Sommaire

INTRODUCTION

Objectifs

Être capable de manipuler les liens sous Unix/Linux.

Être informer sur les différents types de liens existant sur les systèmes d'exploitation modernes.

Manipulations

On va travailler dans l'arborescence suivante :

```
$HOME
      |
      - tv
         |
         - TPOS6
```

Créer l'arborescence de répertoires :

```
$ mkdir -p $HOME/tv/TPOS6
```

Se déplacer dans l'arborescence de travail :

```
$ cd $HOME/tv/TPOS6
```

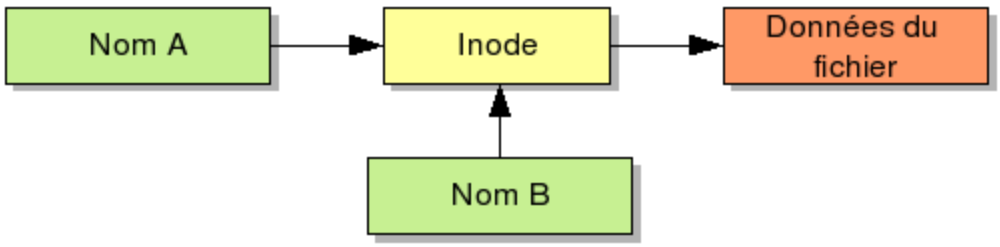
Les Liens

Sous Linux, il existe deux sortes de liens : les liens symboliques (aussi appelés *symlinks*) et les liens en dur (aussi appelés liens matériels, ou liens physiques ; en anglais *hard links*).

Les liens matériels ou liens physiques

On nomme lien matériel (en anglais *hard link*) un pointeur sur des données physiques d'un volume de stockage (c'est-à-dire d'un système de fichiers).

Avec l'introduction des systèmes UNIX, il est devenu possible d'associer plusieurs noms aux mêmes données. Bien que possédant différents noms, les données réelles sont uniques. Un compteur de références permet de savoir combien de noms pointent sur les mêmes données, et donc de savoir si l'effacement d'un nom doit être suivi d'une récupération de l'espace alloué aux données ou non : c'est le cas uniquement lorsqu'on efface le dernier nom du fichier.



Les liens matériels ne peuvent correspondre qu'à des données existantes sur le même système de fichiers. Cette restriction est contournée par un autre dispositif, celui des liens symboliques.

Sur les systèmes de type Unix (GNU/Linux, Mac OSX, BSD, etc), les liens matériels peuvent être créés par l'appel système `link()` ou par la commande `ln`.

```
$ ln un_fichier un_lien_physique_sur_un_fichier

$ ls -il
8131 -rw-r--r-- 2 root root   19 2009-10-28 11:26 un_fichier
8131 -rw-r--r-- 2 root root   19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
```

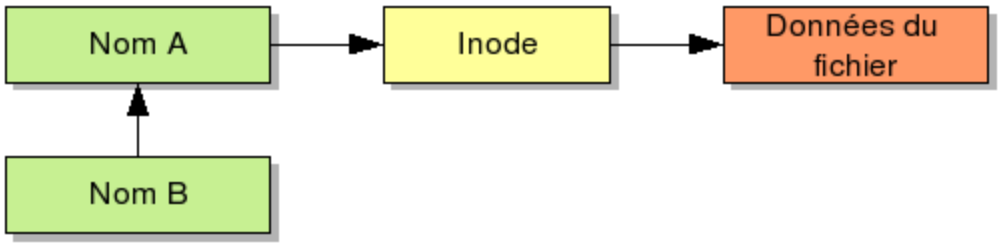
Remarque : les deux fichiers ont bien le même numéro d'inode. Les données sont donc uniques mais deux noms sont associés à ces données (ces deux références sont indiquées devant le nom du créateur/propriétaire).

Sur Microsoft Windows, les liens matériels peuvent être créés uniquement sur les systèmes de fichiers NTFS, avec fsutil hardlink ou mklink. Il existe cependant des utilitaires qui peuvent y simplifier la gestion de ce type de liens.

