8.1 RAID on Linux

RAID Level	Description
RAID 0 *(striping)*	A stripe set breaks data into units and stores the units across a series of disks by reading and writing to all disks simultaneously. Striping: • Provides an increase in performance. • Does not provide fault tolerance. A failure of one disk in the set means all data is lost. • Requires a minimum of two disks. • Has no overhead because all disk space is available for storing data.
RAID 1 *(mirroring)*	A mirrored volume stores data to two (or more) duplicate disks simultaneously. If one disk fails, data is present on another disk. The system switches immediately from the failed disk to a functioning disk. Mirroring: • Provides fault tolerance for a single disk failure. • Does not increase performance. • Requires a minimum of two disks. • Has overhead. Overhead is 1 / n where n is the number of disks. If data is written twice, half of the disk space is used to store the second copy of the data. • RAID 1 is the most expensive fault tolerant system.
RAID 5 *(striping with distributed parity)*	A RAID 5 volume combines disk striping across multiple disks with parity for data redundancy. Parity information is stored on each disk. If a single disk fails, its data can be recovered using the parity information stored on the remaining disks. Striping with distributed parity: • Provides fault tolerance for a single disk failure. • Provides an increase in performance for read operations. Write operations are slower with RAID 5 than with other RAID configurations because of the time required to compute and write the parity information. • Requires a minimum of three disks. • Has an overhead of one disk in the set for parity information (1 / n). ◦ A set with 3 disks has 33% overhead. ◦ A set with 4 disks has 25% overhead. ◦ A set with 5 disks has 20% overhead.
RAID 10 *(stripe of mirrors)*	A RAID 10 volume stripes data across mirrored pairs and across multiple disks for data redundancy. If a single disk fails, its data can be recovered using the mirrored information stored on the remaining disks. If two disks in the same mirrored pair fail, all data will be lost because there is no redundancy in the striped sets. RAID 10: • Provides fault tolerance for a single disk failure. • Provides redundancy and performance. • Uses 50% of the total raw capacity of the drives due to mirroring. • Requires a minimum of four disks.

8.2 Partition Types

Type	Description
Primary	A primary partition is used to store data as well as the operating system. Primary partitions: • Can hold operating system boot files. • Cannot be further subdivided into logical drives. • Can be formatted with a file system. There can be a maximum of four primary partitions or three primary partitions and one extended partition on a single hard disk drive.
Extended	An extended partition is an optional partition that contains logical partitions. Because an operating system can't be booted from a logical partition from within an extended partition, this partition type is not bootable. Extended partitions: • Can be further subdivided into an unlimited number of logical partitions. • Cannot be directly formatted with a file system. However, logical partitions within an extended partition can be formatted with a file system. • Only one extended partition can exist on a single hard disk drive.

8.2 MBR Partition Tools

Tool	Description
fdisk	The fdisk utility is used to manage partitions on a hard disk. The fdisk utility has the following characteristics: • When you create a partition, fdisk requests a beginning/ending sector or size. ◦ The size is indicated using K (kilobytes), M (megabytes), G (gigabytes), or T (terabytes). • When creating a partition, you specify the partition type using a hexadecimal code. Common hexadecimal codes include: ◦ 0x82 (Linux swap) ◦ 0x83 (Linux partition) ◦ 0x85 (Linux extended partition) ◦ 0x8e (Linux LVM partition) • Using the -l option displays the current partition configuration on the system. Type **fdisk *[device_name]*** at the command prompt to enter the fdisk utility. Within the fdisk utility, you can run the following options: • **l** lists the partition types supported. • **m** displays the help screen. • **n** creates a new partition. • **p** displays the partition table for that device. The /proc/partitions file contains a table with major and minor number of partitioned devices, their number of blocks, and the device name in /dev. • **q** exits fdisk without saving changes. • **w** writes the partition table to disk (saving the file) and exits the fdisk utility. • **d** deletes a partition.
partprobe	The partprobe command makes a request to the operating system to re-read the partition table. The operating system kernel reads the partition table and recognizes the table changes.

