

In this chapter:

- *Vinum objects*
- *Creating Vinum drives*
- *Starting Vinum*
- *Configuring Vinum*
- *Vinum configuration database*
- *Installing FreeBSD on Vinum*
- *Recovering from drive failures*
- *Migrating Vinum to a new machine*
- *Things you shouldn't do with Vinum*

12

The Vinum Volume Manager

Vinum is a *Volume Manager*, a virtual disk driver that addresses these three issues:

- Disks can be too small.
- Disks can be too slow.
- Disks can be too unreliable.

From a user viewpoint, *Vinum* looks almost exactly the same as a disk, but in addition to the disks there is a maintenance program.

Vinum objects

Vinum implements a four-level hierarchy of objects:

- The most visible object is the virtual disk, called a *volume*. Volumes have essentially the same properties as a UNIX disk drive, though there are some minor differences. They have no size limitations.
- Volumes are composed of *plexes*, each of which represents the total address space of a volume. This level in the hierarchy thus provides redundancy. Think of plexes as individual disks in a mirrored array, each containing the same data.
- *Vinum* exists within the UNIX disk storage framework, so it would be possible to use UNIX partitions as the building block for multi-disk plexes, but in fact this turns out

to be too inflexible: UNIX disks can have only a limited number of partitions. Instead, Vinum subdivides a single UNIX partition (the *drive*) into contiguous areas called *subdisks*, which it uses as building blocks for plexes.

- Subdisks reside on Vinum *drives*, currently UNIX partitions. Vinum drives can contain any number of subdisks. With the exception of a small area at the beginning of the drive, which is used for storing configuration and state information, the entire drive is available for data storage.

Plexes can include multiple subdisks spread over all drives in the Vinum configuration, so the size of an individual drive does not limit the size of a plex, and thus of a volume.

Mapping disk space to plexes

The way the data is shared across the drives has a strong influence on performance. It's convenient to think of the disk storage as a large number of data sectors that are addressable by number, rather like the pages in a book. The most obvious method is to divide the virtual disk into groups of consecutive sectors the size of the individual physical disks and store them in this manner, rather like the way a large encyclopaedia is divided into a number of volumes. This method is called *concatenation*, and sometimes *JBOD (Just a Bunch Of Disks)*. It works well when the access to the virtual disk is spread evenly about its address space. When access is concentrated on a smaller area, the improvement is less marked. Figure 12-1 illustrates the sequence in which storage units are allocated in a concatenated organization.

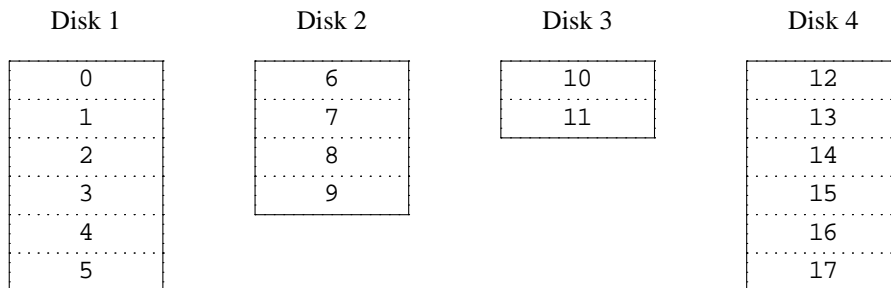


Figure 12-1: Concatenated organization

An alternative mapping is to divide the address space into smaller, equal-sized components, called *stripes*, and store them sequentially on different devices. For example, the first stripe of 292 kB may be stored on the first disk, the next stripe on the next disk and so on. After filling the last disk, the process repeats until the disks are full. This mapping is called *striping* or RAID-0,¹ though the latter term is somewhat misleading: it provides no redundancy. Striping requires somewhat more effort to locate the data, and it can cause additional I/O load where a transfer is spread over multiple disks, but it can also provide a more constant load across the disks. Figure 12-2

1. RAID stands for *Redundant Array of Inexpensive Disks* and offers various forms of fault tolerance.

illustrates the sequence in which storage units are allocated in a striped organization.

Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23

Figure 12-2: Striped organization

Data integrity

Vinum offers two forms of redundant data storage aimed at surviving hardware failure: *mirroring*, also known as RAID level 1, and *parity*, also known as RAID levels 2 to 5.

Mirroring maintains two or more copies of the data on different physical hardware. Any write to the volume writes to both locations; a read can be satisfied from either, so if one drive fails, the data is still available on the other drive. It has two problems:

- The price. It requires twice as much disk storage as a non-redundant solution.
- The performance impact. Writes must be performed to both drives, so they take up twice the bandwidth of a non-mirrored volume. Reads do not suffer from a performance penalty: you only need to read from one of the disks, so in some cases, they can even be faster.

The most interesting of the parity solutions is RAID level 5, usually called *RAID-5*. The disk layout is similar to striped organization, except that one block in each stripe contains the parity of the remaining blocks. The location of the parity block changes from one stripe to the next to balance the load on the drives. If any one drive fails, the driver can reconstruct the data with the help of the parity information. If one drive fails, the array continues to operate in *degraded* mode: a read from one of the remaining accessible drives continues normally, but a read request from the failed drive is satisfied by recalculating the contents from all the remaining drives. Writes simply ignore the dead drive. When the drive is replaced, Vinum recalculates the contents and writes them back to the new drive.

In the following figure, the numbers in the data blocks indicate the relative block numbers.

Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	Parity
3	4	Parity	5
6	Parity	7	8
Parity	9	10	11
12	13	14	Parity
15	16	Parity	17

Figure 12-3: RAID-5 organization

Compared to mirroring, RAID-5 has the advantage of requiring significantly less storage space. Read access is similar to that of striped organizations, but write access is significantly slower, approximately 25% of the read performance.

