Global Knowledge®

Expert Reference Series of White Papers

Understanding AIX Logical Volume Management

Understanding AIX Logical Volume Management

Iain Campbell, UNIX/Linux Open Systems Architect, eLearning Specialist

Introduction

Like many UNIX operating systems, AIX uses the concepts of Logical Volume Management (LVM) in its data management architecture. This white paper explains the specifics of the AIX LVM and provides some tips and best practice recommendations to get the most from your AIX disk storage.

LVM is a generalized approach to managing storage that is not specific to any operating system. All UNIX-like operating systems typically offer an LVM option; in AIX an LVM-based approach is the only way to manage storage. So understanding how AIX implements LVM is necessary for the AIX administrator.

Generally this is a good thing, however, as LVM offers a great deal of flexibility in managing storage while minimizing the need to take data out of production when restructuring is needed.

A Brief History of Logical Volumes

Permanent data storage normally uses hard disk drive (HDD), technology originally developed by IBM in the mid 1950s. Today HDDs are typically aggregated into arrays in storage appliances using various methods to provide redundancy in case of the failure of any single HDD. In this case the array, or a portion of the storage space in the array, will be presented to a client operating system as a Logical Unit (LUN). This is not yet LVM, however, as the LUN still appears to the operating system as a single, large-capacity storage device. For LVM purposes there is no functional difference between a local HDD and a LUN, so we will simply call either type of storage a disk.

LVM use arises when we consider how the client operating system is going to subdivide the space in a disk for its use. This subdivision is necessary as disk space may be needed for different things such as page space, file systems, a kernel dump device, or possibly raw data space for an application that does not want to use files to store its data.

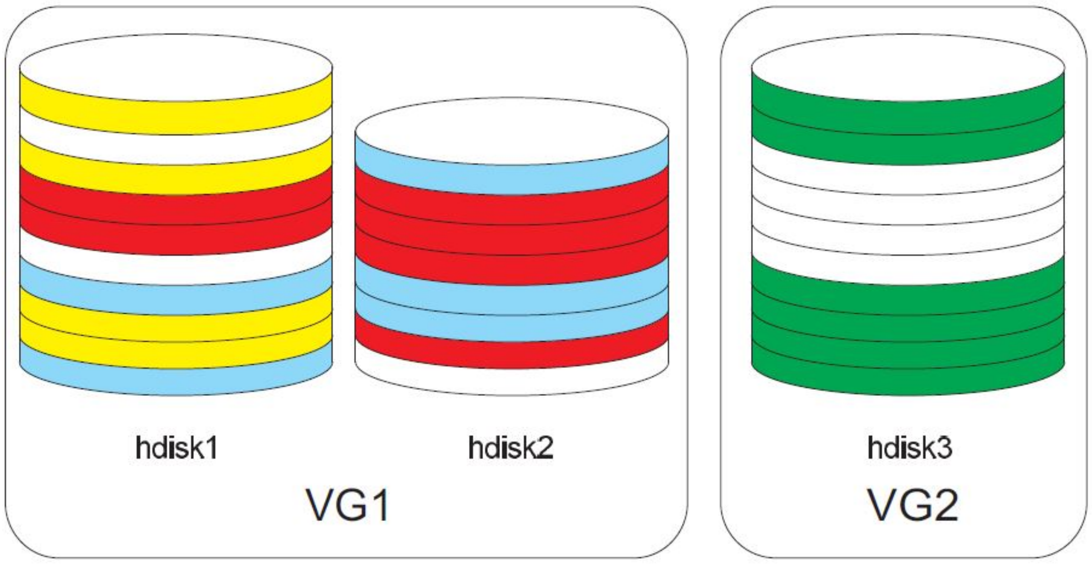
The majority of disk space is normally used for file storage, and there are many reasons to implement multiple file systems to store data in categories that are best handled differently. Each file system will then need to have space allotted from the disks available.

Early operating systems solved this problem by disk partitioning. This involved dividing each disk into a few fixed-size partitions, whose beginning and end points were defined by a partition table written to a reserved portion of each disk. This worked, but had limitations. The size of the partition table typically limited the number of partitions to eight or less. Also, as the partition table was read and stored in memory at boot, changes to it typically required either a reboot or at least the removal of the disk from production to reconfigure it. Either case required data to go out of production. Finally, partitions could not be dynamically resized, making it necessary to try to predict how much space would be needed in a partition as time passed. This was not always easy, and resulted in partitions that were either too large or too small. In this case, restructuring of disk partition tables was necessary, disrupting production.

