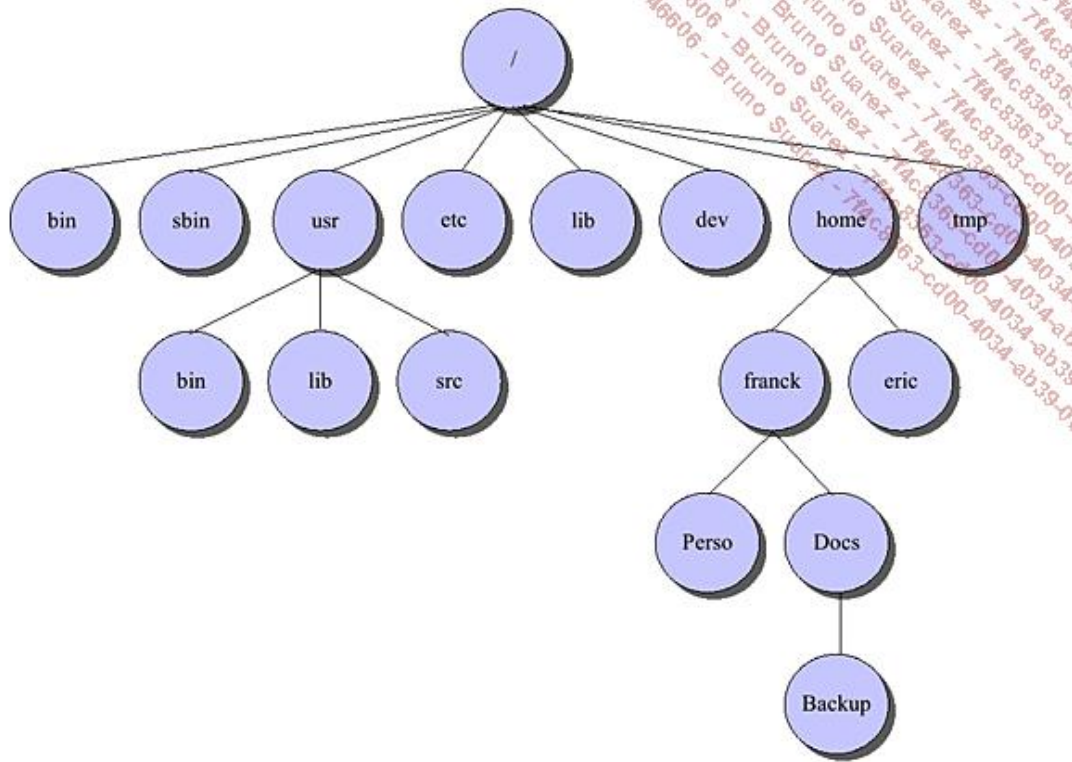


La gestion des fichiers

1. Le système de fichiers

Un système de fichiers, appelé communément File System ou FS, définit l'organisation des données sur un support de stockage, donc comment sont gérés et organisés les fichiers par le système d'exploitation.

Linux est, comme tout Unix, un système d'exploitation entièrement orienté fichier. Tout (ou presque) est représenté par un fichier, tant les données (fichiers de données de tout type comme une image ou un programme), que les périphériques (terminaux, souris, clavier, carte son, etc.) ou encore les moyens de communication (sockets, tubes nommés, etc.). On peut dire que le système de fichier est le coeur de tout système Unix.



Exemple d'arborescence Linux

Le système de fichiers de Linux est hiérarchique. Il décrit une arborescence de répertoires et de sous-répertoires, en partant d'un élément de base appelé la racine ou root directory.

2. Les divers types de fichiers

On distingue trois types de fichiers : ordinaires, catalogue, spéciaux.

a. Les fichiers ordinaires ou réguliers

Les fichiers ordinaires sont aussi appelés fichiers réguliers, ordinary files ou regular files. Ce sont des fichiers tout à fait classiques qui contiennent des données. Par données, comprenez n'importe quel contenu :

- texte
- image
- audio

- programme binaire compilé
- script
- base de données
- bibliothèque de programmation
- etc.

