Comprendre les montages sous Linux

Les données sur nos disques se portent bien, merci pour elles... Avec peu d'efforts elles prospèrent, s'épanouissent sur des espaces de plus en plus larges, des arborescences de plus en plus profondes. Je dois aujourd'hui avoir près d'un tera de disques du dont à peine 1/4 de libre...
Et pourtant, n'en déplaise aux mauvais esprit je n'ai pas ou peu de films dans le tas mais des centaines de milliers de fichiers, de docs, amoureusement accumulées depuis des années, certaines datant de l'époque où un disque dur de 10mo était un évennement que l'on accueillait l'oeil brillant dans son PC...
Nous sommes loin de cela aujourd'hui et un espace de stockage commence à sérieusement se compter en terra... Tout un petit monde qui se doit d'être géré aux plus simple. Et s'il est justement un domaine, avec le réseau, où Linux brille, c'est bien dans sa gestion de fichiers...

Un Slash pour les gouverner tous

L'art de la bouture sous Linux commence avec un /. Notion encore plus importante à comprendre pour les échappés du monde windows (dont je fait parti) car il n'y a pas de telles choses étrange sous linux que des lettres de lecteur (A:, C:, etc...). L'idée de base est que tout support contient des données, organisés en dossier dont ce plus haut, peut être accueilli quelque part au dessous du /.

Le lecteur de disquette (si ce truc qui ne sert plus à grand chose) n'est donc pas un A:, mais un /mnt/fd, autrement dit le contenu de la disquette monté sur le dossier /mnt/fd.

Fort de ce concept, que peut t-on greffer sur notre / ? A peu près tout et n'importe quoi en réalité. De manière plus précise, nous pouvons monter n'importe quel système de fichier.

Les Systèmes de fichier

Pour mieux comprendre la suite, reprenons ensemble la base. Prenons un média quelconque, une disque, un CD ou encore une clef USB. Deux aspects le concernat peuvent être considérer : le format des données que contient le support, leur organisation, et le lecteur, le materiel, capable d'extraire du monde physique (surface magnétique, mémoire flash, etc.) les dites données.

L'organisation interne des données est appelé un système de fichier (ou FS pour File System). Cela correspond à la manière que l'on a d'organiser les 0 et les 1 de sorte à former des noms dossiers, des noms de fichier, des contenus de fichier.

Il existe autant de système de fichier qu'une homme d'église peut en consacrer... C'est à dire vraiment beaucoup... Déjà chaque OS a son propre système de fichier de CPM à DOS (la fameuse FAT), de Windows NT (NTFS voulant dire Never Twice te same File System, ou, Jamais Deux fois le même Système de Fichier) à Linux avec Ext2, Ext3 et bientôt Ext4 (qui intégrera enfin le mode Extend, c'est à dire une réservation, à priori, d'espace pour un fichier écrit, évitant ainsi la défragmentation).

Bon, la traduction de NTFS, c'est évidemment une ancienne blague venue du fait que Microsoft change sa structure à chaque version nouvelle Windows, de manière non documentée, et en perdant du coup la compatibilité descendante (un NT 4 ne pouvait pas lire un disque NTFS formaté avec le même NT 4 en SP4 par exemple).

Chaque système de fichier a des caractéristiques propres : taille de nom de fichier (la fameuse limitation 8.3 de la FAT), profondeur des dossiers, vitesse (ex. ReiserFS vs Ext3), robustesse, correction d'erreur, etc. En bref, chaque FS a son usage ou presque et tout cela nous donnes des centaines de formats différents.

Et, toujours pour les évadés Windowsiens, si notre ancien OS ne reconnaissait que péniblement deux ou trois FS (FAT, FAT32, NTFS), Linux, lui, les comprends tous, ou presque. Oui, ce n'est pas un gag, si vous avez encore dans vos tiroirs une vielles disquette Amiga, Linux peut en comprendre le format. Faut t-il encore posséder un lecteur de disquettes Amiga qui fonctionne avec nos modernes PC...

