**System Startup and Shutdown**

1. **The booting sequence**

When the system is powered on or reset, it goes through the boot process until the login prompt appears. A step-by-step system boot process is presented below:

Power on external devices.

Power on the system (or virtual partition).

System BIOS (Basic Input/output System) is initiated, which runs Power On Self Test (POST) on the system hardware components such as processor, memory and l/O, and initializes them.

System BIOS locates the boot device and loads the initial boot loader program called GRUB (Grand Unified Boot loader) into memory. GRUB is the default boot loader in Linux distributions and typically resides on the Master Boot Record (MBR) which is located on the very first sector of the boot device. GRUB code located in the MBR is referred to as stage 1.

GRUB stage 1 may call GRUB stage 1.5, which understands an ext2 file system called e2fs\_stage1\_5, and refers to files such as fat\_stage1\_5, ffs\_stage1\_5, iso9660\_stage1\_5, jfs\_stagel\_5, ufs2\_stage1\_5, vstafs\_stage1 \_5 and xfs\_stage1\_5. GRUB stage 1.5 consults the /boot/grub/grub.conf file for boot directives such as available kernels, their locations, associated ram disk image, the root file system location and the default kernel to boot.

GRUB stage 1 or stage 1.5 calls and loads GRUB stage 2, which presents a list of available bootable operating systems to boot. Several commands are available at this point that can be executed from the grub> shell prompt for performing tasks such as booting the system interactively.

GRUB stage 2 calls the default kernel, loads it into memory, decompresses and initializes it. It also loads the contents of the root directory into the ram disk.

The kernel initializes and configures hardware as it finds, loads required modules, starts the kernel swapper process called kswapd, initializes RAID and LVM virtual devices, creates a root device and mounts the root file system in read-only mode on /.

The kernel calls the init command from the ram disk and transfers the control over to it to initiate the system initialization process. The init process is also known as the first process and it always possesses PID 1.

The init command references the inittab file to determine the default run level to boot into.

The init command calls the /etc/rc.d/rc.sysinit script to perform configuration tasks such as initializing the hardware clock, loading kernel modules, configuring kernel parameters, setting the hostname, remounting the root file system in read/write mode, checking and mounting local file systems, activating disk quotas, starting local swap spaces and activating SELinux.

The init command calls the /etc/rc.d/rc script to execute all startup scripts needed to bring the system to the default run level.

The init command finally runs the /etc/rc.d/rc.local script and spawns getty messages that display the login prompt. If the default run level is set to 5, the /etc/X11/prefdm script is called to bring up the default display manager.

Helpful link:

<https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Reference_Guide/ch-boot-init-shutdown.html>

<http://www.thegeekstuff.com/2011/02/Linux-boot-process/>

1. **System run levels**

A runlevel is one of the modes that a Unix -based operating system will run in. Each runlevel has a certain number of services stopped or started, giving the user control over the behavior of the machine. Conventionally, seven runlevels exist, numbered from zero to six. System run control (rc) levels are pre-defined and determines the current state of the system. Linux supports eight rc levels of which seven are currently implemented. Not all of them are commonly used though. The default rc level is 5. The table below describes various runlevels.

|  |  |
| --- | --- |
| Run Level | Description |
| 0 | Linux is down and the system is halted. |
| s or S | Single user state with file systems unmounted. The system can be accessed only at the system console. |
| 1 | Single user state with all file systems mounted and SELinux activated. Mode for administrative tasks, Does not configure network interfaces, start daemons, or allow non-root logins. |
| 2 | Multi-user state. Does not configure network interfaces or start daemons. |
| 3 | Multi-user state. All system and network services including NFS server running. X window is not available. |
| 4 | Not implemented. |
| 5 | Fully operaţional multi-user state with X window and GUI desktop running. This is the default run level. |
| 6 | Linux reboots. |
| emergency | Special boot mode to fix any system boot issues. The root file system is mounted in read-only mode. |
| rescue | Special boot mode to fix issues such as reinstalling a corrupted boot loader. The system needs to be booted with Rl IEL installation CD/DVD or a bootable USB or CD. |

The table tells that run levels 0 and 6 are used to shutdown and reboot the system. Run levels **1** and **S** bring the system up into the single user mode where only the system administrator can log in and perform administrative tasks that require all or most system services to be in the stopped state. Network connectivity does not start at these run levels though. Run levels 2 and 3 provide multi- user accessibility into the system. The difference between the two lies at the number of network services started up.

