The Cuddletech Veritas Volume Manager Series

Volume Kreation: VxMake & Her Seductive Ways

Ben Rockwood
CDLTK
Cuddletech

benr@cuddletech.com

This is the second installment of The Cuddletech Veritas Volume Manager Series. Covered topics include adding disks, working with objects using the vxmake command. Complete walkthroughs of building RAID0 and RAID0+1 volumes are presented featuring complete commands and output, so you know exactly what to expect when you start using the Veritas Volume Manager on real systems.

Yeah Baby!

Welcome to the VxMake tutorial. At this point it is assumed that you've worked through the previous two courses. At this point you should have a good grasp what the different type of Veritas objects are, how they work together, and the different classes of RAID. With this understood, lets jump into it.

The title of this section is titled after the exclamation I've screamed several times after having done the "impossible". That being described as something that everything and everyone else told me was impossible, and suddenly I made it work. More often than not this happens for me with Veritas. Because of peoples poor understanding of Veritas and her tools they have little faith in her mystical ways, and redeeming qualities and features. In the course we'll look at how to build real volumes. I've created several Vx volumes from scratch on a test system I have and documented EVERYTHING. We'll walk step by step through these examples to illustrate for you. We're going to do everything together, not like other Veritas volumes that just leave pieces out. You'll see what I saw when I created objects. This should also serve as a good reference until you get to a mid- to advanced Veritas user; at which point you will have all this memorized frontwards and backwards. But first lets talk about....

VxMake: Why it’s the "Kool" Way

This course directly correlates to its sister course "Volume Kreation: The VxAssist Way". But why two ways? I’ll let you read both courses and make up your mind which you prefer, but lets talk about why VxMake is kool.

Let me say it now, you don’t NEED to use vxmake. VxMake differs from vxassist in that you create objects individually, and you can specify EVERYTHING. VxAssist is nice because you can leave alot of your worries and decisions to Veritas, and not have to worry yourself. While there is a time and place for that, vxmake gives you fine grain control over your volume. In cases that you need a quick 20G volume that is for a small or temporary project, vxassist is nice because you can have your volume ready in minutes. However, if you need 30G+ using striping or mirroring or RAID5 that is critical data you will want to spend more time planning and allocating, in this case vxmake is the way to go.

Volume Lessons Background

First, let me give you some data on the system I’m using to do these examples. The system is a Sun Ultra2, running Solaris 8 with a single 296Mhz UltraSparcII chip and 512M of RAM. For the disks, I’m using 4 9G FC-AL (SCSI3) disks in a Sun A5100 Fibre Array connected to the system. I’m using Veritas 3.0.2. I’m using UFS for my filesystems. (Note: The SEVM name changed to just "Veritas Volume Manager" after the 2.6 release of SEVM. I am using the "Sun Supplied" version.) Here is some system output you might want:

```
# uname -a
SunOS nexus6 5.8 Beta_Refresh sun4u sparc SUNW,Ultra-2
# format
Searching for disks...done


AVAILABLE DISK SELECTIONS:
       0. c0t0d0 <SUN2.1G cyl 2733 alt 2 hd 19 sec 80>
          /sbus@1f,0/SUNW,fas@e,8800000/sd@0,0
       1. c0t1d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
          /sbus@1f,0/SUNW,fas@e,8800000/sd@1,0
       2. c2t0d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
          /sbus@1f,0/SUNW,socal@1,0/sf@0,0/ssd@w21000020370e0108,0
       3. c2t1d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
          /sbus@1f,0/SUNW,socal@1,0/sf@0,0/ssd@w2100002037163333,0
       4. c2t2d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
          /sbus@1f,0/SUNW,socal@1,0/sf@0,0/ssd@w210000203716d068,0
       5. c2t6d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
          /sbus@1f,0/SUNW,socal@1,0/sf@0,0/ssd@w2100002037169ef8,0
Specify disk (enter its number): ^D
#
```

[Notice that c0 is the internals. c2 is the A5100 disks]

The Pregame Show

This is the fuzzy part of the course. I don't have alot of examples so I'll leave some of this to imagination, but I want to talk about how you get disks ready to be used by Veritas. First of all, you'll need to have the disks you want to use attached to the system. When using disks in Veritas they DO NOT have to be on the same controller, or anything like that, any disk attached to the system will work. Once the disks are attached you'll need to make the OS aware of them, which on Solaris would be:

```
# drvconfig;devlinks;disks
```

After this, the "format" command should display your new disks in its output. If it is, the disk is now usable by the OS, and by extension, Veritas.

A quick note on partitioning. DO NOT PARTITION DISKS FOR USE WITH VERITAS!!! Why do I scream about it? Veritas does all that for you. Veritas wants a very simple partition layout, which is default on most disks. If the partition layout is wrong, Veritas will tell you when you try to bring it under Veritas control. If you need to repartition disks for Veritas you can do it with the format command, and simply restore one of the "default" layouts.