La commande mount sous tous ses angles

Mount tout nu...

Le montage se fait par la commande *mount*. Cette commande fait directement appel à une primitive du noyeau, elle demande donc (à quelques exceptions près) le privilège root pour fonctionner. Il faut bien imaginer qu'au démarrage, le "système de fichier virtuel" de Linux est vide, il en contient que / qui ne pointe sur rien. Mount va service à greffer des sources de données d'origines diverses et multiples sur ce système de fichier en commençant par le fameux /. L'idée de base est que la commande mount s'effectue toujours sur un répertoire existant, crée dans une autre ressource qui a été précédemen montée sur /. L'art du mount consistera donc à créer un dossier vide et à y accrocher une source de donnée dont le contenu prendra la place de ce dossier (qui ne sera donc plus vide).

La contre-partie de la commande *mount* est la commande *umount* qui va retirer des sources de données. Pour qu'umount fonctionne, il faut que le dossier ne soit plus utilisé par une application. Une commande pratique pour savoir si un dossier est encore utilisée, et par qui est *lsof chemin*.

Enfin la commande mount peut être exécutée sans arguments pour savoir ce qui à un moment T donnée est monté sur notre /. Dans le résultat qui suit, les numéros ont été ajoutés pour mieux se repérer par la suite.

```
    mount

(1) /dev/hda9 on / type reiserfs (rw,notail)
    /dev/hda6 on /home type ext3 (rw)

(2) /dev/sdc1 on /mnt/clef_usb type vfat (rw,noexec,nosuid,nodev,noatime,uid=516,utf8,shortname
    /dev/hdc on /mnt/cdrom type udf (ro,nosuid,nodev,users,umask=0,iocharset=utf8)

(3) distant:/data/partage_nfs on /mnt/partage_nfs type nfs (rw,addr=192.168.0.12)
    //ouingdoze/partage on /mnt/partage_smb type smbfs (rw,addr=192.168.0.12)

(4) none on /proc type proc (rw)

(5) https://www.ma_boite.com/dav/ on /mnt/ma_boite type coda (user=root)
```

mount pour les sources locales

Dans le bloc (1) apparaîssent les montages locaux. Qu'apprend t-on ici ? déjà que l'on utilise 2 partitions du disque /dev/hd (hda1 et hda9). Un montage a cependant plus d'importance que les autres : hda9. En effet, cette partition est montée directement montée sur le /. Les deux notions se confondent. Tout dossier créé directement sur / sera donc en réalité intégré dans la partition n°9 du disque hda.

Sous unix, le fait que / soit une partition d'un disque dur n'est absolument pas une nécessité. Il est tout à fait possible d'utiliser par exemple un espace mémoire sous la forme d'un disque virtuel. C'est typiquement ce qui se passe lorsque vous bootez un live CD.

On remarque aussi qu'une partition est manquante, le swap. En effet, le swap n'est pas à proprement parler géré par *mount* mais par le gestionnaire de mémoire de Linux. Il est donc logique de ne pas la voir ici (et en revanche un peu moins de la voire dans fstab comme nous nous en appercevrons plus loin).

Les autres partitions sont gréffées sur le / à des endroits stratégiques. Dans notre exemple, seule hda1 est gréffée

sur le */home* pour former un espace dédié aux données des utilisateurs. C'est un minimum qui permet de réinstaller un système en 1/4 d'heure en ne reformattant et reinstallant que le /, soit hda9. Il est ainsi possible sous Unix de remettre un système complettement à jour sans perdre ne serais-ce que le papier peind de votre bureau (essayer le même exploit sous Windows...). Ce type de montage convient à un ordinateur de type "bureau" mais pour un serveur, nous aurons beaucoup plus de partitions. Une petite pour le /, une grosser pour /usr (là où sont stockées les applications), une sur /var qui contient beaucoup de données "vivantes" du linux et que l'on ne veut pas perdre en cas de réinstallation d'urgence, le /home bien sur et aussi le /tmp qui contiendra les fichiers temporaires. Ce dernier point est crucial pour un serveur car il n'est pas rare que des petits malin se débrouillent pour saturer le /tmp. Si ce dernier était directement un dossier du / on obtiendrait rapidement une paralysie du système par saturation des resources. En mettant /tmp sur une partition dédiée nous limitons le risque à cette seule partition.