The 1989 release of AIXv3 introduced the mandatory use of LVM to address these limitations. Next we will examine how this introduction addressed the problems of fixed partition allocation, and then look at the specifics of the AIX LVM implementation.

The Concept of Logical Volume Management

If partition tables divide disks into a small number of large partitions, the LVM reverses this idea, dividing disks into a large number of small partitions. To distinguish the difference, LVM schemes use the term physical extent, or in the case of AIX, physical partition (PP) to describe these small units of disk space. The disk itself is called a physical volume (PV). When multiple PVs are available they can be collected into volume groups (VGs).



In the figure above, we see three disks, hdisk1, hdisk2, and hdisk3. Each disk has been divided into PPs. If we say the PP size is 1 GB, then hdisk1 and hdisk3 are 10 GB in size and hdisk2 is 8 GB. Small numbers of PPs are used here for ease of illustration. In a real system these disks would be much larger—a single disk might contain thousands of PPs.

To allocate space to different purposes, several logical volumes (LVs) have been created, indicated by color. These LVs are summarized in the following list:

yellow_lv	4 GB
red_lv	6 GB
blue_lv	5 GB
free space	3 GB
VG1 Total	18 GB
green_lv	6 GB
free space	4 GB
VG2 Total	10 GB

We can see several things from the illustration:

- Any one LV need not be allocated contiguous space on a disk
- The space allocated to an LV can span disks (but not VGs)
- Not all space on a disk has to be allocated
- Disks in a VG need not be the same size

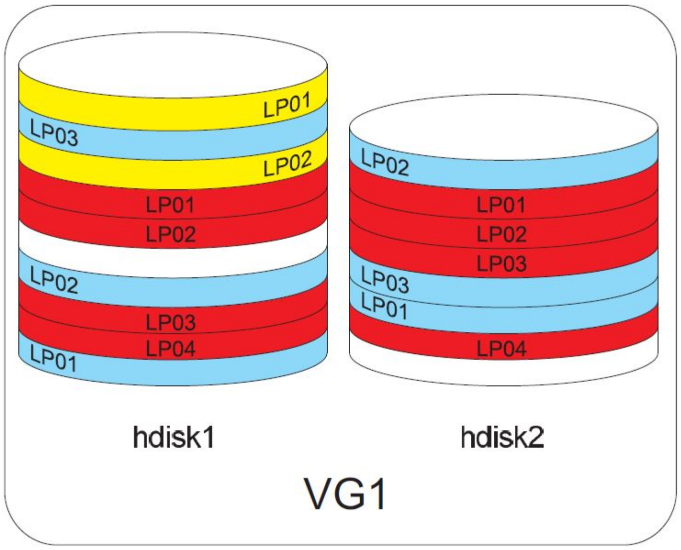
And there are several things that are also true, but are not obvious:

- A single LV will appear to its user as if it were a single contiguous disk partition
- Any one LV must reside entirely within a VG
- New LVs can be added dynamically
- The number of PPs allocated to any LV can be changed dynamically
- Disks in a VG can not only differ in size, but in all ways: they could be different types, SCSI vs. SAS, or could also be a mix of LUNs and local SCSI or SAS
- Disks can be dynamically added to or removed from the VG

This describes the basics of the LVM concept—all LVM implementations look more or less like this, at a minimum. Next we will examine how AIX adds additional features to the base, and then look at some specifics of the implementation.

The AIX LVM Implementation - RAID0 and RAID1

AIX adds a further element to the LVM structure, the logical partition (LP). This allows for the implementation of simple mirroring (i.e., RAID1 redundancy) within the LVM.



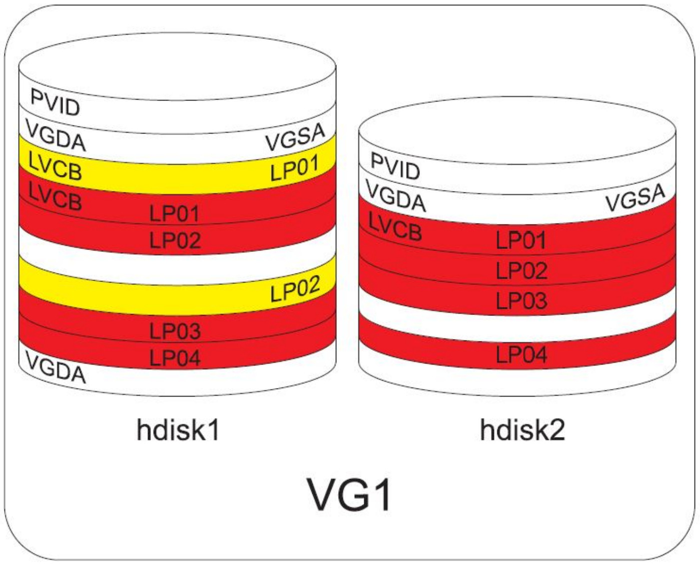
In the illustration above we can see three LVs. The blue LV contains three LPs, each of which is assigned a PP on a different PV. Each of those PPs is in fact a copy; in AIX terms this LV is said to be singly mirrored. In this case the data storage capacity of the LV is determined by the number of LPs, so this will be a 3 GB LV, however, it will require 6 PPs for a total of 6 GB of actual disk space.

