WHITEPAPER

The Logical Volume Manager (LVM)

This document describes the LVM in SuSE Linux. It is freely distributable as long as it remains unchanged.

SuSE has included a Logical Volume Manager since SuSE Linux 6.3. The LVM in SuSE Linux is Heinz Mauelshagen's implementation, the homepage is now available at http://www.sistina.com/lvm/.

SuSE Inc. ©2001
Michael Hasenstein <mha@suse.com>

TABLE OF CONTENTS

LOGICAL VOLUME MANAGEMENT − INTRODUCTION

Volume Management creates a layer of abstraction over the storage. Applications use a virtual storage, which is managed using a volume management software, a *Logical Volume Manager (LVM)*. This LVM hides the details about where data is stored, on which actual hardware and where on that hardware, from the entire system. Volume management lets you edit the storage configuration without actually changing anything on the hardware side, and vice versa. By hiding the hardware details it completely separates hardware− and software storage management, so that it is possible to change the hardware side without the software ever noticing, all during runtime.

With LVM system uptime can be enhanced significantly, because for changes in the storage configuration no application has to be stopped any more.

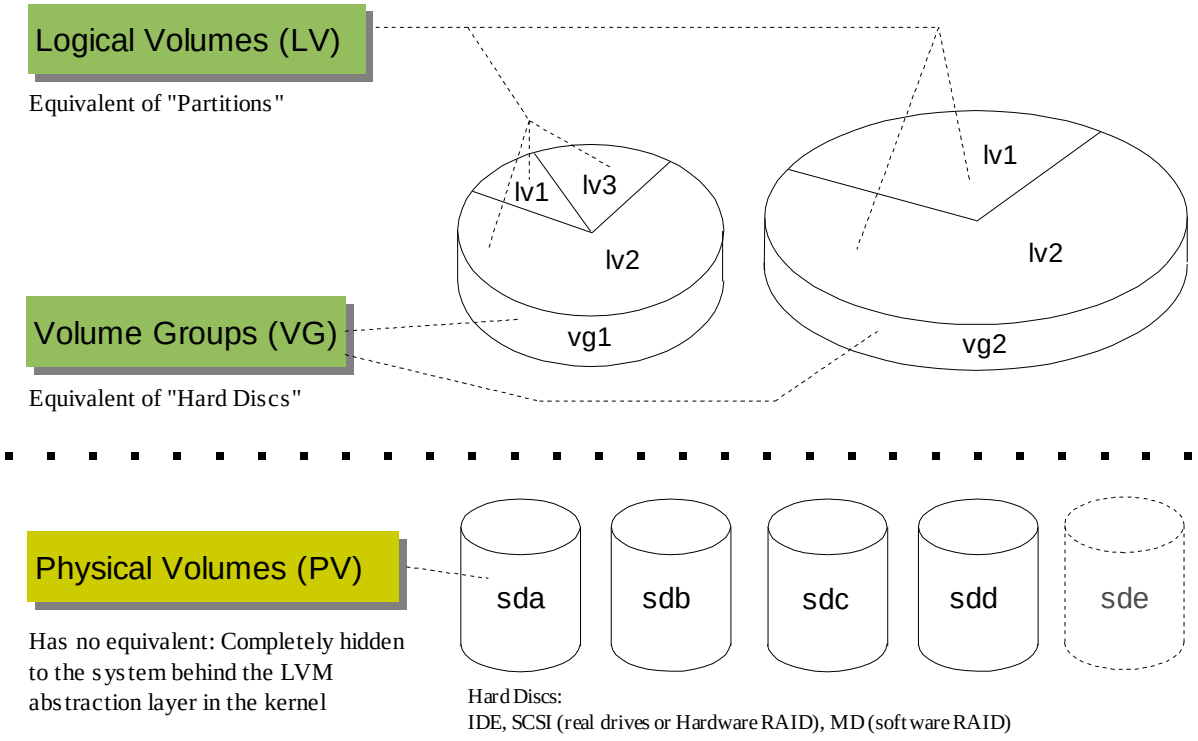
How it works

No LVM:
Without an LVM there are physical disks with partitions. On these partitions are filesystems or they are used *raw* (i.e. not managed by the kernel but with direct low−level access from the application), e.g. by databases like Oracle.

With LVM:
A *Volume Manager* puts the physical entities (*Physical Volumes*) into storage pools called *Volume Groups*. The LVM in SuSE Linux can manage whole SCSI− or IDE−disks and partitions in a Volume Group and also hardware− and even software RAID devices.

A Volume Group is the equivalent of a physical disk from the systems point of view. The equivalent of partitions into which this storage space is divided for creating different filesystems and raw partitions on it is called a *Logical Volume*.

What a *Logical Volume Manager* does:

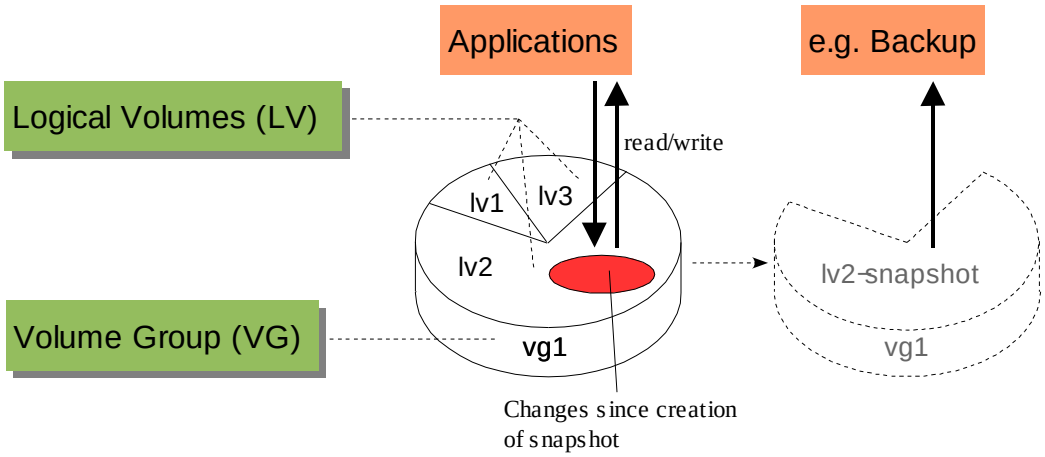


What LVM can do

- **Highly available systems:** The management of a Volume Group and of Logical Volumes can be done while it is being used by the system. For example, it is possible to continue reading and writing into user home directories stored on Logical Volumes while the entire disk subsystem where the Bits and Bytes are stored on is exchanged against a completely different one. We give this extreme example of flexibility because exchanging a single broken disk is possible with 'simple' hardware− or software RAID as well and to show the difference the complete separation of actual physical storage and logical storage used by the applications makes.

- **Moving logical storage:** An example has already been given above. It is possible to move logical storage to a different physical location. This can be done without LVM as well, what makes LVM special is that it can be done during runtime, while the storage is used.

- **Snapshots:** (Since SuSE Linux 7.0 with the kernel− and LVM−update) It is possible to create a *Snapshot Device* that is an alias for an existing Logical Volume, but can only be accessed read−only and contains an image of the LV "frozen in time", i.e. while applications continue to change the data on the LV this logical device contains the unchanging image of the LV of the time when the snapshot was created. This makes it possible to do a consistent backup without shutting anything down or using any special software, because this method is independent of any software since it happens in the LVM abstraction layer.

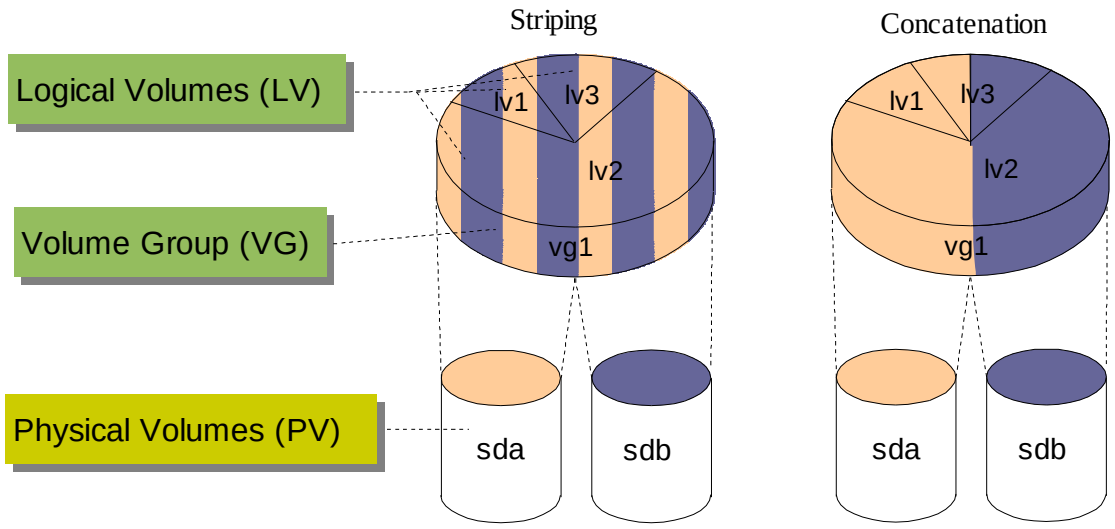


- **Resizing:** It is possible to add storage space to already existing Logical Volume(s), increasing their size(s) during runtime. If the filesystem on the logical volume supports it (like ReiserFS on SuSE Linux does) it can be resized during runtime as well to immediately use the additional space, without unmounting it so that applications and users will not even notice the operation.

- **Concatenation&Striping:** It is possible to group disks and/or partitions on completely different storage subsystems into one Volume Group (one logical disk). For example, there are two IDE drives /dev/hdc and /dev/hdd on the system with 30 GB each, but they are to be used under one mount point as one contiguous storage block. They can both be taken into one Volume Group, forming one big logical disk. Their physical storage space can be concatenated, i.e. the system writes to the next one if one disk is full,

or striped, where one stripe of data is stored on one drive, the next one on the other, and so on in an alternating pattern, increasing throughput by giving each drive smaller jobs to do.
This is something that can be done with hardware− or software−RAID as well, only that with LVM it does not matter where the disks are at all. You can do the same with two big external storage towers, not just with two disks.

Example for concatenation & striping of two disks to one logical disk:



- **Shared Storage Cluster option:** The LVM used by SuSE can also be used in shared storage clusters. The configuration is stored on the physical storage itself. On startup, SuSE Linux scans all physical volumes known to the system (all drives and partitions on all SCSI− and IDE channels and all software RAID drives). If it finds any space used by LVM it reads the configuration stored on it and sets up the local node accordingly, including all device files. This means LVM on a shared storage cluster has to be setup only once on one node, and all other nodes that can "see" the storage at all automatically set themselves up when they are booted. Of course this can also be accomplished without rebooting those other nodes. However, this version of LVM is not yet fully cluster aware, i.e. it does not actively share setup information between the nodes. Certain operations therefore require shutdown of any LVM use on all but one node, with subsequent restart of these nodes after the reconfiguration. For example, if logical storage is moved to a different physical location, only the node performing the operation is aware of it while all others are not aware of it and continue using the old location. Therefore this is an example for a configuration operation that can not be done while all nodes are using the storage.

- **Independent of location of a disk:** Since the LVM configuration is stored on the physical volumes managed by LVM themselves the system is independent of physical device IDs. For example, the drive names in Linux for SCSI devices are /dev/sda for the first disk, /dev/sdb for the second, etc. The order is determined by the order the disks are recognized during boot time. If another disk is inserted into the SCSI chain with a SCSI ID in the middle of the already existing drives it will be seen earlier than those drives during boot time, and get a device ID that so far was used by another drive. Since the LVM setup is stored on the drives, and since the setup of the runtime LVM is rebuild during boot time (or when using the special LVM commands also during runtime) LVM will work without problems in such a situation, because it does not use those variable device IDs but the information stored on each drive itself to identify the drives that are managed by the LVM.