Run level 3 has all networking capabilities up, in contrast with run level 2 which has fewer. If X Window software is not installed, run level 3 becomes the default.

Run level 4 is not implemented; it may be defined and used in a future release of the operating system.

Run level 5 is the default run level at which Linux is fully functional with all services including networking and X/GUI are operational.

There are two special boot methods that are typically used to fix issues with an unbootable system. These are referred to as emergency and rescue.

**Checking Current and Previous Run Control Levels**

To check the current and previous rc levels of the system, use the who command with -r option:

**# who —r**

**run-level 5 2008-12-15 10:02 last=3**

The output indicates that the system is currently running at run level 5 and its last run level was 3. The output also displays the date and time of the last system run level change.

You may also use the runlevel command which displays the previous and current run levels:

**# runlevel**

**3 5**

**Changing Run Control Levels**

Run control levels are also referred to as init levels because the system uses the init command to alter levels. The shutdown, halt, rebout and pow eroff commands are also widely ised to change system run levels.

The **init** Command

The **init** command is used to change run levels. If the system is currently in run level 5 and you want to switch to run level 3, issue the command as follows:

**# init 3**

This command gracefully stops all services and daemons that should not be running at level 3. The command does not affect any other running processes and services.

Similarly, by initiating **init 1**from run level 3, most system services and daemons can be stopped for the system to transition into single user state.

To stop all system services gracefully and take the system to the halt state, pass 0 as an argument to the init command:

**# init 0**

Likewise, to stop all system services gracefully, shut down the system and reboot it to the default run level, specify 6 as an argument to the init command:

**# init 6**

If the system is running in run level 1 and you wish to bring it up to run level 5. do the following:

**# init 5**

Helpful links:

1. <https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Installation_Guide/s1-boot-init-shutdown-sysv.html>
2. <https://www.liquidweb.com/kb/linux-runlevels-explained/>
3. **The init Process for RHEL/CentOS 5/6 ( Systemd on RHEL/CentOS 7 )**

The program init is the process with process ID 1. It is responsible for initializing the system in the required way. init is started directly by the kernel and resists signal 9, which normally kills processes. All other programs are either started directly by init or by one of its child processes.

init is centrally configured in the /etc/inittab file where the runlevels are defined (see Section 13.2.1, “Runlevels”). The file also specifies which services and daemons are available in each of the runlevels. Depending on the entries in /etc/inittab, several scripts are run by init. For reasons of clarity, these scripts, called init scripts, all reside in the directory /etc/init.d (see Section 13.2.2, “Init Scripts”).

The entire process of starting the system and shutting it down is maintained by init. From this point of view, the kernel can be considered a background process whose task is to maintain all other processes and adjust CPU time and hardware access according to requests from other programs.

A systemd is a System Management Daemon named with UNIX convention to add ‘d‘ at the end of daemon. So, that they can be easily recognized. Initially it was released under GNU General Public License, but now the releases are made under GNU Lesser General Public License. Similar to init, systemd is the parent of all other processes directly or indirectly and is the first process that starts at boot hence typically assigned a “pid=1“.

A systemd, may refer to all the packages, utilities and libraries around daemon. It was designed to overcome the shortcomings of init. It itself is a background processes which is designed to start processes in parallel, thus reducing the boot time and computational overhead. It has a lot other features as compared to init.

Helpful links:

1. <http://www.tecmint.com/systemd-replaces-init-in-linux/>
2. <https://access.redhat.com/articles/754933>
3. <https://access.redhat.com/videos/898473>
4. **The kernel**

**The kernel controls everything inside-out on a system that runs the Linux operating system. It controls system hardware including memory, processors, disks and I/O devices; runs, schedules and manages processes and service daemons; enforces security and access controls; and so on. The kernel receives instructions from the shell, engages appropriate hardware resources and acts as instructed.**

The Linux kernel is a set of software components called modules that work together coherently as a single unit to enable programs, services and applications to run smoothly and efficiently on the system. Modules are basically device drivers used for controlling hardware devices such as controller cards and peripheral devices, as well as software components such as LVM, file systems and RAID. Some of these modules are static to the kernel while others are loaded automatically and dynamically as necessary.