Vinum also offers *RAID-4*, a simpler variant of RAID-5 which stores all the parity blocks on one disk. This makes the parity disk a bottleneck when writing. RAID-4 offers no advantages over RAID-5, so it's effectively useless.

Which plex organization?

Each plex organization has its unique advantages:

- Concatenated plexes are the most flexible: they can contain any number of subdisks, and the subdisks may be of different length. The plex may be extended by adding additional subdisks. They require less CPU time than striped or RAID-5 plexes, though the difference in CPU overhead from striped plexes is not measurable. They are the only kind of plex that can be extended in size without loss of data.
- The greatest advantage of striped (RAID-0) plexes is that they reduce hot spots: by choosing an optimum sized stripe (between 256 and 512 kB), you can even out the load on the component drives. The disadvantage of this approach is the restriction on subdisks, which must be all the same size. Extending a striped plex by adding new subdisks is so complicated that Vinum currently does not implement it. A striped plex must have at least two subdisks: otherwise it is indistinguishable from a concatenated plex. In addition, there's an interaction between the geometry of UFS and Vinum that makes it advisable not to have a stripe size that is a power of 2: that's the background for the mention of a 292 kB stripe size in the example above.
- RAID-5 plexes are effectively an extension of striped plexes. Compared to striped plexes, they offer the advantage of fault tolerance, but the disadvantages of somewhat higher storage cost and significantly worse write performance. Like striped plexes, RAID-5 plexes must have equal-sized subdisks and cannot currently be extended. Vinum enforces a minimum of three subdisks for a RAID-5 plex: any smaller number would not make any sense.

- Vinum also offers RAID-4, although this organization has some disadvantages and no advantages when compared to RAID-5. The only reason for including this feature was that it was a trivial addition: it required only two lines of code.

The following table summarizes the advantages and disadvantages of each plex organization.

Table 12-1: Vinum plex organizations

Plex type	Minimum subdisks	Can add subdisks	Must be equal size	Application
concatenated	1	yes	no	Large data storage with maximum placement flexibility and moderate performance.
striped	2	no	yes	High performance in combination with highly concurrent access.
RAID-5	3	no	yes	Highly reliable storage, primarily read access.

Creating Vinum drives

Before you can do anything with Vinum, you need to reserve disk space for it. Vinum drive objects are in fact a special kind of disk partition, of type *vinum*. We've seen how to create disk partitions on page 215. If in that example we had wanted to create a Vinum volume instead of a UFS partition, we would have created it like this:

```
8 partitions:
#      size  offset  fstype  [fsize bsize bps/cpg]
c: 6295133    0  unused    0    0
b: 1048576    0  swap      0    0
h: 5246557 1048576  vinum     0    0      # (Cyl. 0 - 10302)
# (Cyl. 0 - 10302)
# (Cyl. 0 - 10302)
```

Starting Vinum

Vinum comes with the base system as a *klm*. It gets loaded automatically when you run the *vinum* command. It's possible to build a special kernel that includes Vinum, but this is not recommended: in this case, you will not be able to stop Vinum.

FreeBSD Release 5 includes a new method of starting Vinum. Put the following lines in */boot/loader.conf*:

```
vinum_load="YES"
vinum.autostart="YES"
```

The first line instructs the loader to load the Vinum kld, and the second tells it to start Vinum during the device probes. Vinum still supports the older method of setting the variable `start_vinum` in */etc/rc.conf*, but this method may go away soon.

Configuring Vinum

Vinum maintains a *configuration database* that describes the objects known to an individual system. You create the configuration database from one or more configuration files with the aid of the *vinum* utility program. Vinum stores a copy of its configuration database on each Vinum drive. This database is updated on each state change, so that a restart accurately restores the state of each Vinum object.

The configuration file

The configuration file describes individual Vinum objects. To define a simple volume, you might create a file called, say, *config1*, containing the following definitions:

```
drive a device /dev/dals2h
volume myvol
  plex org concat
    sd length 512m drive a
```

This file describes four Vinum objects:

- The `drive` line describes a disk partition (*drive*) and its location relative to the underlying hardware. It is given the symbolic name *a*. This separation of the symbolic names from the device names allows disks to be moved from one location to another without confusion.
- The `volume` line describes a volume. The only required attribute is the name, in this case *myvol*.
- The `plex` line defines a plex. The only required parameter is the organization, in this case *concat*. No name is necessary: the system automatically generates a name from the volume name by adding the suffix *.px*, where *x* is the number of the plex in the volume. Thus this plex will be called *myvol.p0*.
- The `sd` line describes a subdisk. The minimum specifications are the name of a drive on which to store it, and the length of the subdisk. As with plexes, no name is necessary: the system automatically assigns names derived from the plex name by adding the suffix *.sx*, where *x* is the number of the subdisk in the plex. Thus Vinum gives this subdisk the name *myvol.p0.s0*

After processing this file, *vinum(8)* produces the following output:

```

vinum -> create config1
1 drives:
D a                State: up      /dev/dals2h      A: 3582/4094 MB (87%)

1 volumes:
V myvol           State: up      Plexes:         1 Size:         512 MB

1 plexes:
P myvol.p0       C State: up      Subdisks:       1 Size:         512 MB

1 subdisks:
S myvol.p0.s0    State: up      D: a            Size:         512 MB

```

This output shows the brief listing format of *vinum*. It is represented graphically in Figure 12-4.

