Linux Ext2fs Undeletion mini-HOWTO

Aaron Crane, *aaronc@pobox.com* v1.3, 2 Février 1999

(Adaptation française par *Miodrag Vallat* `mailto:miodrag@multimania.com` , anciennement par *Géraud Canet* `mailto:canet@labri.u-bordeaux.fr` et *Sylviane Regnault* `mailto:regnault@axpcalc.crpp.u-bordeaux.fr`). Imaginez un peu. Vous avez passé les trois derniers jours sans dormir, sans manger, sans même prendre une douche. Votre bidouillomanie compulsive a enfin porté ses fruits : vous avez achevé ce programme qui vous apportera gloire et admiration du monde entier. Allez, plus qu'à archiver tout ça et l'envoyer à Metalab. Ah, et puis virer toutes ces sauvegardes automatiques d'Emacs. Alors vous tapez `rm * ~`. Et, trop tard, vous remarquez l'espace en trop. Vous avez détruit votre *oeuvre maîtresse* ! Mais, heureusement, vous avez de l'aide à portée de main. Ce document présente une discussion de la récupération de fichiers supprimés depuis le Second système de fichiers étendu ext2fs. Espérez, peut-être pourrez-vous distribuer votre programme malgré tout...

Contents

1 Introduction

Ce mini-HOWTO tente de fournir un certain nombre de conseils dans le but de récupérer des fichiers supprimés depuis un système de fichiers ext2fs. Il contient également une petite discussion sur les manières de commencer par éviter de supprimer des fichiers.

Mon but est naturellement d'en faire une référence utile à tous ceux qui ont eu un, disons... accident avec `rm` ; mais cependant je souhaite que les gens le lisent de toute façon. On ne sait jamais : un jour, les renseignements donnés ici pourraient vous sauver la couenne.

La lecture de ce texte suppose un minimum de connaissances sur les systèmes de fichiers Unix ; je me suis cependant efforcé de le rendre accessible à la plupart des utilisateurs de Linux. Si vous êtes un grand débutant, je crains que la récupération de fichiers sous Linux *exige* certaines connaissances techniques, ainsi que de la persévérance, au moins dans l'état actuel des choses.

Il vous sera impossible de récupérer des fichiers supprimés depuis un système de fichiers ext2 sans au moins un accès en lecture au périphérique (fichier spécial) sur lequel le fichier était placé. En général, cela signifie que vous devez être *root*, mais plusieurs distributions (comme *Debian GNU/Linux*) disposent d'un groupe `disk` dont les membres ont ces accès. Vous aurez également besoin de la commande `debugfs`, du paquetage `e2fsprogs`, qui devrait avoir été installé par votre distribution.

Pourquoi ai-je écrit ceci ? Principalement par expérience personnelle, souvenir du désastre d'un `rm -r` particulièrement insensé en tant que *root*. J'ai supprimé 97 fichiers JPEG dont j'avais besoin et que je ne pouvais certainement pas récupérer par ailleurs. Suivant quelques conseils (voir la section 15 (Remerciements et bibliographie)) et en persévérant beaucoup, j'ai récupéré 91 fichiers intacts. Je suis parvenu à en retrouver, au moins en partie, cinq autres (suffisamment pour voir quelle était l'image représentée par chacun). Une seule n'était pas affichable, et même pour celle-là, je suis certain de n'avoir pas perdu plus de 1024 octets (mais hélas depuis le début du fichier ; sachant que je ne connais rien du format de fichier JFIF j'ai vraiment fait ce que j'ai pu).

Je discuterai plus bas du taux de récupération que vous pouvez espérer pour les fichiers supprimés.

1.1 Historique des révisions

Les révisions de ce document (en version anglaise, NdT) délivrées au public, ainsi que leurs dates de publication, sont les suivantes :

- v1.0, 18 janvier 1997 ;
- v1.1, 23 juillet 1997 (voir 1.1.1 (Nouveautés v1.1)) ;
- v1.2, 4 août 1997 (voir 1.1.2 (Nouveautés v1.2)) ;

- v1.3, 2 février 1999 (voir 1.1.3 (Nouveautés v1.3)).

1.1.1 Nouveautés de la version 1.1

Quels sont les nouveautés de cette version ? Primo, la réflexion dans l'exemple de la récupération de fichiers a été corrigée. Merci à tous ceux qui m'ont écrit pour me signaler mon erreur ; cela m'apprendra, je l'espère, à faire plus attention en inventant des séquences interactives.

Secundo, la discussion sur le modèle de système de fichier Unix a été récrite afin d'être (espérons-le) plus compréhensible. Je n'en étais pas entièrement satisfait de prime abord, et d'aucuns se sont plaints de son manque de clarté.

Tertio, le gros-tas-de-tar-gzip-uu-encodé de `fsgrab` au milieu du fichier a été retiré. Le programme est désormais disponible sur *ma page* `http://pobox.com/~aaronc/tech/fsgrab-1.2.tar.gz` et sur

Metalab `http://metalab.unc.edu/pub/Linux/utils/file/`

(et ses miroirs).

Quarto, le document a été traduit en langage sgml, utilisé par le Linux Documentation Project. Ce langage peut être facilement converti en un grand nombre d'autres langages (y compris HTML et LaTeX) pour un affichage et une impression simples et pratiques. Cela a pour avantage une belle typographie, dans le cas d'une édition papier ; de plus, le document contient des références et des liens bien commodes si vous le consultez sur le Web.

1.1.2 Nouveautés de la version 1.2

Cette révision est plutôt une augmentation. Elle inclut principalement des modifications proposées par des lecteurs, dont l'une est particulièrement importante.

Le premier changement a été suggéré par Egil Kvaleberg *egil@kvaleberg.no* , qui a signalé la commande `dump` dans `debugfs`. Merci encore, Egil.

Le second changement a été de signaler l'utilisation de `chattr` pour éviter de supprimer des fichiers importants. Merci à Herman Suijs *H.P.M.Suijs@kub.nl* de l'avoir signalé.