When the system is booted, the kernel is loaded into memory with all static modules. The dynamic modules are separately and individually loaded when needed. A Linux kernel that comprises of static modules only is referred to as a monolithic kernel and a Linux kernel that includes dynamic modules as well is known as a modular kernel.

A monolithic kernel is typically larger in size than a modular kernel since it includes all components that may or may not be required at all times. This makes a monolithic kernel occupy more physical memory and less efficient.

A modular kernel, on the other hand, is made up of mere critical and essential components, and loads dynamic modules automatically as and when needed. This makes a modular kernel occupy less physical memory as compared to a monolithic kernel, faster and more efficient in terms of overall performance, and less vulnerable to crashes. Another benefit of a modular kernel is that software driver updates only require the associated module recompiled without a system reboot. It does not need an entire kernel recompile.

Linux distributions come standard with several types of kernels that are designed for diverse processor architectures such as 32-bit and 64-bit Intel/AMD (i386/i686), 64-bit Itanium 2 (ia64), 64-bit PowerPC, and IBM System z and 390 mainframes. The **uname** command with -m option tells the architecture of the system. At Linux installation, the installer program automatically chooses the regular kernel that best suits the processor architecture. Files related to kernel architectures are installed in **/boot** directory. The regular kernel supports not more than 4GB of physical memory on a system with one or more processors. Additional kernels - kernel-PAE and kernel-xen - are also available for each supported architecture, which provide support for systems with multiple processors and physical memory beyond 4GB (up to 64GB), and allow for constructing several virtual machines on a single system. Optional software packages such as **kernel-devel** and **kernel-headers** may also be loaded to get device drivers and associated header information. Linux distributions also include source code for the kernel.

The default kernel installed during installation is usually adequate for most system needs; however, it requires a rebuild when a new functionality is added or removed. The new functionality may be introduced by updating or upgrading the kernel, installing a new hardware device or changing a critical system component. Likewise, an existing functionality that is no longer needed may be removed to make the kernel smaller resulting in improved performance and reduced memory utilization. To control the behavior of the modules, and the kernel in general, several tunable parameters are set, which define a baseline for the kernel functionality. Some of these parameters must be tuned to allow certain applications and database software to be installed smoothly and function properly.

**5.1 Checking and understanding kernel version**

To check the version of the kernel, run the **uname** command:

# **uname –r**

2.6.18-194.el5

The output indicates that the kernel version currently in use is 2.6.18-194.el5. An anatomy of the version is displayed below:

From left to right:

(2) Indicates that this is the second major version of the Linux kernel. The major number changes when there are significant alterations, enhancements and updates to the previous major version.

(6) Indicates that this is the sixth major revision of the second major version.

(18) Indicates that this is the eighteenth patched version of this kernel to fix minor bugs and security holes, add minor enhancements and so on.

(19) Indicates that this is the nineteenth version of Cent OS customized kernel.

(4) Indicates that this is the fourth build of the nineteenth version of the Cent OS customized kernel.

(el5) indicates that this kernel is for Red Hat Enterprise Linux 5.

**5.2 Understanding kernel directory structure**

Kernel files are stored at different locations in the system directory hierarchy of which four locations — /boot, /proc, /lib/modules and /usr/src — are of significance, and are explained below :

**REDHAT 5 & 6:**

**The /boot File System**

The /boot file system is created at system installation time and its purpose is to store kernel-related information including current working kernel and associated files. This file system also stores any updated or modified kernel data. An **ll** on /boot displays the following information:

# **II /boot**

total 27575

total 6133

-rw-r--r-- 1 root root 66887 Apr 2 2010 config-2.6.18-194.el5

drwxr-xr-x 2 root root 1024 Feb 23 16:37 grub

-rw------- 1 root root 2769585 Feb 23 17:46 initrd-2.6.18-194.el5.img

drwx------ 2 root root 12288 Feb 23 16:19 lost+found

-rw-r--r-- 1 root root 80032 Mar 16 2009 message

-rw-r--r-- 1 root root 112656 Apr 2 2010 symvers-2.6.18-194.el5.gz

-rw-r--r-- 1 root root 1242340 Apr 2 2010 System.map-2.6.18-194.el5

-rw-r--r-- 1 root root 1953660 Apr 2 2010 vmlinuz-2.6.18-194.el5

The output indicates that the current kernel is vmlinuz-2.6.18-194.el5, its boot image is stored in the initrd-2.6.18-194.el5.img file and configuration is located in the config-2.6.18-194.el5 file.

A sub directory/boot/grub contains GRUB information as shown below:

# **ll /boot/grub**

total 258

-rw-r--r-- 1 root root 63 Feb 23 16:37 device.map

-rw-r--r-- 1 root root 7584 Feb 23 16:37 e2fs\_stage1\_5

-rw-r--r-- 1 root root 7456 Feb 23 16:37 fat\_stage1\_5

-rw-r--r-- 1 root root 6720 Feb 23 16:37 ffs\_stage1\_5

-rw------- 1 root root 573 Feb 23 16:37 grub.conf

-rw-r--r-- 1 root root 6720 Feb 23 16:37 iso9660\_stage1\_5

-rw-r--r-- 1 root root 8224 Feb 23 16:37 jfs\_stage1\_5

lrwxrwxrwx 1 root root 11 Feb 23 16:37 menu.lst -> ./grub.conf

-rw-r--r-- 1 root root 6880 Feb 23 16:37 minix\_stage1\_5

-rw-r--r-- 1 root root 9248 Feb 23 16:37 reiserfs\_stage1\_5

-rw-r--r-- 1 root root 55808 Mar 16 2009 splash.xpm.gz

-rw-r--r-- 1 root root 512 Feb 23 16:37 stage1

-rw-r--r-- 1 root root 104988 Feb 23 16:37 stage2

-rw-r--r-- 1 root root 7072 Feb 23 16:37 ufs2\_stage1\_5

-rw-r--r-- 1 root root 6272 Feb 23 16:37 vstafs\_stage1\_5

-rw-r--r-- 1 root root 8872 Feb 23 16:37 xfs\_stage1\_5

The key file in /boot/grub is grub.conf, which maintains a list of available kernels and defines the default kernel to load at boot. Below you can see the contents of the file:

# **cat /boot/grub/grub.conf**

# grub.conf generated by anaconda

#

# Note that you do not have to rerun grub after making changes to this file

# NOTICE: You have a /boot partition. This means that

# all kernel and initrd paths are relative to /boot/, eg.

# root (hd0,0)

# kernel /vmlinuz-version ro root=/dev/sda2

# initrd /initrd-version.img

#boot=/dev/sda

default=0

timeout=5

splashimage=(hd0,0)/grub/splash.xpm.gz

hiddenmenu

title CentOS (2.6.18-194.el5)

root (hd0,0)

kernel /vmlinuz-2.6.18-194.el5 ro root=LABEL=/ rhgb quiet

initrd /initrd-2.6.18-194.el5.img

**REDHAT 7:**

# ll /boot

total 149276

-rw-r--r--. 1 root root 126431 Aug 17 03:05 config-3.10.0-327.36.1.el7.x86\_64

-rw-r--r--. 1 root root 126426 Oct 29 2015 config-3.10.0-327.el7.x86\_64

drwxr-xr-x. 2 root root 4096 Sep 22 08:45 extlinux

drwx------. 6 root root 104 Sep 22 07:58 grub2

-rw-r--r--. 1 root root 47742885 Sep 21 04:36 initramfs-0-rescue-ceace3cd5e5643328f3c5ada03711f58.img

-rw-r--r--. 1 root root 21304412 Sep 22 07:58 initramfs-3.10.0-327.36.1.el7.x86\_64.img

-rw-r--r-- 1 root root 19962051 Oct 17 08:30 initramfs-3.10.0-327.36.1.el7.x86\_64kdump.img

-rw-r--r--. 1 root root 21305823 Sep 22 07:57 initramfs-3.10.0-327.el7.x86\_64.img