Par défaut, rien ne permet de différencier les uns des autres, sauf à utiliser quelques options de certaines commandes (ls -F par exemple) ou la commande **file**.

```
$ file nom_fic
nom_fic : 32 Bits ELF Executable Binary (stripped)
```



La notion d'extension de fichier comme composante interne de la structure du système de fichier est inconnue de Linux. Autrement dit une extension n'a aucun rôle au niveau du système de fichier et est simplement considérée comme une partie du nom du fichier. Elle sert simplement à distinguer visuellement et rapidement l'éventuel contenu d'un fichier par rapport à un autre.

Comme les extensions ne sont pas gérées par Linux, le nom d'un programme ne finit quasiment jamais par un « .exe », il faudra trouver autre chose pour le distinguer.

b. Les catalogues

Les fichiers catalogues sont les répertoires, dossiers ou directory. Les répertoires permettent d'organiser le disque dur en créant une hiérarchie. Un répertoire peut contenir des fichiers normaux, des fichiers spéciaux et d'autres répertoires, de manière récursive.

Un répertoire n'est rien d'autre qu'un fichier particulier contenant la liste des fichiers eux-mêmes présents dans ce répertoire. Cette notion se révèlera très utile lorsque la question des droits sera abordée.

c. Les fichiers spéciaux

Le troisième type de fichier est le fichier spécial. Il existe plusieurs genres de fichiers spéciaux. Ils se trouvent principalement dans le répertoire /dev s'ils représentent des périphériques.

Ce sont principalement des fichiers servant d'interface pour les divers périphériques. Ils peuvent s'utiliser, suivant le cas, comme des fichiers normaux. Un accès en lecture ou écriture sur ces fichiers est directement redirigé vers le périphérique (en passant par le pilote associé s'il existe). Par exemple si vous redirigez un fichier d'onde sonore (wave) vers le fichier représentant la sortie de la carte son, il y a de fortes chances que ce son soit audible par vos haut-parleurs.

3. Nomenclature des fichiers

On ne peut pas donner n'importe quel nom à un fichier, il faut pour cela suivre quelques règles simples. Ces règles sont valables pour tous les types de fichiers.

Sur les anciens systèmes Unix un nom de fichier ne pouvait pas dépasser 14 caractères. Sur les systèmes actuels, dont Linux, on peut aller jusqu'à 255 caractères. L'éventuelle extension est comprise dans la longueur du nom du fichier.

Un point extrêmement important : Linux fait la distinction entre les noms de fichiers en minuscules et en majuscules. Toto, TOTO, ToTo et toto sont des noms de fichiers différents, avec un contenu différent.

La plupart des caractères (les chiffres, les lettres, les majuscules, les minuscules, certains signes, les caractères accentués) sont acceptés, y compris l'espace. Cependant quelques caractères sont à éviter car ils ont une signification particulière au sein du shell : & ; () ~ <espace> \ / | ` ? - (en début de nom).

Les noms suivants sont valides :

- Fichier1

- Paie.txt
- 123traitement.sh
- Paie_juin_2002.xls
- 8

Ces noms peuvent poser des problèmes :

- Fichier*
- Paie(decembre)
- Ben&Nuts
- Paie juin 2002.xls
- -f

4. Les chemins

a. Structure et nom de chemin

Les chemins permettent de définir un emplacement au sein du système de fichiers. C'est la liste des répertoires et sous-répertoires empruntés pour accéder à un endroit donné de l'arborescence jusqu'à la position souhaitée (répertoire, fichier). Un nom de fichier est ainsi généralement complété par son chemin d'accès. C'est ce qui fait que le fichier toto du répertoire rep1 est différent du fichier toto du répertoire rep2. Le FS d'Unix étant hiérarchique, il décrit une arborescence.

Le schéma présenté dans la section La gestion des fichiers - Le système de fichiers de ce chapitre représente une arborescence d'un système de fichier Linux. Le / situé tout en haut s'appelle la racine ou root directory (à ne pas confondre avec le répertoire de l'administrateur root). Le nom de chemin ou path name d'un fichier est la concaténation, depuis la racine, de tous les répertoires qu'il est nécessaire de traverser pour y accéder, chacun étant séparé par le caractère /. C'est un chemin absolu comme celui-ci.

```
/home/toto/Docs/Backup/fic.bak
```

Un chemin absolu ou complet :