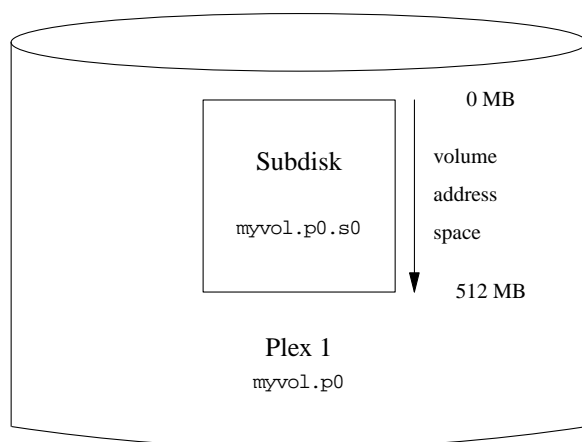


Figure 12-4: A simple Vinum volume

This figure, and the ones that follow, represent a volume, which contains the plexes, which in turn contain the subdisks. In this trivial example, the volume contains one plex, and the plex contains one subdisk.

Creating a file system

You create a file system on this volume in the same way as you would for a conventional disk:

```

# newfs -U /dev/vinum/myvol
/dev/vinum/myvol: 512.0MB (1048576 sectors) block size 16384, fragment size 2048
    using 4 cylinder groups of 128.02MB, 8193 blks, 16512 inodes.
super-block backups (for fsck -b #) at:
    32, 262208, 524384, 786560

```

This particular volume has no specific advantage over a conventional disk partition. It contains a single plex, so it is not redundant. The plex contains a single subdisk, so there is no difference in storage allocation from a conventional disk partition. The following sections illustrate various more interesting configuration methods.

Increased resilience: mirroring

The resilience of a volume can be increased either by mirroring or by using RAID-5 plexes. When laying out a mirrored volume, it is important to ensure that the subdisks of each plex are on different drives, so that a drive failure will not take down both plexes. The following configuration mirrors a volume:

```
drive b device /dev/da2s2h
volume mirror
  plex org concat
    sd length 512m drive a
  plex org concat
    sd length 512m drive b
```

In this example, it was not necessary to specify a definition of drive *a* again, because Vinum keeps track of all objects in its configuration database. After processing this definition, the configuration looks like:

```
2 drives:
D a                State: up          /dev/dals2h      A: 3070/4094 MB (74%)
D b                State: up          /dev/da2s2h      A: 3582/4094 MB (87%)

2 volumes:
V myvol            State: up          Plexes:          1 Size:          512 MB
V mirror           State: up          Plexes:          2 Size:          512 MB

3 plexes:
P myvol.p0         C State: up        Subdisks:        1 Size:          512 MB
P mirror.p0        C State: up        Subdisks:        1 Size:          512 MB
P mirror.pl        C State: initializing Subdisks:        1 Size:          512 MB

3 subdisks:
S myvol.p0.s0      State: up          D: a             Size:          512 MB
S mirror.p0.s0     State: up          D: a             Size:          512 MB
S mirror.pl.s0     State: empty       D: b             Size:          512 MB
```

Figure 12-5 shows the structure graphically.

In this example, each plex contains the full 512 MB of address space. As in the previous example, each plex contains only a single subdisk.

Note the state of *mirror.pl* and *mirror.pl.s0*: *initializing* and *empty* respectively. There's a problem when you create two identical plexes: to ensure that they're identical, you need to copy the entire contents of one plex to the other. This process is called *reviving*, and you perform it with the *start* command:

```
vinum -> start mirror.pl
vinum[278]: reviving mirror.pl.s0
Reviving mirror.pl.s0 in the background
vinum -> vinum[278]: mirror.pl.s0 is up
```

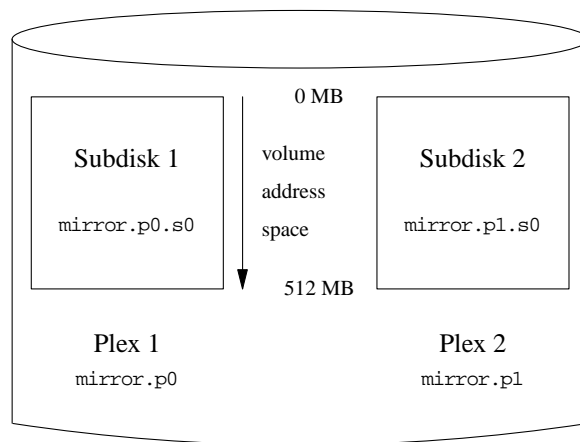


Figure 12-5: A mirrored Vinum volume

During the start process, you can look at the status to see how far the revive has progressed:

```
vinum -> list mirror.p1.s0
S mirror.p1.s0      State: R 43%   D: b           Size:         512 MB
```

Reviving a large volume can take a very long time. When you first create a volume, the contents are not defined. Does it really matter if the contents of each plex are different? If you will only ever read what you have first written, you don't need to worry too much. In this case, you can use the `setupstate` keyword in the configuration file. We'll see an example of this below.

Adding plaxes to an existing volume

At some time after creating a volume, you may decide to add additional plaxes. For example, you may want to add a plex to the volume *myvol* we saw above, putting its subdisk on drive *b*. The configuration file for this extension would look like:

```
plex name myvol.p1 org concat volume myvol
sd size 1g drive b
```

To see what has happened, use the recursive listing option `-r` for the `list` command:

```
vinum -> l -r myvol
V myvol          State: up      Plexes:      2 Size:      1024 MB
P myvol.p0      C State: up    Subdisks:    1 Size:      512 MB
P myvol.p1      C State: initializing Subdisks:    1 Size:      1024 MB
S myvol.p0.s0   State: up      D: a         Size:      512 MB
S myvol.p1.s0   State: empty   D: b         Size:      1024 MB
```

The command *l* is a synonym for *list*, and the *-r* option means *recursive*: it displays all subordinate objects. In this example, plex *myvol.p1* is 1 GB in size, although *myvol.p0* is only 512 MB in size. This discrepancy is allowed, though it isn't very useful by itself: only the first half of the volume is protected against failures. As we'll see in the next section, though, this is a useful stepping stone to extending the size of a file system.

