Partitions, Filesystems, RAID and LVM

Author: Aaron Toponce

Date: 2009-05-09

License

Partition Planning

- Primary use? Data needs? I/O speeds? Sizes?
- RAID? LVM? Raw devices? Partitions?
- /, /var, /tmp, /usr, /home, /opt

When setting up a system from scratch, it's imperative that you spend some time planning out your partitioning structure, whether it be flat partitions, RAID, LVM or any mix. Common questions that should be asked are:

- What is the primary use of this system?

- What data needs to be stored?
- What I/O am I looking for on reads and writes?
- How much storage will be needed for various applications? Users?

Popular mount points include:

- / - Bare minimum for system installation.
- /var - Contains variable data, such as logs, spools, etc.
- /tmp - Contains temporary data, not crucial to persistence.
- /usr - Genearally the largest dir structure containing binaries.
- /home - Useful for keeping user data separate.
- /opt - 3rd party utilities, usually binaries.

Linux Partitions

- 4 primary partitions max
- 1 primary can be an extended partition
- 63 for IDE drives / 15 for SCSI drives
- `fdisk`, `sfdisk`
- `parted`, `gparted`

By default, Linux uses the partition scheme defined by Microsoft and IBM. This partitioning scheme only allowed a maximum of 4 primary partitions. It wasn't long before they realized the serious limitation, and gave the ability for one of the primary partitions to be an extended partition. In this extended partition, you can have more "logical partitions". If this is the case, because the partition table is only 64 bytes in size, four 16-byte segments, one partition each, the extended partition is a pointer to another partition table.

Linux does have limitations on the number of physical partitons that can reside on a single system. If the kernel is using the PATA driver and recognizes your disk as an IDE disk, then you have have a maximum of 63 partitions. If the kernel is using the SATA driver for SCSI or SCSI look-a-like hardware, such as SATA or SAS disks, then Linux can only address up to 15 total partitions. We'll later learn that LVM addresses this limitation rather nicely.

Of the utilities available for editing disk partitions, GNU parted is the most robust. It can create, destroy, resize and copy partition, and the filesystems on them. The GNU parted utility can operate in interactive mode, bugging the user for input as they go along, or non-interactive mode, requiring switches and command line arguments. One nice utility that people mention as a strength with Windows Vista and Windows 7 is the ability to resize their existing partitions without rebooting the box. GNU parted supportes the functionality, although I would recommend going offline for read-only support. There is a graphical utility called "gparted" which provides a slick GUI with a bootable CD for using GNU parted on your disk. Most distributions ship these by default.

Filesystem Creation

- **`mkfs -t filesystem_type`**
  - executes `mkfs.ext2`, `mkfs.ext3`, `mkfs.xfs`, etc.
- **Filesystem-specific utilities**
  - `mke2fs`, `mkreiserfs`, `mkdosfs`, etc

The standard utility that exists on all Linux and Unix operating systems is the `mkfs` utility. This utility requires the appropriate switches for telling the system which filesystem you want to put down on the partition. For example, `mkfs -t ext3 /dev/hda1` would create an `ext3` filesystem on `/dev/hda1`.

However, `mkfs` does not actually do any work creating the filesystem. Rather, it's a wrapper utility that calls filesystem-specific utilities. So, in your previous example, `mkfs -t ext.` would actually call the `mke2fs -j`, as the `ext3` filesystem is just the `ext2` filesystem with an online journal.

Many filesystem-specific utilities exist, such as `mkfs.ext2`, `mkfs.ext3`, `mkfs.xfs`, `mkfs.jfs`, `mkfs.reiserfs`, and so forth. There are also other utilites with more granular control over the placement of the filesystem with `mke2fs`, `mkreiserfs`, `mxdosfs` and so on.

Filesystem Tools

- `e2label device [fslabel]`
- `blkid`
- `mount LABEL=fslabel mount_point`
- `tune2fs device`
- `dumpe2fs`

Many tools exist in a system administrator's tool belt that is crucial for filesystem maintenance. Looking over the ext2/ext3 tools, there are a few tools that can be verbose in what parameters the filesystem has, such as block size, number of inodes, filesystem label, and default mount options, just to name a few.

