X Window System

The X Window System (X11) is a windowing system for bitmap displays, common on UNIX-like computer operating systems. X provides the basic framework for a GUI environment: drawing and moving windows on the display device and interacting with a mouse and keyboard. X does not mandate the user interface — this is handled by individual programs.

X is an architecture-independent system for remote graphical user interfaces and input device capabilities. Each person using a networked terminal has the ability to interact with the display with any type of user input device.

X provides the basic framework, or primitives, for building such GUI environments: drawing and moving windows on the display and interacting with a mouse, keyboard or touch screen. X does not mandate the user interface; individual client programs handle this. Programs may use X's graphical abilities with no user interface. As such, the visual styling of X-based environments varies greatly; different programs may present radically different interfaces.

Unlike earlier display protocols, X was specifically designed to be used over network connections rather than on an integral or attached display device. X features network transparency: the machine where an application program (the client application) runs can differ from the user's local machine (the display server). X's network protocol is based on X command primitives. This approach allows both 2D and 3D operations to be fully accelerated on the remote X server.

X provides no native support for audio; several projects exist to fill this niche, some also providing transparent network support.

Software architecture of X Window System

X uses a client–server model: an X server communicates with various client programs. The server accepts requests for graphical output (windows) and sends back user input (from keyboard, mouse, or touch screen). The server may function as:

- an application displaying to a window of another display system
- a system program controlling the video output of a PC
- a dedicated piece of hardware.

This client–server terminology—the user's terminal being the server and the applications being the clients—often confuses new X users, because the terms appear reversed. But X takes the perspective of the application, rather than that of the end-user: X provides display and I/O services to applications, so it is a server; applications use these services, thus they are clients.

The communication protocol between server and client operates network-transparently: the client and server may run on the same machine or on different ones, possibly with different architectures and operating systems. A client and server can even communicate securely over the Internet by tunnelling the connection over an encrypted network session.

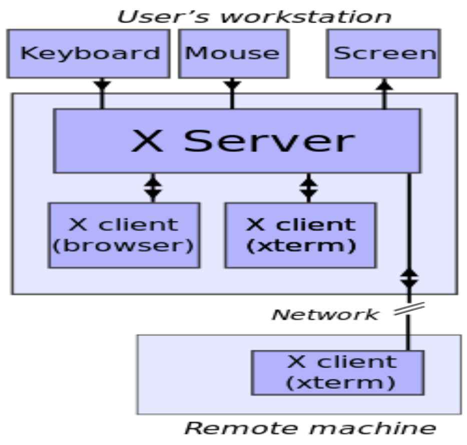
An X client itself may emulate an X server by providing display services to other clients. This is known as "X nesting". Open-source clients such as Xnest and Xephyr support such X nesting.

To use an X client application on a remote machine, the user may do the following:

- On the local machine, open a terminal window
- Use ssh with the X forwarding argument to connect to the remote machine.
- Request local display/input service (e.g., export DISPLAY=[user's machine]:0 if not using SSH with X forwarding enabled).

The remote X client application will then make a connection to the user's local X server, providing display and input to the user.

Alternatively, the local machine may run a small program that connects to the remote machine and starts the client application.



Simple example: the X server receives input from a local keyboard and mouse and displays to a screen. A web browser and a emulator run on the user's workstation and a terminal emulator runs on a remote computer but is controlled and monitored from the user's machine

X11 - X windowing system

X11 is a network protocol designed for UNIX and similar operating systems to enable remote graphical access to applications. The original X windowing system was announced in 1984 and developed at MIT.

A machine running an X windowing system can launch a program on a remote computer. All the CPU processing happens on the remote computer but the display of the application appears on the local machine.

For a time it was popular to have dedicated "X terminals". Similar to a character cell terminal these terminals had no "brains" except for what was needed to operate their X windowing system. Such terminals started to disappear as personal desktop computers became popular, more powerful, and inexpensive enough to run an X windowing system on top of the installed OS (or to have the applications ported to run locally on the personal computer). Interestingly, today the popularity of similar terminals is slowly picking up again as large businesses realize the need for easily maintainable, interchangeable "thin clients".

Although X terminals really did not catch on, the X windowing system did become the standard graphical system for graphical programs running in UNIX and Linux environments. These systems use the X11 protocol to draw graphics to their local video display. The local display is treated as a remote display that just happens to be on the same machine.

Disk Quotas

Disk space can be restricted by implementing disk quotas which alert a system administrator before a user consumes too much disk space or a partition becomes full.

Disk quotas can be configured for individual users as well as user groups. This makes it possible to manage the space allocated for user-specific files (such as email) separately from the space allocated to the projects a user works on (assuming the projects are given their own groups).

In addition, quotas can be set not just to control the number of disk blocks consumed but to control the number of inodes (data structures that contain information about files in UNIX file systems). Because inodes are used to contain file-related information, this allows control over the number of files that can be created.

The **quota** RPM must be installed to implement disk quotas.

Types of quotas

There are two basic types of disk quotas. The first, known as a *usage quota* or *block quota*, limits the amount of disk space that can be used. The second, known as a *file quota*or *inode quota*, limits the number of files and directories that can be created.

In addition, administrators usually define a warning level, or *soft quota*, at which users are informed they are nearing their limit that is less than the effective limit, or *hard quota*. There may also be a small *grace interval*, which allows users to temporarily violate their quotas by certain amounts if necessary.

1. Configuring Disk Quotas

To implement disk quotas, use the following steps:

1. Enable quotas per file system by modifying the **/etc/fstab** file.
2. Remount the file system(s).
3. Create the quota database files and generate the disk usage table.
4. Assign quota policies.

Each of these steps is discussed in detail in the following sections.

1.1. Enabling Quotas

As root, using a text editor, edit the **/etc/fstab** file. Add the **usrquota** and/or **grpquota** options to the file systems that require quotas:

```
/dev/VolGroup00/LogVol00 /        ext3    defaults        1 1
LABEL=/boot              /boot    ext3    defaults        1 2
none                     /dev/pts  devpts  gid=5,mode=620  0 0
none                     /dev/shm  tmpfs   defaults        0 0
none                     /proc     proc    defaults        0 0
none                     /sys      sysfs   defaults        0 0
/dev/VolGroup00/LogVol02 /home     ext3    defaults,usrquota,grpquota  1 2
/dev/VolGroup00/LogVol01 swap      swap    defaults        0 0 . . .
```

In this example, the **/home** file system has both user and group quotas enabled.

1.2. Remounting the File Systems

After adding the **usrquota** and/or **grpquota** options, remount each file system whose **fstab** entry has been modified. If the file system is not in use by any process, use one of the following methods:

- Issue the **umount** command followed by the **mount** command to remount the file system.(See the **man** page for both **umount** and **mount** for the specific syntax for mounting and unmounting various filesystem types.)
- Issue the **mount -o remount <*file-system*>** command (where **<*file-system*>** is the name of the file system) to remount the file system. For example, to remount the **/home** file system, the command to issue is **mount -o remount /home**.

If the file system is currently in use, the easiest method for remounting the file system is to reboot the system.

1.3. Creating the Quota Database Files

After each quota-enabled file system is remounted, the system is capable of working with disk quotas. However, the file system itself is not yet ready to support quotas. The next step is to run the **quotacheck**command.

The **quotacheck** command examines quota-enabled file systems and builds a table of the current disk usage per file system. The table is then used to update the operating system's copy of disk usage. In addition, the file system's disk quota files are updated.

To create the quota files (**aquota.user** and **aquota.group**) on the file system, use the **-c** option of the**quotacheck** command. For example, if user and group quotas are enabled for the **/home** file system, create the files in the **/home** directory:

```
quotacheck -cug /home
```

The **-c** option specifies that the quota files should be created for each file system with quotas enabled, the **-u**option specifies to check for user quotas, and the **-g** option specifies to check for group quotas.

If neither the **-u** or **-g** options are specified, only the user quota file is created. If only **-g** is specified, only the group quota file is created.

