NAME

vinum − Logical Volume Manager control program

SYNOPSIS

vinum [command] [**−options**]

COMMANDS

attach *plex volume* [**rename**] **attach** *subdisk plex [offset]* [**rename**]
: Attach a plex to a volume, or a subdisk to a plex.

checkparity *plex* [**−f**] [**−v**]
: Check the parity blocks of a RAID-4 or RAID-5 plex.

concat [**−f**] [**−n** *name*] [**−v**] *drives*
: Create a concatenated volume from the specified drives.

create [**−f**] *description-file*
: Create a volume as described in *description-file*

debug
: Cause the volume manager to enter the kernel debugger.

debug *flags*
: Set debugging flags.

detach [**−f**] [*plex* | *subdisk*]
: Detach a plex or subdisk from the volume or plex to which it is attached.

dumpconfig [*drive ...*]
: List the configuration information stored on the specified drives, or all drives in the system if no drive names are specified.

info [**−v**] [**−V**]
: List information about volume manager state.

init [**−S** *size*] [**−w**] *plex* | *subdisk*
: Initialize the contents of a subdisk or all the subdisks of a plex to all zeros.

label *volume*
: Create a volume label

list [**−r**] [**−s**] [**−v**] [**−V**] [volume | plex | subdisk]
: List information about specified objects

l [**−r**] [**−s**] [**−v**] [**−V**] [volume | plex | subdisk]
: List information about specified objects (alternative to **list** command)

ld [**−r**] [**−s**] [**−v**] [**−V**] [volume]
: List information about drives

ls [**−r**] [**−s**] [**−v**] [**−V**] [subdisk]
: List information about subdisks

lp [**−r**] [**−s**] [**−v**] [**−V**] [plex]
: List information about plexes

lv [**−r**] [**−s**] [**−v**] [**−V**] [volume]
: List information about volumes

makedev
: Remake the device nodes in */dev/vinum*.

mirror [**−f**] [**−n** *name*] [**−s**] [**−v**] *drives*
: Create a mirrored volume from the specified drives.

mv −f *drive object ...*
move −f *drive object ...*
: Move the object(s) to the specified drive.

printconfig [file]
: Write a copy of the current configuration to `file`.

quit
: Exit the **vinum** program when running in interactive mode. Normally this would be done by entering the *EOF* character.

read *disk* [disk...]
: Read the **vinum** configuration from the specified disks.

rename [**−r**] [*drive* | *subdisk* | *plex* | *volume*] *newname*
: Change the name of the specified object.

rebuildparity *plex* [**−f**] [**−v**] [**−V**]
: Rebuild the parity blocks of a RAID-4 or RAID-5 plex.

resetconfig
: Reset the complete **vinum** configuration.

resetstats [**−r**] [volume | plex | subdisk]
: Reset statistisc counters for the specified objects, or for all objects if none are specified.

rm [**−f**] [**−r**] *volume* | *plex* | *subdisk*
: Remove an object

saveconfig
: Save **vinum** configuration to disk.

setdaemon [value]
: Set dæmon configuration.

setstate *state* [*volume* | *plex* | *subdisk* | *drive*]
: Set state without influencing other objects, for diagnostic purposes only.

start
: Read configuration from all vinum drives.

start [**−i** *interval*] [**−S** *size*] [**−w**] volume | plex | subdisk
: Allow the system to access the objects

stop [**−f**] [volume | plex | subdisk]
: Terminate access to the objects, or stop **vinum** if no parameters are specified.

stripe [**−f**] [**−n** *name*] [**−v**] *drives*
: Create a striped volume from the specified drives.

DESCRIPTION

vinum is a utility program to communicate with the **Vinum** logical volume manager. See vinum(4) for more information about the volume manager. vinum(8) is designed either for interactive use, when started without command line arguments, or to execute a single command if the command is supplied on the command line. In interactive mode, **vinum** maintains a command line history.

OPTIONS

vinum commands may optionally be followed by an option. Any of the following options may be specified with any command, but in some cases they do not make any difference: cases, the options are ignored. For example, the **stop** command ignores the **−v** and **−V** options.

−f
: The **−f** ("force") option overrides safety checks. Use with extreme care. This option is for emergency use only. For example, the command

```
rm -f myvolume
```

removes *myvolume* even if it is open. Any subsequent access to the volume will almost certainly cause a panic.

−i *millisecs*
: When performing the **init** and **start** commands, wait *millisecs* milliseconds between copying each block. This lowers the load on the system.

−n *name* Use the **−n** option to specify a volume name to the simplified configuration commands **concat**, **mirror** and **stripe**.

−r The **−r** ("recursive") option is used by the list commands to display information not only about the specified objects, but also about subordinate objects. For example, in conjnction with the **lv** command, the **−r** option will also show information about the plexes and subdisks belonging to the volume.

−s The **−s** ("statistics") option is used by the list commands to display statistical information. The **mirror** command also uses this option to specify that it should create striped plexes.

−S *size* The **−S** option specifies the transfer size for the **init** and **start** commands.

−v The **−v** ("verbose") option can be used to request more detailed information.

−V The **−V** ("Very verbose") option can be used to request more detailed information than the **−v** option provides.

−w The **−w** ("wait") option tells **vinum** to wait for completion of commands which normally run in the background, such as **init**.

COMMANDS IN DETAIL

vinum commands perform the following functions:

attach *plex volume* [**rename**]
attach *subdisk plex [offset]* [**rename**]

vinum *attach* inserts the specified plex or subdisk in a volume or plex. In the case of a subdisk, an offset in the plex may be specified. If it is not, the subdisk will be attached at the first possible location. After attaching a plex to a non-empty volume, **vinum** reintegrates the plex.

If the keyword **rename** is specified, **vinum** renames the object (and in the case of a plex, any subordinate subdisks) to fit in with the default **vinum** naming convention.

A number of considerations apply to attaching subdisks:

- Subdisks can normally only be attached to concatenated plexes.
- If a striped or RAID-5 plex is missing a subdisk (for example after drive failure), it should be replaced by a subdisk of the same size only.
- In order to add further subdisks to a striped or RAID-5 plex, use the **−f** (force) option. This will corrupt the data in the plex.
- For concatenated plexes, the *offset* parameter specifies the offset in blocks from the beginning of the plex. For striped and RAID-5 plexes, it specifies the offset of the first block of the subdisk: in other words, the offset is the numerical position of the subdisk multiplied by the stripe size. For example, in a plex of block size 256k, the first subdisk will have offset 0, the second offset 256k, the third 512k, etc. This calculation ignores parity blocks in RAID-5 plexes.

checkparity *plex* [**−f**] [**−v**]

Check the parity blocks on the specified RAID-4 or RAID-5 plex. This operation maintains a pointer in the plex, so it can be stopped and later restarted from the same position if desired. In addition, this pointer is used by the **rebuildparity** command, so rebuilding the parity blocks need only start at the location where the first parity problem has been detected.

If the **−f** flag is specified, **checkparity** starts checking at the beginning of the plex. If the **−v** flag is specified, **checkparity** prints a running progress report.

concat [**−f**] [**−n** *name*] [**−v**] *drives*

The **concat** command provides a simplified alternative to the **create** command for creating volumes with a single concatenated plex. The largest contiguous space available on each drive is used to create the subdisks for the plexes.

Normally, the **concat** command creates an arbitrary name for the volume and its components. The name is composed of the text *vinum* and a small integer, for example *vinum3*. You can override this with the **−n** *name* option, which assigns the name specified to the volume. The plexes and subdisks are named after the volume in the default manner.

There is no choice of name for the drives. If the drives have already been initialized as **vinum** drives, the name remains. Otherwise the drives are given names starting with the text *vinumdrive* and a small integer, for example *vinumdrive7*. As with the **create** command, the **−f** option can be used to specify that a previous name should be overwritten. The **−v** is used to specify verbose output.

