**Unit 6**

**System Administration:**

Root Login:

*Root* is the superuser account in Unix and Linux. It is a user account for administrative purposes, and typically has the highest access rights on the system.

Usually, the root user account is called root. However, in Unix and Linux, any account with [user id](https://www.ssh.com/iam/user/id) 0 is a root account, regardless of the name. It is fairly common for certain system administrators to have their own root accounts on a system, with their own passwords.

In Linux and Unix-like computer operating systems, the root is the conventional name of the user who has all rights or permissions (to all files and programs) in all modes (single- or multi-user). The root user can do many things an ordinary user cannot, such as changing the ownership of files and binding to ports numbered below 1024. The term’s etymology may be that root is the only user account with permission to modify the root directory of a Unix system. The [duties of a system administrator](https://www.cyberciti.biz/faq/what-is-the-role-of-the-system-administrator/) are wide-ranging and vary widely from one organization to another. Sysadmins are responsible for installing, supporting, and maintaining servers or other computer systems and planning for and responding to service outages and other problems.

You need to use any one of the following command to log in as superuser or root user on Linux:

1. su command – Run a command with substitute user and group ID in Linux
2. sudo command – Execute a command as another user on Linux
3. doas command – Doas is al alternative to sudo command which comes from OpenBSD project. It works on many Linux distros such as Alpine Linux and others.
4. [/etc/passwd](https://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/) – Linux user account file.
5. [/etc/group](https://www.cyberciti.biz/faq/understanding-etcgroup-file/) – Linux group membership that grants users superuser access.
6. [/etc/shadow](https://www.cyberciti.biz/faq/understanding-etcshadow-file/) -Linux password information for each user account and all passwords are in hashed format.

Administer’s Privileges:

Privilege Management for Unix and Linux enables Unix and Linux system administrators to specify the circumstances under which users may run certain programs as root or as other important accounts. The result is responsibility for actions such as adding user accounts, fixing line printer queues, etc., allowing permissions to be safely assigned to the appropriate person without disclosing the root password. The full power of root is protected from potential misuse or abuse such as:

* Modifying databases
* File permissions
* Erasing disks

Furthermore, Privilege Management for Unix and Linux is capable of selectively recording all input and output from a session. Having this audit trail combined with the safe partitioning of root functionality provides an extremely secure means of sharing the root password.

The pbreplay utility enables you to view sessions while they are happening or at a later date. Privilege Management for Unix and Linux can also require a checksum match before running any program, guarding against viruses or Trojan horse attacks on important accounts.

Privilege Management for Unix and Linux can access existing programs as well as its own set of utilities that execute common system administrative tasks. Utilities that are being developed to run on top of Privilege Management for Unix and Linux can manage passwords, accounts, backups, line printers, file ownership or removal, rebooting, logging people out, killing their programs, deciding who can log in to what machines from where, and so on. Users can work from within a restricted shell or editor to access certain programs or files as root.

Privilege Management for Unix and Linux can also optionally encrypt all network traffic that it generates, including control messages, input that is keyed by users, and output that is generated by commands that are run through Privilege Management for Unix and Linux. This encryption protects sensitive data from network monitoring.

Privilege Management for Unix and Linux is built upon two major concepts:

* Partitioning the functionality of root (and other important accounts) to allow multiple users to carry out system administration actions without full access to the administrative account or its password
* Creating an audit trail of such actions

Privilege Management for Unix and Linux enables system administration actions to be partitioned without compromising root account security. Privilege Management for Unix and Linux enables the system administrator to specify if, and when, a use’s request to run a program is accepted or rejected.

Through Privilege Management for Unix and Linux, each user can request that a program be run on a machine as root (or as another important account, such as oracle or admin). Privilege Management for Unix and Linux evaluates the request. If the request is accepted, Privilege Management for Unix and Linux runs the program locally or across a network, on behalf of the user.

With Privilege Management for Unix and Linux, help desk personnel can reset passwords for users that have forgotten them and reinstate user accounts. Project members can clear a jammed line printer queue, kill hung programs, or reboot certain machines. Administration staff can print or delete resource usage logs or start backups.

