Disk Setup On FreeBSD

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
	2016-05-23		WB

Contents

Last updated 2016-05-23

Available in HTML or PDF. Links to all my articles here. Created with AsciiDoc.

Setting up disks with FreeBSD.

1 The New Standard: GPT

GPT is the newer and more versatile partition method. Use it unless compatibility with the old standard MBR is required.

The disk device being set up here is *da0*.

Create a GPT partition scheme on */dev/da0*. */dev/* is implied:

```
# gpart create -s gpt da0
da0 created
```

Create a boot partition to hold the loader, size of 512K. Give it a GPT label of *gpboot*, which will show up in */dev/gpt* when the device is discovered:

```
# gpart add -t freebsd-boot -l gpboot -b 40 -s 512K da0
da0p1 added
```

Install the GPT bootcode into the boot partition:

```
# gpart bootcode -b /boot/pmbr -p /boot/gptboot -i 1 da0
bootcode written to da0
```

Create a partition for */*. It should start at the 1M boundary for proper sector alignment on 4K sector drives or SSDs. This is compatible with GPT drive layout for many other systems. Give it a GPT label of *gprootfs*.

```
# gpart add -t freebsd-ufs -l gprootfs -b 1M -s 2G da0
da0p2 added
```

Create partitions for swap, */var*, */tmp*, */usr*. Leaving the *-s* size option off the last uses the rest of the disk. As long as these are even sizes in M or G, the alignment will remain correct. Add *-a 1M* to force alignment of both starting position and size to one megabyte.

```
# gpart add -t freebsd-swap -l gpswap -s 512M da0
da0p3 added
# gpart add -t freebsd-ufs  -l gpvarfs -s 1G da0
da0p4 added
# gpart add -t freebsd-ufs  -l gptmpfs -s 256M da0
da0p5 added
# gpart add -t freebsd-ufs  -l gpusrfs -a 1M da0
da0p6 added
# gpart show -l da0
=>       34  19640813  da0  GPT  (9.4G)
         34         6       - free -  (3.0k)
         40      1024    1  gpboot  (512k)
       1064       984       - free -  (492k)
       2048   4194304    2  gprootfs  (2.0G)
    4196352   1048576    3  gpswap  (512M)
    5244928   2097152    4  gpvarfs  (1.0G)
    7342080    524288    5  gptmpfs  (256M)
    7866368  11773952    6  gpusrfs  (5.6G)
```

Tip

If there's a need or desire to redo this partition layout, `gpart (8)` won't destroy the "geom" until all of the partitions in it have been deleted:

```
# gpart delete -i 1 da0
da0p1 deleted
# gpart delete -i 2 da0
da0p2 deleted
# gpart delete -i 3 da0
da0p3 deleted
# gpart delete -i 4 da0
da0p4 deleted
# gpart delete -i 5 da0
da0p5 deleted
# gpart delete -i 6 da0
da0p6 deleted
# gpart destroy da0
```

Later versions of *gpart(8)* have a *-F* (force) option for destroy that makes things quicker but sort of blunt and stabby at the same time:

```
# gpart destroy -F da0
da0 destroyed
```

Format the new filesystems, enabling soft updates for performance. Filesystem labels can be added here, but probably should not be the same as the GPT labels already assigned.

```
# newfs -U /dev/gpt/gprootfs
# newfs -U /dev/gpt/gpvarfs
# newfs -U /dev/gpt/gptmpfs
# newfs -U /dev/gpt/gpusrfs
```

See http://www.freebsd.org/doc/en_US.ISO8859-1/books/faq/disks.html#SAFE-SOFTUPDATES for more information on using soft updates on the root filesystem.

Restore data. Labels are used in the `mount` command here because they're easier to identify, but device names like */dev/da0p2* will work.

```
# mount /dev/gpt/gprootfs /mnt
# (cd /mnt && gzcat root.dump.gz | restore -ruf -)
# umount /mnt
```

Repeat for */var*, */tmp*, */usr*.

Modify */etc/fstab*:

```
# Device                  Mountpoint      FStype  Options         Dump    Pass#
/dev/gpt/gpswap           none            swap    sw              0       0
/dev/gpt/gprootfs         /               ufs     rw              1       1
/dev/gpt/gptmpfs          /tmp            ufs     rw              2       2
/dev/gpt/gpusrfs          /usr            ufs     rw              2       2
/dev/gpt/gpvarfs          /var            ufs     rw              2       2
```

Done!

2 Booting with UEFI

There is nothing so bad that it cannot be made worse with added layers of encryption and security.

— Warren Block *Block's Laws (#11)*

UEFI replaces the old BIOS, and also replaces the boot method. With FreeBSD it is fairly easy to set up. The UEFI bootcode is contained on a small partition. Although it uses the *efi* partition type, it actually contains an MS-DOS FAT filesystem with a boot file. The partition is nominally sized at 800K, although a smaller one works with the current FreeBSD UEFI bootcode. The 492K unused space shown in the earlier example is large enough to hold it.

At present, most UEFI implementations are 64-bit and can only boot into 64-bit operating systems. UEFI can boot MBR or GPT partition schemes. GPT is shown here.

SecureBoot adds more cryptography to make booting less likely. At present, FreeBSD does not support this, so SecureBoot must be disabled in UEFI.