Labeling a filesystem is useful to refer to that filesystem by another name than the default device path. Most distributions have moved from using device paths to using either labels or UUIDs. This provides a major advantage that hit sysadmins in the 2.6.20 kernel change. The kernel developers changed the default driver used for recognizing IDE drives to the SATA driver rather than the PATA driver. This means that the IDE drives now have the 15 total partition limit restriction that SCSI drives have, and they have been renamed from `/dev/hd*` to `/dev/sd*`. This was a problem for the `/etc/fstab`, as you box might not boot, due to the kernel looking for a different device than was now being recognized. If labels were used in the `/etc/fstab`, then this wasn't as big a deal.

`e2label` is used for creating, changing, and displaying filesystem labels on ext2/ext3 filesystems. `blkid` is a more generic tool that can be used to show labels on other filesystem types, including swap devices. The `mount` command, discussed later, can be used for mounting the device by referring to its label rather than the device path itself.

Adjusting or displaying filesystem parameters on ext2/ext3 filesystems is done with the `tune2fs` and `dumpe2fs` commands. These commands should be executed before doing any filesystem adiminastration, such as resizing, tuning or changing parameters.

Mounting Tools

- **Mounting filesystems**
  - `mount device mount_point`
  - *device* can be by device path, `LABEL=`, or `UUID=`
- **Unmounting**
  - `umount device|mount_point`
  - *device* can be by device path, `LABEL=`, or `UUID=`
  - `fuser mount_pont|device`

Mounting and unmounting filesystems is done with the `mount` and `umount` commands. A bit of trivia why `umount` and not `unmount`. The reason is historical, as commands on Unix could not be more than 6 characters. Of course, this was shortly lived, but many core utilities from Unix still exist on Linux today that had this limitation.

Mounting filesystems is referred to as grafting a filesystem into another. Because Linux/Unix uses a reverse tree hierarchy for its filesystem, this means that devices are just files within the filesystem. When I put a new filesystem on this device, I need a place on my already existing filesystem to get to the data. As such, we mount this device to an already existing directory. So, rather than C:, D:, etc residing separately from each other in Windows, we have one global filesystem /, with possible multiple nested, or grafted filesystems beneath.

If the device is already setup in your `/etc/fstab`, which we'll cover on the next slide, then you don't need to give both the `device` and `mount_point`. One or the other will work, and the `mount` command will read the `/etc/fstab` file, and find the appropriate entry, where to mount it and how.

If devices are defined in your `/etc/fstab`, and some devices aren't mounted, running `mount -a` will mount any filesystems in that file that currently are present but not mounted.

Lastly, some filesystems refuse to be unmounted, even as root. As such, the `fuser` utility is used to discover why the filesystem is in use. `man fuser` will show the keys and values to the syntax that might be displayed on the output.

The /etc/fstab

- *device*: The device path, LABEL or UUID
- *mount_point*: The path on the filesystem
- *fs_type*: The type of filesystem to mount
- *options*: The filesystem options comma-separated
- *dump_freq*: Level 0 dump frequency: 1=daily, 2=every other day, etc.
- *fsck_order*: Filesystem check order. 0=ignore, 1-9 valid

The `/etc/fstab` file is used to make mounting devices persistent across reboots. This file should only be used for non-removable media or removable media on headless boxes. GNOME and KDE have automount abilities where plugging in a USB drive, CD/DVDROM, etc will be auto-mounted to `/media`.

The syntax for the `/etc/fstab` file contains 6 columnns, and the description is fairly straight forward. A breakdown of the column descriptions is here:

- 1st: This is the special device file name. It can be referred to by the full device path as the kernel recognizes it, the LABEL= syntax, or UUID=.
- 2nd: This is the mount point on the filesystem where the device will be ounted and its data accessed.
- 3rd: This is the type of filesystem to mount. The kernel when mounting he filesystem must know what driver to use for the filesystem.
- 4th: These are the mount options that you want applied when mounted. The `defaults` option can be seen using `tune2fs -l device | grep ptions`.
- 5th: This is the dump frequency for backing up the filesystem. If using he dump command to backup your data, it will read the `/etc/fstab` file nd apply the dump frequency options here. 0 or 1 are the most common.
- 6th: This tells the kernel which order to do filesystem checks. enerally, the / filesystem should be checked first, then each additional ilesystem in order following. 3-9 are rarely, if ever used.