Le résumé a été revu. Des URLs ont été ajoutées, qui indiquent des organisations ou des logiciels. Ajoutez à cela quelques modifications mineures (dont des corrections de fautes de frappe, etc.).

1.1.3 Nouveautés de la version 1.3

Bien qu'il se soit écoulé 17 mois depuis la dernière version, bien peu de choses ont changé. Cette version corrige quelques erreurs mineures (fautes de frappe, URL incorrectes, etc – principalement le non-lien vers l'Open Group), et les quelques paragraphes qui étaient devenus atrocement démodés, comme ceux sur les versions de noyau et `lde`, ont été revus. Oh, et j'ai remplacé 'Sunsite' par 'Metalab' partout.

Cette version sera probablement la dernière avant la version 2.0, qui sera un vrai HOWTO, du moins je l'espère. J'ai travaillé sur des changements d'importance qui méritent l'incrémentation du numéro de version majeure.

1.2 Où trouver ce document

La version officielle la plus récente de ce document devrait être disponible au format texte auprès du site du

Linux Documentation Project `http://metalab.unc.edu/LDP/`

(et ses miroirs). La dernière version est également disponible sur

ma page `http://pobox.com/~aaronc/` sous divers formats :

- *source SGML* `http://pobox.com/~aaronc/tech/e2-undel/howto.sgml` , tel que je l'ai écrit ;
- *HTML* `http://pobox.com/~aaronc/tech/e2-undel/html/` , généré automatiquement depuis le source SGML ;
- *format texte* `http://pobox.com/~aaronc/tech/e2-undel/howto.txt` , également généré automatiquement depuis le source SGML.

2 Comment ne pas supprimer de fichiers

Il est vital de se rappeler que Linux n'est pas semblable à MS-DOS en matière de récupération de données. Pour MS-DOS (et son bâtard Windows 95), il est généralement très simple de récupérer un fichier supprimé : le « système d'exploitation » (il faut le dire vite) est même accompagné d'un utilitaire qui automatise la procédure. Ce n'est pas le cas de Linux.

Donc... règle numéro un (ou première directive, si vous préférez) :

FAITES DES SAUVEGARDES

peu importe comment. Pensez à toutes vos données. Peut-être, comme moi, conservez-vous plusieurs années d'archives de messages, contacts, documents sur votre ordinateur. Pensez au chamboulement dans votre vie si vous étiez victime d'une panne de disque catastrophique, ou – pire encore ! – si un cracker nettoyait votre disque sans vergogne. Ce n'est pas si improbable ; j'ai correspondu avec un bon nombre de gens placés dans une telle situation. J'exhorte les utilisateurs sensés de Linux de sortir acheter un périphérique de sauvegarde, de planifier leurs sauvegardes dans un emploi du temps digne de ce nom et de *s'y conformer*. En ce qui me concerne, je me sers d'un disque dédié sur une deuxième machine, et régulièrement je fais un mirroir de mon répertoire personnel par le réseau. Pour plus d'information sur la planification des sauvegardes, lisez Frisch (1995) (voir la section 15 (Bibliographie et remerciements)).

En l'absence de sauvegardes, que faire (en fait, même en présence de sauvegardes : dans le cas de données importantes, la ceinture et les bretelles, ce n'est pas du luxe) ?

Essayez de donner aux fichiers importants les droits 440 (ou moins) : ne pas vous laisser les droits en écriture provoque une demande de confirmation explicite de `rm` avant la destruction (mais si je veux supprimer récursivement un répertoire avec `rm -r`, j'interromprai le programme dès la première ou deuxième demande de confirmation pour relancer la commande avec `rm -rf`).

Un bon truc, pour les fichiers importants, est de créer un lien physique vers eux dans un répertoire caché. J'ai entendu parler d'un administrateur système qui, périodiquement, supprimait accidentellement `/etc/passwd` (et par là-même détruisait à moitié le système). Un des remèdes fut de lancer en tant que *root* quelque chose comme :

```
# mkdir /.backup
# ln /etc/passwd /.backup
```

Il est alors assez difficile de supprimer complètement le contenu du fichier : si vous dites

```
# rm /etc/passwd
```

alors

```
# ln /.backup/passwd /etc
```

permettra de le récupérer. Naturellement, cela ne couvre pas le cas où vous avez écrasé le contenu du fichier par un autre fichier, donc de toutes façons gardez vos sauvegardes.

Dans un système de fichiers ext2, il et possible d'utiliser les attributs ext2 dans le but de protéger ses données. Ces attributs sont manipulés à l'aide de la commande `chattr`. Il y a un attribut « ajout seulement » (*append-only*) : il est possible d'ajouter des données à un fichier ayant cet attribut, mais pas de le supprimer, et le contenu du fichier ne peut pas être écrasé. Si un répertoire a cet attribut, tous les fichiers et répertoires qu'il contient peuvent être normalement modifiés, mais aucun fichier ne peut être supprimé. Cet attribut peut être placé en tapant

```
$ chattr +a FICHIER...
```

Il existe aussi un attribut « immuable » (*immutable*), qui ne peut être placé ou retiré qu'en tant que *root*. Un fichier ou répertoire ayant cet attribut ne peut être ni modifié, ni supprimé, ni renommé, ni se faire ajouter un lien (physique). Il peut être placé comme suit :

```
# chattr +i FICHIER...
```

Ext2fs fournit également l'attribut « récupérable » (*undeletable*, option `+u` de `chattr`). Si un fichier ayant cet attribut est supprimé, mais pas réellemnt réutilisé, il est déplacé vers un « endroit sûr » afin d'être supprimé plus tard. Hélas, cette fonctionnalité n'est pas encore implantée dans les noyaux courants ; et bien que, par la passé, il y ait eu un peu d'intérêt concernant une implantation éventuelle, elle n'est pas (à ma connaissance) disponible pour les noyaux actuels.

