Introduction

JUST DO IT ALREADY!!!

VxMake is certainly the best way to build things. You know what you want, you know how you want it, and you want it done right. Right? What if you don’t k now exactly what you want. Even worse, what if you don’t care! Thats where vxassist comes in. You can directly build volumes without building the sub-objc cts and give vxassist just as much or little info as you like. The trade off? The less you tell vxassist the more it decides for you. This can be really b ad. Kinda like telling your kids to put away the groceries without telling them specifically where to put things... you might find places to put things in your kitchen you didn’t know were there.

There is a time and place for everything. Often this actually has to do with what you’re using and how much you care. On a Sun A5200 you might want to put only one volume per disk. You might want each disk in Array A to be perfectly mirrored to a disk in the same slot in Array B. But you might also be using a D1000 where you don’t care what goes where. For these reasons you may want to carefully decide which tool you should use for building. We’ll talk about a number of reasons and ways in which to use the tool. But remember that analogy of the kids in your kitchen, it’s pretty close to letting vxassist decide where to dump all your data.
VxAssist: Why it's the Lazy Way

This course directly correlates to its sister course "Volume Kreation: The VxMake Way". There we saw that we can achieve a huge amount of control over our volume creations by building each volume object by object until we have a well humming bucket for our data to live. But all too often we just need the data quickly, or we don’t care about the nitty gritty details, or, more likely, we simply don’t have the time necessary to broaden over the details of each volume. What’s needed is a simple, quick method of creation. Not to mention the fact that when we build each of our objects by hand we leave ourselves open to accidental errors which may be difficult to catch. Chief among all other reasons this is why vxAssist should be carefully considered as an alternative (if used properly!). A simple example of the dangers of manual creation: when you create a volume from the ground up you’ll quickly realize the need for a shell that has command history (such as the Borne Again SHELL). And you'll be humming along recalling the last command, changing the necessary bits and building along. At some point, I guarantee you’ll accidently name an object wrong doing this. And that little innocent typo, that may or may not cause an error, will cause you headaches. But, with that said, there are times and places for both. For instance, if you simply want a plex to use you’ll need to vxMake it yourself, and therefore you need a good foundation in using it. Furthermore, you can not properly understand the Volume Manager without having a good solid background in vxMake.

But what makes VxAssist such a "lazy" method. It’s the fact that you can, in one line, tell the Volume Manager to create you a, say, 20G volume on disks A, B, and C, and bang! with only that information you’ve got a functional usable ready for filesystem volume. Or you could even go so far as to only tell VxAssist to create you a 20G volume named "newvol" and not even tell it which disks to use, it’ll just find the space in it’s configured disks and go off on it’s merry way. Whatever you lack to tell VxAssist about the volume your creating it will decide for itself, but as we mentioned before, the less you decide the less control you have and the more likely you are to get something you didn’t want. In the examples we’ll go through below we’ll build far more complex volumes than we did in the VxMake course, but we’ll put heavy emphasis in using as much information as we can to get the job done right, the first time. Let’s hop into it.

Volume Lessons Background

Let’s take a quick look at the equipment we’ll be using for our examples below. I’m using a Sun Ultra1 200E with 128M of memory. For our test disks we’ll be using a Sun A5100 Fibre Channel disk array with 14 9G disks. This will provide enough disks to build some much larger structures than we did in the vxMake course using only 4 disks. We’ll be using the UFS file system, VERITAS Volume Manager version 3.0.3 on Solaris8. Here a quick peek at our disks and system for your reference:

```
# uname -a
SunOS gaff 5.8 Generic_108528-15 sun4u sparc SUNW,Ultra-1
# format
Searching for disks...done

AVAILABLE DISK SELECTIONS:
  0. c0t0d0 <SUN4.2G cyl 3880 alt 2 hd 16 sec 135>
      /sbus@1f,0/SUNW,fas@e,8800000/sd@0,0
  1. c0t1d0 <SUN4.2G cyl 3880 alt 2 hd 16 sec 135>
```
You’ll notice that SCSI controller 0 (c0) is the onboard SCSI controller for our internal 4.2G disks and SCSI controller 1 (c1) is our disks in the A5100. The split in the number of targets (c1t6 then we skip to c1t16) is due to the way A5100’s dish out SCSI target numbers. Let’s take another look at how we can add these disks into Volume Manager (VM) control, but using a different method than we used in the VxMake course.

