

Storage



More than just “disks”

- ▶ Long ago, storage used to refer to one or more disks attached to the machine and used to store the operating system and user files
- ▶ The Hard Disk (or, recently, the SSD) is still the dominant part of storage but a lot of layers were built on top of it like:
 - ▶ Different filesystems (ext2, ext3, ext4, exFAT, ZFS...etc.)
 - ▶ Partitioning
 - ▶ RAID
 - ▶ Logical volume managers
 - ▶ Network file systems (NFS, SAMBA...etc.)
- ▶ Storage has evolved from being just a number of disks attached to the machine to a whole compartment of disks, with separate power, and processor, connected to the machines through lightning fast fiber channel cables, or several Gigabit Ethernet cables (iSCSI) called Storage Area Networks (SAN)

But what is a disk?

- It is made up of several rotating platters coated with magnetic film.
- Reading and writing is done through sliding heads that move back and forth just above the surface of the platter but never touch it.
- When a read operation is requested, the head must move the position above the appropriate track. The time consumed by this operation is called *seek delay*.
- It then has to wait for the desired sector to pass under the head as the platter rotates. The time spent in this task is called *rotational latency*.
- A number of tracks that are at the same distance from the spindle are called a *cylinder*. Because the arm does not need to move to read the data, it is read much faster.





Hard Disk Drives



- An HDD refers to the mechanical diagram depicted in the previous slide
- The rotational speed is measured in RPM (Revolutions Per minute). You can see speeds like 7200 RPM, 10000 RPM, or 15000 RPM depending on the type of storage used.
- Hard disk drives suffers frequent failures. That's why technologies like RAID were introduced to help avoid data loss in the event of an HDD failure.
- A failure may occur as a result of bad blocks on platter surface, or the damage in the mechanical components like the spindle or the heads.
- Disk life expectancy is measured in MTBF (Mean Time Between Failures). However, this should not be used to expect failure rate over a long term, as the manufacturer selects the most reliable phase of a drive

Solid State Drives

- ▶ They use flash memory cells to read and write data in parallel.
- ▶ They provide more bandwidth than mechanical HDD's specially in random access
- ▶ They are considerably more expensive than traditional hard disks.
- ▶ SSD pages have a limited number of rewrites: about 100,000 times. The firmware distributes writes across all drive's pages transparently from the OS.



Parallel Advanced Technology Attachment (PATA)

- ▶ ATA used to be known as Integrated Drive Electronics (IDE) because it placed the controller in the same place as the platters
- ▶ They were connected to the motherboard using ribbon cables
- ▶ This is considered now obsolete. Those disks relatively fast, had large capacity and very cheap
- ▶ Two types of disks used this interface: PATA (parallel ATA shown below), and SATA (serial ATA)



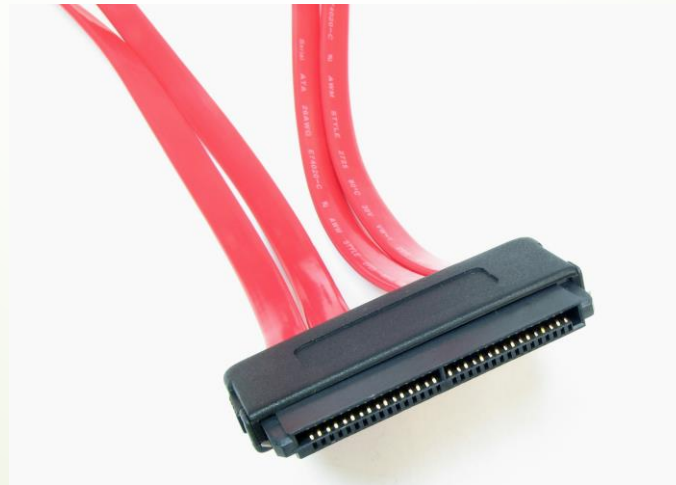
Serial Advanced Technology Interface (SATA)

- It is the successor of PATA
- It used different cabling that supports longer length
- It supported higher transfer rates
- Hot swapping is natively supported, which makes this technology applicable in server environment.