Certains défendent l'idée de faire de `rm` un alias ou une fonction du gestionnaire de commandes qui exécute en fait `rm -i` (qui demande confirmation pour *tous* les fichiers à supprimer). En effet, certaines versions de la distribution

Red Hat `http://www.redhat.com/` le font par défaut pour tous les utilisateurs, y compris *root*. En ce qui me concerne, je ne supporte pas les logiciels incapables de tourner tous seuls, je ne le fais donc pas. Par ailleurs, un jour ou l'autre, vous ferez tourner le programme en mode mono-utilisateur, ou utiliserez un gestionnaire de commandes différent, ou simplement une autre machine, où votre fonction `rm` n'existera pas. Si vous vous attendez à une confirmation, il est assez facile d'oublier où vous êtes et spécifier un peu trop de fichiers à supprimer. De même, les divers scripts et programmes servant à remplacer `rm` sont, à mon humble avis, très dangereux.

Une solution un peu meilleure serait de commencer à utiliser un paquetage qui manipulerait une destruction « recyclable » en fournissant une commande qui ne s'appellerait pas `rm`. Pour plus de détails, voir Peek *et al* (1993) (voir la section 15 (Bibliographie et remerciements)). Cette solution a cependant l'inconvénient d'encourager les utilisateurs à avoir une attitude nonchalante vis-à-vis de la destruction, au lieu de l'attitude circonspecte qui est souvent nécessaire sous Unix.

3 À quel taux de récupération puis-je m'attendre ?

Ça dépend. Parmi les problèmes concernant la récupération de fichiers dans un système d'exploitation de haute qualité, multi-tâches et multi-utilisateurs comme Linux, il se trouve que vous ne savez jamais quand quelqu'un veut écrire sur le disque. Donc, quand le système d'exploitation reçoit l'ordre de supprimer un fichier, il suppose libres les blocs utilisés par ce fichier au moment d'allouer de nouveau de la place pour un nouveau fichier (c'est un exemple typique d'un principe général d'Unix : le noyau et les outils associés supposent que les utilisateurs ne sont pas des idiots). En général, plus votre machine est utilisée, moins vous avez de chances de récupérer vos fichiers avec succès.

De plus, la fragmentation du disque peut affecter la facilité de récupération. Si la partition contenant les fichiers supprimés est très fragmentée, vous avez peu de chances de pouvoir lire un fichier entier.

Si votre machine, comme la mienne, est effectivement une station destinée à un seul utilisateur, et que vous n'utilisiez pas intensivement le disque au moment fatal de la destruction, je m'attendrais à un taux de récupération du même ordre de grandeur que décrit précédemment. J'ai récupéré presque 94 % des fichiers, intacts (et il s'agissait de fichiers binaires, notez bien). Si vous obtenez plus de 80 %, vous pouvez être plutôt content de vous.

4 Bon, alors comment je récupère un fichier ?

La procédure consiste principalement en la recherche de données dans le périphérique de la partition en mode caractère, et en le fait de la rendre à nouveau visible par le système d'exploitation. Il y a principalement deux manières de le faire : la première consiste à modifier le système de fichier existant de telle façon que les inodes supprimés aient leur indicateur « supprimé » retiré, et espérer que les données retombent comme par magie à leur place. L'autre méthode, plus sûre mais plus lente, est de rechercher où se trouvent les données dans la partition et de les écrire dans un nouveau fichier.

Vous devez suivre plusieurs étapes avant de commencer votre tentative de récupération ; voir les sections 5 (Démonter le système de fichiers), 6 (Préparer la modification directe des inodes) et 7 (Préparer l'écriture à un autre endroit) pour plus de détails. Pour découvrir comment récupérer réellement vos fichiers, voir les sections 8 (Trouver les inodes supprimés), 9 (Obtenir des détails sur les inodes), 10 (Récupérer des blocs de données) et 11 (Modifier les inodes directement).

5 Démonter le système de fichiers

Quelle que soit la méthode que vous choisissiez, la première étape consiste à démonter le système de fichiers contenant les fichiers supprimés. Je vous conseille fortement de réfréner toute envie de bricoler un système de fichiers monté. Cette étape doit être effectuée *le plus tôt possible*, dès que vous vous êtes rendu compte que les fichiers sont supprimés.

La méthode la plus simple est la suivante : en supposant que les fichiers supprimés soient dans la partition `/usr`, tapez :

```
# umount /usr
```

Vous pouvez cependant avoir besoin de garder certaines données disponibles dans `/usr`. Dans ce cas, remontez-le en mode lecture seule :

```
# mount -o ro,remount /usr
```

Si les fichiers supprimés étaient dans la partition racine, vous devrez ajouter une option `-n`, afin d'empêcher que l'opération de montage ne déclenche une écriture dans `/etc/mtab` :

```
# mount -n -o ro,remount /
```

Indépendamment de tout cela, il est possible qu'un autre processus utilise à ce moment-là ce système de fichier (ce qui fera échouer le montage avec une erreur du genre *resource busy*). Il y a un programme qui peut envoyer un signal à tout processus utilisant un fichier ou point de montage donné : c'est `fuser`. Pour la partition `/usr`, essayez ceci :

```
# fuser -v -m /usr
```

Cela aura pour effet d'afficher la liste des processus concernés. En admettant qu'aucun d'entre eux n'est vital, vous pouvez taper

```
# fuser -k -v -m /usr
```

afin d'envoyer à chaque processus un `SIGKILL` (qui le tuera d'autorité), ou, par exemple,

```
# fuser -k -TERM -v -m /usr
```

pour envoyer plutôt à chacun un `SIGTERM` (qui priera le processus de terminer proprement).