How it *really* works − Physical Extents (PE), Logical Extents (LE)

Now that we looked at what LVM can do lets have a look at how LVM works on the inside. We assume the setup has already been done and we are looking at a fixed LVM configuration, because we are only interested in how it works right now and not in any administrative issues.

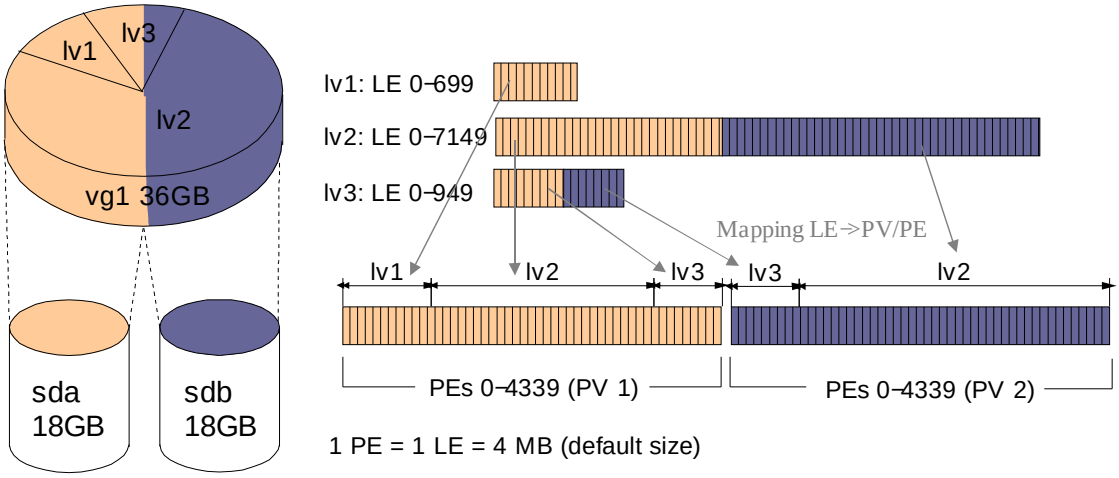
Each Physical Volume (PV) is divided into equally sized PEs. The size of a PE is variable but equal for all PEs in a Volume Group (VG).
Within one PV the ID number of any PE is unique. Each PV has PEs numbered 0 to (size−of−PV / size−of−one−PE = total−number−of−PEs). A PE is the smallest unit that can be addressed by LVM on physical storage.

In the VGDA, an area on each PV used to store the setup metadata (see appendix A), information is kept about which VG the PV belongs to, what the size of a PE is, and − for this paragraph the most important item − which PEs of this PV belong to which LV and what the sequence number of this PV is in the VG. Remember: In a VG consisting of any number of PVs there can be any number of LVs which could potentially be stored physically on any of the PVs of the VG, in whole or in part.

When the system starts up and the VGs and LVs are activated for use this VGDA information is read into the system, which uses it to construct the mapping information which is used to identify where the parts of any LV are really physically stored.

Every LV is divided into LEs. LEs are of the same size as the PEs of the VG the LV is in. Every LV has LEs numbered 0 to size−of−LV / size−of−one−LE = total−number−of−LEs, which is the same as the formula above for the PEs. Every LE is mapped to exactly one PE on a PV, i.e. there is a 1:1 mapping LE:(PV:PE). A PE is not unique globally because it is only unique per PV, so a logical equivalent of PEs had to be introduced.

Example: disk sda is configured to be PV 1, sdb is PV 2.



When an application now wants to access storage on a logical volume, the LE this space is in is identified and by using the unique ID number of the LE in the LV both the PV and the PE are found in the mapping table. Now the I/O can take place because the exact physical location of the accessed space on the LV has been identified.

Using the graphics above, let's look at an example: An application accesses the LV lv3 at Byte 254.123, so the LE is #62 (62*4MB=253.952, #63 would be the next LE). The mapping table tells us that for LV lv3 LEs 0−~500 (approximately) are stored on PV1, and the PE is number of PEs of lv1 on PV1 plus number of PEs of lv2 on PV1 plus 62.

Difference between LVM and RAID Systems

Logical Volume Management is not a replacement for RAID. Just like RAID it is also no replacement for a backup strategy!

Hardware RAID works on a very low device level and is bound to a specific controller and therefore storage subsystem. It is meant to safeguard against failures of one or more (depending on what the RAID configuration is) drives within one storage system, and/or to increase throughput, again depending on the particular RAID configuration.

Software RAID is usually used when no hardware RAID controller is available and is a software level emulation of it.

Logical Volume Management works independent of any particular storage system. It can also provide certain RAID functions like striping or mirroring (the LVM in SuSE Linux only implements striping so far, but you can use MD to do software mirroring with LVM), just like software RAID, but this is only a side effect.

We have mentioned many of the differences already. For example, the independence from the device IDs due to LVMs own IDs stored on each disk. The snapshot device is another example for a special service the LVM abstraction layer can provide. Concatenation and striping across different storage subsystem; hardware RAID only works with the drives directly attached to a RAID controller. The high availability option, since most **volume management operations can be done during runtime, while the logical storage is used by an application**. The ability to do things that go beyond what the particular hardware controller could do, even if a sophisticated RAID system is used: for example, an Oracle database server needs more space for one of its tablespaces. Normally one would just add another drive to the RAID array using the proprietary software the manufacturer of the array provides. What if that array is already full? With LVM, it is possible to add any disk anywhere in the system, i,e, it could be put into a free slot on another RAID controller and added to the Volume Group the tablespace is on so that the Logical Volume used by the tablespace can be resized to use the new space in the VG, and Oracle can be told to extend that tablespace.

While expensive RAID arrays come with sophisticated software, this software is limited to the storage hardware, i.e. the RAID array, and cannot do anything else. The Logical Volume Manager on the other hand is independent of any proprietary storage system and treats everything as "storage".

For example, you have a server with an external (e.g. fiber−channel attached) RAID array. You need more space, but all disks are full. If you use LVM, you can simply attach another storage array to the FC and **during runtime** configure LVM to add it’s space to the Volume Groups, and extend the Logical Volumes. If you would like to replace the current storage array with a new, bigger one altogether, you can do the same, and − again during runtime! − move all the storage (all Logical Volumes and Volume Groups) from one array to the next.

Let’s not forget the ease of administration: storage management can be done during runtime, and the LVM and how it is administered is exactly the same on all SuSE Linux platforms: i386, IA64, AMD x86−64, S/390, PowerPC, Sparc[64].

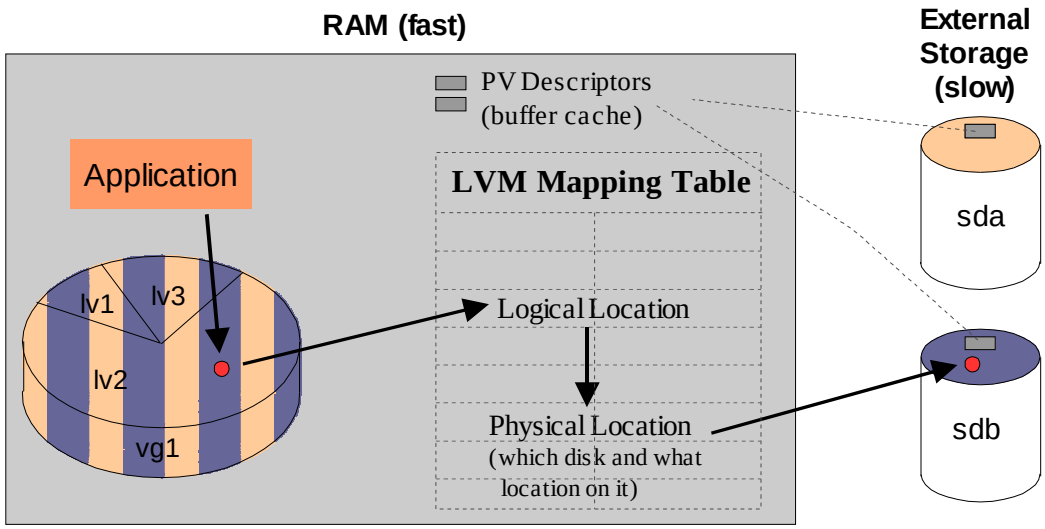
Disk failures and LVM

LVM does not provide any protection against data loss if a disk fails. You can use hardware or software RAID 1 (mirroring) or RAID 5 for this purpose, but note that neither LVM nor RAID are any replacements for having backups. LVM is one more tool to achieve *high availability*, and you have to use many of them together like RAID on the disk level, LVM for all storage management, HA software (heartbeat and application monitoring), and a backup strategy.

Performance Costs of using LVM

The LVM layer in the kernel keeps a mapping of logical storage locations to real (physical) storage locations. That means that for each read or write access to storage LVM has to translate the location of the asked−for piece of information on the disks by looking it up in a table. While this "costs" in terms of CPU usage the operations done are "cheap" (i.e. done very fast), since doing a lookup in a mapping table is one of the fastest and most basic operations computers do.

Each LVM physical volume contains a small area with information about the PV itself (for example the ID of this volume, because LVM identifies it via this ID and **not** via its name on the system, because the name can change, e.g. if the SCSI id changes!), about the volume group the PV belongs to and about the LVs stored on it. Since this area is pretty small (about 128k is used on every PV on a VG of 8.5 G with 5 PVs) and often used − for every single read/write operation − it is stored in the buffer cache of the Linux kernel, i.e. in RAM. Even more, those parts of it actually used by the piece of code the processor executes are most likely stored in the 1st−level cache, i.e. on the processor itself. Doing the mapping lookup therefore takes no time at all on today's ultra−high frequency CPUs compared to the very long time it takes to access the mechanical storage on the disks.



Also keep in mind that LVM data is only written when the LVM configuration is changed. During runtime, all operations involving the LVM mapping table are read−only. That is very good for caching, especially on multi−processor systems where the 1st−level caches on all the CPUs contain parts of the mapping table.

Real Test − A Benchmark

Theory is nice, but how about the real world?

Here are some numbers from a (non scientific) benchmark that compares I/O on raw disks with I/O over LVM and just one disk, to see the LVM overhead directly, and LVM and 4 disks in a striped configuration.

We used the *Bonnie* benchmark, which is also included in SuSE Linux under that same name. See the documentation that comes with that package for details about this particular benchmark. The machine used was a Compaq ProLiant 1850 with 6 disks. The Smart−Array RAID controller in this server has been configured not to do anything, that means we have direct access to the disks. The machine has 512 MB of RAM. Since the filesize used in the benchmark is 1024MB we manage to defeat caching to get to the "real"

I/O performance. The filesystem is ext2 with 4k blocksize in table 1 and reiserfs in table 2. This adds the opportunity to compare ext2 and reiserfs, but please do not let that aspect confuse you: Look at only *one* of the tables at a time when you check for the results for LVM.

At first the four disks are tested with no LVM involved. These numbers can later be used for comparison.

The next test includes LVM. A VG is created that contains just one PV, and that is disk HD 1. We can compare the numbers achieved directly with those we got from testing HD 1 directly, without LVM, and get to the LVM overhead this way. As promised, there is no visible overhead. The data we get there does not look any different from what we got when testing the disks without LVM.

The last test creates a VG that contains all four disks involved in this test. On this VG an LV is created that is striped over all four PVs (disks) to give us the opportunity to assess any possible performance gains or losses when using LVM striping (software RAID0).

