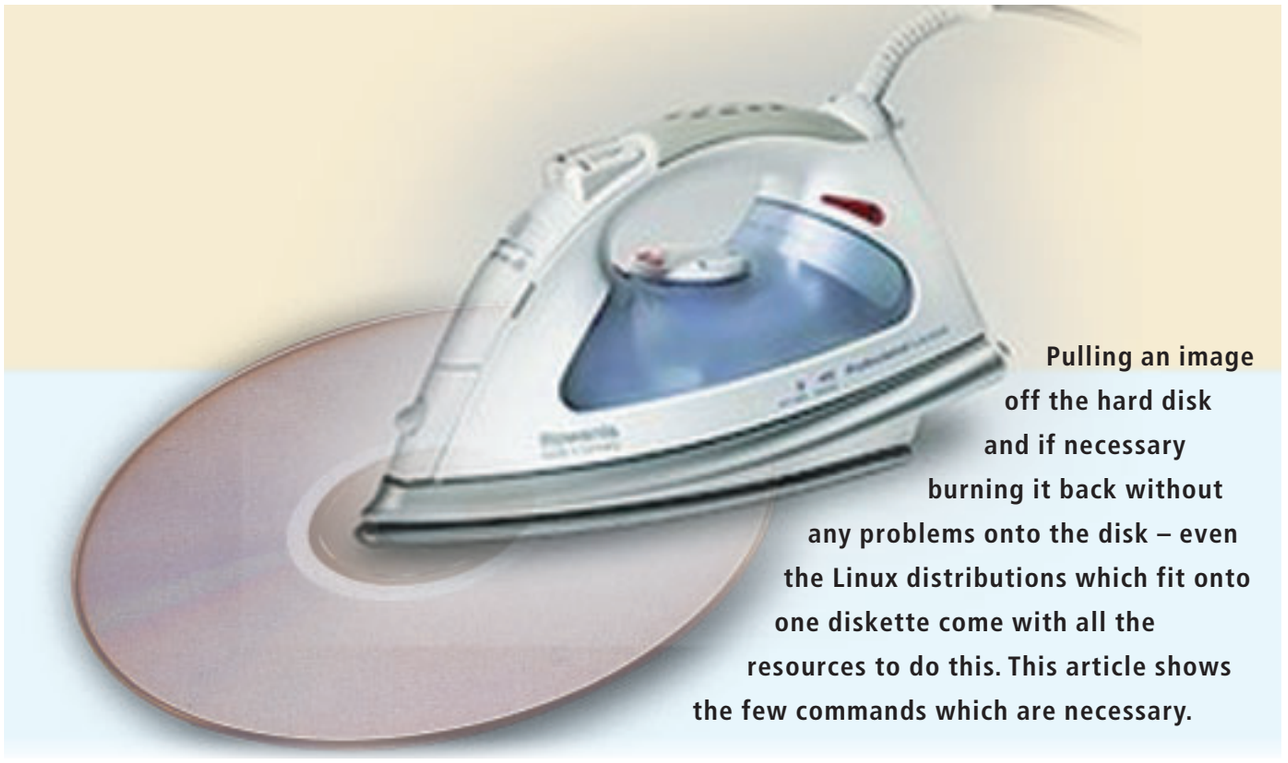
040CDRecover.qxd 29.06.2001 20:02 Uhr Seite 40

DIY Recovery CDs

SPEEDY RECOVERY

BERNHARD BABLOK



Pulling an image off the hard disk and if necessary burning it back without any problems onto the disk – even the Linux distributions which fit onto one diskette come with all the resources to do this. This article shows the few commands which are necessary.

What gave rise to this whole story was a friend whose Windows operating system went out on strike after his wife had installed a game for children. At this point, it would be fairer to point out that this was due neither to the fact that a woman had loaded it, nor that this is because it was running under Windows. Even under Windows neat installation routines can be written, but once the system has been corrupted, usually the only remedy is re-installation.

Under Windows this is as simple or complicated as it is under Linux; so it's not a job which can be performed by real computer amateurs. A remedy is provided by the little project proposed here, with the aid of which even beginners can create a self-booting CD, which will, after asking "Do you really want to ..." make the system workable once more. Configuration work no longer comes into it.