6 Préparer la modification directe des inodes

Mon conseil ? Ne faites pas ça. Je ne pense vraiment pas qu'il soit raisonnable d'espérer un résultat en jouant avec un système de fichiers à un si bas niveau. Du reste, vous ne pourrez récupérer de façon fiable que les 12 premiers blocs de chaque fichier. Donc, si vous avez des fichiers longs à récupérer, vous devrez de toute façon utiliser l'autre méthode (mais lisez tout de même la section 12 (Cela va-t-il se simplifier dans l'avenir˜?) pour plus d'information).

Si vous sentez que vous devez le faire de cette manière, je vous conseille de copier les données de la partition en mode caractère dans une autre partition, puis monter le tout en utilisant le périphérique boucle (*loopback device*) :

```
# cp /dev/hda5 /root/working
# mount -t ext2 -o loop /root/working /mnt
```

(Notez que les anciennes versions de `mount` peuvent avoir des problèmes pour faire cela. Si votre `mount` ne fonctionne pas, je vous recommande fortement de vous procurer la dernière version, ou tout au moins la version 2.7, car plusieurs versions plus anciennes ont de graves problèmes de sécurité).

Le but de la manoeuvre est que, quand vous aurez entièrement détruit le système de fichiers (ce que vous ferez très probablement), il ne vous restera plus qu'à copier la partition dans l'autre sens et repartir à nouveau.

7 Préparer l'écriture à un autre endroit

Vous devez vous assurer d'avoir quelque part une partition de secours. Espérons-le, votre système a plusieurs partitions : peut-être une racine, une `/usr`, et une `/home`. Avec tout ce choix, aucun problème : créez simplement un nouveau répertoire dans l'une d'entre elles.

Si vous n'avez qu'une partition racine dans laquelle vous fourrez tout, ça risque d'être un poil plus délicat. Peut-être avez-vous une partition MS-DOS ou Windows que vous pourriez utiliser ? Ou vous avez le gestionnaire *ramdisk* dans votre noyau, peut-être en module ? Pour utiliser le *ramdisk* (en supposant que votre noyau soit plus récent que 1.3.48), tapez les commandes suivantes :

```
# dd if=/dev/zero of=/dev/ram0 bs=1k count=2048
# mke2fs -v -m 0 /dev/ram0 2048
# mount -t ext2 /dev/ram0 /mnt
```

Cela a pour effet de créer un volume *ramdisk* de 2 Mo, et de le monter en `/mnt`.

Un petit mot d'avertissement : si vous utilisez `kerneld` (ou son remplaçant `kmod` avec les noyaux 2.2.x et les derniers 2.1.x) pour charger et décharger automatiquement les modules du noyau, alors ne démontez pas le *ramdisk* tant que vous n'avez pas copié tous les fichiers qu'il contient sur un support non volatile. Une fois que vous l'aurez démonté, `kerneld` suppose qu'il peut décharger le module (après la période d'attente habituelle), et, dès qu'il l'a fait, la mémoire est réutilisée par d'autres éléments du noyau, causant la perte irrémédiable des heures de travail que vous aurez passées à récupérer soigneusement vos données.

Si vous avez un lecteur Zip, Jaz, ou LS-120, ou quelque chose d'équivalent, il s'agit probablement d'une bonne place pour une partition de secours. Sinon, il faudra faire avec les disquettes.

Une autre chose dont vous devriez avoir besoin est un programme capable de lire les données nécessaires en plein milieu du périphérique contenant la partition. À la rigueur, `dd` pourrait le faire, mais pour lire à partir de, disons, 600 Mo dans une partition de 800 Mo, `dd` tient à lire les 600 premiers mégaoctets, quitte à les ignorer, et il va y passer un temps non négligeable, même sur des disques rapides. Pour éviter cela, j'ai écrit un programme qui peut se positionner en plein milieu de la partition. Il s'appelle `fsgrab` ; vous pouvez trouver le paquetage des sources sur *ma page* `http://pobox.com/~aaronc/tech/fsgrab-1.2.tar.gz` , ou sur *Metalab* `http://metalab.unc.edu/pub/Linux/utils/file/`

(et ses miroirs). Si vous souhaitez utiliser cette méthode, la suite de ce mini-HOWTO suppose que vous avez `fsgrab`.

Si aucun des fichiers que vous voulez récupérer n'occupe plus de 12 blocs (où un bloc occupe habituellement un kilooctet), alors vous n'aurez pas besoin de `fsgrab`.

Si vous avez besoin de `fsgrab` mais n'en voulez pas, il est fort simple de traduire une ligne de commande avec `fsgrab` en une avec `dd`. Si on a

```
fsgrab -c count -s skip device
```

alors la commande `dd` correpondante (et généralement beaucoup plus lente) est

```
dd bs=1k if=device count=count skip=skip
```

Je dois vous avertir que, bien que `fsgrab` ait parfaitement fonctionné pour moi, je ne puis prendre aucune responsabilité sur son comportement. C'était vraiment une bidouille rapide et sale pour arriver à mes fins. Pour plus de détails sur l'absence de garantie, consultez la section *No Warranty* dans le fichier `COPYING` inclus dans la distribution (li s'agit de la GPL, la licence publique générale GNU).

8 Trouver les inodes supprimés

L'étape suivante consiste à demander au système de fichiers quels inodes ont été récemment libérés. C'est une tâche que vous pouvez accomplir avec `debugfs`. Lancez `debugfs` avec le nom du périphérique sur lequel le système de fichiers réside :

```
# debugfs /dev/hda5
```

Si vous souhaitez modifier les inodes directement, ajoutez une option `-w` de manière à activer l'écriture sur le système de fichiers :

```
# debugfs -w /dev/hda5
```

La commande `debugfs` permettant de trouver les inodes détruits est `lsdel`. Donc, tapez la commande suivante à l'invite :

```
debugfs:  lsdel
```

Après moult grincements et gémissements du disque, une longue liste est envoyée par un *pipe* à votre *pager* favori (la valeur de `$PAGER`). Maintenant vous aurez envie d'en sauver une copie autre part. Si vous avez `less`, vous pouvez taper `-o` suivi du nom du fichier qui devra contenir le résultat. Sinon, vous devrez vous arranger pour envoyer la sortie ailleurs. Essayez ceci :

```
debugfs:  quit
# echo lsdel | debugfs /dev/hda5 > lsdel.out
```

Maintenant, d'après la date et l'heure de la suppression, la taille, le type et les indications numériques des permissions et propriétaire, vous devez deviner quelles inodes supprimés vous voulez. Avec un peu de chance, vous les trouverez tout de suite parce c'est le gros paquet que vous avez supprimé il y a à peine cinq minutes. Sinon, prenez bien garde en allant pêcher dans la liste.

