**Basics of UNIX**

* Introduction to UNIX and using the Desktop
  + UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since.
  + By operating system, we mean the suite of programs which make the computer work.
  + It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.
  + UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows which provides an easy to use environment.
  + However, knowledge of UNIX is required for operations which aren't covered by a graphical program, or for when there is no windows interface available, for example, in a telnet session.
  + There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X.
  + The UNIX operating system is made up of three parts; the kernel, the shell and the programs.

**The kernel**

The kernel of UNIX is the hub of the operating system: it allocates time and memory to programs and handles the filestore and communications in response to system calls.The difference between an operating system and a kernel:

The kernel is the part of the operating system. The operating system is the software package that communicates directly to the hardware and our application. The kernel is the lowest level of the operating system. The kernel is the main part of the operating system and is responsible for translating the command into something that can be understood by the computer. The main functions of the kernel are:

1. memory management
2. network management
3. device driver
4. file management
5. process management

### The shell

### The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The adept user can customise his/her own shell, and users can use different shells on the same machine.

### Using Command-Line Features and Help Resources

### ls

### mkdir, rmdir, rm

### cal, date

### man

### cd

### pwd

### cp

### mv

### clear

### cat

### less

### head

### tail

### find

### grep

### wc {-c, -l, -w}

### file / directory creation {cat, vim, touch}

### append (cat >> file)

### wildcard ( \*, ?)

### apropos (When you are not sure of the exact name of a command)

### % apropos keyword

### Chmod

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| u | user |
| g | group |
| o | other |
| a | all |
| r | read |
| w | write (and delete) |
| x | execute (and access directory) |
| + | add permission |
| - | take away permission |

### Ex:

### chmod go-rwx <fileList>

### chmod a+rw <fileList>

### process

### kill <pId>

### ps

### kill -9 <pid> // if process refuse for termination

### Navigating the File System

The UNIX filesystem contains several different types of files:

* Ordinary Files
  + Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
  + Always located within/under a directory file
  + Do not contain other files
* Directories
  + Branching points in the hierarchical tree
  + Used to organize groups of files
  + May contain ordinary files, special files or other directories
  + Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
  + All files are descendants of the root directory, ( named / ) located at the top of the tree.
* Special Files
  + Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Ouput (I/O) operations
  + Unix considers any device attached to the system to be a file - including your terminal:
    - By default, a command treats your terminal as the standard input file (stdin) from which to read its input
    - Your terminal is also treated as the standard output file (stdout) to which a command's output is sent
    - Stdin and stdout will be discussed in more detail later
  + Two types of I/O: character and block
  + Usually only found under directories named /dev
* Pipes
  + UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another
  + For example, to pipe the output from one command into another command:
  + **who | wc -l**

# File Names

UNIX permits file names to use most characters, but avoid spaces, tabs and characters that have a special meaning to the shell, such as:

* **& ; ( ) | ? \ ' " ` [ ] { } < > $ - ! /**
* Case Sensitivity: uppercase and lowercase are not the same! These are three different files:

**NOVEMBER November november**

* Length: can be up to 256 characters
* Extensions: may be used to identify types of files

**libc.a  *- archive, library file***

**program.c  *- C language source file***

**alpha2.f  *- Fortran source file***

**xwd2ps.o  *- Object/executable code***

**mygames.Z  *- Compressed file***

* Hidden Files: have names that begin with a dot (.) For example:
  + **.cshrc .login .mailrc .mwmrc**
* Uniqueness: as children in a family, no two files with the same parent directory can have the same name. Files located in separate directories can have identical names.
* Reserved Filenames:

**/ *- the root directory (slash)***

**. *- current directory (period)***

**.. *- parent directory (double period)***

**~ *- your home directory (tilde)***

## **Directory Structure**

|  |  |
| --- | --- |
| **Directory** | **Description** |
| / | This is the root directory which should contain only the directories needed at the top level of the file structure. |
| /bin | This is where the executable files are located. They are available to all user. |
| /dev | These are device drivers. |
| /etc | Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, ethernet, hosts, where to send critical messages. |
| /lib | Contains shared library files and sometimes other kernel-related files. |
| /boot | Contains files for booting the system. |
| /home | Contains the home directory for users and other accounts. |
| /mnt | Used to mount other temporary file systems, such as cdrom and floppy for the CD-ROM drive and floppy diskette drive, respectively |
| /proc | Contains all processes marked as a file by process number or other information that is dynamic to the system. |
| /tmp | Holds temporary files used between system boots |
| /usr | Used for miscellaneous purposes, or can be used by many users. Includes administrative commands, shared files, library files, and others |
| /var | Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data |
| /sbin | Contains binary (executable) files, usually for system administration. For example *fdisk* and *ifconfig* utlities. |
| /kernel | Contains kernel files |