- démarre de la racine, donc commence par un /
- décrit tous les répertoires à traverser pour accéder à l'endroit voulu
- ne contient pas de . ni de ..

b. Répertoire personnel

Lors de la création d'un utilisateur, l'administrateur lui alloue un répertoire personnel appelé home directory. Lorsqu'il se connecte, l'utilisateur arrive directement dans ce répertoire, qui est son répertoire personnel. C'est dans ce répertoire que l'utilisateur pourra créer ses propres fichiers et répertoires.

```
Login : seb
Password : xxxxxxxxxxxx
$ pwd
/home/seb
```

c. Chemin relatif

Un nom de chemin peut aussi être relatif à sa position courante dans le répertoire. Le système (ou le shell) mémorise la position actuelle d'un utilisateur dans le système de fichier, le répertoire actif. Vous pouvez accéder à un autre répertoire de l'arborescence depuis l'emplacement actuel sans taper le chemin complet uniquement en précisant le chemin le plus court relativement à votre position actuelle au sein de l'arborescence.

Il faut pour cela souvent utiliser deux entrées particulières de répertoires :

- Le point `.` représente le répertoire courant, actif. Il est généralement implicite.
- Les doubles points `..` représentent le répertoire de niveau inférieur.

Un chemin relatif :

- décrit un chemin relatif à une position donnée dans l'arborescence, généralement (mais pas toujours) depuis la position courante.
- décrit en principe le plus court chemin pour aller d'un point à un autre.
- peut contenir des points ou des doubles points.

Ces trois affirmations ne sont pas des obligations :

- `/usr/local/bin` est un chemin complet ou absolu.
- `Documents/Photos` est un chemin relatif : le répertoire `Documents` est considéré comme existant dans le répertoire courant.
- `./Documents/Photos` est un chemin relatif parfaitement identique au précédent, sauf que le répertoire actif (courant) est explicitement indiqué par le point. « `./Documents` » indique explicitement le répertoire `Documents` dans le répertoire actif.
- `/usr/local/../bin` est un chemin relatif : les `..` sont relatifs à `/usr/local` et descendent d'un niveau vers `/usr`. Le chemin final est donc `/usr/bin`.

d. Le tilde

Le bash interprète le caractère tilde `~` comme un alias du répertoire personnel. Les chemins peuvent être relatifs au tilde, mais le tilde ne doit être précédé d'aucun caractère. Pour vous déplacer dans le répertoire `tmp` de votre dossier personnel d'où que vous soyez :

```
$ cd ~/tmp
```

Si vous entrez ceci, vous obtenez une erreur :

```
$ cd /~
```

e. cd

Pour vous déplacer dans les répertoires, vous utilisez la commande **cd** (*change directory*). La commande **pwd** (*print working directory*) que vous avez déjà rencontrée affiche le chemin complet du répertoire courant.

Si vous saisissez `cd .`, vous ne bougez pas. Le point sera très utile lorsque vous devrez spécifier des chemins explicites à des commandes situées dans le répertoire où vous êtes positionné.

Le `cd ..` remonte d'un niveau. Si vous étiez dans `/home/seb`, vous vous retrouvez dans `home`.

La commande `cd` sans argument permet de retourner directement dans son répertoire utilisateur.

Voici un petit exemple. L'utilisateur `seb` démarre de son répertoire personnel. Il se déplace via un chemin relatif vers `/home/public`. Le `..` remonte vers `/home`, donc `../public` se déplace dans `/home/public`. De là, via un chemin complet, il se dirige vers `/usr/local/bin`, puis décide à l'aide d'un chemin relatif de se rendre dans `/usr/lib` : le premier `..` descend vers `/usr/local`, le second vers `/usr`, puis remonte vers `/usr/lib`. Enfin `seb` retourne dans son répertoire personnel avec `cd` sans argument. L'invite est ici donnée complète pour une meilleure compréhension.

```

seb@slyserver:~> pwd
/home/seb
seb@slyserver:~> cd ../public
seb@slyserver:/home/public> cd /usr/local/bin
seb@slyserver:/usr/local/bin> cd ../../lib
seb@slyserver:/usr/lib> cd
seb@slyserver:~>

```

5. Les commandes de base

a. Lister les fichiers et les répertoires

La commande **ls** permet de lister le contenu d'un répertoire (catalogue) en lignes ou colonnes. Elle supporte plusieurs paramètres dont voici les plus pertinents.

