摘要

ext2 文件系统下恢复误删除的文件

本系的 BBS 系统真是多灾多难 (嗯 .... 其实是因为我的疏忽，才会这么多灾多难 ....) ，继这几日系统时间不正确，造成许多人的 ID 被误砍后，又一次因系统设定上的问题，将 BBS 的重要备份档给杀了。这件事是学弟发现后告诉我的，当我上站来一见到他的 mail, 当真是欲哭无泪，差点没去撞墙。

那时已是周六晚 11:00 左右，我一边想着要编一套说辞向大家解释无法替大家恢复旧信件与设定了，一边还在想是否能够挽回局面。大家知道，UNIX like 的系统是很难像 M$ 的系统一样，做到 undelete 的，所有网管前辈都曾再三警告我们，要小心! 小心! 砍档之前三思而后行，砍了之后再后悔也没用。虽然我已渐渐做到砍档三思而后行，但之次误砍事件是系统在背景中定时执行的，等到我找出原因时已是档案被砍后一个多小时。我凭着一点点的印象，想起在网络上，有人讨论过在 Linux ext2 filesystem中 undelete 的可能性，但我所见到的多半是负面的答案，但好象真的有人做过这件事，于是我第一个所做的，就是马上将该档案原来所在的 partition mount成 read-only, 禁止任何的写入动作，不是怕再有档案被误砍 (因为已没什么可砍的了) ，而是怕有新档案写进来，新资料可能会覆盖到旧资料原本存在的磁区 (block) 。我们现在唯一个指望，就是企图将档案原来存在的磁区一个个找回来，并且「希望」这些磁区上的旧资料都还在，然后将这些磁区串成一个档案。终于被我找到了!! 原来这方面的技术文件就存在我自己的系统中 :-))

/usr/doc/HOWTO/mini/Ext2fs-Undeletion.gz

于是我就按照这份文件的指示一步步来，总算将一个长达 8MB 的压缩档救回了 99%, 还有一个长达 1.1 MB 的压缩档完整无缺地救了回来。感谢上帝、 Linux 的设计者、写那篇文件的作者、曾经讨论过此技术的人、以及 Linux 如此优秀的 ext2 filesystem, 让我有机会抢救过去。现在，我将我的抢救步骤做一个整理让大家参考，希望有派得上用场的时候 (喔! 不，最好是希望大家永远不要有机会用到以下的步数 :-)))

在此严正声明!! 写这篇文章的目的，是给那些处于万不得已情况下的人们，有一个挽回的机会，并不意味着从此我们就可以大意，砍档不需要三思。前面提到，我有一个档案无法 100% 救回，事实上，长达 8MB 的档案能救回 99% 已是幸运中的幸运，一般的情况下若能救回 70% - 80% 已经要愉笑了。所以，不要指望 undelete 能救回一切。预防胜于治疗! 请大家平时就养成好习惯，砍档前请三思!!!

理论分析

我们能救回的机会有多大? 在 kernel-2.0.X 系列中 (本站所用的 kernel 是 2.0.33) ，取决以下两点:

档案原来所在的磁区是否没有被覆写?

档案是否完全连续?

第一点我们可以与时间竞赛，就是当一发现档案误砍时，要以最快的速度 umount 该 filesystem, 或将该 filesystem remount 成唯读。就这次的情况而言，档案误砍是在事发一个小时后才发现的，但由于该 filesystem 写入的机会很少 (我几乎可确定一天才只有一次，做 backup)，所以第一点算是过关了。

第二点真的是要听天由命了，就本站所使用的 kernel, 必须要在假设「长档案」所占的 block 完全连续的情况下，才有可能完全救回来! 一个 block 是 1024 bytes,长达 8 MB 的档案就有超过 8000 个 block。在经常读写的 filesystem 中，可以想见长档案很难完全连续，但在我们的系统中，这一点似乎又多了几分指望。同时，Linux ext2 如此精良的 filesystem, 能做到前 7950 多个 block 都连续，这一点也功不可没。

好了，以下我就讲一下我的步骤。

抢救步骤 I - mount filesystem readonly

该档案的位置原来是在 /var/hda/backup/home/bbs 下，我们系统的 filesystem 组态是:

```
root@bbs:/home/ftp/rescue# df
Filesystem 1024-blocks Used Available Capacity Mounted on
/dev/sda1 396500 312769 63250 83% /
/dev/sda3 777410 537633 199615 73% /home
/dev/hda1 199047 36927 151840 20% /var/hda
/dev/hda2 1029023 490998 485710 50% /home/ftp
```