The Pregame Show: Disk Groups and vxdiskadd

In the VxMake tutorial we added disks using the helpful "vxdiskadm" ncurses app. Something you’ll find is that most of the options you are given via "vxdiskadm" are not actually a part of the app, but rather it simply acts like a hub for them. This is really helpful for those of us with bad memories who just can’t remember the name or the args, but sometimes a quicker method is nice. Therefore we’ll add in our disks this time using the "vxdiskadd" command. But before we do that, let’s talk about Disk Groups (dg’s), a topic we avoided for simplicity sake in the VxMake tutorial, but mentioned in the Krash Kourse. Disk Groups are a method to segregate VM objects. Each system is required to have at least one disk group named "rootdg". You must have at least one disk in rootdg, whether you use it or not. The rootdg is used to store some fundamental information about the VM on the system so it’s not an option. However
we can create as many other disk groups as we like. Now, you might wonder why we don’t simply do
everything in rootdg all the time, like we did in the vxMake tutorial. There are two good reasons: first is
that each disk group is "self contained", if you add 5 disk to a disk group and create a volume you ONLY
have those 5 disks to work with (unless you add more to the DG, obviously) and therefore can be used as
a sort of logical seperation from the rest of your storage subsystem. The second reason is that you can
import and deport disk groups. When you import a disk group all volumes, disks, plexes, and any other
objects in that disk group become usable by the system. When you deport a disk group all the objects and
the disk group itself vanish from the system. If this seems vague, think of a file system; you create a
filesystem on a disk, create files on it, etc. When you mount the filesystem all the contents are available
and ready to use, the system is fully aware of that disks filesystem. But when you umount the filesystem
suddenly all that data sort of "disappears" back into the darkness of it’s disks... the data is there, you
know it is, but unless that file system was noted in the /etc/vfstab a passerby wouldn’t even know it
existed. This is similar to the concept of disk groups. To illustrate why import/deporting of disk groups
is so powerful, lets again draw an analogy to a filesystem on disk: when you umount a filesystem on disk
you suddenly have the option of taking that disk out of the system, putting it in another and by knowing
what type of filesystem it is, and it’s device number you can re-mount that filesystem on a completely
different system, which can be a real asset! You can do the same with disk groups! You can deport a disk
group full of volumes, then move the disks with that disk group (you must move ALL of them, if you
break up the disks it won’t import) to another system where you simply import the disk group, a start all
the volumes. You can see how useful that could be. For this reason, in this tutorial we will do all our
example using an added options to almost all VxVM commands (the -g option) which specifies the disk
group to work with. If you do not specify the disk group you are working with rootdg is assumed.

So let’s start setting up our test enviroment for this course. First let’s add our disks. We’re going to add
disks in the A5100 (all 14) and we’ll be adding them to a new disk group. We’ll need to check the
disks via "format", then use "vxdiskadd" to add them and put them in a disk group. Here we go............

# format
Searching for disks...done

AVAILABLE DISK SELECTIONS:

0. c0t0d0 <SUN4.2G cyl 3880 alt 2 hd 16 sec 135>
   /sbus81f,0/SUNW,fas@e,88000000/sd@0,0
1. c0t1d0 <SUN4.2G cyl 3880 alt 2 hd 16 sec 135>
   /sbus81f,0/SUNW,fas@e,88000000/sd@1,0
2. c1t0d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
   /sbus81f,0/SUNW,socal@1,0/sf@0,0/ssd@w2200002037096ef0,0
3. c1t1d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
   /sbus81f,0/SUNW,socal@1,0/sf@0,0/ssd@w2200002037096ef0,0
4. c1t2d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
   /sbus81f,0/SUNW,socal@1,0/sf@0,0/ssd@w2200002037096ef0,0
5. c1t3d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
   /sbus81f,0/SUNW,socal@1,0/sf@0,0/ssd@w2200002037096ef0,0
6. c1t4d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
   /sbus81f,0/SUNW,socal@1,0/sf@0,0/ssd@w2200002037096ef0,0
7. c1t5d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
   /sbus81f,0/SUNW,socal@1,0/sf@0,0/ssd@w2200002037096ef0,0
8. c1t6d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
   /sbus81f,0/SUNW,socal@1,0/sf@0,0/ssd@w2200002037096ef0,0
9. c1t7d0 <SUN9.0G cyl 4924 alt 2 hd 27 sec 133>
The Cuddletech Veritas Volume Manager Series

Specify disk (enter its number): ^D
#
# vxdiskadd cl
Add or initialize disks
Menu: VolumeManager/Disk/AddDisks

Here are the disks selected. Output format: [Device_Name]
c1t0d0 c1t16d0 c1t17d0 c1t18d0 c1t19d0 c1t1d0 c1t20d0 c1t21d0 c1t22d0
c1t2d0 c1t3d0 c1t4d0 c1t5d0 c1t6d0

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

You can choose to add these disks to an existing disk group, a new disk group, or you can leave these disks 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 disks available for future use, specify a disk group name of "none".

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

There is no active disk group named cuddledg.

Create a new group named cuddledg? [y,n,q,?] (default: y) y

Use default disk names for these disks? [y,n,q,?] (default: y) y

Add disks as spare disks for cuddledg? [y,n,q,?] (default: n) n

A new disk group will be created named cuddledg and the selected disks will be added to the disk group with default disk names.
c1t0d0 c1t16d0 c1t17d0 c1t18d0 c1t19d0 c1t1d0 c1t20d0 c1t21d0 c1t22d0
c1t2d0 c1t3d0 c1t4d0 c1t5d0 c1t6d0

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

The following disk devices appear to have been initialized already. The disks are currently available as replacement disks.
Output format: [Device_Name]
c1t0d0 c1t16d0 c1t17d0 c1t18d0 c1t19d0 c1t20d0 c1t21d0 c1t22d0
c1t2d0 c1t3d0 c1t4d0 c1t5d0 c1t6d0

Use these devices? [Y,N,S(elect),q,?] (default: Y) Y

The following disks you selected for use appear to already have been initialized for the Volume Manager. If you are certain the disks already have been initialized for the Volume Manager, then you do not need to reinitialize these disk devices.