8.2 Storage Device Types

Storage Device Type	Description
Hard disk drive (HDD)	A hard disk drive identifies where data can be stored on its platters using several parameters that are collectively called the drive's geometry. The following parameters are used by the storage device interface to determine how the drive is accessed and where data can be stored: • **Heads** specifies the number of read/write heads on the drive. • **Cylinders** specifies the # of concentric parallel tracks on all sides of all platters in the hard disk drive. • **Sectors Per Track** specifies the number of wedge-shaped areas the platters have been divided into. Hard disk drives are connected to the system motherboard using a storage interface. The interface is commonly integrated within the motherboard itself. However, the interface may also be implemented using an expansion card that's installed in an expansion slot. In a modern desktop computer system, the following storage interfaces may be used: • **Serial ATA (SATA)** • **Small Computer System Interface (SCSI)** • ~~Parallel ATA (PATA)~~ *(This interface is obsolete)* Hard disks provide several **advantages**, including the following: • They can store a large amount of data. • They provide reasonably fast access speeds. • The store data at a relatively low cost per megabyte. Hard disks also have several **disadvantages**, including the following: • Hard disks wear out over time because they're mechanical devices that contain moving parts. • Hard disks are vulnerable to physical damage. For example, dropping a hard drive while it's spinning can cause the read/write heads to dig into the platter, destroying any data stored there.
Solid state drive (SSD)	A solid-state drive is a storage device that functions much like a hard disk drive, using the same block-based I/O operations. However, instead of aluminum platters, SSDs use flash memory to store data. SSDs typically provide storage capacity comparable to that of a small hard drive. They're beginning to replace standard hard disk drives in computer systems. Some of the **advantages** of SSDs include that they: • Are much faster than hard drives. • Have no moving parts, so they last longer. • Have lower power consumption than hard drives. • Are less susceptible to physical damage. • Are smaller and lighter than hard drives. • Use the same SATA interface found in standard hard disk drives. The main **disadvantage** currently for solid-state drives is cost. They are several times more expensive than comparable hard drives.
External flash storage device	Like an SSD, external flash storage devices store information using programmable, non-volatile flash memory. External flash storage devices most commonly connect to the computer using a USB interface. Advantages of flash devices include: • Portability • Larger storage capacity than optical discs • Relatively fast read access Some of the disadvantages of flash devices are: • Less storage capacity than hard disks • Relatively slow write speeds Common external flash storage devices include: • CompactFlash cards • eMMC cards • SD cards • SSD cards • MiniSD cards • MicroSD cards • xD cards • Hybrid cards (combines SSD and HDD technology) • Memory sticks

8.2 Linux Storage Device Files

Device File	Description
/dev/sdxn	**sd files identify hard drives**. A letter (beginning with a) follows the sd designation and identifies the hard drive's ID. At the end is appended a number (beginning with 1) that identifies the partition on the drive. Examples are listed below. **sda2** is the second partition **(2)** on the hard drive with the lowest ID number **(a)**. **sdc1** is the first partition **(1)** on the hard drive with the third lowest ID number **(c)**. **sda1** is the first partition **(1)** on the hard drive with the lowest ID number **(a)**. **sdb3** is the third partition **(3)** on the hard drive with the second lowest ID number **(b)**. **sdc2** is the second partition **(2)** on the hard drive with the third lowest ID number **(c)**. **sdd1** is the first partition **(1)** on the drive with the fourth lowest ID number **(d)**.
/dev/srn	This is a special designation used to **identify optical drives in the system**. The optical drive with the lowest ID number is addressed as **sr0**. The optical drive with the next lowest ID number is addressed as **sr1**, and so on. Many distributions include symbolic links named /dev/cdrom or /dev/dvd that point to the actual device file (sr0).
/dev/fdn	**fd** files **identify floppy drives**. Device numbering begins at 0. For example, /dev/fd0 is the first floppy drive.
/dev/ttyn	**tty** files **identify local terminals** on the system. Device numbering begins at 0. Subsequent terminals are represented with files that increment by one (for example, the file for terminal two is /dev/tty1, and so on).
/dev/ttySn	**ttyS** files **identify serial ports**. Device numbering begins at 0. Files for subsequent serial ports are represented by files that increment by one (for example, the file for serial port two is /dev/ttyS1, and so on).
/dev/lpn	**lp** files **identify parallel ports**. Device numbering begins at 0. Files for subsequent parallel ports are represented by files that increment by one (for example, the file for parallel port two is /dev/lp1, and so on).
/dev/stn	**st** files **identify SCSI tape devices**. Device numbering begins at 0.

8.3 GPT Management Tools

Command	Function
gdisk	gdisk does the following: • Creates and deletes GPT partitions • Displays information about a partition • Changes the partition name and type • Verifies a hard disk • Backs up and restores a disk's partition table • Converts an MBR partition table to a GPT partition table. The syntax for using gdisk is **gdisk *device_name*** . The following options can be used within gdisk: **?** displays the help screen. **b** backs up GPT information to a file. **c** changes a partition's name. **d** deletes a partition. **i** displays detailed partition information. **l** lists partition type codes. **n** adds a new partition. **o** creates a new GUID partition table. **p** prints the partition table. **q** quits gdisk without saving changes. **s** sorts the list of partitions. **t** changes a partition's type code. **v** verifies a storage device. **w** writes changes to the partition table of the storage device and exits gdisk.
parted	parted does the following: • Creates and deletes GPT partitions • Modifies GPT partitions The parted command writes partition changes to disk immediately. Carefully plan any partition changes you want to make before using this command. Run parted at the shell prompt. The following commands can be used within parted: • select device_name identifies which storage device to edit. • mkpart partition_type start_point end_point creates a new partition. For example: ◦ To create a standard Linux partition, specify Linux as the partition type. ◦ To create a partition that starts at 1 GB and ends at 21 GB, specify a starting point of 1024 and an end point of 21504. • print displays a list of partitions on the device. • name partition_name renames a partition. • move partition start_point end_point moves a partition to a different location on the storage device. • resize partition start_point end_point resizes a partition. • rm partition deletes a partition.