Now we need to bring the disk under the control of Veritas. Once Veritas gets a disk NO PART of that disk is usable outside of Veritas. This is why you can't partition disks before using Veritas, it doesn't want you trying to mess with ITS disks! (Veritas and the disk form a bit pact of something)

There are two ways to bring a disk under Veritas control, "encapsulation" and "initialization". They ARE different. Both methods get your disk ready for use in Veritas volumes. The difference is that "encapsulation" will preserver your data on the disks. Veritas just kind of gels around the data and assimilates the disk. Whereas with "initialization" the data is wiped free and brought into Veritas control. When initializing all data on the disk being initialized will me lost. Some times Veritas and it's manuals tend to blur the difference between the two, so be careful.

To add a disk to Veritas control you can use "vxdiskadm". This is a ascii-menu driven tool, which allows for some help assisted, interactive operations. VxDiskAdm is invoked with the command "/usr/sbin/vxdiskadm". It's output looks like this:

```
Volume Manager Support Operations
Menu: VolumeManager/Disk

 1      Add or initialize one or more disks
 2      Encapsulate one or more disks
 3      Remove a disk
 4      Remove a disk for replacement
 5      Replace a failed or removed disk
 6      Mirror volumes on a disk
 7      Move volumes from a disk
 8      Enable access to (import) a disk group
 9      Remove access to (deport) a disk group
 10     Enable (online) a disk device
 11     Disable (offline) a disk device
 12     Mark a disk as a spare for a disk group
 13     Turn off the spare flag on a disk
 list   List disk information
```

```
 ?      Display help about menu
 ??     Display help about the menuing system
 q      Exit from menus

Select an operation to perform:
```

To add a disk you'll need to choose with is more appropriate, option "1" (initialize), or "2" (encapsulate). For the disks I'm going to use, I'll just initialize.

Before initializing my new disks, it's a good idea to using the "list" operation in vxdiskadm, just to make sure that Veritas sees the disks too. Just type "list", and here is my output:

```
---------------------------------------------------------------
List disk information
Menu: VolumeManager/Disk/ListDisk

  Use this menu operation to display a list of disks.  You can
  also choose to list detailed information about the disk at
  a specific disk device address.

Enter disk device or "all" [<address>,all,q,?] (default: all) all

DEVICE       DISK         GROUP        STATUS
c0t0d0       -            -            error
c0t1d0       -            -            error
c2t0d0       disk01       rootdg       online
c2t1d0       disk02       rootdg       online
c2t2d0       disk03       rootdg       online
c2t6d0       -    -            online

Device to list in detail [<address>,none,q,?] (default: none)
```