Enfin, du côté des systèmes de fichiers, nous avons de l'EXT3 et du ReiserFS. Nous n'allons pas examiner en détail ces deux formats mais juste indiquer que traditionellement EXT3 est choisi pour son ultra fiabilité (nous l'utilisons donc pour les données du /home). ReiserFS est quant à lui moins fiable (tout est relatif) et surtout beaucoup plus véloce (rapport 1/10 selon mes tests). Il est donc parfait pour la partition qui contient le /usr (la même que pour le / dans le cas d'un ordinateur de bureau) dans la mesure où ce système de fichier rendra beaucoup plus rapide le chargement des applications un peu lourdes (firefox, openoffice, etc...).

Toutes ces partitions sont montées par Linux au démarrage (c'est plus pratique ;-). Imaginons cependant que ce ne soit pas le cas et examinons pour clore ce chapitre les commandes *mount* qui auraient permis entre autre ces montages :

```
# montage d'un disque dur IDE formaté en EXT3 connecté au premier contrôleur
mount -t ext3 /dev/hda /mnt/mon_disque

# montage d'un disque dur SATA formaté en ReiserFS
mount -t reiserfs /dev/sda1 /mnt/mon_disque

# montage de la partition n°1 du second disque formatée en NTFS
mount -t ntfs /dev/hdb1 /mnt/mon_disque_sata
```

mount pour les médias amovibles

Nous allons maintenant bouturer nos médias (cd-rom, floppy, dvd, clef usb, etc..) sur notre /. Ce sont les resources (2) dans le résulat plus haut. Nous allons pour cela analyser la syntaxe de mount abordée plus haut un peu plus en détail :

```
mount –t typeFS Source Cible –o Options
```

TypFS est le type de système de fichier que l'on cherche à greffer (par exemple, FAT32 pour une disquette du monde Windows). Source est le source de données à monter et Cible est un dossier préexistant dans l'arborescence de notre fameux /. Les options sont elle totalement dépendante du type de FS que l'on cherche à monter, nous verrons cela plus loin.

La source mérite un arrêt. Il y a en gros trois grand type de source : physique, FS locale, ou FS distant. Physique, c'est le plus simple, c'est notre lecteur de disquette, un périphérique ou device. Comme toutes les périphériques, le lecteur disquette a une entrée sous la forme d'un fichier dans le dossier */dev*, c'est */dev/fd0* pour le lecteur de disquette. Ainsi, si je veut monter, une disquette formatée en FAT32, je taperais:

```
mkdir /mnt/ma_disquette
mount –t fat32 /dev/fd0 /mnt/ma_disquette
```

Généralement *mount* est suffisamment intelligent pour comprendre tout seul le type de FS pour un device, on peut ainsi racourçir la syntaxe avec :

```
mount /dev/fd0 /mnt/ma_disquette
```

Maintenant, il suffit d'aller lire le contenu du dossier */mnt/ma_disquette* pour y retrouver son contenu. Si elle n'est pas protégée en écriture, je peux écrire dans ce dossier et cela écrira sur la disquette. Ma disquette fait maintenant partie intégrante du grand /...