Note that you can't use the *setupstate* keyword here. Vinum can't know whether the existing volume contains valid data or not, so you *must* use the *start* command to synchronize the plexes.

Adding subdisks to existing plexes

After adding a second plex to *myvol*, it had one plex with 512 MB and another with 1024 MB. It makes sense to have the same size plexes, so the first thing we should do is add a second subdisk to the plex *myvol.p0*.

If you add subdisks to striped, RAID-4 or RAID-5 plexes, you will change the mapping of the data to the disks, which effectively destroys the contents. As a result, you must use the *-f* option. When you add subdisks to concatenated plexes, the data in the existing subdisks remains unchanged. In our case, the plex is concatenated, so we create and add the subdisk like this:

```
sd name myvol.p0.s1 plex myvol.p0 size 512m drive c
```

After adding this subdisk, the volume looks like this:

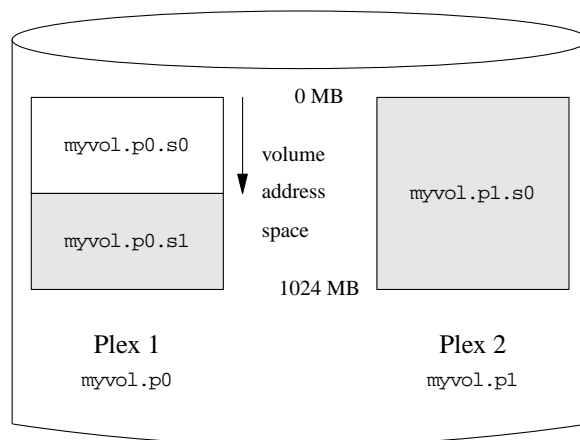


Figure 12-6: An extended Vinum volume

It doesn't look too happy, however:

```
vinum -> l -r myvol
V myvol          State: up      Plexes:      2 Size:      1024 MB
P myvol.p0       C State: corrupt Subdisks:    2 Size:      1024 MB
P myvol.p1       C State: initializing Subdisks: 1 Size:      1024 MB
S myvol.p0.s0    State: up      D: a        Size:       512 MB
S myvol.p0.s1    State: empty   D: c        Size:       512 MB
S myvol.p1.s0    State: stale   D: b        Size:      1024 MB
```

In fact, it's in as good a shape as it ever has been. The first half of *myvol* still contains the file system that we put on it, and it's as accessible as ever. The trouble here is that there is *nothing* in the other two subdisks, which are shown shaded in the figure. Vinum can't know that that is acceptable, but we do. In this case, we use some maintenance commands to set the correct object states:

```
vinum -> setstate up myvol.p0.s1 myvol.p0
vinum -> l -r myvol
V myvol          State: up      Plexes:      2 Size:      1024 MB
P myvol.p0       C State: up      Subdisks:    2 Size:      1024 MB
P myvol.p1       C State: faulty  Subdisks:    1 Size:      1024 MB
S myvol.p0.s0    State: up      D: a        Size:       512 MB
S myvol.p0.s1    State: up      D: c        Size:       512 MB
S myvol.p1.s0    State: stale   D: b        Size:      1024 MB
vinum -> saveconfig
```

The command *setstate* changes the state of individual objects without updating those of related objects. For example, you can use it to change the state of a plex to *up* even if all the subdisks are down. If used incorrectly, it can cause severe data corruption. Unlike normal commands, it doesn't save the configuration changes, so you use *saveconfig* for that, *after* you're sure you have the correct states. Read the man page before using them for any other purpose.

Next you start the second plex:

```
vinum -> start myvol.p1
Reviving myvol.p1.s0 in the background
vinum[446]: reviving myvol.p1.s0
vinum -> vinum[446]: myvol.p1.s0 is up
1
3 drives:
D a          State: up      /dev/dals2h   A: 3582/4094 MB (87%)
D b          State: up      /dev/da2s2h   A: 3070/4094 MB (74%)
D c          State: up      /dev/da3s2h   A: 3582/4094 MB (87%)

1 volumes:
V myvol      State: up      Plexes:      2 Size:      1024 MB

2 plexes:
P myvol.p0   C State: up      Subdisks:    2 Size:      1024 MB
P myvol.p1   C State: up      Subdisks:    1 Size:      1024 MB

3 subdisks:
S myvol.p0.s0 State: up      D: a        Size:       512 MB
S myvol.p1.s0 State: up      D: b        Size:      1024 MB
S myvol.p0.s1 State: up      D: c        Size:       512 MB
```

The message telling you that *myvol.pl.s0* is up comes after the prompt, so the next command doesn't have a prompt. At this point you have a fully mirrored, functional volume, 1 GB in size. If you now look at the contents, though, you see:

```
# df /mnt
Filesystem      1048576-blocks Used Avail Capacity  Mounted on
/dev/vinum/myvol      503      1   461      0%      /mnt
```

The volume is now 1 GB in size, but the file system on the volume is still only 512 MB. To expand it, use *growfs*:

```
# umount /mnt
# growfs /dev/vinum/myvol
We strongly recommend you to make a backup before growing the Filesystem

Did you backup your data (Yes/No) ? Yes
new file systems size is: 524288 frags
Warning: 261920 sector(s) cannot be allocated.
growfs: 896.1MB (1835232 sectors) block size 16384, fragment size 2048
        using 7 cylinder groups of 128.02MB, 8193 blks, 16512 inodes.
super-block backups (for fsck -b #) at:
 1048736, 1310912, 1573088
# mount /dev/vinum/myvol /mnt
# df /mnt
Filesystem      1048576-blocks Used Avail Capacity  Mounted on
/dev/vinum/myvol      881      1   809      0%      /mnt
```