Paramètre	Signification
-l	Pour chaque fichier ou dossier, fournit des informations détaillées.
-a	Les fichiers cachés sont affichés (ils commencent par un point).
-d	Sur un répertoire, précise le répertoire lui-même et non son contenu.
-F	Rajoute un caractère à la fin du nom pour spécifier le type : / pour un répertoire, * pour un exécutable, @ pour un lien symbolique, etc.
-R	Si la commande rencontre des répertoires, elle rentre dans les sous-répertoires, sous-sous-répertoires, etc., de manière récursive.
-t	La sortie est triée par date de modification du plus récent au plus ancien. Cette date est affichée.
-c	Affiche / tri (avec -t) par date de changement d'état du fichier.
-u	Affiche / tri (avec -t) par date d'accès du fichier.
-r	L'ordre de sortie est inversé.
-i	Affiche l'inode du fichier.
-C	L'affichage est sur plusieurs colonnes (par défaut).
-1	L'affichage est sur une seule colonne.

Le paramètre qui vous fournit le plus d'informations est le **-l** : il donne un certain nombre de détails sur les fichiers.

```

$ ls -l
total 4568
-rw-r--r-- 1 seb users 69120 sep 3 2006 3i_rattrapage_2006.doc
-rw-r--r-- 1 seb users 9632 sep 3 2006 3i_rattrapage_2006.odt
-rw-r--r-- 1 seb users 6849 nov 17 2003 controle_1I2_mardi.sxw
...

```

La ligne total indique la taille totale en kilo-octets du contenu du répertoire. Cette taille est celle de l'ensemble des fichiers ordinaires du répertoire et ne prend pas en compte les éventuels sous-répertoires et leur contenu (pour ceci, il faudra utiliser la commande **du**).

Vient ensuite la liste détaillée de tout le contenu.

-rw-r--r--	1	seb	users	69120	sep 3 2006	3i_rattrapage_2006.doc
------------	---	-----	-------	-------	------------	------------------------

1	2	3	4	5	6	7
---	---	---	---	---	---	---

- 1 : Le premier caractère représente le type de fichier (- : ordinaire, d : répertoire, l : lien symbolique...) ; les autres, par blocs de trois, les droits pour l'utilisateur (rw-), le groupe (r--) et tous (r--). Les droits sont expliqués au chapitre Les disques et le système de fichiers.
- 2 : Un compteur de liens (chapitre Les disques et le système de fichiers).
- 3 : Le propriétaire du fichier, généralement celui qui l'a créé.
- 4 : Le groupe auquel appartient le fichier.
- 5 : La taille du fichier en octets.
- 6 : La date de dernière modification (parfois avec l'heure), suivant le paramètre (t, c, u).
- 7 : Le nom du fichier.

Vous pouvez trouver très utile de pouvoir lister vos fichiers de manière à ce que ceux modifiés le plus récemment soient affichés en fin de liste. Ainsi en cas de présence d'un très grand nombre de fichiers, cela vous évite de remonter tout en haut de la console. Le tri par date de modification se fait avec `-t` et dans l'ordre inverse avec `-r`. Rajoutez-y les détails avec `-l`.

```
$ ls -lrt
-rw-r--r-- 1 seb users 66107 jan 9 17:24 Partiel_1_1I_2008.pdf
-rw-r--r-- 1 seb users 13777 jan 10 17:58 partiel_3I_ppa_2007.odt
-rw-r--r-- 1 seb users 64095 jan 10 17:58 partiel_3I_ppa_2007.pdf
-rw-r--r-- 1 seb users 100092 fév 22 22:21 cours_shell_unix.odt
```



`ls -l -r -t` est strictement identique à `ls -lrt` comme indiqué dans la syntaxe générale des commandes.