Notice in the output that disks at t0, t1, and t2 have already been initialized, so they have a disk name. However, I haven't initialize c2t6d0. (Again, notice that Veritas isn't interested in partitions. Where's not specifying sX in the device name). The following is the full output of an initialization process of a new disk into Veritas using "vxdiskadm", option 1. This is kinda long, but notice how easy and helpful it is!

```
Add or initialize disks
Menu: VolumeManager/Disk/AddDisks

  Use this operation to add one or more disks to a disk group.  You can
  add the selected disks to an existing disk group or to a new disk group
  that will be created as a part of the operation. The selected disks may
  also be added to a disk group as spares. The selected disks may also
  be initialized without adding them to a disk group leaving the disks
  available for use as replacement disks.

  More than one disk or pattern may be entered at the prompt.  Here are
  some disk selection examples:

  all:          all disks
  c3 c4t2:      all disks on both controller 3 and controller 4, target 2
  c3t4d2:       a single disk
```

```
Select disk devices to add:
[<pattern-list>,all,list,q,?] list

DEVICE       DISK         GROUP        STATUS
c0t0d0       -            -            error
c0t1d0       -            -            error
c2t0d0       disk01       rootdg       online
c2t1d0       disk02       rootdg       online
c2t2d0       disk03       rootdg       online
c2t6d0       disk04       rootdg       online

Select disk devices to add:

Add or initialize disks
Menu: VolumeManager/Disk/AddDisks

  Use this operation to add one or more disks to a disk group.  You can
  add the selected disks to an existing disk group or to a new disk group
  that will be created as a part of the operation. The selected disks may
  also be added to a disk group as spares. The selected disks may also
  be initialized without adding them to a disk group leaving the disks
  available for use as replacement disks.

  More than one disk or pattern may be entered at the prompt.  Here are
  some disk selection examples:

  all:          all disks

Add or initialize disks
Menu: VolumeManager/Disk/AddDisks

  Use this operation to add one or more disks to a disk group.  You can
  add the selected disks to an existing disk group or to a new disk group
  that will be created as a part of the operation. The selected disks may
  also be added to a disk group as spares. The selected disks may also
  be initialized without adding them to a disk group leaving the disks
  available for use as replacement disks.

  More than one disk or pattern may be entered at the prompt.  Here are
  some disk selection examples:

  all:          all disks
  c3 c4t2:      all disks on both controller 3 and controller 4, target 2
  c3t4d2:       a single disk

Select disk devices to add:
[<pattern-list>,all,list,q,?] list

DEVICE       DISK         GROUP        STATUS
c0t0d0       -            -            error
c0t1d0       -            -            error
c2t0d0       disk01       rootdg       online
```

```
c2t1d0       disk02       rootdg       online
c2t2d0       disk03       rootdg       online
c2t6d0       -            -            online

Select disk devices to add:
[<pattern-list>,all,list,q,?] c2t6d0

  Here is the disk selected.  Output format: [Device_Name]

  c2t6d0

Continue operation? [y,n,q,?] (default: y)

  You can choose to add this disk to an existing disk group, a
  new disk group, or leave the disk available for use by future
  add or replacement operations.  To create a new disk group,
  select a disk group name that does not yet exist.  To leave
  the disk available for future use, specify a disk group name
  of "none".

Which disk group [<group>,none,list,q,?] (default: rootdg)

Use a default disk name for the disk? [y,n,q,?] (default: y)

Add disk as a spare disk for rootdg? [y,n,q,?] (default: n)

  The selected disks will be added to the disk group rootdg with
  default disk names.

  c2t6d0

Continue with operation? [y,n,q,?] (default: y)

  The following disk device appears to have been initialized already.
  The disk is currently available as a replacement disk.
  Output format: [Device_Name]

  c2t6d0

Use this device? [y,n,q,?] (default: y)

  The following disk you selected for use appears to already have
  been initialized for the Volume Manager.  If you are certain the
  disk has already been initialized for the Volume Manager, then you
  do not need to reinitialize the disk device.
  Output format: [Device_Name]

  c2t6d0

Reinitialize this device? [y,n,q,?] (default: y)

  Initializing device c2t6d0.
```

```
  Adding disk device c2t6d0 to disk group rootdg with disk
  name disk04.

Add or initialize other disks? [y,n,q,?] (default: n)
```

Easy right? Totally! Now that our disk is added to Veritas control, we can start using the disk. Since I'm using 4 disks, after all of them were added to Veritas control "vxprint" output looks like this:

```
# vxprint
Disk group: rootdg

TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg rootdg       rootdg       -        -        -        -        -       -

dm disk01       c2t0d0s2     -        17678493 -        -        -       -
dm disk02       c2t1d0s2     -        17678493 -        -        -       -
dm disk03       c2t2d0s2     -        17678493 -        -        -       -
dm disk04       c2t6d0s2     -        17678493 -        -        -       -
#
```

Understand the output? Vxprint is our window into the world of Veritas. You'll use it extensively. Every time we do something we'll want to check it with vxprint. Right now, notice that we have 4 disks, named "disk01", "disk02", "disk03", and "disk04". They are labeled with "dm", which represents a VM disk. The VM Disks are a member of the "rootdg" which is the default disk group. The length displayed in the fifth column is the VM disk length (aka size) in SECTORS! [Sectors can be converted to kilobytes by dividing by 2. On this system anyway.]

We've now created the foundation for our volumes to come. We've built VM disks, which are members of a diskgroup (the default in this case... nothing happens without a disk group). We can now build "subdisks" which will form our plex!

Volume Lesson1: A "Simple" RAID Volume

In this lesson we're going to build a Simple RAID. We're going to create it from our 4 9G drives which we added earlier. We're going to build this volume object by object, and we're going to put the objects together until we've built out volume. If it sounds like I'm talking about Lego's, it's not too far from the truth. It's very similar.

First, we need to great SubDisks from our VM Disks. We'll need these SubDisks to build a plex, later. As always, I'm going to start by verifying my current configuration with "vxprint" and then I'm going to create my subdisks. This creation will be done with one-line per subdisk creation. To do this creation I'm going to use, our hero, VxMake. The syntax for building a SubDisk from a VM Disk using VxMake is this:

```
vxmake sd <subdiskname> <VMDiskName>,<offset>,<len>
```

The syntax is pretty easy. Vxmake is the command, "sd" specifies that a subdisk is being created. "subdiskname" is the name that the new subdisk will be know as; typically this name is the name of the VM Disk is was created from (for reference) with a "-01" after it. The "-0X" would be number sequentially for each new subdisk created from that VM Disk. The "vmdiskname" is the name of the VM

Disk we want to create this subdisk from. "offset" would be used for, um, setting offset. We'll talk more about this later; generally you won't need it. "len" is the length of the disk in sectors. To allocate the whole VM Disk to the new subdisk you could just "cut-and-paste" the length from the vxprint output. That's it! That's all you need to know and do to create a subdisk. Let me interject now that creating subdisks is the Veritas way to "partition" disks. For instance, if you wanted to create 2 volumes, striped across 4 disks, and you only had 4 disks you could make 2 subdisks from each VM Disk and then use the "disk0?-01" subdisks for the first volume, and the "disk0?-02" subdisks for the second. I hope this helps clarify subdisks.