Je vous suggère, autant que possible, d'imprimer la liste des inodes que vous voulez récupérer. Cela vous facilitera nettement la vie.

9 Obtenir des détails sur les inodes

`debugfs` a une commande `stat`, qui imprime des détails sur un inode. Utilisez la commande pour chacun des inodes de votre liste à récupérer. Par exemple, si vous êtes intéressé par l'inode numéro 148003, essayez ceci :

```
debugfs:  stat <148003>
Inode: 148003   Type: regular    Mode:  0644   Flags: 0x0   Version: 1
User:   503   Group:   100   Size: 6065
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 12
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x31a9a574 -- Mon May 27 13:52:04 1996
atime: 0x31a21dd1 -- Tue May 21 20:47:29 1996
mtime: 0x313bf4d7 -- Tue Mar  5 08:01:27 1996
dtime: 0x31a9a574 -- Mon May 27 13:52:04 1996
BLOCKS:
594810 594811 594814 594815 594816 594817
TOTAL: 6
```

Si vous avez de nombreux fichiers à récupérer, vous souhaiterez automatiser tout cela. En suposant que votre liste (d'après `lsdel`) d'inodes à récupérer est dans `lsdel.out`, essayez ceci :

```
# cut -c1-6 lsdel.out | grep "[0-9]" | tr -d " " > inodes
```

Ce nouveau fichier `inodes` contient uniquement les numéros des inodes à récupérer, à raison d'un par ligne. On le sauvegarde parce qu'il va nous être sûrement très utile par la suite. Il ne vous reste plus qu'à taper :

```
# sed 's/^.*$/stat <\0>/' inodes | debugfs /dev/hda5 > stats
```

et `stats` contient la sortie de toutes les commandes `stat`.

10 Récupérer les blocs de données

Cette partie est soit très facile, soit nettement moins, selon que les fichiers que vous essayez de récupérer occupent moins ou plus de 12 blocs.

10.1 Les fichiers courts

Si le fichier n'occupait pas plus de 12 blocs, alors les numéros de blocs où sont situées toutes ses données sont écrits dans l'inode : vous pouvez les lire directement sur la sortie de `stat` correspondant à l'inode. De surcroît, `debugfs` a une commande qui automatise cette tâche. Pour reprendre l'exemple précédent :

```
debugfs:  stat <148003>
Inode: 148003   Type: regular    Mode:  0644   Flags: 0x0   Version: 1
User:   503   Group:   100   Size: 6065
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 12
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x31a9a574 -- Mon May 27 13:52:04 1996
atime: 0x31a21dd1 -- Tue May 21 20:47:29 1996
mtime: 0x313bf4d7 -- Tue Mar  5 08:01:27 1996
dtime: 0x31a9a574 -- Mon May 27 13:52:04 1996
BLOCKS:
594810 594811 594814 594815 594816 594817
TOTAL: 6
```

Ce fichier a six blocs. Puisqu'il est en-dessous de la limite des 12, nous demandons à `debugfs` d'écrire le fichier dans un nouvel endroit, comme par exemple `/mnt/recovered.000` :

```
debugfs:  dump <148003> /mnt/recovered.000
```

Bien sûr, on peut faire ça aussi avec `fsgrab` ; je le montre ici en guise d'exemple d'utilisation :

```
# fsgrab -c 2 -s 594810 /dev/hda5 > /mnt/recovered.000
# fsgrab -c 4 -s 594814 /dev/hda5 >> /mnt/recovered.000
```

Que ce soit avec `debugfs` ou avec `fsgrab`, il y aura un peu de déchet à la fin de `/mnt/recovered.000`, mais ce n'est pas très important. Si vous voulez vous en débarrasser, la méthode la plus simple est de prendre le champ `Size` de l'inode, et le brancher sur l'option `bs` d'une ligne de commande `dd`.

```
# dd count=1 if=/mnt/recovered.000 of=/mnt/resized.000 bs=6065
```

Bien sûr, il est possible qu'un ou plusieurs blocs où était écrit votre fichier aient été écrasés. Si c'est le cas, pas de chance : le bloc est mort et enterré (rendez-vous compte, si seulement vous aviez démonté plus tôt !).

10.2 Les fichiers plus longs

Les problèmes apparaissent lorsque le fichier tient sur plus de 12 blocs de données. Ici, il vaut mieux en savoir un peu sur la manière dont sont structurés les systèmes de fichiers Unix. Les données du fichier sont stockées dans des unités appelées « blocs ». Ces blocs peuvent être numérotés séquentiellement. Un fichier a également un « inode », où sont placées des informations telles que propriétaire, permissions ou type. Comme les blocs, les inodes sont numérotés séquentiellement, bien que la séquence soit différente. Une entrée de répertoire consiste en un nom de fichier associé à un numéro d'inode.

Mais, si on en restait là, le noyau ne saurait toujours pas trouver les données correspondant à une entrée de répertoire. Ainsi l'inode indique également l'endroit où se trouvent les blocs de données du fichier, comme suit :

- Les numéros de blocs des 12 premiers blocs sont indiqués directement dans l'inode (on les appelle parfois *blocs directs*) ;
- L'inode contient le numéro de bloc d'un *bloc indirect*. Un bloc indirect contient les numéros de bloc de 256 blocs de données additionnels ;
- L'inode contient le numéro de bloc d'un *bloc doublement indirect*. Un bloc doublement indirect contient les numéros de bloc de blocs indirects supplémentaires ;
- L'inode contient le numéro de bloc d'un bloc *triplement indirect*. Un bloc triplement indirect contient les numéros de bloc de 256 blocs doublement indirects supplémentaires.