Un moyen mnémotechnique de se rappeler cette séquence d'arguments est de l'utiliser sous sa forme `-rtl` (l'ordre des arguments n'a pas d'importance ici) et de penser ainsi à la célèbre radio.

b. Gérer les fichiers et les répertoires

Créer des fichiers vides

Pour vos tests ou durant vos actions vous pouvez avoir besoin de créer des fichiers vides. Une commande pratique pour cela est **touch**. Utilisée avec uniquement le nom d'un fichier en argument, elle crée un fichier avec une taille nulle.

```
$ touch fictest
$ ls -l fictest
-rw-r--r-- 1 seb users 0 fév 29 15:13 fictest
```

La création de fichiers vides n'est pas à l'origine le principal usage de `touch`. Si vous relancez la même commande sur le fichier, vous remarquez que la date de modification a changé. Le manuel de `touch` vous informera qu'il est ainsi possible de modifier complètement l'horodatage d'un fichier. Ceci peut être utile pour forcer les sauvegardes incrémentales sur des fichiers.

Créer des répertoires

La commande **mkdir** (*make directory*) permet de créer un ou plusieurs répertoires, ou une arborescence complète. Par défaut la commande ne crée pas d'arborescence. Si vous passez comme arguments `rep1/rep2` et que `rep1` n'existe pas, la commande retourne une erreur. Dans ce cas, utilisez le paramètre `-p`.

```
mkdir [-p] rep1 [rep2] ... [repn]
```

```
$ mkdir Documents
```

```
$ mkdir Documents/Photos
$ mkdir -p Archives/vieilleries
$ ls -R
.:
Archives Documents fictest

./Archives:
vieilleries

./Archives/vieilleries:

./Documents:
Photos
```

Supprimer des répertoires

La commande **rmdir** (*remove directory*) supprime un ou plusieurs répertoires. Elle ne peut pas supprimer une arborescence. Si des fichiers sont encore présents dans le répertoire, la commande retourne une erreur. Le répertoire ne doit donc contenir ni fichiers ni répertoires et ceci même si les sous-répertoires sont eux-mêmes vides.

```
rmdir rep1 [rep2] ... [repn]
```



Il n'y a pas de paramètre **-r** (pour récursif) à la commande **rmdir**. Pour supprimer une arborescence vous devrez utiliser la commande **rm**.

```
$ rmdir Documents/
rmdir: Documents/: Le répertoire n'est pas vide.
$ rmdir Documents/Photos
$
```

Copier des fichiers

La commande **cp** (copy) copie un ou plusieurs fichiers vers un autre fichier ou vers un répertoire.

```
cp fic1 [fic2 ... ficn] Destination
```

Dans le premier cas, fic1 est recopié en Destination. Si Destination existe, il est écrasé sans avertissement selon le paramètre passé et selon les droits. Dans le second cas, fic1, fic2 et ainsi de suite sont recopiés dans le répertoire Destination. Les chemins peuvent être absolus ou relatifs. La commande peut prendre, entre autres, les options suivantes :

Paramètre	Signification
-i	Demande de confirmation de copie pour chaque fichier.
-r	Récursif : copie un répertoire et tout son contenu.
-p	Les permissions et dates sont préservées.
-f	Forcer la copie.

Votre attention doit être attirée sur le fonctionnement de cp avec les copies de répertoires. Le fonctionnement est différent selon la présence du répertoire de destination ou non. Dans le premier cas, rep2 n'existe pas. Le répertoire rep1 est copié en rep2. À la fin rep2 est une copie exacte de rep1.

```
$ ls -d rep2
ls: ne peut accéder rep2: Aucun fichier ou répertoire de ce type
$ cp -r rep1 rep2
$ ls
rep1 rep2
```

Maintenant que rep2 existe, exécutez de nouveau la commande cp. Cette fois, comme rep2 existe, il n'est pas écrasé comme vous pourriez le penser. La commande détermine que la destination étant le répertoire rep2, rep1 doit être copiée dans la destination : rep1 est copié dans rep2.

```
$ cp -r rep1 rep2
$ ls rep2
rep1
```

Déplacer et renommer un fichier

La commande **mv** (*move*) permet de déplacer, de renommer un fichier, ou les deux en même temps. Elle fonctionne comme la commande cp. Les paramètres **-f** et **-i** ont le même effet. Avec les trois commandes mv successives suivantes :

- txt1 est renommé en txt1.old.
- txt2 est déplacé dans rep1.
- txt3 est déplacé dans rep1 et renommé en txt3.old.

```
$ touch txt1 txt2 txt3
$ mv txt1 txt1.old
$ mv txt2 rep1/txt2
$ mv txt3 rep1/txt3.old
```

Notez l'existence du paramètre **-u** : si le fichier de destination existe avec une date plus récente, cela vous évite de l'écraser.