And for all those who often install and compare software, such a solution is interesting, too; because this is a simple way to guarantee identical initial conditions. And those who feel that they would happily shoot their system dead, can also be helped by it. Anyone who has to look after educational PCs has probably already implemented a similar procedure in order to get round the constant installation orgies.

First off, a warning: A recovery CD is no substitute for data back up. The procedure described here depends completely on the hardware. For example, if the hard disk is replaced, the CD is usually unusable. Another, somewhat more demanding approach (which will be discussed at the end), does get round this problem. But there again, the emphasis lies on the restoration of the system and not the user data.

040CDRecover.qxd 29.06.2001 20:02 Uhr Seite 41

Create an image

In order to create an image of a hard disk, this must of course be mounted. So that leaves two options: Install the hard disk in a second computer or boot from diskette or CD – either with a mini-distribution or a special boot CD and back up the image via the network.

In the first variant the creation of the image does go somewhat more quickly, but screwing it into the hardware is not always desirable. The second variant requires a Linux-supported network card. This could even be an old ISA-NE2000 card or a cheap PCI clone for around £10.

One very useful, single-disk version of Linux is Tomsrtbt. This packs almost everything the heart could desire onto one oversized diskette. It can be downloaded via *http://www.toms.net/rb/* or a mirror. It is also very simple to adapt it to your own requirements to recompose the diskette after making your own modifications, since the necessary scripts are also present.

Once the computer has rebooted, the image is created using the following command:

```
# dd if=/dev/hda bs=... count=... | \
rsh -l burner myhost \
"bzip2 -c > /home/burner/image/hda.bz2"
```

The command assumes that an image of the first hard disk is to be dragged to the IDE adapter. Under SCSI that would be the hard disk */dev /sda*. Similarly, the designations for additional hard disks must be altered. The command *rsh* is in fact normally frowned upon, but the computer, with the CD burner, must nevertheless still be physically accessible. This is why the security loophole (due to *rsh*) can be ignored in this case.

For the above command to function, a user *burner* must be installed on the computer *myhost*, which allows remote access via a suitable entry in */home/burner/.rhosts*:

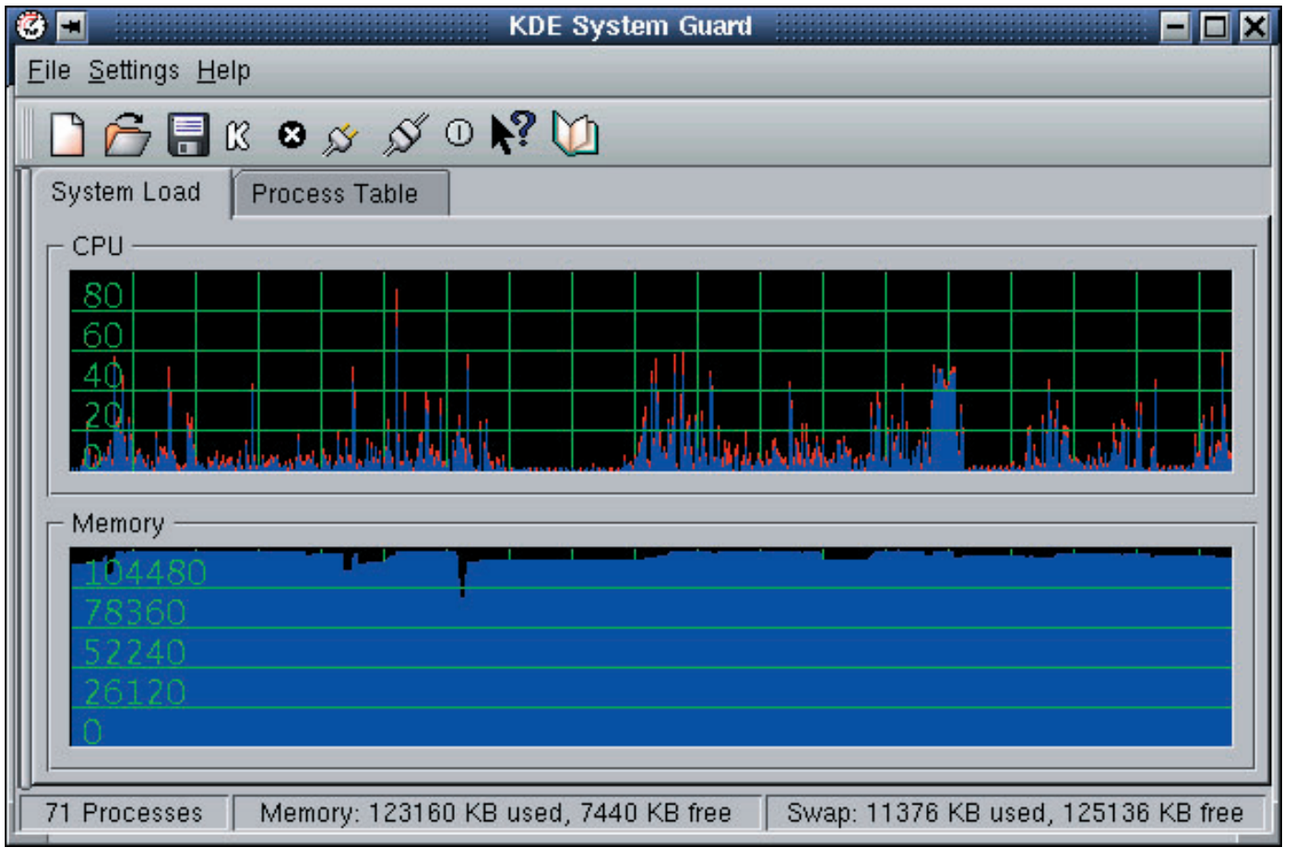
```
$ cat .rhosts
floppy.bablokb-local.de root
```