See the section SIMPLIFIED CONFIGURATION below for some examples of this command.

create [**−f** *description-file*]

vinum *create* is used to create any object. In view of the relatively complicated relationship and the potential dangers involved in creating a **vinum** object, there is no interactive interface to this function. If you do not specify a file name, **vinum** starts an editor on a temporary file. If the environment variable EDITOR is set, **vinum** starts this editor. If not, it defaults to **vi**. See the section CONFIGURATION FILE below for more information on the format of this file.

Note that the **vinum** *create* function is additive: if you run it multiple times, you will create multiple copies of all unnamed objects.

Normally the **create** command will not change the names of existing **vinum** drives, in order to avoid accidentally erasing them. The correct way to dispose of no longer wanted **vinum** drives is to reset the configuration with the **resetconfig** command. In some cases, however, it may be necessary to create new data on **vinum** drives which can no longer be started. In this case, use the **create −f** command.

debug

vinum *debug* is used to enter the remote kernel debugger. It is only activated if **vinum** is built with the *VINUMDEBUG* option. This option will stop the execution of the operating system until the kernel debugger is exited. If remote debugging is set and there is no remote connection for a kernel debugger, it will be necessary to reset the system and reboot in order to leave the debugger.

debug *flags*

Set a bit mask of internal debugging flags. These will change without warning as the product matures; to be certain, read the header file `sys/dev/vinumvar.h`. The bit mask is composed of the following values:

DEBUG_ADDRESSES (1)
Show buffer information during requests

DEBUG_RESID (4)
Go into debugger in **complete_rqe.**()

DEBUG_LASTREQS (8)
Keep a circular buffer of last requests.

DEBUG_REVIVECONFLICT (16)
Print info about revive conflicts.

DEBUG_EOFINFO (32)
Print information about internal state when returning an EOF on a striped plex.

DEBUG_MEMFREE (64)
Maintain a circular list of the last memory areas freed by the memory allocator.

DEBUG_REMOTEGDB (256)
Go into remote **gdb** when the **debug** command is issued.

DEBUG_WARNINGS (512)
Print some warnings about minor problems in the implementation.

detach [**−f**] *plex*
detach [**−f**] *subdisk*

vinum *detach* removes the specified plex or subdisk from the volume or plex to which it is attached. If removing the object would impair the data integrity of the volume, the operation will fail unless the **−f** option is specified. If the object is named after the object above it (for example, subdisk vol1.p7.s0 attached to plex vol1.p7), the name will be changed by prepending the text ‘‘ex-’’ (for example, ex-vol1.p7.s0). If necessary, the name will be truncated in the process.

detach does not reduce the number of subdisks in a striped or RAID-5 plex. Instead, the subdisk is marked absent, and can later be replaced with the **attach** command.

dumpconfig [*drive ...*]

vinum *dumpconfig* shows the configuration information stored on the specified drives. If no drive names are specified, **dumpconfig** searches all drives on the system for Vinum partitions and dumps the information. If configuration updates are disabled, it is possible that this information is not the same as the information returned by the **list** command. This command is used primarily for maintenance and debugging.

info

vinum *info* displays information about **vinum** memory usage. This is intended primarily for debugging. With the **−v** option, it will give detailed information about the memory areas in use.

With the **−V** option, *info* displays information about the last up to 64 I/O requests handled by the **vinum** driver. This information is only collected if debug flag 8 is set. The format looks like:

```
vinum -> info -V
Flags: 0x200    1 opens
Total of 38 blocks malloced, total memory: 16460
Maximum allocs:       56, malloc table at 0xf0f72dbc

Time             Event       Buf         Dev     Offset          Bytes   SD

14:40:00.637758 1VS Write 0xf2361f40    91.3  0x10           16384
14:40:00.639280 2LR Write 0xf2361f40    91.3  0x10           16384
14:40:00.639294 3RQ Read  0xf2361f40    4.39   0x104109        8192    19
14:40:00.639455 3RQ Read  0xf2361f40    4.23   0xd2109         8192    17
14:40:00.639529 3RQ Read  0xf2361f40    4.15   0x6e109         8192    16
14:40:00.652978 4DN Read  0xf2361f40    4.39   0x104109        8192    19
14:40:00.667040 4DN Read  0xf2361f40    4.15   0x6e109         8192    16
```

```
14:40:00.668556 4DN Read  0xf2361f40    4.23    0xd2109         8192    17
14:40:00.669777 6RP Write 0xf2361f40    4.39    0x104109        8192    19
14:40:00.685547 4DN Write 0xf2361f40    4.39    0x104109        8192    19
11:11:14.975184 Lock      0xc2374210    2       0x1f8001
11:11:15.018400 7VS Write 0xc2374210            0x7c0           32768   10
11:11:15.018456 8LR Write 0xc2374210    13.39   0xcc0c9         32768
11:11:15.046229 Unlock    0xc2374210    2       0x1f8001
```

The *Buf* field always contains the address of the user buffer header. This can be used to identify the requests associated with a user request, though this is not 100% reliable: theoretically two requests in sequence could use the same buffer header, though this is not common. The beginning of a request can be identified by the event *1VS* or *7VS*. The first example above shows the requests involved in a user request. The second is a subdisk I/O request with locking.

The *Event* field contains information related to the sequence of events in the request chain. The digit *1* to *6* indicates the approximate sequence of events, and the two-letter abbreviation is a mnemonic for the location

1VS (vinumstrategy) shows information about the user request on entry to **vinumstrategy**(). The device number is the **vinum** device, and offset and length are the user parameters. This is always the beginning of a request sequence.

2LR (launch_requests) shows the user request just prior to launching the low-level **vinum** requests in the function **launch_requests**(). The parameters should be the same as in the *1VS* information.

In the following requests, *Dev* is the device number of the associated disk partition, *Offset* is the offset from the beginning of the partition, *SD* is the subdisk index in vinum_conf, *SDoff* is the offset from the beginning of the subdisk, *Doffset* is the offset of the associated data request, and *Goffset* is the offset of the associated group request, where applicable.

3RQ (request) shows one of possibly several low-level **vinum** requests which are launched to satisfy the high-level request. This information is also logged in **launch_requests**().

4DN (done) is called from **complete_rqe**(), showing the completion of a request. This completion should match a request launched either at stage *4DN* from **launch_requests**(), or from **complete_raid5_write**() at stage *5RD* or *6RP*.

5RD (RAID-5 data) is called from **complete_raid5_write**() and represents the data written to a RAID-5 data stripe after calculating parity.

6RP (RAID-5 parity) is called from **complete_raid5_write**() and represents the data written to a RAID-5 parity stripe after calculating parity.

7VS shows a subdisk I/O request. These requests are usually internal to **vinum** for operations like initialization or rebuilding plexes.

8LR shows the low--level operation generated for a subdisk I/O request.

Lockwait specifies that the process is waiting for a range lock. The parameters are the buffer header associated with the request, the plex number and the block number. For internal reasons the block number is one higher than the address of the beginning of the stripe.

Lock specifies that a range lock has been obtained. The parameters are the same as for the range lock.

Unlock specifies that a range lock has been released. The parameters are the same as for the range lock.

init [**−S**] *size* [**−w**] *plex* | *subdisk*

vinum *init* initializes a subdisk by writing zeroes to it. You can initialize all subdisks in a plex by specifying the plex name. This is the only way to ensure consistent data in a plex. You must perform this initialization before using a RAID-5 plex. It is also recommended for other new plexes. **vinum** initializes all subdisks of a plex in parallel. Since this operation can take a long time, it is normally performed in the background. If you want to wait for completion of the command, use the **−w** (wait) option.