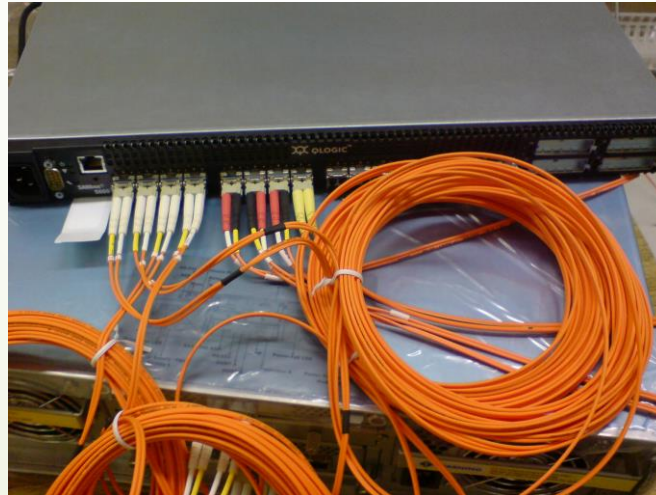
Small Computer System Interface (SCSI)

- It is a parallel interface used to attach peripheral devices to computers
- It can be thought of as an I/O bus rather than an interface because multiple devices could be attached to a single SCSI port
- Serial Attached SCSI (SAS) is a successor to traditional SCSI. They offer performance improvement and faster data transfer rates.



Fiber Channel (FC)

- It is a serial interface that uses fiber optic cables for data transfer
- They provide much higher speeds than other interfaces
- They are mostly used in enterprise environments





Linux disk devices

- When you attach a disk to the system, the kernel automatically discovers it and creates the appropriate device file.
- Those files can be found in /dev directory. SCSI devices are preceded by letter s, so /dev/sda refers to the first SCSI disk.
- Partitions are mentioned in numerical order after the device name. So /dev/sda1 refers to the first partition of the first SCSI disk
- The directory /dev/disk provides useful information about the currently attached disk devices. You can list them by path, id, UUID, and label.
- You can also use fdisk -l and parted -l to display attached-disks information
- The filesystem internally uses unique ID strings to differentiate different disk devices. This is hidden from the user.

The basics – adding a disk

- These are the basic steps of adding a single disk to a Linux-run machine, with no RAID, all the disk space will be in a single partition:
 - Attach the disk physically, and reboot the machine
 - Run `fdisk -l` to see the attached disks
 - If the disk is less than 2TB, you can use MBR (Master Boot Record) partition table
 - If it is larger than 2TB you must apply GPT (GUID Partition Table) using `parted` or `gparted` (in GNOME desktop environment)
 - Assuming that the disk is less than 2TB, partition it using `fdisk`. Do not select a partition type.
 - Use `pvcreate` to create a physical volume to be used in a volume manager
`pvcreate /dev/sdx`
 - Use `vgcreate` to create the volume manager accepting the physical volume
`vgcreate vgname /dev/sdx`
 - Use `lvcreate` to create a logical volume that has all the space of the physical volume like this `lvcreate -l 100%FREE -n lvname vgname`
 - Finally, format the `lv` using `mkfs`: `mkfs -t ext4 /dev/vgname/lvname`
 - Now the disk is ready. Create a mount point for it in the `/etc/fstab` file



Why partition a disk?