Relisez bien tout ça : je sais que c'est compliqué, mais c'est important, aussi.

Maintenant, l'implantation du noyau pour toutes les versions actuelles (2.0.36 inclue) efface malheureusement tous les blocs indirects (et doublement indirects, etc.) lors de la suppression d'un fichier. Alors, si votre fichier occupait plus de 12 blocs, vous n'êtes pas garanti de pouvoir retrouver les numéros de tous les blocs dont vous avez besoin (sans parler de leur contenu).

La seule méthode que j'aie pu trouver jusqu'ici consiste à supposer que le fichier n'est pas fragmenté : s'il l'est, vous aurez des ennuis. En supposant que le fichier n'est pas fragmenté, il y a plusieurs dispositions de blocs de données, selon le nombre de blocs de données utilisés par le fichier :

0 à 12

les numéros de bloc sont indiqués dans l'inode, comme décrit précédemment ;

13 à 268

après les blocs directs, comptez un pour le bloc indirect, puis vous avez 256 blocs de données ;

269 à 65804

comme avant, il y a 12 blocs directs, un bloc indirect (inutile), et 256 blocs. Ils sont suivis d'un bloc doublement indirect (inutile), et 256 répétitions de : un bloc indirect (inutile) et 256 blocs de données ;

65805 ou plus

la disposition des 65804 premiers blocs est identique à ce qui est décrit di-dessus. Suivent un bloc triplement indirect (inutile) et 256 répétitions d'une séquence « doublement indirect ». Chaque séquence doublement indirecte consiste en un bloc doublement indirect (inutile), suivi de 256 répétitions de : un bloc indirect (inutile) et 256 blocs de données.

Bien entendu, même si ces blocs sont supposés corrects, rien ne garantit que les données qu'ils contiennent sont intactes. De plus, plus le fichier est long, moins vous avez de chances qu'il ait pu être écrit dans le système de fichiers sans fragmentation raisonnable (sauf dans certaines circonstances particulières).

Notez que j'ai supposé depuis le début que vos blocs occupaient la taille de 1024 octets, c'est-à-dire la valeur standard. Si vos blocs sont plus grands, une partie des nombres écrits plus haut doivent être changés. Typiquement, puisque chaque numéro de bloc occupe 4 octets, le nombre de numéros de bloc pouvant être placés dans chaque bloc indirect est taille_du_bloc/4. Donc, chaque fois que le nombre 256 apparaît dans la dicussion qui précède, remplacez-le par taille_du_bloc/4. Les limitations « nombre de blocs requis » devront également être modifiées.

Examinons un exemple de récupération de fichier plus long.

```
debugfs:  stat <1387>
Inode: 148004   Type: regular    Mode:  0644   Flags: 0x0   Version: 1
User:   503   Group:   100   Size: 1851347
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 3616
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x31a9a574 -- Mon May 27 13:52:04 1996
atime: 0x31a21dd1 -- Tue May 21 20:47:29 1996
mtime: 0x313bf4d7 -- Tue Mar  5 08:01:27 1996
dtime: 0x31a9a574 -- Mon May 27 13:52:04 1996
BLOCKS:
8314 8315 8316 8317 8318 8319 8320 8321 8322 8323 8324 8325 8326 8583
TOTAL: 14
```

Il semble y avoir de bonnes chances pour que ce fichier ne soit pas fragmenté : de façon évidente, les 12 premiers blocs listés dans l'inode (qui sont tous des blocs de données) sont contigus. Nous pouvons donc commencer par récupérer ces blocs :

```
# fsgrab -c 12 -s 8314 /dev/hda5 > /mnt/recovered.001
```

Maintenant, le bloc suivant listé dans l'inode, 8326, est un bloc indirect, que nous pouvons ignorer. Mais nous nous fions à notre intuition qu'il sera suivi de 256 blocs de données (du numéro 8327 au numéro 8582).

```
# fsgrab -c 256 -s 8327 /dev/hda5 >> /mnt/recovered.001
```

Le dernier bloc listé dans l'inode est le 8583. Notez que ça ressemble toujours bien à un fichier contigu : le numéro du dernier bloc que nous ayons écrit était le 8582, donc 8327 + 255. Ce bloc 8583 est un bloc doublement indirect, que nous pouvons ignorer. Il est suivi par jusqu'à 256 répétitions d'un bloc indirect (ignoré) suivi de 256 blocs de données. Après un petit calcul mental, on en déduit les commandes suivantes. Remarquez qu'on saute le bloc doublement indirect 8583 et le bloc indirect 8584, qui suivent immédiatement (espérons-le) et qu'on commence directement à lire les données depuis le bloc 8585.

```
# fsgrab -c 256 -s 8585 /dev/hda5 >> /mnt/recovered.001
# fsgrab -c 256 -s 8842 /dev/hda5 >> /mnt/recovered.001
# fsgrab -c 256 -s 9099 /dev/hda5 >> /mnt/recovered.001
# fsgrab -c 256 -s 9356 /dev/hda5 >> /mnt/recovered.001
# fsgrab -c 256 -s 9613 /dev/hda5 >> /mnt/recovered.001
# fsgrab -c 256 -s 9870 /dev/hda5 >> /mnt/recovered.001
```

En rassemblant tout, on voit qu'on a écrit depuis le début 12 + (7 * 256) blocs, c'est-à-dire 1804. La commande « stat » nous a indiqué pour l'inode un « *blockcount* » de 3616 ; mais ces blocs occupaient malheureusement 512 octets (un reliquat d'Unix), ce que nous voulons réellement est alors 3616/2 = 1808 blocs de 1024 octets. Cela signifie que nous avons seulement besoin de quatre blocs de plus. Le dernier bloc de données écrit portait le numéro 10125. De la même façon que depuis le début, on saute un bloc indirect (numéro 10126) ; on peut alors écrire ces quatre derniers blocs.

```
# fsgrab -c 4 -s 10127 /dev/hda5 >> /mnt/recovered.001
```

Et maintenant, avec un peu de chance, le fichier complet a été récupéré avec succès.