Common mount options are ro (read-only), rw (read-write), nosuid (suid or sgid file modes are not honored), dev (device files permitted), noexec (do not allow permission of executable binaries), async (file changes managed asynchronously), acl (Posix ACLs are honored), uid/gid= (all files of the mounted filesystem are owned by uid or gid), loop (mount the filesystem over a loopback device) and owner (the mount request and device must be owner by the same EUID).

Other Mounting

- `udev` is kernel level device creation and destruction
- Automounter with autofs
- Network mounting with NFS, SMB, SSHFS (fuse), etc.

This is just scraping the tip of the ice berg with mounting and filesystems. The kernel auto-creates devices based on discovery with help of the `udev` system. These devices are also removed when the kernel no longer needs them, or they've been removed from the system.

There is also the automounter which is useful for mounting filesystems on the fly just by entering a specific directory. The automounter will keep an eye on a directory that you specify in the `/etc/auto.*` files. Then, when you enter that watched directory, the device is automatically mounted. It remains mounted for a default of 5 minutes, then unmounts.

Of course, filesystem mounting isn't limited to local filesystems. Network filesystems can be mounted locally, and treated as local data. Such network filesystems include, but not necessarily limited to, NFS, Samba and SSH.

Software RAID

- **Advantages**
  - Cost
  - Speed
  - Reliability
- **Disadvantages**
  - Platform dependent
  - Speed

Linux software RAID has three main advantages and two main disadvantages. Of the three main advantages, cost is probably the biggest factor. Hardware RAID controller cards can be extremely pricey. Especially if you want a controller card that can meet the processing power and caching capabilities of your CPU and motherboard. Which bring about another advantage of Linux software RAID- speed. Due to the ever increasing bus speeds, the amount of cores packed into a CPU, and increasing RAM sizes for pennies on the dollar, finding a capable hardware controller card can be very expensive. Lastly, because Linux software RAID does not employ extra silicone, it is not a potential failure point. In fact, Linux software RAID is So reliable, that many hardware companies are using it for the fimware on their controller cards.

Linux software RAID also employs two major disadvantages. Because Linux software RAID is a Linux kernel module, this means that the RAID array is only visible to a Linux-based operating system. Unlike hardware RAID, which presents a disk to any operating system, regardless. Speed is also a major disadvantage of Linux software RAID. Wait, how can that be if it's an advantage? Well, if you have multiple arrays that will be doing some serious I/O, then you could completely consume your frontside bus, leaving no room for other applications to communicate with the CPU, or take advantage of RAM in a timely manner. As such, even if the cards aren't as specked out, having dedicated silicone does mean taking the stress of your motherboard, for other applications to move about freely.

Linux kernel RAID Levels

- Linear
- 0 or striping
- 1 or mirroring
- 5 or striping with distributed parity
- 6 or striping with dual distributed parity
- 10 or striping with mirroring (experimental)

Linear RAID is contiguous disk storage. An array is built from the first disk, and the size is calculated in an adjacent manner from the start of the first disk, to the end of the last. More disks can be added to the array, and data is written to the disks in a linear fashion. One disk is the minimum for this level, and it offers NO redundancy.

RAID 0 or striping requires a minimum of 2 disks. This RAID level was an after thought when system administrators wanted to increase the read and write speeds to their disk. So, to stripe the date across all the disks means each hard drive can write the same data in an asynchronous manner. Performance is optimal if drives of the same size are used.

RAID 1 or mirroring requires a minimum of 2 disks as well. However, unlike Linear and RAID 0, RAID 1 offers redundancy. When data is written to the array, every disk receives a copy. Read and write times should be comparable to reading and writing to a single local disk.

