

ВОССТАНОВЛЕНИЕ УДАЛЕННЫХ ФАЙЛОВ ПОД LINUX

Каждый из нас хотя бы однажды удалял ценный файл, а то и весь корневой каталог целиком! Резервной копии нет. Времени на поиски утилит для восстановления — тоже. Как быть?

К счастью, все необходимое уже включено в ваш любимый дистрибутив и всегда находится под рукой. Если говорить кратко: `debugfs`, `lsdel`, `stat`, `cat`, `dump`, `undel`. Если чуть подробнее — читайте эту статью, рассказывающую о восстановлении данных на `ext2fs` и отчасти `ext3fs`-разделах.

КРИС КАСПЕРСКИ

Информации по восстановлению данных под Linux практически нет. Как будто у пользователей этой ОС данные не исчезают. Исчезают, еще как! Ошибочное удаление файлов — достаточно распространенное явление, наверное, даже более частое, чем в мире Microsoft. Под Windows большинство файловых операций осуществляется вручную с помощью «Проводника» или других интерактивных средств типа FAR. Интерактивные среды есть и в Linux (KDE, GNOME, Midnight Commander), но есть там и поклонники командной строки, а командная строка — это регулярные выражения и скрипты, то есть автоматизированные средства управления — мощные, удобные, и... разрушительные. Малейшая небрежность их проектирования и... прощай, мои файлы! Помнится, год-два назад по сети распространялся «Албанский вирус». То тут, то там на форумах появлялись сообщения с просьбой объяснить, что делает очень запутанный код на Perl. Задетые за живое гуру, поленившись вникнуть в его суть, просто запускали непонятную программу на выполнение... и напрасно! Посредством нетривиальных подстановок строковая конструкция разворачивалась в «`rm -rf /`»! Восстановить весь корневой каталог под `ext2fs`, а тем более `ext3fs` очень и очень сложно. Можно даже сказать, практически не-

реально. Однако если пользователь был зарегистрирован в системе не как root, жертвами неосторожного эксперимента оказывались «всего лишь» файлы из его домашнего каталога. Поэтому в данной статье речь пойдет только о восстановлении отдельных, наиболее ценных файлов.

Перефразируя Булгакова, можно сказать: мало того, что файл смертен, так он еще и внезапно смертен! Беда никогда не предупреждает о своем приходе, и администратору приходится быть постоянно начеку. Несколько секунд назад все было хорошо: цвела весна, винчестер оживленно стрекотал всеми своими головками, администратор отхлебывал кофе из черной кружки с надписью root, раздумывая: то ли поиграть в DOOM, то ли поболтать с секретаршей, как вдруг... бабах! Сотни гигабайт ценнейших данных разлетелись на мелкие осколки. Все силы брошены на разгребания завалов и спасения всех, кого еще можно спасти.

Инструментарий

Программ, пригодных для восстановления данных, под Linux совсем немного, намного меньше, чем под Windows NT, да и тем до совершенства как до Луны. Но ведь не разрабатывать же весь необходимый инструментарий самостоятель-

но?! Будем использовать то, что дают, утешая себя мыслью, что нашим предкам, работающим на динозаврах машинной эры, приходилось намного сложнее.

Редактор диска

Под Linux диски чаще всего редактируются при помощи lde (расшифровывается как Linux Disk Editor). Это, конечно, не Norton Disk Editor, но и не Microsoft Disk Probe. Профессионально-ориентированный инструмент консольного типа с разумным набором функциональных возможностей. Понимает ext2fs, minix, xiafs и отчасти FAT (в перспективе обещана поддержка NTFS, которая на Linux никому не нужна, а вот отсутствие в этом списке UFS и FFS очень огорчает).

Поддерживает: отображение/редактирование содержимого в HEX-формате, просмотр суперблока (super-block), файловых записей (inode) и директорий в удобочитаемом виде; контекстный поиск, поиск файловых записей, ссылающихся на данный блок (полезная штука, только, к сожалению, реализованная кое-как и срабатывающая не всегда), режим восстановления с ручным редактированием списка прямых/косвенных блоков, сброс дампа на диск и некоторые другие второстепенные операции.

Может работать как в пакетном, так и в интерактивном режимах. В пакетном режиме все управление осуществляется посредством командной строки, что позволяет полностью или частично автоматизировать некоторые рутинные операции.

Распространяется в исходных текстах по лицензии GPL (<http://lde.sourceforge.net>), денег не требует, работает практически под любой UNIX-совместимой операционной системой (включая FreeBSD) и входит во все «правильные» дистрибутивы (например, в Knoppix).