Optimizing performance

The mirrored volumes in the previous example are more resistant to failure than unmirrored volumes, but their performance is less: each write to the volume requires a write to both drives, using up a greater proportion of the total disk bandwidth. Performance considerations demand a different approach: instead of mirroring, the data is striped across as many disk drives as possible. The following configuration shows a volume with a plex striped across four disk drives:

```
drive c device /dev/da3s2h
drive d device /dev/da4s2h
volume stripe
plex org striped 480k
  sd length 128m drive a
  sd length 128m drive b
  sd length 128m drive c
  sd length 128m drive d
```

When creating striped plexes for the UFS file system, ensure that the stripe size is a multiple of the file system block size (normally 16 kB), but not a power of 2. UFS frequently allocates cylinder groups with lengths that are a power of 2, and if you allocate stripes that are also a power of 2, you may end up with all inodes on the same drive, which would significantly impact performance under some circumstances. Files are allocated in blocks, so having a stripe size that is not a multiple of the block size can cause significant fragmentation of I/O requests and consequent drop in performance. See the man page for more details.

Vinum requires that a striped plex have an integral number of stripes. You don't have to calculate the size exactly, though: if the size of the plex is not a multiple of the stripe size, Vinum trims off the remaining partial stripe and prints a console message:

```
vinum: removing 256 blocks of partial stripe at the end of stripe.p0
```

As before, it is not necessary to define the drives that are already known to Vinum. After processing this definition, the configuration looks like:

```
4 drives:
D a                State: up      /dev/dals2h    A: 2942/4094 MB (71%)
D b                State: up      /dev/da2s2h    A: 2430/4094 MB (59%)
D c                State: up      /dev/da3s2h    A: 3966/4094 MB (96%)
D d                State: up      /dev/da4s2h    A: 3966/4094 MB (96%)

3 volumes:
V myvol            State: up      Plexes:        2 Size:        1024 MB
V mirror           State: up      Plexes:        2 Size:        512 MB
V stripe           State: up      Plexes:        1 Size:        511 MB

5 plexes:
P myvol.p0         C State: up      Subdisks:      1 Size:        512 MB
P mirror.p0        C State: up      Subdisks:      1 Size:        512 MB
P mirror.pl        C State: initializing Subdisks:      1 Size:        512 MB
P myvol.pl         C State: up      Subdisks:      1 Size:        1024 MB
P stripe.p0        S State: up      Subdisks:      4 Size:        511 MB

8 subdisks:
S myvol.p0.s0      State: up      D: a           Size:        512 MB
S mirror.p0.s0     State: up      D: a           Size:        512 MB
S mirror.pl.s0     State: empty   D: b           Size:        512 MB
S myvol.pl.s0      State: up      D: b           Size:        1024 MB
S myvol.p0.s1      State: up      D: c           Size:        512 MB
S stripe.p0.s0     State: up      D: a           Size:        127 MB
S stripe.p0.s1     State: up      D: b           Size:        127 MB
S stripe.p0.s2     State: up      D: c           Size:        127 MB
S stripe.p0.s3     State: up      D: d           Size:        127 MB
```

This volume is represented in Figure 12-7. The darkness of the stripes indicates the position within the plex address space: the lightest stripes come first, the darkest last.

Resilience and performance

With sufficient hardware, it is possible to build volumes that show both increased resilience and increased performance compared to standard UNIX partitions. Mirrored disks will always give better performance than RAID-5, so a typical configuration file might be:

```
drive e device /dev/da5s2h
drive f device /dev/da6s2h
drive g device /dev/da7s2h
drive h device /dev/da8s2h
drive i device /dev/da9s2h
drive j device /dev/da10s2h
volume raid10 setupstate
  plex org striped 480k
    sd length 102480k drive a
    sd length 102480k drive b
```

vinum.mm,v v4.20 (2003/06/29 04:33:42) (modified 29 Jul 2003, 01:01:03 UTC)

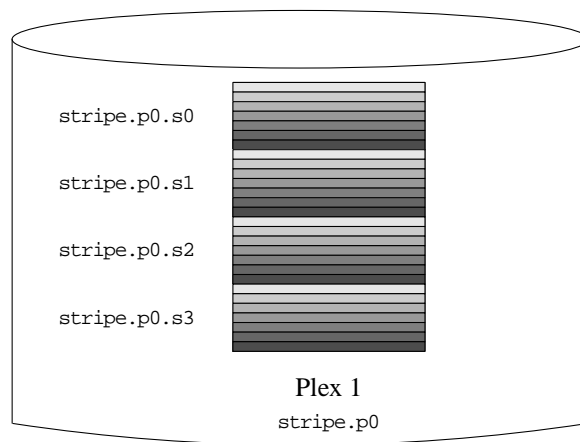


Figure 12-7: A striped Vinum volume

```
sd length 102480k drive c
sd length 102480k drive d
sd length 102480k drive e
plex org striped 480k
sd length 102480k drive f
sd length 102480k drive g
sd length 102480k drive h
sd length 102480k drive i
sd length 102480k drive j
```

In this example, we have added another five disks for the second plex, so the volume is spread over ten spindles. We have also used the `setupstate` keyword so that all components come up. The volume looks like this:

```
vinum -> l -r raid10
V raid10          State: up      Plexes:      2 Size:      499 MB
P raid10.p0       S State: up    Subdisks:    5 Size:      499 MB
P raid10.p1       S State: up    Subdisks:    5 Size:      499 MB
S raid10.p0.s0    State: up     D: a         Size:       99 MB
S raid10.p0.s1    State: up     D: b         Size:       99 MB
S raid10.p0.s2    State: up     D: c         Size:       99 MB
S raid10.p0.s3    State: up     D: d         Size:       99 MB
S raid10.p0.s4    State: up     D: e         Size:       99 MB
S raid10.p1.s0    State: up     D: f         Size:       99 MB
S raid10.p1.s1    State: up     D: g         Size:       99 MB
S raid10.p1.s2    State: up     D: h         Size:       99 MB
S raid10.p1.s3    State: up     D: i         Size:       99 MB
S raid10.p1.s4    State: up     D: j         Size:       99 MB
```