Similarly the red LV is a singly mirrored 4 GB in size. The yellow LV has two LPs, but there is only one PP allocated to each; it is not mirrored. Note that in AIX we are not mirroring the disk, rather we mirror the LV. So disks that are involved in mirroring can be different sizes or types, and the allocation of PPs to LVs does not have to be identical, as seen in the illustration. We can also mix mirrored and unmirrored LVs. Although not illustrated, we can also have double mirrored LVs, in which three PPs (allocated across three different disks) would be allocated to each LP. Mirroring can also be dynamically changed. For example, an unmirrored LV can have one or two copies added to it dynamically (assuming sufficient free space is available in the VG) or can also have mirror copies dynamically removed.

The AIX LVM also allows for striping (i.e., RAID0 performance). In this case, the LV will be flagged as a striped LV at the time of creation. A minimum of two disks is required in the VG. The PPs allocated to this LV will then be evenly divided across as many PVs as requested when the LV is created; the number of disks specified is the stripe width. Also at creation time a stripe size is specified; typical sizes are 16 kB, 64 kB, or 128 kB. Data written to this LV is then striped such that once a stripe size quantity of data is written to the first PP, the next stripe size data chunk is written to the next PP, and so on. As these writes (or reads, as the case may be) are served by different disks, all IO operations for any complete stripe can in theory happen in parallel, yielding a significant increase in IO throughput.

LVM Metadata

The traditional disk partition table data structure can't accommodate the requirements of LVM. In AIX it is replaced by several structures. With few exceptions these exist on any disk used by AIX.



We will examine each of the major metadata components illustrated above.

The Physical Volume Identifier (PVID)

Each disk is allocated a unique Physical Volume Identifier (PVID). This is a 32-byte number allocated to a PV when it is first added to a VG. The goal of the PVID is to create a worldwide unique identifier for any AIX disk. A few things to know about the PVID:

- It is written to a predetermined hard-coded location at the beginning of the disk when the disk is first added to a VG
- It is not derived from the disk hardware, that is it is not related to a the serial number or any other disk specific hardware information
- The first 8 bytes of the PVID is the serial number of the system the PV was on at the time the PVID was created
- The second 8 bytes are derived from the time the PVID was created
- The last 16 bytes are zeroes, and can only be seen in the actual ODM object for the PV

The Volume Group Descriptor (VGDA)

The next metadata component is the volume group descriptor area (VGDA). This is the key data object containing a complete description of the VG. Because the VGDA is critical to the operation of the LVM, multiple redundant copies of the VGDA are maintained in case of disk failure. A single-disk VG will retain two copies of the VGDA, a dual-disk VG retains two copies on one disk and a third on the other disk. VGs having three or more PVs have a copy of the VGDA on each PV. In order to sustain IO operations on a VG the LVM must be able to access a quorum (defined as 50% + 1) of the VGDAs. If quorum is lost the entire VG will go offline (known as "varying off") and all its data will be unavailable.

A few more things to know about the VGDA:

- The VGDA keeps data tables detailing all PP allocations to LVs, as well as a number of optional parameters available to modify LV operations
- The VGDA contains the volume group identifier (VGID); similar to the PVID, a VGID is dynamically generated when a VG is first created and becomes a static worldwide unique identifier for the VG
- The VGDA does not contain the name of the VG; the name is merely an identifier chosen by the system hosting that VG, either at boot or when the VG is imported by the host (more on importing and exporting VGs in a moment)
- The VGDA contains a list of all PVIDs in the VG

Volume Group Types

There are three types of VG, driven primarily by history. The original LVM introduced in AIX V3 had a VGDA with data tables of fixed size. In time, disks grew in size as did the amount of data, and these tables were soon too small. AIX V4.3 increased the table sizes, calling these Big VGs. Then AIX V5.3 increased them yet again, calling these Scalable VGs. This table details the changes:

VG Type	Max PVs	Max LVs	Max PPs per VG	Max PP Size
Original	32	256	32,512	1 GB
Big	128	512	130,048	1 GB
Scalable	1024	4096	2,097,152	128 GB

Keep in mind that:

- If not specified at creation time, the default VG type is still Original, even in AIX V7
- An Original VG can be converted to a Big VG easily; there is no outage required and it takes only a moment
- A Big VG can be converted to a Scalable VG. Like converting from Original to Big the conversion itself does not take more than a few moments, but the Big VG needs to be taken out of production to do the conversion to Scalable

The Volume Group Status Area (VGSA)

The Volume Group Status Area (VGSA) contains state information and is primarily used by the LVM to keep track of mirrored LVs. A mirrored LV has an option to choose active mirror consistency checking. In this case write activity to the LV is monitored and recorded in the VGSA so that in the case of a system crash the LVM will know which mirror copies are in need of synchronization when the system comes back up. Without this the entire LV would need to be synced.

Keep in mind, however, that the primary use for the VGSA is for tracking mirroring. If there are no mirrored LVs in a VG, or the mirrored LVs choose passive or no mirror consistency checking, then the VGSA is little used.

The Logical Volume Control Block (LVCB)

The final piece of LVM metadata is the Logical Volume Control Block (LVCB). In Original and Big VGs this takes up the first 512 bytes of the LV and contains attributes of the LV such as the number of mirror copies. In cases where an LV is given to an application as raw data storage (once commonplace for databases) this location can be problematic if the application is not AIX-aware and does not deal properly with the existence of the LVCB. To avoid this in Scalable VGs, the LVCB is now located in the VGDA.

Volume Group Portability

The fact that all necessary LVM metadata is stored on the PVs themselves means that a VG is a portable entity. If we remove all of the disks of a VG from one server and attach them to another, all of the information necessary to recognize the storage configuration in that VG is carried in the VG metadata. It is only necessary to instruct the server inheriting the VG to scan a copy of the VGDA from any of the VG disks and it can reconstruct all it needs from there.

As this is AIX, the Object Data Manager (ODM)is involved, but all ODM objects necessary to access the data in the VG are created or deleted as necessary based on the information in the VGDA, so the ODM data is only ever a working copy of the disk-based LVM metadata. We will look at an example of VG portability in the final section of this paper.

Non LVM Disks

Normally any disk accessed on an AIX server would be a member of an AIX VG. AIX cannot use a disk for a file system, page space or dump device, nor can it make a raw LV available for application use unless the disk is in a VG. When AIX loads a driver for a disk, an application can do direct IO operations on that disk, but such applications are rare.

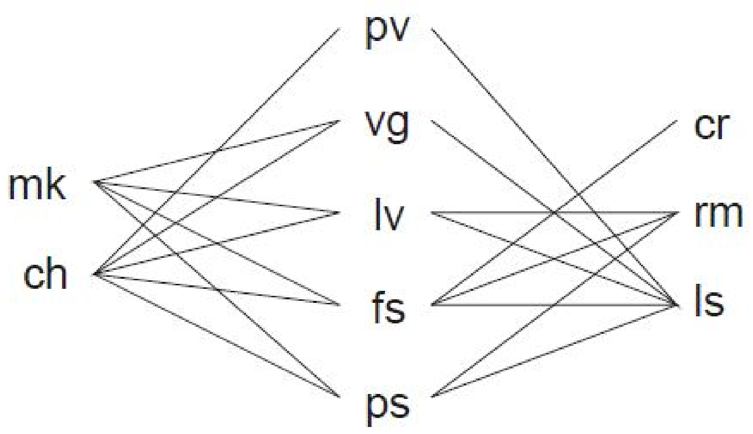
One of the few applications that does this is Oracle Automatic Storage Management (ASM). Interestingly, ASM is in fact an LVM and file system implementation on its own, essentially replacing the AIX LVM in functionality.

Another situation in which disks may appear to be used outside of the LVM is the case of a RAID-enabled local disk controller. In this case the physical disks in the array are identified by AIX as pdisk devices. The arrays configured using these pdisks are identified as the more familiar hdisks, and those disks are then seen as PVs by the LVM, and used in the normal fashion. So, the pdisks do in fact get used in an LVM context, although it is not immediately evident.

We will conclude by looking at the command set available to work with the LVM.

LVM Commands

The basic set of LVM commands can be built from this matrix:



While, as shown, not all of the logical combinations of action (on the left and right) and target (in the middle) are implemented, it works for most of the commands. For example, to make a VG, the command is `mkvg`; to list information about an LV, the command is `lslv`; and to change a file system, the command is `chfs`. Let us examine some examples of commands to list LVM information.