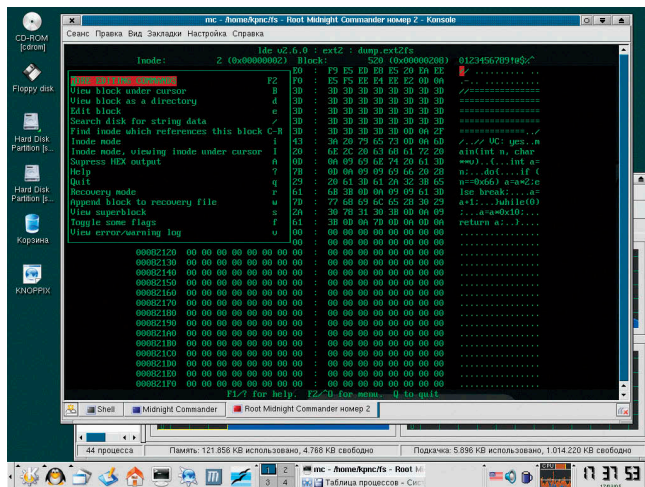


Рисунок 1. Дисковый редактор LDE (Linux Disk Editor)

Работа с lde на первых порах производит довольно странное впечатление: чувствуешь себя неандертальцем, пересевшим с IBM PC на УНК/ZX Spectrum и добывающим огонь трением. Впрочем, со временем это проходит (программист, как известно, существо неприхотливое и ко всему привыкающее). Как вариант можно загрузиться со «спасательной дискеты» и использовать знакомый Norton Disk Editor или Runtime NT Explorer, запущенный из-под Windows PE (версия Windows NT, запускающаяся с CD-ROM без предварительной установки на жесткий диск). С одной стороны, это

удобно (привычный интерфейс и все такое), с другой – ни Disk Editor, ни NT Explorer не поддерживают ext2fs/ext3fs, и все структуры данных приходится декодировать вручную.

hex-редакторы

UNIX – это вам не Windows! Без дисковых редакторов здесь в принципе можно и обойтись. Берем любой hex-редактор, открываем соответствующее дисковое устройство (например, /dev/sdb1) и редактируем его в свое удовольствие. Красота! Старожилы, должно быть, помнят, как во времена первой молодости MS-DOS, когда ни HIEW, ни QVIEW еще не существовало, правка исполняемых файлов на предмет «отлома» ненужного 7xh обычно осуществлялась DiskEdit, т.е. дисковый редактор использовался как hex. А в UNIX, наоборот, hex-редакторы используются для редактирования диска.

Какой редактор выбрать? В общем-то, это дело вкуса (причем не только вашего, но еще и составителя дистрибутива). Одни предпочитают консольный hexedit, другие тяготеют к графическому khedit, а третьи выбирают BIEW (урезанная калька со всем известного HIEW).

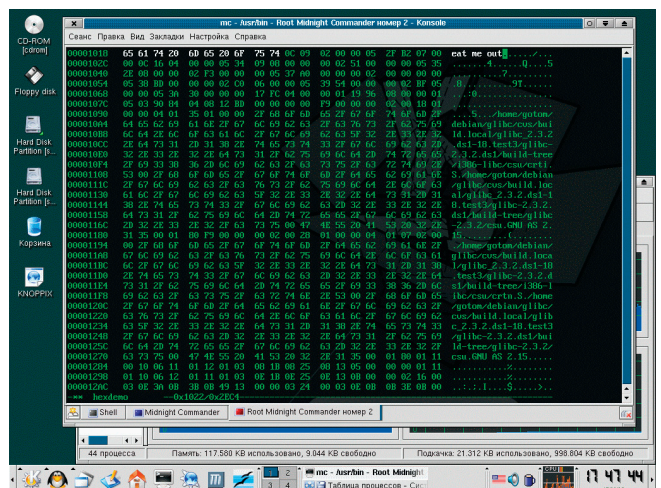


Рисунок 2. Внешний вид редактора hexedit

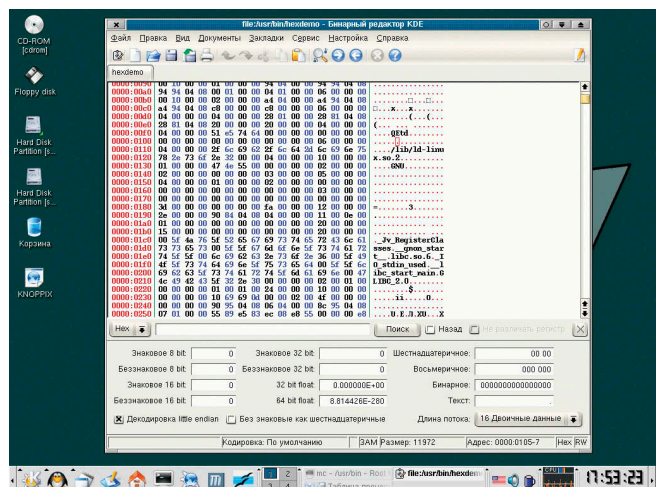


Рисунок 3. Внешний вид редактора khedit

Отладчики файловой системы

Отладчиками файловой системы называют утилиты, дающие доступ к «святым святым» файловой системы и позво-

ляющие манипулировать ключевыми структурами данных по своему усмотрению.

Чем они отличаются от простых редакторов? Редактор работает на более низком уровне – уровне блоков или секторов. Он в принципе может представлять некоторые структуры в наглядном виде, однако в их «физический» смысл никак не вникает.