Floppy is the hostname of the computer booted via the diskette.

The *dd* command reads out the hard disk. The necessary parameters *bs=* (block size) and *count=* can be determined using an *fdisk -l*. On a test computer, an old laptop with a 2.1GB hard disk, the command showed that the hard disk possesses 525 cylinders, each with 8064 - 512 bytes. This means the *dd* command reads:

```
# dd if=/dev/hda bs=8064b count=525 | ...
```

The character *b* here stands for 512 bytes. Additional figures stand for other factors (which can be viewed via *dd --help*). Here, 525 blocks 4MB in volume have been read out. Since 525 = 5 x 5 x 7 x 3, an equivalent alternative would be the block size 56448b (7 x 8064 = 56448) with the count 75.



The system guard showing memory and CPU load. The drop in the middle was while writing zeros.

040CDRecover.qxd 29.06.2001 20:02 Uhr Seite 42

Simple inefficiency

It is simple to create the image via the aforementioned command, but inefficient and very slow. But on the other hand it is also valid indefinitely. It functions regardless of how many operating systems are on the hard disk, and regardless of the file systems used. It is precisely this latter which is now always a game of chance under Linux. Reiser FS, Ext3 JFS, each with and without logical volume manager or software Raid are just a few of the more recent developments which are causing more and more problems for classic rescue disks and backup programs.

In the figure below the KDE-2 system monitor can be seen on the destination computer during the creation of the image. In the lower part the bytes received and in the upper part the CPU loading are displayed. Notice the gaps that appear in the network traffic. These are due to the fact that *dd* is either reading or writing. Whenever a new block is being read by the hard disk, the network and the destination computer take a break.

Otherwise the slowest link in the chain is *bzip2*, which processes the arriving data in 900k blocks. If the sending computer is faster than the destination computer, it makes sense to compress the data before sending.

The whole image creation in this case took more than an hour, and the destination computer, at 700MHz, is comparatively fast. More later on additional details about this figure.

Restore

Once the image has been created, it goes onto a CD. How that works is explained in the "Creating the recovery CD" box. Restore then functions again after a boot with a floppy local via the following two commands:

```
# mount /dev/hdc /mnt
# bunzip2 -c /mnt/hda.bz2 > /dev/hda
```

Of course, a restore via the network would also be possible:

```
# rsh -l burner myhost \
"bunzip2 -c /home/burner/image/hda \.bz2"
$ /dev/hda
```

In the last case it is vital to make sure the output

Creating the recovery CD

The recovery CD is created in the usual way under Linux via Mkisofs/Cdrecord. The file structure appears as follows:

```
build
  | hda.bz2
  |
  | boot
         | hal91.img
```

The command

```
# mkisofs -b boot/hal91.img -c boot/hal91.cat -o reccd.iso
 build
```

creates the bootable CD-ROM image. The option -b refers to the bootable diskette image. Another file, the boot catalogue, also has to be created (option -c), but is otherwise unimportant. The output file is specified via -o. If hda.bz2 is a link to the image, the -f option must also be stated. If the ISO image has been created, the CD can be burnt using CDrecord or one of its front-ends. In our case, it looks like this:

```
# cdrecord -v -isosize fs=8m speed=4dev=x,y,z reccd.iso
```

fs is a buffer memory, speed the rate of the burner and dev the device of the burner, which can be determined via cdrecord -scanbus. These three parameters must be adapted by each person to suit their own circumstances.