Listing LVM Information

The VG is the central entity in the LVM, so to list base information about a VG:

```
# lsvg rootvg
VOLUME GROUP:     rootvg            VG IDENTIFIER:
0000nnnn000000000000013cad7a50d7
VG STATE:         active            PP SIZE:        64 megabyte(s)
VG PERMISSION:    read/write        TOTAL PPs:      542 (34688 megabytes)
MAX LVs:          256               FREE PPs:       310 (19840 megabytes)
LVs:              12                USED PPs:       232 (14848 megabytes)
OPEN LVs:         11                QUORUM:         2 (Enabled)
TOTAL PVs:        1                 VG DESCRIPTORS: 2
STALE PVs:        0                 STALE PPs:      0
ACTIVE PVs:       1                 AUTO ON:        yes
MAX PPs per VG:   32512
MAX PPs per PV:   1016              MAX PVs:        32
LTG size (Dynamic):256 kilobyte(s)  AUTO SYNC:      no
HOT SPARE:        no                BB POLICY:      relocatable
PV RESTRICTION:   none              INFINITE RETRY: no
```

The `rootvg` is the VG created at install time on any AIX server. We can see from this that the `rootvg` on this server has one disk (`TOTAL PVs: 1`), and it uses a PP size of 64 MB (in the AIX LVM the PP size is an attribute of the VG—all PVs in a VG use the same PP size). The VG has a total storage space of just under 35 GB (`TOTAL PPs: 542`), of which a bit less than 15 GB has been allocated (`USED PPs: 232`) leaving just less than 20 GB still unused (`FREE PPs: 310`).

To list the LVs in the VG:

```
# lsvg -l rootvg
rootvg:
LV NAME             TYPE       LPs     PPs     PVs  LV STATE      MOUNT POINT
hd5                 boot       1       1       1    closed/syncd  N/A
hd6                 paging     5       5       1    open/syncd    N/A
hd8                 jfs2log    1       1       1    open/syncd    N/A
hd4                 jfs2       6       6       1    open/syncd    /
hd2                 jfs2       62      62      1    open/syncd    /usr
hd9var              jfs2       5       5       1    open/syncd    /var
hd3                 jfs2       2       2       1    open/syncd    /tmp
hd1                 jfs2       1       1       1    open/syncd    /home
hd10opt             jfs2       128     128     1    open/syncd    /opt
hd11admin           jfs2       2       2       1    open/syncd    /admin
lg_dumplv           sysdump    16      16      1    open/syncd    N/A
paging00            paging     3       3       1    open/syncd    N/A
```

This is what the `rootvg` is going to look like in most AIX servers. The LVs starting with `hd` are all of the AIX LVs created at install time. Note in this case there are several LVs used for purposes other than holding a file system. The boot LV (BLV) is `hd5`, and contains the AIX boot image. This includes the kernel as well as some ODM data and commands used to perform device configuration at boot time. As shown, this LV is normally in a closed state. The primary page space is always `hd6`, this particular VG also has a second user-defined page space LV named `paging00`. Finally, `hd8` is the journal log for file systems residing in the `rootvg`, and `lg_dumplv` is a kernel dump LV. The rest are file systems, as can be seen from the mount points listed in the far right column.
To list the PVs in the VG:

```
# lsvg -p rootvg
rootvg:
PV_NAME           PV STATE          TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk0            active            542         310
108..00..00..93..109
```

The free distribution list indicates where on the physical disk platter the 310 (in this case) free PPs are located. The five numbers indicate the number of PPs free in five regions of the disk, starting at the outer edge and moving through outer middle, middle, inner middle and inner edge, respectively. If the disk in question is a single local disk, then in theory this can be used to place data on the platter for optimal performance. However, it is much more likely that this disk is a LUN and then this information has no practical utility and can be disregarded.