Supprimer un fichier ou une arborescence

La commande **rm** (*remove*) supprime un ou plusieurs fichiers, et éventuellement une arborescence complète, suivant les options. La suppression est définitive.


```
rm [Options] fic1 [fic2...]
```

Les options sont classiques mais vu la particularité et la dangerosité de la commande il est bon de faire un rappel.

Paramètre	Signification
-i	La commande demandera une confirmation pour chacun des fichiers à supprimer. Suivant la version d'Unix, le message change et la réponse aussi : y, Y, O, o, N, n, parfois toutes.
-r	Le paramètre suivant attendu est un répertoire. Dans ce cas, la suppression est réursive : tous les niveaux inférieurs sont supprimés, les répertoires comme les fichiers.
-f	Force la suppression.

Dans l'ordre, les commandes suivantes suppriment un simple fichier, suppriment un répertoire, et une arborescence de manière forcée :

```
$ rm fic1
$ rm -r rep1
$ rm -rf /home/public/depots
```

 L'utilisation combinée des paramètres **-r** et **-f** bien que très utile et pratique est très dangereuse, notamment en tant que root. Aucune confirmation ne vous est demandée. À moins d'utiliser des outils de récupération de données spécifiques, chers et peu performants, vos données sont irrémédiablement perdues.

Voici une astuce. Vous pouvez créer des fichiers qui commencent par un tiret. Mais avez-vous essayé de les supprimer avec rm ?

```
$ >-i # voir les redirections
$ rm -i
rm: opérande manquante
Pour en savoir davantage, faites: « rm --help ».
```


Il est impossible de supprimer le fichier « -i » de cette manière car rm l'interprète comme un paramètre et non comme un argument. Aussi faut-il ruser. Il y a deux solutions :

- Utiliser l'option GNU `--` signifiant la fin des paramètres et le début des arguments.
- Rajouter un chemin, relatif ou complet, avant le tiret.

Cette dernière solution a l'avantage d'être standard. Les deux lignes sont équivalentes :

```
$ rm -- -i
$ rm ./-i
```

Les liens symboliques

Vous pouvez créer des liens, qui sont un peu comme des raccourcis. Un lien est un fichier spécial contenant comme information un chemin vers un autre fichier. C'est une sortie d'alias. Il existe deux types de liens : le lien dur (hard link) que vous verrez plus loin, lors de l'étude des systèmes de fichiers, et le lien symbolique (soft link) qui correspond à la définition donnée.

Il est possible de créer des liens symboliques vers n'importe quel type de fichier, quel qu'il soit et où qu'il soit. La commande de création des liens symboliques ne vérifie pas si le fichier pointé existe. Il est même possible de créer des liens sur des fichiers qui n'existent pas.

```
ln -s fichier lien
```

Le cas échéant le lien se comportera à l'identique du fichier pointé avec les mêmes permissions et les mêmes propriétés :

- si le fichier pointé est un programme, lancer le lien lance le programme.
- si le fichier pointé est un répertoire, un cd sur le lien rentre dans ce répertoire.
- si le fichier pointé est un fichier spécial (périphérique), le lien est vu comme périphérique.
- etc.

Le seul cas où le lien symbolique se détache du fichier pointé est la suppression. La suppression d'un lien symbolique n'entraîne que la suppression de ce lien, pas du fichier pointé. La suppression du fichier pointé n'entraîne pas la suppression des liens symboliques associés. Dans ce cas le lien pointe dans le vide.

```
$ touch fic1
$ ln -s fic1 lienfic1
$ ls -l
-rw-r--r-- 1 seb users 0 mar  4 19:16 fic1
lrwxrwxrwx 1 seb users 4 mar  4 19:17 lienfic1 -> fic1
$ ls -F
fic1 lienfic1@
$ echo titi>fic1
$ cat lienfic1
titi
```

Cet exemple montre bien qu'un lien symbolique est en fait un fichier spécial de type « l » pointant vers un autre fichier. Notez dans la liste détaillée la présence d'une flèche indiquant sur quel fichier pointe le lien. On distingue le caractère @ indiquant qu'il s'agit d'un lien symbolique lors de l'utilisation du paramètre -F. Si vous disposez d'un terminal en couleur, il est possible que le lien symbolique apparaisse, par convention sous Linux, en bleu ciel. S'il apparaît en rouge, c'est qu'il pointe dans le vide.