Table 1: Filesystem is ext2

---Sequential Output (nosync)---				---Sequential Input--				--Rnd Seek-	
-Per Char-		--Block---		-Rewrite--		-Per Char-		--Block---		--04k (03)-	
K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	/sec	%CPU
HD 1 (no LVM)											
8500	98.9	20073	17.2	9070	15.7	7970	88.4	22082	16.2	251.2	2.6
HD 2 (no LVM)											
8358	97.1	20927	18.0	9129	16.2	7968	88.6	22096	15.7	258.9	3.6
HD 3 (no LVM)											
8309	96.5	20525	17.8	9079	15.8	7729	85.9	22091	15.7	249.9	3.4
HD 4 (no LVM)											
8343	97.0	21295	18.4	9065	16.3	7938	88.2	22017	14.9	258.7	3.2
LVM – HD 1 only, shows pure LVM overhead, compare with result 1st HD above											
8395	97.8	21574	18.8	9106	16.0	7957	88.5	22097	16.6	256.0	2.6
LVM – HDs 1,2,3,4 in one VG, one striped LV (striped over all 4 HDs)											
8474	98.9	32048	27.8	6138	11.1	7142	79.8	23983	17.8	385.6	6.4
8428	98.3	31942	28.4	6064	10.7	6875	76.5	24198	17.6	372.1	4.4

Table 2: Filesystem is reiserfs

---Sequential Output (nosync)---				---Sequential Input--				--Rnd Seek-	
-Per Char-		--Block---		-Rewrite--		-Per Char-		--Block---		--04k (03)-	
K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	/sec	%CPU
HD 1 (no LVM)											
7775	98.9	22793	76.2	8686	16.9	7265	81.4	21856	17.1	255.9	3.1
HD 2 (no LVM)											
8399	97.8	22599	19.8	9233	16.2	7958	88.4	22077	16.6	250.4	2.6
HD 3 (no LVM)											
8481	98.8	21080	18.5	9080	15.9	7969	88.5	22069	16.3	249.5	3.1
HD 4 (no LVM)											
8487	98.8	21149	18.4	9052	16.0	7956	88.5	22010	15.0	260.9	3.8
LVM – HD 1 only, shows pure LVM overhead, compare with result 1st HD above											
7827	98.7	22609	81.8	8682	16.9	7223	80.9	21787	18.0	263.7	2.8
LVM – HDs 1,2,3,4 in one VG, one striped LV (striped over all 4 HDs)											
7837	99.1	26304	85.9	5718	11.1	6744	75.6	23229	18.3	372.2	5.7
7824	98.9	29375	93.4	5787	11.1	6822	76.6	24296	18.9	392.7	4.7

HOW TO SETUP THE LVM

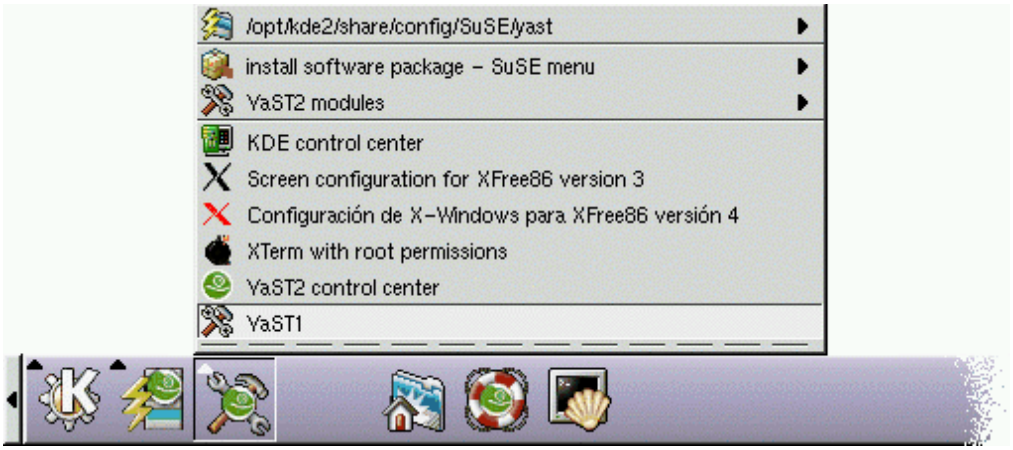
Command line tools, ASCII UI (YaST1), Graphical UI (YaST2)

Command line: As everything in Unix/Linux, the logical volume management is controlled using command line tools. For a list see the appendix B or the manual page for lvm(8) on your system.

YaST1: This interface has been there since SuSE Linux 7.0. To make it easier for those who do not want to remember the name of all the various commands and the sequence and arguments how to call them e.g. to create a configuration from scratch SuSE Linux provides a GUI interface to these tools. There is one in YaST (the SuSE setup− and configuration tool), and starting with SuSE Linux 7.2 there is also a module for YaST2 (successor of YaST).
To get to the LVM setup function of yast1. start YaST via the command *yast* or from the menu in KDE or Gnome, and go to *Installation settings −> Configure the Logical Volume Manager*.

YaST2: A module for LVM setup was introduced with SuSE Linux 7.2. To start the YaST2 module for the LVM configuration, if available on your system, call the YaST2 control center either by starting *yast2* from the command line (as *root*) or from the KDE− or Gnome menu. Note that YaST2 can also be started in an ASCII mode, without X. In the control center, look for and start the LVM module, in the graphical mode by clicking on the icon, in text mode by selecting it using *TAB* and then pressing *ENTER*.

How to start YaST1 or YaST2 from the KDE menu, here: SuSE Linux 7.1 default KDE screen



In the following sections we will show how to set up LVM using the yast1 interface and then again using only the command line tools. We will not introduce the YaST2 interface since it follows the same principles and rules.

Appendix E lists the various LVM setup options available in the different SuSE Linux releases.

Note: If available at all, LVM and the setup tools are exactly the same on all architectures SuSE Linux is available on, i.e. you can use this paper on SuSE Linux for S/390 the same way as on the IA64 release.

PLEASE check ftp://ftp.suse.com/pub/suse/[architecture]/update/[SuSE Linux release]/ **for any updates** for your SuSE Linux release on your architecture (s390,i386,ia64,sparc/sparc64,ppc,axp)!

The three LVM setup steps

To start any new LVM configuration, we

- have to **define and initialize the Physical Volumes** that we plan to use, then
- **define Volume Groups** where we group the PVs together that we want to manage in one group, next
- we **set up Logical Volumes** on each Volume Group.

Using any interface, setting up any LVM, not just this one, always consists of these three steps. Step 1 (PVs) may be part of Step 2 (VGs) because selecting any disk or partition for inclusion in a VG automatically means they can be prepared as a PV for LVM and the more intelligent setup tools will do it for you. If you use the command line tools you will have to do it manually.

Setting up a PV: The device is initialized with an LVM ID and space is allocated and reserved for LVM data structures later storing information about the LVM setup on that PV (which VG is is part of, what LVs does it contain in what PEs, etc.).

Setting up VGs: All PVs that belong to the VG get an entry in their LVM metadata structures that marks them as belonging to that VG, and information about where in the VG they belong (sequence number).

Setting up LVs: Again, the metadata is stored on the PVs where, in the end, all user data is stored. Each PV stores the data about the VG it belongs to, and what parts of which LVs are stored on the PV.

LVM setup is Machine independent

The primary storage area for the LVM setup information are the PVs and not the machine where the setup was created. The servers that are actually using the storage only have local copies of that data, but at any time they can recreate their local copy of the LVM setup, including device files for the LVs, completely out of the information stored on the PVs.

This means that you can take all disks or maybe an external storage array you may be using and attach it to any other server, and they will automatically recognize and copy the LVM setup stored on the disks and no manual setup has to be done on the server. This requires that the LVM versions of the server and the one used for creating the setup are compatible, of course. See appendix E for that information.

The way SuSE Linux does it is via /etc/init.d/boot, where during boot time all physical devices are scanned if they contain any LVM setup data, and if yes, that setup is reconstructed on the server and the LVM is ready immediately after boot, regardless of if this machine was the one used to setup the LVM or not.

LVM also gives each PV a unique LVM ID, so that should you change SCSI IDs, for example, and what used to be /dev/sdd is now /dev/sdc, LVM will not have any problems since it does not use the name of the device but its LVM ID stored on it during the setup phase. This is of course a necessary part of creating the server independent setup LVM does.

Note about using whole disks

We have said that there are three types of devices that can be used as PVs: partitions, whole disks, and multiple devices (MD, Linux software RAID).

An example for a partition would be */dev/*sda3, which is the 3rd partition on the 1st SCSI disk. There is one important "limitation" when using whole disks. The LVM tools used to setup LVM will recognize a whole disk as possible device for LVM use **only** if it has **no** partition table! If the disk already has one, even an emtpy one (i.e. you delete all partitions), LVM will refuse to use that disk. This limitation of the LVM tools (not one of LVM itself) can easily be overcome by deleting the partition table using this command:

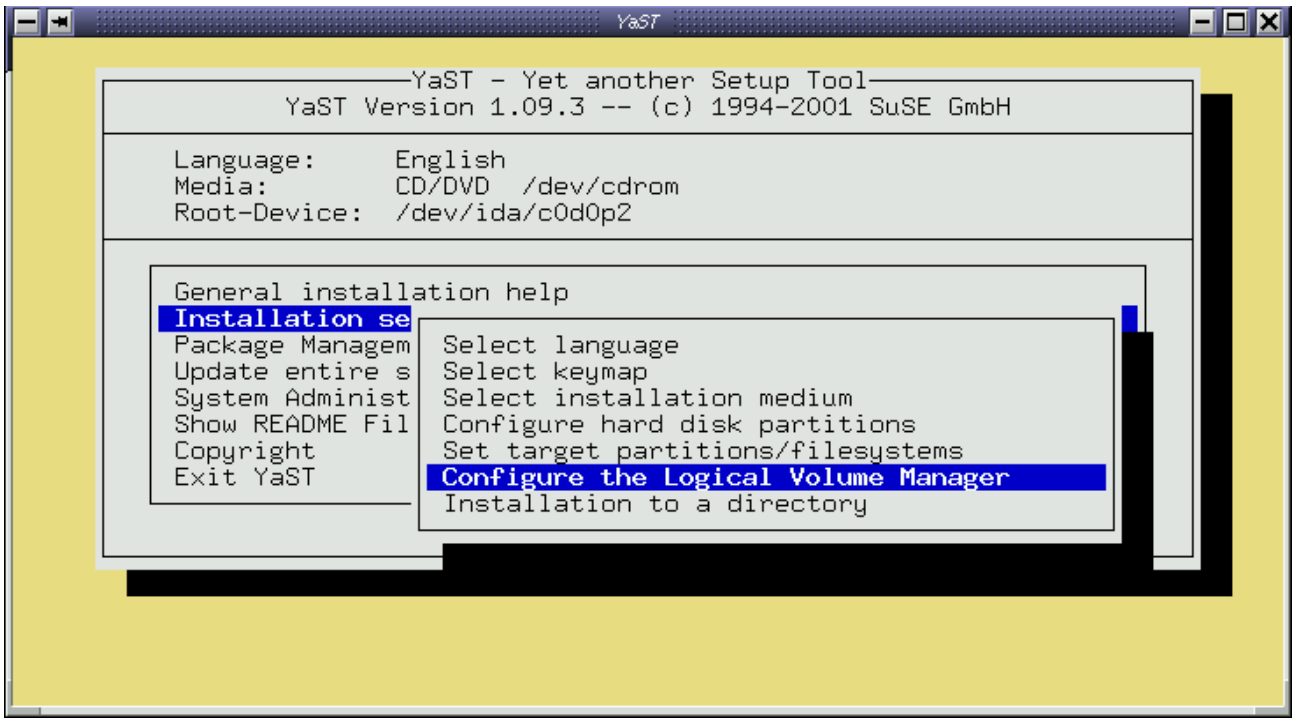
```
dd if=/dev/zero of=/dev/[device-name of disk] bs=1k count=1
blockdev --rereadpt <devicename>  # tell kernel to re-read p.table
```

This command overwrites the first 10k Bytes of a disk with zeros, deleting any partition table.

