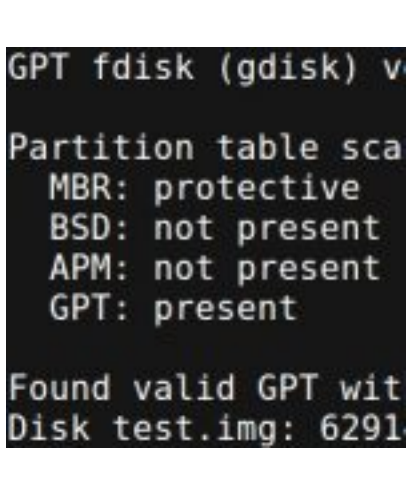
May 28 / 2017 | 6 | Bash, BIOS, GPT, Linux, Scripts, UEFI



LINUX

Creating a BIOS/GPT and UEFI/GPT Grub-bootable Linux system

Good old Master Boot Record (MBR) unfortunately cannot address anything beyond 2TB, so partitioning large disks and making them bootable is impossible using MBR. The GUID Partition Table (GPT) solves this problem: It supports disks up to 16EB. However, installing grub does not work without a special BIOS boot partition. If you also want to support booting the same system via UEFI, another partition, the EFI System Partition (ESP), is necessary.

This should post shows you how to partition a disk with GPT and make a **bootable Linux system via BIOS/Legacy and UEFI**.

Content

Requirements & Assumptions

For this post, I'll assume that you are running everything on a Debian-based system, and all commands shown must be run as root.

1. Create a minimal Linux (optional)

For this example, we'll use a small Linux install that we can create with the following commands. This will bootstrap (`debootstrap`) a minimal Ubuntu 16.04 system to the `chroot/` folder which we will later use to boot our system. To create a bootable system, we need a kernel and the bootloader modules and files, so let's install `linux-image-generic` and `grub-pc`:

Bootstrapping a new Ubuntu 16.04 system (with a kernel and grub) — Shell

```
# Bootstrap minimal system
debootstrap --variant=minbase xenial chroot

# Install kernel and grub
for d in dev sys proc; do
  mount --bind /$d chroot/$d
done

DEBIAN_FRONTEND=noninteractive chroot chroot apt-get install linux-image-generic grub-pc -y --forc
umount chroot/{dev,proc,sys}
```

Alternatively, you can of course use any Linux root file system, provided that it has a kernel and grub. You could, for instance, just copy your own root file system to the `chroot/` folder.

2. Creating a GPT with a BIOS boot partition and an EFI System Partition

Now that we have the Linux root file system in the `chroot/` folder, we'll create a sparse file that represents our disk. You may of course do all this with a proper HDD/SSD, i.e. with `/dev/sdX`, but for testing things using a raw disk file is much easier. The following snippet will create 3 partitions:

- A 1 MB BIOS boot partition (of type `ef02`) that Grub will use to store its core image.
- A 100 MB EFI System Partition (of type `ef00`) formatted as FAT32 in which we will store the EFI boot image
- And a partition for our root file system (in our example formatted with ext4)

Partitioning a disk/file with GPT to boot via UEFI and BIOS — Shell

```
# Create sparse file to represent our disk
truncate --size 30G test.img

# Create partition layout
sgdisk --clear \
  --new 1::+1M --typecode=1:ef02 --change-name=1:'BIOS boot partition' \
  --new 2::+100M --typecode=2:ef00 --change-name=2:'EFI System' \
  --new 3::-0 --typecode=3:8300 --change-name=3:'Linux root filesystem' \
  test.img
```

Once this is done, you can list the partitions with `gdisk -l test.img`:

Listing the partitions we just created — Shell

```
# Now list the partitions
gdisk -l test.img
GPT fdisk (gdisk) version 1.0.1

# ...

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048            4095   1024.0 KiB  EF02  BIOS boot partition
   2            4096          208895   100.0 MiB   EF00  EFI System
   3          208896        62914526   29.9 GiB    8300  Linux root filesystem
```

After the partitions are created, the EFI and root partition need to be formatted:

Formatting the root partition and the EFI System Partition — Shell

```
# Loop sparse file
LOOPDEV=$(losetup --find --show test.img)
partprobe ${LOOPDEV}

# Create filesystems
mkfs.fat -F32 ${LOOPDEV}p2
mkfs.ext4 -F -L "demoroot" ${LOOPDEV}p3 # << Note the label 'demoroot'

# Get rid of loop device
losetup -d ${LOOPDEV}
```

Note that I named the root partition `demoroot` (the ext4 label). This will be important later in the Grub configuration.

