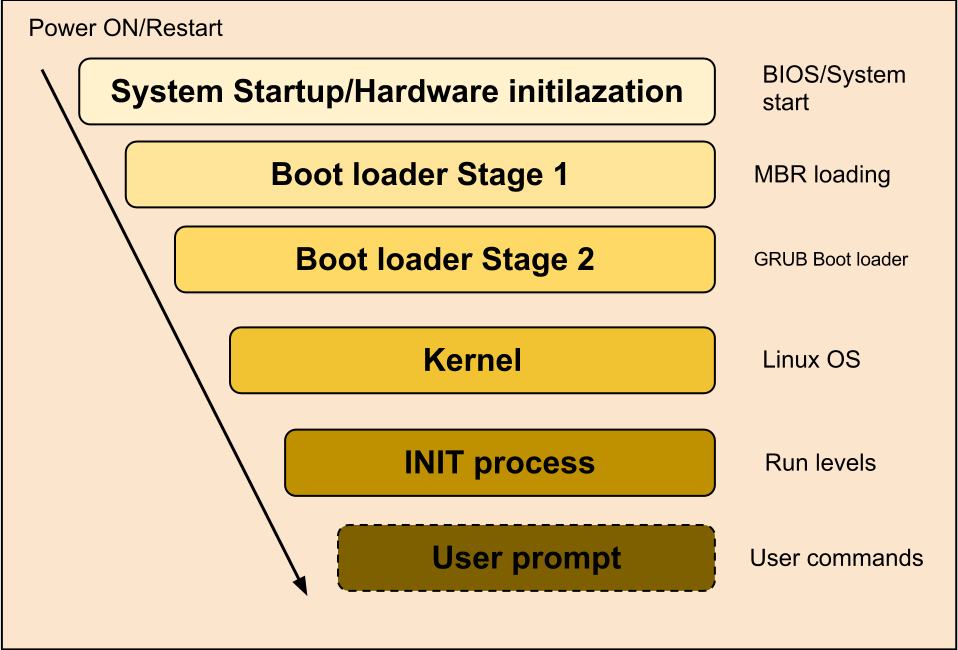
**Linux Booting process explained**



## Stage 1: System startup

## Stage 2: MBR loading

Once the BIOS gives control back to CPU, it will try to load MBR of the first boot device(We will consider it as HDD). MBR is a small part of Hard Disk with just a size of 512 Bytes, **I repeat its just 512 Bytes**. This MBR resides at the starting of HDD or end of HDD depending on manufacturer.

**What is MBR?**

MBR(Master Boot recorder) is a location on disk which have details about

* **Primary boot loader code(This is of 446 Bytes)**
* **Partition table information(64 Bytes)**
* **Magic number(2 Bytes)**

Which will be equal to 512B (446+64+2)B.

**Primary Boot loader code:** This code provides boot loader information and location details of actual boot loader code on the hard disk. This is helpful for CPU to load second stage of Boot loader.

**Partition table:** MBR contains 64 bytes of data which stores Partition table information such as what is the start and end of each partition, size of partition, type of partition(Whether it's a primary or extended etc). As we all know HDD support only 4 partitions, this is because of the limitation of its information in MBR. For a partition to represent in MBR, it requires 16 Bytes of space in it so at most we will get 4 partitions. [**Check our detail post on this concept**](http://www.linuxnix.com/2009/05/why-we-can-create-only-up-to-4-primary-partations.html)to know more about this.

**Magic Number:** The magic number service as validation check for MBR. If MBR gets corrupted this magic number is used to retrieve it. What to take backup of your MBR[**try this**](http://www.linuxnix.com/2009/11/how-to-take-the-backup-of-mbrmaster-boot-recorder.html).

Once your CPU knows all these details, it will try to analyse them and read the first portion of MBR to load Second stage of Boot loader

**How To Take The Backup Of MBR(Master Boot Recorder)**

**1.How to take the backup and restore MBR? Why do you require to take the backup of your MBR?  
Ans :** **MBR (Master Boot Recorder)** is a vital part of your hard disk which contains booting information, without it its difficult to boot the system. Suppose you have windows and Linux duel boot on your machine and as you know windows is more prone to virus attacks. So it’s always better to backup your MBR to be in safe place.  
  