-rw-r--r--. 1 root root 19764820 Sep 22 07:58 initramfs-3.10.0-327.el7.x86\_64kdump.img

-rw-r--r--. 1 root root 601575 Sep 21 04:35 initrd-plymouth.img

-rw-r--r--. 1 root root 252739 Aug 17 03:06 symvers-3.10.0-327.36.1.el7.x86\_64.gz

-rw-r--r--. 1 root root 252612 Oct 29 2015 symvers-3.10.0-327.el7.x86\_64.gz

-rw-------. 1 root root 2965270 Aug 17 03:05 System.map-3.10.0-327.36.1.el7.x86\_64

-rw-------. 1 root root 2963044 Oct 29 2015 System.map-3.10.0-327.el7.x86\_64

-rwxr-xr-x. 1 root root 5154912 Sep 21 04:36 vmlinuz-0-rescue-ceace3cd5e5643328f3c5ada03711f58

-rwxr-xr-x. 1 root root 5155840 Aug 17 03:05 vmlinuz-3.10.0-327.36.1.el7.x86\_64

-rwxr-xr-x. 1 root root 5154912 Oct 29 2015 vmlinuz-3.10.0-327.el7.x86\_64

The output indicates that the current kernel is vmlinuz-3.10.0-327.36.1.el7.x86\_64, its boot image is stored in the initramfs-3.10.0-327.36.1.el7.x86\_64.img file and configuration is located in the config-3.10.0-327.36.1.el7.x86\_64.

A sub directory /boot/grub2 contains GRUB information as shown below:

# ll /boot/grub2

total 32

-rw-r--r--. 1 root root 64 Sep 21 04:37 device.map

drwxr-xr-x. 2 root root 24 Sep 21 04:37 fonts

-rw-------. 1 root root 5263 Sep 22 07:58 grub.cfg

-rw-r--r--. 1 root root 1024 Sep 22 07:58 grubenv

drwxr-xr-x. 2 root root 8192 Sep 21 04:37 i386-pc

drwxr-xr-x. 2 root root 4096 Sep 21 04:37 locale

drwxr-xr-x. 3 root root 19 May 9 2012 themes

The key file in /boot/grub2 is grub.cfg, which maintains a list of available kernels and defines the default kernel to load at boot. Below you can see the contents of the file:

/boot/grub2/grub.cfg

….

### BEGIN /etc/grub.d/10\_linux ###

menuentry 'Red Hat Enterprise Linux Server (3.10.0-327.36.1.el7.x86\_64) 7.2 (Maipo)' --class red --class gnu-linux --class gnu --class os --unrestricted $menuentry\_id\_option 'gnulinux-3.10.0-327.el7.x86\_64-advanced-ec1c6b59-c97a-4fd4-a1b2-4b7c9522e8a3' {

load\_video

set gfxpayload=keep

insmod gzio

insmod part\_msdos

insmod xfs

set root='hd0,msdos1'

if [ x$feature\_platform\_search\_hint = xy ]; then

search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1 --hint='hd0,msdos1' e54aa668-59f3-41bc-bfd0-09bacf20ae5a

else

search --no-floppy --fs-uuid --set=root e54aa668-59f3-41bc-bfd0-09bacf20ae5a

fi

linux16 /vmlinuz-3.10.0-327.36.1.el7.x86\_64 root=UUID=ec1c6b59-c97a-4fd4-a1b2-4b7c9522e8a3 ro elevator=noop barrier=off crashkernel=auto rd.lvm.lv=vg00/lv\_swap rhgb quiet LANG=en\_US.UTF-8

initrd16 /initramfs-3.10.0-327.36.1.el7.x86\_64.img

….

A important file related to the grub configuration is /etc/default/grub:

# cat /etc/default/grub

GRUB\_TIMEOUT=5

GRUB\_DISTRIBUTOR="$(sed 's, release .\*$,,g' /etc/system-release)"

GRUB\_DEFAULT=saved

GRUB\_DISABLE\_SUBMENU=true

GRUB\_TERMINAL\_OUTPUT="console"

GRUB\_CMDLINE\_LINUX="elevator=noop barrier=off crashkernel=auto rd.lvm.lv=vg00/lv\_swap rhgb quiet"

GRUB\_DISABLE\_RECOVERY="true"

Other directories, files and commands important for the boot process:

**The /proc File System**

The /proc file system is a virtual file system that is created in memory. Its contents are created at system boot and destroyed when the system is powered off. Underneath this file system lie current hardware configuration and status information. A directory listing of /proc is provided below:

**# ll /proc**

………………….