Now, let's see what it looks like to create our subdisks, I'm going to create 4 subdisks one after the other, after checking vxprint. Here it is:

```
# vxprint
Disk group: rootdg

TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg rootdg       rootdg       -        -        -        -        -       -

dm disk01       c2t0d0s2     -        17678493 -        -        -       -
dm disk02       c2t1d0s2     -        17678493 -        -        -       -
dm disk03       c2t2d0s2     -        17678493 -        -        -       -
dm disk04       c2t6d0s2     -        17678493 -        -        -       -
# vxmake sd disk01-01 disk01,0,17678493
# vxmake sd disk02-01 disk02,0,17678493
# vxmake sd disk03-01 disk03,0,17678493
# vxmake sd disk04-01 disk04,0,17678493
#
```

That's it! We've now made our subdisks! Notice the "len" is identical to the "Length" from the vxprint output. Also notice the naming conventions. Very simple huh? Further, I want to point out that vxmake doesn't output anything in return. If the operation was successful it will return a prompt. If there is an error you'll get the error message. When creating several (more than 6) subdisks at one time, make sure to check vxprint often in case you make a typo that is "syntactically correct". With that said, lets look at what vxprint outputs now:

```
# vxprint
Disk group: rootdg

TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg rootdg       rootdg       -        -        -        -        -       -

dm disk01       c2t0d0s2     -        17678493 -        -        -       -
dm disk02       c2t1d0s2     -        17678493 -        -        -       -
dm disk03       c2t2d0s2     -        17678493 -        -        -       -
dm disk04       c2t6d0s2     -        17678493 -        -        -       -

sd disk01-01    -            ENABLED  17678493 -        -        -       -
sd disk02-01    -            ENABLED  17678493 -        -        -       -
sd disk03-01    -            ENABLED  17678493 -        -        -       -
sd disk04-01    -            ENABLED  17678493 -        -        -       -
```

See? We're just building and building. Next, we need to build our plex. The plex is actually the trickiest part, because striping and RAID5 is done at the plex level, not the volume level (remember that the volume is just a dumb container for plexes.... plexes do the work). Because we're going to build a "simple" plex we don't need to give any fancy options. In this case, the syntax will be:

```
vxmake plex plexname
```

Vxmake is the command we're using. "plex" tells vxmake that we want to build a new plex. And "plexname" is the name of our plex. You can name the plex anything you want. However, normally 95% of people name plexes as the volume name, dash 01, after the volume they will be used in. This is completely voluntary and is merely helpful to increase readability when troubleshooting. In our example I'm going to name our plex "dataplex1". Here's what it looked like when I build the plex, followed by the vxprint output:

```
# vxmake plex dataplex1
# vxprint
Disk group: rootdg

TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg rootdg       rootdg       -        -        -        -        -       -

dm disk01       c2t0d0s2     -        17678493 -        -        -       -
dm disk02       c2t1d0s2     -        17678493 -        -        -       -
dm disk03       c2t2d0s2     -        17678493 -        -        -       -
dm disk04       c2t6d0s2     -        17678493 -        -        -       -

sd disk01-01    -            ENABLED  17678493 -        -        -       -
sd disk02-01    -            ENABLED  17678493 -        -        -       -
sd disk03-01    -            ENABLED  17678493 -        -        -       -
sd disk04-01    -            ENABLED  17678493 -        -        -       -

pl dataplex1    -            DISABLED 0        -        -        -       -
#
```

See the new plex in the vxprint output? Notice that there are no subdisks in it. With vxmake I can actually specify which subdisks should be assigned to the plex, but I want you to do this the long way. This is 100% manual!