- Partitioning refers to *dividing* a disk or a number of disks into separate, distinct *pieces* of known sizes. It's like creating new disks from one or more disks.
- It helps preserve disk space by allocating only the needed size to the filesystem
- It is advised to put system directories like /var, /usr, and /tmp on their own filesystems to protect the root filesystem from running out of space
- /home is also advised to be put on a separate partition to preserve user's data in the event of having to reinstall the system.
- SWAP space can be split among several disks (through volume management) for increased performance, as this will distribute the “busy” load on multiple disks/partitions.
- Partitioning makes backing up data easier; as you can use the dump command to backup the entire device to media or another disk



The partitioning process – MBR based

- A label is written at the start of the disk to indicate the number of blocks used in each partition
- This label exists side-by-side with other startup labels such as the boot block, in addition to other information like the disk ID. Sometimes called MBR (Master Boot Record)
- The MBR occupies the first 512 bytes of the disk. Because the boot data occupies most of those bytes, the remaining ones are only enough for defining four *primary* partitions. They are called primary because they are defined in MBR.
- To add more partitions than four, you have to label one of the primary ones as *extended*. This means that it has its own table for defining extra partitions. Those are called *secondary partitions*. There cannot be more than one extended partition on a disk. It is recommended that the extended partition is the last one in the disk.
- When the partition is marked *active*, the boot loader selects it to find the boot loader program
- This process is platform agnostic. That is, it does not matter where you partition the disk (Linux, Windows, or UNIX) as long as you use MBR.



The partitioning process – GPT based

- MBR falls short of supporting disks larger than 2 TB. Accordingly, EFI partitioning scheme was introduced. It's also called GUID partition table or GPT.
- It defines only one type of partition but multiple partitions can be created
- The partition is defined by a 16 byte representing a unique ID (GUID, hence the name)
- GPT places an MBR block at the start of the disk for backward compatibility, however it is not of real use.
- Windows (starting from Vista) can use GPT disks, but the system must use EFI firmware to boot from one. On the other hand. Linux GRUB understands GPT on any system



Redundant Array of Inexpensive Disks (RAID)

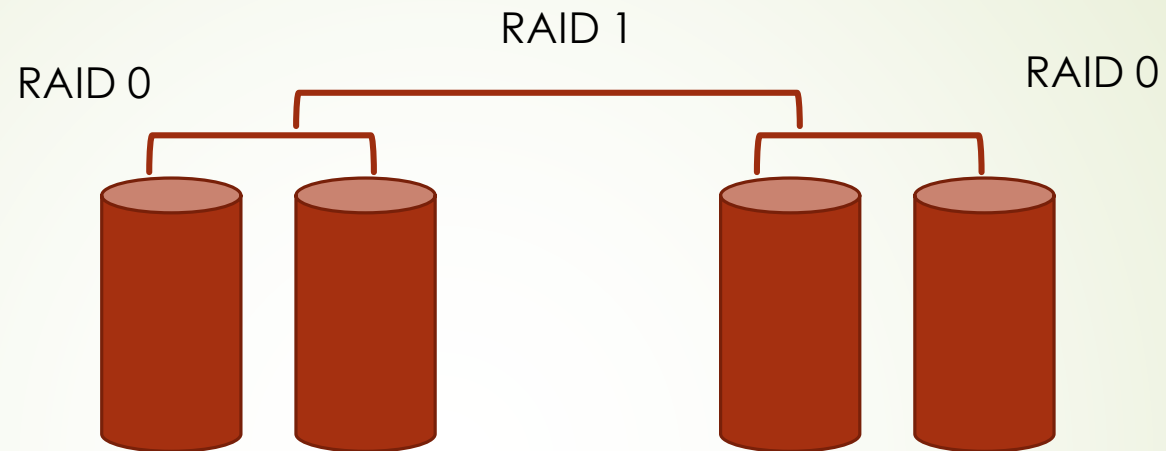
- Sometimes “Inexpensive” is replaced with “Independent”, both are correct.
- It refers to a system that clones data among multiple disks. This helps mitigate the risk of data loss out of a disk failure. It also reduces the downtime required to replace a failing disk (often no downtime at all).
- The operating systems “sees” the disks as one unit (disk) rather than multiple. The RAID system handles data distribution and management.
- Hardware RAID used to be thought of as a better alternative to software (OS-based) RAID because of the lack of RAID support in operating systems, in addition to some performance improvements gained by the hardware write buffering into memory.
- Nowadays Linux fully supports RAID, and it’s use is encouraged over hardware ones, specially that most RAID cards lack the buffer memory.