After the files are created, run the following command to generate the table of current disk usage per file system with quotas enabled:

```
quotacheck -avug
```

The options used are as follows:

- **a** — Check all quota-enabled, locally-mounted file systems
- **v** — Display verbose status information as the quota check proceeds
- **u** — Check user disk quota information
- **g** — Check group disk quota information

After **quotacheck** has finished running, the quota files corresponding to the enabled quotas (user and/or group) are populated with data for each quota-enabled locally-mounted file system such as **/home**.

1.4. Assigning Quotas per User

The last step is assigning the disk quotas with the **edquota** command.

To configure the quota for a user, as root in a shell prompt, execute the command:

```
edquota username
```

Perform this step for each user who needs a quota. For example, if a quota is enabled in **/etc/fstab** for the**/home** partition (**/dev/VolGroup00/LogVol02** in the example below) and the command **edquota testuser** is executed, the following is shown in the editor configured as the default for the system:

```
Disk quotas for user testuser (uid 501):
Filesystem              blocks    soft    hard    inodes   soft   hard
/dev/VolGroup00/LogVol02  440436      0       0    37418      0      0
```

The first column is the name of the file system that has a quota enabled for it. The second column shows how many blocks the user is currently using. The next two columns are used to set soft and hard block limits for the user on the file system. The **inodes** column shows how many inodes the user is currently using. The last two columns are used to set the soft and hard inode limits for the user on the file system.

The hard block limit is the absolute maximum amount of disk space that a user or group can use. Once this limit is reached, no further disk space can be used.

The soft block limit defines the maximum amount of disk space that can be used. However, unlike the hard limit, the soft limit can be exceeded for a certain amount of time. That time is known as the *grace period*. The grace period can be expressed in seconds, minutes, hours, days, weeks, or months.

If any of the values are set to 0, that limit is not set. In the text editor, change the desired limits. For example:

```
Disk quotas for user testuser (uid 501):   Filesystem
blocks    soft    hard   inodes   soft   hard   /dev/VolGroup00/LogVol02  440436   500000
550000    37418     0      0
```

To verify that the quota for the user has been set, use the command:

```
quota testuser
```

1.5. Assigning Quotas per Group

Quotas can also be assigned on a per-group basis. For example, to set a group quota for the **devel** group (the group must exist prior to setting the group quota), use the command:

```
edquota -g devel
```

This command displays the existing quota for the group in the text editor:

```
Disk quotas for group devel (gid 505):
Filesystem               blocks    soft    hard   inodes   soft   hard
/dev/VolGroup00/LogVol02 440400      0       0    37418      0      0
```

Modify the limits, and then save the file.

To verify that the group quota has been set, use the command:

```
quota -g devel
```

2. Managing Disk Quotas

If quotas are implemented, they need some maintenance — mostly in the form of watching to see if the quotas are exceeded and making sure the quotas are accurate.

Of course, if users repeatedly exceed their quotas or consistently reach their soft limits, a system administrator has a few choices to make depending on what type of users they are and how much disk space impacts their work. The administrator can either help the user determine how to use less disk space or increase the user's disk quota.

2.1. Enabling and Disabling

It is possible to disable quotas without setting them to 0. To turn all user and group quotas off, use the following command:

```
quotaoff -vaug
```

If neither the **-u** or **-g** options are specified, only the user quotas are disabled. If only **-g** is specified, only group quotas are disabled. The **-v** switch causes verbose status information to display as the command executes.

To enable quotas again, use the **quotaon** command with the same options.

For example, to enable user and group quotas for all file systems, use the following command:

```
quotaon -vaug
```

To enable quotas for a specific file system, such as **/home**, use the following command:

```
quotaon -vug /home
```

If neither the **-u** or **-g** options are specified, only the user quotas are enabled. If only **-g** is specified, only group quotas are enabled.

2.2. Reporting on Disk Quotas

Creating a disk usage report entails running the **repquota** utility. For example, the command **repquota /home**produces this output:

```
*** Report for user quotas on device /dev/mapper/VolGroup00-LogVol02 Block grace
time: 7days; Inode grace time: 7days                    Block limits              File limits User
used    soft    hard  grace    used  soft  hard  grace -----------------------------------------------
--------------------- root      --      36       0       0              4     0     0 kristin   --      540       0       0
125     0     0 testuser  --  440400  500000  550000         37418     0     0
```

To view the disk usage report for all (option **-a**) quota-enabled file systems, use the command:

```
repquota -a
```

While the report is easy to read, a few points should be explained. The **--** displayed after each user is a quick way to determine whether the block or inode limits have been exceeded. If either soft limit is exceeded, a **+**appears in place of the corresponding **-**; the first **-** represents the block limit, and the second represents the inode limit.

The **grace** columns are normally blank. If a soft limit has been exceeded, the column contains a time specification equal to the amount of time remaining on the grace period. If the grace period has expired, **none**appears in its place.

2.3. Keeping Quotas Accurate

Whenever a file system is not unmounted cleanly (due to a system crash, for example), it is necessary to run**quotacheck**. However, **quotacheck** can be run on a regular basis, even if the system has not crashed. Running the following command periodically keeps the quotas more accurate (the options used have been described inSection 7.1.1, "Enabling Quotas"):

```
quotacheck -avug
```

The easiest way to run it periodically is to use **cron**. As root, either use the **crontab -e** command to schedule a periodic **quotacheck** or place a script that runs **quotacheck** in any one of the following directories (using whichever interval best matches your needs):

- **/etc/cron.hourly**
- **/etc/cron.daily**
- **/etc/cron.weekly**
- **/etc/cron.monthly**

The most accurate quota statistics can be obtained when the file system(s) analyzed are not in active use. Thus, the cron task should be schedule during a time where the file system(s) are used the least. If this time is various for different file systems with quotas, run **quotacheck** for each file system at different times with multiple cron tasks.

Users and Groups in Linux

The control of *users* and *groups* is a core element of Red Hat Enterprise Linux system administration.

Users can be either people (meaning accounts tied to physical users) or accounts which exist for specific applications to use.

Groups are logical expressions of organization, tying users together for a common purpose. Users within a group can read, write, or execute files owned by that group.

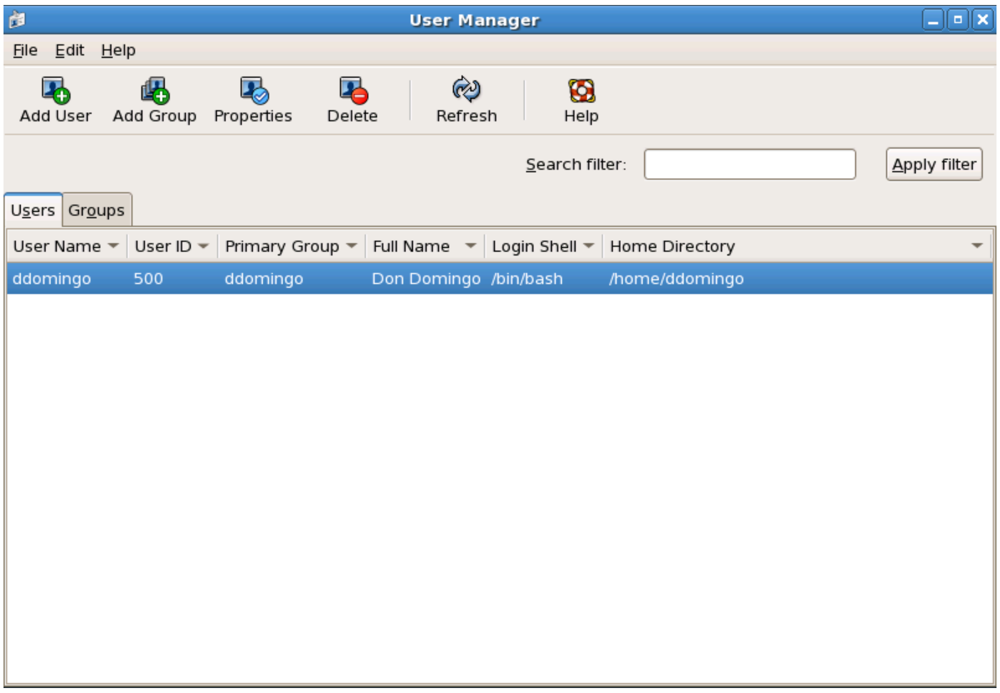
Each user and group has a unique numerical identification number called a *userid* (*UID*) and a *groupid* (*GID*), respectively.