**Navigating the File System**

* Df
* Du
* Mounting the File System (mount)
* Unmounting the File System (umount)

1. **Managing & search files / directories**
   * Cat
   * Vim
   * Touch
   * Find
2. **File permissions & access**
   * Owner permissions
   * Group permissions
   * Other (world) permissions
   * chmod

## File Access Modes

* Read
* Write
* Execute

|  |  |  |
| --- | --- | --- |
| **Number** | **Octal Permission Representation** | **Ref** |
| **0** | No permission | --- |
| **1** | Execute permission | --x |
| **2** | Write permission | -w- |
| **3** | Execute and write permission: 1 (execute) + 2 (write) = 3 | -wx |
| **4** | Read permission | r-- |
| **5** | Read and execute permission: 4 (read) + 1 (execute) = 5 | r-x |
| **6** | Read and write permission: 4 (read) + 2 (write) = 6 | rw- |
| **7** | All permissions: 4 (read) + 2 (write) + 1 (execute) = 7 | rwx |

**Shell basics**

* The shell provides you with an interface to the UNIX system.
* It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.
* A shell is an environment in which we can run our commands, programs, and shell scripts.
* In UNIX there are two major types of shells:
* The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.
* The C shell. If you are using a C-type shell, the default prompt is the % character.
* There are again various subcategories for Bourne Shell which are listed as follows −
  + Bourne shell ( sh)
  + Korn shell ( ksh)
  + Bourne Again shell ( bash)
  + POSIX shell ( sh)
* The different C-type shells follow −
  + C shell ( csh)
  + TENEX/TOPS C shell ( tcsh)

**Process**

* When you execute a program on your UNIX system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system.

**State:-**  As a process executes it changes state according to its circumstances. Linux processes have the following states

* *Running*

The process is either running (it is the current process in the system) or it is ready to run (it is waiting to be assigned to one of the system's CPUs).

* *Waiting*

The process is waiting for an event or for a resource. Linux differentiates between two types of waiting process; interruptible and uninterruptible. Interruptible waiting processes can be interrupted by signals whereas uninterruptible waiting processes are waiting directly on hardware conditions and cannot be interrupted under any circumstances.

* *Stopped*

The process has been stopped, usually by receiving a signal. A process that is being debugged can be in a stopped state.

* *Zombie*

This is a halted process which, for some reason, still has a task\_struct data structure in the task vector. It is what it sounds like, a dead process.

**Filters**

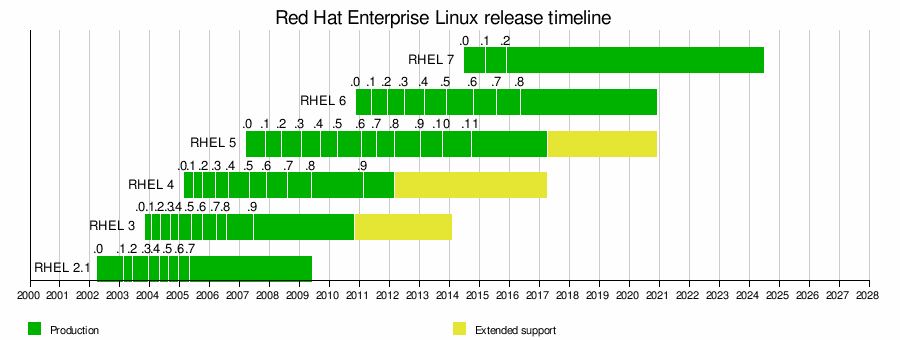
Linux distributions come with various powerful file filtering commands. You can get fast results just with the help of some simple commands.  
Different file filter commands used in Linux are as follows:   
**1) wc:**  
To see the no of characters in a file - $*wc - c <file name>*  
To see the no of words in a file - $*wc -w <file name>*  
To see the no of lines in a file - $*wc -l <file name>*  
To see the no of lines, words, characters at a time - $*wc <file name>*  
**2) Pipe ( | )** :  
We can carry out piping between two commands. Here, the standard output of first command is taken as the standard input for the second command.  
*<Command1> | <command2>*e.g. $*ll | wc*  
**3) head** :  
To see the top 10 lines of a file - $*head <file name>*  
To see the top 5 lines of a file - $*head -5 <file name>*    
**4) tail** :  
To see last 10 lines of a file - $*tail < file name>*  
To see last 20 lines of a file - $*tail -20 <file name>*   
**5) more :**  
To see the contents of a file in the form of page views -$*more <file name>*  
*e.g .*$*more f1.txt*  
**6) sed :**  
It is used to cut the information horizontally.  
To see the first line in file *sun,*  
$*sed -n 1p sun*  
To see 3 to 5 lines  
$*sed -n '3,5p' sun*  
**7) grep :**  
To search a pattern of word in a file, grep file is used.  
syntax $*grep < word name> < file name>*  
$*grep hi file\_1*   
To search multiple words in a file  
$*grep -E ' word1|word2|word3|' <file name>*  
e.g. $*grep -E 'hi|beyond|good' file\_1*  
**8) sort :**  
This command is used to sort the file .  
$*sort  <file name>*  
e.g.: $*sort file\_1*  
To sort the files in reverse order  
$*sort -r <file name>*