11 Modifier les inodes directement

Cette méthode est apparemment beaucoup plus facile. Cependant, comme souligné plus haut, elle ne peut pas venir à bout de fichiers occupant plus de 12 blocs.

Pour chaque inode que vous voulez récupérer, vous devez mettre à 1 le nombre de liens, et à 0 la date de suppression. Cela peut être fait grâce à la commande `mi` (modifier inode) de `debugfs`. Voici un exemple de sortie concernant la modification de l'inode 148003 :

```
debugfs:  mi <148003>
                          Mode    [0100644]
                       User ID    [503]
                      Group ID    [100]
                          Size    [6065]
                 Creation time    [833201524]
             Modification time    [832708049]
                   Access time    [826012887]
                 Deletion time    [833201524] 0
                    Link count    [0] 1
                   Block count    [12]
                    File flags    [0x0]
                     Reserved1    [0]
                      File acl    [0]
                 Directory acl    [0]
              Fragment address    [0]
               Fragment number    [0]
                 Fragment size    [0]
               Direct Block #0    [594810]
               Direct Block #1    [594811]
               Direct Block #2    [594814]
               Direct Block #3    [594815]
               Direct Block #4    [594816]
               Direct Block #5    [594817]
               Direct Block #6    [0]
               Direct Block #7    [0]
               Direct Block #8    [0]
               Direct Block #9    [0]
              Direct Block #10    [0]
              Direct Block #11    [0]
                Indirect Block    [0]
         Double Indirect Block    [0]
         Triple Indirect Block    [0]
```

C'est-à-dire que je mets à 0 la date de suppression et le nombre de liens à 1, puis j'envoie juste un retour chariot pour chacun des autres champs. D'accord, ce n'est pas très souple si vous avez beaucoup de fichiers à récupérer, mais je pense que vous pourrez faire face. Si vous vouliez du velours, il fallait utiliser un « système d'exploitation » graphique avec une jolie « corbeille ».

À propos, le texte de sortie de `mi` indique un champ « création » (*creation time*). Il est totalement mensonger (ou en tout cas trompeur) ! En fait, sur un système de fichiers Unix, vous ne pouvez pas déterminer quand un fichier a été créé. Le champ `st_ctime` d'une `struct stat` fait référence à la date de modification de l'inode (*inode change time*), c'est-à-dire la dernière fois qu'un quelconque des détails de l'inode a été changé. Si finit la lessons d'huy.

Notez que les versions plus récentes de `debugfs` que celle que j'utilise n'incluent probablement pas certains des champs de la liste donnée plus haut (typiquement `Reserved1` et des champs sur les fragments).

Une fois que vous aurez modifié les inodes, vous pourrez quitter `debugfs` et taper :

```
# e2fsck -f /dev/hda5
```

L'idée est que chacun des fichiers supprimés a été littéralement « dé-supprimé », mais qu'aucun d'entre eux n'apparaît en entrée de répertoire. Le programme `e2fsck` peut le détecter, et ajoutera une entrée dans le répertoire `/lost+found` du système de fichiers (Donc, si la partition est normalement montée dans `/usr`, les fichiers vont apparaître dans `/usr/lost+found`). Tout ce qui reste à faire est de redonner son nom à chaque fichier d'après son contenu, et le remettre à sa place dans l'arborescence du système de fichiers.

Quand vous lancerez `e2fsck`, vous obtiendrez des messages d'information, ainsi que des questions à propos des problèmes à réparer. Répondez oui (*yes*) partout où vous voyez *'summary information'* ou à chaque référence aux inodes que vous avez modifiés. Tout le reste vous regarde, bien qu'il soit en général une bonne idée de répondre oui à toutes les questions. Lorsque `e2fsck` a terminé, vous pouvez remonter le système de fichiers.

En fait, il y a un autre moyen que de demander à `e2fsck` de laisser les fichiers dans `/lost+found` : vous pouvez utiliser `debugfs` pour créer un lien vers l'inode dans le système de fichiers. Utilisez la commande `link` de `debugfs` quand vous avez fini de modifier l'inode.

```
debugfs:  link <148003> toto.txt
```

Ceci crée un fichier appelé `toto.txt` dans ce que `debugfs` suppose être le répertoire courant ; `toto.txt` sera votre fichier. Vous aurez quand même besoin de lancer `e2fsck` pour corriger le *'summary information'*, le nombre de blocs, etc.

12 Cela va-t-il se simplifier dans l'avenir ?

Oui. En fait, je pense que c'est déjà le cas. Bien qu'au moment où ces lignes sont écrites (2 février 1999), les noyaux stables actuels (la série 2.0.x) effacent les blocs indirects, ce n'est plus le cas des noyaux de développement 2.1.x, ni des noyaux stables 2.2.x, dont le 2.2.1 qui vient d'être diffusé ; nous allons voir apparaître des distributions à base de noyaux 2.2.x d'ici un ou deux mois.

Une fois cette limitation retirée des noyaux stables, bon nombre de mes objections au fait de modifier les inodes à la main disparaîtront. Il sera également possible d'utiliser la commande `dump` de `debugfs` sur des fichiers longs, et d'utiliser d'autres outils de récupération.

13 Existe-t-il des outils qui automatisent le processus ?

En fait, il y en a. Hélas, je crains qu'ils souffrent du même problème que la technique de modification manuelle des inodes : les blocs indirects sont irrécupérables. Cependant, selon la probabilité que cela ne soit plus un problème d'ici peu, ça vaut sûrement le coup de chercher ces programmes maintenant.