8.4 LVM Components

Component	Description
Physical volume	Physical volumes are physical block devices or other disk-like devices that are used by the LVM as the building blocks for volume groups. Physical volumes can be: • Regular storages devices, such as a whole hard disk. • Partitions on a hard disk. • Devices created by the device mapper, like a RAID array.
Volume group	The LVM combines physical volumes into storage pools known as volume groups. A volume group consists of all the space available on the physical volumes grouped together. • The storage space within a volume group can come from many different physical volumes on many different storage devices. • Additional hard disks or additional partitions can be added to the volume group at any time.
Logical volume	A volume group can be divided up into any number of logical volumes. • Logical volumes are the primary component that users and applications interact with. • Logical volumes are functionally equivalent to a partitions on a physical disk. • Logical volumes can be formatted to accommodate a file system. • Logical volumes can be resized and moved while they are still mounted and running. • Logical volumes may be identified by using descriptive names (e.g., Research or Marketing) instead of physical disk names such as SDA and SDB.

8.4 LVM Commands

Command	Description
pvcreate	Initializes the physical volume for later use by the LVM
pvscan	Scans all disks for physical volumes and displays all found physical volumes on the system and their associated volume groups. The pvs command does nearly the identical thing and is frequently used in place of pvscan.
vgcreate	Creates a new volume group
vgscan	Search for all volume groups. The vgs command does nearly the identical thing and is frequently used in place of vgscan.
vgextend	Adds one or more physical volumes to an existing volume group, increasing its available storage space
lvcreate	Creates a new logical volume from the space available in a volume group. Below are some options for this command. **-L** specifies the size. Use the following size suffixes: **K** for kilobytes **M** for megabytes **G** for gigabytes **T** for terabytes **P** for petabytes **E** for exabytes **-n** specifies the name.
lvchange	Change the attributes of logical volumes
lvscan	Scans all known volume groups in the system for logical volumes and displays the results. The lvs command does nearly the identical thing and is frequently used in place of lvscan.
lvresize	Resize a logical volume
lvextend	Extends the size of a logical volume (The -L option specifies the new volume size.)

8.5 Linux File System Types

Type	Description
ext2	The Second Extended File System (ext2) is one of the oldest Linux file systems still available. • ext2 stores data in a standard directory and file hierarchy. • The maximum file size supported is 2 TB. • An ext2 volume can be up to 4 TB in size. • File names can be up to 255 characters long. • Linux users, groups, and permissions are supported. • ex2 does not use journaling (which is used in most modern file systems). As a result, ext2 takes a long time to recover if the system shuts down abruptly.
ext3	The Third Extended File System (ext3) is an updated version of ext2 that supports journaling. Before committing a transaction to a storage device, the ext3 file system records the transaction to the journal and marks it as incomplete. After the disk transaction is complete, the file system marks the transaction as complete in the journal. By doing this, ext3 can keep track of the most recent file transactions and whether or not they were completed. This allows ext3 to recover much more quickly than ext2 in the event of an unclean system shutdown.
ext4	ext4 is the fourth generation file system in the ext file system family. ext4 includes all of the features found with ext2 and ext3 with the addition of the following features: • Support for file sizes up to 16 TB and disk sizes up to 1 exabyte (EB). • Supports up to four billion files in the file system. • Uses checksums to verify the integrity of the journal file itself. Checksums help improve the overall reliability of the system because the journal file is the most heavily used file of the disk.
swap	A swap file system is used as virtual memory (the portion of the hard disk used to temporarily store portions of main memory) by the operating system. A recommended practice is to make the swap file size between 1 and 1.5 times the amount of memory on the computer.
NTFS	Microsoft operating systems use NTFS (New Technology File System). Linux provides limited support for NTFS.
VFAT	VFAT is a FAT32 file system for Linux and does not support journaling. VFAT includes long name support. Support for VFAT must be compiled into the kernel for the system to recognize the VFAT format.
XFS	The XFS file system was developed for the Silicon Graphics IRIX operating system. An XFS file system is proficient at handling large files, offers smooth data transfers, and provides journaling. It also can reside on a regular disk partition or on a logical volume.
Btrfs	Btrfs is a Linux file system that uses a copy-on-write file system. Using copy-on-write technology, Btrfs provides several key features not found in earlier file systems, including storage pools and snapshots. • Instead of using traditional disk partitions, Btrfs allows you to create storage pools from the storage devices in your system. From the storage pool, you can then allocate space to specific storage volumes. Instead of mounting partitions, you mount storage volumes at mount points in the file system. • The snapshot functionality provided by Btrfs protects data. It can be configured to take snapshots of your data at specified intervals and save it on separate media. If a file ever gets lost or corrupted, you can restore a previous version of the file from a snapshot.

