

Mounting disks with Linux's loopback device

Jason Boxman

<jasonb@edseek.com>

Copyright © 2003-2004 Jason Boxman

Revision History	
Revision 0.9	20030412
Initial draft	
Revision 0.95	20030413
Enhanced loopback device and dd extraction sections	
Revision 1.0	20030419
Finished enhanced loopback section	
Revision 1.1	20030609
Mentioned the enhanced loopback patch applies to 2.4.20 cleanly now	
Revision 1.2	20040213
Converted to DocBook XML format	
Revision 1.2	20040213
Converted to DocBook XML format	
Revision 1.21	20040824
Explained adding one to sector count and 2.4 kernel bug	
Revision 1.22	20050129
Update for util-linux versions >= 2.12b	

Abstract

Discussion of mounting partitions within disk images under Linux using the loop back device.

Table of Contents

[Obtaining a Disk Image](#)[Verifying the Sanity of Your Image](#)[Accessing Specific Partitions in the Image](#)[Links and Useful Resources](#)

Disk space. There's never enough. Whilst preping my [Inspiron 3800](#) for its new 20GB Toshiba 4500 RPM disk I thought I'd play around some with disk imaging. Playing with partition images is boring, so let's spice it up!

Obtaining a Disk Image

To start, you will want an exact image of a disk; Preferably one with filesystems you have support available for in your kernel, but any will do. As always, **dd** is your friend. To obtain my disk image, I simply issued:

```
rachael:# dd if=/dev/hda of=/mnt/nebula/hda_dd.image
4757130+0 records in
4757130+0 records out
```

You can't simply mount a disk with the loopback device, however. You need some additional information. You will want to fetch a copy of the partition table, including the all important cylinder number we will use later. Invoke the magic of **fdisk**:

```
rachael:/home/jasonb# fdisk -l
```

```
Disk /dev/hda: 4871 MB, 4871301120 bytes
255 heads, 63 sectors/track, 592 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1	* 1	463	3719016	7	HPFS/NTFS	
/dev/hda2	464	592	1036192+	5	Extended	
/dev/hda5	464	479	128488+	82	Linux swap	
/dev/hda6	480	592	907641	83	Linux	

Later, you can use this information to verify your image is sane.

Verifying the Sanity of Your Image

fdisk is quite effective for this task, too. You will need the cylinder number you obtained earlier either from **fdisk**, as shown above, or via some other means. (The 'C' option to **fdisk** is relatively recent. v2.11z has it; v2.11n that shipped with RedHat 7.3 does not. You can specify this from within **fdisk** by loading the image and using the e'x'pert mode and specifying the 'c' option from there.)

```
faith:/home/jasonb# fdisk -C 592 /nebula/hda_dd.image
```

```
Command (m for help): p
```

```
Disk /nebula/hda_dd.image: 0 MB, 0 bytes
255 heads, 63 sectors/track, 592 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/nebula/hda_dd.image1	*	1	463	3719016	7	HPFS/NTFS
/nebula/hda_dd.image2		464	592	1036192+	5	Extended
/nebula/hda_dd.image5		464	479	128488+	82	Linux swap
/nebula/hda_dd.image6		480	592	907641	83	Linux

Looks familiar, no? If all went well, it should be identical to the image yanked from the original disk.

Accessing Specific Partitions in the Image

Now, the fun begins. There are three ways to mount partitions from the image. You can simply use the stock kernel's loopback device, an enhanced loopback device offered by NASA, or extract the partition from the image and mount that directly with the loopback device. In all instances, the loopback device is the final destination. The journey varies with each, however. Let's look at the former most approach first.

The simplest method, you mount the partition of your choice from within the image. You will need to specify an offset for the loopback device into the image file. You can obtain this number by running **fdisk** against the image to obtain the starting and ending sectors for each partition. (Again, the -C option is only available in very recent versions of **fdisk**, like 2.11z.)

```
faith:/home/jasonb# fdisk -l -u -C 592 /nebula/hda_dd.image
```

```
Disk /nebula/hda_dd.image: 0 MB, 0 bytes
255 heads, 63 sectors/track, 592 cylinders, total 0 sectors
Units = sectors of 1 * 512 = 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/nebula/hda_dd.image1	*63	7438094	3719016	7	HPFS/NTFS	
/nebula/hda_dd.image2		7438095	9510479	1036192+	5	Extended
/nebula/hda_dd.image5		7438158	7695134	128488+	82	Linux
swap						
/nebula/hda_dd.image6		7695198	9510479	907641	83	Linux

The offset must be specified in bytes, so now you must take the starting offset, in this instance 63, and multiply it by 512 bytes. From this we obtain 32256. (This assumes 63 sectors per track and 512 bytes per sector.) The file system type in this case is NTFS, so let us mount this partition from within the image using the usual loopback method.