Specify the **−S** option if you want to write blocks of a different size from the default value of 16 kB. **vinum** prints a console message when the initialization is complete.

label *volume*

The **label** command writes a *ufs* style volume label on a volume. It is a simple alternative to an appropriate call to *disklabel*. This is needed because some *ufs* commands still read the disk to find the label instead of using the correct *ioctl* call to access it. **vinum** maintains a volume label separately from the volume data, so this command is not needed for *newfs*. This command is deprecated.

list [**−r**] [**−V**] [volume | plex | subdisk]
l [**−r**] [**−V**] [volume | plex | subdisk]
ld [**−r**] [**−s**] [**−v**] [**−V**] [volume]
ls [**−r**] [**−s**] [**−v**] [**−V**] [subdisk]
lp [**−r**] [**−s**] [**−v**] [**−V**] [plex]
lv [**−r**] [**−s**] [**−v**] [**−V**] [volume]

list is used to show information about the specified object. If the argument is omitted, information is shown about all objects known to **vinum**. The *l* command is a synonym for *list*.

The **−r** option relates to volumes and plexes: if specified, it recursively lists information for the subdisks and (for a volume) plexes subordinate to the objects. The commands *lv*, *lp*, *ls* and *ld* commands list only volumes, plexes, subdisks and drives respectively. This is particularly useful when used without parameters.

The **−s** option causes **vinum** to output device statistics, the [**−v**] (verbose) option causes some additional information to be output, and the [**−V**] causes considerable additional information to be output.

makedev

The **makedev** command removes the directory /dev/vinum and recreates it with device nodes which reflect the current configuration. This command is not intended for general use, and is provided for emergency use only.

mirror [**−f**] [**−n** *name*] [**−s**] [**−v**] *drives*

The **mirror** command provides a simplified alternative to the **create** command for creating mirrored volumes. Without any options, it creates a RAID-1 (mirrored) volume with two concatenated plexes. The largest contiguous space available on each drive is used to create the subdisks for the plexes. The first plex is built from the odd-numbered drives in the list, and the second plex is built from the even-numbered drives. If the drives are of different sizes, the plexes will be of different sizes.

If the **−s** option is provided, **mirror** builds striped plexes with a stripe size of 256 kB. The size of the subdisks in each plex is the size of the smallest contiguous storage available on any of the drives which form the plex. Again, the plexes may differ in size.

Normally, the **mirror** command creates an arbitrary name for the volume and its components. The name is composed of the text *vinum* and a small integer, for example *vinum3*. You can override this with the **−n** *name* option, which assigns the name specified to the volume. The plexes and subdisks are named after the volume in the default manner.

There is no choice of name for the drives. If the drives have already been initialized as **vinum** drives, the name remains. Otherwise the drives are given names starting with the text *vinumdrive* and a small integer, for example *vinumdrive7*. As with the **create** command, the **−f** option can be used to specify that a previous name should be overwritten. The **−v** is used to specify verbose output.

See the section SIMPLIFIED CONFIGURATION below for some examples of this command.

mv −f *drive object ...*

move −f *drive object ...*

Move all the subdisks from the specified objects onto the new drive. The objects may be subdisks, drives or plexes. When drives or plexes are specified, all subdisks associated with the object are moved.

The **−f** option is required for this function, since it currently does not preserve the data in the subdisk. This functionality will be added at a later date. In this form, however, it is suited to recovering a failed disk drive.

printconfig [file] Write a copy of the current configuration to file in a format that can be used to recreate the **vinum** configuration. Unlike the configuration saved on disk, it includes definitions of the drives. If you omit file, **vinum** writes the list to stdout.

quit Exit the **vinum** program when running in interactive mode. Normally this would be done by entering the *EOF* character.

read *disk* [disk...]

The **read** command scans the specified disks for **vinum** partitions containing previously created configuration information. It reads the configuration in order from the most recently updated to least recently updated configuration. **vinum** maintains an up-to-date copy of all configuration information on each disk partition. You must specify all of the slices in a configuration as the parameter to this command.

The **read** command is intended to selectively load a **vinum** configuration on a system which has other **vinum** partitions. If you want to start all partitions on the system, it is easier to use the **start** command.

If **vinum** encounters any errors during this command, it will turn off automatic configuration update to avoid corrupting the copies on disk. This will also happen if the configuration on disk indicates a configuration error (for example, subdisks which do not have a valid space specification). You can turn the updates on again with the **setdaemon** and **saveconfig** commands. Reset bit 2 (numerical value 4) of the daemon options mask to re-enable configuration saves.

rebuildparity *plex* [**−f**] [**−v**] [**−V**]

Rebuild the parity blocks on the specified RAID-4 or RAID-5 plex. This operation maintains a pointer in the plex, so it can be stopped and later restarted from the same position if desired. In addition, this pointer is used by the **checkparity** command, so rebuilding the parity blocks need only start at the location where the first parity problem has been detected.

If the **−f** flag is specified, **rebuildparity** starts rebuilding at the beginning of the plex. If the **−v** flag is specified, **rebuildparity** first checks the existing parity blocks prints information about those found to be incorrect before rebuilding. If the **−V** flag is specified, **rebuildparity** prints a running progress report.

rename [**−r**] [*drive* | *subdisk* | *plex* | *volume*] *newname*

Change the name of the specified object. If the **−r** option is specified, subordinate objects will be named by the default rules: plex names will be formed by appending .p***number*** to the volume name, and subdisk names will be formed by appending .s***number*** to the plex name.

resetconfig

The **resetconfig** command completely obliterates the **vinum** configuration on a system. Use this command only when you want to completely delete the configuration. **vinum** will ask for confirmation: you must type in the words NO FUTURE exactly as shown:

```
# vinum resetconfig

WARNING!  This command will completely wipe out your vinum
configuration.  All data will be lost.  If you really want
to do this, enter the text

NO FUTURE
Enter text -> NO FUTURE
Vinum configuration obliterated
```

As the message suggests, this is a last-ditch command. Don't use it unless you have an existing configuration which you never want to see again.

resetstats [**−r**] [volume | plex | subdisk]

vinum maintains a number of statistical counters for each object. See the header file `vinumvar.h` for more information. Use the **resetstats** command to reset these counters. In conjunction with the **−r** option, **vinum** also resets the counters of subordinate objects.

rm [**−f**] [**−r**] *volume* | *plex* | *subdisk*

rm removes an object from the **vinum** configuration. Once an object has been removed, there is no way to recover it. Normally **vinum** performs a large amount of consistency checking before removing an object. The **−f** option tells **vinum** to omit this checking and remove the object anyway. Use this option with great care: it can result in total loss of data on a volume.

Normally, **vinum** refuses to remove a volume or plex if it has subordinate plexes or subdisks respectively. You can tell **vinum** to remove the object anyway by using the **−f** option, or you can cause **vinum** to remove the subordinate objects as well by using the **−r** (recursive) option. If you remove a volume with the **−r** option, it will remove both the plexes and the subdisks which belong to the plexes.

saveconfig

Save the current configuration to disk. This is primarily a maintenance function. For example, if an error occurs on startup, updates will be disabled. When you reenable them, the configuration is not automatically saved to disk. Use this command to save the configuration.

setdaemon [value]

setdaemon sets a variable bitmask for the **vinum** dæmon. This command is temporary and will be replaced. Currently, the bit mask may contain the bits 1 (log every action to syslog) and 4 (don't update configuration). Option bit 4 can be useful for error recovery.

setstate *state* [*volume* | *plex* | *subdisk* | *drive*]

setstate sets the state of the specified objects to the specified state. This bypasses the usual consistency mechanism of **vinum** and should be used only for recovery purposes. It is possible to crash the system by incorrect use of this command.

start [**−i** *interval*] [**−S** *size*] [**−w**] [plex | subdisk]

start starts (brings into to the *up* state) one or more **vinum** objects.

If no object names are specified, **vinum** scans the disks known to the system for **vinum** drives and then reads in the configuration as described under the **read** commands. The **vinum** drive contains a header with all information about the data stored on the drive, including the names of the other drives which are required in order to represent plexes and volumes.

If **vinum** encounters any errors during this command, it will turn off automatic configuration update to avoid corrupting the copies on disk. This will also happen if the configuration on disk indicates a configuration error (for example, subdisks which do not have a valid space specification). You can turn the updates on again with the **setdaemon** and **saveconfig** command. Reset bit 4 of the daemon options mask to re-enable configuration saves.

