NAME

lvm — LVM2 tools

SYNOPSIS

lvm [*command*|*file*]

DESCRIPTION

The Logical Volume Manager (LVM) provides tools to create virtual block devices from physical devices. Virtual devices may be easier to manage than physical devices, and can have capabilities beyond what the physical devices provide themselves. A Volume Group (VG) is a collection of one or more physical devices, each called a Physical Volume (PV). A Logical Volume (LV) is a virtual block device that can be used by the system or applications. Each block of data in an LV is stored on one or more PV in the VG, according to algorithms implemented by Device Mapper (DM) in the kernel.

The lvm command, and other commands listed below, are the command-line tools for LVM. A separate manual page describes each command in detail.

If **lvm** is invoked with no arguments it presents a readline prompt (assuming it was compiled with readline support). LVM commands may be entered interactively at this prompt with readline facilities including history and command name and option completion. Refer to **readline**(3) for details.

If **lvm** is invoked with argv[0] set to the name of a specific LVM command (for example by using a hard or soft link) it acts as that command.

On invocation, **lvm** requires that only the standard file descriptors stdin, stdout and stderr are available. If others are found, they get closed and messages are issued warning about the leak. This warning can be suppressed by setting the environment variable **LVM_SUPPRESS_FD_WARNINGS**.

Where commands take VG or LV names as arguments, the full path name is optional. An LV called "lvol0" in a VG called "vg0" can be specified as "vg0/lvol0". Where a list of VGs is required but is left empty, a list of all VGs will be substituted. Where a list of LVs is required but a VG is given, a list of all the LVs in that VG will be substituted. So **lvdisplay vg0** will display all the LVs in "vg0". Tags can also be used − see **−−addtag** below.

One advantage of using the built-in shell is that configuration information gets cached internally between commands.

A file containing a simple script with one command per line can also be given on the command line. The script can also be executed directly if the first line is #! followed by the absolute path of **lvm**.

Additional hyphens within option names are ignored. For example, **−−readonly** and **−−read−only** are both accepted.

BUILT-IN COMMANDS

The following commands are built into lvm without links normally being created in the filesystem for them.

config	The same as **lvmconfig**(8) below.
devtypes	Display the recognised built-in block device types.
dumpconfig	The same as **lvmconfig**(8) below.
formats	Display recognised metadata formats.
fullreport	Report information about PVs, PV segments, VGs, LVs and LV segments, all at once.
help	Display the help text.
lastlog	Display log report of last command run in LVM shell if command log reporting is enabled.
lvpoll	Complete lvmpolld operations (Internal command).
segtypes	Display recognised Logical Volume segment types.
systemid	Display any system ID currently set on this host.
tags	Display any tags defined on this host.
version	Display version information.

COMMANDS

The following commands implement the core LVM functionality.