Through partitioning, Privilege Management for Unix and Linux allows different users to perform those root actions for which they are responsible, but not anything else. Privilege Management for Unix and Linux enables you to specify:

* Which users can perform a particular task
* Which tasks can be run through the system
* When the user can perform the task
* On which machine the task can be performed
* From which machine the user may initiate a request to perform the task
* Whether another user’s permission (in the form of a password) is required before the task is started
* The decisions that are to be made by a program that you supply and which Privilege Management for Unix and Linux calls to determine if a request should be accepted or rejected
* Many other miscellaneous properties of requests

Privilege Management for Unix and Linux is capable of recording all activity that passes through it to the I/O level. The power to accurately log activities in a safe environment enables you to implement a secure system administration regime with an audit trail. You always know exactly what is happening in root, as well as who did it, when it happened, and where.

Because root can modify any file, special precautions must be taken to ensure the Privilege Management for Unix and Linux logs are secure. Privilege Management for Unix and Linux can be configured to receive user requests from the submitting machine, execute tasks on the execution machine, and log all of the activities on yet another, very secure machine.

If necessary, the machines that contain the policy files and the log files can be made physically inaccessible to users and isolated from remote login over the network. In addition, the logs can be printed to hardcopy on a secure printer or recorded to a WORM drive if required.

User Management (useradd, usermod, userdel ,groupadd)

Since Linux is a multi-user operating system, several people may be logged in and actively working on a given machine at the same time. Security-wise, it is never a good idea to allow users to share the credentials of the same account. In fact, best practices dictate the use of as many user accounts as people needing access to the machine.

At the same time, it is to be expected that two or more users may need to share access to certain system resources, such as directories and files. User and group management in Linux allows us to accomplish both objectives.

Adding a new user involves dealing with an account other than your own which requires superuser (aka root) privileges.

The same applies to other user or group management tasks, such as deleting an account, updating accounts, and creating and removing groups.

These operations are performed using the following commands:

* adduser: add a user to the system.
* userdel: delete a user account and related files.
* addgroup: add a group to the system.
* delgroup: remove a group from the system.
* usermod: modify a user account.

Managing Disk space,

Check Disk Space in Linux Using the df Command:

df, which stands for Disk Filesystem, is used to check disk space. It will display available and used storage of file systems on your machine.

When executing this command, you will see the default columns: Filesystem, Size, Used, Available, Use%, and Mounted On.

* FileSystem — provides the name of the file system.
* Size — gives us the total size of the specific file system.
* Used — shows how much disk space is used in the particular file system.
* Available — shows how much space is left in the file system.
* Use% — displays the percentage of disk space that is used.
* Mounted On — tells us the mount point of a particular file system.

By adding a certain option to the df command, you can check the disk space in Linux more precisely. These are the most popular options:

* df -h — it will display the result in a human-readable format.
* df -m — this command line is used to display information of file system usage in MB.
* df -k — to display file system usage in KB.
* df -T — this option will show the file system type (a new column will appear).
* df /home — it allows you to view information about a specific file system in a readable format (in this case /home file system).
* df — help — it lists down other useful options that you can use, complete with their descriptions.

### Check Disk Usage in Linux Using the du Command

Another important command is **du**, short for **Disk Usage**. It will show you details about the disk usage of files and directories on a Linux computer or server. With the **du** command, you need to specify which folder or file you want to check. The syntax is as follow:

du <options> <location of directory or file>

Let’s take a look at real-world use of the **du** command with the Desktop directory:

* **du /home/user/Desktop** — this command line allows users to see into the disk usage of their Desktop folders and files (subdirectories are included as well).
* **du -h /home/user/Desktop** — just like with **df**, the option **-h** displays information in a **human-readable** format.
* **du -sh /home/user/Desktop** — the **-s** option will give us the total size of a specified folder (Desktop in this case).
* **du -m /home/user/Desktop** — the **-m** option provides us with folder and file sizes in **Megabytes** (we can use **-k** to see the information in **Kilobytes**).
* **du -h — time /home/user/Desktop** — this informs the last modification date of the displayed folders and files.
* **df –help** — it displays a list of available options and what they can be used

Device files,

Device files allow user programs to access hardware devices on the system through the kernel. They are not "files" per se, but look like files from the program's point of view: you can read from them, write to them, mmap() onto them, and so forth.

When you access such a device "file," the kernel recognizes the I/O request and passes it a device driver, which performs some operation, such as reading data from a serial port, or sending data to a sound card.

Device files are located in the directory /dev on nearly all Unix-like systems. Each device on the system should have a corresponding entry in /dev. For example, /dev/ttyS0 corresponds to the first serial port, known as COM1 under MS-DOS; /dev/hda2 corresponds to the second partition on the first IDE drive. In fact, there should be entries in /dev for devices you do not have. The device files are generally created during system installation and include every possible device driver. They don't necessarily correspond to the actual hardware on your system.