8.5 Formatting Commands

Command	Function
mkfs	Creates an ext family file system or a fat file system. The mkfs command uses the following options: **-t [*file_system_type*]** determines the file system. File system types include: ◦ **ext2** (identical to the mkfs.ext2 command) ◦ **ext3** (identical to mkfs.ext3) ◦ **ext4** (identical to mkfs.ext4) ◦ **vfat** (identical to mkfs.vfat) **-b** specifies the block size. Supported values are 1024, 2048, or 4096. **-i** determines how many inodes are on the partition and uses the same values as -b. **-j** appends a journal to an ext2 file system. Without the **-b** and **-i** options, **mkfs** calculates the optimal values for you automatically.
mkswap	Creates a swap partition. A swap partition is the location on the hard drive where an operating system writes memory information when it runs out of RAM. • The swapon command must be run to activate the swap partition. • The swapoff command is used to deactivate swap partitions. Both swapon and swapoff use the **-a** option to enable or disable all swap partitions listed in **/etc/fstab.**
mke2fs	Create an ext2, ext3, or ext4 file system. Command options include the following: **-b** specifies the block size of the file system in bytes (valid sizes are 1024, 2048, and 4096 bytes per block). **-j** creates the file system with an ext3 journal. **-L** sets the volume label for the file system. **-n** displays what mke2fs would do if it created a file system, but does not actually create the file system. **-t** specifies the file system type (such as ext2, ext3, or ext4) to be created.

8.6 Manage and Monitor Mountings

File	Description
/etc/fstab	The /etc/fstab file identifies devices to mount each time the system boots. When the system boots, it automatically mounts the volumes identified in the file. The file contains entries with six fields that control how a device is mounted. The following is a typical fstab entry: **/dev/sda3 /mnt/disk1 ext3 auto,ro,nosuid,users 0 1** An entry consists of the following variables, which are described below: **[device_to_mount] [mount_point] [file_system_type] [options] [dump] [fsck]** • **Device_to_mount** is the path to the device file or the label that describes the storage device to be mounted. • **Mount_point** specifies where to mount the device. This is the directory where the data on the device can be accessed. • **File_system_type** specifies the type of file system that has been created on the storage device. • Options specify the additional options to be used when mounting the device. Multiple options are separated by commas. ◦ **sync** enables synchronous I/O. Changes are written to disk immediately. This option is normally used for removable storage devices (async disables this function). ◦ **async** enables asynchronous I/O. Changes are cached and then written when the device isn't busy. This option is normally used for non-removable devices such as hard drives (sync disables this function). ◦ **atime** updates the timestamp on each file's inode (noatime disables this function). ◦ **auto** allows the device to be mounted automatically when the system boots. ◦ **noauto** prevents the device from being mounted automatically when the system boots. ◦ **dev** allows block files to be read from the device (nodev disables this function). ◦ **exec** allows programs and script files in the file system to be run (noexec disables this function). ◦ **owner** identifies that only the device owner can mount the file system. ◦ **ro** mounts the storage device as read-only. ◦ **rw** mounts the storage device as read/write. ◦ **suid** allows the SUID bit to be set on files in the file system (nosuid disables this function). ◦ **nouser** allows only the root user to mount the file system. ◦ **users** allows any user to mount the file system. ◦ **defaults** uses the following default settings: rw, suid, dev, exec, auto, nouser, and async. • Dump determines whether the file system needs to be dumped. If set to a value of 0, it is assumed that the file system does not need to be dumped. If set to a value of 1, the file system will be dumped. • fsck determines the order to run fsck (file system check) during system boot. This field should always be set to a value of 1 for the device containing the root file system (/). All other file systems should be set to a value of 2.
/etc/mtab	The /etc/mtab file tracks the currently mounted volumes on the system.
/procs/mounts	The /procs/mounts file contains entries for all currently mounted volumes on the system.
systemd.mount	A unit file that encodes information about a file system mount point controlled and supervised by systemd.

8.6 Manage File System Mountings

Command	Description
mount /dev/[device] [mountpoint]	Mount a volume or device. Common mount options: **-a** mounts all file systems listed in the /etc/fstab file. **-r, ro** mounts the volume as read-only. **-w, rw** mounts the volume as read/write. **-t** specifies the volume type. (If you mount an ext3 file system without the -t, the system recognizes it as an ext2 file system.) **-o loop** mounts an ISO file.
mount	View the currently mounted volumes on the system.
df	View which file systems are mounted to specific mount points.
umount [device] **umount [mountpoint]**	Unmount a volume or device from the system. If a "**disk is busy**" error message is displayed when unmounting a device: • Make sure that the current working directory is not in that file system. • Close any open files located on that file system.