If object names are specified, **vinum** starts them. Normally this operation is only of use with subdisks. The action depends on the current state of the object:

- If the object is already in the *up* state, **vinum** does nothing.
- If the object is a subdisk in the *down* or *reborn* states, **vinum** changes it to the *up* state.
- If the object is a subdisk in the *empty* state, the change depends on the subdisk. If it is part of a plex which is part of a volume which contains other plexes, **vinum** places the subdisk in the *reviving* state and attempts to copy the data from the volume. When the operation completes, the subdisk is set into the *up* state. If it is part of a plex which is part of a volume which contains no other plexes, or if it is not part of a plex, **vinum** brings it into the *up* state immediately.
- If the object is a subdisk in the *reviving* state, **vinum** continues the *revive* operation offline. When the operation completes, the subdisk is set into the *up* state.

When a subdisk comes into the *up* state, **vinum** automatically checks the state of any plex and volume to which it may belong and changes their state where appropriate.

If the object is a plex, **start** checks the state of the subordinate subdisks (and plexes in the case of a volume) and starts any subdisks which can be started.

To start a plex in a multi-plex volume, the data must be copied from another plex in the volume. Since this frequently takes a long time, it is normally done in the background. If you want to wait for this operation to complete (for example, if you are performing this operation in a script), use the **−w** option.

Copying data doesn't just take a long time, it can also place a significant load on the system. You can specify the transfer size in bytes or sectors with the **−S** option, and an interval (in milliseconds) to wait between copying each block with the **−i** option. Both of these options lessen the load on the system.

stop [**−f**] [volume | plex | subdisk]

If no parameters are specified, **stop** removes the **vinum** kld and stops vinum(8). This can only be done if no objects are active. In particular, the **−f** option does not override this requirement. Normally, the **stop** command writes the current configuration back to the drives before terminating. This will not be possible if configuration updates are disabled, so **vinum** will not stop if configuration updates are disabled. You can override this by specifying the **−f** option.

The **stop** command can only work if **vinum** has been loaded as a kld, since it is not possible to unload a statically configured driver. **vinum stop** will fail if **vinum** is statically configured.

If object names are specified, **stop** disables access to the objects. If the objects have subordinate objects, they subordinate objects must either already be inactive (stopped or in error), or the **−r** and **−f** options must be specified. This command does not remove the objects from the configuration. They can be accessed again after a **start** command.

By default, **vinum** does not stop active objects. For example, you cannot stop a plex which is attached to an active volume, and you cannot stop a volume which is open. The **−f** option tells **vinum** to omit this checking and remove the object anyway. Use this option with great care and understanding: used incorrectly, it can result in serious data corruption.

stripe [**−f**] [**−n** *name*] [**−v**] *drives*

The **stripe** command provides a simplified alternative to the **create** command for creating volumes with a single striped plex. The size of the subdisks is the size of the largest contiguous space available on all the specified drives. The stripe size is fixed at 256 kB.

Normally, the **stripe** command creates an arbitrary name for the volume and its components. The name is composed of the text *vinum* and a small integer, for example *vinum3*. You can override this with the **−n** *name* option, which assigns the name specified to the volume. The plexes and subdisks are named after the volume in the default manner.

There is no choice of name for the drives. If the drives have already been initialized as **vinum** drives, the name remains. Otherwise the drives are given names starting with the text *vinumdrive* and a small integer, for example *vinumdrive7*. As with the **create** command, the **−f** option can be used to specify that a previous name should be overwritten. The **−v** is used to specify verbose output.

See the section SIMPLIFIED CONFIGURATION below for some examples of this command.

SIMPLIFIED CONFIGURATION

This section describes a simplified interface to **vinum** configuration using the **concat**, **mirror** and **stripe** commands. These commands create convenient configurations for some more normal situations, but they are not as flexible as the **create** command.

See above for the description of the commands. Here are some examples, all performed with the same collection of disks. Note that the first drive, /dev/da1h, is smaller than the others. This has an effect on the sizes chosen for each kind of subdisk.

The following examples all use the **−v** option to show the commands passed to the system, and also to list the structure of the volume. Without the **−v** option, these commands produce no output.

Volume with a single concatenated plex

Use a volume with a single concatenated plex for the largest possible storage without resilience to drive failures:

```
vinum ->  concat -v /dev/da1h /dev/da2h /dev/da3h /dev/da4h
volume vinum0
  plex name vinum0.p0 org concat
drive vinumdrive0 device /dev/da1h
    sd name vinum0.p0.s0 drive vinumdrive0 size 0
drive vinumdrive1 device /dev/da2h
    sd name vinum0.p0.s1 drive vinumdrive1 size 0
```

```
drive vinumdrive2 device /dev/da3h
    sd name vinum0.p0.s2 drive vinumdrive2 size 0
drive vinumdrive3 device /dev/da4h
    sd name vinum0.p0.s3 drive vinumdrive3 size 0
V vinum0                State: up       Plexes:       1 Size:       2134 MB
P vinum0.p0           C State: up       Subdisks:     4 Size:       2134 MB
S vinum0.p0.s0          State: up       PO:         0  B Size:       414 MB
S vinum0.p0.s1          State: up       PO:       414 MB Size:       573 MB
S vinum0.p0.s2          State: up       PO:       988 MB Size:       573 MB
S vinum0.p0.s3          State: up       PO:      1561 MB Size:       573 MB
```

In this case, the complete space on all four disks was used, giving a volume 2134 MB in size.

Volume with a single striped plex

A volume with a single striped plex may give better performance than a concatenated plex, but restrictions on striped plexes can mean that the volume is smaller. It will also not be resilient to a drive failure:

```
vinum -> stripe -v /dev/da1h /dev/da2h /dev/da3h /dev/da4h
drive vinumdrive0 device /dev/da1h
drive vinumdrive1 device /dev/da2h
drive vinumdrive2 device /dev/da3h
drive vinumdrive3 device /dev/da4h
volume vinum0
  plex name vinum0.p0 org striped 256k
    sd name vinum0.p0.s0 drive vinumdrive0 size 849825b
    sd name vinum0.p0.s1 drive vinumdrive1 size 849825b
    sd name vinum0.p0.s2 drive vinumdrive2 size 849825b
    sd name vinum0.p0.s3 drive vinumdrive3 size 849825b
V vinum0                State: up       Plexes:       1 Size:       1659 MB
P vinum0.p0           S State: up       Subdisks:     4 Size:       1659 MB
S vinum0.p0.s0          State: up       PO:         0  B Size:       414 MB
S vinum0.p0.s1          State: up       PO:       256 kB Size:       414 MB
S vinum0.p0.s2          State: up       PO:       512 kB Size:       414 MB
S vinum0.p0.s3          State: up       PO:       768 kB Size:       414 MB
```

In this case, the size of the subdisks has been limited to the smallest available disk, so the resulting volume is only 1659 MB in size.

Mirrored volume with two concatenated plexes

For more reliability, use a mirrored, concatenated volume:

```
vinum -> mirror -v -n mirror /dev/da1h /dev/da2h /dev/da3h /dev/da4h
drive vinumdrive0 device /dev/da1h
drive vinumdrive1 device /dev/da2h
drive vinumdrive2 device /dev/da3h
drive vinumdrive3 device /dev/da4h
volume mirror setupstate
  plex name mirror.p0 org concat
    sd name mirror.p0.s0 drive vinumdrive0 size 0b
    sd name mirror.p0.s1 drive vinumdrive2 size 0b
  plex name mirror.p1 org concat
    sd name mirror.p1.s0 drive vinumdrive1 size 0b
    sd name mirror.p1.s1 drive vinumdrive3 size 0b
V mirror                State: up       Plexes:       2 Size:       1146 MB
```

```
P mirror.p0           C State: up       Subdisks:     2 Size:        988 MB
P mirror.p1           C State: up       Subdisks:     2 Size:       1146 MB
S mirror.p0.s0          State: up       PO:         0  B Size:        414 MB
S mirror.p0.s1          State: up       PO:       414 MB Size:        573 MB
S mirror.p1.s0          State: up       PO:         0  B Size:        573 MB
S mirror.p1.s1          State: up       PO:       573 MB Size:        573 MB
```

This example specifies the name of the volume: *mirror*. Since one drive is smaller than the others, the two plexes are of different size, and the last 158 MB of the volume is non-resilient. To ensure complete reliability in such a situation, use the **create** command to create a volume with 988 MB.

