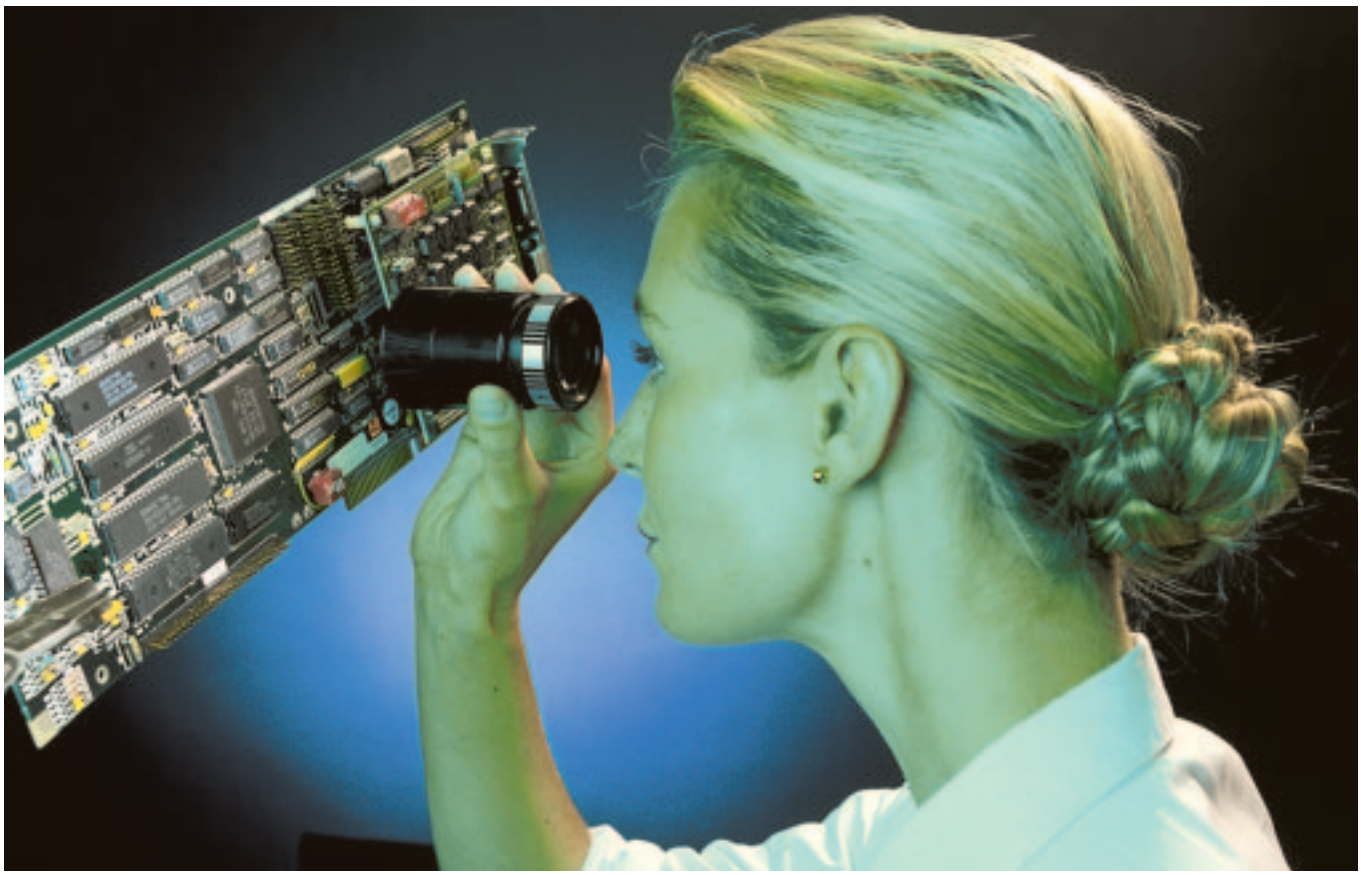
Dr. Linux

Safe and Sound

Is your CD image file or floppy in good working order? Doctor Linux can help you find out.