因此 /var/hda 这个 filesystem 要马上 mount 成 readonly (以下请用 root 身份):

```
mount -o remount,ro /var/hda
```

当然也可以直接 umount 它，但有时候可能有某些 process 正在此 filesystem下运作，您可能无法直接 umount 它。因此我选择 mount readonly。但您也可以用:

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

看一下目前是那些 process 在用这个 filesystem, 然后一一砍掉，再 umount。

抢救步骤 II

执行

```
echo lsdel | debugfs /dev/hda1 | less
```

看一下该 filesystem 最近被砍的 inode (档案) 有那些 (为什么是 /dev/hda1? 请见上头的 df 列表)? 在这奶F档案的重要资讯，如大小、时间、属性等等。就我们的系统而言，其列示如下:

```
debugfs: 92 deleted inodes found.
Inode Owner Mode Size Blocks Time deleted
.....
29771 0 100644 1255337 14/14 Sat Jan 30 22:37:10 1999
29772 0 100644 5161017 14/14 Sat Jan 30 22:37:10 1999
29773 0 100644 8220922 14/14 Sat Jan 30 22:37:10 1999
29774 0 100644 5431 6/6 Sat Jan 30 22:37:10 1999
```

请注意!我们必须要在档案大小、被砍时间等资讯中判断出要救回的档案是那一个。在此，我们要救回 29773 这个 inode。

抢救步骤 III

执行

```
echo "stat <29773>" | debugfs /dev/hda1
```

列出该 inode 的所有资讯，如下:

```
debugfs: stat <29773>
Inode: 29773 Type: regular Mode: 0644 Flags: 0x0 Version: 1
User: 0 Group: 0 Size: 8220922
File ACL: 0 Directory ACL: 0
Links: 0 Blockcount: 16124
Fragment: Address: 0 Number: 0 Size: 0
```

ctime: 0x36b31916 -- Sat Jan 30 22:37:10 1999
atime: 0x36aebee4 -- Wed Jan 27 15:23:16 1999
mtime: 0x36adec25 -- Wed Jan 27 00:24:05 1999
dtime: 0x36b31916 -- Sat Jan 30 22:37:10 1999
BLOCKS:
123134 123136 123137 123138 123140 131404 131405 131406
131407 131408 131409 131 410 131411 131668
TOTAL: 14

现在的重点是，必须将该 inode 所指的档案，所指的 block 全部找回来。在这它?14 个 block? 不对啊! 应该要有 8000 多个 block 才对啊! 在这卯ilesystem 的「奥密」了。上头所列的前 12 个 block 是真正指到档案资料的 block, 称之为 direct block 。第 13 个称为第一阶 indirect block, 第 14 个称为第二阶 indirect block 。什么意思? 该档的资料所在的 block 位置如下:

各位明白吗? 第 13 个 (131411) 与第 14 个 block 其实不是 data, 而是 index,它指出接下来的 block 的位置。由于一个 block 的大小是 1024 bytes, 一个 int 在 32 位系统中是 4 bytes, 故一个 block 可以记录 256 笔资料。以 131411 block 为例，它所记录的资料即为 (在档案未砍前):

131412 131413 131414 .... 131667 (共 256 笔)

而这 256 个 block 就真正记录了档案资料，所以我们称为第一阶。同理，第二阶就有两个层 index, 以 131668 来说，它可能记录了:

131669 131926 132182 .... (最多有 256 笔)

而 131669 的 block 记录为:

131670 131671 131672 .... 131925 (共 256 笔)

而这 256 个 block 才是真正储存档案资料的。而我们要的，就是这些真正储存档案资料的 block 。理论上，我们只要将这些 index block 的内容全部读出来，然后照这些 index 把所有的 block 全部读到手，就能 100% 救回档案 (假设这些 block 全部没有被新档案覆写的话)。工程很大，但是可行。不幸的是，在 kernel-2.0.33, 其设计是，如果该档案被砍了，则这些 index block 全部会规零，因此我所读到的是

0 0 0 0 0 ..... (共 256 笔)

哇! 没办法知道这些 data block 真正所在的位置。所以，在此我们做了一个很大的假设: 整个档案所在的 block 是连续的! 也就是我上头的例子。这也就是为什么说，只有连续 block (是指后头的 indirect block) 的档案才能完整救回，而这一点就要听天由命了。