La suppression de fichier (`unlink` ou `rm` dans les systèmes UNIX) désassocie le nom des données présentes sur le disque. Les données sont toujours accessibles tant qu'au moins un lien pointe dessus. Lorsque ce dernier lien est supprimé, l'espace disque où se trouvaient les données est considéré comme libre, et est alors, seulement à ce moment-là, physiquement récupérable.

Les liens symboliques

Un lien symbolique (ou *symlink*) est une entrée spéciale de répertoire dans les systèmes Unix/Linux qui permet de référencer de manière quasi-transparente d'autres entrées de répertoire, typiquement, des fichiers ou des répertoires. On peut dire qu'un lien symbolique est un alias d'un fichier ou d'un répertoire.



On appelle déréférencement l'action du système d'exploitation consistant à remplacer le nom du *symlink* par celui qu'il pointe. La commande permettant de retrouver le fichier pointé par le lien est `readlink`.

Les liens symboliques sont créés par la présence de l'option `-s` dans la commande `ln` :

```
$ ln -s  nom_pointé  nom_du_symlink

$ ln -s un_fichier un_lien_sur_un_fichier
$ ls -il
8131 -rw-r--r-- 1 root root   19 2009-10-28 11:26 un_fichier
8132 lrwxrwxrwx 1 root root   10 2009-10-28 11:50 un_lien_sur_un_fichier
-> un_fichier
```

Remarque : les deux fichiers ont bien deux numéros d'inode distincts. Le fichier lien contient le nom du fichier sur lequel il pointe.

La plupart des opérations (ouverture, lecture, écriture) sur un *symlink* le déréférence automatiquement et opèrent sur sa destination (le fichier ou répertoire réel). Par contre, la suppression (`rm`) ou le déplacement/renommage (`mv`) portent sur le lien et n'affectent pas le fichier.

Un *symlink* a toujours les mêmes droits d'accès que le fichier sur lequel il pointe. En réalité, les droits d'accès indiqués pour un lien symbolique sont sans signification. La commande `chmod` déréférence toujours les fichiers qui lui sont passé en argument et il n'est donc pas possible de donner des autorisations spécifiques au *symlink* (voir séquence 3 du TP).

Contrairement aux liens matériels, les liens symboliques peuvent pointer sur des fichiers, des répertoires, sur eux-mêmes ou sur des destinations qui n'existent pas (l'existence du nom_pointé n'est même pas vérifiée lors de la création du lien par la commande `ln`). C'est seulement au moment d'accéder à un *symlink* que la vérification est faite. Quand la destination d'un lien symbolique n'existe pas, on dit que le « lien est cassé » (*broken link* en anglais). Une tentative d'ouvrir un lien cassé conduit à un message d'erreur de type « fichier non trouvé », assez surprenant car le *symlink* existe bel et bien.

Sous Windows de Microsoft, les liens symboliques sont connus sous le nom de « raccourcis ». Cependant, à la différence des systèmes Unix, ils sont implémentés sous forme de fichiers classiques portant l'extension « .lnk ». De plus, le déréférencement n'est pas effectué par le système, mais par l'interface graphique, ce qui oblige les programmes à utiliser une interface de programmation spécifique pour s'en servir. À partir de la version Windows 2000 de NTFS, il existe des « points de jonction » qui permettent de simuler le comportement des liens symboliques vers des répertoires locaux.

Utilité des liens

1 . Centraliser pour rendre l'accès plus facile

C'est l'utilisation la plus connue : par exemple le bureau (*desktop*) des interfaces graphiques que ce soit sous Windows ou sous Linux. On parle alors le plus souvent de raccourcis.

2 . Un seul exemplaire vaut mieux que plusieurs

Un document de travail, un fichier qu'on consulte souvent, il est parfois utile de l'avoir sous la main dans différents répertoires, si l'on veut pouvoir l'ouvrir très vite dans des contextes divers. Mais il est, en général, très difficile de tenir à jour plusieurs copies identiques d'un même fichier en différents points du système. Il vaut donc mieux entreposer une unique copie de ce fichier quelque part sur le disque, avec des liens qui pointent vers lui en fonction des besoins.