To display only files   
$*ll | grep "^-"*  
To display only directories  
$*ll | grep "^d"*

**RHEL 6 Administration**

**Introduction to Red Hat**

* Red Hat Enterprise Linux (RHEL) is a Linux distribution developed by Red Hat and targeted toward the commercial market. Red Hat Enterprise Linux is released in server versions for x86, x86-64, Itanium, PowerPC and IBM System z, and desktop versions for x86 and x86-64.
* All of the Red Hat's official support and training, together with the Red Hat Certification Program, focuses on the Red Hat Enterprise Linux platform. Red Hat Enterprise Linux is often abbreviated to RHEL, although this is not an official designation.



Latest kernel version - 4.6.3 (stable)

**History of LINUX**

* The history of Linux began in 1991 with the commencement of a personal project by Finnish student Linus Torvalds to create a new free operating system kernel.
* After AT&T had dropped out of the Multics project, the Unix operating system was conceived and implemented by Ken Thompson and Dennis Ritchie in 1969 and first released in 1970.
* Later they rewrote it in a new programming language, C, to make it portable. The availability and portability of Unix caused it to be widely adopted, copied and modified by academic institutions and businesses.

Naming

* Linus Torvalds had wanted to call his invention Freax, a portmanteau of "free", "freak", and "x" (as an allusion to Unix).
* During the start of his work on the system, he stored the files under the name "Freax" for about half of a year.
* Torvalds had already considered the name "Linux," but initially dismissed it as too egotistical.

Open Source Development Lab and Linux Foundation

* The Open Source Development Lab (OSDL) was created in the year 2000, and is an independent nonprofit organization which pursues the goal of optimizing Linux for employment in data centers and in the carrier range.
* It served as sponsored working premises for Linus Torvalds and also for Andrew Morton (until the middle of 2006 when Morton transferred to Google). Torvalds worked full-time on behalf of OSDL, developing the Linux kernels.

10 of the Most Popular Linux Distributions Compared

* Ubantu
* Mint
* Debian
* Fedora
* CentOS / RHEL
* OpenSuse / SUSE Linux Enterprise
* Mandriva
* Arch Linux
* Slackware Linux
* Puppy Linux

**Linux Principal**

* Everything is a file. ( Including hardware )
* Small, single-purpose programs.
* Ability to chain programs together to perform complex tasks.
* Avoid captive user interfaces.
* Configuration data stored in text.

**Everything is a File: –**

UNIX systems have many powerful utilities designed to create and manipulate files. The UNIX security model is based around the security of files. By treating everything as a file, a consistency emerges. You can secure access to hardware in the same way as you secure access to a document.

**Small, single-purpose programs: –**

UNIX provides many small utilities that perform one task very well. When new functionality is required, the general philosophy is to create a separate program – rather than to extend an existing utility with new features.

**Ability to chain programs together to perform complex tasks:-**

A core design feature of UNIX is that the output of one program can be the input for another. This gives the user the flexibility to combine many small programs together to perform a larger, more complex task.

**Avoid captive user interfaces:-**

Interactive commands are rare in UNIX. Most commands expect their options and arguments to be typed on the command line when the command is launched. The command completes normally, possibly producing output, or generates an error message and quits. Interactivity is reserved for programs where it makes sense, for example, text editors (of course, there are non-interactive text editors too.)

**Configuration data stored in ,text: –**

Text is a universal interface, and many UNIX utilities exist to manipulate text. Storing configuration in text allows an administrator to move a configuration from one machine to another easily. There are several revision control applications that enable an administrator to track which change was made on a particular day, and provide the ability to roll back a system configuration to a particular date and time.