Une fois mes écritures terminées, je veux démonter ma disquette, c'est à dire l'enlever du /. Au grand dieu jamais on ne retire une disquette comme cela, sans prévenir sous peine de la rendre illisible !! En effet, linux prends sont temps pour écrire sur un média. En réalité il le fait quant il a le temps car tant qu'elle est montée, pour lui rien ne presse. Pour démonter ma disquette je vais utiliser la commande suivante :

```
umount /mnt/ma_disquette
```

Et c'est terminée, vous pouvez éjecter la disquette sans risque, mount s'est assuré que les données était bien écrite avant de vous rendre la main (vous pouvez faire cela vous même par la commande *sync*).

Même si mount reconnait généralement le type du média que l'on cherche à bouturer, voici malgrès tout quelques exemples de syntaxes:

```
# montage d'une floppy en FAT
mount -t fat32 /dev/fd0 /mnt/floppy

# montage d'une disque Amiga
mount -t affs /dev/fd1 /mnt/mon_jeu

# montage d'un CD-ROM
mount -t iso9660 /dev/hdc /mnt/cdrom

# montage de la partition n°1 d'une clef USB formatée en NTFS
mount -t ntfs /dev/sdb1 /mnt/usb
```

mount pour les systèmes de fichier distants

Dans les informations données plus haut par la commande *mount* la section (3) nous montre des montages un peu étrange comme un *distant:/data/partage_nfs* qui serait monté sur */mnt/partage_nfs* avec un type de fichier *nfs*. Jusqu'à maintenant nous avons monté des systèmes de fichier locaux, des médias amovibles, voyons maintenant comment monter des systèmes de fichiers auquel on accède à distance (souvent appelé des partages dans le monde de bilou).

distant:/data/partage_nfs est donc un partage distant que je monte par le protocole NFS (Network File System) sur mon /. Là cela devient très pratique car sur mon /, en fonction des entdroits où j'écris, j'impacte en réalité une ou l'autre de mes partitions dans des FS différents mais aussi un disque distant sans même m'en rendre compte.

Vous pouvez aussi monter des partages Windows, par ce système. Si par exemple, l'ordinateur du voisin, qui a une connection Wifi ouverte à tout vents, est (en plus !) sous Windows et qu'il "partage" gracieusement son dossier *ma_musique*, je peux monter ce partage sur mon / en tapant :

```
mkdir /mnt/mon_voisin_sympa
mount -t smbfs //mon_voisin/ma_musique /mnt/mon_voisin_sympa
```

Aussi simple que cela.... Voici quelques exemples de syntaxes de montage distant:

```
# Montage d'un partage en NFS
mount -t nfs 192.168.1.12:/dossier/partage /mnt/partage_nfs

# Montage d'un partage en SMB (protocole de partage des anciens windows). Attention de bien
# mettre les / dans le bon sens ;-)
mount -t smbfs //nom_machine/partage /mnt/partage_windows

# Montage d'un partage en CIFS (protocole de partage des nouveaux windows XP & co.).
mount -t cifs //nom_machine/partage /mnt/partage_windows_mieux
```

mount pour les psoeudo système de fichier

La ligne (4) donnée par ma commande *mount* est un peu spéciale (*none on /proc type proc (rw)*). En réalité les fichiers et dossier contenu dans */proc* n'existent pas. Ce sont des "vues de l'esprit". En fait, chaque fichier est une manière d'obtenir des informations sur le système (si on le lit) ou d'en change (si on écrit dedans). par sexemple, dans le cas de *proc*, il s'agit tand des informations système de linux que de son paramétrage. Par exemple, en tapant la commande suivante connaîtrez les caractéristiques de votre micro processeur :

```
cat /proc/cpuinfo
```