抢救步骤 IV

好了，现在我们只好假设所有的档案处于连续的 block 上，现在请用http://archie.ncu.edu.tw/去找这个工具: fsgrab-1.2.tar.gz, 并将它安装起来。因为步骤很简单，故在此我就不多谈。我们要用它将所需的 block 全部抓出来。它的用法如下:

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

其中 count 是只要 (连续) 读几个， skip 是指要从第几个开始读，例如我要从 131670 开始连续读 256 个，就这样下指令:

```
fsgrab -c 256 -s 131670 /dev/hda1 > recover
```

现在我们就开始救档案吧! 以上头的资料，我们必须用以下的指令来救: (注意到头开的 12 个 block 并没有完全连续!!!)

```
fsgrab -c 1 -s 123134 /dev/hda1 > recover
fsgrab -c 3 -s 123136 /dev/hda1 >> recover
fsgrab -c 1 -s 123140 /dev/hda1 >> recover
fsgrab -c 7 -s 131404 /dev/hda1 >> recover
```

这是开头的 12 个 block, 对于第一阶 indirect, 就资料来看好象是连续的 :-))

```
fsgrab -c 256 -s 131412 /dev/hda1 >> recover
```

注意要跳过 131411, 因为它是 index block。对于第二阶 indirect, 我们 *假设* 它们都是连续的:

```
fsgrab -c 256 -s 131670 /dev/hda1 >> recover
fsgrab -c 256 -s 131927 /dev/hda1 >> recover
fsgrab -c 256 -s 132184 /dev/hda1 >> recover
.....
```

要一直做，直到 recover 的大小超过我们所要救回的档案大小 (8220922) 为止。要注意在这市 p心地跳过那些 index block (如 131668, 131669, 131926, 132183, ...) 了。

抢救步骤 V

最后一步，就是把档案「剪」出来，并看看我们救回多少了。在这戊]我们重复上述步骤，弄出来的 recover 档大小为 8294400，而我们要的大小是 8220922, 那就这样下指令:

```
split -b 8220922 recover rec
```

则会做出两个档，一个是 recaa, 大小是 8220922, 另一个是 recab 则是剩下的大小，后者是垃圾，扔了即可。现在我们可以检查这个档案是不是「完整」的那个被误砍的档案了。由于我们的那个档案是 .tar.gz 的格式，于是我们这个方法来检查:

```
mv recaa recaa.tar.gz
zcat recaa.tar.gz > recaa.tar
```

如果没有错误讯息，那表示成功了！完全救回来了。但不幸的是，我们没有成功，将弄出的 recaa.tar 改名再 gzip 之后，与原来的 recaa.tar.gz 比一下大小，发现少了 1%, 表示说该档原来所在的 block 中最后有 1% 是不连续的 (或者被新写入的档案覆写了)，但这已是不幸中的大幸了。

后记

对于在 undelete 时 *必需* 假设所有 block 连续的问题，那份 HOWTO 文件说 Linus 与其它 kernel 设计者正着手研究，看能否克服这个困难，也就是在档案砍掉时，不要将 index block 规零。我刚刚试一

下 kenrel-2.2.0 的环境，发现已做到了!! 以下是一个已砍的档案的 inode data (由 debugfs 所读出):

debugfs: Inode: 36154 Type: regular Mode: 0600 Flags: 0x0 Version: 1
User: 0 Group: 0 Size: 2165945
File ACL: 0 Directory ACL: 0
Links: 0 Blockcount: 4252
Fragment: Address: 0 Number: 0 Size: 0
ctime: 0x36b54c3b -- Mon Feb 1 14:39:55 1999
atime: 0x36b54c30 -- Mon Feb 1 14:39:44 1999
mtime: 0x36b54c30 -- Mon Feb 1 14:39:44 1999
dtime: 0x36b54c3b -- Mon Feb 1 14:39:55 1999
BLOCKS:
147740 147741 147742 147743 147744 147745 147746 147747 147748 147769
147770 157642 157643 157644 157645 157646 157647 157648 157649 157650
157651 157652 157653 157654 157655 157656 157657 157658 157659 157660
157661 157662 157663 157664 157665 157666 157667 157668 157669 157670
157671 157672 157673 157674 157675 157676 157677 157678 157679 157680
157681 157682 157683 157684 157685 157686 157687 1.....
.....9745 159746 159747 159748 159749 159750 159751 159752 159753
159754 159755 159756
TOTAL: 2126

真是太完美了!! 这意味着在 kernel-2.2.X 的环境下，我们不必假设所有的 block 都连续，而且可以百分之百找回所有砍掉的 block! 因此上述的第二个风险就不存在了。

以上资料，谨供参考。