Отладчик файловой системы работает через драйвер, и потому испортить раздел с его помощью намного сложнее. Он реализует довольно высокоуровневые операции, такие как установка/снятие флага занятости блока, создание новой символической ссылки и т. д. А вот «посекторного» hex-редактора отладчика файловой системы обычно не содержат, поэтому обе категории программ взаимно дополняют друг друга.

Большинство дистрибутивов Linux (если не все из них) включают в себя отладчик debugfs, поддерживающий ext2fs и отчасти ext3fs.

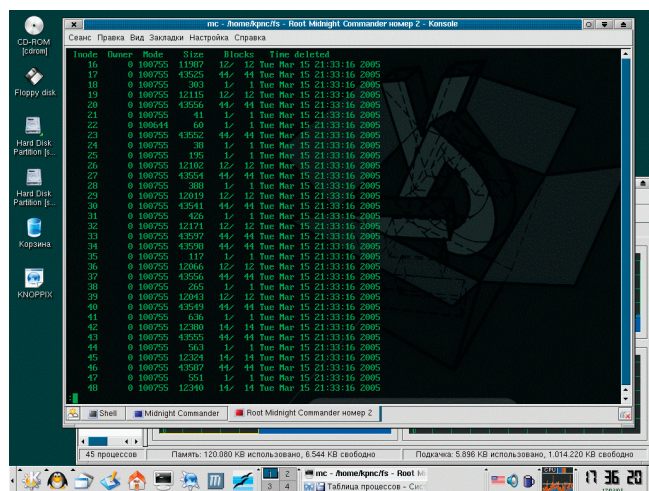


Рисунок 4. Debugfs за работой

Дисковые доктора

В мире UNIX проверка целостности файловой системы обычно осуществляется программой fsck (аналог chkdsk под Windows NT), представляющей собой консольную утилиту, практически лишенную пользовательского интерфейса и входящую в штатный комплект поставки любого дистрибутива. Как и любое другое полностью автоматизированное средство, она не только лечит, но случается, что и калечит, так что пользоваться ей следует с очень большой осторожностью.

LiveCD

За последние несколько лет появилось множество дистрибутивов Linux, загружающихся прямо с CD-ROM и не требующих установки на винчестер. Очень удобная штука для восстановления данных. Однако далеко не все дистрибутивы для этого пригодны.

Knoppix 3.7 – самый лучший дистрибутив из всех, что мне доводилось видеть. Основан на Debian GNU/Linux. За-

нимает всего один диск, но содержит практически все: от дисковых утилит и компиляторов до офисных пакетов и мультимедийных приложений¹. Очень шустро работает, требует от 128 Мб оперативной памяти (если меньше – будет вести свопинг на диск). В 2004 году издательство O'Reilly выпустило шикарную книгу «KNOPPIX Hacks», содержащую главы, посвященные технике восстановления данных.

Frenzy 0.3 – дистрибутив, основанный на FreeBSD с небольшим количеством дисковых утилит, ориентированных на ext2fs, в то время как основной файловой системой самой BSD является UFS/FFS и ext2fs она поддерживает постольку-поскольку². Тем не менее для восстановительных работ данный диск вполне пригоден, и всем поклонникам BSD его можно смело рекомендовать.

SuSE 9.2 – по классификации, предложенной Винни-Пухом, это неправильный дистрибутив. Занимает два диска (один с KDE, другой с GNOME). Требуется не менее 256 Мб оперативной памяти (правда, KDE-версия запускается и при 220 Мб). Очень медленно работает и не содержит ничего, кроме щепотки офисных программ.

Подготовка к восстановлению

Прежде чем приступать к восстановлению, обязательно размонтируйте дисковый раздел или на худой конец перемонтируйте его в режим «только на чтение». Лечение активных разделов зачастую только увеличивает масштабы разрушения. Если восстанавливаемые файлы находятся на основном системном разделе, у нас два пути – загрузиться с LiveCD или подключить восстанавливаемый жесткий диск на Linux-машину вторым.

Чтобы чего-нибудь не испортить, никогда не редактируйте диск напрямую. Работайте с его копией, которую можно создать командой:

```
cp /dev/sdb1 my_dump
```

где sdb1 – имя устройства, а my_dump – имя файла-дампа или использовать dd (некоторым так привычнее). Файл-дамп можно разместить на любом свободном разделе или перегнать на другую машину по сети. Все дисковые утилиты (lde, debugfs, fschk) не заметят подвоха и будут работать с ним, как с «настоящим» разделом.

При необходимости его даже можно смонтировать на файловую систему:

```
mount my_dump mount_point -o loop
```

чтобы убедиться, что восстановление прошло успешно.

Команда:

```
cp my_dump /dev/sdb1
```

копирует восстановленный файл-дамп обратно в раздел, хотя делать это совсем необязательно. Проще (и безопаснее) копировать только восстанавливаемые файлы.

¹ Обзор дистрибутива Knoppix смотрите на стр. 4-6. (Прим. ред.)

² Обзор дистрибутива Frenzy – в статьях Александра Байрака «Безумный чертёнок», №1, 2004 г. и «Frenzy: FreeBSD в кармане сисадмина» Сергея Можайского в №2, 2004 г.