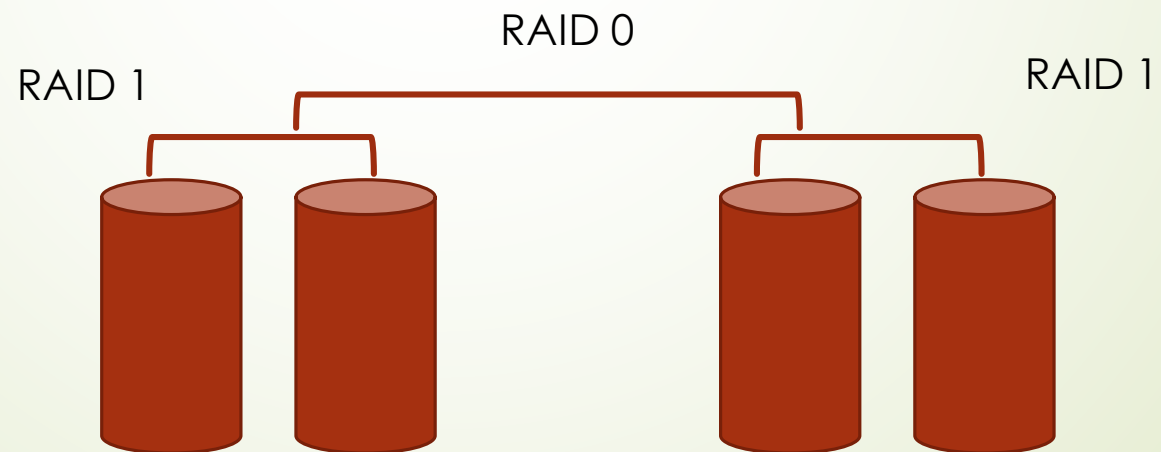
RAID types (levels)

- RAID levels provide two advantages: performance gain, by distributing data among multiple disks at the same time, and protection by replicating the same data on a number of disk devices. However, both features are not provided at the same magnitude; they vary from one RAID level to another.
- RAID 0 (striping): more geared towards performance. It stripes data among two or more disks at the same time. This decreases read and write access times. The only protection it provides is increasing the lifetime expectancy of the disk; as the load is shared. 100% of disk space is available.
- RAID 1 (mirroring): data is replicated on two or more disks. This provides more protection than level 0, faster reads, yet slower writes. It requires 50% of the disks to be functioning in order to operate. 50% of disk space is available.
- RAID 0 and 1 may be combined to provide mirroring across stripes or stripes of mirrors. This attempts to gain performance and protection at the same time.

RAID 1 + 0 (Mirroring the stripes)