```
faith:/usr/src# mount -o loop,offset=32256 \
  -t ntfs /nebula/hda_dd.image /mnt
faith:/usr/src# ls /mnt
AUTOEXEC.BAT
boot.ini
CONFIG.SYS
Corel
Documents and Settings
IO.SYS
MSDOS.SYS
NTDETECT.COM
ntldr
PUTTY.RND
Program Files
pagefile.sys
RECYCLER
System Volume Information
WINNT
```

If you are using util-linux prior to version 2.12b, specifying an offset that required more than 32-bits was not possible. If you have util-linux 2.12b or newer, you can safely skip the next few sections. (You may still wish to extract individual partitions from your disk image using **dd** discussed at the end of this guide.)

Attempting to mount my ext3 partition near the end of the disk with a 2.11 version of util-linux yields (7695198 * 512 = 3939941376):

```
faith:/usr/src# mount -o loop,offset=3939941376 \
  -t ext3 /nebula/hda_dd.image /mnt
mount: wrong fs type, bad option, bad superblock on /dev/loop0,
```

or too many mounted file systems

Fortunately, we aren't done yet. The second method utilizes a loopback device designed to mount partitions within the image without an offset limitation. In fact, no offset need be specified at all.

You will need to patch your kernel to use the enhanced loopback device. This patch alters the way the loopback device works. You will no longer be able to mount partitions via the loopback device beyond `/dev/loop0`. If you use `/dev/loop[1-7]` this could be a show stopper for you; Check out the last method.

The patch is currently available against 2.4.20 and 2.4.21 prepatch 4. You will need to fetch the [patch](#) from NASA HQ's public FTP server. It's the `enhanced_loop-x.x-linux-2.4.x-xfs.patch` file located there. You can also fetch the XFS patch for 2.4.21-pre4 and the 2.4.21-pre4 patch itself as of this writing. I used 2.4.21-pre4 with Alan Cox's `-ac7`. For convenience, a patched kernel ready for compiling is also [available](#).

```
faith:/usr/src/linux-2.4.20# patch \
  -p1 < ../enhanced_loop-0.2-linux-2.4.21-pre4-xfs.patch
patching file drivers/block/loop.c
patching file Makefile
Hunk #1 FAILED at 1.
1 out of 1 hunk FAILED -- saving rejects to file Makefile.rej
```

Don't worry about the Makefile reject; It's just the `EXTRA_VERSION` variable. (That's because I used `-ac7`.)

Now, recompile your kernel in the usual way (I use Debian GNU/Linux's `make-kpkg` command) and make sure you enable the loopback device if it isn't already. When that task is complete, reboot with your shiny new kernel.

To accomodate the enhanced loopback device, some new entries need to be created in `/dev`. A script named **createdev** is available to handle that task for you, and it can be run at start up if you're running `devfs` to recreate the entries for you at boot. You can fetch the script from [NASA HQ](#). You may need to comment out the sourcing of the RedHat functions within the script if you aren't on a RedHat based distribution, like Debian. By default the script will create enough entries in `/dev` for a fifteen disks with up to fifteen partitions. You can adjust that to your requirements within the script. It will blow away any existing `/dev` entries it has added if you change configurations, so you need not tend to them yourself.

```
faith:/nebula# vi createdev
faith:/nebula# bash createdev start
faith:/nebula#
```

Once you've run the script, you should find a entries like the following in your /dev directory:

```
faith:/home/jasonb# ls /dev/loop[a-zA-Z]*
/dev/loopa      /dev/loopd12  /dev/loopg2
/dev/loopj6     /dev/loopn    /dev/loopa1
/dev/loopd13   /dev/loopg3   /dev/loopj7
/dev/loopn1    /dev/loopa10  /dev/loopd14
/dev/loopg4    /dev/loopj8   /dev/loopn10
/dev/loopa11   /dev/loopd15  /dev/loopg5
/dev/loopj9    /dev/loopn11
```

With the kernel up and running, you also need to acquire a modified copy of **losetup**, the loopback setup program. If you're running an RPM based distribution, you're in luck. You can fetch the [modified losetup](#) by making another journey to NASA HQ's FTP server. Rebuild it with **rpmbuild -bb** and install. If you're running Debian GNU/Linux, as I am, you can install the **rpm** package with the usual **apt-get** command. Then, you could either build the RPM package and use **alien** to convert it to a Debian package or use **rpm2cpio** to create a cpio archive of the RPM. For the latter, you can extract the source from the resultant cpio archive and compile:

```
faith:/usr/src# rpm2cpio loop-utils-0.0.1-1.src.rpm > loop-utils.cpio
```

```
faith:/usr/src# cpio -i < loop-utils.cpio
39 blocks
```

```
faith:/usr/src# tar -zxvf loop-utils-0.0.1.tar.gz
loop-utils-0.0.1/
loop-utils-0.0.1/COPYING
loop-utils-0.0.1/Makefile
loop-utils-0.0.1/loimginfo.c
loop-utils-0.0.1/lomount.c
loop-utils-0.0.1/lomount.h
loop-utils-0.0.1/loop.h
loop-utils-0.0.1/loop.sgml
loop-utils-0.0.1/losetgeo.c
loop-utils-0.0.1/lotest.c
loop-utils-0.0.1/nls.h
loop-utils-0.0.1/partinfo.c
```

```
faith:/usr/src# cd loop-utils-0.0.1
```

You may wish to edit the Makefile, which sticks things in /usr by default. I changed it to /usr/local and added \${prefix} as the path for the sbin_prefix variable. It originally had no value at all, but is later used when installing the **losetup** binary, which would instead end up in your /sbin directory. Oops.