A user who creates a file is also the owner and group owner of that file. The file is assigned separate read, write, and execute permissions for the owner, the group, and everyone else. The file owner can be changed only by the root user, and access permissions can be changed by both the root user and file owner.

1. User and Group Configuration

The **User Manager** allows you to view, modify, add, and delete local users and groups.

To use the **User Manager**, you must be running the X Window System, have root privileges, and have the**system-config-users** RPM package installed. To start the **User Manager** from the desktop, go to **System** (on the panel) => **Administration** => **Users & Groups**. You can also type the command **system-config-users**at a shell prompt (for example, in an XTerm or a GNOME terminal).



User Name	User ID	Primary Group	Full Name	Login Shell	Home Directory
ddomingo	500	ddomingo	Don Domingo	/bin/bash	/home/ddomingo

Figure 1. User Manager

To view a list of local users on the system, click the Users tab. To view a list of local groups on the system, click the Groups tab.

To find a specific user or group, type the first few letters of the name in the Search filter field. Press Enter or click the Apply filter button. The filtered list is displayed.

To sort the users or groups, click on the column name. The users or groups are sorted according to the value of that column.

Red Hat Enterprise Linux reserves user IDs below 500 for system users. By default, **User Manager** does not display system users. To view all users, including the system users, go to **Edit** => **Preferences** and uncheck**Hide system users and groups** from the dialog box.

1.1. Adding a New User

To add a new user, click the Add User button. A window as shown in 2, "New User" appears. Type the username and full name for the new user in the appropriate fields. Type the user's password in thePassword and Confirm Password fields. The password must be at least six characters.

Note
It is advisable to use a much longer password, as this makes it more difficult for an intruder to guess it and access the account without permission. It is also recommended that the password not be based on a dictionary term; use a combination of letters, numbers and special characters.

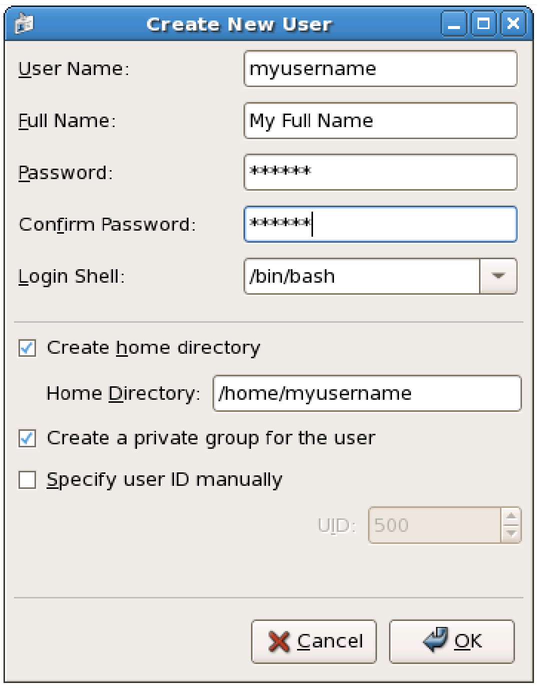
Select a login shell. If you are not sure which shell to select, accept the default value of **/bin/bash**. The default home directory is **/home/<*username*>/**. You can change the home directory that is created for the user, or you can choose not to create the home directory by unselecting Create home directory.

If you select to create the home directory, default configuration files are copied from the **/etc/skel/** directory into the new home directory.

Red Hat Enterprise Linux uses a *user private group* (UPG) scheme. The UPG scheme does not add or change anything in the standard UNIX way of handling groups; it offers a new convention. Whenever you create a new user, by default, a unique group with the same name as the user is created. If you do not want to create this group, unselect Create a private group for the user.

To specify a user ID for the user, select Specify user ID manually. If the option is not selected, the next available user ID above 500 is assigned to the new user. Because Red Hat Enterprise Linux reserves user IDs below 500 for system users, it is not advisable to manually assign user IDs 1-499.

Click OK to create the user.


Create New User

User Name:	myusername
Full Name:	My Full Name
Password:	******
Confirm Password:	******
Login Shell:	/bin/bash

☑ Create home directory

Home Directory: /home/myusername

☑ Create a private group for the user

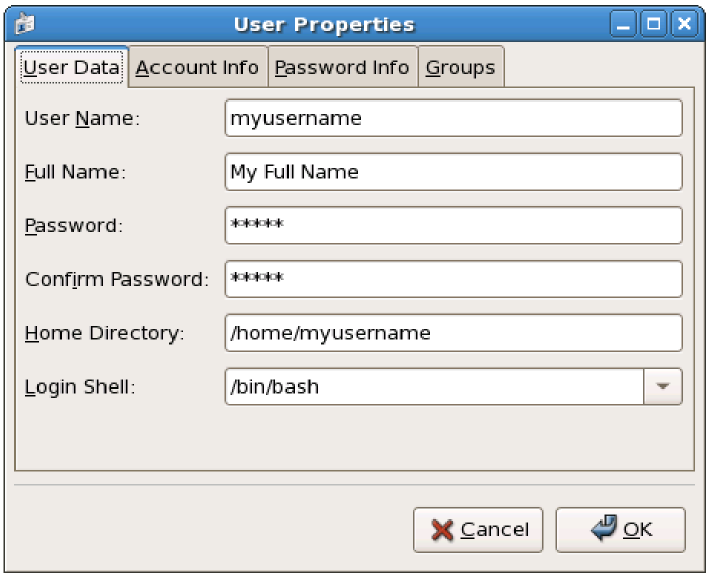
☐ Specify user ID manually

UID: 500

Figure2. New User

1.2. Modifying User Properties

To view the properties of an existing user, click on the Users tab, select the user from the user list, and clickProperties from the menu (or choose **File** => **Properties** from the pulldown menu). A window similar toFigure 3, "User Properties" appears.


User Properties

User Name:	myusername
Full Name:	My Full Name
Password:	*****
Confirm Password:	*****
Home Directory:	/home/myusername
Login Shell:	/bin/bash

Figure 3. User Properties

The User Properties window is divided into multiple tabbed pages:

- User Data — Shows the basic user information configured when you added the user. Use this tab to change the user's full name, password, home directory, or login shell.
- Account Info — Select Enable account expiration if you want the account to expire on a certain date. Enter the date in the provided fields. Select Local password is locked to lock the user account and prevent the user from logging into the system.
- Password Info — Displays the date that the user's password last changed. To force the user to change passwords after a certain number of days, select Enable password expiration and enter a desired value in theDays before change required: field. The number of days before the user's password expires, the number of days before the user is warned to change passwords, and days before the account becomes inactive can also be changed.
- Groups — Allows you to view and configure the Primary Group of the user, as well as other groups that you want the user to be a member of.

1.3. Adding a New Group

To add a new user group, click the Add Group button. A window similar to Figure 4, "New Group" appears. Type the name of the new group to create. To specify a group ID for the new group, selectSpecify group ID manually and select the GID. Note that Red Hat Enterprise Linux also reserves group IDs lower than 500 for system groups.

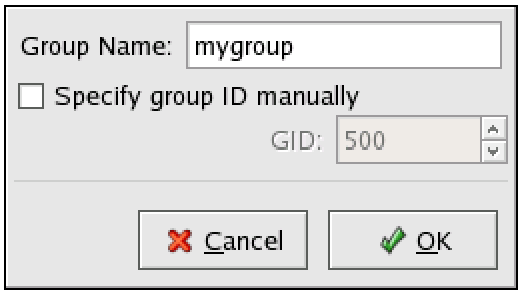

Group Name: mygroup
☐ Specify group ID manually
GID: 500

Figure 4. New Group

Click OK to create the group. The new group appears in the group list.

1.4. Modifying Group Properties

To view the properties of an existing group, select the group from the group list and click Properties from the menu (or choose **File** => **Properties** from the pulldown menu). A window similar to Figure 5, "Group Properties" appears.