Восстановление удаленных файлов на ext2fs

Ext2fs все еще остается базовой файловой системой для многих Linux, поэтому рассмотрим ее первой. Концепции, которые она исповедует, во многом схожи с NTFS, так что культурного шока при переходе с NTFS на ext2fs с нами не случится.

Подробное описание структуры хранения данных ищите в документе «Design and Implementation of the Second Extended Filesystem», а также книге Э. Таненбаума «Operating Systems: Design and Implementation» и статье В. Мешкова «Архитектура файловой системы ext2», опубликованной в журнале «Системный администратор», №11, 2003 г. Исходные тексты дисковых утилит (драйвер файловой системы, lde, debugfs) также не помешают.

Структура файловой системы

В начале диска расположен boot-сектор (на незагруженных разделах он может быть пустым). За ним по смещению 1024 байта от начала первого сектора лежит суперблок (super-block), содержащий ключевую информацию о структуре файловой системы. (В FAT и NTFS эта информация хранится непосредственно в boot).

В первую очередь нас будет интересовать 32-разрядное поле `s_log_block_size`, расположенное по смещению 18h байт от начала супер-блока. Здесь хранится размер одного блока (block), или в терминологии MS-DOS/Windows кластера, выраженный в виде показателя позиции, на которую нужно сдвинуть число 200h. В естественных единицах это будет звучать так: `block_size = 200h << s_log_block_size` (байт). То есть если `s_log_block_size` равен нулю, размер одного блока составляет 400h байт или два стандартных сектора.

Листинг 1. Структура дискового тома, размеченного под ext2fs

Смещение	Размер	Описание
0	1	boot record ; загрузочный сектор
-- block group 0 -- ; группа блоков 0		
(1024 bytes)	1	superblock ; суперблок
2	1	group descriptors ; дескриптор группы
3	1	block bitmap ; карта свободных блоков
4	1	inode bitmap ; карта свободных inode
5	214	inode table ; массив inode (сведения о файлах)
219	7974	data blocks ; блоки данных (файлы, директории)
-- block group 1 -- ; группа блоков 1		
8193	1	superblock backup ; копия суперблока
8194	1	group descriptors backup ; копия дескриптора группы
8195	1	block bitmap ; карта свободных блоков
8196	1	inode bitmap ; карта свободных inode
8197	214	inode table ; массив inode (сведения о файлах)
8408	7974	data blocks ; блоки данных (файлы, директории)
-- block group 2 -- ; группа блоков 2		
16385	1	block bitmap ; карта свободных блоков
16386	1	inode bitmap ; карта свободных inode
16387	214	inode table ; массив inode (сведения о файлах)
16601	3879	data blocks ; блоки данных (файлы, директории)

Вслед за суперблоком идут дескрипторы групп (group descriptors) и карты свободного пространства, в просторечии – битмапы (block bitmap/inode bitmap), которые нас мало интересуют, а вот inode-таблицу, расположенную за ними, мы рассмотрим поподробнее.

В ext2fs (как и многих других файловых системах из мира UNIX) inode играет ту же самую роль, что и FILE Record в NTFS. Здесь сосредоточена вся информация о файле: тип файла (обычный файл, директория, символическая ссылка и т. д.), логический и физический размер, схема размещения на диске, время создания, модификации, последнего доступа и удаления, права доступа и количество ссылок на файл.

Листинг 2. Формат представления inode

Смещение	Размер	Описание
0	2	i_mode ; формат представления описание
2	2	i_uid ; uid пользователя
4	4	i_size ; размер файла в байтах
8	4	i_atime ; время последнего доступа к файлу
12	4	i_ctime ; время создания файла
16	4	i_mtime ; время модификации файла
20	4	i_dtime ; время удаления файла
24	2	i_gid ; gid группы
26	2	i_links_count ; количество ссылок на файл (0 – файл удален)
28	4	i_blocks ; количество блоков, принадлежащих файлу
32	4	i_flags ; разные флаги
36	4	i_osdl ; OS dependant value
40	12 x 4	i_block ; 12 DIRECT BLOCKS (ссылки на первые 12 блоков файла)
88	4	i_iblock ; 1x INDIRECT BLOCK
92	4	i_2iblock ; 2x INDIRECT BLOCK
96	4	i_3iblock ; 3x INDIRECT BLOCK
100	4	i_generation ; поколение файла (используется NFS)
104	4	i_file_acl ; внешние атрибуты
108	4	i_dir_acl ; higher size
112	4	i_faddr ; положение последнего фрагмента
116	12	i_osd2 ; OS dependant structure