3 . Gagner de la place

Un lien ne prend qu'une place négligeable sur le disque, créer un lien consomme donc en général beaucoup moins d'espace sur le disque qu'effectuer une copie complète d'un fichier.

4 . Maintenir une compatibilité ascendante

Le répertoire tmp ne se trouve plus dans l'arborescence /usr mais dans /var. Grâce à un lien symbolique, on peut maintenir sa (pseudo-)présence dans /usr pour d'anciens programmes qui souhaiteraient toujours y accéder par ce chemin d'accès :

```
$ ls -l /usr/
...
lrwxrwxrwx   1 root root     10 2009-07-18 16:32 tmp -> ../var/tmp/
```

5 . Permettre l'accès à une "version par défaut"

Lorsque vous disposez sur votre machine de plusieurs versions d'un logiciel ou d'une bibliothèque, il peut être utile de définir parmi elles une « version par défaut » qui sera invoquée sauf indication contraire explicite.

A . exemple : différentes versions de gcc

```
$ ll /usr/bin/gcc
lrwxrwxrwx 1 root root     21 2009-07-18 21:15 /usr/bin/gcc ->
/etc/alternatives/gcc

# ll /etc/alternatives/gcc
lrwxrwxrwx 1 root root 18 2003-07-18 02:24 /etc/alternatives/gcc ->
/usr/bin/gcc-4.3.2

# ll /usr/bin/gcc*
lrwxrwxrwx 1 root root     21 2009-07-18 21:15 /usr/bin/gcc ->
/etc/alternatives/gcc
-rwxr-xr-x 1 root root 210404 2008-11-07 21:20 /usr/bin/gcc-4.3.2
-rwxr-xr-x 1 root root     23 2008-11-07 21:19 /usr/bin/gcc4.3-version
-rwxr-xr-x 1 root root  97776 2005-09-01 10:29 /usr/bin/gcc-4.0.1
-rwxr-xr-x 1 root root     23 2005-09-01 10:29 /usr/bin/gcc4.0-version
```

Cette technique permet d'installer une nouvelle version sans perdre l'ancienne. On peut même l'utiliser explicitement ou en recréant les liens.

B . exemple : différentes versions du noyau linux

Par exemple, il n'est pas rare que vous disposiez dans votre répertoire /boot de plusieurs versions du noyau Linux. L'une d'elles est la version par défaut, ce sera toujours celle vers laquelle pointe le lien /boot/vmlinuz.

```
$ ls -l /boot
...
lrwxrwxrwx 1 root root      29 2009-08-18 13:56 vmlinuz -> vmlinuz-
2.6.29.6-desktop-2mnb
-rw-r--r-- 1 root root 2336464 2009-04-21 01:08 vmlinuz-2.6.29.1-
desktop-4mnb
-rw-r--r-- 1 root root 2336016 2009-07-06 02:02 vmlinuz-2.6.29.6-
desktop-1mnb
-rw-r--r-- 1 root root 2336016 2009-08-17 05:28 vmlinuz-2.6.29.6-
desktop-2mnb
```

C . exemple : les bibliothèques logicielles

```
$ ls -l /usr/lib/
...
lrwxrwxrwx  1 root root       19 2006-07-22 00:55 libgconf-1.so.1 ->
libgconf-1.so.1.0.4
-rwxr-xr-x  1 root root   241104 2005-09-08 16:41 libgconf-1.so.1.0.4
lrwxrwxrwx  1 root root       19 2006-07-22 00:50 libgconf-2.so.4 ->
libgconf-2.so.4.1.0
-rwxr-xr-x  1 root root   204288 2005-12-01 14:41 libgconf-2.so.4.1.0
lrwxrwxrwx  1 root root       21 2006-07-22 00:48 libglib-1.2.so.0 ->
libglib-1.2.so.0.0.10
-rwxr-xr-x  1 root root   151852 2005-02-01 12:09 libglib-1.2.so.0.0.10
-rw-r--r--  1 root root   799086 2005-12-05 12:30 libglib-2.0.a
-rwxr-xr-x  1 root root      820 2005-12-05 12:30 libglib-2.0.la
lrwxrwxrwx  1 root root       22 2006-07-22 01:03 libglib-2.0.so ->
libglib-2.0.so.0.800.4
lrwxrwxrwx  1 root root       22 2006-07-22 00:49 libglib-2.0.so.0 ->
libglib-2.0.so.0.800.4
```