3. Copying the root file system, installing grub into the BIOS partition

Now that the partition to be used for our root file system is formatted, let's copy our `chroot/` directory to it and then install grub to the disk with `grub-install`. Because the disk is GPT formatted, grub will use the BIOS partition to install its core image:

Shell

```
# Loop sparse file
LOOPDEV=$(losetup --find --show test.img)
partprobe ${LOOPDEV}

# Mount OS partition and copy from chroot
MOUNTDIR=$(mktemp -d -t demoXXXXXX)
mount ${LOOPDEV}p3 $MOUNTDIR
rsync -a chroot/ ${MOUNTDIR}/

# Install grub, create config
for d in dev sys proc; do mount --bind /$d ${MOUNTDIR}/$d; done
chroot ${MOUNTDIR}/ grub-install --modules="ext2 part_gpt" ${LOOPDEV}
chroot ${MOUNTDIR}/ update-grub

# Unmount OS partition
umount $MOUNTDIR/{dev,proc,sys,}
rmdir $MOUNTDIR

# Remove loop
losetup -d ${LOOPDEV}
```

After this, you should be able to boot the system via BIOS from the root GPT partition. I always do that via KVM like this:

Test booting the image via KVM — Shell

```
kvm -drive format=raw,file=test.img -serial stdio -m 4G -cpu host -smp 2
```

4. Preparing the EFI partition

After you've verified that you can boot via BIOS, let's make sure that we can also boot on UEFI systems. We formatted the EFI partition earlier. All that's remaining is to create/copy the `bootx64.efi` image file and a valid `grub.cfg` to it.

Grub can create the EFI image via the `grub-mkimage` command using the Grub modules in `/usr/lib/grub/x86_64-efi` (part of the `grub-efi-amd64-bin` package). In order to avoid having to load anything from the file system, we include all the modules in the image. If you are not booted via EFI, you may need to install the `grub-efi-amd64-bin` package.

The Grub config file in the EFI partition `grub.cfg` is rather simple: It uses `search --label demoroot` to look for the root partition and then simply includes the actual grub config file via `configfile ...`.

Let's mount the EFI partition again, create the image and the grub config:

```
# Loop sparse file
LOOPDEV=$(losetup --find --show test.img)
partprobe ${LOOPDEV}

# Mount EFI partition
MOUNTDIR=$(mktemp -d -t demoXXXXXX)
mount ${LOOPDEV}p2 $MOUNTDIR

# Create EFI boot image
apt-get install grub-efi-amd64-bin -y --force-yes

mkdir -p ${MOUNTDIR}/EFI/BOOT
grub-mkimage \
  -d /usr/lib/grub/x86_64-efi \
  -o ${MOUNTDIR}/EFI/BOOT/bootx64.efi \
  -p /efi/boot \
  -O x86_64-efi \
    fat iso9660 part_gpt part_msdos normal boot linux configfile loopback chain efifwsetup efi_gop
    efi_uga ls search search_label search_fs_uuid search_fs_file gfxterm gfxterm_background \
    gfxterm_menu test all_video loadenv exfat ext2 ntfs btrfs hfsplus udf

# Create grub config
cat <<GRUBCFG > ${MOUNTDIR}/EFI/BOOT/grub.cfg
search --label demoroot --set prefix # << Note again, this searches by the 'demoroot' label!
configfile (\$prefix)/boot/grub/grub.cfg
GRUBCFG

# Unmount and clean up
umount ${MOUNTDIR}
losetup -d ${LOOPDEV}
```

After that, the EFI partition should contain the two files `/EFI/BOOT/bootx64.efi` and `/EFI/BOOT/grub.cfg`. You can verify that like this:

```
# Loop sparse file
LOOPDEV=$(losetup --find --show test.img)
partprobe ${LOOPDEV}

# Mount EFI partition
MOUNTDIR=$(mktemp -d -t demoXXXXXX)
mount ${LOOPDEV}p2 $MOUNTDIR

# List things
(cd $MOUNTDIR; find .)

# Should output this:
# ./EFI
# ./EFI/BOOT
# ./EFI/BOOT/bootx64.efi
# ./EFI/BOOT/grub.cfg

# Unmount and clean up
umount ${MOUNTDIR}
losetup -d ${LOOPDEV}
rmdir ${MOUNTDIR}
```

That's it. You should now be able to boot this image via UEFI. I always test this with KVM and OVMF:

Booting the image via UEFI/EFI — Shell

```
apt-get install ovmf -y --force-yes
kvm --bios /usr/share/qemu/OVMF.fd -net none -drive format=raw,file=test.img -serial stdio -m 4G -
```

5. One script to do it all

If you are an impatient man/woman, then you may just want to run the following script to do all of this in one go. You'll of course have to adjust it to your needs for production use.