Il y a plusieurs autres psoeudo système de fichier au dessous de /. Nous avons */dev* pour les périphériques, */proc* et */sys* pour les informations système. Ces psoeudo systèmes de fichiers rendent plein de services qu'il serait un peu long de lister ici. Mais ce qu'il est important de comprendre c'est qu'ils sont vitales pour permettre à toutes les applications de communiquer avec le système Linux. Par exemple, si par malheur vous détruisez le dossier */dev/snd*, et bien vous n'aurez tout simplement plus accès à votre carte son. La puissance des fichiers sous Linux ;-)

mount pour les systèmes de fichier virtuels

Les systèmes de fichiers virtuels sont un peu à part (section (5) dans la liste du haut). Comme les psoeudo systèmes de fichiers, ils sont une vue de l'esprit, comme les systèmes de fichiers distants, il sont donne souvent accès à des données présentes ailleur et comme les systèmes de fichiers locaux c'est bien de fichier de données dont on parle. Par exemple un serveur SSH/SFTP (sorte de serveur FTP sécurisé) est un système de fichier virtuel, le

serveur est distant, on manipule bien des fichiers et pourtant ce n'est pas un "vrai" système de fichier. Mais on peut aller plus loin, imaginez l'encyclopédie wikipedia, si je pouvais la monter sur mon / ce serait aussi un système de fichier virtuel... Je rêve ? pas tant que cela ;-)

A l'origine, pour monter par exemple un serveur sftp sur le /, c'téit à peu prés comme pour les autres système de fichier, il fallait être root et utiliser la commande mount. Ainsi nous pouvions exécuter :

```
mount -t sshfs login@server:/dossier /mnt/serveur_sftp
```

L'inconvénient est qu'à contrario des systèmes de fichiers précédent, c'est plus souvent un utilisateur normal qu'un administrateur qui a besoin de faire de tel montage... Hors seul root a le droit d'exécuter mount... C'est sur ce constat qu'est né le projet FUSE pour Filesystem in User SpacE. Comme son nom l'indique, FUSE permet à tout à chacun, sans droit administrateur, de monter des systèmes de fichiers virtuels... En réalité FUSE est une librairie, un couche qui se place entre le noyau et l'espace utilisateur et qui est ensuite utilisé par toute application qui désire créer un système de fichier.

La syntaxe de montage est du coup différente. L'utilisation proscrite pour un non-root de *mount* est du coup remplacée par autant de commande spécialisées pour un système de fichier. Par exemple, le montage d'un serveur SSH/FTP se fait par la commande *sshfs machine: point_de_montage*.

FUSE a ouvert un grand champ de possibilité dans le bouturage sur /. En effet, vu que c'estune librairie, n'importe qui peut maintenant l'utiliser pour créer son propre système et ceci en C++, en C#, en Java, en Python, en Perl, en Ruby, etc. Du coup nous pouvons grâce à cela monter des systèmes de fichier virtuel connus comme curlftpfs, sshfs, WebDav, NTFS (et oui ;-).

Mais FUSE a aussi permis l'apparition d'un écosystème de Système de Fichier virtuels beaucoup plus exotiques comme GMailFS, BlogFS (pour éditer vos aricles de blog) , BitTorrentFS, etc... et WikipediaFS !! Oui, ce n'est pas un gag, quelqu'un a bien développé une commande utilisant FUSE et permettant de naviaguer dans Wikipedia comme s'il s'agissait de simples fichiers... Avouez que c'est assez magique !

Aussi simple que cela.... Voici quelques exemples de syntaxes de montage FUSE:

```
# Montage d'un serveur webdav
mount.davfs http://serveur_webdav -o username=identifiant,password=xxxxxx
```

```
# Montage d'un serveur ftp
curlftpfs ftp://serveur_ftm/ mon_dossier -o user=identifiant:password

# mettre les / dans le bon sens ;-)
mount -t smbfs //nom_machine/partage /mnt/partage_windows

# Montage d'un partage en CIFS (protocole de partage des nouveaux windows XP & co.).
mount -t cifs //nom_machine/partage /mnt/partage_windows_mieux
```