Output format: [Device_Name]
c1t0d0 c1t16d0 c1t17d0 c1t18d0 c1t19d0 c1t20d0 c1t21d0 c1t22d0
c1t2d0 c1t3d0 c1t4d0 c1t5d0 c1t6d0

Reinitialize these devices? [Y,N,S(elect),q,?] (default: Y) Y

Initializing device c1t0d0.
Initializing device c1t16d0.
Initializing device c1t17d0.
Initializing device c1t18d0.
Initializing device c1t19d0.
Initializing device c1t20d0.
Initializing device c1t21d0.
Initializing device c1t22d0.
Initializing device c1t2d0.
Initializing device c1t3d0.
Initializing device c1t4d0.
Initializing device c1t5d0.
Initializing device c1t6d0.

Creating a new disk group named cuddledg containing the disk device c1t0d0 with the name cuddled01.

Adding disk device c1t16d0 to disk group cuddledg with disk name cuddled02.

Adding disk device c1t17d0 to disk group cuddledg with disk
Adding disk device c1t18d0 to disk group cuddledg with disk name cuddled04.

Adding disk device c1t19d0 to disk group cuddledg with disk name cuddled05.

Adding disk device c1t1d0 to disk group cuddledg with disk name cuddled06.

Adding disk device c1t20d0 to disk group cuddledg with disk name cuddled07.

Adding disk device c1t21d0 to disk group cuddledg with disk name cuddled08.

Adding disk device c1t22d0 to disk group cuddledg with disk name cuddled09.

Adding disk device c1t2d0 to disk group cuddledg with disk name cuddled10.

Adding disk device c1t3d0 to disk group cuddledg with disk name cuddled11.

Adding disk device c1t4d0 to disk group cuddledg with disk name cuddled12.

Adding disk device c1t5d0 to disk group cuddledg with disk name cuddled13.

Adding disk device c1t6d0 to disk group cuddledg with disk name cuddled14.

Goodbye.

Great! We’ve just verified and added our disks and created a new disk group for them in the process!

Some comments on what we did above. The format command and output is fairly obvious to all. But next we add the disks, and instead of using "vxdiskadm" like we did in the vxMake course, we opted to employ the "vxdiskadd" command instead. This command helps you add any disks given as it’s arguments, given by SCSI id’s. You can actually add disks in several ways.... Using the argument "all", which (duh) attempts to add all disks in the system to VxVM control. By specifying a controller number you can attempt to add all disks on a given controller, which is what we did (adding all disks on c1). The other method is to specify each disk individually (eg: vxdiskadd c1t3d0 c1t4d0 c1t5d0, which would add the 3 specified disks). I highly discourage anyone from using the "all" option.

Two other things I’d like you to notice in the above output is that I created the disk group by specifying the name of a disk group that didn’t exist. The VM took this as a hint to create it, which was confirmed by it’s next questions after I told it to use "cuddledg" as the DG to add the disks to: "Create a new group named cuddledg?". That it’s easy enough to get a nice new disk group to use. The second thing I wanted
The Cuddletech Veritas Volume Manager Series

you to notice is the little suttle warning given to us during the addition. Just before it actually initializes
the disks it warns that some of the disks have already been initialized before. This is important because
you may be inadvertently initializing disks in a deported DG or disks that were set aside after old VM
usage. In our case, we see the warning because I create and destroy volumes and vmdisks on this array
all the time, so it’s just a friendly reminder to think before we continue.

One more thing before we move on. Notice that even though there is a default (just hit enter) answer for
most questions we didn’t use it! ALWAYS ANSWER THE QUESTIONS! And if you dont’ you’ll just
like thoughs people on afterschool TV specials saying stupid things like “I never thought it’d happen to
me! I thought I was special!”. Just remember, mistakes get people fired from time to time, so don’t take
anything for granted that you don’t have to.

Now, let’s take a little "vxdisk list" peek at our disk configuration:

```
# vxdisk list
DEVICE TYPE DISK GROUP STATUS
-- -- -- -- --
c0t0d0s2 sliced - - error
c0t1d0s2 sliced rootdisk rootdg online
c1t0d0s2 sliced cuddled01 cuddledg online
c1t1d0s2 sliced cuddled06 cuddledg online
c1t2d0s2 sliced cuddled10 cuddledg online
c1t3d0s2 sliced cuddled11 cuddledg online
c1t4d0s2 sliced cuddled12 cuddledg online
c1t5d0s2 sliced cuddled13 cuddledg online
c1t6d0s2 sliced cuddled14 cuddledg online
c1t16d0s2 sliced cuddled02 cuddledg online
c1t17d0s2 sliced cuddled03 cuddledg online
c1t18d0s2 sliced cuddled04 cuddledg online
c1t19d0s2 sliced cuddled05 cuddledg online
c1t20d0s2 sliced cuddled07 cuddledg online
c1t21d0s2 sliced cuddled08 cuddledg online
c1t22d0s2 sliced cuddled09 cuddledg online
```