RAID 5 requires a minimum of 3 devices. Every disk except for one, and any spare disks you explicitly set, gets a striped version of the data, much like RAID 0. However, the last disk is used to write a parity. The parity is meta-information about the data that exists on the rest of the array's disks. In a 3 disk scenario, if

a disk were to go bad, the other two could rebuild the data using the data on one and the parity on the other. This RAID level gives a good balance of performance and reliability, and seems to be the default level in many a large corporation. The sweet spot for this level is 7 active disks + 1 parity, or 8 total disks.

RAID 6 is the same as RAID 5, except that rather than just having an parity disk, we have 2 parity disks. This does mean that we lose a full disk size in our array, at the cost of extra reliability. This level isn't as good a performer as level 5. The minimum amount of disks needed for this level is 4, and with that array, can suffer 2 disk failures, and still be operational, although degraded.

RAID 10 is marked experimental in the Linux kernel, and is not available by default unless you compile from scratch and enable it. This is the case for most distributions. This level can operate with a minimum of 3 disks, despite what many websites will tell you. However, it's best visualized with 4 disks. If 4 disks are in the array, then 2 disks are mirrored together and the other two disks are mirrored together. Then, those two mirrors are striped. This is known as a plaid array. This level gives much better performance over level 5 and 6 due to increased striping and no parity calculation. It even offers exceptional redundancy, allowing for multiple disk failure scenarios. As long as the mirrored set remains operational, this array can fail any series of disks. It's becoming the hotness in system administration, due to these reasons.

Software RAID - mdadm

- **Create an array**
  - `mdadm -C /dev/md0 -l 5 -n 5 -x 1 devices`
- **Format the array with a filesystem**
  - `mke2fs -j /dev/md0`
- **Information of the array**
  - `mdadm --detail /dev/md0`
  - `cat /proc/mdstat`

Creating RAID arrayis is done using eth `mdadm` utility. `mdadm` stands for Multiple Disk ADMinistration. Familiarizing yourself with the documentation on `mdadm` will be beneficial. Here are some of the common options:

- -C /dev/md0 - Creating device /dev/md0
- -n 5 - Five active decices will be in the array
- -l 6 - Uing RAID level 6 for the array
- -x 2 - Two hot spares will be available should one of the other 5 die
- -a part - Make a partitionable array

Now that your RAID array is being constructed, you can watch the status of the array with `mdadm --detail /dev/md0` or `cat /proc/mdstat`. You can watch the build of the arry, see the active devices, failed devices, hot spares, and other useful information regarding the array.

When the array is finished building, you'll need to put a filesystem down on your array with the filesystem utility of your choice. Then of course, at this point, you treat the array just like ayn other device in your system, by mounting it to a mount point, putting a label on the filesystem, and so forth.

Introducing LVM

- No partitions!
- Physical Volumes
- Volume Groups
- Logical Volumes
- Physical Extents
- Logical Extents

The first thing to note about LVM is that it provides flexibility to what is normally a chore with plain partitioning schemes. Rather than moving and resizing partitions, I can take the "moving" aspect away. There might be times when I need to do some resizing, but I never need to move a logical volume on an individual volume group.

First, physical volumes are created on a one-to-one ratio with your devices or partitions. This creates a low-level formatted filesystem on the device, which means that LVM cannot be applied to drives or partitions that already have data, as you will lose the data. On the physical volume, resides physical extents. Think of these the same way you think or blocks on a filesystem. If each extent is 1MB in size, and I have a 1GB device, then I can have a physical volume with 1024 extents.

After all the physical volumes are created, they are pooled together in any manner to creat a large volume group. Think of this volume group as a massive disk of storage, or a local SAN. Physical volumes can be added and removed from the volume group at anytime, without destroying data, provided you typed the commands correctly and in the proper order. If I had four 500GB drives, I could add these to a single volume group that would total 2TB in size, or 2 volume groups of 1TB in size, and so forth.

Now that my volume group is created, I can "slice" up the group, to create my individual data containers. It's best to think of LVM in terms of containers actually, as it will help you sort out the order of commands that you need to perform. One this container is created, I can put my filesystem on the container, mount it to a directory, and begin modifying and accessing the data. These logical volumes also have extents called "logical extents" similar to to the physical volumes. These exents behave the same as physical extents, but can be smaller, larger or the same size as the physical extents.

