Loopback Device

From OSDev Wiki

A Loopback Device is a mechanism used to interpret files as real devices. The main advantage of this method is that all tools used on real disks can be used with a loopback device.

Note: This article only covers UNIX environments (including Cygwin). For information on how to use loopback devices on Windows, see diskpart.

Contents

Loopback Device under Linux

The linux loopback device can be used by root only, and needs to be enabled in the kernel before use.

Floppy Disk Images With GRUB and EXT2

First, lets create an empty image.

```
dd if=/dev/zero of=floppy.img bs=512 count=2880
```

Now, lets set it up for mounting.

```
losetup /dev/loop0 floppy.img
```

Now lets make it EXT2 formatted.

```
mkfs -t ext2 /dev/loop0
```

Mount!

```
mount -t ext2 /dev/loop0 /mnt/myfloppy
```

Create GRUB directory.

```
cd /mnt/myfloppy
mkdir grub
```

Copy GRUB's second stage files. (GRUB stage[12] could also be located in /usr/lib /grub/)

```
cp /lib/grub/i386-pc/stage[12] /mnt/myfloppy/grub
```

Create a device mapping for the GRUB installation. *You need quotations around the first part.*

```
echo "(fd0) /dev/loop0" > /mnt/myfloppy/grub/device.map
```

Start GRUB console for installation into the boot record.

```
grub --device-map=/mnt/myfloppy/grub/device.map /dev/loop0
```

In the GRUB console:

```
root (fd0)
setup (fd0)
```

NOTE: You must unmount /mnt/myfloppy before using a emulator to directly read /dev/loop0, such as:

```
qemu -fda /dev/loop0
```

NOTE: When deleting the loop device, the original floppy.img file will be saved with the modified contents.

Floppy Disk Images With FAT16

Create an empty image.

```
dd if=/dev/zero of=floppy.img bs=512 count=2880
```

Set it up for mounting.

```
losetup /dev/loop0 floppy.img
```

Make it MSDOS formatted.

```
mkdosfs /dev/loop0
```

Mount!

```
mount -t msdos /dev/loop0 /mnt/myfloppy
```

Hard Disk Images

A hard disk image contains an MBR, then a number of partitions, but the 'mount' instruction in Linux works with disk partitions, not full disks. To mount a partition contained in our disk image, we need to make sure the 'mount' command only sees our partition, not the whole disk.

Creating an image

First create the empty file that we will use for our disk image. We will assume a disk geometry of #cylinders, 16 heads, 63 sectors/track, 512 bytes/sector, which means that each cylinder contains 516096 bytes (16*63*512). Decide how large you want your disk image to be, and choose an appropriate number of cylinders (I'll be using #cylinders throughout).

Example: If I want a 500Mb disk, I would choose 1000 cylinders (approximation of (500*1000*1024)/516096).

Write the disk image (I'll assume the filename c.img throughout):

```
dd if=/dev/zero of=/path/to/c.img bs=516096c count=#cylinders
```

Explanation:

dd	Linux command for copy and convert a file
if=/dev/zero	Source file is /dev/zero which is...*drumroll*...an infinite source of zeros
of=/path/to/c.img	Destination file is our disk image (dd will create the file if it doesn't exist)
bs=516096c	Means read and write 516096 bytes at a time (This is just here to keep things simple)
count=#cylinders	Copy this number of blocks. Since we have set bs to 516096 bytes each block is one cylinder long

That leaves us with a nice sized file full of zeros that we'll use for our disk image.

Partitioning

Now to create the MBR and partition table on the disk image (Usually you need to be root).

```
fdisk -u -C#cylinders -S63 -H16 /path/to/c.img
```

Explanation:

fdisk	Linux DOS partition maintenance program.
-u	Display units in sectors not cylinders (We will need this).
-C#cylinders	Set the cylinders of disk to our value.
-S63	Set the sectors/track to 63.
-H16	Set the heads/track to 16.
/path/to/c.img	fdisk is capable of partitioning image files directly.