pvchange	Change attributes of a Physical Volume.
pvck	Check Physical Volume metadata.
pvcreate	Initialize a disk or partition for use by LVM.
pvdisplay	Display attributes of a Physical Volume.
pvmove	Move Physical Extents.
pvremove	Remove a Physical Volume.
pvresize	Resize a disk or partition in use by LVM2.
pvs	Report information about Physical Volumes.
pvscan	Scan all disks for Physical Volumes.
vgcfgbackup	Backup Volume Group descriptor area.
vgcfgrestore	Restore Volume Group descriptor area.
vgchange	Change attributes of a Volume Group.
vgck	Check Volume Group metadata.
vgconvert	Convert Volume Group metadata format.
vgcreate	Create a Volume Group.
vgdisplay	Display attributes of Volume Groups.
vgexport	Make volume Groups unknown to the system.
vgextend	Add Physical Volumes to a Volume Group.
vgimport	Make exported Volume Groups known to the system.
vgimportclone	Import and rename duplicated Volume Group (e.g. a hardware snapshot).
vgmerge	Merge two Volume Groups.
vgmknodes	Recreate Volume Group directory and Logical Volume special files
vgreduce	Reduce a Volume Group by removing one or more Physical Volumes.
vgremove	Remove a Volume Group.
vgrename	Rename a Volume Group.
vgs	Report information about Volume Groups.
vgscan	Scan all disks for Volume Groups.
vgsplit	Split a Volume Group into two, moving any logical volumes from one Volume Group to another by moving entire Physical Volumes.
lvchange	Change attributes of a Logical Volume.
lvconvert	Convert a Logical Volume from linear to mirror or snapshot.
lvcreate	Create a Logical Volume in an existing Volume Group.
lvdisplay	Display attributes of a Logical Volume.
lvextend	Extend the size of a Logical Volume.
lvmconfig	Display the configuration information after loading **lvm.conf**(5) and any other configuration files.
lvmdiskscan	Scan for all devices visible to LVM2.
lvmdump	Create lvm2 information dumps for diagnostic purposes.
lvreduce	Reduce the size of a Logical Volume.
lvremove	Remove a Logical Volume.
lvrename	Rename a Logical Volume.
lvresize	Resize a Logical Volume.
lvs	Report information about Logical Volumes.
lvscan	Scan (all disks) for Logical Volumes.

The following LVM1 commands are not implemented in LVM2: **lvmchange**, **lvmsadc**, **lvmsar**, **pvdata**. For performance metrics, use **dmstats**(8) or to manipulate the kernel device-mapper driver used by LVM2 directly, use **dmsetup**(8).

VALID NAMES

The valid characters for VG and LV names are: **a−z A−Z 0−9 + _ . −**

VG names cannot begin with a hyphen. The name of a new LV also cannot begin with a hyphen. However, if the configuration setting **metadata/record_lvs_history** is enabled then an LV name with a hyphen as a prefix indicates that, although the LV was removed, it is still being tracked because it forms part of the history of at least one LV that is still present. This helps to record the ancestry of thin snapshots even after some links in the chain have been removed. A reference to the historical LV 'lvol1' in VG 'vg00' would be 'vg00/\-lvol1' or just '−lvol1' if the VG is already set. (The latter form must be preceded by '−−' to terminate command line option processing before reaching this argument.)

There are also various reserved names that are used internally by lvm that can not be used as LV or VG names. A VG cannot be called anything that exists in */dev/* at the time of creation, nor can it be called '.' or '..'. An LV cannot be called '.', '..', 'snapshot' or 'pvmove'. The LV name may also not contain any of the following strings: '_cdata', '_cmeta', '_corig', '_mlog', '_mimage', '_pmspare', '_rimage', '_rmeta', '_tdata', '_tmeta', '_vorigin' or '_vdata'. A directory bearing the name of each Volume Group is created under */dev* when any of its Logical Volumes are activated. Each active Logical Volume is accessible from this directory as a symbolic link leading to a device node. Links or nodes in */dev/mapper* are intended only for internal use and the precise format and escaping might change between releases and distributions. Other software and scripts should use the */dev/VolumeGroupName/LogicalVolumeName* format to reduce the chance of needing amendment when the software is updated. Should you need to process the node names in /dev/mapper, you may use **dmsetup splitname** to separate out the original VG, LV and internal layer names.

UNIQUE NAMES

VG names should be unique. vgcreate will produce an error if the specified VG name matches an existing VG name. However, there are cases where different VGs with the same name can appear to LVM, e.g. after moving disks or changing filters.

When VGs with the same name exist, commands operating on all VGs will include all of the VGs with the same name. If the ambiguous VG name is specified on the command line, the command will produce an error. The error states that multiple VGs exist with the specified name. To process one of the VGs specifically, the −−select option should be used with the UUID of the intended VG: '-−select vg_uuid=<uuid>'.