To list information about a specific LV:

```
# lslv hd1
LOGICAL VOLUME:     hd1                     VOLUME GROUP:   rootvg
LV IDENTIFIER:      0000nnnn00000000000013cad7a50d7.8 PERMISSION:
read/write
VG STATE:           active/complete         LV STATE:       opened/syncd
TYPE:               jfs2                    WRITE VERIFY:   off
MAX LPs:            512                     PP SIZE:        64 megabyte(s)
COPIES:             1                       SCHED POLICY:   parallel
LPs:                1                       PPs:            1
STALE PPs:          0                       BB POLICY:      relocatable
INTER-POLICY:       minimum                 RELOCATABLE:    yes
INTRA-POLICY:       center                  UPPER BOUND:    32
```

```
MOUNT POINT:        /home                  LABEL:          /home
MIRROR WRITE CONSISTENCY: on/ACTIVE
EACH LP COPY ON A SEPARATE PV ?: yes
Serialize IO ?:     NO
INFINITE RETRY:     no
```

This LV is not mirrored (`COPIES: 1`, number of LPs = number of PPs—this makes sense as we already know there is only one PV in this VG) and is allowed to grow to a maximum size of 512 LPs (`MAX LPs: 512`, although this number can be dynamically increased as necessary within the limit imposed by the VG type, Original, Big, or Scalable), it contains a file system (`TYPE: jfs2`) which is currently mounted at /home (LV `STATE: opened/syncd, MOUNT POINT: /home`).

If we want to query PVs:

```
# lspv
hdisk0          0000nnnn9306e04b                    rootvg          active
hdisk1          0000nnnn74cb5311                    None
```

So `hdisk0` has PVID `0000nnnn9306e04b` and is a member of `rootvg`, which is currently varied on, `hdisk1`, PVID `0000nnnn74cb5311` is currently not assigned to a VG. The fact that `hdisk1` has a PVID implies that at some point it was a member of a VG, but isn't anymore.

To discover more about `hdisk0`:

```
# lspv hdisk0
PHYSICAL VOLUME:    hdisk0                   VOLUME GROUP:     rootvg
PV IDENTIFIER:      0000nnnn9306e04b VG IDENTIFIER
0000nnnn00000000000013cad7a50d7
PV STATE:           active
STALE PARTITIONS:   0                        ALLOCATABLE:      yes
PP SIZE:            64 megabyte(s)           LOGICAL VOLUMES:  12
TOTAL PPs:          542 (34688 megabytes)    VG DESCRIPTORS:   2
FREE PPs:           310 (19840 megabytes)    HOT SPARE:        no
USED PPs:           232 (14848 megabytes)    MAX REQUEST:      256 kilobytes
FREE DISTRIBUTION:  108..00..00..93..109
USED DISTRIBUTION:  01..108..108..15..00
MIRROR POOL:        None
```

And, to find out which LVs have allocations on a disk:

```
# lspv -l hdisk0
hdisk0:
LV NAME             LPs     PPs     DISTRIBUTION          MOUNT POINT
lg_dumplv           16      16      00..16..00..00..00    N/A
paging00            3       3       00..00..03..00..00    N/A
hd11admin           2       2       00..00..02..00..00    /admin
hd8                 1       1       00..00..01..00..00    N/A
hd6                 5       5       00..05..00..00..00    N/A
hd2                 62      62      00..00..53..09..00    /usr
hd4                 6       6       00..00..06..00..00    /
hd3                 2       2       00..00..02..00..00    /tmp
hd9var              5       5       00..00..04..01..00    /var
hd10opt             128     128     00..87..36..05..00    /opt
hd1                 1       1       00..00..01..00..00    /home
hd5                 1       1       01..00..00..00..00    N/A
```

The DISTRIBUTION column again refers to the region of the physical disk platter; not usually relevant for virtual disks or LUNs.

Now, let us examine some commands can we use to manipulate the LVM.

Creating LVM Entities

Previously we saw that hdisk1 is not a member of a VG; we could add it to the rootvg:

```
# extendvg rootvg hdisk1
0516-1398 extendvg: The physical volume hdisk1, appears to belong to
another volume group. Use the force option to add this physical volume
to a volume group.
0516-792 extendvg: Unable to extend volume group.
# extendvg -f rootvg hdisk1
# lspv
hdisk0          0000nnnn9306e04b                    rootvg          active
hdisk1          0000nnnn74cb5311                    rootvg          active
```