8.7 File System Integrity Commands

Command	Description
df	Displays the free space in the partition holding the specified directory. If no directory is given, the space available on all currently mounted file systems is shown. Disk space is shown in 1-K blocks by default. Common options include: **-h** displays the output in get human readable format (bytes, KB, MB, GB, TB). **-i** displays inode information. **-l** limits the list to local file systems.
du	Displays files and file sizes in and below a specified directory. Common options include: **-c** lists the total amount of space used in the directory. **-h** displays the output in human readable format (bytes, KB, MB, GB, TB). **-s** lists only the total, not each file. **-a** evaluates all files, not just directories.
lsof	Displays open files in the file system. lsof gives the following information by default: • The command used to access the file. • Process ID. • Name of the user who is accessing the file. • A file descriptor (these are described in the lsof man pages). • File node type. • Device numbers. • File size. • Inode address. • File path. Common options include: **+D [directory_name]** recursively lists files in a directory. **-c [command_name]** lists all files for processes that are executing the specified command. **-u [user]** lists open files owned by the specified user. **-g [process_ID]** lists files opened by a specific process.
fsck	Checks and optionally repairs one or more Linux file systems. Common options include: **-s** serializes fsck when multiple file systems are checked. **-t** specifies the type(s) of file system to be checked. **-a** automatically repairs the file system without any questions. **-r** prompts for confirmation when errors are found and asks permission to fix the errors (only when -a is not specified). Be aware of the following: • The file system must be unmounted before using fsck. • When manually running fsck, use runlevel 1 (init) or rescue.target (systemd) to ensure that other users do not mount the file system.
dumpe2fs	Prints superblock and block information for an ext2, ext3, or ext4 file system. This includes information for each sector on the partition about sector type, block ranges, inode information, free blocks, and similar information. Command options include: **-b** prints blocks reserved as bad in the file system. **-h** prints only superblock information. **-x** prints group information block numbers in hexadecimal format.
tune2fs	Adjusts tunable file system parameters on ext2, ext3, and ext4 file systems. Some of the adjustable parameters include volume label, reserved blocks, inode sizes, and journaling. Tune2fs can also implement access control lists for individual users. Command options include: **-c** adjust the number of mounts, after which the file system will be checked. **-e remount-ro** remounts the file system as read-only. **-l** lists the contents of the file system superblock. **-o acl** enables Posix access control lists. **-j** converts ext2 file systems to ext3 file systems.

debugfs

An ext2/ext3/ext4 file system debugger. Can be used for information gathering about target partitions, including directory listings with deleted file entries. Also allows file system modification and deleted file recovery.

Command options include:

- **-w** the file system should be opened in read-write mode. If not included, the file system will be read-only.
- **-c** open the file system in catastrophic mode. This ignores inodes and group bitmaps initially. Useful when a file system has significant corruption.
- **-f *cmd_file*** will read in commands from the cmd_file and execute them.
- **-V** print the debugfs version number and exit.

iostat

Monitors system I/O device loading by observing the time devices are active in relation to their average transfer rates. The iostat command generates reports that can be used to change system configuration to better balance the input/output load between physical disks. Running iostat without any options displays CPU usage and I/O statistics in the form of how much has been written per second and in total.

Command options include:

- **-m** displays the results in megabytes (MB) instead of kilobytes (KB).
- **-d** only displays the statistics for the devices connected on the system.
- **-p device** displays the results for the specified device.
- **-x** adds extended statistics, such as avgqu-sz. This statistic shows the number of operations that were either queued or being serviced on a device. If this is not in the single digits (with an occasional double-digit spike), more troubleshooting may be required.
- **number** When a number (such as 5) is used, iostat will continue displaying statistics for that specified time in seconds. Press Ctrl + c to exit.

See the man pages for additional options.

ioping

This tool generates various I/O patterns and lets you monitor I/O speed and latency in real-time. This tool shows disk latency in the same way as ping command shows network latency on Linux or Unix-like systems.

Command options include:

- **-c *count device*** runs for the number of specified count requests for the specified device.
- **-R *device*** shows the disk seek rate for the specified device.

See the man pages for additional options.

badblocks

A bad sector or block is a section on a disk drive to which data can no longer be written to read from. Included by most Linux distributions, badblocks are used to search for bad blocks on a device (usually a disk partition), where the device is the special file corresponding to the device (e.g., /dev/sda).

Command options include:

- **-b *block-size*** specifies the size of blocks in bytes. The default is 1024.
- **-c *number of blocks*** is the number of blocks that are tested at a time. The default is 64.
- **-e *max bad block count*** specifies a maximum number of bad blocks before aborting the test. The default is 0, meaning the test will continue until the end of the test r is reached.
- **-i *input_file*** reads a list of already existing known bad blocks. Badblocks will skip testing these blocks since they are known to be bad.
- **-n** uses non-destructive read-write mode. By default, only a non-destructive read-only test is done. This option must not be combined with the -v as they are mutually exclusive.
- **-o *output_file*** writes the list of bad blocks to the specified file.
- **-s** shows the progress of the scan by writing out rough percentage completion of the current badblocks pass over the disk.
- **-v** Verbose mode.
- **-w** uses write-mode test. With this option, badblocks scans for bad blocks by writing some patterns (0xaa, 0x55, 0xff, 0x00) on every block of the reading every block, and comparing the contents. This option may n combined with the -n option, as they are mutually exclusive.
- **-X** an internal flag to be used only by **e2fsck** and **mke2fs**. It bypasses the exclusive mode in-use device safety check.

Warning

Never use the -w option on a device containing an existing file system. This option erases data! If you want to do write-mode testing on an existing file system, use the -n option instead. It is slower, but it will preserve your data.

See the man pages for additional options.

xfs_metadump

This command is used to copy the metadata (such as filenames and file sizes) from an XFS file system to a file, but can only be used to copy unmounted file systems or read-only mounted file systems.