**Ext 2, Ext 3, Ext 4 and XFS**

**Ext2**

* Ext2 stands for second extended file system.
* It was introduced in 1993. Developed by Rémy Card.
* This was developed to overcome the limitation of the original ext file system.
* Ext2 does not have journaling feature.
* On flash drives, usb drives, ext2 is recommended, as it doesn’t need to do the over head of journaling.
* Maximum individual file size can be from 16 GB to 2 TB
* Overall ext2 file system size can be from 2 TB to 32 TB

**Ext3**

* Ext3 stands for third extended file system.
* It was introduced in 2001. Developed by Stephen Tweedie.
* Starting from Linux Kernel 2.4.15 ext3 was available.
* The main benefit of ext3 is that it allows journaling.
* Journaling has a dedicated area in the file system, where all the changes are tracked. When the system crashes, the possibility of file system corruption is less because of journaling.
* Maximum individual file size can be from 16 GB to 2 TB
* Overall ext3 file system size can be from 2 TB to 32 TB
* There are three types of journaling available in ext3 file system.
* Journal – Metadata and content are saved in the journal.
* Ordered – Only metadata is saved in the journal. Metadata are journaled only after writing the content to disk. This is the default.
* Writeback – Only metadata is saved in the journal. Metadata might be journaled either before or after the content is written to the disk.
* You can convert a ext2 file system to ext3 file system directly (without backup/restore).

**Ext4**

* Ext4 stands for fourth extended file system.
* It was introduced in 2008.
* Starting from Linux Kernel 2.6.19 ext4 was available.
* Supports huge individual file size and overall file system size.
* Maximum individual file size can be from 16 GB to 16 TB
* Overall maximum ext4 file system size is 1 EB (exabyte). 1 EB = 1024 PB (petabyte). 1 PB = 1024 TB (terabyte).
* Directory can contain a maximum of 64,000 subdirectories (as opposed to 32,000 in ext3)
* You can also mount an existing ext3 fs as ext4 fs (without having to upgrade it).
* Several other new features are introduced in ext4: multiblock allocation, delayed allocation, journal checksum. fast fsck, etc. All you need to know is that these new features have improved the performance and reliability of the filesystem when compared to ext3.
* In ext4, you also have the option of turning the journaling feature “off”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Ext2** | **Ext3** | **Ext4** | **XFS** |
| **Full form** | 2nd extended file system | 3rd extended file system | 4th extended file system | extent file system |
| **intro in** | 1993 | 2001 | 2008 | 1994 with IRIX 5.3 |
| **Journaling feature** | No | Yes - dedicated area | yes - but can be turned off | Yes |
| **Individusal File size** | 16 GB to 2 TB | 16 GB to 2 TB | 16 GB to 16 TB | 16 TB to 16 Exabyte |
| **Overall File size** | 2 TB to 32 TB | 2 TB to 32 TB | 1 ExaByte to 1 PetaByte | 16 TB to 18 Exabyte |

**Boot Process**

*[Power on -> BIOS -> MBR -> GRUB -> Kernel -> Run level -> INIT -> user logon]*

BIOS

* Basic Input Output System
* Responsible for searching, loading & executing the boot loader program.
* Searches boot loader in HDD, SD card, CD/DVD.
* Once loaded into memory, it gives the control to MBR.
* BIOS loads & executes the MBR.

MBR

* Master Boot Record.
* Located in the 1st sector of the HDD.
* Size is equal to 512 byte
  + 446 byte = Primary boot loader info
  + 64 byte = partition table info.
  + 2 byte = MBR validation checks
* It contains info about GRUB
* It loads & executes boot loader.

GRUB

* Grand Unified Boot loader.
* Used for selecting the operating system at the time of boot up.
* Has the knowledge of the file system in “/boot/grub/grub.conf”.
* Grub loads & executes the ‘Kernel’ & INITRD (INITial Ram Disk image).
* INITRD is used by kernel as temp root file system until kernel is booted & real root is mounted.

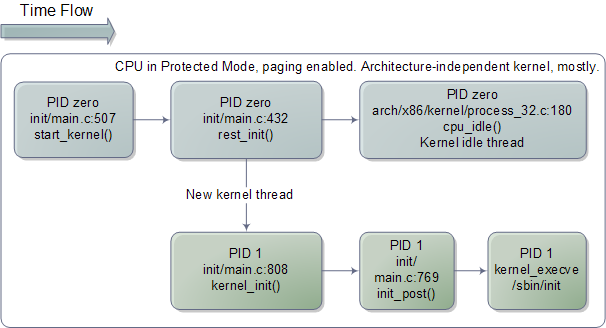
Kernel

* Mounts root file system.
* Executes the INIT program located in “/sbin/init”
* INIT becomes the very 1st process which get executes and got PID as 1.