BY MARIANNE WACHHOLZ



Before burning the **ISOs** I just downloaded off the Internet to CD, I would like to check whether the files are 100% error free. What Linux program can I use to do so?

Dr. Linux: One possible way of verifying an ISO image is to **mount** the file. If this works, you can assume that the data was transferred correctly during download, but this does not give you a 100% guarantee that the file is error free, only that you have a working copy of what was available.

So-called loop devices can be used by the superuser, *root*, to insert files such as hard disks or floppies into the directory tree. You can use *mount* with the *-o loop* option to do so. Make sure that you change the syntax in the following example to reflect your own directory structure names!

```
perle@linux:~/iso> su root
Password: Your_Root_Password
linux:/home/perle/iso # mount ➥
-o loop -t iso9660 /this/is/➥
my.iso /mnt/
```

SuSE users tend to mount in */media* or a subdirectory below this level:

```
mount -o loop -t iso9660 /➥
path/to/my.iso /media/
```

The complete content of the ISO image is now available below the directory supplied in the last argument, the mount point, exactly like it would be if you burned and mounted a CD with this content later. After verifying the files you can remove the image from the directory tree by typing *umount mountpoint* and then drop your superuser status by typing *exit*.

If you want to ensure that an ISO file you downloaded to your home PC is error free, you can compare the checksum of the original file with the

Doctor Linux

Complicated organisms, which is just what Linux systems are, have some little complaints all of their own. Dr. Linux observes the patients in the Linux newsgroups, issues prescriptions here for the latest problems and proposes alternative healing methods.

checksum of the file you downloaded. The checksum is a numerical value calculated by an algorithm with reference to the total sum of the bits that the file contains.

There are various programs for calculating checksums, such as *sum* or *cksum*. As they all use different algorithms, the checksums created by them are not compatible, so you will need to use the same tool to create your checksum as was used to create the checksum for the original file.

At present you will almost always discover that the *MD5* algorithm [1] has been used to checksum a downloadable file. The *md5sum* program is included as a standard component of any known Linux distributions. *md5sum* creates a 128 bit value for any file.

Checksums are typically stored in files ending in *.md5* or *.md5sum* on FTP or web servers (Figure 1):

```
ddee9456051785ebdd92f3d28a033e61➥
  gentoo-ix86-1.2.iso
```

MD5 checksum files thus contain only a few bytes – in contrast to ISO images – and it makes sense to save them in the same directory as the file used to create them. Sometimes administrators will collate several checksums to a single file or add them to files, such as *Readme.txt* or the like.

It is more or less impossible to generate the same checksum for two different files with *md5sum*. Even the slightest change to a file – and this could be caused by a transmission error – will immediately lead to a different checksum as the sum of the bits will now be different.

The uniqueness of checksums, which are often referred to as fingerprints, is used by administrators to discover system manipulations caused by files that have been injected or exchanged.

How can you verify a checksum that you have just downloaded? To see how this works let us look at this process using an ISO file from *Gentoo Linux* [2] as an example:

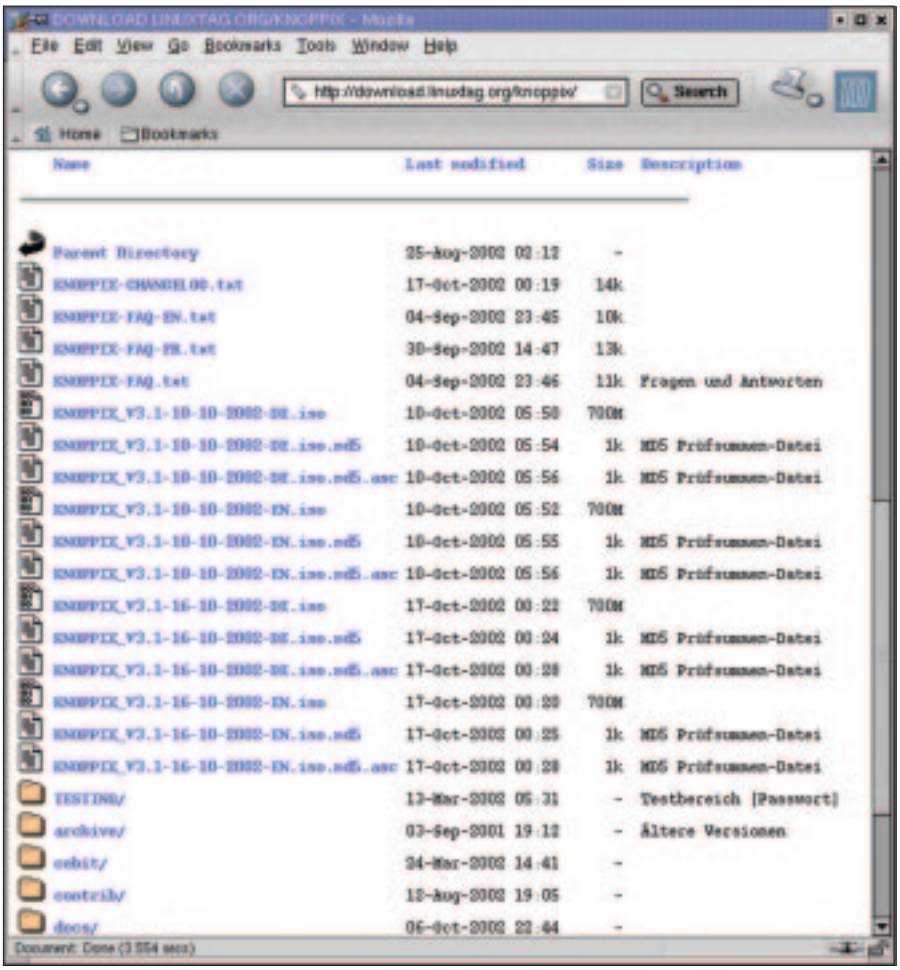


Figure 1: Knoppix ISO files [3] and their md5sum files

```
perle@linux:~/iso> ls -l
insgesamt 16540
-rw-r--r--    1 perle    users ➥
   16908288 Jun 23 13:48 ➥
gentoo-ix86-1.2.iso
-rw-r--r--    1 perle    users ➥
         54 Jun 23 13:49 ➥
gentoo-ix86-1.2.iso.md5
```

The ISO image and the corresponding MD5 file are both stored in the current working directory (in this case, */iso*). We now want to pass the checksum file to the */usr/bin/md5sum* program with the *-c* ("check") flag set. If everything turns out okay, the answer will be a single *ok*:

```
perle@linux:~/iso> md5sum ➥
-c gentoo-ix86-1.2.iso.md5
gentoo-ix86-1.2.iso: Ok
```

If the file, however, fails the test, *md5sum* issues a warning:

```
perle@linux:~/iso> md5sum ➥
-c gentoo-ix86-1.2.iso.md5
gentoo-ix86-1.2.iso: Error
md5sum: Warning: 1 of 1 ➥
calculated checksums did ➥
NOT match
```

GLOSSARY

ISOs: *Popular expression for files whose file system reflects the system independent ISO 9660 standard, which is used for burning CD ROMs.*

Mount: *Storage media are inserted into the Linux file system tree by means of the mount command, which requires root access. Before removing a mounted CD or floppy from the drive, you will need to issue the umount command. Access to hard disk partitions can also be disable in this way on Linux. The sysadmin can use the /etc/fstab file to allow unprivileged users to insert or remove certain media, such as CD ROMs for example.*

su: *You can use the "su username" command to assume the identity and rights of the selected user in the shell. After entering the correct password, you are returned to the current directory, but with the privileges of a superuser, for example, and can carry on working with root privileges.*

e2fsck: *This command-line tool verifies, and if needed (and possible), repairs Extended 2 file systems. This was the standard file system type for most Linux hard disk partitions, although many distributors have now switched to more modern file systems, such as the successor Ext3 or ReiserFS.*

If the MD5 value is included in a readme file, you therefore need to create a checksum yourself before you can compare it, pass the name of the ISO file to *md5sum* and verify the results visually:

```
perle@linux:~/iso> md5sum ➥
gentoo-ix86-1.2.iso
ddee9456051785ebdd92f3d28a033e61 ➥
 gentoo-ix86-1.2.iso
```

If your distribution happens not to include *md5sum*, you can download it from [4]; type *md5sum* as a search key. The program is part of the *Text utilities* package. Additional information is available from the GNU project at [5].