Mirrored volume with two striped plexes

Alternatively, use the **−s** option to create a mirrored volume with two striped plexes:

```
vinum -> mirror -v -n raid10 -s /dev/da1h /dev/da2h /dev/da3h /dev/da4h
drive vinumdrive0 device /dev/da1h
drive vinumdrive1 device /dev/da2h
drive vinumdrive2 device /dev/da3h
drive vinumdrive3 device /dev/da4h
volume raid10 setupstate
  plex name raid10.p0 org striped 256k
    sd name raid10.p0.s0 drive vinumdrive0 size 849825b
    sd name raid10.p0.s1 drive vinumdrive2 size 849825b
  plex name raid10.p1 org striped 256k
    sd name raid10.p1.s0 drive vinumdrive1 size 1173665b
    sd name raid10.p1.s1 drive vinumdrive3 size 1173665b
V raid10                State: up       Plexes:       2 Size:       1146 MB
P raid10.p0           S State: up       Subdisks:     2 Size:        829 MB
P raid10.p1           S State: up       Subdisks:     2 Size:       1146 MB
S raid10.p0.s0          State: up       PO:         0  B Size:        414 MB
S raid10.p0.s1          State: up       PO:       256 kB Size:        414 MB
S raid10.p1.s0          State: up       PO:         0  B Size:        573 MB
S raid10.p1.s1          State: up       PO:       256 kB Size:        573 MB
```

In this case, the usable part of the volume is even smaller, since the first plex has shrunken to match the smallest drive.

CONFIGURATION FILE

vinum requires that all parameters to the **create** commands must be in a configuration file. Entries in the configuration file define volumes, plexes and subdisks, and may be in free format, except that each entry must be on a single line.

Scale factors

Some configuration file parameters specify a size (lengths, stripe sizes). These values can be specified as bytes, or one of the following scale factors may be appended:

s	specifies that the value is a number of sectors of 512 bytes.
k	specifies that the value is a number of kilobytes (1024 bytes).
m	specifies that the value is a number of megabytes (1048576 bytes).
g	specifies that the value is a number of gigabytes (1073741824 bytes).

b is used for compatibility with VERITAS. It stands for blocks of 512 bytes. This abbreviation is confusing, since the word ''block'' is used in different meanings, and its use is deprecated.

For example, the value 16777216 bytes can also be written as **16m**, **16384k** or **32768s**.

The configuration file can contain the following entries:

drive *name devicename* [options]

Define a drive. The options are:

device *devicename* Specify the device on which the drive resides. *devicename* must be the name of a disk partition, for example `/dev/da1e` or `/dev/wd3s2h`, and it must be of type **vinum**. Do not use the **c** partition, which is reserved for the complete disk.

hotspare Define the drive to be a "hot spare" drive, which is maintained to automatically replace a failed drive. **vinum** does not allow this drive to be used for any other purpose. In particular, it is not possible to create subdisks on it. This functionality has not been completely implemented.

volume *name* [options]

Define a volume with name *name*.

Options are:

plex *plexname* Add the specified plex to the volume. If *plexname* is specified as ∗, **vinum** will look for the definition of the plex as the next possible entry in the configuration file after the definition of the volume.

readpol *policy* Define a *read policy* for the volume. *policy* may be either **round** or **prefer** *plexname*. **vinum** satisfies a read request from only one of the plexes. A *round* read policy specifies that each read should be performed from a different plex in *round-robin* fashion. A *prefer* read policy reads from the specified plex every time.

setupstate

When creating a multi-plex volume, assume that the contents of all the plexes are consistent. This is normally not the case, and correctly you should use the **init** command to first bring them to a consistent state. In the case of striped and concatenated plexes, however, it does not normally cause problems to leave them inconsistent: when using a volume for a file system or a swap partition, the previous contents of the disks are not of interest, so they may be ignored. If you want to take this risk, use this keyword. It will only apply to the plexes defined immediately after the volume in the configuration file. If you add plexes to a volume at a later time, you must integrate them.

Note that you *must* use the **init** command with RAID-5 plexes: otherwise extreme data corruption will result if one subdisk fails.

plex [options]

Define a plex. Unlike a volume, a plex does not need a name. The options may be:

name *plexname* Specify the name of the plex. Note that you must use the keyword *name* when naming a plex or subdisk.

org *organization* [stripesize]

Specify the organization of the plex. *organization* can be one of *concat*, *striped* or *raid5*. For *striped* and *raid5* plexes, the parameter *stripesize* must be specified, while for *concat* it must be omitted. For type *striped*, it specifies the width of each stripe. For type *raid5*, it specifies the size of a group. A group is a portion of a plex which stores the parity bits all in the same subdisk. It must be a factor of the plex size (in other words, the result of dividing the plex size by the stripe size must be an integer), and it must be a multiple of a disk sector (512 bytes).

For optimum performance, stripes should be at least 128 kB in size: anything smaller will result in a significant increase in I/O activity due to mapping of individual requests over multiple disks. The performance improvement due to the increased number of concurrent transfers caused by this mapping will not make up for the performance drop due to the increase in latency. A good guideline for stripe size is between 256 kB and 512 kB.

A striped plex must have at least two subdisks (otherwise it is a concatenated plex), and each must be the same size. A RAID-5 plex must have at least three subdisks, and each must be the same size. In practice, a RAID-5 plex should have at least 5 subdisks.

volume *volname* Add the plex to the specified volume. If no **volume** keyword is specified, the plex will be added to the last volume mentioned in the configuration file.

sd *sdname offset* Add the specified subdisk to the plex at offset *offset*.

subdisk [options]

Define a subdisk. Options may be:

name *name* Specify the name of a subdisk. It is not necessary to specify a name for a subdisk—see OBJECT NAMING above. Note that you must specify the keyword *name* if you wish to name a subdisk.

plexoffset *offset*

Specify the starting offset of the subdisk in the plex. If not specified, **vinum** allocates the space immediately after the previous subdisk, if any, or otherwise at the beginning of the plex.

driveoffset *offset*

Specify the starting offset of the subdisk in the drive. If not specified, **vinum** allocates the first contiguous *length* bytes of free space on the drive.

length *length* Specify the length of the subdisk. This keyword must be specified. There is no default, but the value 0 may be specified to mean
''use the largest available contiguous free area on the drive''.
If the drive is empty, this means that the entire drive will be used for the subdisk.
length
may be shortened to
len.

plex *plex* Specify the plex to which the subdisk belongs. By default, the subdisk belongs to the last plex specified.

drive *drive* Specify the drive on which the subdisk resides. By default, the subdisk resides on the last drive specified.