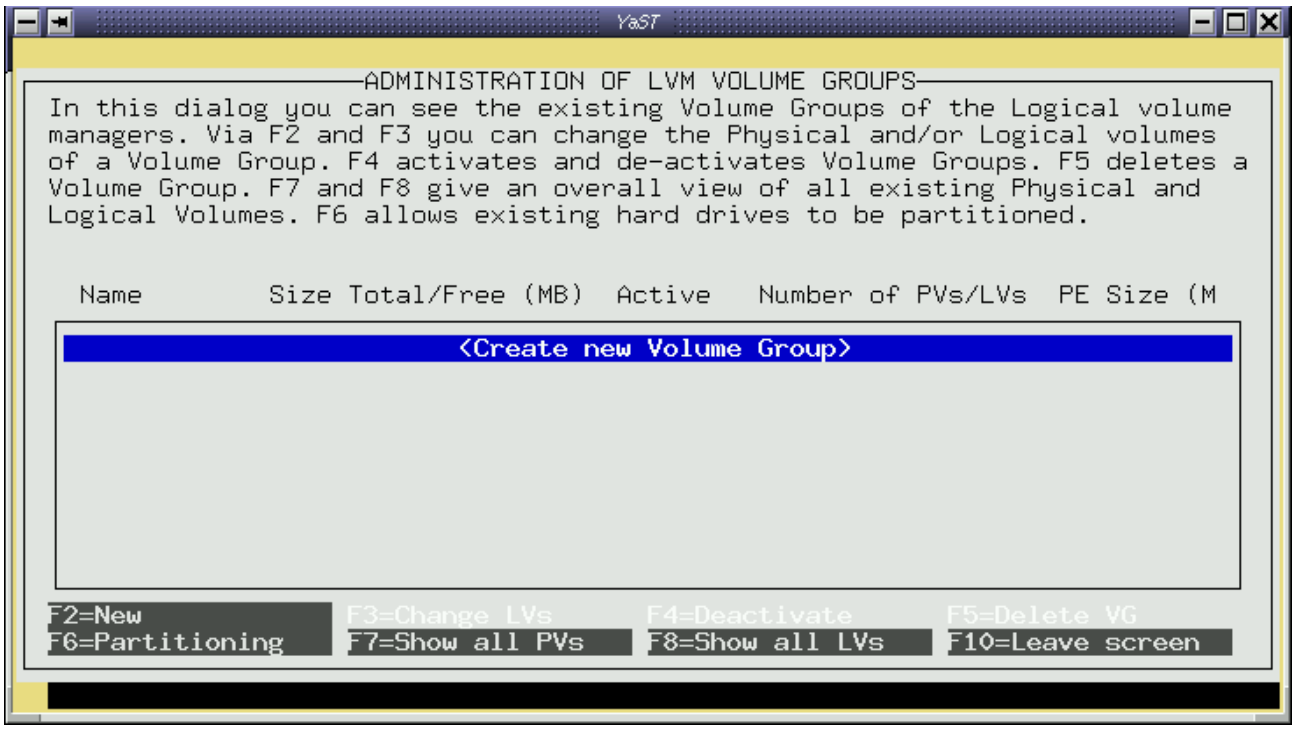
SETTING UP LVM USING YAST1

After starting YaST1 we go to the LVM configuration by selecting *Installation settings −> Configure the Logical Volume Manager*.

YaST 1 screen (showing the main menu with the Installation menu folded out)



YaST 1 LVM setup screen − no LVM setup exists so far



Steps 1 and 2: Initializing PVs and creating VGs

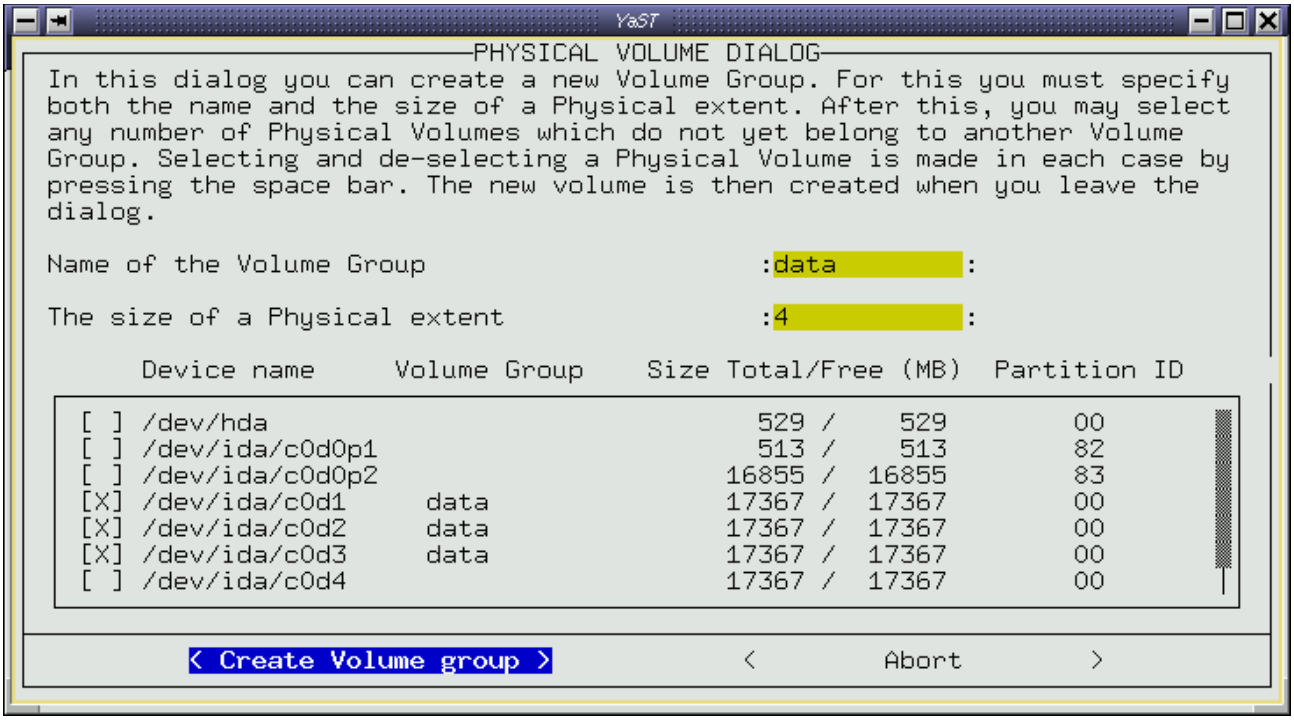
The YaST1 LVM interface is "intelligent", i.e. in each screen it will show us only those options and action items that are reasonable at this point. If there are no VGs we will not see the item *Edit LVs*, for example.

When we start with no pre−existing LVM setup the only action item we have available is to create a new VG. In the VG dialog we are presented with all devices (disks, partitions, multiple devices) available for inclusion in the new VG. **Caution:** The dialog also includes partitions currently being used by the system, for example the partition holding our root filesystem. Make sure you don't accidentally use one of those for LVM! Basically, this dialog lets us select all devices that are not already in some other VG.

The dialog includes both steps 1 and 2, i.e. if you select a device it is initialized as a new PV (step 1) and then included into the VG (step 2).

The size of a Physical Extent (PE) is important when you want to create very large storage areas, because any PV and any LV can have only 65534 of them. Also, this is the smallest amount by which you can resize an LV later, and the smallest storage area you can allocate on a PV. For most purposes the default of 4 MB can be used, which gives a maximum of 65534*4MB ~255GB per PV and per LV. Also see appendix C, limits of LVM.

YaST1: Create/Edit a Volume Group (VG)



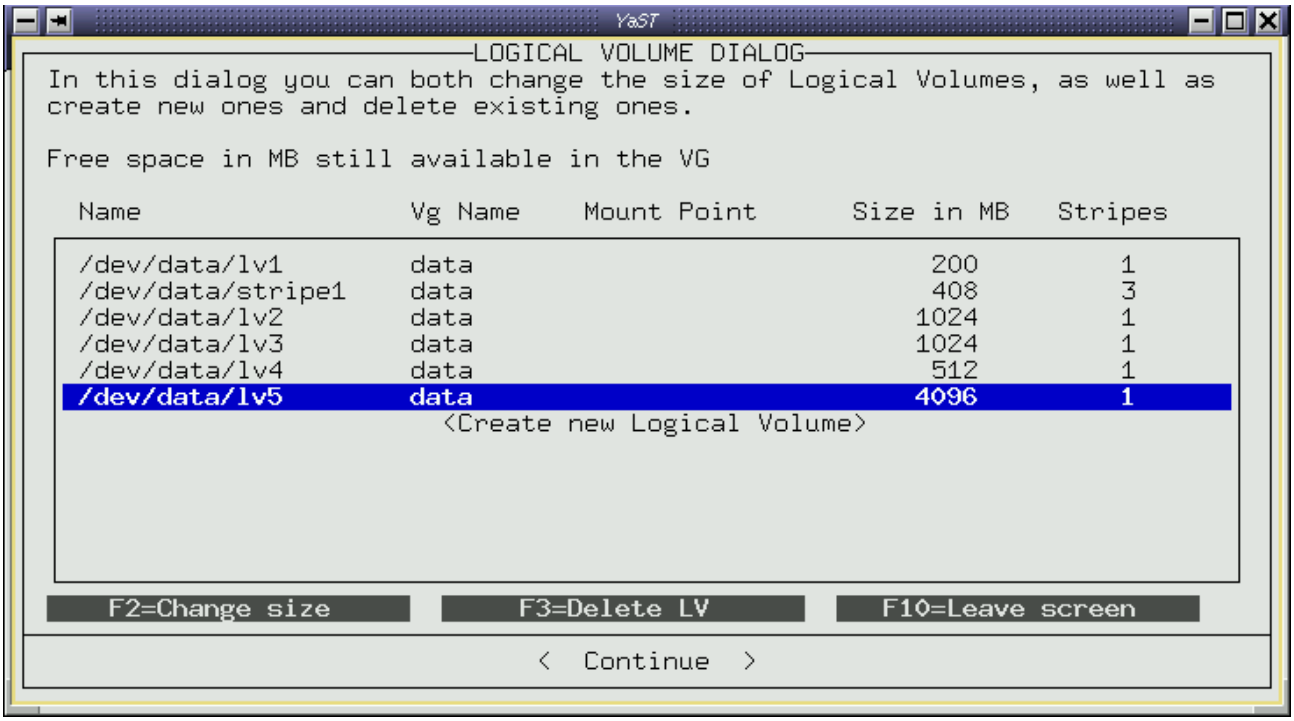
For the selection of the name of the VG please note that the device name of the LVs will be */dev/[name−of−VG]/[name−of−LV]*. This means you must not use a name that exists under /dev/, because that would lead to a conflict. For example, you cannot give the name *zero* to a VG, because a file */dev/zero* already exists, so that no directory */dev/zero/* can be created for that VG. You will get an error "file already exists" if you choose a name that already exists under */dev*. It may be possible that YaST1 will list the PV as belonging to a VG of that name even after the error. In this case, just restart YaST1. A VG is not created if this happens.

Note that YaST1 scans your physical devices only when you start the LVM setup dialog. For example, should you find out during the setup that one whole disk is missing because of *"Note when using whole disks"* above and you solve that problem using the command given above, you will have to exit and restart YaST1 so that it scans the physical devices again.