Exchange the /linuxrc with HAL91

A Linux boot diskette almost always consists of three parts: a bootloader, the kernel and a pre-compressed file system.

The file linuxrc is in the root directory of this file system. To get to the system, the following steps are necessary:

```
# mkdir /tmp/floppy.mnt
# mount -o loop hal91.img /tmp/floppy.mnt
```

The diskette image is mounted via a loop device. The ability to mount such loop devices has to be compiled into the kernel, but this is usually the case with standard kernels in distributions. Then the compressed file system is unpacked and also mounted via a loop device:

```
# gunzip -c /tmp/floppy.mnt/initrd.gz > initrd
# mkdir /tmp/initrd.mnt
# mount -o loop initrd /tmp/initrd.mnt
```

Now we have access to linuxrc and can edit the file as described in the article:

```
# emacs /tmp/initrd.mnt/linuxrc
```

After that all the steps are to be executed more or less backwards:

```
# umount /tmp/initrd.mnt
# gzip -9c initrd > /tmp/floppy.mnt/initrd.gz
# umount /tmp/floppy.mnt
```

The HAL91 kernel does not support SCSI devices. So anyone who has a SCSI device should also swap the kernel. In HAL91 it is called vmloop. Since for our purposes the kernel hardly has to be able to do anything, apart from access the corresponding block device and CD-ROM support, the swap should not be a problem.

040CDRecover.qxd 29.06.2001 20:03 Uhr Seite 43

diversion is not in quotation marks, otherwise the hard disk will be overwritten by *myhost*.

But the original objective has not yet quite been reached, because a normal user cannot be expected to cope with booting by floppy and composing cryptic commands. Luckily, everything can be done automatically.

A CD as floppy substitute

Bootable CDs in accordance with the El Torrito standard do none other than make the Bios believe they are a bootable diskette. The recovery CD will thus, together with the hard disk image, also contain an image of our boot diskette. Since the Tomsrtbt floppy is a portrait format diskette it cannot be used for this purpose.

But mini-distributions are as common as pebbles on the beach. The best suited for the recovery CD is for example the boot diskette called HAL91.

Directly after booting, the kernel executes the file */linuxrc*. We are thus replacing this file from HAL91 with our own version, which basically contains the two commands mentioned above (*mount* and *bunzip2*). How this works in detail is explained at greater length in the listing "A modified */linuxrc*".

Since the restore is a fairly destructive matter and because a normal user is accustomed to the constant challenges "Do you really want to ...", it is advisable to give the user a last opportunity to stop. The listing shows one option for this.

Space problems

One important question has not yet been dealt with: What will fit onto a CD? The aforementioned laptop hard disk has three partitions. The first with a capacity of 1GB contains a freshly installed Windows 98. In addition to this there is a Linux partition of 800MB with a Mandrake 7.2 installation and a swap partition with the remaining space.

Originally the computer was only installed in this configuration in order to test which stunts were necessary to install Windows 98 as an addon to a Linux computer – in almost all tests it is only the reverse case which is investigated and evaluated.

Of the FAT32 partition, 210MB was occupied. The Ext2 partition on the other hand 674MB. An image was created, as described above. Surprisingly, this had more than 635MB and thus did fit onto one CD, but was much too big for this specific example.

To check the process, the content of both partitions was backed up with a classic *tar -cvpl*. The compressed Tar archive of the Windows partition was 87MB in size, the Linux archive 195MB. Then the hard disk was repartitioned (one big Ext2 partition) and formatted. A newly created image of this almost empty hard disk still came to a solid 616MB.