This assumes the availability of ten disks. It's not essential to have all the components on different disks. You could put the subdisks of the second plex on the same drives as the subdisks of the first plex. If you do so, you should put corresponding subdisks on different drives:

```

plex org striped 480k
  sd length 102480k drive a
  sd length 102480k drive b
  sd length 102480k drive c
  sd length 102480k drive d
  sd length 102480k drive e
plex org striped 480k
  sd length 102480k drive c
  sd length 102480k drive d
  sd length 102480k drive e
  sd length 102480k drive a
  sd length 102480k drive b

```

The subdisks of the second plex are offset by two drives from those of the first plex: this helps ensure that the failure of a drive does not cause the same part of both plexes to become unreachable, which would destroy the file system.

Figure 12-8 represents the structure of this volume.

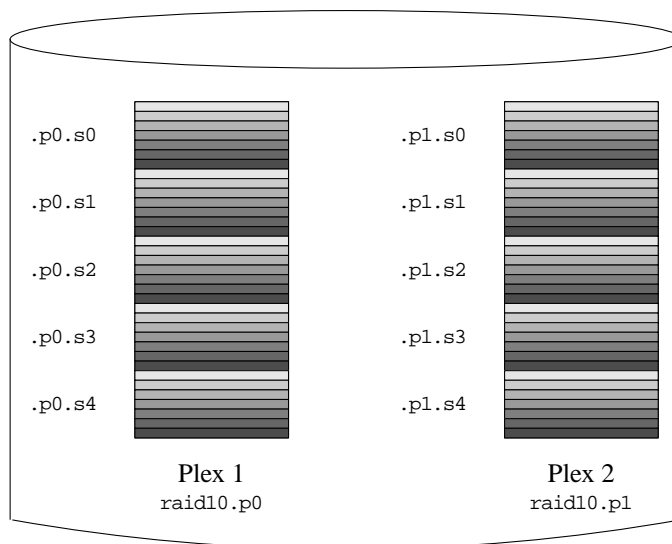


Figure 12-8: A mirrored, striped Vinum volume

Vinum configuration database

Vinum stores configuration information on each drive in essentially the same form as in the configuration files. You can display it with the *dumpconfig* command. When reading from the configuration database, Vinum recognizes a number of keywords that are not allowed in the configuration files, because they would compromise data integrity. For example, after adding the second plex to *myvol*, the disk configuration would contain the following text:

```
vinum.mm,v v4.20 (2003/06/29 04:33:42) (modified 29 Jul 2003, 01:01:03 UTC)
```

```

vinum -> dumpconfig
Drive a:      Device /dev/dals2h
              Created on bumble.example.org at Tue Nov 26 14:35:12 2002
              Config last updated Tue Nov 26 16:12:35 2002
              Size:      4293563904 bytes (4094 MB)
volume myvol state up
plex name myvol.p0 state up org concat vol myvol
plex name myvol.p1 state up org concat vol myvol
sd name myvol.p0.s0 drive a plex myvol.p0 len 1048576s driveoffset 265s state up ple
xoffset 0s
sd name myvol.p1.s0 drive b plex myvol.p1 len 2097152s driveoffset 265s state up ple
xoffset 0s
sd name myvol.p0.s1 drive c plex myvol.p0 len 1048576s driveoffset 265s state up ple
xoffset 1048576s

Drive /dev/dals2h: 4094 MB (4293563904 bytes)

Drive b:      Device /dev/da2s2h
              Created on bumble.example.org at Tue Nov 26 14:35:27 2002
              Config last updated Tue Nov 26 16:12:35 2002
              Size:      4293563904 bytes (4094 MB)
volume myvol state up
plex name myvol.p0 state up org concat vol myvol
plex name myvol.p1 state up org concat vol myvol
sd name myvol.p0.s0 drive a plex myvol.p0 len 1048576s driveoffset 265s state up ple
xoffset 0s
sd name myvol.p1.s0 drive b plex myvol.p1 len 2097152s driveoffset 265s state up ple
xoffset 0s
sd name myvol.p0.s1 drive c plex myvol.p0 len 1048576s driveoffset 265s state up ple
xoffset 1048576s

```

The obvious differences here are the presence of explicit location information and naming (both of which are also allowed, but discouraged, for use by the user) and the information on the states (which are not available to the user). Vinum does not store information about drives in the configuration information: it finds the drives by scanning the configured disk drives for partitions with a Vinum label. This enables Vinum to identify drives correctly even if they have been assigned different UNIX drive IDs.

When you start Vinum with the *vinum start* command, Vinum reads the configuration database from one of the Vinum drives. Under normal circumstances, each drive contains an identical copy of the configuration database, so it does not matter which drive is read. After a crash, however, Vinum must determine which drive was updated most recently and read the configuration from this drive. It then updates the configuration, if necessary, from progressively older drives.

Installing FreeBSD on Vinum

Installing FreeBSD on Vinum is currently complicated by the fact that *sysinstall* and the loader don't support Vinum, so it is not possible to install directly on a Vinum volume. Instead, you need to install a conventional system and then convert it to Vinum. That's not as difficult as it might sound.