In relation to extents, it's important to understand that you don't know where you data physically resides on the harddrive. LVM is tracking this for you. This means that if you were to lose a disk, it could spell trouble for your data. While LVM does offer limited redundancy if built in that manner, it's best if you're using many disks, to use software or hardware RAID before applying LVM.

Notice, I never mentioned the word 'partition' anywhere.

Why LVM?

- Flexibility
- Low overhead
- Jenga anyone?

The number one reason any system administrator choosed LVM over flat partitions are due to the flexibility LVM provides. Creating, resizing, and moving filesystems are really quite trivial, and can usually be done with the root filesystem onilne, if you kept / small on install. LVM also comes with very minimal overhead, making it suitable for everyday use that isn't I/O intensive. Benchmarks put the overhead of LVM about 1% slower in sheer speed than filesystems, and about 5% sloewer than raw devices.

However, with the extensive flexibility that LVM provides, it doesn't ship with extensive redundancy capabilities, and most system administrators use external RAID with LVM, rather than the redundancy that LVM provides. As such, if redundancy is not being used, then if a drive goes south, the entire LVM structure collapses. Very similar to the way the wooden tower game Jenga behaves. Pull the wrong block, and the tower falls.

Practical LVM

- `pvcreate devices`
- `vgcreate VGname devices`
- `lvcreate VGname`

First, there are only 3 letter pairs that you need to remember: pv, vg and lv. Using tab completion in your shell, you can see the rest of the commands that follow. Such as `pvdisplay` or `lvremove` and so forth.

When building an LVM structure, the order is first creating each physical device, then adding the physical devices do a volume group, then finally chopping up the group creating individual logical volumes. You can either use up the entire volume group space with logical volumes, or you can leave some unused

space for later, assigning just what you need now.

`pvcreate` is run first to create the physical volumes. `pvcreate` is a low level formatter, and will destroy an existing filesystem on the device if it already exists.

`vgcreate` is used to create a pool of disk space with the physical volumes called a volume group. You must give the volume group a name. I tend to shy away from "VG00" or similar silly names, and prefer to give it a more meaningful name that represents the data on the physical volumes. Something like "personal" and "business" or using the hostname are more appropriate.

`lvcreate` is finally used to create logical volumes from the volume group. It makes sense to also name the logical volume, although this is not required like it is for the volume group. Some useful switches are `-l LogicalExtentsNumber[%{VG|PVS|FREE}]` or `-L LogicalVolumeSize[kKmMgGtT]` for sizing the volumes and -s for creating an LVM snapshot (covered later).

RAID / LVM Example

- `mdadm -C /dev/md0 -a yes -n 5 -l 6 -x 2 /dev/sd{a..g}`
- `pvcreate /dev/md0`
- `vgcreate personal /dev/md0`
- `lvcreate -L 50G -n music personal`
- `lvcreate -L 100G -n videos personal`
- `mke2fs -j /dev/personal/music`
- `mke2fs -j /dev/personal/videos`
- `mount /dev/personal/music /media/music`
- `mount /dev/personal/videos /media/videos`

Because of the limited redundancy support that LVM offers, it is not uncommon for system administrators to use RAID separate from LVM. Because LVM offers extreme flexibility with resizing your volumes, but not the needed redundancy, if you lose a disk, you could lose your data, wherease putting RAID beneath LVM could lessen that chance greatly.

The above is an example of creating a RAID-6 array with 5 active drives and 2 hot spares. Once the array is created, the array is turned into a physical volume and added to a volume group. Supposing the volume group is 1TB in size, we have then created two mount points from the group. We called the group 'personal' and two logical volumes called 'music' and 'videos'. We created an ext3 filesystem on the devices then mounted them to their respective locations, so we could access the data.