The reason for this astonishing size lies in the fact that repartitioning and reformatting only changes the administrative information of the hard disk and partitions. The actual data remains unaffected. Before the created Tar archives were played back, the whole hard disk was overwritten with zeroes:

```
# dd if=/dev/zero of=/dev/hda bs=... count=...
```

An image of the empty Ext2 partition, with this pre-processing, only comes to just under 109KB, an enormous difference from the 616MB determined at first. Equally, after restoration to its original condition (with Windows 98 and Mandrake Linux) the image came to a reasonable size of just under 283MB. In the "System loading" display, you can see quite clearly when the zeroes are transferred and compressed. In the central part the CPU loading drops dramatically, while the network is more heavily loaded.

30GB on one CD

The compressed Tar archives show that a Windows installation can be reduced to just under 41 per cent, while with Linux it is even possible to attain a value of under 30 per cent. This is probably due to the high proportion of text files (for example HTML

Listing: a modified /linuxrc

```
01: #!/bin/sh
02:
03: PATH="/bin:."
04: TERM=linux
05: export PATH TERM
06:
07: mount /proc/ /proc -t proc
08:
09: mount -o ro /dev/hdc /mnt
10: echo "Should the hard disk be overwritten (all data will be lost)?"
11: until [ "x$answer" = "xYES" -o "x$answer" = "xNO" ]; do
12:   echo -n "Confirm with YES or stop with NO! "
13:   read answer
14:   if [ "x$answer" = "xYES" ] ; then
15:     echo "overwriting the hard disk. Please wait..."
16:     bunzip2 -c /mnt/hda.bz2 > /dev/hda
17:   elif [ "x$answer" = "xNO" ] ; then
18:     echo "Stop!"
19:   fi
20: done
21: umount /mnt
22: echo "Please remove the CD-ROM and press CTRL-ALT-DEL"
23: sh
```

Comparison of bzip2 with gzip

Command\File	zero	random	opt-kde2.tar
bzip2 (bytes)	113	105.321.149	27.893.829
gzip (bytes)	101.801	104.874.289	31.631.186
bzip2 (time)	20.4s	264s	150s
gzip (time)	8.1s	48s	88s
bunzip2 (time)	4.2s	89s	42s
gunzip (time)	4s	12s	5.7s

040CDRecover.qxd 29.06.2001 20:03 Uhr Seite 44

documentation, scripts, configuration files) under Linux.

An empty (zeroed) 30GB hard disk in compressed condition could take up some 1.5MB. If 2GB of the hard disk are taken up by a Linux system, then the image should still fit onto a recovery CD. For Windows the limit is around 1.5GB. Mind you, these are operating systems and programs. If compressed applications files, maybe in MP3 format, are present, the calculation will look very different. If there are several operating systems on the hard disk, space on the CD will also soon run out. But a recovery CD, as described here, is not suited to such systems anyway.

Optimisations

It may just be acceptable to wait more than an hour for the image of a 2.1GB disk. But for a really large hard disk the whole process adds up to more than a whole day. So which optimisation options exist?

As described, *bzip2* and *bunzip2* are ultimately responsible for the time taken to create the image and to do the restore. A highly practical alternative is to use *gzip/gunzip* for this task. In the "Comparison" table, sizes and times for compression and decompression of three files are listed.

The file *zero* consists of 100MB zeroes (created from */dev/zero*), the file *random* out of 100MB random numbers (created out of */dev/urandom*) and the file *opt-kde2.tar* is an uncompressed Tar archive from the */opt /kde2* directory of my computer. The archive also comes to almost 100MB. It is apparent from the table that with real data a time gain of about 40 per cent balances out a reduction in size of some 10 per cent. When the data is already compressed, the performance gap is even more marked, plus in this case *bzip2* also has the greater overhead.

But it is only in the case of blank data that a really significant difference can be noted in the size of files: *bzip2* reduces the 100MB zeroes in the *zero*-file to a total of 113 bytes, while *gzip* still produces a result which is almost three powers of ten higher.

With an image, which typically contains data from these three basic types, the result is obviously somewhat less extreme. The time saving is only about 30 per cent, while the size increases by 10 per cent. In the case of large, still mostly vacant disks, though, the calculation may look rather different again.