To add the subdisks to our new plex, we're going to use the "vxsd" command, which is the subdisk control tool. Vxsd has all kinds of options and functions, but we're going to use it for "subdisk association". As the name suggests, we're merely means we're "associating" the subdisks, which implies we can later "disassociate" the subdisks. Get the idea? We'll look at this more later. The syntax used to associate disk using vxsd is:

```
vxsd assoc <plexname> <subdisk> <subdisk> .....
```

"vxsd" is the command. "assoc" means we're associating disks to a plex. "plexname" is the name of the plex to associate the subdisks with. And then the rest of the command is the list of subdisks we're associating, separated by spaces. I'm going to be associating all 4 of our disks, and then check my change with vxprint. Here's what it looks like:

```
# vxsd assoc dataplex1 disk01-01 disk02-01 disk03-01 disk04-01
```

```
# vxprint
Disk group: rootdg

TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg rootdg       rootdg       -        -        -        -        -       -

dm disk01       c2t0d0s2     -        17678493 -        -        -       -
dm disk02       c2t1d0s2     -        17678493 -        -        -       -
dm disk03       c2t2d0s2     -        17678493 -        -        -       -
dm disk04       c2t6d0s2     -        17678493 -        -        -       -

pl dataplex1    -            DISABLED 70713972 -        -        -       -
sd disk01-01    dataplex1    ENABLED  17678493 0        -        -       -
sd disk02-01    dataplex1    ENABLED  17678493 17678493 -        -       -
sd disk03-01    dataplex1    ENABLED  17678493 35356986 -        -       -
sd disk04-01    dataplex1    ENABLED  17678493 53035479 -        -       -
#
```

Looking good so far! Now that we're got a plex built, we only need to attach the now completed plex to a volume. We'll build a volume using vxmake, and the following syntax:

```
vxmake -U <usagetype> vol <volname> plex=<plexname>
```

This one is alittle more confusing. "vxmake" is the command we're using. Here we'll use an argument "-U" after which we specify the "usage type". We won't talk about usage types now, but for a volume to contain a file system you'll need to use the "fsgen" usage type. "vol" tells vxmake we're building a new volume. "volname" is the name we want our new volume to have. And this time, we are going to let vxmake attach the plex for us, instead of having to do it ourselves. We do this with "plex=" followed by the name(s) of the plex(es) we want to attach. We could attach multiple plexes by listing them separated by coma's (then you'd have a mirror!), but we're only going to attach our "dataplex1" plex. That's it. So lets build the volume and then run vxprint. Notice I'm going to run vxprint this type with the "-hrt" (which I remember as "dash hurt") which shows us much more detail, than normal vxprint output does. Let's look at it, and then we'll discus it:

```
# vxmake -U fsgen vol datavol plex=dataplex1
# vxprint -hrt
Disk group: rootdg

DG NAME         NCONFIG      NLOG     MINORS   GROUP-ID
DM NAME         DEVICE       TYPE     PRIVLEN  PUBLEN   STATE
V  NAME         USETYPE      KSTATE   STATE    LENGTH   READPOL   PREFPLEX
PL NAME         VOLUME       KSTATE   STATE    LENGTH   LAYOUT    NCOL/WID MODE
SD NAME         PLEX         DISK     DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE
SV NAME         PLEX         VOLNAME  NVOLLAYR LENGTH   [COL/]OFF AM/NM    MODE

dg rootdg       default      default  0        952738334.1025.nexus6

dm disk01       c2t0d0s2     sliced   3590     17678493 -
dm disk02       c2t1d0s2     sliced   3590     17678493 -
dm disk03       c2t2d0s2     sliced   3590     17678493 -
dm disk04       c2t6d0s2     sliced   3590     17678493 -
```

```
v  datavol      fsgen        DISABLED EMPTY    70713972 ROUND     -
pl dataplex1    datavol      DISABLED EMPTY    70713972 CONCAT    -        RW
sd disk01-01    dataplex1    disk01   0        17678493 0         c2t0d0   ENA
sd disk02-01    dataplex1    disk02   0        17678493 17678493  c2t1d0   ENA
sd disk03-01    dataplex1    disk03   0        17678493 35356986  c2t2d0   ENA
sd disk04-01    dataplex1    disk04   0        17678493 53035479  c2t6d0   ENA
#
```

Kool huh? Don't let all the vxprint headers generated because of the "-hrt" scare you. They are keys for each lines output. The two letter designations should start looking familiar, with the exception of the "SV" which is something you'll rarely see, and is for the advanced course. Anyway, What I wanted you to see is that with the "-hrt" on vxprint we see on the "pl" (plex) line of the output vxprint reports that "dataplex1" is a "CONCAT" or concatenated RAID (aka Simple RAID). Also notice that each time we put objects together that they regroup themselves in the vxprint output. This just makes everything easier and easier to read and understand quickly.