**2. How to take backup of your MBR?  
Ans :** Using **dd command** (**d**ataset **d**efinition). Here are the steps to take backup of you**MBR** and keep it in safe place to restore your system if it get corrupted.  
**#dd if=/dev/hdx of=/safe/location bs=512 count=1**

**Let me explain the above command how it will work.  
“If”** in the command is nothing but to specify Input File, here we are specifying our input file as hard disk(if the hard disk is **/dev/hda** it is primary master, so for general purpose I given **‘x’**). **“of”** in the command is nothing but to specify Output File, here we are specifying our output file as **/safe/location.**Then **“bs”** this is nothing but**block size** to write in to hard disk. And then **“count”** nothing but how many times you want to write date this many **block sizes**. Here in this example **count=1** that means first **512 bytes** of the hard disk is copied to the specified location.

**3.How to restore the MBR?  
#dd if=/safe/location of=/dev/hdx bs=512 count=1**

**Note :** Please replace **“hdx”** with your **hard disk** name.  
This is bit complex, **Is there any other way to restore MBR?**Yes, if you have Linux or Windows bootable CD, we can easily restore your MBR if you forgot to take backup(And this method is very much easy to do restoration of MBR when compared to previous method).

**Method1 :**With Redhat Linux bootable CD.  
For this you have to boot your system to **rescue mode**, then mount your file system to rescue mode and execute below command to restore your MBR  
**#grub-install /dev/hdx**

**Note :** Please replace **hdx**with your **hard disk** name. After that you just reboot your system. Your system will be live and working.

**Method2 :**With Windows XP bootable CD.  
**Step1 :**Boot the system with XP bootable cd  
**Step2 :**Press **f8** to go to repair mode in Windows  
**Step3 :**Once you got the **c**drive prompt just type below command  
**Fixmbr**This command will fix the **MBR record**.

**Some FAQ’s  
1. What is the MBR size?  
Ans :** MBR size is just **512 bytes**.

**2.What MBR conations?  
Ans :** Mainly MBR can be divided in two parts  
**a.**Boot loader information block(which is of **448 bytes**)  
**b.** Partition table information(which is of just**64 bytes**)

**3.How many partition we can create on a hard disk?  
Ans :** Totally we can create **four partitions** as below  
**a.**Four primary parathions.  
**b.T**hree primary and one extended partition.  
**c.T**wo primary and one extended parathion.  
**d.O**ne primary and one extended parathion.

**Note :** In extended parathion we can create logical partitions up to **24** in number.

**4.Why we cannot create more then 4 partition as mention above?  
Ans :** In MBR, the partition table info is just stored in **64 bytes**, and one parathion information to store in MBR requires **16 bytes** of space. So at most you can create only **4 partitions** as mention above.

## Stage 3: Boot loader Stage 2 (GRUB loader)

Once the Bootloader stage 1 is completed and able to find the actual bootloader location, Stage 1 bootloader start second stage by loading Bootloader into memory. In this stage GRUB(Grand Unified Bootloader) which is located in the first 30 kilobytes of hard disk immediately following the MBR is loaded into RAM for reading its configuration and displays the GRUB boot menu (where the user can manually specify the boot parameters) to the user. GRUB loads the user-selected (or default) kernel into memory and passes control on to the kernel. If user do not select the OS, after a defined timeout GRUB will load the default kernel in the memory for starting it.

## Stage 4: Kernel

Once the control is given to kernel which is the central part of all your OS and act as a mediator of hardware and software components. Kernel once loaded into to RAM it always resides on RAM until the machine is shutdown. Once the Kernel starts its operations the first thing it do is executing INIT process.