6 . Réunir en un même répertoire des liens vers des fichiers qui ont « quelque chose en commun »

Un cas classique et particulièrement sophistiqué de regroupement de liens par répertoires concerne les « niveaux d'exécution » du système. Le répertoire /etc/rc.d contient des sous-répertoires rc0.d, rc1.d etc. jusqu'à rc6.d. Chacun de ces répertoires est destiné à abriter des liens vers les scripts lançant les différents services qui doivent être activés dans le niveau d'exécution considéré (niveau 0 pour rc0.d, niveau 1 pour rc1.d etc.), ainsi que des liens vers les scripts des services qui doivent être désactivés dans ce niveau d'exécution.

L'astuce (une des astuces plutôt) est que c'est le nom même du lien qui indique au système s'il doit invoquer le script avec l'argument start pour lancer le service ou l'argument stop pour l'arrêter. Et c'est aussi le nom du lien qui détermine l'ordre dans lequel les services seront lancés et désactivés.

7 . Permettre l'accès à un fichier (même si les droits ne le permettent pas et sans avoir à copier le fichier ailleurs)

Vous pourrez accéder au fichier par un lien en dur, si vous avez les permissions pour lire le fichier lui-même : peu importe alors que le lien en dur accessible ait été créé par root à partir du répertoire d'un autre utilisateur qui vous est interdit. Le lien en dur peut donc être un bon moyen de circonvenir cette interdiction (cela peut donc être un bon moyen pour mettre à la disposition d'un autre utilisateur un fichier qui est dans votre répertoire personnel, et cela le sera d'autant plus que le fichier sera très volumineux, une image ISO par exemple, qui ainsi n'aura pas à être copiée).

En revanche, si un lien symbolique pointe vers un fichier qui se trouve dans un répertoire pour lequel vous n'avez pas de permissions d'accès appropriées, vous ne pourrez pas davantage accéder au fichier par le lien que par un accès direct à l'original.

8 . Protection contre les fichiers effacés

Etant donné qu'ils permettent de « récupérer » le contenu d'un fichier effacé, il a parfois été proposé d'utiliser les liens en dur comme une sorte de « sauvegarde » partielle. Effacer un fichier n'est irrémédiable que si ce fichier ne possède qu'un seul « nom ». Créer un lien en dur pour un fichier c'est lui donner un « nom » supplémentaire. L'idée de base du mécanisme de protection est donc de créer un lien en dur supplémentaire pour chacun des fichiers à protéger, lien qui sera stocké ailleurs sur le disque dur, bien à part : une fois ceci fait si, dans un répertoire protégé, vous effacez maintenant un fichier, alors vous pourrez encore y accéder (et le restaurer par copie) à partir du lien en dur créé précédemment.

9 . Augmenter l'espace disque

Pour éviter de re-partitionner (ou de redimensionner une partition, opération qui comporte de nombreux risques), il est possible d'augmenter l'espace libre en créant un lien symbolique vers un répertoire dans une autre partition qui en dispose.