Within fdisk use the following commands:

```
o - Create a new empty DOS partition table.
n - Create a new partition (For simplicity just make 1 primary partition covering the whole disk).
a - Toggle the bootable flag (Optional).
p - Print the partition table.
```

You should end up with a screen that looks something like this:

```
Disk /path/to/c.img: 516 MB, 516096000 bytes
16 heads, 63 sectors/track, 1000 cylinders, total 1008000 sectors
Units = sectors of 1 * 512 = 512 bytes
```

```
     Device    Boot      Start         End      Blocks   Id  System
/path/to/c.img1   *          63     1007999      503968+  83  Linux
```

Obviously the cylinder count, partition end and blocks will be different depending on the size of your image.

Make a note of the start sector (63 here) and the block count (503968 here).

Note: If you are intending to format the partition to something other than ext2fs then change the partition id here using the t command. I should also point out that disk manufacturers and programmers don't agree on how many bytes are in a megabyte.

```
w - Write partition table to our 'disk' and exit.
```

Ignore any errors about rereading the partition table. Since it's not a physical device we really don't care.

We now have a partition table on our disk image.

Unfortunately this also means that from here on out we have to account for the fact that our partition does not start at byte 0 of the image.

Mounting

Ok, now we attach the file to the loopback device, in such a way that we skip everything before the start of our partition.

```
losetup -o32256 /dev/loop0 /path/to/c.img
```

Explanation

-o32256	Move the start of data 32256 bytes into the file

The reason we move 32256 bytes into the file is this is where the partition starts. Remember I said to note the start sector of the partition (63 is usual)? Well, since each sector is 512 bytes long we therefore know the starting byte of the partition is 32256 (63*512) bytes into the file. The reason behind this gap is that most

(there is no real standard) fdisk programs don't use the first track for anything but the MBR. That space isn't always wasted though, some bootloaders (Eg GRUB) use it to store parts of their program.

Note: If you aren't using the suggested geometry then you'll have to calculate this for yourself.

We now have a device (/dev/loop0) which we can use in a similar fashion to a normal one for a partition (eg /dev/hda1).

Formatting the partition

For ext2fs, use:

```
mke2fs -b1024 /dev/loop0 #blocks
```

Explanation:

mke2fs	Create an ext2 filesytem
-b1024	Use block size of 1024
/dev/loop0	Device to make the filesystem on (Here /dev/loop0 is our 'partition')
#blocks	Remember I said to note the number of blocks from the fdisk section? This is why.

This gives us a clean ext2 formatted partition.

Note: mke2fs is smart enough to figure out block size and #blocks for itself, but if you ever want to use multiple partitions you'll need to know how to use those values.

For FAT32, use:

```
mkdosfs -F32 /dev/loop0 #blocks
```

Explanation:

mkdosfs	Create a DOS filesystem (This may be absent on some Linux systems, search for the dosfstools package if it is)
-F32	FAT 32 allocation tables (It should be obvious how to use FAT12/FAT16)
/dev/loop0	Same as for the ext2fs version

#blocks	Same as for the ext2fs version

This gives us a clean FAT32 formatted partition (Ignore the floppy warning).

Note: The reason for #blocks is the same as for ext2fs, ie possible multiple partitions.

Mount Partition

You should now be able to mount the partition (Because it is still setup on the loopback device).

Command:

```
mount -text2 /dev/loop0 /mnt/wherever
```

or:

```
mount -tvfat /dev/loop0 /mnt/wherever
```

Explanation:

mount	Linux command to mount a filesystem
-text2 / -tvfat	Filesystem being used, Linux can usually figure this out on its own.
/dev/loop0	The device representing our partition
/mnt/wherever	A directory to mount the partition on.

This should leave you with a nicely mounted partition. If you run df -Th you should end up with a line similar to:

```
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/loop0    vfat    492M  4.0K  492M   1% /mnt/wherever
```

...or for ext2fs:

```
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/loop0    ext2    477M   13K  452M   1% /mnt/wherever
```

(Yup, these are for the same disk image. By default ext2fs reserves/uses quite a bit of space even empty.)

Unmount, Detach

Ok, unmount the partition and detach the loopback device.

Command:

```
umount /dev/loop0
losetup -d /dev/loop0
```

Explanation:

umount	Linux command to unmount a filesystem.
/dev/loop0	The device that was mounted

Making it Easier

One final thing to do, which is to simplify mounting and unmounting that partition.