## Stage 5: INIT

This is the main stage of Booting Process

**init(initialization)** process is the root/parent process of all the process which run under Linux/Unix. The first process it runs is a script at /etc/rc.d/rc.sysinit which check all the system properties, hardware, display, SElinux, load kernel modules, file system check, file system mounting etc. Based on the appropriate run-level, scripts are executed to start/stop various processes to run the system and make it functional. INIT process read /etc/inittab which is an initialization table which defines starting of system programs. INIT will start each run level one after the other and start executing scripts corresponds to that runlevel. [**Know more about runlevels here**](http://www.linuxnix.com/2008/05/the-runlevels.html). The script information is stored in different folders in /etc/ folder

/etc/rc0.d/ –Contain Start/Kill scripts which should be run in Runlevel 0  
/etc/rc1.d/ –Contain Start/Kill scripts which should be run in Runlevel 1  
/etc/rc2.d/ –Contain Start/Kill scripts which should be run in Runlevel 2  
/etc/rc3.d/ –Contain Start/Kill scripts which should be run in Runlevel 3  
/etc/rc4.d/ –Contain Start/Kill scripts which should be run in Runlevel 4  
/etc/rc5.d/ –Contain Start/Kill scripts which should be run in Runlevel 5  
/etc/rc6.d/ –Contain Start/Kill scripts which should be run in Runlevel 6

[**Know more about S and K convention used in the script names under /etc/rc\*.d here.**](http://www.linuxnix.com/2010/12/how-to-set-priority-for-a-service-at-booting-of-linux.html)

Once the initialization process completes mandatory run level and reach to default runlevel set in **/etc/inittab,** init process run one more file **/etc/rc.local** which are the last commands run in initialization process or even booting process. Once everything is completed the control is given back to the kernel

## Stage 6: User prompt

This is actually not part of booting process but thought of including it here for better understating. Once the Kernel get the control it start multiple instances of "getty" which waits for console logins which spawn one's user shell process and gives you user prompt to login.

########## Notes for /etc/inittab file

After the root filesystem is mounted as read-write, the kernel executes a program called *init*. When *init* has completed its execution, the system is up and running. The *init* process is highly configurable, via the */etc/inittab* file and files and directories with the *init* configuration process.  
  
The following syntax is used for configuration lines in the */etc/inittab* file:  
id:run-levels:action:process [arguments]

* The first field is a unique label, which identifies an entry in the *inittab* file.
* The second field specifies which run-levels this entry applies to.
* The third field specifies the action to be taken.
* The fourth field identifies the process to be run and any arguments that apply to that process. Command-line arguments may also be specified in the fourth field.

The state of a Linux system is specified by its run-level. There are seven run-levels available on a Linux system.

1. 0—Halt (Do not set *initdefault* to this run-level.)
2. 1—Single-user mode
3. 2—Multiuser—with no networking support
4. 3—Full multiuser mode—with networking support
5. 4—Unused
6. 5—X11—with multiuser and networking support
7. 6—Reboot (Do not set *initdefault* to this run-level.)

It’s possible to specify multiple run-levels in the same *init* configuration line. For example, to specify a configuration for both run-level 1 and run-level 3, use the following syntax:  
id:13:action:process [arguments].  
  
Table A lists the possible actions for system run-levels.

|  |  |
| --- | --- |
| Table A | |
| Action | Description |
|  |  |
| respawn | The process is restarted whenever it terminates. |
| wait | The process is run once, and *init* waits until it terminates. |
| once | The process is run once. |
| boot | The process is run during system boot, regardless of the run-level. |
| bootwait | The process is run during system boot, and *init* waits for the |
|  | process to terminate. |
| off | No action is taken. Used to disable a configuration line without |
|  | removing it. May be used instead of a comment (#). |
| ondemand | Normally not used. |
| initdefault | Specifies the default run-level for the system. The process filed is |
|  | ignored. |
| sysinit | The process is run once during system boot. A *sysinit* action takes |
|  | precedence over boot or bootwait actions. |
| powerwait | Run when *init* receives a SIGPWR signal. An uninterruptible power |
|  | supply (UPS) will issue the SIGPWR signal when a power problem |
|  | Is detected. When the SIGPWR signal is issued, *init* will wait until |
|  | the process terminates. |
| powerfail | Same as powerwait, but *init* does not wait for the process to |
|  | terminate. |
| powerokwait | Run when *init* receives a SGIPWR signal and the /*etc/powerstatus* |
|  | file contains the text string *OK*. This file is normally created by the |
|  | UPS monitoring software and is used to indicate that the power |
|  | problem is corrected. |
| ctrlaltdel | Run when *init* receives a SIGINT signal. |
| kbrequest | Run when *init* receives a keyboard signal from the keyboard handler. |

Possible actions for system run-levels.