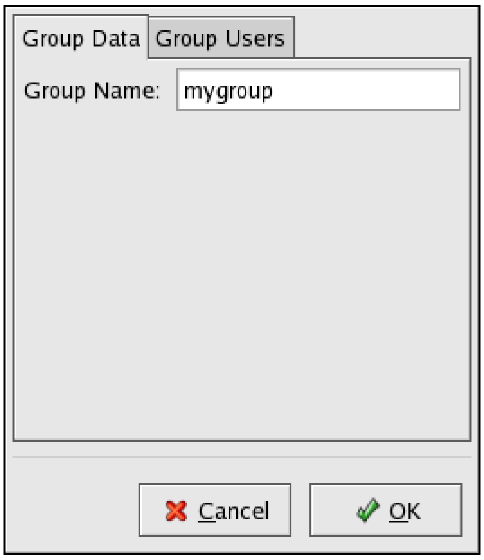


Figure 5. Group Properties

The Group Users tab displays which users are members of the group. Use this tab to add or remove users from the group. Click OK to save your changes.

2. User and Group Management Tools

Managing users and groups can be a tedious task; this is why Red Hat Enterprise Linux provides tools and conventions to make them easier to manage.

There are three types of accounts on a Linux system:

1. **Root account:** This is also called superuser and would have complete and unfettered control of the system. A superuser can run any commands without any restriction. This user should be assumed as a system administrator.

2. **System accounts:** System accounts are those needed for the operation of system-specific components for example mail accounts and the sshd accounts. These accounts are usually needed for some specific function on your system, and any modifications to them could adversely affect the system.

3. **User accounts:** User accounts provide interactive access to the system for users and groups of users. General users are typically assigned to these accounts and usually have limited access to critical system files and directories.

Linux supports a concept of *Group Account* which logically groups a number of accounts. Every account would be a part of any group account. Unix groups plays important role in handling file permissions and process management.

Managing Users and Groups:

There are three main user administration files:

1. **/etc/passwd:** Keeps user account and password information. This file holds the majority of information about accounts on the Unix system.
2. **/etc/shadow:** Holds the encrypted password of the corresponding account. Not all the system support this file.
3. **/etc/group:** This file contains the group information for each account.
4. **/etc/gshadow:** This file contains secure group account information.
   Check all the above files using **cat** command.

Following are commands available on the majority of Unix systems to create and manage accounts and groups:

Command	**Description**
useradd	Adds accounts to the system.
usermod	Modifies account attributes.
userdel	Deletes accounts from the system.
groupadd	Adds groups to the system.
groupmod	Modifies group attributes.
groupdel	Removes groups from the system.

Create a Group

You would need to create groups before creating any account otherwise you would have to use existing groups at your system. You would have all the groups listed in */etc/groups* file.

All the default groups would be system account specific groups and it is not recommended to use them for ordinary accounts. So following is the syntax to create a new group account:

```
groupadd [-g gid [-o]] [-r] [-f] groupname
```

Here is the detail of the parameters:

Option	**Description**

-g GID	The numerical value of the group's ID.
-o	This option permits to add group with non-unique GID
-r	This flag instructs groupadd to add a system account
-f	This option causes to just exit with success status if the specified group already exists. With -g, if specified GID already exists, other (unique) GID is chosen
groupname	Actaul group name to be created.

If you do not specify any parameter then system would use default values.

Following example would create *developers* group with default values, which is very much acceptable for most of the administrators.

```
$ groupadd developers
```

Modify a Group:

To modify a group, use the **groupmod** syntax:

```
$ groupmod -n new_modified_group_name old_group_name
```

To change the developers_2 group name to developer, type:

```
$ groupmod -n developer developer_2
```

Here is how you would change the financial GID to 545:

```
$ groupmod -g 545 developer
```

Delete a Group:

To delete an existing group, all you need are the groupdel command and the group name. To delete the financial group, the command is:

```
$ groupdel developer
```

This removes only the group, not any files associated with that group. The files are still accessible by their owners.

Create an Account

Let us see how to create a new account on your Linux system. Following is the syntax to create a user's account:

```
useradd -d homedir -g groupname -m -s shell -u userid accountname
```

Here is the detail of the parameters:

Option	Description
-d homedir	Specifies home directory for the account.
-g groupname	Specifies a group account for this account.
-m	Creates the home directory if it doesn't exist.
-s shell	Specifies the default shell for this account.
-u userid	You can specify a user id for this account.
accountname	Actual account name to be created

If you do not specify any parameter then system would use default values. The useradd command modifies the /etc/passwd, /etc/shadow, and /etc/group files and creates a home directory.

Following is the example which would create an account *mcmohd* setting its home directory to*/home/mcmohd* and group as *developers*. This user would have Korn Shell assigned to it.

```
$ useradd -d /home/mcmohd -g developers -s /bin/ksh mcmohd
```

Before issuing above command, make sure you already have *developers* group created using *groupadd* command.
Once an account is created you can set its password using the **passwd** command as follows:

```
$ passwd mcmohd20

Changing password for user mcmohd20.

New UNIX password:

Retype new UNIX password:

passwd: all authentication tokens updated successfully.
```

When you type *passwd accountname*, it gives you option to change the password provided you are super user otherwise you would be able to change just your password using the same command but without specifying your account name.

Modify an Account:

The **usermod** command enables you to make changes to an existing account from the command line. It uses the same arguments as the useradd command, plus the -l argument, which allows you to change the account name.
For example, to change the account name *mcmohd* to *mcmohd20* and to change home directory accordingly, you would need to issue following command:

```
$ usermod -d /home/mcmohd20 -m -l mcmohd mcmohd20
```

Delete an Account:

The **userdel** command can be used to delete an existing user. This is a very dangerous command if not used with caution.

There is only one argument or option available for the command: .r, for removing the account's home directory and mail file.

For example, to remove account *mcmohd20*, you would need to issue following command:

```
$ userdel -r mcmohd20
```

If you want to keep her home directory for backup purposes, omit the -r option. You can remove the home directory as needed at a later time.

User Private Groups

Red Hat Enterprise Linux uses a *user private group* (*UPG*) scheme, which makes UNIX groups easier to manage. A UPG is created whenever a new user is added to the system. A UPG has the same name as the user for which it was created and that user is the only member of the UPG.

UPGs make it safe to set default permissions for a newly created file or directory, allowing both the user and *the group of that user* to make modifications to the file or directory.

The setting which determines what permissions are applied to a newly created file or directory is called a*umask* and is configured in the **/etc/bashrc** file. Traditionally on UNIX systems, the **umask** is set to **022**, which allows only the user who created the file or directory to make modifications. Under this scheme, all other users, *including members of the creator's group*, are not allowed to make any modifications. However, under the UPG scheme, this "group protection" is not necessary since every user has their own private group.

1. Group Directories

Many IT organizations like to create a group for each major project and then assign people to the group if they need to access that project's files. Using this traditional scheme, managing files has been difficult; when someone creates a file, it is associated with the primary group to which they belong. When a single person works on multiple projects, it is difficult to associate the right files with the right group. Using the UPG scheme, however, groups are automatically assigned to files created within a directory with the *setgid* bit set. The setgid bit makes managing group projects that share a common directory very simple because any files a user creates within the directory are owned by the group which owns the directory.

Let us say, for example, that a group of people need to work on files in the **/usr/share/emacs/site-lisp/**directory. Some people are trusted to modify the directory, but certainly not everyone is trusted. First create an **emacs** group, as in the following command:

```
/usr/sbin/groupadd emacs
```

To associate the contents of the directory with the **emacs** group, type:

```
chown -R root.emacs /usr/share/emacs/site-lisp
```

Now, it is possible to add the proper users to the group with the **gpasswd** command:

```
/usr/bin/gpasswd -a <username> emacs
```

To allow users to create files within the directory, use the following command:

```
chmod 775 /usr/share/emacs/site-lisp
```

When a user creates a new file, it is assigned the group of the user's default private group. Next, set the setgid bit, which assigns everything created in the directory the same group permission as the directory itself (**emacs**). Use the following command:

```
chmod 2775 /usr/share/emacs/site-lisp
```

At this point, because the default umask of each user is 002, all members of the **emacs** group can create and edit files in the **/usr/share/emacs/site-lisp/** directory without the administrator having to change file permissions every time users write new files.