Первые 12 блоков, занимаемых файлом, хранятся непосредственно в самом inode в массиве DIRECT BLOCKS (непосредственные блоки для наглядности выделены полужирным шрифтом). Каждый элемент массива представляет собой 32-битный номер блока. При среднем значении `BLOCK_SIZE` в 4 Кб DIRECT BLOCK могут адресовать до $4 * 12 = 48$ Кб данных. Если файл превышает этот размер, создаются один или несколько блоков косвенной адресации (INDIRECT BLOCK). Первый блок косвенной адресации (1x INDIRECT BLOCK или просто INDIRECT BLOCK) хранит ссылки на другие непосредственные блоки. Адрес этого блока хранится в поле `i_indirect_block` в inode. Как легко посчитать, он адресует порядка $BLOCK_SIZE/sizeof(DWORD) * BLOCK_SIZE = 4096/4 * 4$ Мб данных. Если этого вдруг окажется недостаточно, создается дважды косвенный блок (2x INDIRECT BLOCK или DOUBLE INDIRECT BLOCK), хранящий указатели на косвенные блоки, что позволяет адресовать $(BLOCK_SIZE/sizeof(DWORD))^{**2} * BLOCK_SIZE = 4096/4^{**} 4096 = 4$ Гб данных. Если же этого все равно недостаточно, создается трижды косвенный блок (3x INDIRECT BLOCK или TRIPLE INDIRECT BLOCK), содержащий ссылки на дважды косвенные блоки (на данном рисунке трижды косвенный блок не показан).

Отметим, что по сравнению с NTFS такая схема хранения информации о размещении гораздо проще устроена, но вместе с тем и прожорлива. Однако она обладает одним несомненным достоинством, которое оставляет NTFS далеко позади. Поскольку все ссылки хранятся в неупакованном виде, для каждого блока файла мы можем быстро найти соответствующий ему косвенный блок, даже если inode полностью разрушен.

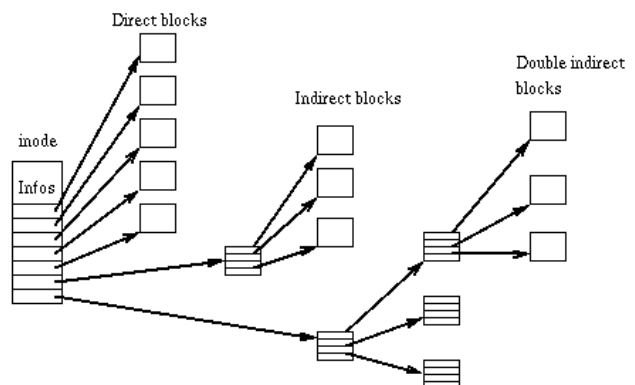


Рисунок 5. Описание порядка размещения файла на диске, иерархия непосредственных и косвенных блоков

Имя файла в inode не хранится. Ищите его внутри директорий, представляющих собой массив записей следующего вида:

Листинг 3. Формат представления массива директорий

Смещение	Размер	Описание
0	4	inode ; ссылка на inode
4	2	rec_len ; длина данной записи
6	1	name_len ; длина имени файла
7	1	file_type ; тип файла
8	...	name ; имя файла

При удалении файла операционная система находит соответствующую запись в директории. Обнуляет поле inode и увеличивает размер предшествующей записи (поле rec_len) на величину удаляемой. То есть предшествующая запись как бы «поглощает» удаленную. И хотя имя файла до поры до времени остается нетронутым, ссылка на соответствующий ему inode оказывается уничтоженной. Вот и попробуем разобраться, какому файлу какое имя принадлежит!

В самом inode при удалении файла тоже происходят большие изменения. Количество ссылок (i_links_count) сбрасывается в нуль и обновляется поле последнего удаления (i_dtime). Все блоки, принадлежащие файлу, в карте свободного пространства (block bitmap) помечаются как неиспользуемые, после чего данный inode также освобождается (в inode bitmap).

Техника восстановления удаленных файлов

В ext2fs полное восстановление даже только что удаленных файлов невозможно. В этом смысле она проигрывает как FAT, так и NTFS. Как минимум теряется имя файла. Точнее говоря, теряется связь имен файлов с их содержимым. При удалении небольшого количества хорошо известных файлов это еще терпимо (имена-то ведь сохранились!), но что делать, если вы удалили несколько служебных подкаталогов, в которых никогда ранее не заглядывали?

Достаточно часто inode назначаются в том же порядке, в котором создаются записи в таблице директорий, к тому же существует такая штука, как «расширения» (.c, .gz, .tmp и т. д.), так количество возможных комбинаций соответствий имен файлов и inode чаще всего бывает относительно небольшим. Тем не менее восстановить удаленный корневой каталог в автоматическом режиме никому не удастся, а вот NTFS с этим справляется без труда.

В общем, стратегия восстановления выглядит приблизительно так: сканируем таблицу inode и отбираем все записи, чье поле i_links_count равно нулю. Сортируем их по дате удаления, чтобы файлы, удаленные последними, оказались наверху списка. Как вариант можно просто наложить фильтр (мы ведь помним примерное время удаления, не правда ли?). Если соответствующие inode еще не затерты вновь создаваемыми файлами, извлекаем список прямых/косвенных блоков и записываем их в дампы, корректируя его размер с учетом «логического» размера файла, за которое отвечает поле i_size.

Восстановление при помощи lde

Открываем редактируемый раздел или его файловую копию: «lde my_dump» или «lde /dev/sdb1». Редактор автома-

тически определяет тип файловой системы (в данном случае ext2fs) и предлагает нажать «any key» для продолжения. Нажимаем! Редактор переключается в режим отображения суперблока и предлагает нажать <I> для перехода в режим inode и для перехода в блочный режим (block-mode). Жмем <I> и оказываемся в первом inode, описывающем корневой каталог. <Page Down> перемещает нас к следующему inode, а <Page Up> – к предыдущему. Пролистываем inode вниз, обращая внимание на поле LINKS. У удаленных файлов оно равно нулю, и тогда «DELETION TIME» содержит время последнего удаления (мы ведь помним его, правда?).

