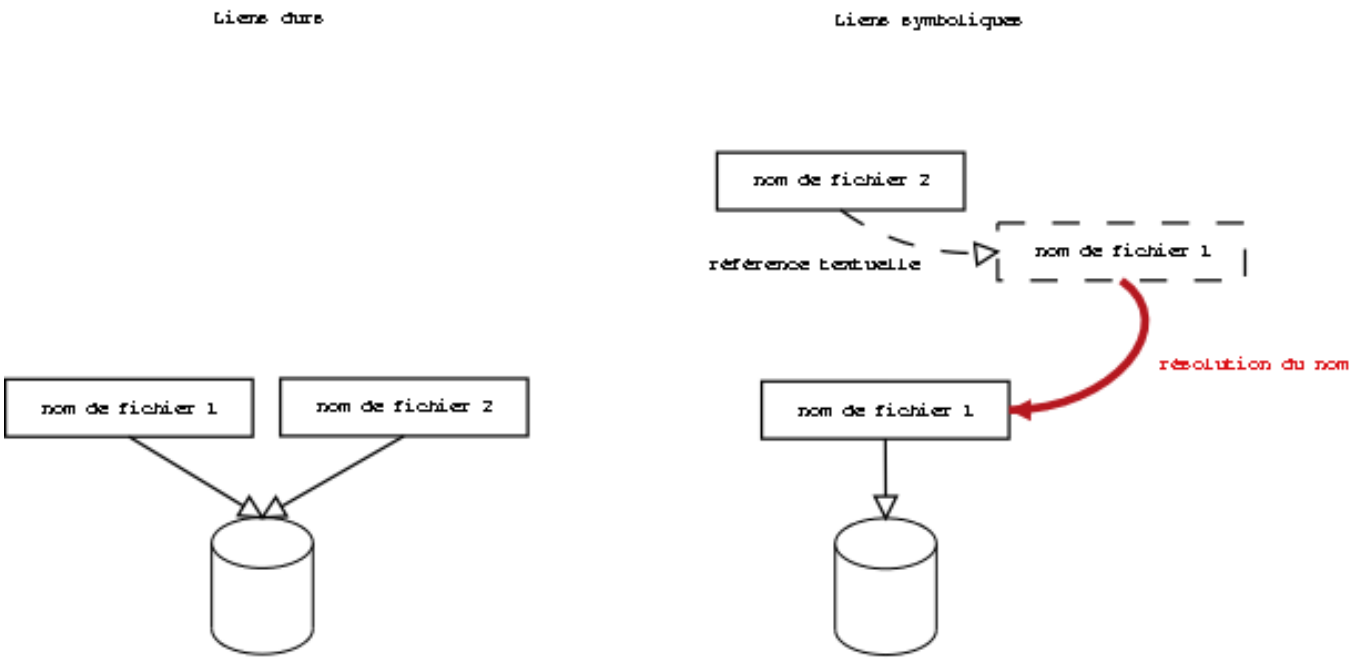

FIG. 1 – Différences entre liens durs et symboliques

TBD : schéma de différence des liens, tableau comparatifs de avantages/limitations

2.5 Les fichiers tube nommé

TBD : para d'intro

Exemple 2.6 Utilisation d'un tube nommé

Pour cet exemple, nous avons besoin de deux terminaux pour visualiser le travail du tube nommé. Dans le premier terminal, tapons :

```
$ mkfifo mafifo ❶[1]
$ ls -l mafifo
prw-rw-r--    1 jnavila  jnavila         0 fÃ©v 21 20:57 mafifo| ❷
$ echo ceci est un texte que je mets en fifo > mafifo ❸
```

❶ Nous créons un fichier tube nommé nommé `mafifo` au moyen de la commande **mkfifo**.

❷ Lorsque nous visualisons les caractéristiques du fichier, la premier caractère sur la ligne [**?**] et le signe [**?**] en fin du nom de fichier indique que nous avons affaire à un tube nommé.

❸ Nous cherchons à remplir le fichier fifo avec du texte. A ce moment, la ligne de commande ne nous rend pas la main. La commande est bloquée, elle ne peut s'executer. Ceci s'explique par le fait qu'il faut qu'un autre processus lise le tube pour que le premier puisse écrire. Dans le second terminal :

```
$ cat mafifo ❶
ceci est un texte que je mets en fifo ❷
$ cat mafifo ❸
```

❶ Nous cherchons à lire le contenu du tube,

❷ et effectivement nous y trouvons ce que la commande du premier terminal cherchait à y écrire. On remarque que dans le même temps, la ligne de commande a été rendue dans le premier terminal.

❸ Par contre, un deuxième essai de lecture bloque le retour à la ligne. Le processus en lecture est bloqué. Il faudra relancer la même commande dans le premier terminal pour obtienir une sortie dans ce terminal et le déblocage du processus lecteur.

2.6 Les fichiers Socket

TBD

3 Des fichiers et des droits

TBD

4 Des fichiers et une hiérarchie

4.1 Histoire

TBD

4.2 Objectifs du FHS

TBD

4.3 Typage des fichiers selon le FHS

TBD

4.4 Détail de l'arborescence

TBD