An exception is if all but one of the VGs with the shared name is foreign (see **lvmsystemid**(7).) In this case, the one VG that is not foreign is assumed to be the intended VG and is processed.

LV names are unique within a VG. The name of an historical LV cannot be reused until the historical LV has itself been removed or renamed.

ALLOCATION

When an operation needs to allocate Physical Extents for one or more Logical Volumes, the tools proceed as follows:

First of all, they generate the complete set of unallocated Physical Extents in the Volume Group. If any ranges of Physical Extents are supplied at the end of the command line, only unallocated Physical Extents within those ranges on the specified Physical Volumes are considered.

Then they try each allocation policy in turn, starting with the strictest policy (**contiguous**) and ending with the allocation policy specified using **−−alloc** or set as the default for the particular Logical Volume or Volume Group concerned. For each policy, working from the lowest-numbered Logical Extent of the empty Logical Volume space that needs to be filled, they allocate as much space as possible according to the restrictions imposed by the policy. If more space is needed, they move on to the next policy.

The restrictions are as follows:

Contiguous requires that the physical location of any Logical Extent that is not the first Logical Extent of a Logical Volume is adjacent to the physical location of the Logical Extent immediately preceding it.

Cling requires that the Physical Volume used for any Logical Extent to be added to an existing Logical

Volume is already in use by at least one Logical Extent earlier in that Logical Volume. If the configuration parameter **allocation/cling_tag_list** is defined, then two Physical Volumes are considered to match if any of the listed tags is present on both Physical Volumes. This allows groups of Physical Volumes with similar properties (such as their physical location) to be tagged and treated as equivalent for allocation purposes.

When a Logical Volume is striped or mirrored, the above restrictions are applied independently to each stripe or mirror image (leg) that needs space.

Normal will not choose a Physical Extent that shares the same Physical Volume as a Logical Extent already allocated to a parallel Logical Volume (i.e. a different stripe or mirror image/leg) at the same offset within that parallel Logical Volume.

When allocating a mirror log at the same time as Logical Volumes to hold the mirror data, Normal will first try to select different Physical Volumes for the log and the data. If that's not possible and the **allocation/mirror_logs_require_separate_pvs** configuration parameter is set to 0, it will then allow the log to share Physical Volume(s) with part of the data.

When allocating thin pool metadata, similar considerations to those of a mirror log in the last paragraph apply based on the value of the **allocation/thin_pool_metadata_require_separate_pvs** configuration parameter.

If you rely upon any layout behaviour beyond that documented here, be aware that it might change in future versions of the code.

For example, if you supply on the command line two empty Physical Volumes that have an identical number of free Physical Extents available for allocation, the current code considers using each of them in the order they are listed, but there is no guarantee that future releases will maintain that property. If it is important to obtain a specific layout for a particular Logical Volume, then you should build it up through a sequence of **lvcreate**(8) and **lvconvert**(8) steps such that the restrictions described above applied to each step leave the tools no discretion over the layout.

To view the way the allocation process currently works in any specific case, read the debug logging output, for example by adding **-vvvv** to a command.

LOGICAL VOLUME TYPES

Some logical volume types are simple to create and can be done with a single **lvcreate**(8) command. The linear and striped logical volume types are an example of this. Other logical volume types may require more than one command to create. The cache (**lvmcache**(7)) and thin provisioning (**lvmthin**(7)) types are examples of this.

DIAGNOSTICS

All tools return a status code of zero on success or non-zero on failure. The non-zero codes distinguish only between the broad categories of unrecognised commands, problems processing the command line arguments and any other failures. As LVM remains under active development, the code used in a specific case occasionally changes between releases. Message text may also change.

ENVIRONMENT VARIABLES

HOME
Directory containing *.lvm_history* if the internal readline shell is invoked.

LVM_OUT_FD
File descriptor to use for common output from LVM commands.

LVM_ERR_FD
File descriptor to use for error output from LVM commands.