Resizing LVMs

- LVM is a container- a bucket
- The filesystem resides inside - water
- Reduce Order: 1) Filesystem, 2) Logical Volume
- Extend Order: 1) Logical Volume 2) Filesystem
- `pvmove device`
- `vgextend VGname device`
- `vgreduce VGname device`
- `lvextend -l|-L size device`
- `lvreduce -l|-L size device`
- `resize2fs device [size]`

As mentioned extensively through out this presentation, LVM make is trivial to resize volumes. So, it would be draconian to not introduce a slide on it. When resizing logical volumes, you need to think in

terms containment. Think of the logical volume itself as a data contained, and the filesystem resides inside the container, much like water in a bucket. When you want to make the volume bigger, you would need to extend the size of the container (bucket) before adding more filesystem (water). If you wanted to reduce the volume, then you would need to displace some filesystem before making the container smaller.

Extending volumes can be done while the volume is currently mounted if it's an ext-based filesystem. Other filesystems might require you to unmount them. The order for extending an ext-based volume is as follows:

- `lvextend -L size[kKmMgGtT|%|FREE] device`
- `resize2fs device`

Reducing volumes can not be done while the volume is currently mountes as data might reside towards the end of the volume, and reducing it would mean truncating the data. As such, we need to make sure all data resides at the front of the filesystem first. In order to do this, the filesystem must be unmounted. Then we can reduce our filesytme to the appropriate size, the shrink the container:

- `umount device|mount_point`
- `e2fsck -f device`
- `resize2fs devise size[kKmMgG]`
- `lvreduce -L size[kKmMgGtT] device`
- `mount device mount_point`

Not only can you resize the logical volumes, but resizing volume groups is also possible. This is done by adding or removing physical volumes. Adding physical volumes is trivial, but care must be taken to removing physical volumes. LVM is keeping track of where you data is stored, and it's not always in a contiguous manner. Which means that your data could reside on the disk that you want to remove. So, before removing this disk, you need to make sure that there is enough space residing on the other physical volumes to store your data. If not, you'll need to add more physical volumes, or delete some data. At any event, here is the path you must take when removing a physical volume from the volume group:

- `pvchange device`
- `vgreduce VGname device`

Notice that you didn't need to worry about filesystem checks, or unmounting any filesystems. You are operating on devices at a much more lower level.

Extending the volume is just as easy. All you need to do is turn your device into a physical volume, then add the device to the volume group. This would be handled in this manner:

- `pvcreate device`
- `vgextend VGname device`

Now you should have more storage in your volume group to create additional logical volumes, or extend existing volumes with.

Advanced LVM

- LVM RAID level 0 & 1
- Snapshots with `lvcreate -s`
- LVM with GFS2, OCFS2 and Lustre

Lastly, without going into extensive detail, LVM does have some advance feature sets, such as minimal striping and mirroring redundancy, snapshot capability and clustered filesystems.

In regards to striping and mirroring, LVM can take advantage of RAID level 0 and 1. As we already know by know, level 0 offers no redundancy, but instead offers speed through asynchronouly writing the data in stripes. This can provide enhanced performance to LVM. With level 1 RAID, you can take adantage of mirroring, providing highly redundant data.

LVM snapshots work much the way a photograph does. If I were to stand at the freeway, and take a picture of the cars on the road, I would have a representation of the data at that specific point in time, even though the data is changing constantly. LVM snapshots behave much the same way. First, I create a new logical volume, which really cany be any size. I pass the `-s|--snapshot` switch to the `lvcreate` command to tell LVM that this volume is a snapshot of an already existing volume. At that point in time, LVM has a representation of what the data looks like through the use of pointers no inodes and physical extents mapped through logical extents. This makes it great to perform backups, even if the data is changing, because I don't care about any new data at all, only the data that existed exactly at that point in time. However, if any data that belongs to the snapshot were to be removed, then I would need adequate space on my new logical volume to hold the removed data, asd I might need it for the backup. Once the backup has been performed, the volume can be removed safely, without any concern for the data on the other volumes.

LVM can also take advantage of clustered filesystems, such as the Global FileSystem, GFS or the Oracle Clustered FileSystem OCFS2. This means that LVM can not only span multiple disks on a single hosts, but multiple disks over multiple hosts, creating a SAN-like implementation.

Fin

- Comments, questions, rude remarks?