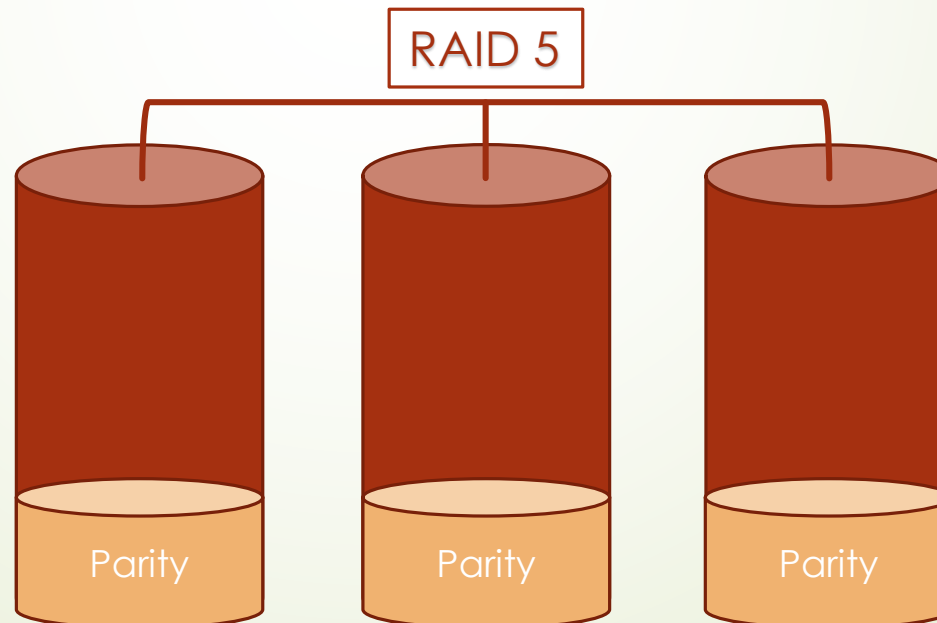


RAID 0 + 1 (Striping the mirrors)



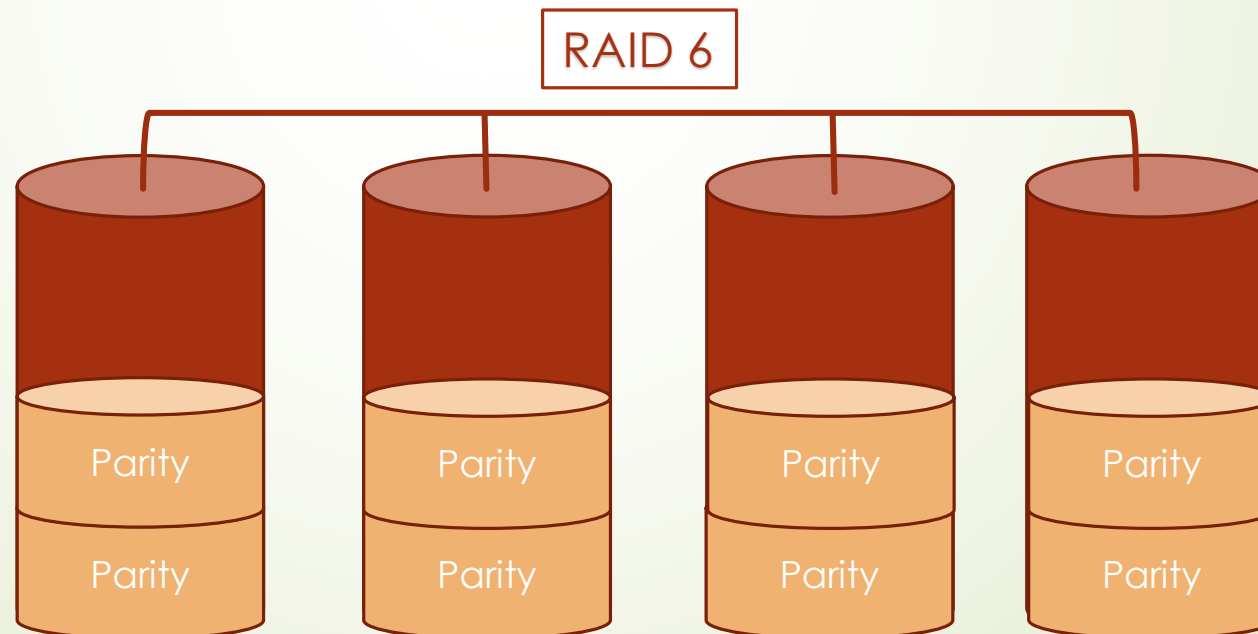
RAID 5


- It uses striping with parity. Parity is the result of a mathematical XOR operation. It is distributed among all disks. If one of them fails, the remaining data on other disks, combined with parity, can be used to reconstruct the missing data. RAID 5 requires at least three disks to operate. Two of the three disks are used for storage. Accordingly, at least 67% of the disk space is available.