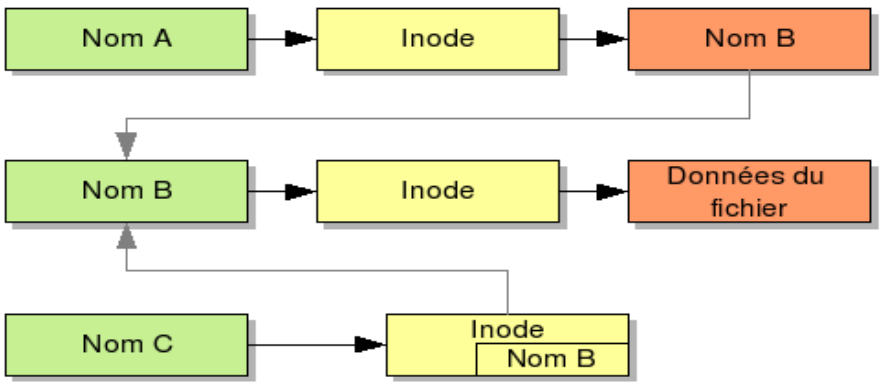
Stockage des liens symboliques

Les premières implémentations des liens symboliques traitaient les informations concernant la destination comme les données d'un fichier ordinaire. Ce fichier contenait juste la chaîne de caractère représentant le nom pointé et seul un drapeau indiquait au système de ne pas ouvrir ce fichier, mais celui dont le nom y était indiqué.

Cette méthode a l'avantage d'être simple à réaliser mais présente cependant deux inconvénients. Premièrement, chaque ouverture d'un fichier via un symlink ouvre en réalité deux fichiers, et même plus si le chemin d'accès au fichier est lui-même constitué de *symlinks*. Ces ouvertures multiples ralentissent le système. Deuxièmement, les quelques octets nécessaires au stockage du nom pointé étant considérés comme un fichier, ils occupent la place d'une unité d'allocation complète sur le disque (un bloc soit par défaut 4096 octets), ce qui entraîne un gaspillage de l'espace de stockage. Cette première implémentation a été rétroactivement appelé *slow symlink*.

Une évolution appelée *fast symlink* est à la fois plus rapide et moins dispendieuse en capacité de stockage. Elle consiste à stocker la chaîne de caractère représentant le nom pointé dans une zone supplémentaire de l'inode. Ceux-ci contenant les informations vitales du système de fichier, ils sont souvent utilisés par le système, qui les maintient donc en mémoire centrale pour leur garantir un accès immédiat. Le nom pointé faisant partie de cette structure, il bénéfice lui-aussi de cet accès rapide en mémoire, ce qui accélère grandement son déréférencement. Cependant, le système peut se replier sur l'ancienne méthode (lente) si la longueur de la chaîne de caractère dépasse les capacités limitées de stockage d'un inode (128 ou 256 octets au total pour un inode).

Dans l'exemple ci-dessous, Nom A est un *slow symlink* et Nom C est un *fast symlink*.



Exemple d'un fast symlink :

```
# ls -il

8131 -rw-rw-r-- 2 tv   tv     19 2009-10-28 11:26 un_fichier
8132 lrwxrwxrwx 1 tv   tv     10 2009-10-28 11:50 un_lien_sur_un_fichier
-> un_fichier

# echo "stat <8132>" | debugfs /dev/sdb5
debugfs:  Inode: 8132   Type: symlink    Mode:  0777   Flags: 0x0
User:   500   Group:   500   Size: 10
Links: 1   Blockcount: 0
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x4ae869ae -- Wed Oct 28 16:56:30 2009
atime: 0x4ae869b2 -- Wed Oct 28 16:56:34 2009
mtime: 0x4ae821e1 -- Wed Oct 28 11:50:09 2009
Fast_link_dest: un_fichier

# readlink un_lien_sur_un_fichier
un_fichier
```

Exemple d'un slow symlink :

```
# ln -s
../usr/../usr/local/../local/lib/../lib/xorg/../xorg/modules/drivers/int
el_drv.la un_autre_lien_sur_un_fichier

# ls -il
8133 lrwxrwxrwx 1 root root   81 2009-10-29 11:23
un_autre_lien_sur_un_fichier ->
../usr/../usr/local/../local/lib/../lib/xorg/../xorg/modules/drivers/int
el_drv.la

# echo "stat <8133>" | debugfs /dev/sdb5
debugfs:  Inode: 8133   Type: symlink    Mode:  0777   Flags: 0x0
User:     0   Group:     0   Size: 81
Links: 1   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x4ae96d20 -- Thu Oct 29 11:23:28 2009
atime: 0x4ae96d20 -- Thu Oct 29 11:23:28 2009
mtime: 0x4ae96d20 -- Thu Oct 29 11:23:28 2009
BLOCKS:
(0):36865
TOTAL: 1
```

