How to install GRUB into a disk image

Disk and filesystem images

A disk image is an ordinary file that holds the exact contents and layout of a physical disk or any other block device. Similar a filesystem image is an exact copy of all the blocks that belong to a filesystem. Working with image files allows the developer to do much of the work without access to the target storage hardware, which is typically more expensive, slower, harder to access, and limited in numbers.

Many tools that work on block devices also work on image files. For example, an image file can be partitioned using `sfdisk`, and an image file can be formatted with a filesystem using `mkfs`.

Creating a block device image

To create a block device image file for a physical disk with a size of `LBA` logically addressable blocks of 512 bytes each (often called a sector), use `dd` as follows:

```
dd if=/dev/zero of=disk.img bs=512 count=${LBA}
```

If your host filesystem supports sparse files[1] (files containing large chunks of zero data that do not occupy disk space), this is a faster and more space efficient alternative to the above:

```
dd if=/dev/zero of=disk.img bs=512 count=0 seek=${LBA}
```

Creating partitions

If you are targetting a IDE hard disk drive device on a x86 platform, or a device that behaves as such, we must take care of the Master Boot Record (MBR), the partition table and the partitions themselves. The first sector contains the MBR along with the partition table. Conventionally, the rest of the first track (this is the track on the first head of the first cylinder) is not used as partition space; the first partition starts at the second head of the first cylinder. Any subsequent partitions start on cylinder boundaries.

Suppose you want to partition the block device with two partitions laid out in a conventional way respecting the physical block device geometry. The geometry is assumed to be (cylinders, heads, sectors) = (C, H, S) here. The last partition is chosen to occupy the last L cylinders. With some math we end up with these offsets and sizes:

Disk Layout

disk element	**offset (sectors)**	**size (sectors)**
MBR	`0`	`1`
unused	`1`	`(S-1)`
partition 1	`S`	`((C-L)*H*S) - S`
partition 2	`(C-L)*H*S`	`L*H*S`

1 ext3 supports sparse files.

The `sfdisk` tool can be used to partition the disk in a scripted manner as follows:

```
sudo /sbin/sfdisk -C$C -H$H -S$S -uS -f -L --no-reread disk.img <<EOF
$S,$(($(($C-$L)*$H*$S - $S),83,*
$(($(($C-$L)*$H*$S),$(($L*$H*$S),83
EOF
```

We can create filesystems on the partitions inside the disk image directly, but this needs creating a loop device, as well as recent version of the loop device setup tool `losetup` that understands the `size` parameter. An easier alternative is to create partition images independent of the disk image, create filesystems on these, populate these and only as the final step before deployment, move the partition images inside the disk image. We use the latter approach:

```
dd if=/dev/zero of=part1.img bs=512 count=0 seek=$(($(($C-$L))*$H*$S - $S))
dd if=/dev/zero of=part2.img bs=512 count=0 seek=$(($L*$H*$S))
```

Creating filesystems

After creating the partition images, filesystems can be created on them as follows:

```
sudo /sbin/mke2fs -j -L"/" -F part1.img
sudo /sbin/mke2fs -j -L"etc" -F part2.img
```

For filesystems that are mounted read-only, disable the period filesystem check:

```
sudo /sbin/tune2fs -c0 -f part1.img
sudo /sbin/tune2fs -c0 -f part2.img
```

The partitions are now available to be populated with the intended content; executables, libraries etc. In the next step, we install GRUB onto it.

Mounting a filesystem image file

In order to populate a filesystem inside the image file, that filesystem must be mounted first. For the purpose of mounting these filesystems not associated with block devices, there exist loop devices. Using a loop device we can present our image file as a block device to the filesystem subsystem.

The resulting virtual block device is presented as a `/dev/loop0` or `/dev/loop/0` entry.

Installing GRUB into a filesystem image file

If you want to install a bootloader that uses a filesystem on disk such as GRUB, this can most easily be done now, while the seperate filesystem images are still available. By first mounting the partition's filesystem we can copy the necessary GRUB files onto it afterwards:

```
mkdir -p /tmp/mnt/part1
sudo mount -o loop /tmp/part1.img /tmp/mnt/part1
```

Now, create a GRUB directory and copy the necessary stage 1.5 and stage 2 files onto it:

```
mkdir -p /tmp/mnt/part1/boot/grub
sudo cp /usr/share/lib/grub/i386-pc/* /tmp/mnt/part1/boot/grub/
sudo umount /tmp/mnt/part1
```

We still need to install the stage 1 itself. Installing the GRUB bootloader into a filesystem image file on the build host needs a bit of trickery to make GRUB believe it is installing itself onto a real block device. We setup a loop device for this:

```
sudo /sbin/losetup /dev/loop5 disk.img
```

Also, GRUB needs access to the filesystem where the stage 1.5 and stage 2 files are.

```
sudo mount -o loop,offset=$(($S*512)) disk.img /tmp/mnt/part1
```

We now trick GRUB into believing that the loop device is a hard disk device:

```
echo “(hd0) /dev/loop5” >/tmp/device.map
```

Finally, we execute GRUB to install itself on this device (image, but GRUB doesn’t know that).

```
sudo /tmp/grub-resumo/sbin/grub --batch --no-floppy --device-map=/tmp/device.map <<EOF
device (hd0) /dev/loop5
geometry (hd0) ${C} ${H} ${S}
root (hd0,0)
setup --stage2=/tmp/mnt/part1/boot/grub/stage2 (hd0)
quit
EOF
```

Unmount the filesystem and drop the looped block device image:

```
sudo umount /tmp/mnt/part1
sudo /sbin/losetup -d /dev/loop5
```

Moving filesystems into a block device image

Earlier we have created a disk image on which we assembled a MBR in its first sector, containing a partition table for two partitions. For the partitions themselves we created seperate images and layed out a filesystem on each. Optionally, you added a bootloader or files to the filesystems. We now move these filesystem images onto the disk image, carefully respecting the offsets we have chosen earlier:

```
dd if=part1.img of=disk.img bs=512 seek=$S
dd if=part2.img of=disk.img bs=512 seek=$(($(($C-$L))*$H*$S))
```