Shadow Passwords

In multiuser environments it is very important to use *shadow passwords* (provided by the **shadow-utils**package). Doing so enhances the security of system authentication files. For this reason, the installation program enables shadow passwords by default.

The following lists the advantages pf shadow passwords have over the traditional way of storing passwords on UNIX-based systems:

- Improves system security by moving encrypted password hashes from the world-readable **/etc/passwd** file to **/etc/shadow**, which is readable only by the root user.
- Stores information about password aging.
- Allows the use the **/etc/login.defs** file to enforce security policies.

Most utilities provided by the **shadow-utils** package work properly whether or not shadow passwords are enabled. However, since password aging information is stored exclusively in the **/etc/shadow** file, any commands which create or modify password aging information do not work.

The following is a list of commands which do not work without first enabling shadow passwords:

- **chage**
- **gpasswd**
- **/usr/sbin/usermod -e** or **-f** options
- **/usr/sbin/useradd -e** or **-f** options

Root Account

The "root" account is the most privileged account on a UNIX system. This account gives you the ability to carry out all facets of system administration, including adding accounts, changing user passwords, examining log files, installing software, etc. *root* is the user name or account that by default has access to all commands andfiles on a Linux or other Unix-like operating system. It is also referred to as the *root account*, *root user* and the *superuser*.

When using this account it is crucial to be as careful as possible. The "root" account has no security restrictions imposed upon it. This means it is easy to perform administrative duties without disturb. Therefore it is easy, with a mistyped command, to wipe out crucial system files.

When you are signed in as, or acting as "root", the shell prompt displays '#' as the last character (if you are using bash). This is to serve as a warning to you of the absolute power of this account.

The rule of thumb is, never sign in as "root" unless absolutely necessary. While "root", type commands carefully and double-check them before pressing return. Sign off from the "root" account as soon as you have accomplished the task you signed on for. Finally, (as with any account but especially important with this one), keep the password secure!

Root privileges are the powers that the root account has on the system. The root account is the most privileged on the system and has absolute power over it (i.e., complete access to all files and commands). Among root's powers are the ability to modify the system in any way desired and to grant and revoke *access permissions* (i.e., the ability to read, modify and execute specific files and directories) for other users, including any of those that are by default reserved for root.

The use of the term *root* for the all-powerful administrative user may have arisen from the fact that root is the only account having *write permissions* (i.e., permission to modify files) in the root directory. The root directory, in turn, takes its name from the fact that the *filesystems* (i.e., the entire hierarchy of directories that is used to organize files) in Unix-like operating systems have been designed with a tree-like (although inverted) structure in which all directories branch off from a single directory that is analogous to the root of a tree.

Every user account is automatically assigned an identification number, the UID(i.e., user ID), by a Unix-like system, and the system uses these numbers instead of the user names to identify and keep track of the users. Root always has a UID of zero. This can be verified by logging in as root (if using a home computer or other system that permits this operation) and running the *echo*command to display the UID of the current user, i.e.,

```
echo $UID
```

echo is used to repeat on the screen what is typed in after it. The dollar sign preceding UID tells echo to display its value rather than its name.

Root privileges are usually required for installing software in RPM (Red Hat Package Manager) package format because of the need to write to system directories. If an application program is being *compiled* (i.e., converted into runnable form) from *source code* (i.e., its original, human-readable form), however, it can usually be configured to install and run from a user's home directory. Root privileges are not needed by an ordinary user to compile and install software in its home directory. Compiling software as root should be avoided for security reasons.

Introduction to sudo

There are two ways to run administrative applications in Linux. You can either switch to the super user (root) with the su command, or you can take advantage of sudo. How you do this will depend upon which distribution you use.

Sudo stands for either "substitute user do" or "super user do". What sudo does is very important and crucial to many Linux distributions. Effectively, sudo allows a user to run a program as another user (most often the root user). There are many that think sudo is the best way to achieve "best practice security" on Linux. There are some, however, that feel quite the opposite. Regardless of where you stand, and what distribution you are currently using, there will come a time when you will have to take advantage of sudo.

Difference Between sudo and su

If you are accustomed to a more traditional Linux setup, then you are used to using the su command to gain root privileges. You can even issue the command su - to effectively log in as the root (root's home becomes your home). With these types of distributions you can also log in as the root user. To many (including myself) this is a bad idea. NEVER log in as the root user. If you are using a distribution that relies on su and allows root user log in, log in as your standard user and su to the root user.

Now with sudo-based distributions you will most likely notice that you can not log in as a root user. In fact, in distributions such as Ubuntu, the root user account has been "disabled." You cannot log in as root and you cannot su to become the root user. All you can do is issue commands with the help of sudo to gain administrative privileges.

Configuration

Sudo is VERY particular about syntax in the configuration file. So always double check your configurations before you save your file. Fortunately there is only one file you need to concern yourself with and that is /etc/sudoers. You may notice that, even in order to view the /etc/sudoers file you have to use the sudo command.

To make changes to the sudo configuration file you need to use a specific command - sudo visudo. When you open up this file you will notice that the sudoers file is fairly small in size. There really isn't much to it, but what there is to it is key. Let's take a look at how to add a user to the sudoers file.

The basic entry for a user looks like this:

user hostlist = (userlist) commandlist

Typically you will find an entry like this:

root ALL=(ALL) ALL

Which indicates that the user root on all hosts using any user can run all commands? Fairly straight-forward. But let's say you want to allow a single user access to one administrative command without having to enter a password. Let's use the command dpkg (not wise, but an easy means of illustration) and allow the user mary to issue those commands without having to issue a password. To do this you would add a line similar to this:

mary ALL = NOPASSWD: /usr/sbin/synaptic

to the **/etc/sudoers** file. Now the user mary can run synaptic by entering *sudo synaptic* but will not be prompted for a password. This is handy on a single-user system but should be used with caution. You do not want to allow just any command to be run sans password or you open yourself to all sorts of vulnerabilities.

Now, let's say you want to prevent certain users from using sudo. You can do this as well. If you have one user that is to be administrator of a machine, say bethany, and all other users should be uses without admin privileges, you can do this a couple of ways. The first (and less desirable method) is to do the following:

Add an entry for bethany like so:

bethany ALL=(ALL) ALL

And now comment out the entry:

```
%admin ALL=(ALL) ALL
```

by adding a "#" character at the beginning of the line.

At this point the only user on the system that will be able to run administrative commands is bethany. Now this can cause issues if you have certain applications that must run with administrative privileges and are allowed such privileges by being a member of the admin group. You can avoid this issue by simply opening up the Users administrative tool and removing all users, except for those you want to be allowed to have admin rights from the admin group. Let's stick with our example. You want all users other than Bethany to have restricted access to run administrative commands and tools. To do this, follow these steps:

1. Open up the User administrator.
2. Go to the Groups manager.
3. Select the admin group.
4. Click properties.
5. Uncheck all users but bethany from the list.
6. Close the Groups manager and the User administrator.

Now only the user bethany will have administrative rights on the machine.

The mount Command

On Linux, UNIX, and similar operating systems, file systems on different partitions and removable devices like CDs, DVDs, or USB flash drives can be attached to a certain point (that is, the *mount point*) in the directory tree, and detached again. To attach or detach a file system, you can use the mount or umount command respectively. This chapter describes the basic usage of these commands, and covers some advanced topics such as moving a mount point or creating shared subtrees.

2.1. Listing Currently Mounted File Systems

To display all currently attached file systems, run the mount command with no additional arguments:

```
mount
```

This command displays the list of known mount points. Each line provides important information about the device name, the file system type, the directory in which it is mounted, and relevant mount options in the following form:

device on *directory* type *type* (*options*)

By default, the output includes various virtual file systems such as sysfs, tmpfs, and others. To display only the devices with a certain file system type, supply the -t option on the command line:

mount -t *type*

For a list of common file system types, refer to Table 2.1, “Common File System Types”. For an example on how to use the mount command to list the mounted file systems, see Example 2.1, “Listing Currently Mounted ext3 File Systems”.