The /etc/rc.d/rc.sysinit script  
When a Linux system is booted, the kernel runs *init*, which in turn runs the*/etc/rc.d/rc sysinit* script. This script is run before any other actions are taken because of the following line in the /*etc/inittab* file:  
si::sysinit: /etc/rc.d/rc.sysinit  
  
The run-level field in this line is empty because *init* recognizes *sysinit* as a system initialization action. The *rc.sysinit* performs several tasks, including setting the hostname, enabling the swap partition, checking filesystems, and loading kernel modules. Normally, you will not have to modify the */etc/rc.sysinit*script.  
  
The /etc/rc.d/rc script  
When *init* is told to change to one of the seven possible run-levels, a script specified in one of the following seven lines in the /*etc/inittab* file is run:  
l0:0:wait:/etc/rc.d/rc 0  
l1:1:wait:/etc/rc.d/rc 1  
l2:2:wait:/etc/rc.d/rc 2  
l3:3:wait:/etc/rc.d/rc 3  
l4:4:wait:/etc/rc.d/rc 4  
l5:5:wait:/etc/rc.d/rc 5  
l6:6:wait:/etc/rc.d/rc 6  
  
For each run-level [0-6], a script named /*etc/rc.d/rc* is run, with the run-level used as the argument to the script. This script is used to start and stop all services in the specified run-level. For example, if init is told to change the run-level to 5, the /*etc/rc.d/rc* script will be run with an argument of 5 in a command-line like the following:  
/etc/rc.d/rc 3  
  
The *rc* script performs a three-step process:

1. It checks to ensure that a subdirectory exists for the specified run-level. If the script is run with an argument of 5, it will make sure that the /*etc/rc.d/rc5.d* subdirectory exists. If this directory exists, the script continues to the next step. If this directory does not exist, the run-level is not changed.
2. It determines whether any of the programs (often called *services* or*processes*) that are supposed to run in the specified run-level are already running. If a service is already running, the script will kill it so that the service may be started in the next step.
3. Any script with the *K* argument is stopped and restarted, and any script with the *S* argument is started.

A typical /*etc/rc.d/rc5.d* file is shown below. This example shows an abbreviated version of the /*etc/rc.d/rc.5* directory on my Red Hat 6.0 system. The listing shows the services that are to be started or stopped and the symbolic link to the scripts in the */etc/rc.d/init.d* directory.  
lrwxrwxrwx 1 root root 16 May 21 12:54 K08autofs -> ../init.d/autofs  
lrwxrwxrwx 1 root root 15 May 21 12:54 K15httpd -> ../init.d/httpd  
lrwxrwxrwx 1 root root 18 May 21 12:54 S75keytable -> ../init.d/keytable  
lrwxrwxrwx 1 root root 13 May 21 12:54 S85gpm -> ../init.d/gpm  
  
Note that the scripts in the /*etc/rc.d/init.d* directory are linked in two ways. Some are linked with the syntax  
K[two-digit number] script-name  
  
while others use the syntax  
S[two-digit number] script-name  
  
The scripts beginning with *S* are started, and the scripts beginning with *K* are stopped and then restarted if necessary. The *two-digit* specifies the order of execution for these services, with the lowest numbered services being executed first. In the above example, the script *K08autofs* is executed before the service *K15httpd*. Once the *rc* script is finished, the processing required for*init* is also finished, and the system becomes available in the specified run-level.  
  
The /*etc/rc.d/init.d* directory  
The /*etc/rc.d/init.d* directory is used to hold all the scripts needed by all run-levels. Each script in this directory is used to start or stop a particular service.  
  
All the scripts in this directory use a command-line syntax. For example, to start the *ipchains* service, which is a program used to configure a Linux firewall, the following script would be run:  
/etc/rc.d/init.d/ipchains start  
  
And to stop the *ipchains* service, use this command:  
/etc/rc.d/init.d/ipchains stop  
  
All scripts in the /*etc/rc.d/init.d* directory used *start* and *stop* as arguments to start or terminate a service. These scripts are read via symbolic links to the /*etc/rc[0-6]* directories.  
  