It was noted earlier that hdisk1 already had a PVID, indicating that this disk was previously a member of a VG. The extending command sees the old VGDA and requires the use of the force option to the extending command to add the disk to rootvg. If we re-examine rootvg we will see the effect of the addition:

```
# lsvg rootvg
VOLUME GROUP:       rootvg              VG IDENTIFIER:
0000nnnn000000000000013cad7a50d7
VG STATE:           active              PP SIZE:        64 megabyte(s)
VG PERMISSION:      read/write          TOTAL PPs:      1084 (69376 megabytes)
MAX LVs:            256                 FREE PPs:       852 (54528 megabytes)
LVs:                12                  USED PPs:       232 (14848 megabytes)
OPEN LVs:           11                  QUORUM:         2 (Enabled)
TOTAL PVs:          2                   VG DESCRIPTORS: 3
STALE PVs:          0                   STALE PPs:      0
ACTIVE PVs:         2                   AUTO ON:        yes
MAX PPs per VG:     32512
MAX PPs per PV:     1016                MAX PVs:        32
LTG size (Dynamic): 256 kilobyte(s)     AUTO SYNC:      no
HOT SPARE:          no                  BB POLICY:      relocatable
PV RESTRICTION:     none                INFINITE RETRY: no
# lsvg -p rootvg
rootvg:
PV_NAME           PV STATE          TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk1            active            542         542
109..108..108..108..109
hdisk0            active            542         310
108..00..00..93..109
```

Now we will remove the disk from rootvg and create a second VG:

```
# reducevg rootvg hdisk1
# mkvg -f -y myvg hdisk1
myvg
# lsvg myvg
```

```
VOLUME GROUP:       myvg                   VG IDENTIFIER:
0000nnnn00000000000014570781b25
VG STATE:           active                 PP SIZE:        64 megabyte(s)
VG PERMISSION:      read/write             TOTAL PPs:      542 (34688 megabytes)
MAX LVs:            256                    FREE PPs:       542 (34688 megabytes)
LVs:                0                      USED PPs:       0 (0 megabytes)
OPEN LVs:           0                      QUORUM:         2 (Enabled)
TOTAL PVs:          1                      VG DESCRIPTORS: 2
STALE PVs:          0                      STALE PPs:      0
ACTIVE PVs:         1                      AUTO ON:        yes
MAX PPs per VG:     32512
MAX PPs per PV:     1016                   MAX PVs:        32
LTG size (Dynamic): 256 kilobyte(s)        AUTO SYNC:      no
HOT SPARE:          no                     BB POLICY:      relocatable
PV RESTRICTION:     none                   INFINITE RETRY: no
```

```
# lsvg -p myvg
myvg:
PV_NAME           PV STATE          TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk1            active            542         542
109..108..108..108..109
```

```
# lsvg -l myvg
myvg:
LV NAME             TYPE       LPs     PPs     PVs  LV STATE      MOUNT POINT
```

Now we can create an LV:

```
# mklv -t jfs2 myvg 10
fslv00
```

```
# lsvg -l myvg
myvg:
LV NAME             TYPE       LPs     PPs     PVs  LV STATE      MOUNT POINT
fslv00              jfs2       10      10      1    closed/syncd  N/A
```

```
# lslv fslv00
LOGICAL VOLUME:     fslv00                     VOLUME GROUP:   myvg
LV IDENTIFIER:      0000nnnn00000000000014570781b25.1 PERMISSION:
read/write
VG STATE:           active/complete            LV STATE:       closed/syncd
TYPE:               jfs2                       WRITE VERIFY:   off
MAX LPs:            512                        PP SIZE:        64 megabyte(s)
COPIES:             1                          SCHED POLICY:   parallel
LPs:                10                         PPs:            10
STALE PPs:          0                          BB POLICY:      relocatable
INTER-POLICY:       minimum                    RELOCATABLE:    yes
INTRA-POLICY:       middle                     UPPER BOUND:    32
MOUNT POINT:        N/A                        LABEL:          None
MIRROR WRITE CONSISTENCY: on/ACTIVE
EACH LP COPY ON A SEPARATE PV ?: yes
Serialize IO ?:     NO
INFINITE RETRY:     no
```

Note that the LV is named `fslv00`—this is the default naming convention for an LV destined to hold a file system (we asked for a type of `jfs2` in the `mklv` command). We can override this default and use our own name for the LV using the `-y` option to the `mklv` command, however, in practice if we want to create a file system we normally use the `crfs` command to create both an LV and also a file system in that LV.

Showing how this can be done both ways, first we will create a file system to reside in our new LV, then we will directly create a second file system and hosting LV in a single command:

```
# crfs -v jfs2 -d fslv00 -m /fs1
File system created successfully.
655136 kilobytes total disk space.
New File System size is 1310720

# crfs -v jfs2 -g myvg -m /fs2 -a size=500M
File system created successfully.
524068 kilobytes total disk space.
New File System size is 1048576

# lsvg -l myvg
myvg:
LV NAME             TYPE       LPs     PPs     PVs  LV STATE      MOUNT POINT
fslv00              jfs2       10      10      1    closed/syncd  /fs1
loglv00             jfs2log    1       1       1    closed/syncd  N/A
fslv01              jfs2       8       8       1    closed/syncd  /fs2
```

Note that the actual size of the second file system turned out to be a bit larger than the requested 500 MB. As this VG has a PP size of 64 MB, the `crfs` command will take the desired file system size and round that up to the next even multiple of a PP, yielding in this case a size of 512 MB. Also note that the file systems show in a closed state, as they do not automatically get mounted after creation. Note that the file system creation command has also automatically taken care of creating a journal log LV in the VG.