Step 3: Setting up LVs

This step requires that we already have at least one VG to put the LV(s) on. YaST1 will let you select the *Create New LV* option only if that is the case. In the main LVM setup dialog the menu action item *F3=Change LVs* will become available after you finished steps 1&2 and created at least one VG.

YaST1: Create/Edit Logical Volumes (LVs) on a Volume Group (VG)



Select the VG you want to create or edit LVs in by highlighting it and then press F3 (if you are in a KDE Terminal press the number 3 instead of F3, because the F−keys will act like ESC).

To create a new LV highlight the item <*Create new Logical Volume*> and press RETURN.

The name you choose will become the device name of the LV. Remember: */dev/[VG−name]/[LV−name]* will be the full name of the device which you can use to create a filesystem, mount it, or create a raw device mapping to.

The size parameter is obvious. Also read the help text in the dialogs.

The *Number of stripes* specifies over how many PVs this LV will be striped (RAID0). A value of 1 means there will be no striping. Usually you don't want to do striping, because you use hardware RAID to do any RAID. If you want to use it anyway you must know how many PVs you have (you can't have a stripe size larger than the number of PVs, obviously). You must also make sure that the size of the LV is within the size of the smallest PV used for the LV, because you cannot stripe a 1GB LV over 1 disk that is 1GB and another one that only has 512MB (just an example!), because there are another 512 MB missing on the smaller disk. This is something to keep in mind if you use very different disks in one VG.
If you do not use striping the PVs are simply concatenated.

You can change the size of an already existing LV at any time, during runtime. See section *Using LVM* further down for more information.

That is all, you are now done setting up LVM! You can go ahead and start using the LVs like you use partitions now.

SETTING UP LVM USING THE COMMAND LINE TOOLS

This is very useful to automate the setup, e.g. when you have lots of LVs to be created. We again follow the three basic steps:

Step 1 (PVs):

```
pvcreate /dev/device [/dev/device] ...
```

The *pvcreate* command initializes (a) physical device(s) as (a) physical volume(s) (PV). Partitions must have the partition ID 8E (use fdisk, cfdisk or yast for partitioning), (whole) disks must have no partition table (none, not just an empty one). To see all physical devices that are available, use the command `lvmdiskscan`. The command `pvscan` shows all existing PVs.

If a disk (the whole disk, e.g. /dev/sdc) cannot be used and pvcreate complains about it having a partition table, after making sure you really want to use this disk for LVM and do not have anything else on it you can erase the partition table with `dd if=/dev/zero of=/dev/device-name bs=1k count=1` (followed by `blockdev --rereadpt <devicename>` to tell the kernel to re-read the p.table).

Step 2 (VGs):

```
vgcreate vg-name /dev/device [/dev/device] ...
```

The *vgcreate* command creates a volume group with all the physical volumes specified. The default extent size is 4 MB. The limit for the number of physical extents/logical extents on a PV/LV is 65534 due to kernel limits. Therefore, the default size of 4MB limits the max. size of any PV or LV to 4M*65534~255GB. Should you have larger PVs or require bigger LVs you can increase the extent size. A PE is also the smallest unit by which you can increase, decrease or address (e.g. to move PEs to other PVs) storage in LVM.

Step 3 (LVs):

```
lvcreate -L size[K|M|G|T] -n lv-name vg-name
```

The *lvcreate* command creates a logical volume on the volume group *vg-name*. If no name is given for the LV the default name lvol# is used, where # is the internal number of the LV. The LV can be created as striped (RAID0) over several or all available PVs using the *-i #* parameter, where # specifies the number of stripes. Normally LVs use up any space available on the PVs on a next-free basis, i.e. if you do the creation of LVs one by one you essentially get concatenation, until you start shrinking, growing, deleting and so on, when space gets freed and reallocated anywhere on the PVs.

This is all. From now on you can use the LVs by referring to them via */dev/vg-name/lv-name*. Use them like you would use any partition.

Automatic server setup from stored LVM setup

The LVM of the server can be reset using `lvmchange`. This is a last resort if something goes wrong. It does not change the LVM setup stored on the physical devices, only the data in the running system is reset. To restore an LVM setup from the one stored on disk, run first `vgscan` to find all VGs and then `vgchange -a y` to activate them. This is what happens at each boot time, see init file */etc/init.d/boot* (look for *vcgscan*). You can completely delete the local LVM setup information with `rm -rf /etc/lvm*`, including all device files for accessing the LVs (*/dev/vg-name/lv-name*), and it will automatically be recreated next time you boot or issue the *vgscan* command manually. This means that you do not have to do anything after a reboot to be able to use your LVs again, the runtime LVM setup happens automatically during a system boot, if any LVM setup is detected on any of the attached physical devices.

INSTALLING THE ROOT FILESYSTEM ON LVM

This is not a complete guide, just a few tips. Installation to logical volumes is supported since SuSE Linux 7.1. Except for this missing but hardly necessary option the LVM in SuSE Linux 7.0 is fine.

Note that this setup has the potential for serious administrative trouble unless there is an administrator available able to solve any issues. For example, if you upgrade the user level LVM tools but forget to also update the kernel part, or if you install a new kernel but forget to install a new initrd or run lilo afterwards you end up with a system that won't boot any more. Recovery is much easier without LVM, since all you have to do is mount a partition, here you also have to start LVM in the rescue system. Therefore we recommend this only for the experienced administrator. Using LVM for any data partitions not critical for system boot is much less of an issue. Be especially careful when updating the LVM version, be absolutely sure to update **both** the kernel and the lvm tools to the exact same version.

When installing SuSE Linux to logical volume(s) the SuSE installer will detect it and automatically create an initrd (initial ramdisk) with LVM support and configure the Linux loader appropriately, so that the root filesystem can be mounted from the LV. The difference to a setup where the root−system is stored on a physical partition and LVM only manages extra space is that LVM is usually activated during boot time, but for this setup it already has to be active before the just−started kernel can even mount the root filesystem to start the init process.

Also note that the kernel image and the initrd image must **not** be located on a logical volume since the LInux LOader (LILO) is not (yet?) LVM aware, so you have to have at least one physical boot partition to boot from somewhere (size: 4−30 MB). Installation to logical volume(s) is supported starting in SuSE Linux 7.1. You can do it with SuSE Linux 7.0 as well but have to do much more manual work, especially to setup the LILO and the initrd***(*)***.

To configure LVM during installation, type *manual* at the installation boot prompt to start manual installation (yast1 based; in SuSE Linux 7.2 it can also be done from the graphical installer). Start by selecting *Partitioning*. Do not immediately go into LVM setup, although it is presented to you right at the beginning next to the partitioning option and there is a partitioning option in the LVM setup dialog which could be used. The problem is there is a bug in the installer that lets the automated initrd and LILO configuration fail if one goes into the LVM setup immediately and does the partitioning from within that dialog. If you do the partitioning first, and press *Abort* when you get out of the partitioning dialog you will jump right back to the very beginning, where you are presented with the choice of doing nothing, *Partitioning*, *LVM setup* again. **Now** you can select the LVM setup option, **after** all the partitioning has been done.

***(*)** What is so special during an installation and has to be done manually in SuSE Linux 7.0 to logical volume(s) is that*

a) A special initrd (initial ramdisk, mounted as RAM filesystem during boot time to do initial setup stuff in preparation to mounting the real root filessystem; mostly used to load kernel modules needed to proceed) has to be created that contains support for LVM, i.e. the LVM kernel module and those command line tools needed to scan for and activate an existing LVM configuration. See /sbin/mk_initrd *to see what is done to the initrd.*

b) When the LILO installs the initial boot record on the disk/partition it checks if the specified root filesystem is valid. The problem is that at installation time the LVM setup including the device files for the logical volumes exists in the installation system only, but LILO runs chroot *in the installed system, which does not have those device files yet since they are created automatically as soon as that system boots up. Therefore the device files created in the RAM installation filesystem have to be transferred to the installed system before LILO can be run.*

These two steps happen automatically in SuSE Linux 7.1 and later if you follow the procedure above. The bug in the installer mentioned above prevents these two steps from happening, so should you encounter it because you didn't follow the advice you can do them manually.

USING LVM

With a logical volume you can do anything you can do with a physical partition. You can create filesystems and mount them, you can create a raw device mapping for raw (unbuffered character device−) access to the LV, you can even format and use them as swap space.

Creating a Filesystem on an LV

It works just like with any physical device, just use the device name of an LV instead:

ext2: `mkfs.ext2 /dev/vg−name/lv−name`

reiserfs: `mkreiserfs /dev/vg−name/lv−name`

You can of course also create and use any other filesystems supported by Linux.

Using an LV as Swap Space

First format the LV you want to use as swap space and then enter the device name of the LV into /etc/fstab:

```
mkswap /dev/vg−name/lv−name
swapon −a
```

and/or an entry in /etc/fstab:

```
/dev/vg−name/lv−name    swap    swap    defaults  0  2
```

You may or may not be able to do so during the installation already, depending on if the installer supports it. If not, just do it manually afterwards. Note that we do not discuss how useful this option is. It *may* be useful when you use striping (RAID0) to increase throughput. Otherwise it probably does not have any advantage over using regular swap partitions. We merely mention it in case the question comes up.

Resizing of Volume Groups (VGs)

Using either of the YaST1 LVM setup dialog, the YaST2 LVM module or of course the LVM command line tools (*vgreduce, vgextend)*.

YaST1: See the YaST1 screenshot in "Step 1 & 2" of *Setting up LVM using YaST1*, it is the same dialog. To remove a PV from a VG highlight it and press *SPACE*. Do the same to add a physical device (it will be initialized as a new PV automatically).

Command line tools: There is one command for extending a VG by more PVs (*vgextend*) and another one to remove PVs (*vgreduce*).

Add two more disks to a VG:

```
pvcreate /dev/device1 /dev/device2
vgextend vg−name /dev/device1 /dev/device2
```

Remove one PV:

```
vgreduce vg−name /dev/device
```

If you get an error message that the PV cannot be removed because it is still being used by an LV you will have to make sure there is no LV using it before you can proceed. If you have a striped LV one stripe may be on the PV you are trying to remove. See below *Moving Physical Extents (PEs)* for how to move LV storage to other PVs, or reduce the size of the LV(s) until the PV is not used any more.

Resizing of Logical Volumes (LVs)

To extend an LV there must be enough unallocated space available on the VG.

LVs itself can be extended or shrunk while they are being used, but this may not be true for a filesystem on them. If you extend an LV you can always do it online, i.e. while it is being used. However, the filesystem on the LV does not grow with the LV. You have to use a different command to also grow the filesystem.