RAID 6

- It is the same as RAID 5, but it uses double parity on each disk. It can sustain the loss of two disks concurrently, but it needs an additional disk for the extra parity. RAID 6 may cost you a write slowness penalty as two parity pieces will have to be calculated for each block of data written.
- RAID 6 gives you *at least* 50% of the available disk space. This amount will increase as more and more disks are added. For example, if you use 10 disks you will have 80% available.





What happens when a disk fails in a RAID array?

- As illustrated, RAID 0 does not provide any protection against corrupted disks. If a disk fails, data should be restored from the most recent backup.
- In RAID 5, only one failed disk is tolerated, two in RAID 6. Once that happen, the failed disk(s) must be replaced as soon as possible to protect data from another disk failure.
- After replacement, the RAID array starts writing parity data (or mirrored data in case of RAID 1) to the new disk. This is a lengthy operation in which performance is generally degraded.
- As a precaution, some systems provide *hot spares*, which are “standby” disks ready to automatically replace any failed disks in the array, after which synchronization immediately starts.



RAID shortcomings

- It does not protect against non-disk related data loss like filesystem corruptions, accidental file deletions, or data theft to name a few.
- Having to write parity information (in RAID 5 and 6) for each block of data will slow write operations.
- If power surge occurs during the write operation, it would be impossible to know whether or not the blocks have been mirrored (RAID 1) or the parity information have been written (RAID 5 and 6). In such a case, parity information will be inconsistent with the blocks of data. To make things worse, this will not be discoverable unless a disk actually fails and parity information is consulted. In RAID 5 and 6, this is often referred to as a “write hole”.

LAB: create a RAID5 array on Red Hat Linux

- Add three disks to the machine
- [Optional] Use `fdisk` to create a single whole-disk partition on each disk
- Assuming that the disks are `/dev/sda`, `/dev/sdb`, and `/dev/sdc`, use `mdadm` command to create the array:

```
mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sdb1 /dev/sdc1 /dev/sdd1
```
- Create a new filesystem on the device array:

```
mkfs -t ext4 /dev/md0
```
- Mount the filesystem

```
mount /dev/md0 /mnt
```
- Observe the size of the file system and notice that is $N - 1$ the combined size of the three disks.
- You can view information about the current array configuration by reading the `/proc/mdstat` file

LAB: making the array persist reboots

- The `mdadm` command automatically activates the array upon creation. But the array has to be manually activated upon reboot (although Debian systems activate is automatically on startup)
- To get activated on startup, the array must have a configuration file, which has to be created manually. Use the following command to do this:
`mdadm --detail --scan >> /etc/mdadm.conf`
On Debian this file is under `/etc/mdadm/mdadm.conf`
- Now you can start the array using the command
`mdadm -As /dev/md0`
- You can stop the array using the following command:
`mdadm -S /dev/md0`

LAB: monitoring the array

- Md can monitor the array the whole time, and send notifications to the administrator when a failure occurs
- To enable this feature add the following line to mdadm.conf:
MAILADDR root@company.com
- Then enable the monitoring daemon to start when the system starts up
sudo update-rc.d mdadm enable [Ubuntu]
chkconfig mdmonitor on [Red Hat]
- You can test this by *faking* a failed disk in the array:
mdadm /dev/md0 -f /dev/sdb1
which will make md regard /dev/sdb1 disk as faulty
- Observe /var/log/messages and /proc/mdstat to note the notifications that warn about the failing disk
- You can completely remove the disk from the array using -r switch
mdadm /dev/md0 -r /dev/sdb1
- Add a new disk (either with or without rebooting the machine depending on whether or not it is hot swappable), create a partition on it and add it to the array using the -a switch:
mdadm /dev/md0 -a /dev/sdb1



Logical Volume Management (LVM)

- It offers great deal of flexibility over traditional partitioning when dealing with disks. For example:
 - You can grow and shrink volumes easily without having to reboot the machine
 - Data can be moved (migrated) from one physical volume to another
 - Logical volume snapshots give you a “copy-on-write” solution
 - Striping and mirroring can be implemented to offer data protection and data redundancy
- A volume manager is container of one or more physical volumes
- A physical volume itself can represent a physical disk or a disk partition