Automatisation des montages

Tout ceci est puissant mais peut devenir un peu fastidieux si l'on doit, à chaque redémarrage de la machine lancer 5 commandes mount pour boutturer correctement notre /. C'est pour cela qu'il existe un fichier, */etc/fstab*, qui contient la liste des montages à faire automatiquement au démarrage. Dans mon cas voici son contenu, cela se passe de commentaires :

```
# montage d'une source locale de type (1)
/dev/hda9 / reiserfs defaults 1 1

# faux montage du swap
/dev/hda7 swap swap defaults 0 0

# Montage d'un média amovible de type (2)
/dev/hdc /mnt/cdrom auto umask=0,users,iocharset=utf8,noauto,ro,exec 0 0

# Montage d'une source distante de type (3)
distant:/store/data /store/data nfs defaults 0 0

# Montage d'une psoeudo source de type (4)
none /proc proc defaults 0 0
```

Nous retrouvons les montages donné par la commande *mount* plus quelques autres. Tout d'abord */dev/hdc*, c'est le CDROM, normal qu'il n'apparaisse pas dans mount vu que je n'ai pas de CD-ROM inséré. Swap est quant à lui une partition spéciale utilisée par Linux pour étendre artificiellement la mémoire lorsqu'il en manque. Elle n'a pas de point de montage et c'est donc normal qu'elle n'apparaisse pas dans la commande *mount*.

Toujours est t-il que vous pouvez ajouter dans *fstab* autant d'entrées que vous voulez. Elles seront ainsi

automatiquement montées au démarrage.

Conclusion

Les possibilités de la commande mount sont infinies car en réalité tout développeur peu fabriquer un "driver" pour que mount puisse greffer une source à notre /. Il existe des drivers qui permettent de monter des serveurs FTP, de WebDav, un dossier distant via SSH, bref, c'est à peu près sans limites et fois que l'on a compris ce qu'est le / et la commande *mount*

Les FS aux oignons

Dernier aspect du bouturage, le projet UnionFS. J'en ai rêvé pendant des années et c'est arrivé. L'idée est assez simple. J'ai deux dossiers et je veux fusionner ces deux dossiers en un seul. Voilà ce que fait en gros unionFS.

Plus précisement, UnionFS permet d'empiler N dossiers les uns au dessus des autres. Si deux dossiers ont le même fichier au même endroit, le plus haut sera câché par le plus bas. Enfin, unionFS permet de définir qui sera impacté lors de l'écriture dans le dossier fusionné. On peut faire ainsi N-1 couches en lecture seule et une couche en écriture.

Il y a des tonnes d'applications d'un tel système. Imaginez par exemple un Linux qui boot sur CD-ROM et une clef USB. Il suffit que j'indique à UnionFS de fusionner le CD-ROM (en lecture seule) avec la Clef USB (en lecture écriture) pour disposer d'un OS qui boot n'importe où et qui fonctionne comme un vrai Linux.

Autre application, je suis administrateur système et j'ai installé des Linux tout propres sur mes postes clients. Je transforme ce linux tout neuf en couche lecture-seule et j'ajoute une couche d'écriture au dessus. Ainsi si le système devient instable (fausse manip en tant que root), j'efface la couche du dessus pour retrouver un OS tout neuf...

Pour lancer UnionFS, il faut installer les outils UnionFS (urpmi unionfs-tools) et monter (en tant que root) le noyau en mémoire :

```
modprobe unionfs
```

Ensuite, imaginons que je veuille fusionner un dossier en lecture seule */mnt/lecture* et un autre, */mnt/sda1/writable*,

pour l'écriture. Je vais écrire cela comme suit :

```
mkdir /mnt/fusion
mkdir /mnt/ecriture
mkdir /mnt/lecture
# montage de la premiere couche en lecture seule dans le dossier /mnt/fusion
mount -t unionfs ma_fusion -o DIRS=/mnt/lecture:ro /mnt/fusion
# montage de la couche en écriture via la commande des Unionfs tools
unionctl ma_fusion /mnt/fusion --add --before /mnt/lecture /mnt/ecriture --mode rw
```

Et voilà, le tour est joué, je n'ai plus qu'à travailler dans /mnt/fusion pour que tout ce que j'écrive termine dans /mnt/ecriture sans jamais altérer /mnt/lecture

Conclusion

Voilà, c'est la fin du petit dossier de bouturage sous Linux. Comme vous l'aurez compris, tout ceci n'a de limite que l'imagination...