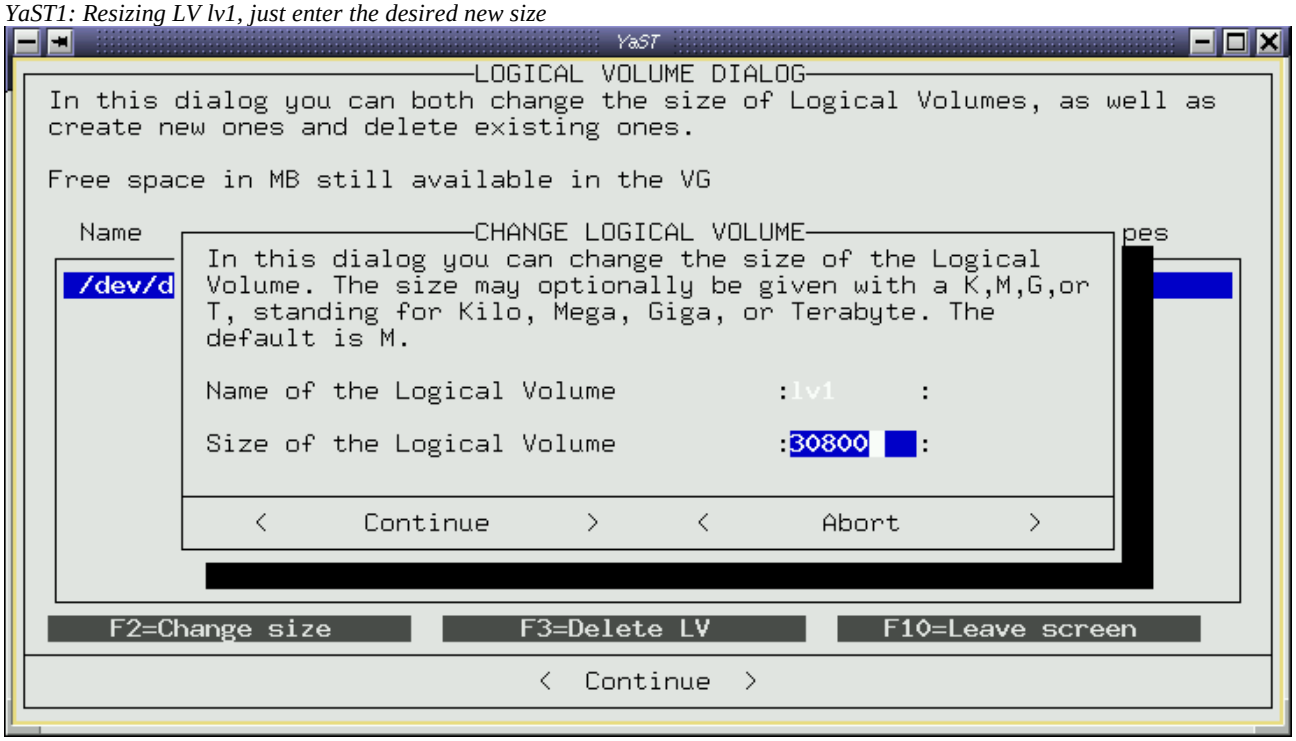
Make sure you use the right sequence: if you extend an LV, you have to extend the LV before the filesystem, if you shrink an LV you have to shrink the filesystem **first**.

For ext2, the command *e2fsadm* combines resizing of an LV and of the ext2 filesystem on it. It takes care of the correct sequence and uses the commands lvextend/lvreduce (growing/shrinking LVs) and e2fsck (ext2 fs check) and resize2fs (resize an ext2 fs) to accomplish the resizing. Ext2 filesystems **have** to be unmounted before they can be resized.

With ReiserFS you can extend the filesystem while it is mounted and being used, you have to unmount it for shrinking it only. You have to use separate commands for the LV and for the filesystem on it, there is no command that combines the two steps as for ext2.

The YaST1 interface (the Logical Volume dialog) can be used to resize LVs only but not the filesystem on them, so you will have to use some command line tool in any case.

YaST1: Resizing LV lv1, just enter the desired new size



Oracle and raw I/O: If you do not have a filesystem on the LV because you use the LV with raw I/O in Oracle, for example, the same holds true nethertheless. To extend the LV just extend it at any time, and then tell Oracle that it should use more space, e.g. through the Oracle Enterprise Manager or using the command line interface tools *sqlplus* or *svrmgrl*. Do the reverse for shrinking a tablespace: first tell Oracle to shrink the tablespace, and then you can go ahead and shrink the LV. The LV operations can be done online. Make sure you do not shrink the LV by a larger space than you shrunk the tablespace in Oracle, or data will be lost with no way to recover it!

Command line examples for resizing LVs:

Extend an LV with an ext2 filesystem on it by 1 GB:

```
umount /mountpoint-of-LV
e2fsadm -L+1G /dev/vg-name/lv-name
.... (output from the command) ....
mount /dev/vg-name/lv-name /mountpoint-of-LV
```

Extend an LV with a (**mounted and active**) ReiserFS on it by 512 MB:

```
lvextend -L+512M /dev/vg-name/lv-name
resize_reiserfs -s+512M -f /dev/vg-name/lv-name
```

Shrink an LV with a ReiserFS on it by 768 MB:

```
umount /mountpoint-of-LV
resize_reiserfs -s-768M /dev/vg-name/lv-name
lvreduce /dev/vg-name/lv-name
mount /dev/vg-name/lv-name /mountpoint-of-LV
```

Moving Physical Extents (PEs)

You can move all space utilized by an LV on a certain PV to somewhere else − while the LV is active. This has to be done using the command line tools. You can use the command *pvdata −E /dev/vg−name/lv−name* to see which PEs are used by which LV.

Note: You can use PEs while they are being used, i.e. while the LV is being used. However, if the system crashes while you are moving you may have a serious problem, so you better make sure you have a current backup before you start. *pvmove* may be interrupted by SIGINT while moving "next free" allocated LVs, but **not** when moving striped LVs, where moving must not be interrupted at all.

Also note that this command may take a while to finish, at least in the several−minutes range for one GB or slightly more, and exceeding one hour easily if there are lots of PEs to be moved. The speed was about 1 PE/second on our test machine with our specific test setup, and 1 PE had a size of 4 MB.

The *pvmove* command is very flexible. You always have to specify the PV you want to move things **away** from. If you don't give a destination PV it will allocate the next available space on any other PV(s) automatically.

As the source it is possible to use either only the PV, the PV and the LV and optionally a range of LEs, or the PV and a range of PEs. The easiest way is to only specify the LV to be moved and all PEs belonging to this LV will be moved somewhere else.

Should you want more control over what LEs/PEs are moved, note that because of the 1:1 mapping between LV/LEs on one side and PV/PEs on the other which of the two you choose is merely a matter of what is more convenient to you.

Examples:

Move all PEs from any LV off PV /dev/sdd (will be completely unused afterwards), where in the VG they are moved to, i.e. what other PV(s), is left to the system

```
pvmove /dev/sdd
```

Move all PEs that belong to LV *lv1* on PV /dev/sdd to PV /dev/ida/c0d1, because we are moving everything to an external RAID array **while the system is running:**

```
pvmove -n lv1 /dev/sdd /dev/ida/c0d1
```

Snapshot Logical Volumes

Snapshot logical volumes are possible since SuSE Linux 7.1 (LVM version 0.9). Snapshot devices provide **read−only** access to a logical volume image **frozen at the time of creation** of the snapshot, while the real LV continuous to be use and changed. This is useful for making consistent backups. Snapshot LVs have a relatively short lifetime, they are removed as soon as their purpose − e.g. a consistent backup of the LV − is fulfilled.

The size of the snapshot LV must be large enough to hold all changes made to the original LV during the life−time of the snapshot. The max. size is 1.1x the size of the original volume. If a snapshot LV becomes full it will become unusable!

Step 1: Create a snapshot LV:

```
lvcreate −L 512M −s −n snap−lv−name /dev/vg−name/lv−name
```

Step 2: Do the backup using the snapshot LV

Step 3: Delete a snapshot LV:

```
lvremove /dev/vg−name/snap−lv−name
```

You can create as many snapshot devices of an LV as you like.

Note: You should get at least LVM 0.9.1_beta4 if you plan to use snaphots.

Raw I/O on Logical Volumes

What is it

Raw I/O is often used by database software, for example Oracle®. It means an application gets direct access to the disk bypassing any kernel buffers via a character device. Such applications do their own buffering optimized for the application and prefer not to have the OS's buffering code interfer.

In Linux, raw devices exist independent of the block device for the physical or logical device, and the raw device is just an additional way of access. Raw devices are */dev/raw[0−9]+*. Some place them in a subdirectory like */dev/raw/raw**, but the Linux kernel documentation (linux/Documentation/devices.txt) says it should be */dev/raw**, so that's what SuSE does. Up to 255 raw devices are possible, the device */dev/raw* is a special device to control all raw devices.

We again benefit from LVM keeping its own hardware independent IDs on each PV: Normally, if you change the hardware IDs, e.g. /dev/sdc becomes /dev/sdd because you inserted a disk with a SCSI id lower than the one the disk that used to be sdc had, you have to change your raw I/O mappings to point to the correct device again. No such thing in LVM. Whatever changes on the hardware level − and that includes taking the disk and putting it in a completely different place than before so that it gets a completely different device name (from the internal controller to an external array, for example) − the raw I/O mapping stays the same, since it refers to the *logical* device.

***Tip: dd* on raw devices**

If the command *dd if=/dev/raw# of=(somewhere)* does not work and you get "Invalid argument" try to add *bs=4k* to the *dd* command. Also, you can always do a *dd* on the block device the raw device is bound to.

How to set it up

To setup and query raw device settings you can use the tool *raw:*

```
Usage:
   raw /dev/rawN <major> <minor>
   raw /dev/rawN /dev/<blockdev>
   raw -q /dev/rawN
   raw -qa
```

To make for example */dev/raw1* a raw device to access */dev/sda3* (the 3rd partition of the 1st SCSI disk), use the command `raw /dev/raw1 /dev/sda3`.

The raw device settings are lost after shutdown, since they are stored in kernel memory only. That means you will have to issue all *raw* commands again after a reboot. A good place to store all raw device settings is the file */etc/init.d/boot.local*. We recommend that you put all your raw device settings there. Another advantage of having them all in a file over issuing them manually is that when you have lots of raw devices − e.g. for Oracle Parallel Server certification more than 100 are used for the different controlfiles, tablespaces, etc. − you can look in this file which raw device stands for what physical or logical device, since per default the raw devices all have the same name (*raw#*). You can also recreate the raw device files using your own names, of course, using *mknod*. Raw devices have the major number 162, and minor numbers 1−255 (0 is reserved for the raw I/O control device, the master device).

To use LVs as the devices for raw I/O, simply use the device name of the LV */dev/vg−name/lv−name* with the *raw* command. There is no difference to using physical devices like partitions. The one difference that does exist is that LVs are much more flexible and easier to control than partitions. It makes running Oracle on raw devices much easier, from an administrators perspective! You can of course resize, move, snapshot, etc. the LVs as usual.

By default SuSE Linux comes with 63 raw device files (*/dev/raw**) in place out of 255 possible. If you need more than that you will have to create additional raw devices. These commands will create all 255:

```
cd /dev; rm -f raw[0-9]*   # to avoid error messages below
for ((i=1; $i<=255; i=$i+1)); do mknod raw$i c 162 $i; done
```

Optionally, if you prefer to have meaningful names rather than *raw#*, you can recreate the device files using your own names. This may make it a lot easier later when you have maintain the setup.

Access control

To control access to the raw devices you may want to change the ownership or the modes on the raw device files. For Oracle, for example, you may want to make the user *oracle* the owner of the files with a

```
chown oracle:dba /dev/raw[0-9]*
```

You do **not** have to change owner, group or mode of the blockdevices, but **only** of the raw devices, because Oracle will never use the block device files!

Setting up raw I/O for Oracle

First, the support page for running Oracle on SuSE Linux is at http://www.suse.com/en/support/oracle/, a pretty active mailinglist *suse−oracle@suse.com* is also available, see the URL.

The use of raw devices is a **must** when using Oracle Parallel Server, which uses a storage shared by all cluster nodes, with equal access for all of them. This is because of the OS's buffer caches for block devices. Oracle must be sure that something it writes to the database is really written. This can also be achieved on block devices, but what can not be done there is that other nodes on the shared storage pick up the change. They may think that they have the requested blocks in their buffer cache already and not look on the disk. That is why the buffer cache needs to be bypassed and raw I/O is the only option for OPS.

The use of raw devices is recommended for "regular" (non−OPS) production databases as well.