The /etc/rc.d/rc.local script  
The /*etc/rc.d/rc.local* script is run once at the end of run-levels 2, 3, and 5. Any command or script that needs to be run once per boot may be added to this file.  
  
The /etc/rc.d/rc.serial script  
The /*etc/rc.d/rc.serial* script is typically run once at the end of run-level 3 or 5 to initialize serial ports.  
  
Booting up your system  
During boot up, *init* first runs the *rc.sysinit* script and then runs the script for the default run-level. The default run-level is set in the /*etc/inittab* file with an entry similar to the example shown here:  
id:3:initdefault:  
  
In this example, the default is set to 3. This means that *init* will run the script required to put the system in multiuser mode, with networking support. The section following the default run-level setting contains a line specifying what needs to be done to change to each run-level. For run-level 3, the scripts in the */etc/rc.d/rc3.d* directory that begin with *S* will be run. The *rc* script will also provide a *start* command to each script that begins with *S*. When all of these scripts are run, the *rc* script will finish and the system will be available in the specified run-level.

[inittab explained](http://www.networknuts.net/inittab-runlevels-redhat-training/)

<http://www.networknuts.net/tag/rc-sysinit/>

When the init daemon needs to change the runlevel of the system by starting or stopping daemons, it consults the /etc/inittab file. This file is also consulted when bringing the system to a certain runlevel at boot time.

The format of entries in the /etc/inittab file are as follows:

**label : runlevel(s) : action : command**

The label is an identifier that allows the init daemon to examine this file in alphabetical order; the runlevel specifies to which runlevel the line in /etc/inittab corresponds; the command tells the init daemon what to execute when entering the runlevel; and the action tells the init daemon how to execute the command.

Thus, the line

**id:5:initdefault:**

in the /etc/inittab file tells the init daemon that runlevel 5 is the default runlevel to boot to when initializing the Linux system at system startup.

Secondly, the line

**si::sysinit:/etc/rc.d/rc.sysinit**

tells the init daemon to run the program /etc/rc.d/rc.sysinit before entering a runlevel at system initialization. This program initializes  
the hardware components of the system, sets environment variables such as PATH and HOSTNAME, checks filesystems, & performs system tasks required for daemon loading. The output from the /etc/rc.d/rc.sysinit program is displayed on the terminal screen during system startup as soon as the graphical boot starts. GEEKS, you can always press alt+d to check this output.

Now, GEEKS here the default runlevel is 5, so the line l5:5:wait:/etc/rc.d/rc 5 will be executed in the next section of the /etc/inittab file seen earlier. This will throw the control to line /etc/rc.d/rc 5 & waits for it to finish before proceeding to the rest of the /etc/inittab file.

# The /etc/inittab File

When you boot the system or change run levels with the init or shutdown command, the init daemon starts processes by reading information from the /etc/inittab file. This file defines three important items for the init process:

* The system's default run level
* What processes to start, monitor, and restart if they terminate
* What actions to be taken when the system enters a new run level

Each entry in the /etc/inittab file has the following fields:

id:rstate:action:process

The following table describes the fields in an inittab entry.

Table 8-2 Fields in the inittab File

| **Field** | **Description** |
| --- | --- |
| id | A unique identifier for the entry. |
| rstate | A list of run levels to which this entry applies. |
| action | How the process specified in the process field is to be run. Possible values include: initdefault, sysinit, boot,bootwait, wait, and respawn. |
| process | The command to execute. |

## Example--Default inittab File

The following example shows an annotated default inittab file:

|  |
| --- |
| 1 ap::sysinit:/sbin/autopush -f /etc/iu.ap  2 ap::sysinit:/sbin/soconfig -f /etc/sock2path  3 fs::sysinit:/sbin/rcS sysinit >/dev/msglog 2<>/dev/msglog </dev/console  4 is:3:initdefault:  5 p3:s1234:powerfail:/usr/sbin/shutdown -y -i5 -g0 >/dev/msglog 2<>/dev/...  6 sS:s:wait:/sbin/rcS >/dev/msglog 2<>/dev/msglog </dev/console  7 s0:0:wait:/sbin/rc0 >/dev/msglog 2<>/dev/msglog </dev/console  8 s1:1:respawn:/sbin/rc1 >/dev/msglog 2<>/dev/msglog </dev/console  9 s2:23:wait:/sbin/rc2 >/dev/msglog 2<>/dev/msglog </dev/console  10 s3:3:wait:/sbin/rc3 >/dev/msglog 2<>/dev/msglog </dev/console  11 s5:5:wait:/sbin/rc5 >/dev/msglog 2<>/dev/msglog </dev/console  12 s6:6:wait:/sbin/rc6 >/dev/msglog 2<>/dev/msglog </dev/console  13 fw:0:wait:/sbin/uadmin 2 0 >/dev/msglog 2<>/dev/msglog </dev/console  14 of:5:wait:/sbin/uadmin 2 6 >/dev/msglog 2<>/dev/msglog </dev/console  15 rb:6:wait:/sbin/uadmin 2 1 >/dev/msglog 2<>/dev/msglog </dev/console  16 sc:234:respawn:/usr/lib/saf/sac -t 300  17 co:234:respawn:/usr/lib/saf/ttymon -g -h -p "`uname -n` console login: "  -T terminal-type -d /dev/console -l console -m ldterm,ttcompat |

1. Initializes STREAMS modules
2. Configures socket transport providers
3. Initializes file systems
4. Defines default run level
5. Describes a power fail shutdown
6. Defines single-user mode
7. Defines run level 0
8. Defines run level 1
9. Defines run level 2
10. Defines run level 3
11. Defines run level 5
12. Defines run level 6
13. Defines an unused level, firmware
14. Defines an unused level, off
15. Defines an unused level, reboot
16. Initializes Service Access Controller
17. Initializes console

## What Happens When the System Is Brought to Run Level 3

1. The init process is started and reads the /etc/default/init file to set any environment variables. By default, only the TIMEZONEvariable is set.
2. Then init reads the inittab file to do the following:
   1. Identify the initdefault entry, which defines the default run level (3).
   2. Execute any process entries that have sysinit in the action field so that any special initializations can take place before users login.
   3. Execute any process entries that have 3 in the rstate field, which matches the default run level, 3.

See [init(1M)](http://docs.oracle.com/docs/cd/E19455-01/806-0625/6j9vfilo2/index.html) for a detailed description of how the init process uses the inittab file.

The following table describes the key words used for run level 3's action field.

Table 8-3 Run Level 3 Action Key Word Descriptions

| **Key Word** | **Starts the Specified Process ...** |
| --- | --- |
| powerfail | Only when the system receives a power fail signal. |
| Wait | And waits for its termination. |
| Respawn | If it does not exist. If the process already exists, continue scanning the inittab file. |

The following table describes the processes (or commands) executed at run level 3.

Table 8-4 Run Level 3 Command Descriptions

| **Command or Script Name** | **Description** |
| --- | --- |
| /usr/sbin/shutdown | Shuts down the system. The init process runs the shutdowncommand only if the system has received a powerfail signal. |
| /sbin/rcS | Mounts and checks root (/), /usr, /var, and /var/adm file systems. |
| /sbin/rc2 | Starts the standard system processes, bringing the system up into run level 2 (multiuser mode). |
| /sbin/rc3 | Starts NFS resource sharing for run level 3. |
| /usr/lib/saf/sac -t 30 | Starts the port monitors and network access for UUCP. This process is restarted if it fails. |
| /usr/lib/saf/ttymon -g -h -p "`uname -n` console login: " -T terminal\_type -d /dev/console -l console | Starts the ttymon process that monitors the console for login requests. This process is restarted if it fails.  The terminal\_type on a SPARC based system is sun  The terminal\_type on an IA based system is AT386 |

The remainder of the /etc/inittab file loads optional components & allows for login programs to run on terminals. For terminal logins, the mingetty program is started on tty1 through tty6 and restarted (respawn) continuously to allow for login after login.

In addition, gdm is started only upon entering runlevel 5 from the last entry in /etc/inittab.

After the entries in /etc/inittab have been executed, the /etc/rc.d/rc.local file is executed to perform tasks that must occur after system startup. The entire Linux initialization process is summarized and illustrated in the snapshot attached.