Be aware that by default, xfs_metadump obfuscates most file (regular file, directory and symbolic link) names and extended attribute names to allow the dumps to be sent without revealing confidential information.

Command options include:

- **-a** Copies entire metadata blocks.
- **-e** Stops the dump on a read error.
- **-g** Shows dump progress.
- **-o** Disables obfuscation of file names and extended attributes.

See the man pages for additional options.

8.7 I/O Scheduler

Scheduler	Description
Noop	Noop is the simplest scheduler. It places all I/O requests into a First in/First Out (FIFO) queue. In addition, read/write requests of a similar purpose are also combined to reduce the number of disk operations and increase the length of system calls. This scheduler is often used for systems that do not need an I/O scheduler. For example, when a virtual machine (VM) is running on a host computer that's already using its own I/O scheduler.
Deadline	The Deadline scheduler creates a read queue and a write queue. Since each I/O request has an associated timestamp (used by the kernel for an expiration time), the Deadline scheduler utilizes this timestamp to push I/O requests that've reached their deadline to their highest priority. The default Deadline values are 500 ms for read operations and 5,000 ms for write operations. If needed, these values can be adjusted. Because of these values, the Deadline scheduler is often considered the optimal scheduler for read-heavy workloads.
CFQ	The Complete Fairness Queuing (CFQ) input/output (I/O) scheduler works by creating a per-process I/O queue. The goal of CFQ is to provide a fair I/O priority to each process. This is accomplished by first ordering the queues to reduce disk seeking and then servicing these per-process I/O queues, in a round-robin fashion. The benefits of using the CFQ scheduler is that it tries to provide each process with the same priority for disk access. The disadvantage is that it makes this schedule less optimal for environments that might need to prioritize one request type (such as reads) from a single process.

8.8 Implement Disk Quotas

Step	Procedure
Install the quota package	Use yum, zypper, or apt-get to install the quota package on the system where you want to set the quota limits.
Edit the mount options in /etc/fstab	Edit the /etc/fstab file to add the mount options for the file system and enable quotas. • **usrquota** enables quotas for users. • **grpquota** enables quotas for groups.
Create the quota files	Create the aquota.user and aquota.group files in the directory where the partition is mounted.
Enable the quotas and view a quota report	Enable disk quotas and then generate a disk usage and quota report. The report shows: • How much space to allocate to each user. • How much space is currently consumed by each user. • Whether some users are using a significant amount of disk space.
Edit the quotas	Edit a quota for the specified user or group. Be aware of the following when editing quotas: • Set the soft and hard quotas for blocks. This limits the total amount of disk space per user or group. • Set the hard and soft quotas for inodes. This limits the total number of files and directories per user or group. • Users may exceed soft quotas for the number of days specified in the grace period (seven by default). When the grace period expires, users can't create additional files. • Users cannot exceed hard quotas. • When setting block quotas, 1,000 blocks is about 1 MB, and 1,000,000 blocks is about 1 GB. • Setting the quota limits to 0 removes all quotas.

8.8 Disk Quota Commands

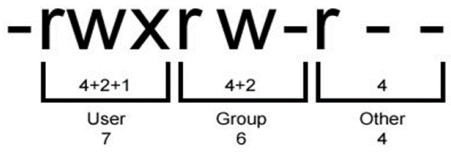
Command	Function
quotacheck -mavug	Creates the aquota.user and aquota.group files in the file system (after placing the quota entries in /etc/fstab). Consider the common options below. **-m** updates the quota database even if other processes are running on the file system. **-a** updates the quota database. **-v** runs the command in Verbose Mode. **-u** run the database updates for users and groups, respectively. **-g** " "
quotaon	Enables quotas for the mounted file system. Consider the common options below. **-a** enables all mounted file systems listed in /etc/mtab. **-v** runs the command in Verbose Mode.
quotaoff	Disables quotas for the mounted file system.
repquota	Displays a summary of the disc usage and quotas for the specified file systems, including the specific number of files and space by user. Consider the common options below. **-v** reports all quotas, even if there is no usage. **-n** does not resolve user and group names to speed printing time. **-u** give reports for users and groups, respectively. **-g** " " **-a** gives information for all file systems listed in /etc/mtab.
edquota	Opens and edits a user's quota, a group's quota, or changes the grace period. Consider the common options below. **-u** changes a user's quota. **-g** changes a group's quota. **-t** changes the grace period.
quota	Displays the current user's or group's quota. Consider the common options below. **-u** shows the current user's quota. **-g** shows the current group's quota. **-v** shows current usage (the hard and soft quota for blocks and inodes).

8.9 File Ownership

Command	Function
ls -l	View a long listing of files and directories. The long listing shows the mode of each file and directory along with ownership information. The output listed is (from left to right): • Permissions • Number of links • Owner name • Group name • Number of bytes in the file • Last modified time • File name
chown	Change the ownership of a file or directory. Be aware of the following options: **-R** changes the ownership of the file recursively throughout the directory tree. ***user*** changes the file ownership only. ***user:group*** or ***user.group*** change the user and group ownership of the file. ***:group*** or ***.group*** changes the group ownership only.
chgrp	Change the group owner of a file or directory.