LVM_REPORT_FD
File descriptor to use for report output from LVM commands.

LVM_COMMAND_PROFILE
Name of default command profile to use for LVM commands. This profile is overriden by direct use of **−−commandprofile** command line option.

LVM_RUN_BY_DMEVENTD
This variable is normally set by dmeventd plugin to inform lvm2 command it is running from dmeventd plugin so lvm2 takes some extra action to avoid comunication and deadlocks with dmeventd.

LVM_SYSTEM_DIR
Directory containing **lvm.conf**(5) and other LVM system files. Defaults to "*/etc/lvm*".

LVM_SUPPRESS_FD_WARNINGS
Suppress warnings about unexpected file descriptors passed into LVM.

LVM_SUPPRESS_SYSLOG
Suppress contacting syslog.

LVM_VG_NAME
The Volume Group name that is assumed for any reference to a Logical Volume that doesn’t specify a path. Not set by default.

LVM_LVMPOLLD_PIDFILE
Path to the file that stores the lvmpolld process ID.

LVM_LVMPOLLD_SOCKET
Path to the socket used to communicate with lvmpolld..

LVM_LOG_FILE_EPOCH
A string of up to 32 letters appended to the log filename and followed by the process ID and a startup timestamp using this format string "_%s_%d_%llu". When set, each process logs to a separate file.

LVM_LOG_FILE_MAX_LINES
If more than this number of lines are sent to the log file, the command gets aborted. Automated tests use this to terminate looping commands.

LVM_EXPECTED_EXIT_STATUS
The status anticipated when the process exits. Use ">N" to match any status greater than N. If the actual exit status matches and a log file got produced, it is deleted. **L VM_LOG_FILE_EPOCH** and **LVM_EXPECTED_EXIT_STATUS** together allow automated test scripts to discard uninteresting log data.

LVM_SUPPRESS_LOCKING_FAILURE_MESSAGES
Used to suppress warning messages when the configured locking is known to be unavailable.

DM_ABORT_ON_INTERNAL_ERRORS
Abort processing if the code detects a non-fatal internal error.

DM_DISABLE_UDEV
Avoid interaction with udev. LVM will manage the relevant nodes in /dev directly.

DM_DEBUG_WITH_LINE_NUMBERS
Prepends source file name and code line number with libdm debugging.

FILES

/etc/lvm/lvm.conf
$HOME/.lvm_history

SEE ALSO

lvm(8) **lvm.conf**(5) **lvmconfig**(8)

pvchange(8) **pvck**(8) **pvcreate**(8) **pvdisplay**(8) **pvmove**(8) **pvremove**(8) **pvresize**(8) **pvs**(8) **pvscan**(8)

vgcfgbackup(8) **vgcfgrestore**(8) **vgchange**(8) **vgck**(8) **vgcreate**(8) **vgconvert**(8) **vgdisplay**(8) **vgexport**(8) **vgextend**(8) **vgimport**(8) **vgimportclone**(8) **vgmerge**(8) **vgmknodes**(8) **vgreduce**(8) **vgremove**(8) **vgrename**(8) **vgs**(8) **vgscan**(8) **vgsplit**(8)

lvcreate(8) **lvchange**(8) **lvconvert**(8) **lvdisplay**(8) **lvextend**(8) **lvreduce**(8) **lvremove**(8) **lvrename**(8) **lvresize**(8) **lvs**(8) **lvscan**(8)

lvm-fullreport(8) **lvm-lvpoll**(8) **lvm2−activation−generator**(8) **blkdeactivate**(8) **lvmdump**(8)

dmeventd(8) **lvmpolld**(8) **lvmlockd**(8) **lvmlockctl**(8) **cmirrord**(8) **lvmdbusd**(8)

lvmsystemid(7) **lvmreport**(7) **lvmraid**(7) **lvmthin**(7) **lvmcache**(7)

dmsetup(8), **dmstats**(8), **readline**(3)