```

faith:/usr/src/loop-utils-0.0.1# make
gcc -Wall -Wstrict-prototypes -O6 -DVERSION='"0.3.9"' \
  -DLOG2_NR_PARTITION='4' -c -o losetgeo.o losetgeo.c
gcc losetgeo.o -o losetgeo
gcc -Wall -Wstrict-prototypes -O6 -DVERSION='"0.3.9"' \
  -DLOG2_NR_PARTITION='4' -c -o loimginfo.o loimginfo.c
gcc loimginfo.o -o loimginfo
gcc -Wall -Wstrict-prototypes -O6 -DVERSION='"0.3.9"' \
  -DLOG2_NR_PARTITION='4' -c -o partinfo.o partinfo.c
gcc partinfo.o -o partinfo
gcc -DMAIN -D_FILE_OFFSET_BITS=64 lomount.c -o losetup.o
<warnings...>
ld losetup.o -o losetup
gcc -Wall -Wstrict-prototypes -O6 -DVERSION='"0.3.9"' \
  -DLOG2_NR_PARTITION='4' -c -o lotest.o lotest.c
gcc lotest.o -o lotest
sgml2latex loop.sgml
Processing file loop.sgml
sgml2html -s 0 loop.sgml
Processing file loop.sgml
sgml2info loop.sgml
Processing file loop.sgml
echo "START-INFO-DIR-ENTRY" > loop.info.2
echo "* Loop: (loop). Block device loopback package." \
  >> loop.info.2
echo "END-INFO-DIR-ENTRY" >> loop.info.2
cat loop.info.2 loop.info > loop.info.3
rm loop.info.2
mv loop.info.3 loop.info

```

Now, let's test drive our new loopback device.

```

faith:/nebula# /usr/local/sbin/losetup -d /dev/loopa
faith:/nebula# /usr/local/sbin/losetup /dev/loopa hda_dd.image
faith:/nebula# mount -t ntfs /dev/loopa1 /mnt
faith:/nebula# ls /mnt
AUTOEXEC.BAT

```

```

boot.ini
CONFIG.SYS
Corel
Documents and Settings
IO.SYS
MSDOS.SYS
NTDETECT.COM
ntldr
PUTTY.RND
Program Files
pagefile.sys
RECYCLER
System Volume Information
WINNT
faith:/nebula# umount /mnt
faith:/nebula# /usr/local/sbin/losetup -d /dev/loopa
faith:/nebula#

```

Nifty, eh?

Last, you can use **dd** to extract the partition of interest manually and then mount it via loopback. Again, the assumption of 512 bytes per sector is assumed here. As explained in Brian Carrier's [March 15th Sleuth Kit Informer](#) column, [Splitting The Disk](#), we can pass **dd** the starting sector of the partition in question and calculate the size and allow it to extract it for us. For example, let's extract my ext3 partition, then mount it on loopback.

We pass **dd** bytes at a time size (bs option) of 512. Next, we pass it the starting sector of my ext3 partition from the fdisk output above, 7695198, as the number of blocks to skip ahead in the image. Last, we calculate the size as explained in the Sleuth Kit Informer above by taking the starting and ending sectors of the partition, subtracting them, then adding one ($9510479 - 7695198 + 1 = 1815282$).

Ronald Woelfel raised an interesting question about a missing sector on partitions with an odd number of sectors, which was explained thusly by Brian Carrier of Sleuth Kit fame: *"The reason that you noticing the difference is likely because your linux system has the 2.4 kernel, which has a bug when accessing disk or partition devices. If a partition or disk has an odd number of sectors, the last sector is not read."*

```

faith:/home/jasonb# dd if=/nebula/hda_dd.image of=/nebula/test.image \
  bs=512 skip=7695198 count=1815282
1815282+0 records in
1815282+0 records out

```

Once **dd** completes, you can mount the image as you normally would:

```
faith:/home/jasonb# mount -o loop -t ext3 /nebula/test.image /mnt
faith:/home/jasonb# ls /mnt
bin      dev      home     lib  opt     sbin   var
boot    etc      import  lost+found  proc  tmp     vmlinuz
cdrom   floppy  initrd  mnt  root   usr     vmlinuz.old
faith:/home/jasonb# umount /mnt
```

Enjoy!

Links and Useful Resources

- [Mounting disk images](#) on the loopback device
- [C/H/S information](#) for PC hardware
- Security Focus mailing list thread on [dd and mounting disk images](#) that inspired much of this article
- How to use dd to extract [individual partitions](#) from a disk image