Boot Loader configuration

* Configuring LILO
  + LILO (LInux LOader), oldest bootloader.
  + It writes the physical address of the kernel to boot on the MBR, which is why each update to LILO (or its configuration file) must be followed by the command lilo.
  + LILO's configuration file is /etc/lilo.conf
* GRUB 2 Configuration
  + GRand Unified Bootloader.
  + It is not necessary to invoke it after each update of the kernel.
  + GRUB knows how to read the filesystems and find the position of the kernel on the disk by itself.
  + To install it on the MBR of the first disk, simply type grub-install /dev/sda.

Kernel Initialization



The boot loader passes two pieces of information to the Linux kernel initilization routine:

* A pointer to the beginning of the initilization RAM disk.
* A pointer to a string containing parameters for initializing the kernel.

This string is commonly referred to as the kernel command-line parameters. These will be saved into kernel variables during early initialization.

On most platforms, the kernel file is compressed to save space on disk. The compressed data is then preceded with a relatively small assembly-language routine that will decompress the kernel proper and then transfer control to the "real" kernel it just decompressed.

The "real" Linux kernel initialization then begins, consisting of the following steps:

1. The kernel initializes hardware as well as kernel data structures.
2. The kernel command line is saved.
3. Hard-drive information is retrieved from the BIOS.
4. The memory size is determined, again through the BIOS.
5. The hardware is prepared to move to protected mode. (Protected mode is the normal memory mode that the system runs in. It allows for the use of virtual memory that allows to use more memory (virtual memory) than is physically present on the processor. Protected mode is also the mode that allows individual programs to run in isolation from one another so that one program cannot accidentally or maliciously damage another program.
6. Segment registers are set and Interrupt Descriptor Table is initialized. These are used in protected mode. Segment registers are used with vitual memory. The Interrupt Descriptor Table is a list of routines in the kernel to be given control when an interrupt has occurred. (An interrupt is a hardware mechanism for giving control to the kernel from an application program.)
7. All interupts are disabled.
8. The "Protected Mode Enabled" bit is set in the Machine Status Word. This step actually turns on protected mode.
9. The page tables are initialized. These are part of virtual memory support.
10. Paging is enabled by setting the PG bit in control register 0. This turns on virtual memory.
11. Exception handlers are installed. These routines get control when a programming error is detected by the hardware.
12. The time of day is read from the CMOS clock. The CMOS clock runs (on a battery) whether the computer is running or not. Thus the date and time can be preserved even when the computer is powered off.
13. The timer interrupt is installed.
14. Interrupts are enabled.
15. INITRD (the intialization ram disk) is mounted. (Mounting a file system makes the files within that file system available to the kernel and to application programs.)
16. Various subsystem initialization routines are called. The kernel is made up of a number of subsystems and most of them have their on initialization code which is called at this time.
17. Other SMP processors are started. Most modern systems have more than one processor. Up until now only one processor has been running. This step starts up the other processor(s).
18. At this stage kernel initialization is essentially completed with the exception of I/O devices. At this point in time, the kernel is ready to enter its normal processing mode which consists of the following cycle:
    * Call the scheduler to pick and run an application program (or "task" as it is known to the kernel.)
    * Either the application program or some other hardware event creates an interrupt. This causes the CPU to switch back to the kernel.
    * The kernel "services the interrupt", i.e. it does whatever is needed to handle the cause of the interrupt.
    * Return to step 1 to resume application-program execution. Note that the application selected to run may or may not be the same program that was interrupted.

At this point in the startup sequence, there are no tasks (or programs) to run. The kernel therefore creates the first task in the task queue. This task (process ID 1) is called the init task. The kernel then creates a second task, known as kthreadd.

1. The kthreadd task, running as a kernel thread then manages the other kernel threads which kthreadd creates.
2. The init task, also running as a kernel thread, does additional initialization.
3. After the above is completed, the init thread executes the program init which is found in the root directory of the initrd file system.

INIT initialization

* The init process reads the file "/etc/inittab" and uses this file to determine how to create processes.
* INIT is the parent of all the process, it is executed by kernel & is responsible for starting all other process.