Something about this that bothers me is that the names of the disks aren’t very helpful. While we’re
discussing disks and disk groups let’s explore how easy it is to rename objects. Using the tool "vxedit" we
can easily rename any object. The only object that you should not rename is a volume, because the
volume name is also a block device (/dev/vx/dsk/diskgroup/volume) and changing the block device can
cause problems if you forget to update /etc/vfstab, not to mention the fact that you will need to umount
the volume before you can change the name. Let’s look at an example of how we can rename the disks in
a way that’s more helpful:

```
# vxedit -g cuddledg rename cuddled01 cuddle-f0
# vxedit -g cuddledg rename cuddled06 cuddle-f1
# vxedit -g cuddledg rename cuddled10 cuddle-f2
# vxedit -g cuddledg rename cuddled11 cuddle-f3
# vxedit -g cuddledg rename cuddled12 cuddle-f4
# vxedit -g cuddledg rename cuddled13 cuddle-f5
# vxedit -g cuddledg rename cuddled14 cuddle-f6
# vxedit -g cuddledg rename cuddled02 cuddle-r0
# vxedit -g cuddledg rename cuddled03 cuddle-r1
# vxedit -g cuddledg rename cuddled04 cuddle-r2
# vxedit -g cuddledg rename cuddled05 cuddle-r3
```
You can see that the syntax for vxedit is pretty simple for renaming, the syntax is:

```
vxedit -g <diskgroup> rename <object_name> <new_object_name>
```

In this way you can rename plexes, subdisks, vmdisks, you name it. The reason I’ve renamed the way I have is that in our case with an A5100 there are 7 disks in the front (f0-f6) and 7 in the rear (r0-r6) of the array. This gives me a quicker idea of which disks is where physically. But you must remember that this naming scheme is in no way perfect, because of the fact that a VM disks is not tied to a physical disk. I can tell VxVM to move a VM disk from one physical disk to another and the VM disk name won’t change, and because the VM disks name is a physical representation of where the disk is I’ll get the wrong impression. Therefore this naming convention only serves as a guide, not as a rule. Never trust a VM disks name. And the nice thing about vxedit is that you can do these renames online, and change them as often as you like. As always treat all VM commands with care.

Something to note in this course is that all of my commands will use a "-g" option which specifies the diskgroup to perform the action on (like you see in the vxedit syntax example above). You should ALWAYS specify the disk group, even if it’s rootdg that you’re working on. If you do for some reason make a stupid mistake you want the damage to be as contained as possible, and specifying the disk group will help accomplish that.

So now that we’ve got our test setup ready to play with, let’s take a preliminary look at our disk group and get on to the volume building!

```
# vxprint -g cuddledg -hrt
```

You can see that the syntax for vxprint is pretty simple for listing, the syntax is:

```
vxprint <diskgroup> -hrt
```

```
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
```
Volume Lesson 1: No Frills Volumes

In our first lesson we’ll build a very simple volume. Let’s make a 4G volume name “simplevol”. The syntax for this creation will look like this:

```
vxassist -g <diskgroup> -U <usagetype> make <volname> <size>
```

We discussed the use of "-g" earlier. The "-U" can specify two types of usage: gen and fsgen. In effect, “gen” is a raw volume (character device only) and “fsgen” is a regular volume (character and block device). The only times I see people use "gen" type volumes is in correlation with large database systems which sometimes use raw volumes rather than filesystems. We’ll use the type “fsgen” for this course. The directive “make” tells vxassist that we want to make a volume. The “volname” is the name we want to give to the volume we’re creating and “size” is the size of the volume to build. Size can be specified in the following forms: 1, 1m, 1g. That would be: 1 sector (512KB), 1 megabyte, and 1 gigabyte.

Now that we know the syntax, let’s build it:

```
# vxassist -g cuddledg -U fsgen make simplevol 4g
# vxprint -g cuddledg -hrt
```

```
Volume Lession1: No Frills Volumes

In our first lesson we’ll build a very simple volume. Let’s make a 4G volume name "simplevol". The syntax for this creation will look like this:

```
vxassist -g <diskgroup> -U <usagetype> make <volname> <size>
```

We discussed the use of "-g" earlier. The "-U" can specify two types of usage: gen and fsgen. In effect, "gen" is a raw volume (character device only) and "fsgen" is a regular volume (character and block device). The only times I see people use "gen" type volumes is in correlation with large database systems which sometimes use raw volumes rather than filesystems. We’ll use the type "fsgen" for this course. The directive "make" tells vxassist that we want to make a volume. The "volname" is the name we want to give to the volume we’re creating and "size" is the size of the volume to build. Size can be specified in the following forms: 1, 1m, 1g. That would be: 1 sector (512KB), 1 megabyte, and 1 gigabyte.

Now that we know the syntax, let’s build it:

```
# vxassist -g cuddledg -U fsgen make simplevol 4g
# vxprint -g cuddledg -hrt
```

```
```
The Cuddletech Veritas Volume Manager Series

dm cuddle-f3 c1t3d0s2 sliced 3590 17678493 -
dm cuddle-f4 c1t4d0s2 sliced 3590 17678493 -
dm cuddle-f5 c1t5d0s2 sliced 3590 17678493 -
dm cuddle-f6 c1t6d0s2 sliced 3590 17678493 -
dm cuddle-r0 c1t16d0s2 sliced 3590 17678493 -
dm cuddle-r1 c1t17d0s2 sliced 3590 17678493 -
dm cuddle-r2 c1t18d0s2 sliced 3590 17678493 -
dm cuddle-r3 c1t19d0s2 sliced 3590 17678493 -
dm cuddle-r4 c1t20d0s2 sliced 3590 17678493 -
dm cuddle-r5 c1t21d0s2 sliced 3590 17678493 -
dm cuddle-r6 c1t22d0s2 sliced 3590 17678493 -

v simplevol fsgen ENABLED ACTIVE 8388608 SELECT -
pl simplevol-01 simplevol ENABLED ACTIVE 8392167 CONCAT - RW
sd cuddle-f0-01 simplevol-01 cuddle-f0 0 8392167 0 c1t0d0 ENA
#

Talk about simple! We’ve now got a volume enabled and ready for a filesystem. You’ll notice that the volumes layout is "CONCAT", it’s on one disk, it used the first disk it deemed usable to create the volume and built all the subobject. The volume is even started for us! This is why vxassist is so popular and helpful, it’s very quick and easy.

Just for completeness let’s see a filesystem put on this puppy and put into use:

```
# newfs /dev/vx/rdsk/cuddledg/simplevol
newfs: /dev/vx/rdsk/cuddledg/simplevol last mounted as /test
newfs: construct a new file system /dev/vx/rdsk/cuddledg/simplevol: (y/n)? y
/dev/vx/rdsk/cuddledg/simplevol: 8388608 sectors in 4096 cylinders of 32 tracks, 64 sectors
4096.0MB in 84 cyl groups (49 c/g, 49.00MB/g, 6144 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
  32, 100448, 200896, 301280, 401696, 502112, 602528, 702944, 803360, 903776,
  1004192, 1104688, 1205024, 1305440, 1405856, 1506272, 1606688, 1707104,
  1807520, 1907936, 2008352, 2108768, 2209184, 2309600, 2410016, 2510432,
  2610848, 2711264, 2811680, 2912096, 3012512, 3112928, 3213392, 3313712,
  3412128, 3512544, 3612960, 3713376, 3813792, 3914208, 4014624, 4115040,
  4215456, 4315872, 4416288, 4516704, 4617120, 4717536, 4817952, 4918368,
  5018784, 5119200, 5219616, 5320032, 5420448, 5520864, 5621280, 5721696,
  5822112, 5922528, 6022944, 6123360, 6223776, 6324192, 6424608, 6525024,
  6625440, 6725856, 6826272, 6926688, 7027104, 7127520, 7227936, 7328352,
  7428768, 7529184, 7629600, 7729916, 7830332, 7930648, 8031064, 8131380,
  8231808, 8332224,
# mkdir /simple_vol
# mount -F ufs /dev/vx/dsk/cuddledg/simplevol /simple_vol/
# cd /simple_vol/
# touch new_file
# ls -al
```
```
total 20
drwxr-xr-x 33 root root 1024 Aug 11 23:23 ..
drwx------ 2 root root 8192 Aug 11 23:22 lost+found
-rw-r--r-- 1 root other 0 Aug 11 23:24 new_file
```
There we have it, from start to finish, from SCSI disks to a usable volume with filesystem and it’s even mounted. You’ll notice that the device path is in the form: /usr/vx(r)disk/<diskgroup>/<volumename>. Everything else should be familiar to you.

From this point on I will limit down the amount of output, such as only showing the volume information in the vxprint output instead of all of it, and we won’t cover creation of filesystems since you already know how to do that and every volume (regardless of how it’s created) is created in a similar fashion as above. With that said, let’s get onto the next lesson!

Volume Lesson 2: Striped Volumes

VxAssist will, by default, create concatenated (simple) volumes. If we had decided to create a 40G volumes in the first lesson it would have simply created a 40G concat using up as many disks as it needed. But simple volumes really don’t give us the speed and power that we really want from a volume, so in this lesson we’ll get just a little more specific with vxassist and great a large (40G) striped volume. Let’s look at the syntax:

```
vxassist -g <diskgroup> -U <usagetype> make <volname> <size> \layout=<layouttype> stwidth=<width> ncolumn=<#> <disk1> \<disk2> <disk3> ....
```

Alright, we discussed the first several options, and the others you’ll recall from the VxMake course when you made striped plexes. "layout" specifies the layout type for the volume we’re building (concat, stripe, mirror, RAID5, etc). The options "stwidth" and "ncolumn" are stripe specific options defining the width of the stripe unit for each column. The remaining options specify the vmdisks we want vxassist to use in building this volumes.

It should be noted that we can use as many or as few of these options as we like. For instance, we can make the layout "stripe" but not specify the stripe width or number of columns. We can specify the stripe width and stripe layout, but not the number of columns or disks to use. You should exercise some common sense when omitting options, thinking of what vxassist is going to need to be told to make it’s choice, but without being told it will do the best it can to meet your request. In the following examples we’ll build a striped volume using only the "layout" option, and letting vxassist figure the rest out.

```
# vxassist -g cuddledg -U fsgen make stripevol 40g layout=stripe
# vxprint -g cuddledg -hrt

DG NAME  NCONFIG  NLOG  MINORS  GROUP-ID
DM NAME  DEVICE  TYPE  PRIVLEN  PUBLLEN  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