Finally, let us mount the file systems and then dynamically increase the size of the first file system, and decrease the size of the second:

```
# mount /fs1
# mount /fs2

# chfs -a size=+100M fs1
Filesystem size changed to 1572864

# chfs -a size=-100M fs2
Filesystem size changed to 917504

# lsvg -l myvg
myvg:
LV NAME             TYPE       LPs     PPs     PVs  LV STATE      MOUNT POINT
fslv00              jfs2       12      12      1    open/syncd    /fs1
loglv00             jfs2log    1       1       1    open/syncd    N/A
fslv01              jfs2       7       7       1    open/syncd    /fs2
```

Dynamic LVM Changes

This shows some of the commands used to work with the LVM. Note that all of these examples can be done while the system is up and running, and require no data outages.

LVM Best Practices

We will conclude with some best practice tips:

1. When creating a VG, always request a Scalable VG (`mkvg -S`) so you will not run into any constraints on space that might at some point require you to take a VG out of production to convert it to a Scalable VG.
2. Create LVs large enough to meet only short-term needs and leave free space in VGs that can then be dynamically assigned to the LVs that need it in time.
3. Do not locate non-AIX data in `rootvg`—always create a separate VG (or VGs) to hold application data. This allows for the separate management of the operating system and the application in terms of things like backup/recovery, patching, and data mobility.
4. Keep in mind that in AIX we don't mirror disks, we mirror LVs. So, if we want to use LVM mirroring to protect against the loss of a disk, we must determine all the LVs that have allocations on that disk and ensure that those LVs are mirrored. There is a command (`mirrorvg`) that collates a list of all LVs in a VG and issues commands to mirror all LVs in the VG.
5. If you use `crfs` to auto create the LV for the file system, the name of the LV will be of the form `fslvNN`, which isn't very descriptive. You can choose to use a `chlv` command subsequently to change the name of the LV.
6. If you export a VG and import it to a second system, keep in mind that neither the names of the disks nor the name of the VG is stored in the VGDA. For example, let's say we have a three-disk VG named `datavg` consisting of disks `hdisk7`, `hdisk8` and `hdisk11`. We export it (`exportvg datavg`), remove the disks from the ODM (`rmdev -dl hdisk7/8/11`), and then physically move the storage to a second system. On the second system we issue `cfgmgr` and we should see three new disks, assigned the next three available names, which might be `hdisk5`, `hdisk6`, and `hdisk7`. We can then reference any one of these disks to import the VG, and we will have to supply the VG name also at this point (and it can be different than the original). So the import command might look like:
   ```
   # importvg -y sys5_datavg hdisk6
   ```
7. Migrating data between disks is easy. Allocate the new disk to a server and add it to the VG containing the data you need to move. Use the `migratepv` command to move data between disks, then remove the old disk. For example, if we have a VG named `myvg` containing `hdisk5` and `hdisk6`, and we need to provision a new LUN to hold the data currently held in these two disks, then assuming the new disk is assigned the name `hdisk7` (and it is large enough), the command sequence would be:

   ```
   # extendvg myvg hdisk7
   # migratepv hdisk5 hdisk7
   # migratepv hdisk6 hdisk7
   # reducevg myvg hdisk5 hdisk6
   # rmdev -dl hdisk5
   # rmdev -dl hdisk6
   ```

   This can all be done while data is in full production, the only availability consideration being IO performance, if a large amount of data is involved.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge through Power Systems training.

AIX, UNIX
IBM i
Linux
System Software

Visit **www.globalknowledge.com** or call **1-800-COURSES (1-800-268-7737)** to speak with a Global Knowledge training advisor.

About the Author

Iain Campbell is a mechanical engineer by profession. While responsible for managing several production automation labs at Ryerson Polytechnic University in Toronto he became distracted by UNIX operating systems. His first experience of AIX was a PC RT used in the lab as a machine cell controller. Iain has been teaching and consulting in AIX and Linux since 1997. He is the author of *Reliable Linux* (Wiley, New York, 2002), as well as several technical papers and curriculum material. He holds LPI and Novell certifications in Linux administration, and is an IBM certified AIX Specialist.