Relay race

Another problem is the large number of programs needed to get the image onto the hard disk of the destination computer. *dd* reads the data out and writes it into a pipe. From there it reads *rsh*, only to immediately write it back into a socket. On the other side the *rshd* daemon then reads the data from the socket, writes it into a pipe, from where it is then met by *bzip2*. A real relay race is taking place here between the programs. The ideal would thus be a network-capable *dd*, which writes the data directly into a socket, interacting with an equally network-capable *bzip2*, which can read the data out of a socket. Since the sources are open, these expansions should not mean any great expense. So if anyone is looking for an interesting programming task, they could try their hand at this.

One more important optimisation would be a *dd* which can read and write at the same time. The source computer in this case could send at full network bandwidth and the throughput from *bzip2* would then be the only bottleneck.

Short of just rewriting *dd* (which would certainly be the better solution), the variant was also investigated where the program buffer is interconnected between *dd* and *rsh* and/or before *bzip2*. It stores the data in a ring buffer in the main memory and can read and write at the same time from there. The only thing to watch for here is that the memory volume allocated by *dd* and buffer combined will still fit into the RAM.

With this double buffering on both sides of the network it is possible to achieve a time saving of about 25 per cent with *bzip2* and 10 per cent with *gzip*. Unfortunately buffers are seldom found on rescue floppies. Which is one good reason to simply create your own bootable CD with a comprehensive Linux system yourself.

Regardless of these optimisations, there is one hole in the solution described here. There may be (and in the case of large disks, absolutely certainly) a huge amount of useless zeroes being read, transferred and compressed. At block device level, though, there are only bytes, no contents. An intelligent alternative definitely requires knowledge about the contents at file system level.

Alternatives

A short search of the Internet at Freshmeat and Sourceforge also brings some corresponding solutions to light. The Belgian project MkCDrec creates, from a running Linux system, a recovery CD – or several, if everything will not fit onto one CD. This is intended for system administrators and therefore not automated, but that could probably be easily altered.

The number of file systems supported is limited. In normal Linux systems, however, MkCDrec has the great advantages of efficiency and flexibility. The files are ultimately backed up with Tar. This means that only actually existing data are stored, plus if you do a restore a different partitioning is possible. All in all an extraordinary tool, which is continually being improved and, for all those who do not

Info

Homepage of Tomsrtbt: http://www.toms.net/rb/

HAL91 Homepage: http://home.tu-clausthal.de/~incp/hal91/

Freshmeat: http://www.freshmeat.net

Sourceforge: http://www.sourceforge.net

MkCDrec Homepage: http://mkcdrec.ota.be

Partimage Homepage: http://www.partimage.org

Source of Sfdisk, also part of MkCDrec: ftp://win.tue.nl

040CDRecover.qxd 29.06.2001 20:03 Uhr Seite 45

necessarily need a solution for inexpert users, the right choice.

A different approach is offered by Partimage. This is a low-level tool for backing up partitions. At the moment FAT16/32, Ext2 and Reiser-FS partitions are supported. The contents of the partitions are analysed and only the used blocks are backed up.

The current production version is however even slower than the *dd* solution. Firstly, reading the used blocks is very slow but the compression is done beforehand on the source computer. On the credit side, Partimage offers an intuitive interface, including such things as progress indicators, and the option of distributing the image over several media. In an emergency it would be possible to back up a partition on diskettes.

The latest beta version also has a client-server mode, implementing a whole range of optimisation options, such as simultaneous reading and writing or the encrypted transfer of data to the server.

This version is described as quite stable, but suffers from the lack of documentation for the log-in mechanism.

Building on the basis of Partimage, the following procedure for the creation of an optimised recovery CD would be possible, if only the supported file systems are present:

- Reading out the partition information, maybe with Sfdisk. The program allows the partition information to be output in a format which can be used by Sfdisk again as input.
- For each partition a corresponding image file is created via Partimage.
- The images are burnt, together with the Sfdisk input file, onto the CD.
- The */linuxrc* program of the boot image of the CD repartitions the hard disk via Sfdisk – using the corresponding input file – and writes all the images back onto the hard disk.

The advantage of the last procedure, apart from faster image creation and faster restore is the option of backing up larger hard disks, possibly with several operating systems, on several CDs.

Conclusion

As you have seen, there are various approaches when it comes to restoring a hard disk. All the means necessary for this are either supplied with simple distributions or are freely available on the Internet.

But a final pointer - all the solutions considered are unsuitable as back up procedures for normal user data. Who wants to restore a whole hard disk or partition just because a corrupted file has to be replaced? ■

1/2 ad