8.10 (Permissions) Inode Modes

Permission	Letter Abbreviation	Octal Value	Allowed Actions on Files	Allowed Actions on Directories
Read	**r**	**4**	Open and read the file	List directory contents if the Execute permission is also present
Write	**w**	**2**	Edit the file and save the changes	Add, delete, and rename files if the Execute permission is also present
Execute	**x**	**1**	Execute the file if it's a program file or a shell script (must be used in conjunction with the Read permission)	Enter the directory and access its contents



8.10 Manage Permissions

Command	Function
ls -l	View a long listing of files and directories. A long listing displays the permissions assigned to files and directories (among other information).
chmod	Change the permissions for the specified file. You can use the following syntax options: • ***entity+permission*** adds a permission for a user, group, or other to a file or directory. • ***entity-permission*** removes a permission for a user, group, or other from a file or directory. • ***entity=permission*** sets the permission equal to the permission specified for a user, group, or other for a file or directory. • ***decimal_value*** sets the permissions for the file according to the numbers represented for each mode entity. **-R** sets permissions recursively.
getfacl file	For each file, getfacl displays the file name, owner, group, and the access control list (ACL). If a directory has a default ACL, getfacl also displays the default ACL. Non-directories cannot have default ACLs. getfacl options include the following: **-a** displays the file access control list. **-d** displays the default access control list. **-c** tells the command to NOT display the comment header (the first three lines of each file's output). **-e** prints all effective rights comments, even if they're identical to the rights defined by the ACL entry. **-E** tells the command to NOT print effective rights comments. **-s** skips files that only have the base ACL entries (owner, group, and others). **-R** lists the ACLs of all files and directories recursively. **-L** uses logical walk by following symbolic links to directories. The default behavior is to follow symbolic link arguments and to skip symbolic links encountered in subdirectories. (This is only effective in combination with -R.) **-P** uses physical walk by not following symbolic links to directories. This also skips symbolic link arguments. (This is only effective in combination with -R.) **-t** uses an alternative tabular output format. The ACL and the default ACL are displayed side by side. Permissions that are ineffective due to the ACL mask entry are capitalized. The entry tag names for the ACL_USER_OBJ and ACL_GROUP_OBJ entries are also displayed in capital letters, which helps in spotting those entries. **-p** tells the command to NOT strip leading slash characters ('/'). The default behavior is to strip leading slash characters. **-n** lists numeric user and group IDs. **-v** prints the version of getfacl and exits.

setfacl file

This utility sets access control lists (ACLs) for files and directories.

setfacl options include the following:

-m modifies the ACL of a file or directory. ACL entries for this operation must include permissions.

-x remove ACL entries. It is not an error to remove an entry which does not exist. Only ACL entries without the perms field are accepted as parameters, unless the POSIXLY_CORRECT environment variable is defined.

-b removes all extended ACL entries. The base ACL entries of the owner, group and others are retained.

-k removes the default ACL. If no default ACL exists, no warnings are issued.

-n tells the command to NOT recalculate the effective rights mask. The default behavior of setfacl is to recalculate the ACL mask entry, unless a mask entry was explicitly given. The mask entry is set to the union of all owning group permissions and all named user and group entries. (These are exactly the entries affected by the mask entry.)

-d tells the command that all operations apply to the default ACL. Regular ACL entries in the input set are promoted to default ACL entries. Default ACL entries in the input set are discarded. (A warning is issued if that happens.)

-R applies operations to all files and directories recursively. This option cannot be mixed with --restore.

-L uses logical walk by following symbolic links to directories. The default behavior is to follow symbolic link arguments and to skip symbolic links encountered in subdirectories. { This is only effective in combination with -R and cannot be mixed with --restore.)

-P uses physical walk by not following symbolic links to directories. This also skips symbolic link arguments. (This is only effective in combination with -R and cannot be mixed with --restore.)

-v prints the version of setfacl and exits.

8.11 Calculating umask

Umask Calculation	For Files (binary)	For Directories (binary)	For Files (letter abbreviation)	For Directories (letter abbreviation)
Default permission	666	777	rw- rw- rw-	rwx rwx r
Umask *(minus)*	22	22	--- -w- -w-	--- -w- -v
Result *(equals)*	644	755	rw- r-- r--	rwx r-x r-

8.11 umask Commands

Command	Description
umask	Displays the current umask setting. **-S** switch indicates that a symbolic representation of a mask will be used. ***number*** changes the default umask to a number between 000 and 777. ***symbol*** changes the default umask to a symbolic representation of a mask.

8.11 umask symbolic (letter) representation of the mask.