EXAMPLE CONFIGURATION FILE

```
# Sample vinum configuration file
#
# Our drives
drive drive1 device /dev/da1h
drive drive2 device /dev/da2h
drive drive3 device /dev/da3h
drive drive4 device /dev/da4h
drive drive5 device /dev/da5h
drive drive6 device /dev/da6h
# A volume with one striped plex
volume tinyvol
 plex org striped 512b
  sd length 64m drive drive2
  sd length 64m drive drive4
volume stripe
 plex org striped 512b
  sd length 512m drive drive2
  sd length 512m drive drive4
# Two plexes
volume concat
 plex org concat
  sd length 100m drive drive2
  sd length 50m drive drive4
 plex org concat
  sd length 150m drive drive4
# A volume with one striped plex and one concatenated plex
volume strcon
 plex org striped 512b
  sd length 100m drive drive2
  sd length 100m drive drive4
 plex org concat
  sd length 150m drive drive2
  sd length 50m drive drive4
```

```
# a volume with a  RAID-5 and a striped plex
# note that the RAID-5 volume is longer by
# the length of one subdisk
volume vol5
 plex org striped 64k
  sd length 1000m drive drive2
  sd length 1000m drive drive4
 plex org raid5 32k
  sd length 500m drive drive1
  sd length 500m drive drive2
  sd length 500m drive drive3
  sd length 500m drive drive4
  sd length 500m drive drive5
```

DRIVE LAYOUT CONSIDERATIONS

vinum drives are currently BSD disk partitions. They must be of type *vinum* in order to avoid overwriting data used for other purposes. Use **disklabel** *-e* to edit a partition type definition. The following display shows a typical partition layout as shown by **disklabel:**

```
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  a:    81920   344064    4.2BSD        0     0     0   # (Cyl.  240*- 297*)
  b:   262144    81920      swap                        # (Cyl.   57*- 240*)
  c:  4226725        0    unused        0     0         # (Cyl.    0 - 2955*)
  e:    81920        0    4.2BSD        0     0     0   # (Cyl.    0 - 57*)
  f:  1900000   425984    4.2BSD        0     0     0   # (Cyl.  297*- 1626*)
  g:  1900741  2325984     vinum        0     0     0   # (Cyl. 1626*- 2955*)
```

In this example, partition **g** may be used as a **vinum** partition. Partitions **a**, **e** and **f** may be used as **UFS** file systems or **ccd** partitions. Partition **b** is a swap partition, and partition **c** represents the whole disk and should not be used for any other purpose.

vinum uses the first 265 sectors on each partition for configuration information, so the maximum size of a subdisk is 265 sectors smaller than the drive.

LOG FILE

vinum maintains a log file, by default `/var/tmp/vinum_history`, in which it keeps track of the commands issued to **vinum**. You can override the name of this file by setting the environment variable `VINUM_HISTORY` to the name of the file.

Each message in the log file is preceded by a date. The default format is `%e %b %Y %H:%M:%S` See `strftime`(3) for further details of the format string. It can be overridden by the environment variable `VINUM_DATEFORMAT`.

HOW TO SET UP VINUM

This section gives practical advice about how to implement a **vinum** system.

Where to put the data

The first choice you need to make is where to put the data. You need dedicated disk partitions for **vinum**. They should be partitions, not devices, and they should not be partition **c**. For example, good names are `/dev/da0e` or `/dev/wd3s4a`. Bad names are `/dev/da0` and `/dev/da0s1`, both of which represent a device, not a partition, and `/dev/wd1c`, which represents a complete disk and should be of type **unused**. See the example under DRIVE LAYOUT CONSIDERATIONS above.

Designing volumes

The way you set up **vinum** volumes depends on your intentions. There are a number of possibilities:

1. You may want to join up a number of small disks to make a reasonable sized file system. For example, if you had five small drives and wanted to use all the space for a single volume, you might write a configuration file like:

   ```
   drive d1 device /dev/da2e
   drive d2 device /dev/da3e
   drive d3 device /dev/da4e
   drive d4 device /dev/da5e
   drive d5 device /dev/da6e
   volume bigger
    plex org concat
      sd length 0 drive d1
      sd length 0 drive d2
      sd length 0 drive d3
      sd length 0 drive d4
      sd length 0 drive d5
   ```

   In this case, you specify the length of the subdisks as 0, which means ''use the largest area of free space that you can find on the drive''. If the subdisk is the only subdisk on the drive, it will use all available space.

2. You want to set up **vinum** to obtain additional resilience against disk failures. You have the choice of RAID-1, also called ''mirroring'', or RAID-5, also called ''parity''.

   To set up mirroring, create multiple plexes in a volume. For example, to create a mirrored volume of 2 GB, you might create the following configuration file:

   ```
   drive d1 device /dev/da2e
   drive d2 device /dev/da3e
   volume mirror
    plex org concat
      sd length 2g drive d1
    plex org concat
      sd length 2g drive d2
   ```

   When creating mirrored drives, it is important to ensure that the data from each plex is on a different physical disk so that **vinum** can access the complete address space of the volume even if a drive fails. Note that each plex requires as much data as the complete volume: in this example, the volume has a size of 2 GB, but each plex (and each subdisk) requires 2 GB, so the total disk storage requirement is 4 GB.

   To set up RAID-5, create a single plex of type *raid5*. For example, to create an equivalent resilient volume of 2 GB, you might use the following configuration file:

   ```
   drive d1 device /dev/da2e
   drive d2 device /dev/da3e
   drive d3 device /dev/da4e
   drive d4 device /dev/da5e
   drive d5 device /dev/da6e
   volume raid
    plex org raid5 512k
      sd length 512m drive d1
      sd length 512m drive d2
   ```

```
      sd length 512m drive d3
      sd length 512m drive d4
      sd length 512m drive d5
```

RAID-5 plexes require at least three subdisks, one of which is used for storing parity information and is lost for data storage. The more disks you use, the greater the proportion of the disk storage can be used for data storage. In this example, the total storage usage is 2.5 GB, compared to 4 GB for a mirrored configuration. If you were to use the minimum of only three disks, you would require 3 GB to store the information, for example:

```
drive d1 device /dev/da2e
drive d2 device /dev/da3e
drive d3 device /dev/da4e
volume raid
 plex org raid5 512k
   sd length 1g drive d1
   sd length 1g drive d2
   sd length 1g drive d3
```

As with creating mirrored drives, it is important to ensure that the data from each subdisk is on a different physical disk so that **vinum** can access the complete address space of the volume even if a drive fails.

3. You want to set up **vinum** to allow more concurrent access to a file system. In many cases, access to a file system is limited by the speed of the disk. By spreading the volume across multiple disks, you can increase the throughput in multi-access environments. This technique shows little or no performance improvement in single-access environments. **vinum** uses a technique called ''striping'', or sometimes RAID-0, to increase this concurrency of access. The name RAID-0 is misleading: striping does not provide any redundancy or additional reliability. In fact, it decreases the reliability, since the failure of a single disk will render the volume useless, and the more disks you have, the more likely it is that one of them will fail.

   To implement striping, use a *striped* plex:

```
drive d1 device /dev/da2e
drive d2 device /dev/da3e
drive d3 device /dev/da4e
drive d4 device /dev/da5e
volume raid
 plex org striped 512k
   sd length 512m drive d1
   sd length 512m drive d2
   sd length 512m drive d3
   sd length 512m drive d4
```

   A striped plex must have at least two subdisks, but the increase in performance is greater if you have a larger number of disks.

4. You may want to have the best of both worlds and have both resilience and performance. This is sometimes called RAID-10 (a combination of RAID-1 and RAID-0), though again this name is misleading. With **vinum** you can do this with the following configuration file:

```
drive d1 device /dev/da2e
drive d2 device /dev/da3e
drive d3 device /dev/da4e
drive d4 device /dev/da5e
```

```
volume raid setupstate
 plex org striped 512k
   sd length 512m drive d1
   sd length 512m drive d2
   sd length 512m drive d3
   sd length 512m drive d4
 plex org striped 512k
   sd length 512m drive d4
   sd length 512m drive d3
   sd length 512m drive d2
   sd length 512m drive d1
```

Here the plexes are striped, increasing performance, and there are two of them, increasing reliablity. Note that this example shows the subdisks of the second plex in reverse order from the first plex. This is for performance reasons and will be discussed below. In addition, the volume specification includes the keyword *setupstate*, which ensures that all plexes are *up* after creation.