This example is almost identical to the earlier GPT example. Creating a *freebsd-boot* partition is not necessary, only the *efi* partition is needed to boot on a UEFI system. But the *freebsd-boot* partition does not take much space, and makes it possible to also boot the disk on a BIOS-based computer.

```
# gpart create -s gpt da0
da0 created
# gpart add -t freebsd-boot -l gpboot -b 40 -s 512K da0
da0p1 added
# gpart bootcode -b /boot/pmbr -p /boot/gptboot -i 1 da0
bootcode written to da0
```

Create and format a partition to hold the the small MS-DOS filesystem for UEFI bootcode.

```
# gpart add -t efi -l gpefiboot -a4k -s492k da0
da0p2 added
# newfs_msdos /dev/da0p2
```

Copy the FreeBSD */boot/boot1.efi* bootcode file into the *efi* filesystem.

```
# mount -t msdosfs /dev/da0p2 /mnt
# mkdir -p /mnt/EFI/BOOT
# cp /boot/boot1.efi /mnt/EFI/BOOT/
# umount /mnt
```

The rest of the partitions are created as shown in the GPT example above.

```
# gpart add -t freebsd-ufs -l gprootfs -b 1M -s 2G da0
da0p3 added
# gpart add -t freebsd-swap -l gpswap -s 512M da0
da0p4 added
# gpart add -t freebsd-ufs  -l gpvarfs -s 1G da0
da0p5 added
# gpart add -t freebsd-ufs  -l gptmpfs -s 256M da0
da0p6 added
# gpart add -t freebsd-ufs  -l gpusrfs -a 1M da0
da0p7 added
# gpart show -l da0
=>      34  19640813  da0  GPT  (9.4G)
        34         6       - free -  (3.0k)
        40      1024    1  gpboot  (512k)
      1064       984    2  gpefiboot (492k)
      2048   4194304    3  gprootfs  (2.0G)
   4196352   1048576    4  gpswap  (512M)
   5244928   2097152    5  gpvarfs  (1.0G)
   7342080    524288    6  gptmpfs  (256M)
   7866368  11773952    7  gpusrfs  (5.6G)
```

Done!

3 The Old Standard: MBR

Usually the GPT setup above is the better choice. This older method creates an MBR partition table, which can be useful for backwards compatibility.

Again, we're setting up *da0*.

Warning

If remnants of a GPT partitioning scheme are still present on the disk, it will be seen instead of the MBR. Destroying that old partitioning scheme before creating the new MBR is important.

```
# gpart destroy -F da0
```

Older versions of *gpart(8)* did not have *-F* to easily remove old partition schemes without deleting all partitions first. An alternate way to remove GPT tables is using *dd(1)* to overwrite the GPT primary table at the start of the disk:

```
# dd if=/dev/zero of=/dev/da0 bs=512 count=34
```

Removing the secondary or backup GPT table at the end of the disk requires a little calculation. Use *diskinfo(8)* to get the number of blocks, subtract the 34-block length of the standard table, and write from there to the end of the disk. In this example, the disk has 60030432 blocks. Subtracting 34 from that gives 60030398 as the starting location, and *dd(1)* will write from there to the end of the disk.

```
# diskinfo -v da0
        512             # sectorsize
        30735581184     # mediasize in bytes (28G)
        60030432        # mediasize in sectors
        0               # stripesize
        0               # stripeoffset
        3736            # Cylinders according to firmware.
        255             # Heads according to firmware.
        63              # Sectors according to firmware.
        560482EC2222    # Disk ident.
# echo '60030432 - 34' | bc
60030398
# dd if=/dev/zero of=/dev/da0 bs=512 seek=60030398
```

Create the MBR partitioning scheme:

```
# gpart create -s mbr da0
```

Make it bootable by installing bootcode.

```
# gpart bootcode -b /boot/mbr da0
```

Create an MBR partition. FreeBSD calls these "slices". Set it active so the system will boot from it.

```
# gpart add -t freebsd da0
# gpart set -a active -i 1 da0
```

Inside the FreeBSD slice, create a *bsdlabel* partitioning scheme. Bootcode is needed here also.

```
# gpart create -s bsd da0s1
# gpart bootcode -b /boot/boot da0s1
```

Create the FreeBSD "partitions" inside the slice.

```
# gpart add -t freebsd-ufs  -a 4k -s 2g   da0s1
# gpart add -t freebsd-swap -a 4k -s 512m da0s1
# gpart add -t freebsd-ufs  -a 4k -s 1g   da0s1
# gpart add -t freebsd-ufs  -a 4k -s 256m da0s1
# gpart add -t freebsd-ufs  -a 4k         da0s1
```

Format and label the filesystems before they are mounted, enabling soft updates for better performance:

```
# glabel label swap /dev/da0s1b
# newfs -L rootfs -U /dev/da0s1a
# newfs -L varfs  -U /dev/da0s1d
# newfs -L tmpfs  -U /dev/da0s1e
# newfs -L usrfs  -U /dev/da0s1f
```

See http://www.freebsd.org/doc/en_US.ISO8859-1/books/faq/disks.html#SAFE-SOFTUPDATES for more information on using soft updates on the root filesystem.

Restore data to the new filesystems:

```
# mount /dev/da0s1a /mnt
# gzcat root.dump.gz | (cd /mnt && restore -rf -)
# umount /mnt
```

Repeat for */var*, */tmp*, */usr*.

Modify */etc/fstab*:

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
/dev/label/swap         none            swap    sw              0       0
/dev/ufs/rootfs         /               ufs     rw              1       1
/dev/ufs/tmpfs          /tmp            ufs     rw              2       2
/dev/ufs/usrfs          /usr            ufs     rw              2       2
/dev/ufs/varfs          /var            ufs     rw              2       2
```

Done!