Cpio:

cpio stands for “copy in, copy out“. It is used for processing the archive files like *\*.cpio* or *\*.tar*. This command can copy files to and from archives.

Synopsis:

* Copy-out Mode: Copy files named in name-list to the archive

Syntax:

cpio -o < name-list > archive

* Copy-in Mode: Extract files from the archive

Syntax:

* cpio -i < archive
* Copy-pass Mode: Copy files named in name-list to destination-directory

Syntax:

cpio -p destination-directory < name-list

Policy Options:

* -i, –extract: Extract files from an archive and it runs only in copy-in mode.
* -o, –create: Create the archive and it runs only in copy-out mode.
* -p, –pass-through: Run in copy-pass mode.
* -t, –list: Print a table of contents of all the inputs present.

tar commands :

The Linux ‘tar’ stands for tape archive, is used to create Archive and extract the Archive files. tar command in Linux is one of the important command which provides archiving functionality in Linux. We can use Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them.

**Syntax:**

**tar [options] [archive-file] [file or directory to be archived]**

**Options:**   
**-c :** Creates Archive   
**-x :** Extract the archive   
**-f :** creates archive with given filename   
**-t :** displays or lists files in archived file   
**-u :** archives and adds to an existing archive file   
**-v :** Displays Verbose Information   
**-A :** Concatenates the archive files   
**-z :** zip, tells tar command that creates tar file using gzip   
**-j :** filter archive tar file using tbzip   
**-W :** Verify a archive file   
**-r :** update or add file or directory in already existed .tar file

**What is an Archive file?**   
An Archive file is a file that is composed of one or more files along with metadata. Archive files are used to collect multiple data files together into a single file for easier portability and storage, or simply to compress files to use less storage space.

Advanced Administration:

Partition and file system:

## What is a partition in Linux?

A partition is a segment of a storage device that has been logically separated from the rest of the device. All the partitions can be managed as if they are separate storage devices. To store data on a device you need to have at least 1 partition on the device.

Having multiple partitions on the device helps the device store data more efficiently. This is no longer the case with modern filesystems and storage devices. A good analogy might be that a storage device is similar to an empty plot of land. And to store your items in that empty plot of land, you need to build rooms aka partitions.

## What is a filesystem in Linux?

A filesystem is how we manage data in each partition. It is responsible for indexing, storing, retrieving, naming the files, and maintaining metadata (file owner, size, permissions, etc.) of the files. stored in a partition.

A file is kept in multiple continuous sectors, each sector being around 4096 bytes in modern times. It is the job of the file system to recognize which sectors are ready to be used, which sector a file must be stored in, and which sector contains which file. Without this organization, it would be impossible to retrieve any files because the system doesn’t know the location(block) of the file.

How filesystems store this metadata differs from filesystem to filesystem. For example, while FAT maintains a table for each directory/folder, NTFS has a Master File Table that holds a record of metadata for each file contained by the filesystem including the table itself.

Fdisk:-

**fdisk** stands (for “**fixed disk** or **format disk**“) is an most commonly used command-line based disk manipulation utility for a **Linux/Unix** systems. With the help of fdisk command you can view, create, resize, delete, change, copy and move partitions on a hard drive using its own user friendly text based menu driven interface.

This tool is very useful in terms of creating space for new partitions, organising space for new drives, re-organising an old drives and copying or moving data to new disks. It allows you to create a maximum of four new **primary** partition and number of logical (**extended**) partitions, based on size of the hard disk you have in your system.

Mkfs:-

The **mkfs** command stands for**“make file system”** is utilized to make a file system (which is, a system for organizing a hierarchy of directories, subdirectories, and files) on a formatted storage device usually, a partition on a hard disk drive (HDD) or it can also be a USB drive, etc. A partition is logically an autonomous part of an HDD. An organized segment is one to which a low-level arrangement or format, additionally called a physical format(organization), has been applied. It comprises separating the disk’s tracks into a predetermined number of divisions and filling the information zone of every segment with dummy bytes.

Fsck:-

The fsck command is used to check the integrity of a file system. File system integrity refers to the correctness and validity of a file system. Most systems automatically run the fsck command at boot time so that errors, if any, are detected and corrected before the system is used. File system errors are usually caused by power failures, hardware failures, or improper shutdown of the system.

You should unmount the file system before scanning it with fsck to prevent damage to the file system.