-r--r--r-- 1 root root 0 Feb 24 14:13 buddyinfo

dr-xr-xr-x 6 root root 0 Feb 23 15:48 bus

-r--r--r-- 1 root root 0 Feb 24 14:13 cmdline

-r--r--r-- 1 root root 0 Feb 24 14:13 cpuinfo

-r--r--r-- 1 root root 0 Feb 24 14:13 crypto

-r--r--r-- 1 root root 0 Feb 24 14:13 devices

-r--r--r-- 1 root root 0 Feb 24 14:13 diskstats

-r--r--r-- 1 root root 0 Feb 24 14:13 dma

dr-xr-xr-x 2 root root 0 Feb 24 14:13 driver

-r--r--r-- 1 root root 0 Feb 24 14:13 execdomains

-r--r--r-- 1 root root 0 Feb 24 14:13 fb

-r--r--r-- 1 root root 0 Feb 24 14:13 filesystems

dr-xr-xr-x 4 root root 0 Feb 23 15:48 fs

dr-xr-xr-x 3 root root 0 Feb 24 14:13 ide

-r--r--r-- 1 root root 0 Feb 24 14:13 interrupts

-r--r--r-- 1 root root 0 Feb 24 14:13 iomem

-r--r--r-- 1 root root 0 Feb 24 14:13 ioports

dr-xr-xr-x 21 root root 0 Feb 24 14:13 irq

-r--r--r-- 1 root root 0 Feb 24 14:13 kallsyms

-r-------- 1 root root 1073745920 Feb 24 14:13 kcore

-r--r--r-- 1 root root 0 Feb 24 14:13 keys

-r--r--r-- 1 root root 0 Feb 24 14:13 key-users

-r-------- 1 root root 0 Feb 23 15:48 kmsg

-r--r--r-- 1 root root 0 Feb 24 14:13 loadavg

-r--r--r-- 1 root root 0 Feb 24 14:13 locks

-r--r--r-- 1 root root 0 Feb 24 14:13 mdstat

-r--r--r-- 1 root root 0 Feb 23 15:49 meminfo

…………………..

**# cat /proc/cpuinfo**

processor : 0

vendor\_id : GenuineIntel

cpu family : 6

model : 23

model name : Intel(R) Core(TM)2 Duo CPU P8700 @ 2.53GHz

stepping : 10

cpu MHz : 2527.000

……………………

**# cat meminfo**

MemTotal: 1026816 kB

MemFree: 21840 kB

Buffers: 97784 kB

Cached: 666156 kB

SwapCached: 0 kB

Active: 410112 kB

Inactive: 479148 kB

……………………

The data stored in files in the /proc file system is used by a number of system utilities including **top**, **uname** and **vmstat** to display information.

**The /lib/modules Directory**

This directory holds information about kernel modules. Underneath this directory, there are sub-directories specific to the kernels available on the system. For example, the ***ll*** output on /lib/modules below shows that there is one kernel configuration stored on this system:

**# ll**

total 8

drwxr-xr-x 7 root root 4096 Feb 23 17:46 2.6.18-194.el5

**# pwd**

/lib/modules

**# ll `uname -r`**

total 1336

lrwxrwxrwx 1 root root 46 Feb 23 16:29 build -> ../../../usr/src/kernels/2.6.18-194.el5-x86\_64