Ce n'est pas parce que un lien pointe dans le vide qu'il est forcément mauvais. C'est peut-être fait exprès car il est possible de créer des liens vers des clés USB, des CD-Roms, entre divers systèmes de fichiers, qui peuvent être amovibles. Dans ce cas, le lien redevient actif quand le support est inséré et/ou que la cible est de nouveau présente.

La commande **echo** et le signe **>** seront expliqués plus loin. L'effet est ici l'écriture dans le fichier fic1 de « titi ». La commande **cat** affiche le contenu d'un fichier. Le lien représentant fic1, la sortie est bien celle attendue.



Attention, les droits indiqués sont ceux du fichier spécial et n'ont pas de signification autre : ils ne veulent pas dire que tout le monde a tous les droits sur le fichier pointé. Lors de son utilisation, ce sont les droits du fichier ou du dossier pointés qui prennent le dessus.

c. Wildcards : caractères de substitution

Lors de l'utilisation de commandes en rapport avec le système de fichier, il peut devenir intéressant de filtrer la sortie de noms de fichiers à l'aide de certains critères, par exemple avec la commande **ls**. Au lieu d'afficher toute la liste des fichiers, on peut filtrer l'affichage à l'aide de divers critères et caractères spéciaux.

Caractère(s)	Rôle
*	Remplace une chaîne de longueur variable, même vide.
?	Remplace un caractère unique quelconque.
[...]	Une série ou une plage de caractères.
[a-b]	Un caractère parmi la plage indiquée (de a à b inclus).
[!...]	Inversion de la recherche.
[^...]	Idem.

- Soit le contenu suivant :

```
$ ls
afic  afic2  bfic  bfic2  cfic  cfic2  dfic  dfic2
afic1 afic3  bfic1 bfic3  cfic1 cfic3  dfic1 dfic3
```

- Vous obtenez tous les fichiers commençant par a :

```
$ ls a*
afic1 afic2 afic3
```

- Tous les fichiers de quatre caractères commençant par a :

```
$ ls a???
afic
```

- Tous les fichiers d'au moins trois caractères et commençant par b :

```
$ ls b??*
bfic bfic1 bfic2 bfic3
```

- Tous les fichiers finissant par 1 ou 2 :

```
$ ls *[12]
afic1 afic2 bfic1 bfic2 cfic1 cfic2 dfic1 dfic2
```

- Tous les fichiers commençant par les lettres de a à c, possédant au moins un second caractère avant la terminaison 1 ou 2 :

```
$ ls [a-c]?*[12]
afic1 afic2 bfic1 bfic2 cfic1 cfic2
```

- Tous les fichiers ne finissant pas par 3 :

```
$ ls *[^3]
afic afic1 afic2 bfic bfic1 bfic2 cfic cfic1 cfic2 dfic
dfic1 dfic2
```

Interprétation par le shell

C'est le shell qui est chargé d'effectuer la substitution de ces caractères avant le passage des paramètres à une commande. Ainsi lors d'un `$ cp * Documents`, `cp` ne reçoit pas le caractère `*` mais la liste de tous les fichiers et répertoires du répertoire actif.

Les wildcards sont utilisables au sein de tous les arguments représentant des fichiers ou des chemins. Ainsi la commande suivante va recopier tous les fichiers README de tous les sous-répertoires de Documents à la position actuelle :

```
$ cp Documents/*/README .
```

d. Verrouillage de caractères

Certains caractères spéciaux doivent être verrouillés, par exemple en cas de caractères peu courants dans un nom de fichier.

- L'**antislash** `\` permet de verrouiller un caractère unique. `ls paie\ *.xls` va lister tous les fichiers contenant un espace après paie.
- Les **guillemets** `"..."` permettent l'interprétation des caractères spéciaux, des variables, au sein d'une chaîne.
- Les **apostrophes** `'...'` verrouillent tous les caractères spéciaux dans une chaîne ou un fichier.