We've now got a volume! We now need to start the volume and create a filesystem on it, after which we can mount it. We can start a volume with the "vxvol" command (which is used for vx volume control, and also has tons of kool uses). The syntax is just:

```
vxvol start <volname>
```

I won't even bore you with the analysis, this is self explanatory. So let's just do it! We're going to start the volume and then look at vxprint (YES, AGAIN!!!!):

```
# vxvol start datavol
# vxprint -hrt
Disk group: rootdg

DG NAME         NCONFIG      NLOG     MINORS   GROUP-ID
DM NAME         DEVICE       TYPE     PRIVLEN  PUBLEN   STATE
V  NAME         USETYPE      KSTATE   STATE    LENGTH   READPOL   PREFPLEX
PL NAME         VOLUME       KSTATE   STATE    LENGTH   LAYOUT    NCOL/WID MODE
SD NAME         PLEX         DISK     DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE
SV NAME         PLEX         VOLNAME  NVOLLAYR LENGTH   [COL/]OFF AM/NM    MODE

dg rootdg       default      default  0        952738334.1025.nexus6

dm disk01       c2t0d0s2     sliced   3590     17678493 -
dm disk02       c2t1d0s2     sliced   3590     17678493 -
dm disk03       c2t2d0s2     sliced   3590     17678493 -
dm disk04       c2t6d0s2     sliced   3590     17678493 -

v  datavol      fsgen        ENABLED  ACTIVE   70713972 ROUND     -
pl dataplex1    datavol      ENABLED  ACTIVE   70713972 CONCAT    -        RW
sd disk01-01    dataplex1    disk01   0        17678493 0         c2t0d0   ENA
sd disk02-01    dataplex1    disk02   0        17678493 17678493  c2t1d0   ENA
sd disk03-01    dataplex1    disk03   0        17678493 35356986  c2t2d0   ENA
sd disk04-01    dataplex1    disk04   0        17678493 53035479  c2t6d0   ENA
#
```

Here's the big key to notice. The "KSTATE" (Kernel State) and "STATE" changed from "DISABLED" and "EMPTY", to "ENABLED" and "ACTIVE". By starting the volume it's suddenly jumped to life.

Having fun yet? I just love Veritas. Isn't this logical? If your sailing through this, your doing better than you image. Remember, this is the "hard" way!

Okey. It's show time. We've got a live volume. We need to create a filesystem on it, and mount it.

Like many other volume managers, the volumes are accessible via a device file in the "/dev" file tree. Veritas volumes can be found in /dev/vx/dsk. Let me show you:

```
# ls -al /dev/vx/dsk
total 6
drwxr-xr-x   3 root     root         512 Mar 29 23:37 .
drwxr-xr-x   6 root     other        512 Mar 10 17:32 ..
brw-------   1 root     root      67,  5 Mar 29 23:37 datavol
drwxr-xr-x   2 root     root         512 Mar 29 23:37 rootdg
#
```

Wondering about the "rootdg" showing up? Because our volume is in the "default" disk group (rootdg) we see it in the /dev/vx/dsk directory. If we created a new disk group, and built a volume in it, the volume would be in /dev/vx/dsk/dgname/volname. Simple enough.

Just like any other volume manager, we can access a volume just like a disk, and we see that /dev/vx/dsk/datavol is a block device, so we'll create our filesystem on it. I'm going to use newfs (which defaults UFS) to create the file system on the volume:

```
# newfs /dev/vx/dsk/datavol
newfs: construct a new file system /dev/vx/rdsk/datavol: (y/n)? y
/dev/vx/rdsk/datavol:   70713972 sectors in 34529 cylinders of 32 tracks, 64 sectors
        34528.3MB in 705 cyl groups (49 c/g, 49.00MB/g, 6144 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 100448, 200864, 301280, 401696, 502112, 602528, 702944, 803360, 903776,
 1004192, 1104608, 1205024, 1305440, 1405856, 1506272, 1606688, 1707104,
 1807520, 1907936, 2008352, 2108768, 2209184, 2309600, 2410016, 2510432,
 snipped out
#
```

Newfs created the filesystem, just like it would if we used a normal partitioned disk. Notice that the size of this filesystem is 34.5G, and we used 4 9G disks, so this is about right. Now we're ready to mount the filesystem, which is just like normal. If we wanted to mount our new volume to "/vxrocks", we'd use the mount command like this:

```
# mount /dev/vx/dsk/datavol /vxrocks
#
```

That's it!!!! We're all done! I have a harder time getting dressed in the morning! Again, this was "the hard way". Even though you might not always build volumes this way (I do...) you will at least now have a better understanding of how things are put together, and how objects work with each other.