drwxr-xr-x 2 root root 4096 Apr 2 2010 extra

drwxr-xr-x 9 root root 4096 Feb 23 16:29 kernel

drwxr-xr-x 2 root root 4096 Feb 23 17:45 misc

-rw-r--r-- 1 root root 288180 Feb 23 17:46 modules.alias

-rw-r--r-- 1 root root 69 Feb 23 17:46 modules.ccwmap

-rw-r--r-- 1 root root 217407 Feb 23 17:46 modules.dep

-rw-r--r-- 1 root root 147 Feb 23 17:46 modules.ieee1394map

-rw-r--r-- 1 root root 375 Feb 23 17:46 modules.inputmap

-rw-r--r-- 1 root root 2314 Feb 23 17:46 modules.isapnpmap

-rw-r--r-- 1 root root 74 Feb 23 17:46 modules.ofmap

-rw-r--r-- 1 root root 218408 Feb 23 17:46 modules.pcimap

-rw-r--r-- 1 root root 4033 Feb 23 17:46 modules.seriomap

-rw-r--r-- 1 root root 143691 Feb 23 17:46 modules.symbols

-rw-r--r-- 1 root root 393321 Feb 23 17:46 modules.usbmap

lrwxrwxrwx 1 root root 5 Feb 23 16:29 source -> build

drwxr-xr-x 2 root root 4096 Apr 2 2010 updates

drwxr-xr-x 2 root root 4096 Apr 2 2010 weak-updates

There are several files and a few sub directories displayed in the output above. These files and sub directories hold module specific information. Browse the directory tree to get your self familiarized.

One of the key subdirectories is /lib/modules/2.6.18-194.el5/kernel/drivers where modules categorized in groups are stored in various sub directories as show in the listing below:

**# ll**

total 312

drwxr-xr-x 2 root root 4096 Feb 23 16:29 acpi

drwxr-xr-x 2 root root 4096 Feb 23 16:29 ata

drwxr-xr-x 2 root root 4096 Feb 23 16:29 atm

drwxr-xr-x 4 root root 4096 Feb 23 16:29 block

drwxr-xr-x 2 root root 4096 Feb 23 16:29 bluetooth

drwxr-xr-x 2 root root 4096 Feb 23 16:29 cdrom

drwxr-xr-x 8 root root 4096 Feb 23 16:29 char

drwxr-xr-x 2 root root 4096 Feb 23 16:29 cpufreq

drwxr-xr-x 2 root root 4096 Feb 23 16:29 dca

drwxr-xr-x 2 root root 4096 Feb 23 16:29 dma

drwxr-xr-x 2 root root 4096 Feb 23 16:29 dma\_v3

drwxr-xr-x 2 root root 4096 Feb 23 16:29 edac

………………………….

These sub-directories contain driver modules to control hardware devices associated with them.

**The /usr/src Directory**

This directory contains source code for kernel. All information related to building a new kernel from the source code is located here. The /usr/src directory contains two sub-directories.

1. **Linux shells**

In the Linux world, the *shell* is referred to as the command interpreter. It accepts instructions (or input) from users (or scripts), interprets them and passes to the kernel for processing. The kernel utilizes all hardware and software components required to process the instructions. When finished, the results are returned to the shell and displayed on the screen. The shell also produces appropriate error messages, if generated. In short, the shell operates as an interface between a user and the kernel.

**The Shell**

The shell provides many services such as I/O redirection, tab completion, pattern matching, environment variables, job control, command line editing, command aliasing, command history, tilde substitution, quoting mechanisms, conditional execution, flow control and writing shell scripts.

The shell is changeable providing a user with the flexibility to choose a command interpreter at any time. There are several shells available in Linux some of which are explained below:

**The Bourne Again Shell**

**(BASH)**

The *Bourne Again Shell* is the default shell for all users including *root.* It is identified by the $ prompt and is located in the */bin/bash* file. It supports all the features listed above.

**The Korn Shell**

The *Korn* shell is similar to the BASH shell in terms of features. This shell was created by David Korn at the AT&T labs and its prompt is $. It resides in the */bin/ksh* file.

**The C Shell**

The C shell is mainly used by developers. It provides a programming interface similar to the C language and offers many BASH and Korn shell functions in addition to numerous other features. Created by Bill Joy at the University of California at Berkeley for early BSD UNIX releases, the prompt for the C shell is $ and it resides in the */bin/csh* tile.

**TC Shell**

The TC shell is backward compatible with the C shell with many enhancements. The prompt for the TC shell is $ and it resides in */bin/tcsh* file.