LAB: create a volume group

- ▶ The first step is to create the physical volume, we are using our disks

```
pvccreate /dev/sdb1
pvcreate /dev/sdc1
pvcreate /dev/sdd1
```
- ▶ Then we create the volume group by using the `vgcreate` command

```
vgcreate vgArchive /dev/sdb1 /dev/sdc1 /dev/sdd1
```
- ▶ The `vgcreate` usually take a number of physical volumes
- ▶ To create a logical volume, use the `lvcreate` command with the desired lv name and size:

```
lvcreate -n lun1 -L 100M ARCHIVE
```
- ▶ Create a filesystem on the lv to make it usable:

```
mkfs -t ext4 /dev/vgArchive/lun1
```
- ▶ Create a mount point and mount the logical volume onto it:

```
mkdir /lun1
mount /dev/vgArchive/lun1 /lun1
```



LVM snapshots

- They can be thought of as a pointer to the exact state of the logical volume at one point in time.
- You can create a snapshot of a logical volume, do whatever changes to it and then either *merge* those changes with the original lv or to discard those changes and restore the lv to it's original state when the snapshot was taken
- It becomes very useful when, for example, you want to take backup of a database that needs to be shutdown before it can be backed up. Making a snapshot of its lv can enable the DB to be opened again while the its *shutdown* snapshot is being backed up
- The only shortcoming of snapshots is that you must avail space on the volume group that is large enough to accommodate any increase in the amount of data used by the lv, for which the snapshot is taken. Once the volume group runs out of space, the snapshot is completely corrupted.

LAB: create a snapshot of the logical volume

- First ensure that the volume group has enough space for the snapshot. As a precaution, we'll add a new disk to the volume group:

```
pvcreate /dev/sde1  
vgextend vgArchive /dev/sde1
```
- Unmount the logical volume

```
umount /lun1
```
- Create the snapshot for logical volume lun1

```
lvcreate -L 100M -s -n lun1-snap vgArchive/lun1
```
- Mount the snapshot to the original mount point of the lv

```
mount /dev/vgArchive/lun1 /lun1
```
- Now you can do any backups necessary to the original logical volume, while users can continue working on the snapshot
- It's advised to periodically check the snapshot status using `lvdisplay`, especially for the free space. If the snapshot reports "inactive" this means that it has run out of space and should be discarded.
- When you no longer need the snapshot, you can either discard it

```
umount /lun1  
lvremove /dev/vgArchive/lun1-snap
```


or restore it (merge the changes with the original logical volume)

```
lvconvert --merge /dev/vgArchive/lun1-snap
```



Logical volume resizing

- One of the most strong reasons for using volume groups instead of traditional partitioning.
- Logical volumes can be extended *online*. That is, without needing to reboot the system or even unmount them. Shrinking, however, requires the logical volume to be unmounted
- For this reason, you have to shut down the system, boot from the CD/DVD, and enter rescue mode to be able shrink the root filesystem.
- Resizing is not complete until the `resize2fs` command is used:
 - If you are extending the filesystem, `resize2fs` is used *after* `lvextend`
 - If you are reducing the filesystem, `resize2fs` is used before `lvreduce`
- Be cautious here: using `lvreduce` before `resize2fs` when shrinking a filesystem may corrupt the data.

LAB: extend and shrink the logical volume

- In order to extend the logical volume, you have to ensure first that the volume group contains enough space:
`vgdisplay vgArchive` will show available space in the volume group
`vgs vgArchive` prints a more concise output
- Extend `lun1` by 100MB
`lvextend -L +100m /dev/vgArchive/lun1`
- Use `resize2fs` to apply the change
`resize2fs /dev/vgArchive/lun1`
- Check the new size using `df -h`
- Now shrink the same filesystem by 100MB. Use `resize2fs` first to apply the change (it will prompt you to run `e2fsck`):
`resize2fs /dev/vgArchive/lun1 1G`
`e2fsck -f /dev/vgArchive/lun1`
- Apply the change on the logical volume using `lvreduce`
`lvreduce -L 1G /dev/vgArchive/lun1`
- Check the new size with `df -h`