Example 2.1. Listing Currently Mounted ext3 File Systems

Usually, both / and /boot partitions are formatted to use ext3. To display only the mount points that use this file system, type the following at a shell prompt:

```
~]$ mount -t ext3
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
/dev/vda1 on /boot type ext3 (rw)
```

2.2. Mounting a File System

To attach a certain file system, use the mount command in the following form:

mount [*option…*] *device directory*

When the mount command is run, it reads the content of the /etc/fstab configuration file to see if the given file system is listed. This file contains a list of device names and the directory in which the selected file systems should be mounted, as well as the file system type and mount options. Because of this, when you are mounting a file system that is specified in this file, you can use one of the following variants of the command:

mount [*option…*] *directory*
mount [*option…*] *device*

Note that unless you are logged in as root, you must have permissions to mount the file system (see Section 2.2.2, “Specifying the Mount Options”).

2.2.1. Specifying the File System Type

In most cases, mount detects the file system automatically. However, there are certain file systems, such as NFS (Network File System) or CIFS (Common Internet File System), that are not recognized, and need to be specified manually. To specify the file system type, use the mount command in the following form:

mount -t *type device directory*

Table 2.1, “Common File System Types” provides a list of common file system types that can be used with the mount command. For a complete list of all available file system types, consult the relevant manual page as referred to in Section 2.4.1, “Installed Documentation”.

Table 2.1. Common File System Types

Type	**Description**
ext2	The ext2 file system.

Type	Description
ext3	The ext3 file system.
ext4	The ext4 file system.
iso9660	The ISO 9660 file system. It is commonly used by optical media, typically CDs.
jfs	The JFS file system created by IBM.
nfs	The NFS file system. It is commonly used to access files over the network.
nfs4	The NFSv4 file system. It is commonly used to access files over the network.
ntfs	The NTFS file system. It is commonly used on machines that are running the Windows operating system.
udf	The UDF file system. It is commonly used by optical media, typically DVDs.
vfat	The FAT file system. It is commonly used on machines that are running the Windows operating system, and on certain digital media such as USB flash drives or floppy disks.

Example 2.2. Mounting a USB Flash Drive

Older USB flash drives often use the FAT file system. Assuming that such drive uses the /dev/sdc1 device and that the /media/flashdisk/ directory exists, you can mount it to this directory by typing the following at a shell prompt as root:

```
~]# mount -t vfat /dev/sdc1 /media/flashdisk
```

2.2.2. Specifying the Mount Options

To specify additional mount options, use the command in the following form:

mount -o *options*

When supplying multiple options, do not insert a space after a comma, or mount will incorrectly interpret the values following spaces as additional parameters.

Table 2.2, “Common Mount Options” provides a list of common mount options. For a complete list of all available options, consult the relevant manual page as referred to in Section 2.4.1, “Installed Documentation”.

Table 2.2. Common Mount Options

Option	Description
async	Allows the asynchronous input/output operations on the file system.
auto	Allows the file system to be mounted automatically using the mount -a command.
defaults	Provides an alias for async,auto,dev,exec,nouser,rw,suid.
exec	Allows the execution of binary files on the particular file system.
loop	Mounts an image as a loop device.
noauto	Disallows the automatic mount of the file system using the mount -a command.
noexec	Disallows the execution of binary files on the particular file system.
nouser	Disallows an ordinary user (that is, other than root) to mount and unmount the file system.
remount	Remounts the file system in case it is already mounted.
ro	Mounts the file system for reading only.
rw	Mounts the file system for both reading and writing.
user	Allows an ordinary user (that is, other than root) to mount and unmount the file system.

2.3. Unmounting a File System

To detach a previously mounted file system, use either of the following variants of the umount command:

umount *directory*
umount *device*

Daemon Definition

A *daemon* is a type of program on Unix-like operating systems that runs unobtrusively in the background, rather than under the direct control of a user, waiting to be activated by the occurrence of a specific event or condition.

Unix-like systems typically run numerous daemons, mainly to accommodate requests for services from other computers on a network, but also to respond to other programs and to hardware activity. Examples of actions or conditions that can trigger daemons into activity are a specific time or date, passage of a specified time interval, a file landing in a particular directory, receipt of an e-mail or a Web request made through a particular communication line. It is not necessary that the perpetrator of the action or condition be aware that a daemon is *listening*, although programs frequently will perform an action only because they are aware that they will implicitly arouse a daemon.

Daemons are usually instantiated as *processes*. A process is an *executing* (i.e., running) instance of a program. Processes are managed by the *kernel* (i.e., the core of the operating system), which assigns each a unique *process identification number* (PID).

There are three basic types of processes in Linux: interactive, batch and daemon. Interactive processes are run interactively by a user at the *command line* (i.e., all-text mode). Batch processes are submitted from a queue of processes and are not associated with the command line; they are well suited for performing recurring tasks when system usage is otherwise low.

Daemons are recognized by the system as any processes whose *parent process* has a PID of one, which always represents the process *init*. init is always the first process that is started when a Linux computer is *booted up* (i.e., started), and it remains on the system until the computer is turned off. init *adopts* any process whose *parent* process *dies* (i.e., terminates) without waiting for the *child* process's status. Thus, the common method for launching a daemon involves *forking* (i.e., dividing) once or twice, and making the parent (and grandparent) processes die while the child (or grandchild) process begins performing its normal function.

Some daemons are launched via *System V init scripts*, which are *scripts* (i.e., short programs) that are run automatically when the system is booting up. They may either survive for the duration of the session or be regenerated at intervals.

Many daemons are now started only as required and by a single daemon, *xinetd* (which has replaced *inetd* in newer systems), rather than running continuously. xinetd, which is referred to as a *TCP/IP super server*, itself is started at boot time, and it listens to the *ports* assigned to the processes listed in the */etc/inetd.conf* or in */etc/xinetd.conf* configuration file. Examples of daemons that it starts include *crond* (which runs scheduled tasks), *ftpd* (file transfer), *lpd* (laser printing), *rlogind* (remote login), *rshd* (remote command execution) and *telnetd* (telnet).

In addition to being launched by the operating system and by application programs, some daemons can also be started manually. Examples of commands that launch daemons include *binlogd* (which logs binary events to specified files), *mysqld* (the MySQL databse server) and *apache* (the Apache web server).

Job scheduling system in Linux

Most of the Linux users are aware of how commands are run, processes are manipulated and scripts are executed in terminal. But, if you are a Linux system administrator, you might want them to start and execute automatically in the background. As an example, you might consider running a backup job every day, at a specific time, automatically. Or you might consider an example of collecting inventory data of the systems deployed across your network, by running a script automatically on monthly basis. Linux and UNIX systems allow you to schedule jobs in the future, either just once, or on a recurring schedule. Many administrative tasks must be done frequently and regularly on a Linux system. These include rotating log files so filesystems do not become full, backing up data, and connecting to a time server to keep your system time synchronized

One Time Task Scheduling using `at` Command in Linux

at command is very useful for scheduling one time tasks. It reads commands from standard input or script/file which can be executed later once. But we can't use at command for any recurring tasks. At command can be useful for shutdown system at specified time, Taking one time backup, sending email as reminder at specified time etc. This article will help you to understand the working of at command with useful examples.

Commands used with at:

at : execute commands at specified time.
atq : lists the pending jobs of users.
atrm : delete jobs by their job number.

1. Schedule first job using at command

Below example will schedule "ls -l" command to be executed on next 9:00 AM once.

```
# at 9:00 AM
at> ls -l
at> ^d
job 3 at 2013-03-23 09:00
```

Use ^d to exit from at prompt.

2. List the scheduled jobs using atq

When we list jobs by root account using atq , it shows all users jobs in result. But if we execute it from non root account, it will show only that users jobs.

```
# atq
3       2013-03-23 09:00 a root
5       2013-03-23 10:00 a rahul
1       2013-03-23 12:00 a root
```

Fileds description:

First filed: job id
Second filed: Job execution date
third filed: Job exectuion time
Last fileld: User name, under which job is scheduled.