Creating the volumes

Once you have created your configuration files, start **vinum** and create the volumes. In this example, the configuration is in the file `configfile`:

```
# vinum create -v configfile
   1: drive d1 device /dev/da2e
   2: drive d2 device /dev/da3e
   3: volume mirror
   4:  plex org concat
   5:    sd length 2g drive d1
   6:  plex org concat
   7:    sd length 2g drive d2
Configuration summary

Drives:         2 (4 configured)
Volumes:        1 (4 configured)
Plexes:         2 (8 configured)
Subdisks:       2 (16 configured)

Drive d1:       Device /dev/da2e
                Created on vinum.lemis.com at Tue Mar 23 12:30:31 1999
                Config last updated Tue Mar 23 14:30:32 1999
                Size:      60105216000 bytes (57320 MB)
                Used:       2147619328 bytes (2048 MB)
                Available: 57957596672 bytes (55272 MB)
                State: up
                Last error: none
Drive d2:       Device /dev/da3e
                Created on vinum.lemis.com at Tue Mar 23 12:30:32 1999
                Config last updated Tue Mar 23 14:30:33 1999
                Size:      60105216000 bytes (57320 MB)
                Used:       2147619328 bytes (2048 MB)
                Available: 57957596672 bytes (55272 MB)
                State: up
                Last error: none
```

```
Volume mirror:  Size: 2147483648 bytes (2048 MB)
                State: up
                Flags:
                2 plexes
                Read policy: round robin

Plex mirror.p0: Size:   2147483648 bytes (2048 MB)
                Subdisks:        1
                State: up
                Organization: concat
                Part of volume mirror
Plex mirror.p1: Size:   2147483648 bytes (2048 MB)
                Subdisks:        1
                State: up
                Organization: concat
                Part of volume mirror

Subdisk mirror.p0.s0:
                Size:       2147483648 bytes (2048 MB)
                State: up
                Plex mirror.p0 at offset 0

Subdisk mirror.p1.s0:
                Size:       2147483648 bytes (2048 MB)
                State: up
                Plex mirror.p1 at offset 0
```

The **−v** option tells **vinum** to list the file as it configures. Subsequently it lists the current configuration in the same format as the **list −v** command.

Creating more volumes

Once you have created the **vinum** volumes, **vinum** keeps track of them in its internal configuration files. You do not need to create them again. In particular, if you run the **create** command again, you will create additional objects:

```
# vinum create sampleconfig
Configuration summary

Drives:         2 (4 configured)
Volumes:        1 (4 configured)
Plexes:         4 (8 configured)
Subdisks:       4 (16 configured)

D d1                State: up       Device /dev/da2e       Avail: 53224/57320 MB (92%)
D d2                State: up       Device /dev/da3e       Avail: 53224/57320 MB (92%)

V mirror            State: up       Plexes:       4 Size:       2048 MB

P mirror.p0       C State: up       Subdisks:     1 Size:       2048 MB
P mirror.p1       C State: up       Subdisks:     1 Size:       2048 MB
P mirror.p2       C State: up       Subdisks:     1 Size:       2048 MB
P mirror.p3       C State: up       Subdisks:     1 Size:       2048 MB
```

```
S mirror.p0.s0          State: up       PO:        0  B Size:       2048 MB
S mirror.p1.s0          State: up       PO:        0  B Size:       2048 MB
S mirror.p2.s0          State: up       PO:        0  B Size:       2048 MB
S mirror.p3.s0          State: up       PO:        0  B Size:       2048 MB
```

As this example (this time with the **-f** option) shows, re-running the **create** has created four new plexes, each with a new subdisk. If you want to add other volumes, create new configuration files for them. They do not need to reference the drives that **vinum** already knows about. For example, to create a volume `raid` on the four drives `/dev/da1e`, `/dev/da2e`, `/dev/da3e` and `/dev/da4e`, you only need to mention the other two:

```
drive d3 device /dev/da1e
drive d4 device /dev/da4e
volume raid
  plex org raid5 512k
    sd size 2g drive d1
    sd size 2g drive d2
    sd size 2g drive d3
    sd size 2g drive d4
```

With this configuration file, we get:

```
# vinum create newconfig
Configuration summary

Drives:         4 (4 configured)
Volumes:        2 (4 configured)
Plexes:         5 (8 configured)
Subdisks:       8 (16 configured)

D d1                    State: up       Device /dev/da2e        Avail: 51176/573
D d2                    State: up       Device /dev/da3e        Avail: 53220/573
D d3                    State: up       Device /dev/da1e        Avail: 53224/573
D d4                    State: up       Device /dev/da4e        Avail: 53224/573

V mirror                State: down     Plexes:       4 Size:       2048 MB
V raid                  State: down     Plexes:       1 Size:       6144 MB

P mirror.p0           C State: init     Subdisks:     1 Size:       2048 MB
P mirror.p1           C State: init     Subdisks:     1 Size:       2048 MB
P mirror.p2           C State: init     Subdisks:     1 Size:       2048 MB
P mirror.p3           C State: init     Subdisks:     1 Size:       2048 MB
P raid.p0            R5 State: init     Subdisks:     4 Size:       6144 MB

S mirror.p0.s0          State: up       PO:        0  B Size:       2048 MB
S mirror.p1.s0          State: up       PO:        0  B Size:       2048 MB
S mirror.p2.s0          State: up       PO:        0  B Size:       2048 MB
S mirror.p3.s0          State: up       PO:        0  B Size:       2048 MB
S raid.p0.s0            State: empty    PO:        0  B Size:       2048 MB
S raid.p0.s1            State: empty    PO:      512 kB Size:       2048 MB
S raid.p0.s2            State: empty    PO:     1024 kB Size:       2048 MB
S raid.p0.s3            State: empty    PO:     1536 kB Size:       2048 MB
```

Note the size of the RAID-5 plex: it is only 6 GB, although together its components use 8 GB of disk space. This is because the equivalent of one subdisk is used for storing parity data.

Restarting Vinum

On rebooting the system, start **vinum** with the **start** command:

```
# vinum start
```

This will start all the **vinum** drives in the system. If for some reason you wish to start only some of them, use the **read** command.

Performance considerations

A number of misconceptions exist about how to set up a RAID array for best performance. In particular, most systems use far too small a stripe size. The following discussion applies to all RAID systems, not just to **vinum**.

The FreeBSD block I/O system issues requests of between .5kB and 60 kB; a typical mix is somewhere round 8 kB. You can't stop any striping system from breaking a request into two physical requests, and if you do it wrong it can be broken into several. This will result in a significant drop in performance: the decrease in transfer time per disk is offset by the order of magnitude greater increase in latency.

With modern disk sizes and the FreeBSD I/O system, you can expect to have a reasonably small number of fragmented requests with a stripe size between 256 kB and 512 kB; with correct RAID implementations there is no obvious reason not to increase the size to 2 or 4 MB on a large disk.

When choosing a stripe size, consider that most current ufs file systems have cylinder groups 32 MB in size.

The easiest way to consider the impact of any transfer in a multi-access system is to look at it from the point of view of the potential bottleneck, the disk subsystem: how much total disk time does the transfer use? Since just about everything is cached, the time relationship between the request and its completion is not so important: the important parameter is the total time that the request keeps the disks active, the time when the disks are not available to perform other transfers. As a result, it doesn't really matter if the transfers are happening at the same time or different times. In practical terms, the time we're looking at is the sum of the total latency (positioning time and rotational latency, or the time it takes for the data to arrive under the disk heads) and the total transfer time. For a given transfer to disks of the same speed, the transfer time depends only on the total size of the transfer.

Consider a typical news article or web page of 24 kB, which will probably be read in a single I/O. Take disks with a transfer rate of 6 MB/s and an average positioning time of 8 ms, and a file system with 4 kB blocks. Since it's 24 kB, we don't have to worry about fragments, so the file will start on a 4 kB boundary. The number of transfers required depends on where the block starts: it's (S + F - 1) / S, where S is the stripe size in file system blocks, and F is the file size in file system blocks.