(..... removed ............)
```

v stripevol fsgen ENABLED ACTIVE 83886080 SELECT stripevol-01
pl stripevol-01 stripevol ENABLED ACTIVE 83907654 STRIPE 7/128 RW
sd cuddle-f0-01 stripevol-01 cuddle-f0 0 11986758 0/0 c1t0d0 ENA
sd cuddle-f1-01 stripevol-01 cuddle-f1 0 11986758 1/0 c1t1d0 ENA
sd cuddle-f2-01 stripevol-01 cuddle-f2 0 11986758 2/0 c1t2d0 ENA
The Cuddletech Veritas Volume Manager Series

sd cuddle-f3-01 stripevol-01 cuddle-f3 0 11986758 3/0 clt3d0 ENA
sd cuddle-f4-01 stripevol-01 cuddle-f4 0 11986758 4/0 clt4d0 ENA
sd cuddle-f5-01 stripevol-01 cuddle-f5 0 11986758 5/0 clt5d0 ENA
sd cuddle-f6-01 stripevol-01 cuddle-f6 0 11986758 6/0 clt6d0 ENA

VxAssist decided that to create our volume it would use 7 disks, use a 64k stripe width (128 sectors). Because it doesn’t need the full size of each vmdisk it created 7 subdisks that are 11986758 sectors in length (5.9G). There this really comes to be different than if we had used vxmake to create our own striped volumes is that more than likely we’d have used 5 disks and used the full length of each disk, creating a volume about 45G in size, but vxassist instead gets as close to the desired volume size as possible. This, in and of itself, has advantages.

Let’s build another volume, but this time specify the stripe width, the number of columns and the vmdisks to use.

```bash
# vxassist -U fsgen -g cuddledg make stripevol2 40g layout=stripe stwidth=128k \
ncolumn=5 cuddle-f0 cuddle-f1 cuddle-f2 cuddle-f3 cuddle-f4
# vxprint -g cuddledg -hrt
```

In this example you can see that I told vxassist exactly what I wanted, and that’s exactly what I got. I hope that these two examples help you see just how vxassist can intelligently "fill in the gaps", but may not do what we assume. If you assumed that vxassist would build a 5 disk stripe for our 40g volumes you might have been surprised when you got a 7 disks stripe. This helps illustrate that if you decide to be indifferent about a layout decision you should be sure about it, otherwise you’ll be disappointed in what you get.

Because in this course concepts are much easier to grasp we’ll go even further than we could in the VxMake course. Let’s next jump into building mirrored volumes and RAID0+1 volumes using vxassist.
Volume Lesson 3: RAID1 and RAID0+1 Volumes

With the same ease we used to create simple and striped volumes we’ll now build mirrored volumes. We’ll start with a simple 10G mirrored volume so that we can see how a RAID1 volume will look when using 2 disks for each mirror. But first, let’s examine the syntax:

```bash
vxassist -g <diskgroup> -U <usagetype> make <volname> <size> \layout=<layouttype> <disk1> <disk2> <disk3> ...
```

This syntax is looking real repetitive isn’t it. In this case we’ll use the same options as usual, but we’ll specify the layout as "mirror-concat":

```bash
# vxassist -U fsgen -g cuddledg make mirrorvol3 10g layout=mirror-concat
# vxprint -g cuddledg -hrt
```

```
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

(..... removed ......)
```

```
v mirrorvol  fsgen  ENABLED  ACTIVE  20971520  SELECT  -
pl mirrorvol-01 mirrorvol  ENABLED  ACTIVE  20975031  CONCAT  -  RW
sd cuddle-f4-01 mirrorvol-01 cuddle-f4  0  3296538  0  c1t4d0  ENA
sd cuddle-r4-01 mirrorvol-01 cuddle-r4  0  17678493 3296538  c1t20d0  ENA
pl mirrorvol3-02 mirrorvol  ENABLED  ACTIVE  20975031  CONCAT  -  RW
sd cuddle-f5-01 mirrorvol-02 cuddle-f5  0  3296538  0  c1t5d0  ENA
sd cuddle-r3-01 mirrorvol-02 cuddle-r3  0  17678493 3296538  c1t19d0  ENA
```

The beauty here is that the mirroring is done all in one faul swoop. You can go so far as to even use an optional parameter after specifying the layout to vxassist which will create multiple mirrors! To do this use the option: "nmirror=X" where X is the number of mirrors (plexes) you want.

It should be noted, that you can also create a volume that isn’t mirrored and then mirror it later using the following syntax:

```bash
vxassist -g <diskgroup> mirror <volname>
```

This will create a mirror that is (hopefully) identical to the exsisting plex. Here’s an example of creating a 10g striped volumes and then mirroring it afterwards:

```bash
# vxassist -g cuddledg -U fsgen make stripevol 10g layout=stripe
# vxprint -g cuddledg -hrt stripevol
```

```
v stripevol  fsgen  ENABLED  ACTIVE  20971520  SELECT  stripevol-01
pl stripevol-01 stripevol  ENABLED  ACTIVE  20989653  STRIPE  7/128  RW
sd cuddle-f0-01 stripevol-01 cuddle-f0 0  2998485 0/0  c1t0d0  ENA
sd cuddle-f1-01 stripevol-01 cuddle-f1 0  2998485 1/0  c1t1d0  ENA
sd cuddle-f2-01 stripevol-01 cuddle-f2 0  2998485 2/0  c1t2d0  ENA
sd cuddle-f3-01 stripevol-01 cuddle-f3 0  2998485 3/0  c1t3d0  ENA
sd cuddle-f4-01 stripevol-01 cuddle-f4 0  2998485 4/0  c1t4d0  ENA
```
There are two important things to note about creating mirrored volumes. The first is that creating them takes a very very long time. In the example above (where we created a 10G mirrored volume) it took over 20 minutes for my little Ultra1 just to sync the mirrors. You might say to yourself, however, "hey, it’s a new volume, and therefore there is no data, and therefore there is nothing to sync!". Well, the logic is sound, but VxVM does in fact mirror it. There might be nothing in the volumes, but that nothingness is mirrored to the bit! The second thing to note is that in VxVM 3.x you should create simple mirrored volumes using the layout type "mirror-concat" and not simply "mirror". If you do define the layout as "mirror" you will get a "concat-mirror" not a "mirror-concat". The difference is that a "concat-mirror" is a type of layered volume (also known as ConcatPro), a type of volume that is beyond the scope of this course. You can find out more in the Cuddletech Kourse: Exploring Layered Volumes. But at this point just know that if you decide to use layout "mirror" with multiple disks you won’t get what you might have ordere. This problem does not exist in Vxvm 2.x, since layered volumes weren’t supported until 3.x.

Let’s look at one more example, taking a quick look at building a volume directly into a "mirror-stripe" layout. This will make a volume that looks very similar to the example above where we created a stripe and mirrored that stripe, except that we’re going to do it together. And for this example let’s also get a little more specific, since the 7 disk stripe for a 10g volume wasn’t what I had in mind:

```bash
# vxassist -g cuddledg -U fsgen make raid01-test 10g layout=mirror-stripe 
  ncolumns=3 stwidth=64k nmirror=2
# vxprint -g cuddledg -hrt raid01-test 
(... removed ...)
```

```bash
v raid01-test fsgen ENABLED ACTIVE 20971520 SELECT -
pl raid01-test-01 raid01-test ENABLED ACTIVE 20975165 STRIPE 3/128 RW
sd cuddle-f0-01 raid01-test-01 cuddle-f0 0 6991677 0/0 c1t0d0 ENA
sd cuddle-f1-01 raid01-test-01 cuddle-f1 0 6991677 1/0 c1t1d0 ENA
sd cuddle-f2-01 raid01-test-01 cuddle-f2 0 6991677 2/0 c1t2d0 ENA
```
The Cuddletech Veritas Volume Manager Series

pl raid01-test-02 raid01-test ENABLED ACTIVE 20975165 STRIPE 3/128 RW
sd cuddle-f3-01 raid01-test-02 cuddle-f3 0 6991677 0/0 c1t3d0 ENA
sd cuddle-f4-01 raid01-test-02 cuddle-f4 0 6991677 1/0 c1t4d0 ENA
sd cuddle-f5-01 raid01-test-02 cuddle-f5 0 6991677 2/0 c1t5d0 ENA
#

That’s a little more like it. I got what I wanted, and I didn’t care which disks it used, so by not specifying the vmdisks I left it to the VM to decide. If I had wanted to specify the disks I would have needed to specify 6 vmdisk’s, the first 3 vmdisks would be one plex, the other 3 vmdisks would be the second plex. Also, something that can be confusing at times is the "nmirror" option I tossed in the last example. You might be tempted to think "nmirror is set to 2, therefore there should be 2 mirrors plus the data, so that’s 3 plexes!", which is wrong. Start chanting "a plex is a mirror" over and over again. Setting "nmirror" to 2 means we get 2 plexes.

The options are staying the same, the syntax isn’t changing, but we’re building bigger, better, and kooler volumes! Now that we’ve covered Simple RAIDs, RAID0, RAID0+1, and RAID1 let’s move on to something that isn’t advisable to do by hand via vxmake (it’s arguable whether you can actually manually do it, in fact, few people would even bother to try), building RAID5 volumes.

Volume Lesson 4: RAID5 Volumes

RAID5 is pretty simple to deal with. You can get pretty specific about the volume setup using vxassist, but generally when you are using RAID5 you’re not looking to break any benchmarks. Therefore there is no more to it’s creation than specifying the layout as "raid5", and specifying any disks you want to use. The syntax again is:

```
vxassist -g <diskgroup> -U <usagetype> make <volname> <size> \nlayout=<layouttype> <disk1> <disk2> <disk3> ...
```

RAID5 does have some requirements, though, you must use at least 3 vmdisk (4 is preferable) to create a RAID5 volumes, of which 3 subdisks are used for the volume itself, and 1 subdisk is automatically created as a RAID5 log plex. A RAID5 log is used for volume metadata and parity logging. While you do not absolutely need a RAID5, you would be a fool to not use one, and vxassist just assumes give you on right away.

Let’s look at an example of a RAID5 volumes building built:

```
# vxassist -g cuddledg -U fsgen make raid5vol 4g layout=raid5
# vxprint -g cuddledg -hrt raid5vol
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
dm cuddle-f0 c1t0d0s2 sliced 3590 17678493 -
dm cuddle-f1 c1t1d0s2 sliced 3590 17678493 -
dm cuddle-f2 c1t2d0s2 sliced 3590 17678493 -
dm cuddle-f3 c1t3d0s2 sliced 3590 17678493 -
dm cuddle-f4 c1t4d0s2 sliced 3590 17678493 -
dm cuddle-f5 c1t5d0s2 sliced 3590 17678493 -
```
Here we see that vxassist decided to work with 6 disks for the actual volume, and then created a very small (1.5M) subdisk for a RAID5 log plex. We could alternately have specified the number of columns we wanted in the RAID5 volumes using the option "ncolumns", just like we did to control striped volumes in the previous examples. But remember, if you specify "ncolumns" as 5, and you want to specify disks to use, to specify 6 disks for use so that you don’t end up forcing vxassist to put the log plex on one of the data disks.

Here’s one more RAID5 example with a little more clarity than before:

```
# vxassist -g cuddledg -U fsgen make raid5vol 4g layout=raid5 ncolumns=3 cuddle-f0 cuddle-f1 cuddle-f2 cuddle-f3
# vxprint -g cuddledg -hrt raid5vol
```

Here, just what I asked for, ready to go. It’s VxVM awesome!
Volume Lesson 5: Volume resizing

Lets hit on one more topic, since there have been very few mind numbing concepts in this tutorial, as compared to the 10+ commands you had to learn, each with it’s own options, in the VxMake course. Lets talk about about what we can do to resize volumes and which tools can help us in this task.

At some point you are going to realize that your volume just isn’t big enough. At that point you’ll need to descide whether you should build a new volume and use both, or if you should just resize the existsing volume. You already know how you would go about creating a new vol, so let’s talk about resizing.

In our last example we created a RAID5 volume of 4g on 3 disks plus a log disk. I’ve suddenly descided that’s not going to cut it and I really need a 12g volume instead. But first I need to see how big I can make the volume before I need to add disks, so let’s look at our first tool’s syntax:

```
vxassist -g <diskgroup> maxgrow <volume> [ <disk1> <disk2> <disk3> ]
```

Using the "maxgrow" form of vxassist we can find out exactly how much head room we have left to use. Using vxassist maxgrow with only a volume name as an argument will tell us exactly how big the volume can get by utilizing any free space in the disk group. If instead I want a more specific idea of growth ideas I can pass arguments along with the base option, such as specifying the disks that I wish to use. Using the RAID5 vol created earlier let’s ask the VM just how big we can get this thing by first asking the max possible size, and then by listing the 3 vmdisks we’re using for the volume itself, so that we can find out how big we can grow without using any more disks.

```
# vxassist -g cuddledg maxgrow raid5vol
Volume raid5vol can be extended by 133038080 to 141426688 (69056Mb)
bash-2.03# vxassist -g cuddledg maxgrow raid5vol cuddle-f0 cuddle-f1 cuddle-f2
Volume raid5vol can be extended by 26968064 to 35356672 (17264Mb)
```

So we can grow to 70G using everything unused in the DG or up to 17.2G by growing the vol to consume fully the disks it’s already using. Early I mentioned needing a 12g volume, so growing to 17G and just filling the current disks seems like a good idea. In order to do this we can use another form of vxassist:

```
vxassist -g <diskgroup> growby <volume> <len_to_grow_by>
or
vxassist -g <diskgroup> growto <volume> <new_len>
```

Notice that we can grow the volume in two ways: either by growing by a certain amount, or growing to a certain size. Using the data we collected from the "maxgrow" check we can use either we like. I often prefer the "growto" method. Growby is nice when you just want to passify a DBA who is screaming "I need 2G NOW!!!!!" (not that DBA's do that).

Let’s use the "growto" method using the data from the maxgrow and see what happens:

```
# vxprint -g cuddledg -hrt raid5vol
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

dm cuddle-f0 clt0d0s2 sliced 3590 17678493 -
dm cuddle-f1 clt1d0s2 sliced 3590 17678493 -
dm cuddle-f2 clt2d0s2 sliced 3590 17678493 -
```
The Cuddletech Veritas Volume Manager Series

This is a very easy and "quick" method of growing volumes. There are of course other concerns about growing the filesystem as well, but that is beyond the scope of this doc and you will find great instructions on doing so in the VxFS Admin Guide or in the Solaris Administrators Collection for UFS.

It should also be noted that there are reciprocal commands for shrinking volumes using the vxassist methods "shrinkto" and "shrinkby" in the same manner we did above. For more information on using these take a look at the man page, but know that it's just as easy as we did in the last example.

Another vxassist method exists which allows you to completely augment your volumes, using "relayout". With relayout you can change the number of columns (which is important when adding or removing disks from a stripe or RAID5), change the plex type (from stripe to concat, and visa-versa, from stripe-mirror to RAID5, and visa-versa), and almost any other change you could want to make.

Relayout can also get you out of some hairy situations, and is a valuable tool, but not to be abused. Relayout is a course unto itself, so for information on it check out my Advanced Theory course and the VxVM Administrators Guide.

The Wrap Up

I hope that this course had helped make you feel at home with vxassist. You’ve seen the same syntax over and over and over, and I hope that it’s starting to feel second nature by now, which was the goal of
this course. There are a great deal more things that vxassist can do for us and our volumes than we’ve
done here, but you now have the proper firm basis to venture off into argument land.

A final reminder. Always be very careful what you do. Triple check your syntax, and plan things out
before you build and/or modify. One of vxassist’s greatest assets is that you can do alot while saying very
little, But the downside is that if that one command is formed improperly you can do some horrible
horrible damage that may or may not be easily reversible. If there is one thing that can get you fired in an
instant it’s blowing away a couple hundred gigs of data in one foul swoop, even if you do have a tape. Be
careful and be thoughtful, and you’ll do just fine. (You might have a heart attack at age 23 like I almost
did, but at least you won’t live in a cardboard box!)

Special Thanks!
A very special thank you to the fellows at AnySystem.com. The helped me with an A5100
which as what made this course possible. They are really great and gave me the arrays at a
great price..... If you can, check them out sometime.