The script can be called in two ways:

Examples on how to use the mkbiosefi script — Shell

```
# Makes a BIOS/UEFI bootable test.img using "demoroot" as OS partition label
# and create a minimal chroot via debootstrap in the "chroot/" folder to do so.
./mkbiosefi test.img demoroot

# Makes a BIOS/UEFI bootable test.img using "demoroot" as OS partition label,
# but re-uses the existing "chroot/" folder (this is faster!)
./mkbiosefi test.img demoroot chroot/

# If you want to use your own image file, you may create it beforehand and loop it
truncate -s 40G my.img
myloop=$(losetup --find --show my.img)
./mkbiosefi $myloop demo chroot/

# Or use a raw disk, of course
./mkbiosefi /dev/sdX demo chroot/
```

Here's the script:

mkbiosefi script that can be used to create a BIOS/UEFI bootable disk/image — Shell

```
#!/bin/bash

DISK=$1
BOOTLABEL=$2
ROOTDIR=$3

if [ -z "$DISK" -o -z "$BOOTLABEL" ]; then
  echo "Syntax: $0 <image|disk> <root-label> [<chroot-dir>]"
  exit 1
fi

if [ "$UID" != "0" ]; then
  echo "Must be root."
  exit 1
fi

# Exit on errors
set -xe

# Install dependencies
apt-get install -y --force-yes \
  debootstrap \
  gdisk \
  rsync \
  grub-efi-amd64-bin \
  e2fsprogs

# Create chroot (if requested)
if [ -z "$ROOTDIR" ]; then
  ROOTDIR=chroot/

  # Bootstrap minimal system
  debootstrap --variant=minbase xenial chroot
```

ABOUT ME

Philipp C. Heckel
Principal Engineer and Team Lead at Datto

TAGS

Bash Cloud Computing Code Snippets Cron Debian Distributed Systems Hacking Java Linux Mail Mobile PHP Scripts Security SSL Syncany Synchronization TLS Ubuntu Virtualization

POPULAR POSTS

- Roundcube login via PHP script (209)
- Send WhatsApp messages via PHP using WhatsAPI (160)
- How To: Use mitmproxy to read and modify HTTPS traffic (66)
- How To: Sniff the WhatsApp password from your Android phone or iPhone (60)
- Use SSLsplit to transparently sniff TLS/SSL connections – including non-HTTP(S) protocols (56)

ARCHIVES

Aug '18	Jun '14	Feb '11
Mar '18	May '14	Jan '11
May '17	Mar '14	Aug '10
Jan '17	Feb '14	May '10
Dec '16	Oct '13	Jan '10
Jan '16	Aug '13	Sep '09
Dec '15	Jul '13	Aug '09
Oct '15	Jun '13	Apr '09
Aug '15	May '13	Mar '09
May '15	Mar '13	Nov '08
Mar '15	Jan '13	Oct '08
Jan '15	Dec '12	Sep '08
Oct '14	Jun '12	Jul '08
Aug '14	Mar '11	May '08

6 COMMENTS

Mark Lomas | August 6th, 2017

Hey Philipp,

This is a great article, I found it super useful. Thanks :-)

FYI, I had to apt-get the 'udev' package because partprobe complained about 'udevadm' being missing and I needed to apt-get 'parted' so that partprobe was installed. I also added '–arch=amd64' to debootstrap because I wanted a 64bit distro, it seemed to default to 'i386' which I assumed mean't 32bit.

My next step is to investigate the 'live__boot' package to see if I can turn the distro into one that can install itself somewhere when booted and I'd like to find a way to squeeze the image size down from 30GB to something much smaller. Any ideas?

Thanks once again for the article,

Best regards
Mark

Vincent Perrier | February 4th, 2018

Is it possible to have a working system on a sparse file with only uefi, without any bios partition? Thanks a lot for your script!

Philipp C. Heckel | February 4th, 2018

The BIOS partition is used by grub only for BIOS boot. If you want UEFI only all you need is the EFI partition. The EFI grub.cfg references the grub.cfg on the root partition, not the boot partition, so you should be good there. You'd have to adjust the script not to create a BIOS partition, obviously.

Oscar | March 18th, 2018

mkosi (https://github.com/systemd/mkosi) does the same more or less in a automated manner, but only for UEFI systems.

ewe2 | June 9th, 2018

You probably need to install grml-debootstrap to simplify a lot of this, for Debian Stretch I had to change a lot and it's a lot easier to have grml-debootstrap generate the correct locales for instance.

Gil Montag | July 1st, 2018

Hi
1. Will this technique be good also for buildroot generated rootfs and kernel?
2. I tried the method and the uefi boot didn't work for me. It reverted to normal boot after few error messages of not being able to uefi boot from cdrom and floppy…???
Thanks
Gil

LEAVE A COMMENT

I'd very much like to hear what you think of this post. Feel free to leave a comment. I usually respond within a day or two, sometimes even faster. I will not share or publish your e-mail address anywhere.

NAME*

EMAIL*

WEBSITE

COMMENT*

COMMENT