3. Remove scheduled job using atrm

You can remove any at job using atrm using there job id.

```
# atrm 3
# atq
5       2013-03-23 10:00 a rahul
1       2013-03-23 12:00 a root
```

4. Check the content of scheduled at job

atq command only shows the list of jobs but if you want to check what script/commands are scheduled with that task, below example will helps you.

```
# at -c 5
```

In above example 5 is the job id.

Examples of at Command:

Example 1: Schedule task at comming 10:00 AM.

```
# at 10:00 AM
```

Example 2: Schedule task at 10:00 AM on comming Sunday.

```
# at 10:00 AM Sun
```

Example 3: Schedule task at 10:00 AM on comming 25’th July.

```
# at 10:00 AM July 25
```

Example 4: Schedule task at 10:00 AM on comming 22’nd June 2015.

```
# at 10:00 AM 6/22/2015
# at 10:00 AM 6.22.2015
```

Example 5: Schedule task at 10:00 AM on same date at next month.

```
# at 10:00 AM next month
```

Example 6: Schedule task at 10:00 AM tomorrow.

```
# at 10:00 AM tomorrow
```

Example 7: Schedule task at 10:00 AM tomorrow.

```
# at 10:00 AM tomorrow
```

Example 8: Schedule task to execute just after 1 hour.

```
# at now + 1 hour
```

Example 9: Schedule task to execute just after 30 minutes.

```
# at now + 30 minutes
```

Example 10: Schedule task to execute just after 1 and 2 weeks.

```
# at now + 1 week
# at now + 2 weeks
```

Example 11: Schedule task to execute just after 1 and 2 years.

```
# at now + 1 year
# at now + 2 years
```

Example 12: Schedule task to execute at mid night.

```
# at midnight
```

Above job will execute on next 12:00 AM

Run jobs at regular intervals using cron in Linux

Running jobs at regular intervals is managed by the *cron* facility, which consists of the crond daemon and a set of tables describing what work is to be done and with what frequency. The daemon wakes up every minute and checks the crontabs to determine what needs to be done. The main job of cron daemon is to inspect the configuration regularly (every minute to be more precise) and check if there is any job to be completed. Users manage crontabs using the crontab command. The crond daemon is usually started by the init process at system startup.

For instance, you can automate process like **backup**, **schedule updates** and **synchronization of files** and many more. **Cron** is a daemon to run schedule tasks. Cron wakes up every minute and checks schedule tasks in crontable. **Crontab** (**CRON TABle**) is a table where we can schedule such kind of repeated tasks.

Crontab file consists of command per line and have six fields actually and separated either of space or tab. The beginning five fields represent time to run tasks and last field is for command.

1. Minute (hold values between **0-59**)
2. Hour (hold values between **0-23**)
3. Day of Month (hold values between **1-31**)
4. Month of the year (hold values between **1-12** or **Jan-Dec**, you can use first three letters of each month’s name i.e **Jan or Jun**.)
5. Day of week (hold values between **0-6** or **Sun-Sat**, Here also you can use first three letters of each day’s name i.e **Sun or Wed**.)
6. Command

1. List Crontab Entries

List or manage the task with crontab command with **-l** option for current user.

```
# crontab -l

00 10 * * * /bin/ls >/ls.txt
```

2. Edit Crontab Entries

To edit crontab entry, use **-e** option as shown below. In the below example will open schedule jobs in **VI** editor. Make a necessary changes and quit pressing **:wq** keys which saves the setting automatically.

```
# crontab -e
```

3. List Scheduled Cron Jobs

To list scheduled jobs of a particular user called **tecmint** using option as **-u** (**User**) and **-l** (**List**).

```
# crontab -u tecmint -l

no crontab for tecmint
```

Only **root** user have complete privileges to see other users crontab entry. Normal user can't view it others.

4. Remove Crontab Entry

Caution: Crontab with **-r** parameter will remove complete scheduled jobs without confirmation from crontab. Use **-i** option before deleting user's crontab.

```
# crontab -r
```

5. Prompt Before Deleting Crontab

crontab with **-i** option will prompt you confirmation from user before deleting user's crontab.

```
# crontab -i -r

crontab: really delete root's crontab?
```

6. Allowed special character (*, -, /, ?, #)

1. **Asterik(*)** – Match all values in the field or any possible value.
2. **Hyphen(-)** – To define range.
3. **Slash (/)** – 1st field /10 meaning every ten minute or increment of range.
4. **Comma (,)** – To separate items.

7. System Wide Cron Schedule

System administrator can use predefine cron directory as shown below.

1. /etc/cron.d
2. /etc/cron.daily
3. /etc/cron.hourly
4. /etc/cron.monthly
5. /etc/cron.weekly

8. Schedule a Jobs for Specific Time

The below jobs delete empty files and directory from **/tmp** at **12:30** am daily. You need to mention user name to perform crontab command. In below example **root** user is performing cron job.

```
# crontab -e
```

```
30 0 * * *   root   find /tmp -type f -empty -delete
```

9. Special Strings for Common Schedule

Strings	**Meanings**
@reboot	Command will run when the system reboot.
@daily	Once per day or may use @midnight.
@weekly	Once per week.
@yearly	Once per year. we can use @annually keyword also.
@daily	Once per day.

Need to replace five fields of cron command with keyword if you want to use the same.

10. Multiple Commands with Double amper-sand(&&)

In below example command1 and command2 run daily.

```
# crontab -e
```

```
@daily <command1> && <command2>
```

The Shadow Password

passwd is a tool on most Unix and Unix-like operating systems used to change a user's password. The password entered by the user is run through a key derivation function to create a hashed version of the new password, which is saved. Only the hashed version is stored; the entered password is not saved for security reasons.

When the user logs on, the password entered by the user during the log on process is run through the same key derivation function and the resulting hashed version is compared with the saved version. If the hashes are identical, the entered password are considered to be identical, and so the user is authenticated. In theory, it is possible to occur that two different passwords produce the same hash. However, cryptographic hash functions are designed in such a way that finding any password that produces the given hash is very difficult and practically unfeasible, so if the produced hash matches the stored one, the user can be authenticated.

The passwd command may be used to change passwords for local accounts, and on most systems, can also be used to change passwords managed in a distributed authentication mechanism such as NIS, Kerberos, or LDAP.

Shadow file[edit]

`/etc/shadow` is used to increase the security level of passwords by restricting all but highly privileged users' access to hashed password data. Typically, that data is kept in files owned by and accessible only by the super user.

Systems administrators can reduce the likelihood of brute force attacks by making the list of hashed passwords unreadable by unprivileged users. The obvious way to do this is to make the `passwd` database itself readable only by the root user. However, this would restrict access to other data in the file such as username-to-userid mappings, which would break many existing utilities and provisions. One solution is a "shadow" password file to hold the password hashes separate from the other data in the world-readable *passwd* file. For local files, this is usually `/etc/shadow` on Linux and Unix systems, or `/etc/master.passwd` on BSD systems; each is readable only by *root*. (Root access to the data is considered acceptable since on systems with the traditional "all-powerful root" security model, the root user would be able to obtain the information in other ways in any case). Virtually all recent Unix-like operating systems use shadowed passwords.

The shadow password file does not entirely solve the problem of attacker access to hashed passwords, as some network authentication schemes operate by transmitting the hashed password over the network (sometimes in cleartext, e.g., Telnet[3]), making it vulnerable to interception. Copies of system data, such as system backups written to tape or optical media, can also become a means for illicitly obtaining hashed passwords. In addition, the functions used by legitimate password-checking programs need to be written in such a way that malicious programs cannot make large numbers of authentication checks at high rates of speed.

RAID Technology

RAID (originally **redundant array of inexpensive disks**, now commonly **redundant array of independent disks**) is a data storage virtualization technology that combines multiple physical disk drive components into a single logical unit for the purposes of data redundancy, performance improvement, or both.[1]

Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the required level of redundancy and performance. The different schemas, or data distribution layouts, are named by the word RAID followed by a number, for example RAID 0 or RAID 1. Each schema, or a RAID level, provides a different balance among the key goals: reliability, availability, performance, and capacity. RAID levels greater than RAID 0 provide protection against unrecoverable sector read errors, as well as against failures of whole physical drives.