Volume Lesson 2: Striped Volumes

Look at the following log, and see if you can see what's being done. I'm starting from clean vmdisks again. We'll talk about it afterwords, here goes:

```
# vxprint
Disk group: rootdg

TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg rootdg       rootdg       -        -        -        -        -       -

dm disk01       c2t0d0s2     -        17678493 -        -        -       -
dm disk02       c2t1d0s2     -        17678493 -        -        -       -
dm disk03       c2t2d0s2     -        17678493 -        -        -       -
dm disk04       c2t6d0s2     -        17678493 -        -        -       -
# vxmake sd disk01-01 disk01,0,17678493
# vxmake sd disk02-01 disk02,0,17678493
# vxmake sd disk03-01 disk03,0,17678493
# vxmake sd disk04-01 disk04,0,17678493
# vxmake plex vol01-01 layout=stripe ncolumn=4 stwidth=32k sd=disk01-01,disk02-
01,disk03-01,disk04-01
# vxprint -hrt
Disk group: rootdg

DG NAME         NCONFIG      NLOG     MINORS   GROUP-ID
DM NAME         DEVICE       TYPE     PRIVLEN  PUBLEN   STATE
V  NAME         USETYPE      KSTATE   STATE    LENGTH   READPOL   PREFPLEX
PL NAME         VOLUME       KSTATE   STATE    LENGTH   LAYOUT    NCOL/WID MODE
SD NAME         PLEX         DISK     DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE
SV NAME         PLEX         VOLNAME  NVOLLAYR LENGTH   [COL/]OFF AM/NM    MODE

dg rootdg       default      default  0        952738334.1025.nexus6

dm disk01       c2t0d0s2     sliced   3590     17678493 -
dm disk02       c2t1d0s2     sliced   3590     17678493 -
dm disk03       c2t2d0s2     sliced   3590     17678493 -
dm disk04       c2t6d0s2     sliced   3590     17678493 -

pl vol01-01     -            DISABLED -        70714077 STRIPE    4/64     RW
sd disk01-01    vol01-01     disk01   0        17678493 0/0       c2t0d0   ENA
sd disk02-01    vol01-01     disk02   0        17678493 1/0       c2t1d0   ENA
sd disk03-01    vol01-01     disk03   0        17678493 2/0       c2t2d0   ENA
sd disk04-01    vol01-01     disk04   0        17678493 3/0       c2t6d0   ENA
#
# vxmake -U fsgen vol vol01 plex=vol01-01
# vxprint -hrt
Disk group: rootdg

DG NAME         NCONFIG      NLOG     MINORS   GROUP-ID
DM NAME         DEVICE       TYPE     PRIVLEN  PUBLEN   STATE
V  NAME         USETYPE      KSTATE   STATE    LENGTH   READPOL   PREFPLEX
PL NAME         VOLUME       KSTATE   STATE    LENGTH   LAYOUT    NCOL/WID MODE
SD NAME         PLEX         DISK     DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE
SV NAME         PLEX         VOLNAME  NVOLLAYR LENGTH   [COL/]OFF AM/NM    MODE

dg rootdg       default      default  0        952738334.1025.nexus6

dm disk01       c2t0d0s2     sliced   3590     17678493 -
```

```
dm disk02       c2t1d0s2     sliced   3590     17678493 -
dm disk03       c2t2d0s2     sliced   3590     17678493 -
dm disk04       c2t6d0s2     sliced   3590     17678493 -

v  vol01        fsgen        DISABLED EMPTY    70713885 ROUND     -
pl vol01-01     vol01        DISABLED EMPTY    70714077 STRIPE    4/64     RW
sd disk01-01    vol01-01     disk01   0        17678493 0/0       c2t0d0   ENA
sd disk02-01    vol01-01     disk02   0        17678493 1/0       c2t1d0   ENA
sd disk03-01    vol01-01     disk03   0        17678493 2/0       c2t2d0   ENA
sd disk04-01    vol01-01     disk04   0        17678493 3/0       c2t6d0   ENA
#
# vxvol start vol01
# vxprint -hrt
Disk group: rootdg

DG NAME         NCONFIG      NLOG     MINORS   GROUP-ID
DM NAME         DEVICE       TYPE     PRIVLEN  PUBLEN   STATE
V  NAME         USETYPE      KSTATE   STATE    LENGTH   READPOL   PREFPLEX
PL NAME         VOLUME       KSTATE   STATE    LENGTH   LAYOUT    NCOL/WID MODE
SD NAME         PLEX         DISK     DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE
SV NAME         PLEX         VOLNAME  NVOLLAYR LENGTH   [COL/]OFF AM/NM    MODE

dg rootdg       default      default  0        952738334.1025.nexus6

dm disk01       c2t0d0s2     sliced   3590     17678493 -
dm disk02       c2t1d0s2     sliced   3590     17678493 -
dm disk03       c2t2d0s2     sliced   3590     17678493 -
dm disk04       c2t6d0s2     sliced   3590     17678493 -

v  vol01        fsgen        ENABLED  ACTIVE   70713885 ROUND     -
pl vol01-01     vol01        ENABLED  ACTIVE   70714077 STRIPE    4/64     RW
sd disk01-01    vol01-01     disk01   0        17678493 0/0       c2t0d0   ENA
sd disk02-01    vol01-01     disk02   0        17678493 1/0       c2t1d0   ENA
sd disk03-01    vol01-01     disk03   0        17678493 2/0       c2t2d0   ENA
sd disk04-01    vol01-01     disk04   0        17678493 3/0       c2t6d0   ENA
#
```