Filesystem journaling

- A journal is an area of the disk designated by the operating system. It is treated as a regular file.
- When an operation is made on the filesystem, the change is written to the journal first before being applied to the filesystem itself.
- If a power surge or an unclean shutdown happens while the filesystem is being written to, the information in the journal is used to reconstruct any missing data.
- Journaling is supported on Linux filesystems starting at the ext3 format.
- Luckily, journaling can also be applied to ext2 filesystems using the command `tune2fs`:
`tune2fs -j /dev/sdf1`
But the you'll have to change the corresponding filesystem type in `/etc/fstab` file



Important filesystem vocabulary

- Inodes: pointers to filesystem objects (files and directories). They store attributes and disk block locations of the object's data. You can view the inode of an object using `ls -li`. Sometimes you may need to check for the available inodes on a given filesystem using `df -li`.
- Superblock: a record containing the attributes of the filesystem like the size, the block size, the size and location of the inode table, the disk block map and usage. Because of its importance to the integrity of the filesystem, several copies of it are created when you create the filesystem, they are scattered across different locations.
- Cache: all filesystem blocks can be cached into memory for faster operations. Periodically, the operating system syncs the cached data to the physical disk.

The /etc/fstab file

- Short for filesystem table, it is used to store the mount points of filesystems so that they are available when the system boots.
- The file contains six fields:
 - The device name: this can be a local filesystem, or an NFS or SAMBA share. It can also contain the UUID of the filesystem instead of the physical path to avoid any problems caused by the filesystem device name changes
 - The mount point
 - The filesystem type (ext4, nfs, cifs, iso9660...etc)
 - Mount options: some of the most common are
 - noauto: the filesystem must be mounted explicitly using the mount command
 - noexec: do not allow files to be executed from the mounted filesystem (useful in /tmp)
 - nodev: do not interpret a character or block device. That is, deny mounting an CD or an ISO file
 - Dump: whether or not the backup utility (dump) should backup this filesystem
 - Pass: when fsck is going to check the file on system boot:
 - 0 do not check
 - 1 check it first
 - 2 check it next



SCSI over IP (iSCSI)

- A technology that provides cheap SAN network
- It uses normal Ethernet cables for connecting clients (initiators) to disks hosted on servers or dedicated appliances (targets)
- The client is authenticated before being allowed to access the disks on the target through CHAP protocol
- Targets and initiators have special names called iSCSI Qualified Names (IQNs). For example `iqn.yyyy.mm.domain_in_reverse:any_name`.
- The year and month parts are used to indicate the domain owner, as domains may move from one holder to another
- iSCSI works on port 3260 by default

LAB: create iSCSI target

- The required packages for the initiator can be installed using
`yum -y install iscsi-initiator-utils`
- The required packages for target can be installed using:
`yum -y install scsi-target-utils`
- On the target machine:
 - Edit the file `/etc/tgt/targets.conf` and add an IQN entry with a backing store. For example:
`<target iqn.2015-12.com.example:server.lun1>`
`backing-store /dev/vgArchive/lun1`
 - Reload the configuration
`/etc/init.d/tgtd reload`
 - Verify the configuration
`tgtadm --mode target --op show`

LAB: create the initiator and discover the target

- On the initiator machine:
 - Configure the initiator name by editing the file `/etc/iscsi/initiatorname.iscsi`
 - Edit the `/etc/iscsi/iscsid.conf` to add authentication information if needed
 - Start the initiator daemon
`/etc/init.d/iscsid start`
`chkconfig iscsid on`
 - Discover the targets on the target machine
`iscsiadm --mode discovery -t sendtargets --portal machine_ip`
 - Login to the machine
`iscsiadm --mode node --targetname target_iqn --portal ip_address --login`
 - Ensure that the target disks are available using `fdisk`