Mounting:

```
mount -text2 -oloop=/dev/loop0,offset=32256 /path/to/c.img /mnt/wherever
```

Unmounting:

```
umount /path/to/c.img
```

This is essentially a combination of the losetup and mount commands we used previously when formatting the partition. If used it also means we lose access to the raw 'disk' or 'partition' through /dev/loop0.

See also http://www.pixelbeat.org/scripts/lomount.sh

Finally, if you have to mount and umount that image very frequently and you're too lazy to type the sudo password each time, just add to /etc/fstab:

```
/path/to/c.img    /mnt/wherever    ext2    user,loop    0 0
```

now you can just call:

```
mount /mnt/wherever
umount /mnt/wherever
```

The End

That's it, you now know how to handle hard disk images under Linux. Whilst mounted you can use it in exactly the same way you use a normal disk partition. Multiple partitions are an extension of this, just change the offset of the losetup command according to the partition you want to use (And format using the correct number of blocks).

Things to remember:

- losetup type command will give you the equivalent of a raw disk device (Eg /dev/hda)
- losetup -o type command will give you the equivalent of a raw partition device (Eg /dev/hda1)

Don't forget to flush the filesystem buffers when manipulating with files on mounted disk image. On a Unix-like system, this can be simply done by executing the `sync` program in your shell.

Loopback Device under FreeBSD

FreeBSD 4.x uses vnconfig FreeBSD 5.x uses mdconfig

First, use DD to create an empty floppy image (1.44mb in size)

FreeBSD 4.x

```
dd if=/dev/zero of=floppy.img bs=512 count=2880
vnconfig vn0 floppy.img
newfs_msdos -f 1440 /dev/vn0
mount -t msdosfs /dev/vn0 /mnt/myfloppy
```

To shut and image down, unmount and unconfigure it.

```
umount /mnt/myfloppy
vnconfig -c /dev/vn0
```

FreeBSD 5.x

Memdisks are allocated dynamically, and the name is displayed after the mdconfig command. This assumes that "md0" is printed.)

To mount:

```
dd if=/dev/zero of=floppy.img bs=512 count=2880
mdconfig -a -t vnode -f floppy.img
newfs_msdos -f 1440 /dev/md0
mount -t msdosfs /dev/md0 /mnt/myfloppy
```

To unmount:

```
umount /mnt/myfloppy
mdconfig -d -u md0
```

Loopback Device under OpenBSD

OpenBSD has used vnconfig(8) (http://www.openbsd.org/cgi-bin /man.cgi?query=vnconfig&apropos=0&sektion=0& manpath=OpenBSD+Current&arch=i386&format=html) since version 2.2 (perhaps earlier..).

As root or using su/sudo, Here is an example scenario for configuring a vnode pseudo disk device.

Creating the floppy.img file using dd:

```
dd if=/dev/zero of=/path/to/floppy.img bs=512 count=2880
```

Configuring the vnd0 device:

```
vnconfig vnd0 /path/to/floppy.img
```

Listing configured devices:

```
vnconfig -l

Output:
   vnd0: covering floppy.img on wd0a, inode 270473
   vnd1: not in use
   vnd2: not in use
   vnd3: not in use
```

Creating a FAT12 file system and then mounting the device:

```
newfs_msdos -F 12 -f 1440 /dev/rvnd0c
mount -t msdos /dev/vnd0i /mnt/floppy
```

Removing the device mount and uninstalling the vnd0 device:

```
umount /mnt/floppy
vnconfig -u vnd0
```

More Information: vnd(4) (http://www.openbsd.org/cgi-bin/man.cgi?query=vnd&apropos=0&sektion=4&manpath=OpenBSD+Current&arch=i386&format=html) / vnconfig(8) (http://www.openbsd.org/cgi-bin/man.cgi?query=vnconfig&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html)

Retrieved from "http://wiki.osdev.org/index.php?title=Loopback_Device&oldid=16999"
Category: Disk Image Utilities

- This page was last modified on 8 November 2014, at 06:45.
- This page has been accessed 95,345 times.