Error Free?

I have a few older floppies that I would like to use as "single floppy Linux" versions or boot disks. How can ensure that there are no bad blocks hiding on these disks?

Dr. Linux: You would normally want to use the */sbin/badblocks* tool to ensure that there are no *bad blocks* on the disk. This tool is part of a collection designed for verifying, maintaining and creating file systems on (almost) any Linux system. Maintenance programs, such as ***e2fsck***, can process the output from *badblocks*. As you would normally only want to use floppies that are free from errors (although there might be some strange reason for using a damaged disk), we are not going to look into possible repair procedures in this article.

If you want to test a floppy with *badblocks*, you *cannot* mount it in the Linux directory tree. As superuser privileges are required to access the floppy, you may need to prefix the *badblocks* call with a call to *su* using the *-c* flag. This ensures that only the ensuing command, which must be enclosed in quotes, will be executed with *root* privileges.

You can use the *badblocks* option *-s* ("show") to show which block the program is currently processing. The *-v* flag ("verbose") will keep you up to date on the program's activity. But you are still on the safe side if you leave out these options as a message is given when a bad block is found.

According to the man page, you need to specify the number of blocks to check, but since floppies are verified by reference to the corresponding device file, you can leave this parameter out:

```
perle@linux:~> su -c "/sbin/➥
badblocks -s -v /dev/fd0"
Password:Your_Root_Password
Checking for bad blocks in ➥
read-only mode
>From block 0 to 1440
Checking for bad blocks (read-➥
only test):        16/     1440
```

In our example the program is currently testing blocks 16 through 1440. If the result is negative, for example, when there are no bad blocks, the program will report back to us with

```
Pass completed, 0 bad blocks ➥
found.
```

Listing 1: 1.44 MB Floppy with Bad Blocks

```
01 perle@linux:~> su -c "/sbin/badblocks -s -v /dev/fd0"
02 Password:Your_Root_Password
03 Checking for bad blocks in read-only mode
04 >From block 0 to 1440
05 Checking for bad blocks (read-only test): 12      0/     1440
06 13     13/     1440
07 14
08 15     15/     1440
09 done
10 Pass completed, 4 bad blocks found.
```

Listing 2: Major and Minor Numbers for Floppy Devices

```
perle@maxi:~> ls -al /dev | less
[...]
brw-rw----    1 root     disk       2,   0 Jun  6 17:13 fd0
brw-rw----    1 root     disk       2,  36 Sep 24  2001 fd0CompaQ
brw-r--r--    1 root     root       2,  60 Jun 28 14:28 fd0H1722
brw-rw----    1 root     disk       2,   4 Sep 24  2001 fd0d360
brw-rw----    1 root     disk       2,   8 Sep 24  2001 fd0h1200
[...]
```

Listing 3: Ostensible bad blocks due to an incorrect floppy device

```
01 perle@linux:~> su -c "/sbin/badblocks -s -v /dev/fd0 1722"
02 Password: Your_Root_Password
03 Checking for bad blocks in read-only mode
04 >From block 0 to 1722
05 Checking for bad blocks (read-only test): 1440 1408/     1722
06 1441
07 1442
08 [...]
09 1720
10 1721
11 done
12 Pass completed, 282 bad blocks found.
```

Listing 4: Checking a 1722 kb floppy with badblocks

```
perle@linux:~> su -c "/sbin/badblocks -v /dev/fd0H1722"
Password:Your_Root_Password
Checking for bad blocks in read-only mode
>From block 0 to 1722
Pass completed, 0 bad blocks found.
```