Vos remarques et commentaires...

Akrew, le 27 octobre, 2006 - 00:33

Du jardinage au potager, en passant par la greffe "informatique" de boutures (tu devrais breveter avant l'INRA), je pense que tu nous as concocté là une belle soupe aux oignons que j'ai digéré aussi facilement que linux (di)gère ses fichiers et FS. En bon artisan que tu es, j'aimerai qualifier tout ça de modèle du genre à classer dans la catégorie : "démonstration de gastronomie numérique".

Ah si seulement les cours magistraux en école d'ingénieur étaient aussi peu ennuyeux que ton très didactique article, il y aurait moins d'élèves qui sécheraient et du coup plus de gens intéressés à linux et à l'univers du open source qui en découle etc...

Excellente explication sur les hard-links (même s'il m'a fallut relire deux fois le chapitre pour tout assimiler, et ne plus me perdre dans les "n_1"...). Je regrette toutefois, que le chapitre sur le binding ne soit pas plus étayé car je ne vois toujours pas à quoi ca sert de l'utiliser pour chrooter un nouvel environnement, et du coup je me sens un peu bête de ne pas le savoir (mais peut être le suis-je?). Tu ne nous as pas suffisamment fait rêver avec le futur FS ext4, alors je vais chercher à grapiller quelques infos sur le NET, et je n'ai pas vraiment compris en quoi NTFS étaient une catastrophe.

Avant de me risquer à une indigestion de fichier (chose que Linux ne sait apparemment pas faire aussi bien que moi), je vais tester "oignons FS" (j'adore le jeu de mots : un file système aux petits oignons). Et bizarrement, plus

j'y pense, moins je comprends toutes ces lettres associées à des périphériques sous Microsoft Windows, alors que cela me semblait si familier et normal à une certaine époque. Etrange ? une preuve de début de maturité ;) ? Souhaitons que Linux continue de les gouverner tous avec une barre (même si celle-ci est oblique ;)).

Enfin pour conclure, petite remarque personnelle qui n'interessera sans doute personne à part moi, mais on ne sait jamais :
Quand à la commande "mount", tu m'as ouvert une perspective que je n'avais pas entrevu jusqu'alors : monter un ftp distant via Internet et pas simplement via le réseau local... est-ce dangereux ? en cas de coupures de la connexion, le fait que le fichier ne soit pas démonté, ça donnerait quoi ?

Akrew, le 27 octobre, 2006 - 16:42

NTFS : Never The same File System, tout s'explique maintenant, il ne s'agit jamais du même système de fichier et les nouvelles moutures ne sont pas documentées. Est-ce la raison pour laquelle le support d'écriture NTFS pour Linux est si mal supporté et encore au stade experimental ?

Yoran, le 27 octobre, 2006 - 18:29

La raison est je pense que le support ntfs a été transféré sur FUSE, maintenant il me semble qu'il y a un module noyau NTFS 3ieme génération qui fonctionne.Mais n'utilisant pas, je le pourrais que dire des âneries.

Yoran, le 27 octobre, 2006 - 18:27

Déjà merci pour ton commentaire tout en poésie !

Sinon, j'ai suivi tes conseils et j'ai remodelé les parties dont tu parles (les n_1 revus entièrement, et surtout j'ai rajouté un chapitre sur FUSE pour les ftpfs & co.).

Pour ce qui est de monter un serveur FTP ça ne pose pas de soucis particulier. En effet, tu n'est jamais on-line en permanence. Chaque lecture de fichier et une requête et chaque écriture aussi.