To create a database that uses raw I/O on LVs instead of files with Oracle, follow these steps:

1. Find out what the min. number and sizes of files is that you would need to create your database. That will be control files, tablespaces and redologs, and one very small LV for the cluster manager if you use OPS. You *can* create a mixed setup with some of those items as files and others as raw I/O.
2. Create an LVM setup where you have as many LVs as you would need files for the Oracle database. Create the LVs with the same size or larger than you are going to specify for the Oracle database creation. We recommend to use meaningful LV names, like */dev/oracle/cntl1* for the controlfile or */dev/oracle/system01* for the system tablespace.
3. Create raw device mappings in */etc/init.d/boot.local*. Again, we recommend you add meaningful comments like "*# control files*" and group the *raw* commands. Otherwise it will be very easy later on to get lost and to forget which raw device is supposed to stand for which Oracle file.
4. Make the raw devices writable for the Oracle database: `chown oracle:dba /dev/raw[0-9]*`
5. Execute */etc/init.d/boot.local* to activate the raw device settings
6. In the database creation script or in *dbassist*, the Java GUI tool Oracle provides to create a default database, adjust all values, leave the filenames as they are for now and at the end of the dialogs choose the option to save the database creation script in a file. Do **not** start the db creation process! Unfortunately dbassist has a check built-in if the file you enter for each tablespace already exists and refuses to continue if it does. This function makes it impossible to directly create a DB using raw devices, because those devices exist already, of course.
7. Now edit the files created by dbassist. Replace any paths and filenames with the appropriate device name and path of the raw device (**not** the physical or logical (LV) device). That should be the files $ORACLE_SIDrun.sh (*CREATE DATABASE...*) and $ORACLE_SIDrun1.sh (*CREATE TABLESPACE...*) at least, depending on what options you selected in dbassist there are more files to be edited. Watch for CREATE TABLESPACE commands, because those and the CREATE DATABASE command are the ones where filenames are specified.

   Also edit $ORACLE_HOME/dbs/init$ORACLE_SID.ora, to change the controlfiles to raw devices.

   The result looks like this, here for tablespace ROLLBACK:

   ```
   ...
   REM ********** TABLESPACE FOR ROLLBACK **********
   CREATE TABLESPACE RBS DATAFILE '/dev/raw7' SIZE 256M REUSE
    AUTOEXTEND ON NEXT 5120K
   MINIMUM EXTENT 512K
   DEFAULT STORAGE ( INITIAL 512K NEXT 512K MINEXTENTS 8 MAXEXTENTS 4096);
   ...
   ```

   Make sure you use the correct raw device which is bound to the correct LV which has the correct min. size for what Oracle wants to store in it. This may be a little confusing depending on how carefully you selected your names or if you do it for the first time, but the procedure is no different from using raw I/O on physical devices rather than LVs, only that they are much harder to setup and maintain than LVs.
8. Now create the database!

If you get an error, check the raw device mappings in the script you used to create them (checking them via *raw -qa* is very inconvenient), and check the sizes of the LVs. If the LV is smaller than the tablespace to be created you will get an I/O error.

See Appendix E for info what LVM should be used, because some versions will **not** work with Oracle because of blocksize limitations.

APPENDIX A: NAMING CONVENTIONS

Physical Volume (PV)

A physical volume is any regular harddisk. It can also be a logical drive provided by a hardware RAID controller, which from the operating systems point of view is just a harddisk. Both IDE and SCSI devices are possible. You can even use software RAID to simulate a hard drive, but we recommend a hardware solution is RAID is to be used. A PV is divided into up to 65535 PEs (see below).

Volume Group (VG)

A volume group contains a number of physical volumes (PVs). It groups them into one logical drive − the volume group. The volume group is the equivalent of a hard drive.

Logical Volume (LV)

A logical volume is a part of a volume group. It is the equivalent of a partition. Logical volumes is what really contains the data. Logical volumes are created in volume groups, can be resized within the boundaries of their volume group and even moved to other groups. They are much more flexible than partitions since they do not rely on physical boundaries as partitions on disks do.

Physical Extent (PE)

Logical volumes are made up of physical extends, the smallest physical storage unit in an LVM system. Data is stored in PEs that are part of LVs that are in VGs that consist of PVs. It means any PV consists of lots of PEs, numbered 1−m, where m is the size of the PV divided by the size of one PE. See the command *pvdata*.

Logical Extent (LE)

Similar to PE, but they describe the extents of a logical volume − which in the end are physical extents when it comes to where and how it is stored. Try *pvdata −E /dev/vg−name/lv−name* to see PEs and LEs. LEs are built in RAM out of the PEs. For example, if in one VG you have two PVs with PEs 1−4000 each (i.e. their sizes are 4000*size−of−one−PE each) and one LV spanning the two completely (two concatenated disks), you will have this LVs LEs 1−4000 mapped to PEs 1−4000 of PV 1 and the LEs 4001−8000 mapped to PEs 1−4000 of PV 2.

Snapshot Logical Volume

Creating a snapshot logical volumes grants access to the contents of the original logical volume it is associated with and exposes the read only contents at the creation time of the snapshot. This is useful for backups or for keeping several versions of filesystems online.

VGDA

The volume group descriptor area (or VGDA) holds the metadata of the LVM configuration. It is stored at the beginning of each physical volume. It contains four parts: one PV descriptor, one VG descriptor, the LV descriptors and several PE descriptors. LE descriptors are derived from the PE ones at activation time. Automatic backups of the VGDA are stored in files in /etc/lvmconf/ (please see the commands vgcfgbackup/vgcfgrestore too). Take care to include these files in your regular (tape) backups as well.

APPENDIX B: LVM COMMAND OVERVIEW

All general LVM commands start with *lvm*, all commands for manipulating physical volumes begin with *pv*, the volume group commands start with *pv* of course, and all those beginning with *lv* are for logical volumes.

For all commands there exists a manual page.

Filesystem Commands	
e2fsadm	***LVM/Ext2 tool:*** *Resize an LV with an ext2 filesystem on it. The filesystem must be unmounted. This command uses the commands lvextend/lvreduce and e2fsck and resize2fs. It takes care of resizing the LV and the fs in the correct order and by the same amount.*
resize2fs	***Ext2 tool:*** *Resize an ext2 filesystem on a physical partition. The filesystem must be unmounted.*
resize_reiserfs	***ReiserFS tool:*** *Resize a ReiserFS filesystem. Growing it can be done online, i.e. while the filesystem is mounted and being used! For shrinking it must be unmounted.*

General LVM Commands	
lvmdiskscan	*Displays all SCSI, (E)IDE and multiple devices ("software RAID") available on the system.*
lvmcreate_initrd	***Do not use on SuSE Linux!*** *Creates an initial ramdisk for systems where the root filesystem is on a logical volume. On SuSE Linux the script* /sbin/mk_initrd *already takes care of that!*
lvmchange	*Command to reset LVM. Do not use when any LVM volume is active!*
lvmsadc/lvmsar	*System connectivity data collector and system activity reporter. Collection and reporting of the read/write statistics of the logical volume manager. Equivalent of the sadc/sar tools available for non−LVM storage.*

Physical Volumes (PVs)	
pvscan	*Scan all available SCSI, (E)IDE disks and multiple devices (software RAID) for physical volumes.*
pvcreate	*Create a physical volume. Target can be a partition (partition id must be 8e), a whole disk (must not have a partition table − use dd if=/dev/zero of=/dev/name−of−disk bs=1k count=10 to erase, followed by* `blockdev --rereadpt <device>` to tell the kernel to re−read the partition table*), or multiple devices (software RAID).*
pvchange	*Change attributes on physical volumes.*
pvdisplay	*Allows viewing of the attributes of physical volumes.*
pvdata	*Shows debugging information about a physical volume. Uses the (volume group descriptor array (VGDA) of a physical volume.*
pvmove	*Allows moving the logical/physical extents allocated on one logical/physical volume to one or more other physical volumes.*

Volume Groups (VGs)	
vgscan	*Scan all disks for volume groups and build /etc/lvmtab and /etc/lvmtab.d/* which are the database for all other lvm commands.*
vgdisplay	*Display the attributes of a volume group with it's physical and logical volumes and their sizes etc.*
vgextend	*Add physical volumes to a volume group to extent its size.*
vgmerge	*Merge two volume groups*
vgreduce	*Remove one or more unused physical volumes from a volume group.*
vgrename	*Rename a volume group.*
vgsplit	*Splits physical volume(s) from an existing volume group into a new volume group.*
vgremove	*Allows you to remove one or more volume groups. The volume group(s) must not have any logical volumes allocated and must also be inactive.*
vgchange	*Change the attributes of a volume group, e.g. activate it.*
vgcreate	*Creates a new volume group with at least one physical volume in it.*
vgexport	*Allows making one or more inactive volume groups unknown to the system. You can then de−install the physical volumes and move them to a different system.*
vgimport	*Allows making a volume group known to the system which has previously been exported on this or another system.*
vgmknodes	*Creates the volume group directory and special (device) files for an existing volume group. This is to restore deleted special (device) files. Owner, group and file permissions will not be restored.*
vgcfgbackup/vgcfgrestore	*Backup/restore volume group descriptor data. This allows you to backup the metadata or so called VGDA (volume group descriptor area) of one to all volume groups to files in /etc/lvmconf.*

Logical Volumes (LVs)	
lvscan	*Scans all known volume groups or all SCSI, (E)IDE disks and multiple devices in the system for defined logical volumes.*
lvcreate	*Creates a new logical volume in a volume group.* *The second form supports the creation of **snapshot logical volumes** which keep the contents of the original logical volume for backup purposes.*
lvremove	*Allows removal of one or more inactive logical volumes.*
lvextend	*Allows extending the size of a logical volume by allocating unused space in the volume group the logical volume is located in.*
lvrename	*Renames an existing logical volume.*
lvchange	*Change the attributes of a logical volume, e.g. activate it.*
lvreduce	*Allows reducing the size of a logical volume. Be careful when reducing a logical volume's size, because data in the reduced part is lost!*
lvdisplay	*Allows viewing the attributes of a logical volume like size, read/write status, snapshot information etc.*

APPENDIX C: LIMITS

Max. number of VGs:	99
Max. number of LVs:	256
Max. size of an LV:	512 MB − 2 TB (32 bit), 1 Petabyte (64 bit) depends on PE size
Max. size of storage under LVM control:	256 Terabyte (using all 128 SCSI subsystems)
Max. number of Logical Extents:	65534
Max. number of Physical Extents per PV:	65534

Currently up to 99 volume groups with a grand total of 256 logical volumes can be created. The limit for the logical volumes is not caused by the LVM but by Linux 8 bit device minor numbers.

This means that you can have 99 volume groups with 1−3 logical volumes each or on the other hand 1 volume group with up to 256 logical volumes or anything in between these extreme examples.

Depending on the physical extent size specified at volume group creation time, logical volumes of between a maximum of 512 Megabytes and 1 Petabyte can be created. Actual Linux kernels on IA32 limit these LVM possibilities to a maximum of 2 Terabytes per logical and per physical volume as well. This enables you to have as much as 256 Terabytes under LVM control with all possible 128 SCSI disk subsystems. You can have up to 65534 logical extents (on IA32) in a logical volume at the cost of 1 Megabyte in kernel memory. Physical volumes can have up to 65534 physical extents.

Max. external storage with Linux

Due to driver restrictions there’s hardly any support for more than 1−2 TB per block device.

If you install hardware RAID (like HP’s AutoRaid, EMC’s Symetrics) systems on all 128 possible SCSI IDs, which can offer 2 TB on up to 8 LUNs (if theres enough space for all the disks) you get to the theoretical limit of 128 * 8 * 2 TB = 2 Petabyte.

APPENDIX D: TROUBLESHOOTING

Almost everything here is from the FAQ page on http://www.sistina.com/lvm/

Q: I cannot create a PV on my harddisk (using the whole disk, not just partitions).