Обнаружив подходящий inode, перемещаем курсор к первому блоку в списке DIRECT BLOCKS (где он должен находиться по умолчанию) и нажимаем <F2>. В появившемся меню выбираем пункт «Block mode, viewing block under cursor» (или сразу нажимаем горячую клавишу <Shift-B>). Редактор перемещается на первый блок удаленного файла. Просматривая его содержимое в hex-режиме, пытаемся определить, он ли это или нет. Чтобы возвратиться к просмотру следующего inode, нажимаем <I>, чтобы восстановить файл – <Shift-R>, затем еще раз <R> и имя файла-дампа для восстановления.

Можно восстанавливать файлы и по их содержимому. Допустим, нам известно, что удаленный файл содержит строку «hello, world». Нажимаем <F> и затем <A> (Search all block). Этим мы заставляем редактор искать ссылки на все блоки, в том числе и удаленные. Как вариант можно запустить редактор с ключом «--all». Но так или иначе мы нажимаем , затем, когда редактор перейдет в block-mode, – </> и вводим ASCII-строку для поиска. Находим нужный блок. Прокручивая его вверх-вниз, убеждаемся, что он действительно принадлежит тому самому файлу. Если это так, нажимаем <Ctrl>+<R>, заставляя редактор просматривать все inode, содержащие ссылку на этот блок. Номер текущего найденного inode отображается внизу экрана. (Именно внизу! Вверху отображается номер последнего просмотренного inode в режиме inode). Переходим в режим inode по клавише <I>, нажимаем <#> и вводим номер inode, который хотим просмотреть. Если дата удаления более или менее соответствует действительности, нажимаем <Shift-R>/<R> для сброса файла на диск. Если нет – возвращаемся в block-mode и продолжаем поиск.

В сложных случаях, когда список прямых и/или косвенных блоков разрушен, восстанавливаемый файл приходится собирать буквально по кусочкам, основываясь на его внутреннем содержимом и частично – на стратегии выделения свободного пространства файловой системой. В этом нам поможет клавиша <W> – дописать текущий блок к файлу-дампу. Просто перебираем все свободные блоки один за другим (редактор помечает их строкой NOT USED) и, обнаружив подходящий, дописываем в файл. Конечно, сильно фрагментированный двоичный файл так не восстановить, но вот листинг программы можно вполне.

Вывод – ручное восстановление файлов с помощью lde крайне непроизводительно и трудоемко, зато наиболее «прозрачно» и надежно.

А вот восстанавливать оригинальные имена лучше всего при помощи debugfs.

Восстановление при помощи debugfs

Загружаем в отладчик редактируемый раздел или его копию, «debugfs my_dump» или «debugfs /dev/sdb1». Если мы планируем осуществлять запись на диск, необходимо указать ключ «-w» («debugfs -w my_dump» или «debugfs -w /dev/sdb1»). Чтобы просмотреть список удаленных файлов, даем команду «lsdel» (или «lsdel t_sec», где t_sec – количество секунд, прошедшее с момента удаления файла). Экран заполняется списком удаленных файлов. Естественно, без имен. Файлы, удаленные более чем t_sec секунд назад (если sec задан), в этот список не попадают.

Команда «cat <N>» выводит содержимое текстового файла на терминал, где <N> – номер inode, заключенный в угловые кавычки. При выводе двоичных файлов на экране творится черт знает что, и такие файлы должны сбрасываться в дампы командой «dump <N> new_file_name», где «new_file_name» – новое имя файла (с путем), под которым он будет записан в родную (native) файловую систему, т.е. ту файловую систему, на которой был запущен debugfs. Файловая система восстанавливаемого раздела при этом остается неприкосновенной. Другими словами, наличие ключа «-w» для этого не требуется.

При желании можно «реанимировать» файл непосредственно на самой восстанавливаемой файловой системе (что особенно удобно при восстановлении больших файлов). В этом нам поможет команда «undel <N> undel_file_name», где undel_file_name – имя, которое будет присвоено файлу после восстановления. Внимание! Когда будете это делать, помните, что команда undel крайне агрессивна и деструктивна по своей природе. Она затирает первую свободную запись в таблице директорий, делая восстановление оригинальных имен невозможным!

Команда «stat <N>» отображает содержимое inode в удобочитаемом виде, а команда «mi <N>» позволяет редактировать их по своему усмотрению. Для ручного восстановления файла (которого не пожелаешь и врагу) мы должны установить счетчик ссылок (link count) в единицу, а время удаления (deletion time), наоборот, сбросить в нуль. Затем отдать команду «seti <N>», помечающую данный inode как используемый, и для каждого из блоков файла выполнить «setb X», где X – номер блока (перечень блоков, занимаемых файлом, можно подсмотреть командой stat, причем в отличие от ldd она отображает не только непосредственные, но и косвенные блоки, что несравненно удобнее). Остается только дать файлу имя, что осуществляется путем создания каталожной ссылки (directory link), а делает это команда «ln <N> undel_file_name», где undel_file_name – имя, которое будет дано файлу после восстановления, при необходимости с путем. (Внимание! создание каталожных ссылок необратимо затирает оригинальные имена удаленных файлов). После этого полезно дать команду «dirty», чтобы файловая система была автоматически проверена при следующей загрузке, или выйти из отладчика и вручную запустить fsck с ключом «-f», форсирующим проверку.