Ce lien occupe un bloc de données (ici le numéro 36865) qui contient :

```
# dd if=/dev/sdb5 bs=4096 skip=36865 count=1 | hexdump -C
...
00000000  2e 2e 2f 75 73 72 2f 2e  2e 2f 75 73 72 2f 6c 6f
|../usr/../usr/lo|
00000010  63 61 6c 2f 2e 2e 2f 6c  6f 63 61 6c 2f 6c 69 62  |
cal/../local/lib|
00000020  2f 2e 2e 2f 6c 69 62 2f  78 6f 72 67 2f 2e 2e 2f
|/../lib/xorg/../|
00000030  78 6f 72 67 2f 6d 6f 64  75 6c 65 73 2f 64 72 69  |
xorg/modules/dri|
00000040  76 65 72 73 2f 69 6e 74  65 6c 5f 64 72 76 2e 6c  |
vers/intel_drv.l|
00000050  61 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |
a...............|
```

Ou plus simplement :

```
# readlink un_autre_lien_sur_un_fichier
../usr/../usr/local/../local/lib/../lib/xorg/../xorg/modules/drivers/int
el_drv.la
```

LES LIENS PHYSIQUES

Les liens physiques permettent d'attribuer plusieurs noms à un fichier unique.

Créer un fichier vide fic1 :

```
$ touch fic1
```

1) Indiquer son numéro d'inode et la commande qui permet de l'obtenir.

2) Créer un lien physique fic2 sur fic1. Donner la commande.

3) Indiquer son numéro d'inode et conclure sur fic1 et fic2.

4) Modifier le contenu de fic2 et visualiser alors le contenu de fic1. Expliquer.

5) On veut maintenant effacer ce fichier. Que faut-il faire ?

6) Essayer de créer un lien physique sur un répertoire. Que se passe-t-il ?

Les Liens Symboliques

Créer un fichier vide fic1 :

```
$ touch fic1
```

7) Indiquer son numéro d'inode et la commande qui permet de l'obtenir.

8) Créer un lien symbolique fic2 sur fic1. Donner la commande.

9) Indiquer son numéro d'inode et conclure sur fic1 et fic2.

10) Modifier le contenu de fic2 et visualiser alors le contenu de fic1. Expliquer.

11) On efface maintenant le fichier fic1 (contrôler avec un ls -l). Visualiser le contenu de fic2. Que se passe-t-il ?

12) Essayer de créer un lien symbolique sur un répertoire. Que se passe-t-il ?

ALLER PLUS LOIN

I. Le respect des frontières : liens et partitions

Un lien en dur ne peut pointer que vers un fichier qui se trouve sur la même partition que lui (le même système de fichiers), une telle contrainte ne vaut pas pour les liens symboliques.

Exemples :

```
# ln -s /mnt/win_c/boot.ini ./boot.ini
# vim boot.ini

# ln /mnt/win_c/boot.ini ./boot2.ini
ln: création d'un lien direct de `./boot2.ini' vers
`/mnt/win_c/boot.ini': Lien croisé de périphéque invalide
```

Tester en adaptant suivant vos partitions présentes sur votre machine (utiliser préalablement la commande df).

II. Et si on déplace l'original ?

Créer le contexte suivant (sans tenir compte des numéros d'inodes évidemment) :

```
$ ls -il
8131 -rw-r--r-- 2 tv tv   19 2009-10-28 11:26 un_fichier
8131 -rw-r--r-- 2 tv tv   19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
8132 lrwxrwxrwx 1 tv tv   10 2009-10-28 11:50 un_lien_sur_un_fichier ->
un_fichier
8124 drwxr-xr-x 2 tv tv 4096 2009-09-05 17:23 xstra1/
```

On déplace le fichier original dans un répertoire :

```
$ mv un_fichier xstra1/
$ ls -il
8131 -rw-r--r-- 2 tv tv   19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
8132 lrwxrwxrwx 1 tv tv   10 2009-10-28 11:50 un_lien_sur_un_fichier ->
un_fichier
8124 drwxr-xr-x 2 tv tv 4096 2009-10-28 16:48 xstra1/
$ ls -il xstra1/
8131 -rw-r--r-- 2 tv tv 19 2009-10-28 11:26 un_fichier
```

Bilan : si on déplace le fichier original un_fichier, le lien symbolique un_lien_sur_un_fichier devient inutilisable, en revanche le lien en dur un_lien_physique_sur_un_fichier n'est pas affecté.