1. Stripe size of 4 kB. You'll have 6 transfers. Total subsystem load: 48 ms latency, 2 ms transfer, 50 ms total.
2. Stripe size of 8 kB. On average, you'll have 3.5 transfers. Total subsystem load: 28 ms latency, 2 ms transfer, 30 ms total.
3. Stripe size of 16 kB. On average, you'll have 2.25 transfers. Total subsystem load: 18 ms latency, 2 ms transfer, 20 ms total.
4. Stripe size of 256 kB. On average, you'll have 1.08 transfers. Total subsystem load: 8.6 ms latency, 2 ms transfer, 10.6 ms total.
5. Stripe size of 4 MB. On average, you'll have 1.0009 transfers. Total subsystem load: 8.01 ms latency, 2 ms transfer, 10.01 ms total.

It appears that some hardware RAID systems have problems with large stripes: they appear to always transfer a complete stripe to or from disk, so that a large stripe size will have an adverse effect on performance. **vinum** does not suffer from this problem: it optimizes all disk transfers and does not transfer unneeded data.

Note that no well-known benchmark program tests true multi-access conditions (more than 100 concurrent users), so it is difficult to demonstrate the validity of these statements.

Given these considerations, the following factors affect the performance of a **vinum** volume:

- Striping improves performance for multiple access only, since it increases the chance of individual requests being on different drives.
- Concatenating UFS file systems across multiple drives can also improve performance for multiple file access, since UFS divides a file system into cylinder groups and attempts to keep files in a single cylinder group. In general, it is not as effective as striping.
- Mirroring can improve multi-access performance for reads, since by default **vinum** issues consecutive reads to consecutive plexes.
- Mirroring decreases performance for all writes, whether multi-access or single access, since the data must be written to both plexes. This explains the subdisk layout in the example of a mirroring configuration above: if the corresponding subdisk in each plex is on a different physical disk, the write commands can be issued in parallel, whereas if they are on the same physical disk, they will be performed sequentially.
- RAID-5 reads have essentially the same considerations as striped reads, unless the striped plex is part of a mirrored volume, in which case the performance of the mirrored volume will be better.
- RAID-5 writes are approximately 25% of the speed of striped writes: to perform the write, **vinum** must first read the data block and the corresponding parity block, perform some calculations and write back the parity block and the data block, four times as many transfers as for writing a striped plex. On the other hand, this is offset by the cost of mirroring, so writes to a volume with a single RAID-5 plex are approximately half the speed of writes to a correctly configured volume with two striped plexes.
- When the **vinum** configuration changes (for example, adding or removing objects, or the change of state of one of the objects), **vinum** writes up to 128 kB of updated configuration to each drive. The larger the number of drives, the longer this takes.

Creating file systems on Vinum volumes

You do not need to run **disklabel** before creating a file system on a **vinum** volume. Just run **newfs**. Use the **-v** option to state that the device is not divided into partitions. For example, to create a file system on volume `mirror`, enter the following command:

```
# newfs -v /dev/vinum/mirror
```

A number of other considerations apply to **vinum** configuration:

- There is no advantage in creating multiple drives on a single disk. Each drive uses 131.5 kB of data for label and configuration information, and performance will suffer when the configuration changes. Use appropriately sized subdisks instead.
- It is possible to increase the size of a concatenated **vinum** plex, but currently the size of striped and RAID-5 plexes cannot be increased. Currently the size of an existing UFS file system also cannot be increased, but it is planned to make both plexes and file systems extensible.

STATE MANAGEMENT

Vinum objects have the concept of *state*. See `vinum`(4) for more details. They are only completely accessible if their state is *up*. To change an object state to *up*, use the **start** command. To change an object state to *down*, use the **stop** command. Normally other states are created automatically by the relationship between objects. For example, if you add a plex to a volume, the subdisks of the plex will be set in the *stale* state, indicating that, though the hardware is accessible, the data on the subdisk is invalid. As a result of this state, the plex will be set in the *faulty* state.

The 'reviving' state

In many cases, when you start a subdisk the system must copy data to the subdisk. Depending on the size of the subdisk, this can take a long time. During this time, the subdisk is set in the *reviving* state. On successful completion of the copy operation, it is automatically set to the *up* state. It is possible for the process performing the revive to be stopped and restarted. The system keeps track of how far the subdisk has been revived, and when the **start** command is reissued, the copying continues from this point.

In order to maintain the consistency of a volume while one or more of its plexes is being revived, **vinum** writes to subdisks which have been revived up to the point of the write. It may also read from the plex if the area being read has already been revived.

GOTCHAS

The following points are not bugs, and they have good reasons for existing, but they have shown to cause confusion. Each is discussed in the appropriate section above.

1. **vinum** drives are UNIX disk partitions and must have the partition type *vinum*. This is different from **ccd**, which expects partitions of type *4.2BSD*. This behaviour of ccd is an invitation to shoot yourself in the foot: with **ccd** you can easily overwrite a file system. **vinum** will not permit this.

   For similar reasons, the **vinum** *start* command will not accept a drive on partition *c*. Partition *c* is used by the system to represent the whole disk, and must be of type *unused*. Clearly there is a conflict here, which **vinum** resolves by not using the *c* partition.

2. When you create a volume with multiple plexes, **vinum** does not automatically initialize the plexes. This means that the contents are not known, but they are certainly not consistent. As a result, by default **vinum** sets the state of all newly-created plexes except the first to *stale*. In order to synchronize them with the first plex, you must **start** their subdisks, which causes **vinum** to copy the data from a plex which is in the *up* state. Depending on the size of the subdisks involved, this can take a long time.

   In practice, people aren't too interested in what was in the plex when it was created, and other volume managers cheat by setting them *up* anyway. **vinum** provides two ways to ensure that newly created plexes are *up*:

   - Create the plexes and then synchronize them with **vinum start**.
   - Create the volume (not the plex) with the keyword *setupstate*, which tells **vinum** to ignore any possible inconsistency and set the plexes to be *up*.

3. Some of the commands currently supported by **vinum** are not really needed. For reasons which I don't understand, however, I find that users frequently try the **label** and **resetconfig** commands, though especially **resetconfig** outputs all sort of dire warnings. Don't use these commands unless you have a good reason to do so.

4. Some state transitions are not very intuitive. In fact, it's not clear whether this is a bug or a feature. If you find that you can't start an object in some strange state, such as a *reborn* subdisk, try first to get it into *stopped* state, with the **stop** or **stop** *-f* commands. If that works, you should then be able to start it. If you find that this is the only way to get out of a position where easier methods fail, please report the situation.

5. If you build the kernel module with the *-DVINUMDEBUG* option, you must also build **vinum(8)** with the *-DVINUMDEBUG* option, since the size of some data objects used by both components depends on this option. If you don't do so, commands will fail with the message *Invalid argument*, and a

console message will be logged such as

```
vinumioctl: invalid ioctl from process 247 (vinum): c0e44642
```

This error may also occur if you use old versions of kld or userland program.

6. The **vinum** *read* command has a particularly emetic syntax. Once it was the only way to start **vinum**, but now the preferred method is with **vinum** *start*. **vinum** *read* should be used for maintenance purposes only. Note that its syntax has changed, and the arguments must be disk slices, such as `/dev/da0`, not partitions such as `/dev/da0e`.

FILES

/dev/vinum - directory with device nodes for **vinum** objects.
/dev/vinum/control - control device for **vinum**
/dev/vinum/plex - directory containing device nodes for **vinum** plexes.
/dev/vinum/sd - directory containing device nodes for **vinum** subdisks.

ENVIRONMENT

VINUM_HISTORY The name of the log file, by default /var/log/vinum_history.

VINUM_DATEFORMAT The format of dates in the log file, by default %e %b %Y %H:%M:%S.

EDITOR The name of the editor to use for editing configuration files, by default **vi**.

SEE ALSO

`strftime`(3), `vinum`(4), `disklabel`(8), `newfs`(8), `http://www.lemis.com/vinum.html`, `http://www.lemis.com/vinum-debugging.html`.

AUTHOR

Greg Lehey ⟨grog@lemis.com⟩.

HISTORY

The **vinum** command first appeared in FreeBSD 3.0. The RAID-5 component of **vinum** was developed for Cybernet Inc. `www.cybernet.com` for its NetMAX product.