Теперь перейдем к восстановлению оригинального имени. Рассмотрим простейший случай, когда директория, содержащая удаленный файл (также называемая материнской директорией) все еще цела. Даем команду «stat dir_name» и запоминаем номер inode, который нам сообщает.

Говорим отладчику «dump <INODE> dir_file», где INODE – номер сообщенного нам индексного дескриптора, dir_file – имя файла на родной файловой системе, в которую будет записан дампы. Загружаем полученный дампы в hex-редактор и просматриваем его содержимое в «сыром» виде. Все имена будут там. При желании можно написать утилиту-фильтр, выводющую только удаленные имена. На Perl это не займет и десяти минут.

А как быть, если материнская директория тоже пострадала? Тогда она и будет помечена удаленной! Выводим список удаленных inode, отбираем из них директории, формируем перечень принадлежащих им блоков и записываем дампы в файл, просматриваемый вручную или с помощью утилиты-фильтра. Как уже отмечалось, порядок расположения файлов в списке inode очень часто совпадает с порядком расположения файлов в директории, поэтому восстановление оригинальных имен в четырех из пяти случаев проходит на ура.

При тяжких разрушениях, когда восстанавливаемый файл приходится собирать по кусочкам, помогает команда «dump_unused», выводящая на терминал все неиспользуемые блоки. Имеет смысл перенаправить вывод в файл, запустить hexedit и покопаться в этой куче хлама – это, по крайней мере проще, чем лазить по всему диску (на дисках, заполненных более чем на три четверти, данный трюк сокращает массу времени).

Вывод: debugfs в значительной мере автоматизирует восстановление удаленных файлов (впрочем, до полного комфорта ему далеко), однако восстановить файл с разрушенным inode он не способен и без ldd здесь не обойтись.

Восстановление при помощи R-Studio

Утилита R-Studio for NTFS, уже рассмотренная нами в предыдущих номерах журнала, вопреки своему названию, подерживает не только NTFS, но и ext2fs/ext3fs.

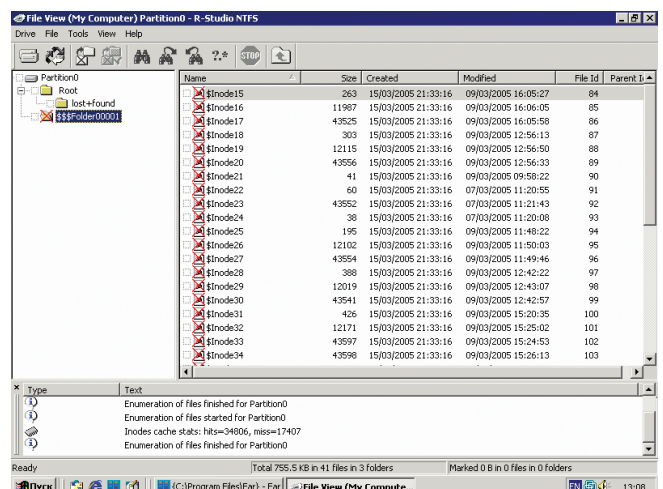


Рисунок 6. Утилита R-Studio for NTFS восстанавливает удаленные файлы на ext2fs разделе. Файлы есть, но нет имен

С точки зрения неквалифицированных пользователей это хорошее средство для автоматического восстановления удаленных файлов: интуитивно-понятный интерфейс, простое управление и прочие прелести прогресса. Файлы восстанавливаются несколькими щелчками. Правда, без имен и без каких-либо гарантий, что они вообще восстановятся.

Ручное восстановление не поддерживается. Файлы с разрушенным inode не восстанавливаются, хотя под ext2fs, в отличие от NTFS, это достаточно просто сделать!

В общем, для профессионального использования R-Studio категорически не подходит (рис. 6).

Восстановление удаленных файлов на ext3fs

Файловая система ext3fs – это ext2fs с поддержкой журналирования (в терминологии NTFS – транзакций). В отличие от ext2fs она намного бережнее относится к массиву директорий и при удалении файла, ссылка на inode уже не уничтожается, что упрощает автоматическое восстановление оригинальных имен. Однако поводов для радости у нас нет никаких, поскольку в ext3fs перед удалением файла список принадлежащих ему блоков тщательно вычищается. Как следствие – восстановление становится невозможным. Ну... практически невозможным. Нефрагментированные файлы с более или менее осмысленным содержимым (например, исходные тексты программ) еще можно собрать по частям, но и времени на это уйдет... К счастью, косвенные блоки не очищаются, а значит, мы теряем лишь первые 12 * BLOCK_SIZE байт каждого файла. На типичном 10 Гб разделе BLOCK_SIZE обычно равен 4 или 8 Кб, т.е. реальные потери составляют менее 100 Кб. По современным понятиям – сущие пустяки! Конечно, без этих 100 Кб большинство файлов просто не запустятся, однако недостающие 12 блоков найти на диске вполне реально. Если повезет, они окажутся расположенными в одном-двух непрерывных отрезках. Даже на сильно фрагментированных разделах непрерывные отрезки из 6-12 блоков встречаются достаточно часто.