J'ai écrit un utilitaire nommé `e2recover`, qui est essentiellement un enrobage Perl à `fsgrab`. Il fait un effort raisonnable pour gérer les blocs indirects effacés, et semble très bien fonctionner en l'absence de fragmentation. Il en profite pour remettre les permissions (et, quand c'est possible, le propriétaire) des fichiers récupérés, et s'assure même que les fichiers récupérés soient à la bonne taille.

J'ai initialement écrit `e2recover` pour la toute proche mise à jour de ce Howto ; malheureusement cela signifie que tous les renseignements utiles sur `e2recover` sont aussi prévus pour cette mise à jour. En attendant, il devrait s'avérer quand même utile dès maintenant ; vous pouvez le télécharger depuis *ma page* `http://pobox.com/~aaronc/tech/e2-undel/` , et prochainement sur Metalab.

Scott D. Heavner est l'auteur de `lde`, ('Linux Disk Editor'). `lde` peut servir aussi bien d'éditeur binaire de disque, que d'un équivalent de `debugfs` pour les systèmes ext2 et minix, et même pour les systèmes xia (bien que le support xia ne soit plus disponible dans les noyaux 2.1.x et 2.2.x). Il dispose de fonctionnalités pour faciliter la récupération, comme le parcours de la liste des blocs, et la recherche dans le contenu du disque. Il possède également une documentation sur les concepts de base des systèmes de fichiers particulièrement utile, ainsi qu'un document expliquant comment l'utiliser afin de récupérer des fichiers supprimés. La version 2.4 de `lde` est disponible sur

Metalab `http://metalab.unc.edu/pub/Linux/system/filesystems/lde-2.4.tar.gz` et ses mirroirs, et sur

la page de son auteur `http://www.geocities.com/CapeCanaveral/Lab/7731/lde.html` .

Une autre possibilité est fournie par le GNU Midnight Commander, `mc`. C'est un gestionnaire de fichiers en plein écran, inspiré autant que je le sache d'un certain programme MS-DOS couramment désigné sous le nom de « nc ». `mc` supporte la souris dans la console Linux et dans un xterm, et fournit des systèmes de fichiers virtuels qui permettent des trucs du genre de se déplacer dans une archive Tar. Parmi ses systèmes de fichiers virtuels, il en est un concernant la récupération sous Ext2. Tout ça semble très commode à manipuler, mais je dois avouer que que je ne l'ai jamais utilisé moi-même – je préfère les bonnes vieilles commandes *shell*. Apparemment il faut configurer le programme avec l'option `-with-ext2undel` ; vous aurez également besoin des bibliothèques de développement et des fichiers d'en-tête (*include*) qui viennent avec le paquetage `e2fsprogs`. La version fournie par *Debian GNU/Linux* `http://www.debian.org/` est ainsi compilé ; c'est peut-être le cas pour d'autres distributions. Une fois que le programme est compilé, vous pouvez y taper `cd undel:/dev/hda5/`, et obtenir, sous forme de contenu de répertoire, le catalogue des fichiers supprimés. Comme la plupart des outils actuels de récupération, il gère très mal les blocs indirects effacés – la plupart du temps il ne récupère que les 12 premiers Ko des gros fichiers.

La dernière version peut être récupérée depuis

le site ftp officiel `ftp://ftp.nuclecu.unam.mx/Midnight/devel` .

14 Achevé d'imprimer...

J'ai l'intention de produire des mises à jour régulières de ce document, tant que j'aurai à la fois suffisamment de temps pour le faire et quelque chose d'intéressant à dire. Ceci signifie que je suis avide de commentaires de la part de mes lecteurs. Ma rédaction peut-elle être plus claire ? Pouvez-vous penser à quelque chose qui pourrait rendre l'affaire plus simple ? Existe-t-il un nouvel outil qui puisse faire tout cela automatiquement ?

Quoi qu'il en soit : si vous avez quoi que ce soit à dire, à propos de ce document ou des outils `fsgrab` et `e2recover`, envoyez-moi un mot à :

aaronc@pobox.com .

15 Remerciements et bibliographie

> *Si j'ai vu plus loin que d'autres, c'est parce que j'étais hissé sur des épaules de géants* (Isaac Newton)

Une grande partie de ce mini-Howto est dérivée d'un article posté sur le groupe de *news* *comp.os.linux.misc* par Robin Glover *swrglovr@met.rdg.ac.uk* .

Je voudrais remercier Robin de m'avoir gracieusement autorisé à reprendre ses idées dans ce mini-Howto.

Je voudrais également profiter de l'occasion pour remercier une fois de plus toutes les personnes qui m'ont contacté à propos de ce Howto. Ce sont les remerciements chaleureux que l'on reçoit qui justifient la peine que l'on se donne.

Quelques références bibliographiques :

- **Frisch**, Æleen (1995), *Essential System Administration*, second edition, O'Reilly and Associates, Inc., ISBN : 1-56592-127-5.
- **Garfinkel**, Simson, Daniel **Weise** et Steven **Strassmann** (1994), *The Unix-Haters Handbook*, IDG Books, ISBN : 1-56884-203-1. Ce livre est composé pour la plus grande partie de pleurs d'adolescents qui pensent que *leur* système d'exploitation est tellement mieux qu'Unix, et le reste ne s'applique pas si vous avez de bons programmes en espace utilisateur tels que les outils GNU. Mais il y a quelques épis de blé parmi la paille ; par exemple, la discussion autour de la facilité d'effacement de fichier sous Unix mérite qu'on s'y arrête.
- **Glover**, Robin (31 Jan 1996), *HOW-TO : undelete linux files (ext2fs/debugfs)*, comp.os.linux.misc Usenet posting.
- **Peek**, Jerry, Tim **O'Reilly**, Mike **Loukides** *et al* (1993), *UNIX Power Tools*, O'Reilly and Associates, Inc./Random House, Inc., ISBN : 0-679-79073-X.

16 Bla-bla juridique