On peut vérifier en éditant les liens :

```
$ vim un_lien_physique_sur_un_fichier

$ vim un_lien_sur_un_fichier
```

Si on replace le fichier original un_fichier, le lien symbolique un_lien_sur_un_fichier redevient utilisable.

```
$ mv xstra1/un_fichier

$ vim un_lien_sur_un_fichier
```

III. Et si le fichier original est dans un répertoire interdit ?

En utilisant le contexte précédent mais avec un autre compte (celui de votre binôme par exemple) :

```
$ ls -il
8131 -rw-r--r-- 2 tv   tv     19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
8132 lrwxrwxrwx 1 tv   tv     10 2009-10-28 11:50 un_lien_sur_un_fichier
-> un_fichier
8124 drwx------ 2 tv   tv   4096 2009-10-28 16:48 xstra1/

# ls -il xstra1/
8131 -rw-r--r-- 2 tv tv 19 2009-10-28 11:26 un_fichier

$ cat xstra1/un_fichier
cat: xstra1/un_fichier: Permission non accordée

$ cat un_lien_physique_sur_un_fichier
Je suis un fichier
```

IV. Copier un lien

Le contexte est le suivant :

```
$ ls -il
8131 -rw-r--r-- 2 tv   tv     19 2009-10-28 11:26 un_fichier
8131 -rw-r--r-- 2 tv   tv     19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
8132 lrwxrwxrwx 1 tv   tv     10 2009-10-28 11:50 un_lien_sur_un_fichier
-> un_fichier
```

a . Copier sans maintien du lien :

```
$ cp un_fichier un_lien_physique_sur_un_fichier un_lien_sur_un_fichier
xstra1/
$ ls -il xstra1/
8133 -rw-r--r-- 1 tv tv 19 2009-10-28 17:10 un_fichier
8134 -rw-r--r-- 1 tv tv 19 2009-10-28 17:10
un_lien_physique_sur_un_fichier
8136 -rw-r--r-- 1 tv tv 19 2009-10-28 17:10 un_lien_sur_un_fichier
8124 drwx------ 2 tv   tv   4096 2009-10-28 16:48 xstra1/
```

b . Copier en maintenant le lien :

```
$ cp -a un_fichier un_lien_physique_sur_un_fichier xstra1/
$ ls -il xstra1/
8133 -rw-r--r-- 2 tv tv 19 2009-10-28 11:26 un_fichier
8133 -rw-r--r-- 2 tv tv 19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
$ cp -d un_fichier un_lien_physique_sur_un_fichier xstra1/
$ ls -il xstra1/
8133 -rw-r--r-- 2 tv tv 19 2009-10-28 17:12 un_fichier
8133 -rw-r--r-- 2 tv tv 19 2009-10-28 17:12
un_lien_physique_sur_un_fichier
```

c . Le lien symbolique copié devient un fichier « normal » :

```
$ cp un_lien_sur_un_fichier xstra1/
$ ls -il xstra1/
8133 -rw-r--r-- 1 tv tv 19 2009-10-28 17:17 un_lien_sur_un_fichier
```

d . Le lien symbolique copié reste un lien symbolique :

```
$ cp -a un_fichier un_lien_sur_un_fichier xstra1/
$ ls -il xstra1/
8133 -rw-r--r-- 1 tv tv 19 2009-10-28 11:26 un_fichier
8134 lrwxrwxrwx 1 tv tv 10 2009-10-28 17:18 un_lien_sur_un_fichier ->
un_fichier
$ cp -d un_fichier un_lien_sur_un_fichier xstra1/
$ ls -il xstra1/
8133 -rw-r--r-- 1 tv tv 19 2009-10-28 17:23 un_fichier
8134 lrwxrwxrwx 1 tv tv 10 2009-10-28 17:23 un_lien_sur_un_fichier ->
un_fichier
```