### Syntax

The syntax of the fsck command is:

# fsck [options] {device/file system name}

shutdown and sync operation :

## The Linux shutdown Command

The version of shutdown found on most Linux systems also has a -t option which may be used to specify the delay period between when the kernel sends the TERM signal to all remaining processes on the system and when it sends the KILL signal. The default is 30 seconds. The following command shuts down the system more rapidly, allowing only 5 seconds between the two signals:

# shutdown -h -t 5 now

The command version also provides a -a option, which provides a limited security mechanism for the shutdown command. When it is invoked with this option, the command determines whether any of the users listed in the file /etc/shutdown.allow are currently logged in on the console (or any virtual console attached to it). If not, the shutdown command fails.

The purpose of this option is to prevent casual passers-by from typing Ctrl-Alt-Delete on the console and causing an (unwanted) system reboot. Accordingly, it is most often used in the inittab entry corresponding to this event.

## Ensuring Disk Accuracy with the sync Command

As we’ve noted previously, one of the important parts of the shutdown process is syncing the disks. The sync command finishes all disk transactions and writes out all data to disk, guaranteeing that the system can be turned off without corrupting the files. You can execute this command manually if necessary:

# sync

# sync

Why is sync executed two or three times (or even more[[15](https://www.oreilly.com/library/view/essential-system-administration/0596003439/ch04s03.html#ftn.ch04-FTNOTE-15)])? I think this is a bit of Unix superstition. The sync command schedules but does not necessarily immediately perform the required disk writes, even though the Unix prompt returns immediately. Multiple sync commands raise the probability that the write will take place before you enter another command (or turn off the power) by taking up the time needed to complete the operation. However, the same effect can be obtained by waiting a few seconds for disk activity to cease before doing anything else. Typing “sync” several times gives you something to do while you’re waiting.

There is one situation in which you do not want sync to be executed, either manually or automatically: when you have run fsck manually on the root filesystem. If you sync the disks at this point, you will rewrite the bad superblocks stored in the kernel buffers and undo the fixing fsck just did. In such cases, on BSD-based systems and under HP-UX, you must use the -n option to reboot or shutdown to suppress the usual automatic sync operation.

Mounting and Unmounting of File systems:

* The Linux file system forms a hierarchical structure *tree* rooted at */*.
* This root */* is mounted during startup, other file systems remain unusable until they are mounted at a mount point. For access to a file system, it needs to be mounted.
* Example: A case on point where we would need mounting is whereby assuming we are running a very popular web service on a 500 GB HDD but data grows very fast and we need to increase the storage space, mounting enables us to mount a new larger-capacity storage device at any point in a directory while maintaining the same file structure.
* Mounting and unmounting file systems requires root privileges on the system and the existence of the directory in which we will mount the file system in the case of mounting
* Mounting:-
* This is the process of associating a file system to a storage device. In Linux the *mount* command is used so as to attach a file system to the file system hierarchy. When mounting we provide information such as files system type, file system and the mount point.
* By mounting a file system onto the file hierarchy, a file system becomes a subtree of the hierarchy and thus it is possible to navigate from the rest for the file hierarchy.
* Mounting can also be done from one system to another by use of a networked file system e.g NFS or AFS.
* We can also opt to create a file in another existing file system and format it as a file system and mount it e.g downloading an ISO file and mounting it rather than copying it to another media.
* A mount point is a directory created as part of the file system e.g the root file system is mounted in the */* directory.
* Mounting of file systems happens during startup and it is managed by the /etc/fstab (file system table) configuration file.
* This file will have a list of all file systems, their selected mount points and other file system specific options.
* To view the fstab file type *cat /etc/fstab* command.
* From the output we can see columns such as,
* *Filesystem*: we can either specifies the file system by UUID(universal unique identifier) or a disk label.
* *Mountpoint*: is the directory on the file system we shall use to access stored data on the disk.
* *Type*: This specifies the type of the file system.
* *Options*: Here we can specify options for tuning a mount.
* *Dump*: We can either enable or disable dumping on the system.
* *Pass num*: Here we set the order user so that *fsck* can check the file system.
* We can view all mounted file systems by using the *mount* command.
* $mount
* Unmounting:-
* All file systems are normally unmounted when the system powers down and any cached data stored in memory is flushed to the device.
* Unmounting can be done manually e.g when removing writable media such as USBs.  
  The *unmount* command is used for this operation as follows,
* $unmount [directory]
* Or
* $unmounts [device]
* After unmounting files in the directory used as mount points, the hidden files if they existed will become visible again.
* Errors can occur when we try to unmount a file system while there are processes with its files open.