Normal disk installations lay out overlapping disk partitions: the *c* partition overlaps all the other partitions. You can do the same thing with a Vinum drive, which is also a partition. You can then create subdisks in the Vinum drive corresponding in length and position to the partitions. In the following diagram, each column represents the entire disk. On the left there are four normal partitions. In the middle is the *c* partition, and on the right is a Vinum drive partition:

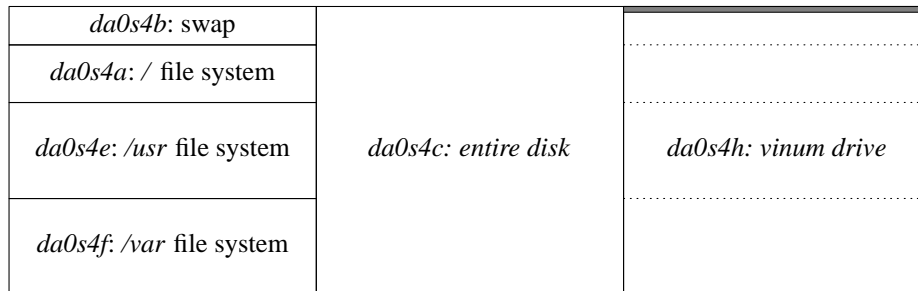


Figure 12-9: Partition layout with Vinum

This layout shows three file system partitions and a swap partition, which is not the layout recommended on page 68. We'll look at the reasons for this below.

The shaded area at the top of the Vinum partition represents the configuration information, which cuts into the swap partition and the bootstrap. To fix that, we redefine the swap partition to start after the Vinum configuration information and to be a total of 281 sectors shorter, 265 sectors for the Vinum configuration and 16 sectors for the bootstrap.

The swap partition isn't normally the first partition on a drive, but you can create this layout with *sysinstall* simply by creating the swap partition before any other partition. Consider installing FreeBSD on a 4 GB drive. Create, in sequence, a swap partition of 256 MB, a root file system of 256 MB, a /usr file system of 2 GB, and a /var file system to take up the rest. It's important to create the swap partition at the beginning of the disk, so you create that first. After installation, the output of *bsdlabel* looks like this:

```

a:  524288  524288  4.2BSD  2048 16384 32776      root file system
b:  524288      0      swap      0      0      # "raw" part, don't edit
c:  8385867      0  unused      0      0
d:  4194304 1048576  4.2BSD  2048 16384 28512      /usr file system
e:  3142987 5242880  4.2BSD  2048 16384 28512      /var file system

```

This corresponds to the left and centre columns in the figure above. To convert to Vinum, you need to:

- create a volume of type *vinum* that starts after the bootstrap in the *c* partition.
- shorten the swap partition by 281 sectors at the beginning.

Boot in single user mode and remount the root file system (to make it read/write), mount

the `/usr` directory and run `bsdlabel` with the `-e` (edit label) option:

```
# mount -u /
# mount /usr
# bsdlabel -e da0s4
```

See page 215 for more information about `bsdlabel`. You need to boot in single user mode because otherwise the swap partition would be mounted, and you can't change the size of the swap partition when it's mounted.

When you finish, the partition table should look like this (changed values in **bold**):

```
#      size  offset  fstype  [fsize bsize bps/cpg]
a:    524288  524288  4.2BSD  2048 16384 32776      root file system
b:    524007    281    swap
c:    8385867    0    unused      0    0    # "raw" part, don't edit
d:    4194304 1048576  4.2BSD  2048 16384 28512      /usr file system
e:    3142987 5242880  4.2BSD  2048 16384 28512      /var file system
h:    8385851    16    vinum
                                Vinum drive
```

Be sure to shorten the length of the swap partition by 281 sectors, or it will overlap the root partition and cause extreme data corruption when swap gets full, which might not happen until months later.

The next step is to create the Vinum objects. The plexes and volumes are straightforward: each plex is concatenated with a single subdisk, and the volume has a single plex. It makes sense to give the volumes names that relate to the mount point.

Creating the subdisks requires a little more care. You can use the size values from the `bsdlabel` output above directly in a Vinum configuration file, but since the Vinum drive starts after the bootstrap, you need to subtract 16 (the length of the bootstrap) from the offset values:

```
drive rootdev device /dev/da0s4h
volume root
  plex org concat
  # a:    524288      524288  4.2BSD  2048 16384 32776
    sd len 524288s  driveoffset 524272s drive rootdev
volume swap
  plex org concat
  # b:    524007      281    swap
    sd len 524007s  driveoffset 265s drive rootdev
volume usr
  plex org concat
  # d:    4194304    1048576  4.2BSD  2048 16384 28512
    sd len 4194304s  driveoffset 1048560s drive rootdev
volume var
  plex org concat
  # e:    3142987    5242880  4.2BSD  2048 16384 28512
    sd len 3142987s  driveoffset 5242864s drive rootdev
```

The comments are the corresponding lines from the `bsdlabel` output. They show the corresponding values for size and offset. Run `vinum create` against this file, and confirm that you have the volumes `/`, `/usr` and `/var`.

Next, ensure that you are set up to start Vinum with the new method. Ensure that you have the following lines in `/boot/loader.conf`, creating it if necessary:

```
vinum.mm,v v4.20 (2003/06/29 04:33:42) (modified 29 Jul 2003, 01:01:03 UTC)
```

```
vinum_load="YES"
vinum.autostart="YES"
```

Then reboot to single-user mode, start Vinum and run *fsck* against the volumes, using the *-n* option to tell *fsck* not to correct any errors it finds. You should see something like this:

```
# fsck -n -t ufs /dev/vinum/usr
** /dev/vinum/usr (NO WRITE)
** Last Mounted on /usr
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
35323 files, 314115 used, 718036 free (4132 frags, 89238 blocks, 0.4% fragmentation)
```

If there are any errors, they will probably be because you have miscalculated size or offset. You'll see something like this:

```
# fsck -n -t ufs /dev/vinum/usr
** /dev/vinum/usr (NO WRITE)
Cannot find file system superblock
/dev/vinum/usr: CANNOT FIGURE OUT FILE SYSTEM PARTITION
```

You need to do this in single-user mode because the volumes are shadowing file systems, and it's normal for open file systems to fail *fsck*, since some of the state is in buffer cache.