Okey! How did it look? Everything looks very similar to our first lesson when we were working with a Simple RAID. There is one difference... and only one! It was where we used VxMake to create the plex. Let's dissect this line. Here's the line I used:

```
# vxmake plex vol01-01 layout=stripe ncolumn=4 stwidth=32k \
 sd=disk01-01,disk02-01,disk03-01,disk04-01
```

This line may look scary, but its really tame. If you read my first course (RAID Theory) this will probably make sense already. Let's break it down..... The syntax is like this:

```
vxmake plex <plexname> layout=<layout> ncolumn=<#> stwidth=<width>
 sd=<subdisk>,<subdisk>,....
```

Okey, "vxmake" is the command we're using. "plex" is what we want vxmake to build for us. "plexname" is the name of our new plex will have, call it anything you like. "layout" specifies the plex layout type, or what you commonly think of as "RAID Type". Veritas supports several layout types, but the 3 most

common are concat, stripe, and raid5. The concat layout is a Simple RAID. The stripe layout is a RAID0. And the raid5 layout is RAID5 - go figure. We're using stripe, to form a striped RAID0. "ncloumns" is used to specify how many columns your striped volume with contain. For the meantime, just figure that you need 1 column per subdisk. In our case we're going to be using 4 subdisks, so we'll specify 4 columns. Next, "stwidth" specifies stripe width. We'll use a 32k stripe width, just for fun. (We'll get to these last two options in a minute). And last we have "sd=" followed by a list of the subdisks we want to have vxmake attach to the plex for us. Using the "sd=" option allows vxmake to just associate the subdisks by itself, so that we don't need to worry about using "vxsd" to manually associate them.

Lets talk about two concepts here: "ncolumn" and "stwidth". Remember why we want to stripe data. We want our data to be stretched across all 4 of our subdisks, so that all 4 are sharing the data load. In order to do this we need to tell Veritas how to divide the data. We do this by dividing the data into chunks, which is specified by "stwidth" and then sequentially placing on different disks, which is specified by "ncolumn". So, in our new plex, when we write data to our volume, the first 32k will be written to disk01-01. The second 32k will be written to disk02-01. The third 32k will be written to disk03-01. Then the fourth 32k will be written to disk04-01. The fifth chunk goes on disk01-01, and so on. We just keep "wrapping" data around our subdisks, evenly splitting the load. In this way, even a single 128k file will exist on all 4 of our disks! So, again, Stripe Width is the amount of data that should be written to a column before moving to the next, and a Column is the number of subdisks to be written to.

Side Note:

Just for safety sake, let me mention that you CAN actually have multiple disks per column. You do this by specifying an offset. This is something that vxassist really likes to do. It's not difficult to understand once you understand striping, and offsets, but just be aware that it does happen alot. The only rule with multiple disks per column is that each column must have the same number of disks. That meaning, if you have 2 subdisks in column 1, and you have a 4 column plex, all four columns must contain 2 subdisks per column. All the columns MUST be identical.

One more thing, look back at the final vxprint output from our example, and notice that the subdisks associated with our plex have numbers in the 7th column (under "STRIPE" in the plex line) that look like this "1/0". This is the "ncolumn/offset" pair. If the number is "1/0" this tells me that this subdisk is the second column (always start with zero!) and it has no offset. If the number was "5/23128" the subdisk would be the sixth subdisk, with an offset of 23128 sectors. This is important to look for, and something you'll only see with the "-hrt" option to vxprint.

At this point, I hope you're confident in building volumes, and working with Veritas objects. I hope you are seeing how they work together. In the "Advanced Veritas Theory" course we'll talk about more kool things you can do with this new found information, and how to work with common problems. We're going to leave RAID5 out of this course, for information on RAID5 and Veritas look at the "Volume Kreation: The VxAssist Way", vxassist is a much better tool for RAID5 creation. If you haven't read any of the Veritas manuals by now, please start flipping through them. Also start reading the manual pages, which are really truly great reading. (I keep them all printed and in a binder!)

Wasn't this easy? I just want to leave this course, by pointing out how "undifficult" Veritas is. It's just different and new, not hard at all. Don't ever let Veritas scare you... she's sweet and kind. Be gentle and kind to her and she'll always treat you right. If you have a problem admit that you probably don't know what you are doing, and study the materials in man pages and in guides. In all my workings with Veritas

99.9995% of the time I have problems in Veritas it's a personal problem, not Veritas' fault. I've become comfortable with this, understanding that Veritas is trying to help me, I just need to listen to what it tells me so that I can find the right and best solution for the problem... she wants me to stop and think, no go away. Be confident, and enjoy Veritas.