Box 1: Floppy Disk Device Types

The manpage for *fd*-("*f*loppy *d*isk"-)devices specifies over 30 different device files that can be used to access floppy drives, including some fairly obscure 5.25 inch drives. The following short excerpt shows just a few of the various possibilities:

In the following list n refers to the drive number: 0 for the first drive, 1 for the second, and so on:

3.5 Inch High Density Devices:

Name	Capac	Cyl	Sect	Heads	Minor Base #
fd*n*H360	360K	40	9	2	12
fd*n*H720	720K	80	9	2	16
fd*n*H820	820K	82	10	2	52
fd*n*H830	830K	83	10	2	68
fd*n*H1440	1440K	80	18	2	28
fd*n*H1600	1600K	80	20	2	124
fd*n*H1680	1680K	80	21	2	44
fd*n*H1722	1722K	82	21	2	60
fd*n*H1743	1743K	83	21	2	76
fd*n*H1760	1760K	80	22	2	96
fd*n*H1840	1840K	80	23	2	116
fd*n*H1920	1920K	80	24	2	100

(provided you specified the *-v* option). In case of positive results, the bad blocks will be listed – refer to Listing 1, where blocks 12-15 are reported. In this case the program will report:: *Pass completed, 4 bad blocks found.*.

Figure 2: Gentoo Linux desktop

The Floppy Format Odyssey

My floppies have been through it all – all those attempts to increase their capacity by using different formats. But *badblocks* always shows an incredible number of bad blocks in this case, although they are not displayed if I stick to the standard 1440 kb format.

Dr. Linux: If the device file you supply does not match the **low level format** of the disk, *badblocks* will return with gibberish. Additionally, assigning the wrong device file can endanger your hardware. Avoid using device files that are inappropriate for your hardware type!

Various floppy drives can be accessed via the device files in the */dev* directory. They all have the **major device number** 2. The **minor device number** represents a floppy format for this type of hardware (see Box 1). If you list the directory */dev*, you are shown the major and minor numbers of the devices instead of file sizes. Listing 2 shows an example, you should not assume that it will be similar to the device files on your own system.

The kernel relies on this information to recognize the format when a floppy is opened and passes this information on to the relevant programs.

Let's look at an example to see how badly a verification with *badblocks* can go wrong if you supply the wrong device file: A floppy has been low level formatted using the *fdformat* program, and now has a capacity of 1722 kB:

```
perle@linux:~> su -c ➥
"fdformat  /dev/fd0H1722"
Password: Your_Root_Password
Double sided, 82 tracks, 21 ➥
sectors/track, total capacity: ➥
1722kB.
Formating ... done
Checking ... done
```

Now you take the floppy out of the drive and insert it again to suggest to the kernel that a new medium has been inserted. When verifying the disk with *badblocks*, you mistakenly refer to the device as */dev/fd0*, although you supply the correct number of blocks to be processed.

As a result 282 bad blocks are shown – blocks 1440 through 1721. That is, the blocks that exceed the capacity of */dev/fd0* (1440 blocks in the case of high density floppies) (Listing 3).

If you now choose the right device file, the floppy passes the test, as you would expect – refer to Listing 4 for details. ■

GLOSSARY

Low level format: *This does not mean writing a file system (minix, msdos) to the disk, but defining tracks and sectors. Disks with "raw" formats of this type can be written to using tar or dd.*

Major and Minor Numbers: *When a program accesses a device file, two numbers are passed to the kernel to indicate the request. The major number typically refers to a particular kernel driver and the minor number to the device for which access is required. This is why all the device files for the serial port have the same major number, but different minor numbers. In short, the kernel uses the major number to pass the request to the appropriate driver, and the driver uses the minor number to determine the device that needs servicing. There are a few exceptions but normal Linux users will normally not come across them.*

INFO

[1] RFC 1321: *http://www.fourmilab.ch/md5/rfc1321.html*

[2] Gentoo Linux: *http://gentoo.org/*

[3] Knoppix Download: *http://download.linuxtag.org/knoppix/*

[4] GNU Software: *http://www.gnu.org/directory/*

[5] Text Utilities: *http://www.gnu.org/software/textutils/textutils.html*