If all is well, remount the root file system read-write:

```
# mount -u /
```

Then edit */etc/fstab* to point to the new devices. For this example, */etc/fstab* might initially contain:

```
# $Id: fstab,v 1.3 2002/11/14 06:48:16 grog Exp $
# Device      Mountpoint  FStype  Options      Dump  Pass#
/dev/da0s4a   /           ufs     rw           1     1
/dev/da0s4b   none       swap    sw           0     0
/dev/da0s4e   /usr       ufs     rw           1     1
/dev/da0s4f   /var       ufs     rw           1     1
```

Change it to reflect the Vinum volumes:

```
# $Id: fstab,v 1.3 2002/11/14 06:48:16 grog Exp $
# Device      Mountpoint  FStype  Options      Dump  Pass#
/dev/vinum/swap none       swap    sw           0     0
/dev/vinum/root /           ufs     rw           1     1
/dev/vinum/usr /usr       ufs     rw           1     1
/dev/vinum/var /var       ufs     rw           1     1
```

Then reboot again to mount the root file system from */dev/vinum/root*. You can also optionally remove all the UFS partitions *except the root partition*. The loader doesn't know about Vinum, so it must boot from the UFS partition.

Once you have reached this stage, you can add additional plexes to the volumes, or you can extend the plexes (and thus the size of the file system) by adding subdisks to the plexes, as discussed on page 229.

Recovering from drive failures

One of the purposes of Vinum is to be able to recover from hardware problems. If you have chosen a redundant storage configuration, the failure of a single component will not stop the volume from working. In many cases, you can replace the components without down time.

If a drive fails, perform the following steps:

1. Replace the physical drive.
2. Partition the new drive. Some restrictions apply:
 - If you have hot-plugged the drive, it must have the same ID, the Vinum drive must be on the same partition, and it must have the same size.
 - If you have had to stop the system to replace the drive, the old drive will not be associated with a device name, and you can put it anywhere. Create a Vinum partition that is at least large enough to take all the subdisks *in their original positions on the drive*. Vinum currently does not compact free space when replacing a drive. An easy way to ensure this is to make the new drive at least as large as the old drive.

If you want to have this freedom with a hot-pluggable drive, you must stop Vinum and restart it.

3. If you have restarted Vinum, create a new drive. For example, if the replacement drive *data3* is on the physical partition */dev/da3s1h*, create a configuration file, say *configfile*, with the single line

```
drive data3 device /dev/da3s1h
```

Then enter:

```
# vinum create configfile
```

4. Start the plexes that were down. For example, *vinum list* might show:

```
vinum -> l -r test
V test                State: up           Plexes:           2 Size:           30 MB
P test.p0             C State: up        Subdisks:         1 Size:           30 MB
P test.pl             C State: faulty    Subdisks:         1 Size:           30 MB
S test.p0.s0          State: up           PO:               0 B Size:         30 MB
S test.pl.s0          State: obsolete    PO:               0 B Size:         30 MB
vinum -> start test.pl.s0
Reviving test.pl.s0 in the background
vinum -> vinum[295]: reviving test.pl.s0
vinum[295]: test.pl.s0 is up
```

*this message appears after the prompt
(some time later)*

Failed boot disk

If you're running your root file system on a Vinum volume, you can survive the failure of the boot volume if it is mirrored with at least two concatenated plexes each containing only one subdisk. Under normal circumstances, you can carry on running as if nothing had happened, but obviously you will no longer be able to reboot from that disk. Instead, boot from the other disk.

The root file system also has individual UFS partitions, so you have a choice of what you mount. For example, if your root file system has UFS partitions `/dev/da0s4a` and `/dev/da1s4a`, you can mount either of these partitions or `/dev/vinum/root`. Never mount more than one of them, otherwise you can cause data corruption.

An even more insidious way to corrupt the root file system is to mount `/dev/da0s4a` or `/dev/da1s4a` and modify it. In this case, the two partitions are no longer the same, but there's no way for Vinum to know that. If this happens, you *must* mark the other subdisk as crashed with the `vinum stop` command.

Migrating Vinum to a new machine

Sometimes you might want to move a set of Vinum disks to a different FreeBSD machine. This is simple, as long as there are no name conflicts between the objects on the Vinum disks and any other Vinum objects you may already have on the system. Simply connect the disks and start Vinum. You don't need to put the disks in any particular location, and you don't need to run `vinum create`: Vinum stores the configuration on the drives themselves, and when it starts, it locates it accordingly.

Things you shouldn't do with Vinum

The `vinum` command offers a large number of subcommands intended for specific purposes. It's easy to abuse them. Here are some things you should not do:

- Do not use the `resetconfig` command unless you genuinely don't want to see any of your configuration again. There are other alternatives, such as `rm`, which removes individual objects or groups of objects.
- Do not re-run the `create` command for objects that already exist. Vinum already knows about them, and the `start` command should find them.
- Do not name your drives after the disk device on which they are located. The purpose of having drive names is to be device independent. For example, if you have two drives *a* and *b*, and they are located on devices `/dev/da1s1h` and `/dev/da2s1h` respectively, you can remove the drives, swap their locations and restart Vinum. Vinum will still correctly locate the drives. If you had called the drives `da1` and `da2`, you would then see something confusing like this:

```
2 drives:
D da2          State: up      /dev/da1s1h   A: 3582/4094 MB (87%)
D da1          State: up      /dev/da1s2h   A: 3582/4094 MB (87%)
```

This is clearly not helpful.

- Don't put more than one drive on a physical disk. Each drive contains two copies of the Vinum configuration, and both updating the configuration and starting Vinum slow down as a result. If you want more than one file system to occupy space on a physical drive, create subdisks, not drives.