Как мы будем действовать? Необходимо найти хотя бы один блок, гарантированно принадлежащий файлу и при этом расположенный за границей в 100 Кбайт от его начала. Это может быть текстовая строка, копирайт фирмы, да все что угодно! Нам нужен номер блока. Пусть для определенности он будет равен 0x1234. Записываем его в обратном порядке так, чтобы младший байт располагался по меньшему адресу, и выполняем поиск 34h 12h 00h 00h – именно это число будет присутствовать в косвенном блоке. Отличить косвенный блок от всех остальных блоков (например, блоков, принадлежащих файлам данных) очень легко – он представляет собой массив 32-битных номеров блоков более или менее монотонно возрастающих. Дважды и трижды косвенные блоки отыскиваются по аналогичной схеме.

Проблема в том, что одни и те же блоки в разное время могли принадлежать различным файлам, а значит, и различным косвенным блокам. Как разобраться, какой из них правильный? Да очень просто! Если хотя бы одна из ссылок косвенного блока указывает на уже занятый блок, данный косвенный блок принадлежит давно удаленному файлу и нам совсем не интересен.

Кстати говоря, debugfs не вполне хорошо поддерживает ext3fs. В частности, команда lsdel всегда показывает ноль удаленных файлов, даже если был стерт весь раздел. Так что вопрос выбора файловой системы отнюдь не так прост, каким его пытаются представить книги из серии «Linux для

начинающих», а преимущества ext3fs на рабочих станциях и домашних компьютерах далеко небесспорны и неочевидны. Поддержка транзакций реально требуется лишь серверам (да и то не всем), а вот невозможность восстановления ошибочно удаленных файлов зачастую приносит намного большие убытки, чем устойчивость файловой системы к внезапным отказам питания.

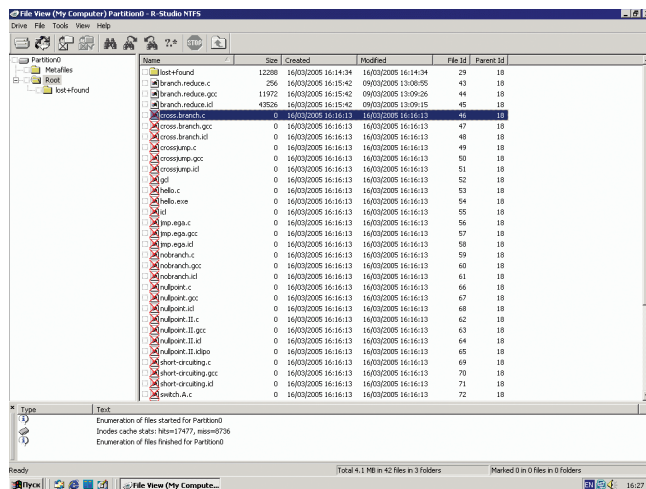


Рисунок 7. Утилита R-Studio, восстанавливающая удаленные файлы на разделе ext3fs. Есть имена, нет самих файлов (их длина равна нулю, т.к. список непосредственных блоков затерт)

Заключение

Доступность исходных текстов драйвера файловой системы значительно упрощает исследование ее внутренней структуры, которая, кстати говоря, очень проста и восстановление данных на ext2fs/ext3fs – обычное дело. Файловые системы UFS и FFS, работающие под FreeBSD, устроены намного сложнее, к тому же достаточно скудно документированы. А ведь FreeBSD занимает далеко не последнее место в мире UNIX-совместимых операционных систем и разрушения данных даже в масштабах небольшого городка происходят сплошь и рядом. К счастью, в подавляющем большинстве случаев информацию можно полностью восстановить, но об этом – в следующий раз.

Литература:

1. Design and Implementation of the Second Extended File system – подробное описание файловой системы ext2fs от разработчиков проекта на английском языке: <http://e2fsprogs.sourceforge.net/ext2intro.html>.
2. Linux Ext2fs Undeletion mini-HOWTO – краткая, но доходчивая инструкция по восстановлению удаленных файлов на ext2fs-разделах на английском языке: <http://www.praeclarus.demon.co.uk/tech/e2-undel/howto.txt>.
3. Ext2fs Undeletion of Directory Structures mini-HOWTO – краткое руководство по восстановлению удаленных директорий на ext2fs-разделах на английском языке: <http://www.faqs.org/docs/Linux-mini/Ext2fs-Undeletion-Dir-Struct.html>.
4. HOWTO-undelete – еще одно руководство по восстановлению удаленных файлов на ext2fs-разделах при помощи редактора lde на английском языке: <http://lde.sourceforge.net/UNERASE.txt>.