Symbol	Definition
u	User (the owner of the file)
g	Group (any member of the file's defined group)
o	Other (anyone else)
a	All (equivalent to all of the above - u, g, and o)

8.12 Special Permissions

Permission	Letter Abbreviation	Example	Octal Value	Description
SUID (Set User ID)	**s** in the execute permission position of the user permissions	rwsrw-rw-	4	If the SUID bit is set, the program will run with the permissions of the file owner, not with the permissions of the user who runs the program. • The most common use of SUID is to allow users to run a command as the root user. • Users do not become the root user, but rather the command or program runs as if executed by the root user. • Some programs require the SUID bit to be set for proper functionality. • Be careful in setting the SUID bit, as it could give a program too many permissions.
SGID (Set Group ID)	s in the execute permission position of the group permissions	rwxrwsrw-	2	If the SGID bit is set: • On a file: the program will run with the group permissions of the group owner. • On a directory: a newly created file will receive/inherit the same group owner as assigned to the parent directory.
Sticky bit	**t** in the execute permission position of the other permissions	rwxrw-rwt	1	This marks the file in such a way as to prevent the file's deletion from the system by anyone except the file owner. Setting the sticky bit works particularly well with shared files. *Sticky bits can also be set on directories.*

8.13 ACL

Command	Options
getfacl	The most commonly used getfacl options are: **-a** displays access control lists **-d** displays default access control lists
setfacl	The most commonly used setfacl options are: **-m** to modify or add an ACL **-d** to change default ACLs **-m u:*<user>*:*<permissions>*** to change user permissions **-m g:*<group>*:*<permissions>*** to change group permissions **-x** to remove an ACL **-b** to remove all ACLs except the owner and group owner

8.14 (Archive) Using tar

Command	Options and Descriptions	
tar	**-A**	Appends one tar file to another archive file.
	-c	Creates a new archive.
	-d	Identifies differences between the files in an archive file and the same files in the file system.
	-v	Displays a list of all files being written into the archive.
	-f	Specifies the file to create or unpack. Without this option, tar uses standard input and output as the destination.
	-x	Extracts the files. If no destination directory is specified, then tar extracts the files to the current working directory.
	-z	Compresses and decompresses a file using the gzip utility (normally named with a .gz extension).
	-j	Compresses and decompresses a file using the bzip2 utility (normally named with a .bz2 extension).
	-J	Compresses and decompresses a file using the xz utility (normally named with a .xz or .lzma extension.
	-C	Changes to a specific directory to extract the files.
	-t	Lists the contents of an archive.
	-P	Tells tar to not strip the leading / from filenames as they are added to the archive.
	-r	Adds files to the end of an existing tar archive.
	-u	Adds files to the end of an existing tar archive only if they are newer than the existing files in an archive.
	-X *file_name*	Causes tar to exclude the file names contained in the specified file when creating an archive file.

8.14 (Archive) Using gzip

Command	Options and Description	
gzip	**-c**	Writes the file to standard output.
	-d	Decompresses the file.
	-l	Displays information about files in an archive.
	-r	Recursively compresses all files in directories and subdirectories.
		This is the same as the tar -z command.

8.14 (Archive) Using xz

Command	Options and Description	
xz	**-z**	Compresses a file.
	-d	Decompresses a file.
	-k	Keeps the original file unchanged.
		This is the same as the tar -J command.

8.14 (Archive) Using bzip2

Command	Options and Description	
bzip2	**-z**	Compresses a file.
	-d	Decompresses a file.
	-k	Keeps the original file unchanged.
		This is the same as the tar -j command.

8.14 (Archive) Using zip

Command	Options and Description	
zip	**-d**	Removes a file from the zip archive. When a zip archive includes multiple files, use this option to remove a file from the archive.
	-u	Updates the file in the zip archive. The opposite of -d, meaning you can use this option to add a new file to the zip file already created.
	-m	Deletes the original files after zipping.
	-r	Lets you zip a directory recursively.
	-x	Lets you exclude the file's files while creating the zip of multiple files, such as a directory.
	-v	Verbose mode or print diagnostic version information.

8.14 (Archive) Using cpio

Command	Options and Description	
cpio	**-o**	Creates the archive in copy-out mode.
	-v	Causes cpio to display verbose output, showing file names as they're added or removed.
	-i	Extracts files by invoking copy-in mode.
	-u	Overwrites existing files.
	-d	Creates directory paths (if needed) during extraction.
	-t	Displays archive contents without extracting files.
	-p	Copies files to a new directory (copy-pass mode).

8.14 (Archive) Using dd

Command	Options and Description	
dd	**bs=BYTES**	Read and write up to bytes at a time (the default is 512; overrides ibs and obs)
	cbs=BYTES	Convert bytes at a time
	conv=CONVS	Convert the file as per the comma-separated symbol list
	count=N	Copy only N input blocks
	ibs=BYTES	Read up to bytes at a time (the default is 512)
	if=FILE	Read from FILE instead of stdin
	iflag=FLAGS	Read as per the comma-separated symbol list
	obs=BYTES	Write bytes at a time (the default is 512)
	of=FILE	Write to FILE instead of stdout
	oflag=FLAGS	Write as per the comma separated-symbol list
	seek=N	Skip N obs-sized blocks at start of output
	skip=N	Skip N ibs-sized blocks at start of input
	status=LEVEL	The level of information to print to stderr - none suppresses everything but error messages, noxfer suppresses the final transfer statistics, and progress shows periodic transfer statistics