Overview[edit]

Many RAID levels employ an error protection scheme called "parity", a widely used method in information technology to provide fault tolerance in a given set of data. Most use simple XOR, but RAID 6 uses two separate parities based respectively on addition and multiplication in a particular Galois field or Reed–Solomon error correction.[14]

RAID can also provide data security with solid-state drives (SSDs) without the expense of an all-SSD system. For example, a fast SSD can be mirrored with a mechanical drive. For this configuration to provide a significant speed advantage an appropriate controller is needed that uses the fast SSD for all read operations. Adaptec calls this "hybrid RAID".[15]

Standard levels[edit]

A number of standard schemes have evolved. These are called *levels*. Originally, there were five RAID levels, but many variations have evolved, notably several nested levels and many non-standard levels (mostly proprietary). RAID levels and their associated data formats are standardized by the Storage Networking Industry Association (SNIA) in the Common RAID Disk Drive Format (DDF) standard:[16][17]

RAID 0

RAID 0 consists of striping, without mirroring or parity. The capacity of a RAID 0 volume is the sum of the capacities of the disks in the set, the same as with a spanned volume. There is no added redundancy for handling disk failures, just as with a spanned volume. Thus, failure of one disk causes the loss of the entire RAID 0 volume, with reduced possibilities of data recovery when compared to a broken spanned volume. Striping distributes the contents of files roughly equally among all disks in the set, which makes concurrent read or write operations on the multiple disks almost inevitable and results in performance improvements. The concurrent operations make the throughput of most read and write operations equal to the throughput of one disk multiplied by the number of disks. Increased throughput is the big benefit of RAID 0 versus spanned volume,[11] at the cost of increased vulnerability to drive failures.

RAID 1

RAID 1 consists of data mirroring, without parity or striping. Data is written identically to two (or more) drives, thereby producing a "mirrored set" of drives. Thus, any read request can be serviced by any drive in the set. If a request is broadcast to every drive in the set, it can be serviced by the drive that accesses the data first (depending on itsseek time and rotational latency), improving performance. Sustained read throughput, if the controller or software is optimized for it, approaches the sum of throughputs of every drive in the set, just as for RAID 0. Actual read throughput of most RAID 1 implementations is slower than the fastest drive. Write throughput is always slower because every drive must be updated, and the slowest drive limits the write performance. The array continues to operate as long as at least one drive is functioning.[11]

RAID 2

RAID 2 consists of bit-level striping with dedicated Hamming-code parity. All disk spindle rotation is synchronized and data is striped such that each sequential bit is on a different drive. Hamming-code parity is calculated across corresponding bits and stored on at least one parity drive.[11] This level is of historical significance only; although it was used on some early machines (for example, the Thinking Machines CM-2),[18] as of 2014 it is not used by any of the commercially available systems.[19]

RAID 3

RAID 3 consists of byte-level striping with dedicated parity. All disk spindle rotation is synchronized and data is striped such that each sequential byte is on a different drive. Parity is calculated across corresponding bytes and stored on a dedicated parity drive.[11] Although implementations exist,[20] RAID 3 is not commonly used in practice.

RAID 4

RAID 4 consists of block-level striping with dedicated parity. This level was previously used by NetApp, but has now been largely replaced by a proprietary implementation of RAID 4 with two parity disks, called RAID-DP.[21] The main advantage of RAID 4 over RAID 2 and 3 is I/O parallelism: in RAID 2 and 3, a single read/write I/O operation requires reading the whole group of data drives, while in RAID 4 one I/O read/write operation does not have to spread across all data drives. As a result, more I/O operations can be executed in parallel, improving the performance of small transfers.[2]

RAID 5

RAID 5 consists of block-level striping with distributed parity. Unlike RAID 4, parity information is distributed among the drives, requiring all drives but one to be present to operate. Upon failure of a single drive, subsequent reads can be calculated from the distributed parity such that no data is lost. RAID 5 requires at least three disks.[11] RAID 5 is seriously affected by the general trends regarding array rebuild time and the chance of drive failure during rebuild.[22] Rebuilding an array requires reading all data from all disks, opening a chance for a second drive failure and the loss of the entire array. In August 2012, Dell posted an advisory against the use of RAID 5 in any configuration on Dell EqualLogic arrays and RAID 50 with "Class 2 7200 RPM drives of 1 TB and higher capacity" for business-critical data.[23]

RAID 6

RAID 6 consists of block-level striping with double distributed parity. Double parity provides fault tolerance up to two failed drives. This makes larger RAID groups more practical, especially for high-availability systems, as large-capacity drives take longer to restore. RAID 6 requires a minimum of four disks. As with RAID 5, a single drive failure results in reduced performance of the entire array until the failed drive has been replaced.[11] With a RAID 6 array, using drives from multiple sources and manufacturers, it is possible to mitigate most of the problems associated with RAID 5. The larger the drive capacities and the larger the array size, the more important it becomes to choose RAID 6 instead of RAID 5.[24] RAID 10 also minimizes these problems.[25]

Logical Volume Manager (Linux)

In Linux, **Logical Volume Manager** (**LVM**) is a device mapper target that provides logical volume management for the Linux kernel. Most modern Linux distributions are LVM-aware to the point of being able to have their root file systems on a logical volume.[3][4][5]

Common uses[edit]

LVM is commonly used for the following purposes:

- Managing large hard disk farms by allowing disks to be added and replaced without downtime or service disruption, in combination with hot swapping.
- On small systems (like a desktop at home), instead of having to estimate at installation time how big a partition might need to be in the future, LVM allows file systems to be easily resized later as needed.
- Performing consistent backups by taking snapshots of the logical volumes.
- Creating single logical volumes of multiple physical volumes or entire hard disks (somewhat similar to RAID 0, but more similar to JBOD), allowing for dynamic volume resizing.

LVM can be considered as a thin software layer on top of the hard disks and partitions, which creates an abstraction of continuity and ease-of-use for managing hard drive replacement, re-partitioning, and backup.

Features[edit]

Basic functionality[edit]

- Volume groups (VGs) can be resized online by absorbing new physical volumes (PVs) or ejecting existing ones.
- Logical volumes (LVs) can be resized online by concatenating extents onto them or truncating extents from them.
- LVs can be moved between PVs.
- Creation of read-only snapshots of logical volumes (LVM1), or read-write snapshots (LVM2).
- VGs can be split or merged *in situ* as long as no LVs span the split. This can be useful when migrating whole LVs to or from offline storage.
- LVM objects can be tagged for administrative convenience.[6]
- VGs and LVs can be made active as the underlying devices become available through use of the `lvmetad` daemon.[7]

Metadevice :

- A virtual device composed of several physical devices – slices/disks . All the operations are carried out using metadevice name and transparently implemented on the individual device.
- **RAID :** A group of disks used for creating a virtual volume is called array and depending on disk/slice arrangement these are called various types of RAID (Redundant Array of Independent Disk).
- RAID 0 Concatenation/Striping
- RAID 1 Mirroring
- RAID 5 Striped array with rotating parity.
- **Concatenation :**Concatenation is joining of two or more disk slices to add up the disk space . Concatenation is serial in nature i.e. sequential data operation are performed serially on first disk then second disk and so on . Due to serial nature new slices can be added up without having to take the backup of entire concatenated volume ,adding slice and restoring backup .
- **Striping** :Spreading of data over multiple disk drives mainly to enhance the performance by distributing data in alternating chunks – 16 k interleave across the stripes . Sequential data operations are performed in parallel on all the stripes by reading/writing 16k data blocks alternatively form the disk stripes.
- **Mirroring** : Mirroring provides data redundancy by simultaneously writing data on to two sub mirrors of a mirrored device . A submirror can be a stripe or concatenated volume and a mirror can have three mirrors . Main concern here is that a mirror needs as much as the volume to be mirrored.
- **RAID 5** : RAID 5 provides data redundancy and advantage of striping and uses less space than mirroring . A RAID 5 is made up of at least three disk which are striped with parity information written alternately on all the disks . In case of a single disk failure the data can be rebuild using the parity information from the remaining disks .