Par contre, attention si on ne copie que le lien, il devient inutilisable :

```
$ cp -a un_lien_sur_un_fichier xstra1/
$ ls -il xstra1/
8133 lrwxrwxrwx 1 tv tv 10 2009-10-28 17:24 un_lien_sur_un_fichier ->
un_fichier (lien brisé !)
```

Aller plus loin : faire un man cp pour décomposer l'option -a (-a = -dpPR) et les options -p, -d, -R, -P, -L et --preserve=

V. Prendre en compte le lien ou ce vers quoi il pointe ?

Prenons deux exemples très simples. Ces deux exemples impliquent des liens symboliques pointant vers des fichiers (et non des répertoires).

La commande rm qui permet d'effacer un fichier, appliquée à un argument qui a le statut de lien symbolique pointant vers un fichier, effacera le lien lui-même et n'affectera pas le fichier vers lequel pointait ce lien (ce qui paraît d'une prudence raisonnable).

La commande cat, au contraire, qui envoie vers la « sortie standard » (stdout), c'est-à-dire le plus souvent vers l'écran, le contenu d'un fichier texte qu'on lui passe en argument, affichera le contenu du fichier texte vers lequel pointerait un lien qu'on lui passerait en argument (elle n'afficherait pas les quelques octets contenus dans le lien lui-même, pour cela on utilise la commande readlink).

rm opère donc, du moins dans ce cas, sur le lien lui-même, (attention : si le lien pointe vers un répertoire, il en va différemment !), tandis que cat opère sur le fichier vers lequel pointe le lien.

Attention au comportement de cd et de la barre oblique (/) en fin de nom dans les cas des liens symboliques vers des répertoires.

Vous pouvez mettre en oeuvre un certain nombre de manipulations pour vérifier les différents comportements énoncés ci-dessus. Certaines commandes proposent des options pour activer ou désactiver le déférencement.

VI. Les droits d'accès sur les liens

Tester les situations suivantes :

```
$ chmod 444 un_fichier

$ ls -il
8131 -r--r--r-- 2 tv   tv     19 2009-10-28 11:26 un_fichier
8131 -r--r--r-- 2 tv   tv     19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
8132 lrwxrwxrwx 1 tv   tv     10 2009-10-28 11:50 un_lien_sur_un_fichier
-> un_fichier

$ chmod 664 un_lien_sur_un_fichier

$ ls -il
8131 -rw-rw-r-- 2 tv   tv     19 2009-10-28 11:26 un_fichier
8131 -rw-rw-r-- 2 tv   tv     19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
8132 lrwxrwxrwx 1 tv   tv     10 2009-10-28 11:50 un_lien_sur_un_fichier
-> un_fichier
```

Conclure sur les droits d'accès sur les liens.

VII. Afficher le contenu d'un lien symbolique :

```
$ readlink un_lien_sur_un_fichier
un_fichier
```

Qu'aurait affiché la commande cat un_lien_sur_un_fichier ?

VIII. Le nombre de liens physiques

La commande ls affiche devant le nom du propriétaire d'un fichier le nombre de « noms » (le nombre de liens physiques) que ce fichier possède : chaque création d'un nouveau lien en dur incrémente (augmente) donc ce nombre d'une unité. A noter pour un répertoire : la valeur affichée indique le nombre des sous-répertoires qu'il contient (inclus les deux répertoires cachés bien connus que tout répertoire contient sous Linux : le répertoire . et le répertoire ..) et dans ce cas le décompte ne concerne pas des liens en dur.

Commenter les valeurs en gras :

```
$ ls -Ail
8135 drwx------ 2 root root 4096 2009-10-25 17:07 kde-root/
8121 drwxr-xr-x 4 root root 4096 2009-09-08 09:25 pear/
8131 -rw-rw-r-- 2 tv   tv     19 2009-10-28 11:26 un_fichier
8131 -rw-rw-r-- 2 tv   tv     19 2009-10-28 11:26
un_lien_physique_sur_un_fichier
8132 lrwxrwxrwx 1 tv   tv     10 2009-10-28 11:50 un_lien_sur_un_fichier
-> un_fichier
```