A: If the harddisk contains a partition table − even an empty one − it is not possible to use the entire harddisk device (e.g. /dev/sda) as PV. After erasing the partition table by *dd if=/dev/zero of=/dev/name−of−hd−device bs=1k count=1* it will work.

Q: Where can I find more information about the Logical Volume Manager for Linux?

A: Please have a look at the LVM home page at http://www.sistina.com/lvm/ and/or join the LVM mailing list by sending "subscribe linux−lvm" in the body of a mail to majordomo@msede.com. Information on the LVM mailing list and the list archive is available at http://www.sistina.com/lvm/mlist/.

Q: Why can't I create my volume group "mygroup"?

A: Possible reasons:
* You can't create a volume group with the same name as an existing one.
* You can't create more than 99 volume groups at this point in time.
* You are only able to use inititalized physical volumes (see *pvcreate*(8)).

Q: Why can't I create a physical volume with *pvcreate*(8)?

A: Remember to set the partition system id to 0x8e with *fdisk*(8) before trying *pvcreate*(8) on it (previous versions of LVM used partition id 0xfe).
Maybe pvcreate(8) complains that the physical volume has already been initialized. You can force the operation by using the "*pvcreate −f[f] ...*" option. Be careful: Don't try this if the physical volume belongs to another volume group!
If you are trying to use a whole disk remember that the first sector must be zeroed (there must be no partitioon table). Use the command *dd if=/dev/zero of=/dev/disk−device bs=1k count=10* to achieve that.

Q: Why am I not able to extend a logical volume?

A: Your volume group is full or you've already reached the maximum logical volume size in that volume group. Logical volume size is limited by the size of the physical extents times their maximum amount, which only can be set at volume group creation time.
The default physical extent size is 4MB which limits ;−) logical volumes to a maximum of 256 Gigabyte (see *vgcreate*(8), *vgdisplay*(8)). If your volume group isn't full or you didn't reach the current logical volume size limit, your logical volume may have striped or contiguous allocation policy. Have a look at the physical volumes with *vgdisplay*(8) or *pvdisplay*(8) to figure out, if there are not enough free (contiguous) physical extents.

Q: Why can't I move my logical volume(s) away from a physical volume with *pvmove*(8).

A: Look at the free space on all destination disks you want to use (or which are implicit used) AND at the attributes of the logical volumes to be moved.
* You can't move a contiguous logical volume when there isn't enough free contiguous space on any destination disk. In this case you can think about changing from contiguous allocation policy to next free and do the attribute change with *lvchange*(8).
* You can't move a striped logical volume either, if there isn't enough space for the complete stripe on any destination physical volume.
* You can't move to physical volumes which are NOT allocatable. Think about changing this with *pvchange*(8).

Q: My striped logical volume is horribly slow.

A: If you put it on two or more physical volumes based on partitions on one disk, you are not able gain performance.
* You are allowed to use two or more partitions of one disk as physical volumes (this only makes sense for next free allocated logical volumes on those physical volumes).
* If you have attached two IDE disks to one adapter, you can't get parallel I/O on these two disks.

Q: Why am I not able to rename my volume group / logical volume?

A: You have to deactivate them before you are allowed to rename them (see *lvrename*(8), *vgrename*(8)).

Q: The LVM kernel patch in the LVM distribution is not compatible with my Linux version. Where's help?

A: The safest thing is to use what is provided as part of SuSE Linux or as an update for SuSE Linux on ftp.suse.com. Only if you know what you are doing should you try to patch anything yourself. Insthis case, please have a look at http://www.sistina.com/lvm/ for additional LVM kernel patches. If there is no corresponding one available send a mail request with the subject "LVM kernel patch request" to linux−lvm@ez−darmstadt.telekom.de.
Don't forget to bring your LVM tools to the same version number!

Q: A LVM command was just working when my system crashed...

A: Bring your system back online and look at the volume group backup files in /etc/lvmconf.

There's at least one called /etc/lvmconf/VolumeGroupName.conf and possible more in the backup history called /etc/lvmconf/VolumeGroupName.conf.*.old. You can use these backup files to bring the configuration back to the one before the crash (see *vgcfgrestore*(8)).

Q: Why are my logical volumes limited to 256 GB in size?

A: This is NO absolute limit but it depends on the physical extent size you configured at volume group creation time.
Please use option −s of the *vgcreate* command to give a larger physical extent size. For example with a physical extent size of 524288 KB (512 MB) you are able to map a logical volume of 32 Terabyte. Remember that current Linux kernels are limited to 1 Terabyte.

Q: Why can't I split my volume group my_vg?

A: The physical volumes you want to split of into another volume group may NOT have logical extents of logical volumes staying in the original volume group you started with. Please use *pvmove* to seperate the logical volumes.

Q: Why can't I merge my two volume groups my_vg1 and my_vg2?

A: A merged volume group can't go beyond the physical or logical volume limits of the destination volume group. This means for eg. that you can't merge my_vg1 with 20 logical volumes and my_vg2 with 30 logical volumes getting my_vg1, if my_vg1 has a 31 logical volume limit. You are only able to merge (up to now) volume groups with equal physical extent sizes.

Q: How can I move parts of my logical volume with very intensive i/o to a different physical volume?

A: Please look at pvmove(8) and use the logical extent syntax to do the job.

Q: I have a compile problem and/or a runtime problem with the LVM.

A: Please send a bug mail request to linux−lvm−bug@ez−darmstadt.telekom.de or to the linux−lvm mailing list.

Q: Where can I ask for a missing feature or for a better way to implement something in the LVM?

A: Please send me an enhancement mail request to linux−lvm−enhancement@ez−armstadt.telekom.de or to the linux−lvm mailing list.

Q: Where can I send a patch for the LVM?

A: Please remember to make a unified diff to the original LVM distribution and mail the diff with your comments to linux−lvm−patch@ez−darmstadt.telekom.de or to the linux−lvm mailing list.

Q: Why am I not able to create my 211th logical volume? vgdisplay tells me about a limit of 256 logical volumes.

A: The total amount of 256 logical volumes is shared among all volume groups. You have to delete logical volumes in different volume groups to be able to create the new one.

Q: Can I have my root filesystem in a logical volume?

A: Yes you can. Starting with SuSE Linux 7.1 the script */sbin/mk_initrd* includes everything needed to support this on SuSE Linux. Therefore you can ignore the *lvmcreate_initrd* script that comes with the LVM tools. Also, in SuSE Linux 7.1 the yast1 installer and in SuSE Linux 7.2 also the YaST2 graphical installer support this. Please see the section *Installing the root filesystem on LVM* above in this document for more information!

Q: LVM tools complain about "invalid I/O protocol version"

A: The LVM kernel and user mode parts use common data structures to exchange information. The I/O protocol version specifies the version of this data structures. The kernel and user mode parts must use the same I/O protocol version in order to communicate.
If you get this error message you need to obtain a new version of the LVM tools which uses a compatible protocol version. Maybe you upgraded only the kernel or only the lvm.rpm package with an update package from ftp.suse.com?

Q: Why does LVM not support mirroring?

A: Mirroring or other levels of RAID on a per Logical Volume base decreases reliability because you have to take care of which Logical Volumes are mirrored and which ones are not.
I (Heinz Mauelshagen, author of LVM) recommend to use dedicated hardware RAID subsystems or Multiple Devices to have the redundancy below the LVM. In this case you just don’t care about Logical Volume data redundancy and you don’t run into the dangerous situation that your data is not redundant by accident.
If you do have these kind of devices in place you can setup RAID 10/40/50 if you like to because the LVM supports RAID0. Combining for eg. several hardware RAID5 subsystems in volume group to set up LVM−RAID0 gives you reliability, performance and flexibility at the same time.

Q: When I create a database using raw I/O on LVs I get an error *ORA−27072: skgfdisp: I/O error*.

A: Your LV is smaller than the tablespace or other object you are trying to create in it. Please make sure the LV the raw device is bound to has indeed at least the size of the object you are trying to create.

APPENDIX E: LVM IN SUSE LINUX

The following table contains an overview over which LVM version is bundled with which SuSE Linux, what upgrade options to newer LVM versions exist and if the newer versions are compatible with older versions or not. The updates can be found in the update section for the particular SuSE Linux release on *ftp.suse.com*.

The last column shows what level of LVM setup is possible with a given SuSE Linux version during the installation. It is of course possible to do about anything using the command line tools in 7.0 already. The one difficult option is support for the root filesystem *"/"* on a logical volume. Using an LV for */usr* or for */home* is possible with all SuSE Linux versions starting with 6.4. Support for the root filesystem on an LV was added in SuSE Linux 7.1.

SuSE Linux	LVM	LVM−Updates	Configuration Tools	Options during SuSE Linux Installation
6.x	none	none	none	N/A
6.3	0.8	none	command line, yast1	YaST Use LVs for any mount point except for "/"
6.4	0.8	none	command line, yast1	YaST Use LVs for any mount point except for "/"
7.0	0.8	0.9.1_beta7	command line, yast1	YaST Use LVs for any mount point except for "/"
7.1	0.9	0.9.1_beta7	command line, yast1	YaST1 Use LVs for any mount point
7.2	0.9.1_beta7		command line, yast1, yast2	YaST1, (graph.) YaST2 Use LVs for any mount point

LVM Compatibility:

Get the kernel and LVM updates from ftp.suse.com. This is what we have tested. Other combinations may not work. For example, when updating an LVM 0.8 system like SuSE Linux 7.0 to LVM 0.9.1_beta4 the user level LVM tools do not recognize your LVM setup! That is not an issue with the LVM update for 7.0 on ftp.suse.com, however. If updating to SuSE Linux 7.1 also get the kernel/lvm update for 7.1, otherwise you may run into this issue!

Please note that the last 2.2 kernel LVM version is 0.9.1_beta4. It is safe (and recommended) to use the user level tools version 0.9.1_beta7 even with that 2.2 kernel LVM. You can safely ignore error messages by the user level tools (ioctl errors), they try to use the new ones in kernel 2.4 but if that doesn't work (error message) they automatically use the older 2.2 kernel ones for LVM.

Blocksize issues when using raw I/O on LVM

LVM version 0.9 introduced a problem with blocksizes when doing I/O, so that for example Oracle, which insists on doing 512−Byte blocksize I/O for some critical (atomic) I/O operations would not work any more.

After finding the issue in SuSE Linux 7.1, which was released with LVM 0.9, we fixed it in the update kernel and lvm tools available from ftp://ftp.suse.com/pub/suse/i386/update/7.1/

There is also an issue when you upgrade SuSE Linux 7.0 LVM 0.8 created setups to 7.1 and LVM 0.9. After the update *vgscan* may report "no volume groups found". This kernel and lvm update fixes this issue as well.

CHANGELOG FOR THIS DOCUMENT

To obtain the latest version of this document go to **http://www.suse.com/en/support/oracle/**

27 April, 2001: Initial release

2 May, 2001: Major corrections to the R*aw I/O with Oracle* section. You cannot use dbassist.

3 May, 2001: Many minor corrections all over the document. Completely reworked the section *How it really works*, since the example for the LE−>PE mapping was very wrong.
Added info to Appendix E about which LVM/SuSE kernel versions can be used for raw I/O on LVM.

7 May, 2001: Some minor corrections (~10) like pointing to a different LVM website or pointing out that you can have any number of snaphot LVs at the request of the author.

6 June, 2001: One typo fixed. Appendix E (version compatibility) updated.

23 July, 2001:
− Changed path /etc/rc.d/ to /etc/init.d/ (rc.d is a link to init.d).
− Added to each place where the partition table is erased with *dd*: How to inform the kernel the partition table has changed (it won't re−read it if it's cashed), with `blockdev −−rereadpt <diskname>`.
− Added note to *LVM compatibility* section about user level tools 0.9.1_beta7 and 2.2 kernel LVM 0.9.1_beta4.